API Security

API(Application Programming Interface)
one application to communicate with another without having to have direct interaction.
Ex – User of Application A communicate & use data of Application B

Types?
REST (Representational State transfer) & SOAP(Simple object access protocol)
REST is an architecture where when a request is made through HTTP, RESTfull Api response is variety of formats such as HTML, XML JSON

SOAP – has more rules & constraints. Built with diffrent labguages & on diff platforms which leads to longer load times.

API Attack
Lets discuss how to find out Vulnerablities in API's

- Always have a wordlist(even ur own)
- Api Versioning
  - api/v3/users/1/edit exists then check for
  -/api/**v1/users/1**/edit also exists

Vulnerabltites to find out during attacking API

- Information Disclosure
- Authorization Issues
- Business Logic
- Insecure direct object references
  XXS and CSRF

Tools Used

Burp Intuder – Burp is a great tool which helps us test different api endpoints, Intruder allows us to add our own wordlist during attack
ffuf – If wordlist is lengthy
arjun – if you go blank with the api endpoint,Use It. it finds valid HTTP parameters with a huge default dictionary of 10,985 parameter names.
Postman – Postman is tailored to test API. We can import the api files for testing. Connect with Burp & both makes a best of testing API

Lab 1 - IDOR/BAC
Since we know what is an api call, what all the attacks been made on it, lets explore how to perform an attack. Here i have built a python application on flask f/w which stores defence related information of a country

lets run through the code

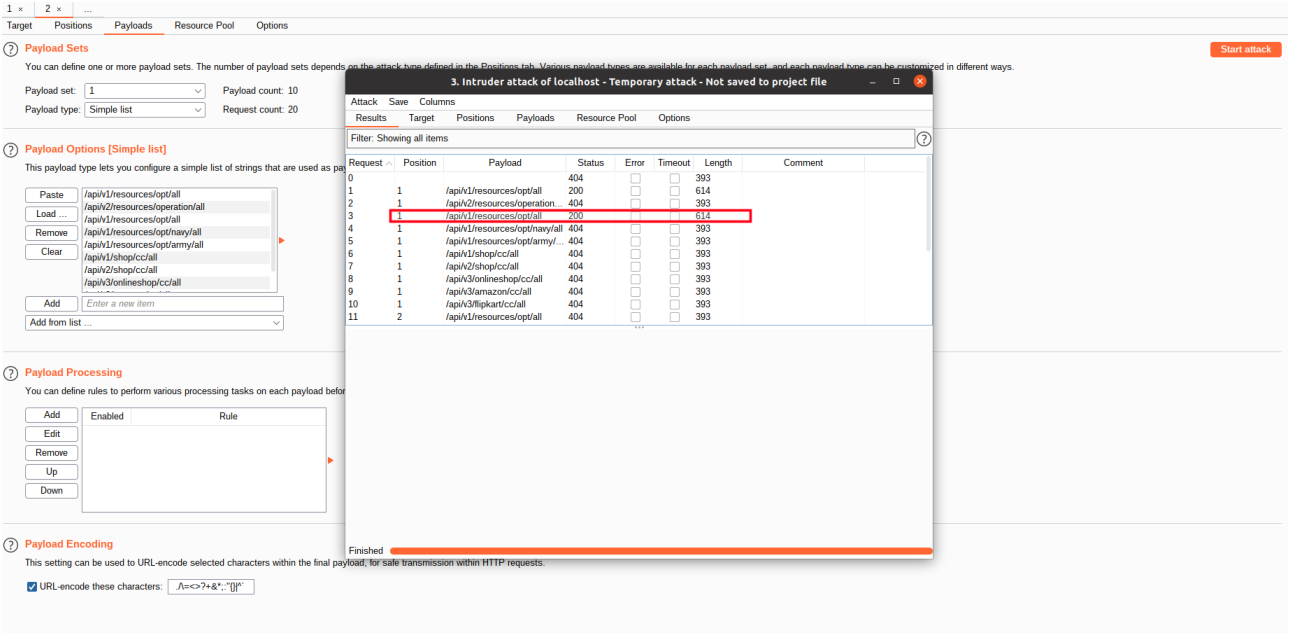
Here just imagine these were the secrete army missions conducted by a country, whose data must never be leaked for for what so ever purposes. But while the attacker gets to know he can sniff into an application which has all the defense data, what all can be done?

Basics – Versioning, /v1 /v2

>run – python3 lab1.py
>you will get a server running on http://0.0.0.0:5001/
>Now go to your browser and type localhost:5001
>Open Burp Suit(or any other proxy) and capture the request

Now lets try to brute force the parameters, as we all know the endpoint for any api starts with /*api*/v1 , with this and as mentioned earlier we know this is a milirary based website we shall build our payload accordingly(i have uploaded the payload for you in the git ripo) since this is demo application the payload is limited.

Go to Intruder in burp and add the "/" which is attack vector and paste the payload in the field. Should look like below



>check the 200 status code vector, send to repeater or view the respose. You shall Find what should'nt have been ;)

attack1 - http://localhost:5001/api/v1/resources/opt/all

Let's not stop there, as mentioned above about versioning lets see if we can go to v2 and check for particular id.

attack 2 - http://localhost:5001/api/v2/resources/opt?id=3

here you can replace the digit to 0,1,2 which is predictable and gives you the information which is not intended to.

Mitigation for IDOR can be, Rather giving id as 1,2,3 we can provide non guessable random numbers
As we can see the vulnerablities here are **IDOR/BAC**(broken access control) We are able to see the data which even though we are not suppose to see it without logging in.

And for broken access control we can use an firewall which i'll give a demo on.

If you do not have BurpSuite you can directly use the urls for checking it out

1. attack 1 - http://localhost:5001/api/v1/resources/opt/all
2. attack 2 - http://localhost:5001/api/v2/resources/opt?id=3

Lab 2 – Information Disclosure
Consider the user who enters a shop accidently sees the billing section and checks the url of the billing desktop browser & get the credit card details of all the customers.

Repeat the above steps of lab 1 including the payload and you will get the credit card information which is very dangerous.

Attack - http://localhost:5005/api/v1/shop/cc/all

Mitigation – Enforce message mediation policy at API gateway level to filter any data that is being returned from API call, which ensures no sensitive data is leaked.

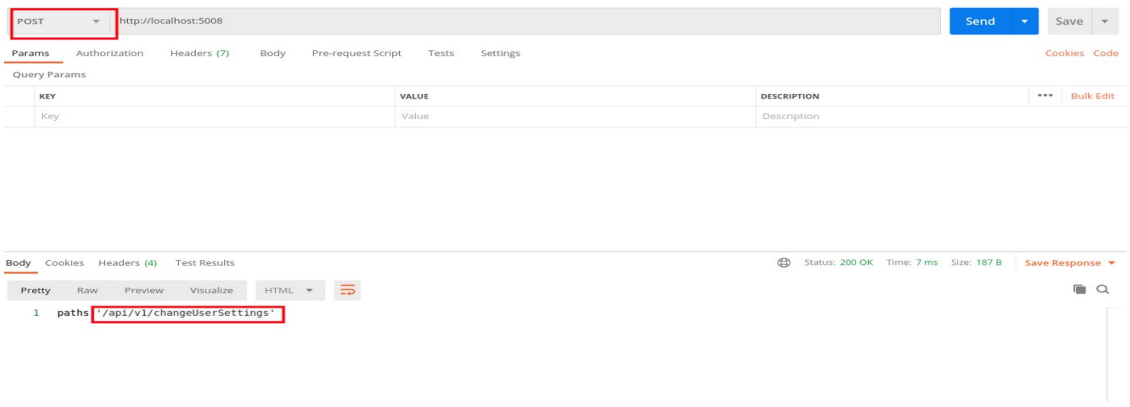
Lab 3 – Mass Assignment
haha!! more of a guess work but ill keep it minimal.
Mass Assignment is sending a data (eg . JSON) to data model. This is done by guessing an object, reading docs, allows attackers to modify object properties they are not supposed to.

Usually only the following fields in your account settings:
Name + lastname
Username
Adress

but here we can see "accountType"

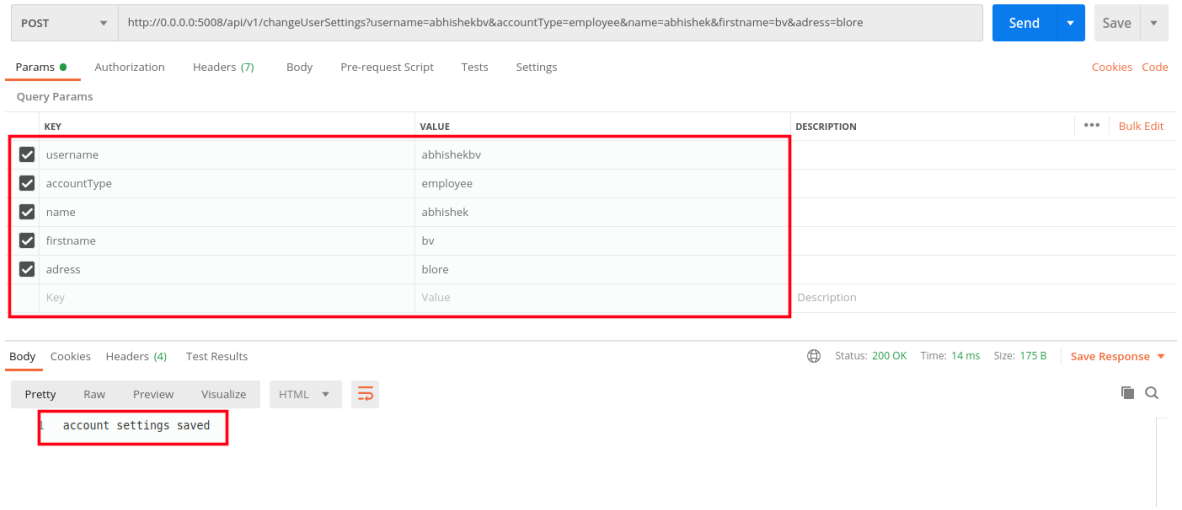
lets Run the application, wait whats this? How to change the method. Here comes the goldmine – POSTMAN tool.

lets use postman to exploit the vulnerablity



Use the path , may give us leads,

we get the error but we have the parameters outlined there for us using which we can find something, always follow the errors



accountType seems to be fishy, lets remove the accountType and check for response

Now we get a different message that is saying that the account type can only be certain types but what if we try admin?
BOOM!!!

Whats wrong? The object “user” supports feild accountType. The developers should not allow the client UI to change this but it does which leads to privilege escalation from mass assignment bugs.

Mitigation – Whitelist the required words for the user and block the remaining.

Api Firewall

What ? Api firewall is a light-weight program to protect your end api endpoints in cloud-native environments with api schema validation, It has a security model which has a set of rules which allows calls that match api specifications while rejecting everything else. Efficient usage in Cloud native environments.

Built on?
Api firewall is built-in OpenAPI, written in Go.

Pre req – Docker, Git, pip

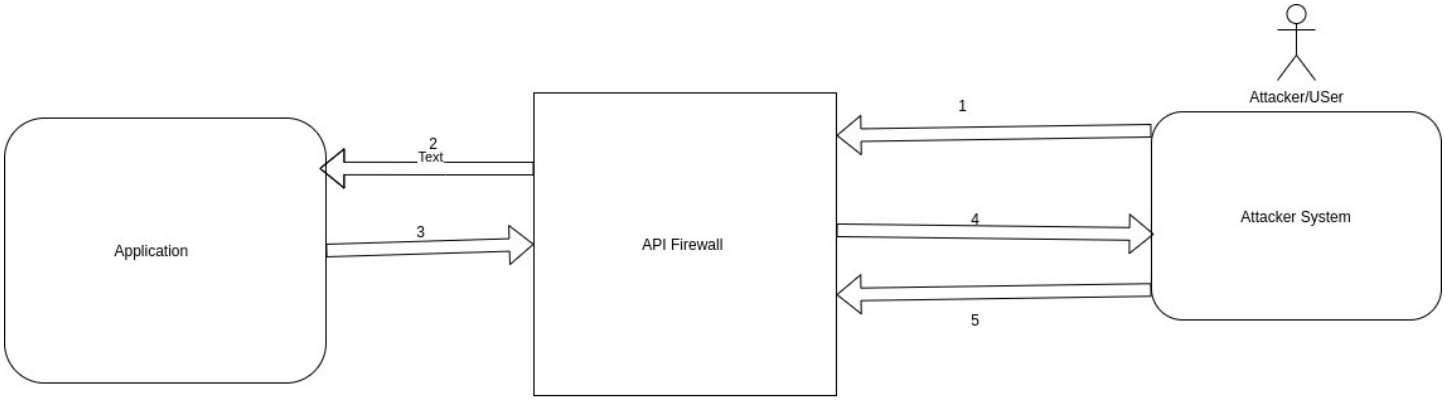
- API firewall - https://hub.docker.com/r/wallarm/api-firewall
- We shall use a demo python application built on Connexion framework that auto handles HTTP requests based on OpenApi Sepcs written in YAML format

Lets Get our Hands Dirty!!

Here i have used two systems(local and vm(to send request) , local ip – 192.168.1.14)

I have the initial set up done for us, installed the api-firewall in docker, have the python program, cd /tmp

- git clone https://github.com/zalando/connexion
- Install the firewall - docker pull wallarm/api-firewall



1 Sends API request

2 Forwards the request if it complies to the rules

3 Response by the Application

4 Response reaches the user

5 The malicious request gets blocked

Step 1 Run the api firewall with

docker run -d -v /tmp/connexion/examples/openapi3/methodresolver/openapi/:/tmp -e APIFW_SERVER_URL=http://192.168.1.14:9090/v1.0/ -e APIFW_API_SPECS=/tmp/pets-api.yaml -e APIFW_REQUEST_VALIDATION=BLOCK -e APIFW_RESPONSE_VALIDATION=BLOCK -p 8282:8282 wallarm/api-firewall

-d – run docker in background
-v mounts the volume of the YAML file in the open API
-e are req environment variables

This will run the docker container in the background and give us container id and in the last line we can see which YML file is specified

Step 2 - We shall run the python application

Step 3 - From the VM (attacker machine)we shall try to do the PUT request

curl -X PUT -H 'content-type: application/json' -d '{"name":"homyak-2", "tag":"aa"}' http://192.168.1.14:8282/v1.0/pets/3 gives us proper response result

lets deviate and think like attacker,

curl -X PUT -H 'content-type: application/json' -d '{"name":123123, "tag":"aa"}' http://192.168.1.14:8282/v1.0/pets/3 where the name parameter is integer , now lets check the result in the docker logs,

Step 4 – Lets check the YAML file for the rules being written
Here we can see that

According to the specified rules our API firewall will either be blocking or logging the requests being made
There's not much on the front end , lets check whats happening to docker logs

error for /name must be string , schema being used to request. Also since we need to defend our application .

So from these we can protect our application from api attacks by creating a simple and free api-firewall. Main advantage is we can prevent DoS attacks by creating and defining rules

Mitigation/Best Practice to prevent API attack

- Always Have a Threat Model
- Implement Oauth – a token based auth f/w that allows info to be access by 3rd party services without exposing user creds.
- Encrypt Data – Sensitive data such as user id, credit card details must be encrypted
- Use Rate Limiting & Throttling – To prevent DdoS
- Use API Firewall

POC images stepwise
1.

```
:/tmp$ git clone https://github.com/zalando/connexion
Cloning into 'connexion'...
remote: Enumerating objects: 8369, done.
remote: Counting objects: 100% (644/644), done.
remote: Compressing objects: 100% (375/375), done.
remote: Total 8369 (delta 379), reused 452 (delta 264), pack-reused 7725
Receiving objects: 100% (8369/8369), 13.57 MiB | 5.20 MiB/s, done.
Resolving deltas: 100% (5847/5847), done.
```

2.

```
: docker pull wallarm/api-firewall
Using default tag: latest
latest: Pulling from wallarm/api-firewall
540db60ca938: Already exists
344f9db7ac86: Pull complete
acb021cd8834: Pull complete
Digest: sha256:c532fe1d9a2d7c0269549e2941e2c479b6b48b57fb4e57c9790fa76d493550eb
Status: Downloaded newer image for wallarm/api-firewall:latest
docker.io/wallarm/api-firewall:latest
```

3.

```
docker run -d -v /tmp/connexion/examples/openapi3/methodresolver/openapi/:/tmp -e APIFW_SERVER_URL=http://192.168.1.14:9090/v1.0/ -e APIFW_API_SPECS=/tm
p/pets-api.yaml -e APIFW_REQUEST_VALIDATION=BLOCK -e APIFW_RESPONSE_VALIDATION=BLOCK -p 8282:8282 wallarm/api-firewall
748e447f4d10da9df69636eb5a5b17a73e64a385a8b8de18fe6ebb236c68e3af
```

4

```
python3 app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
INFO:werkzeug: * Running on http://0.0.0.0:9090/ (Press CTRL+C to quit)
INFO:werkzeug: * Restarting with stat
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 650-077-626
```

5

```
docker logs -f 748e447f4d10da9df69636eb5a5b17a73e64a385a8b8de18fe6ebb236c68e3af
time=2021-08-26T06:24:21Z level=info msg=main : Started : Application initializing : version "develop"
time=2021-08-26T06:24:21Z level=info msg=main: Configuration Loaded :
--svn=develop
--desc=Wallarm API-Firewall
--tls-certs-path=certs
--tls-cert-file=localhost.crt
--tls-cert-key=localhost.key
--server-url=http://192.168.1.14:9090/v1.0/
--server-max-conns-per-host=512
--server-read-timeout=5s
--server-write-timeout=5s
--server-dial-timeout=200ms
--api-host=http://0.0.0.0:8282
--read-timeout=5s
--write-timeout=5s
--log-level=DEBUG
--log-format=TEXT
--request-validation=BLOCK
--response-validation=BLOCK
--custom-block-status-code=403
--add-validation-status-header=false
--api-specs=/tmp/pets-api.yaml
--shadow-api-exclude-list=[404]

time=2021-08-26T06:24:21Z level=info msg=main: Initializing API support
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : GET - /v1.0/pets
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : POST - /v1.0/pets
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : DELETE - /v1.0/pets/{petId}
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : GET - /v1.0/pets/{petId}
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : PUT - /v1.0/pets/{petId}
time=2021-08-26T06:24:21Z level=info msg=main: API listening on http://0.0.0.0:8282
```

6. From Virtual machine type

curl -X PUT -H 'content-type: application/json' -d '{"name":"homyak-2", "tag":"aa"}' http://192.168.1.14:8282/v1.0/pets/3

you get the logs as below

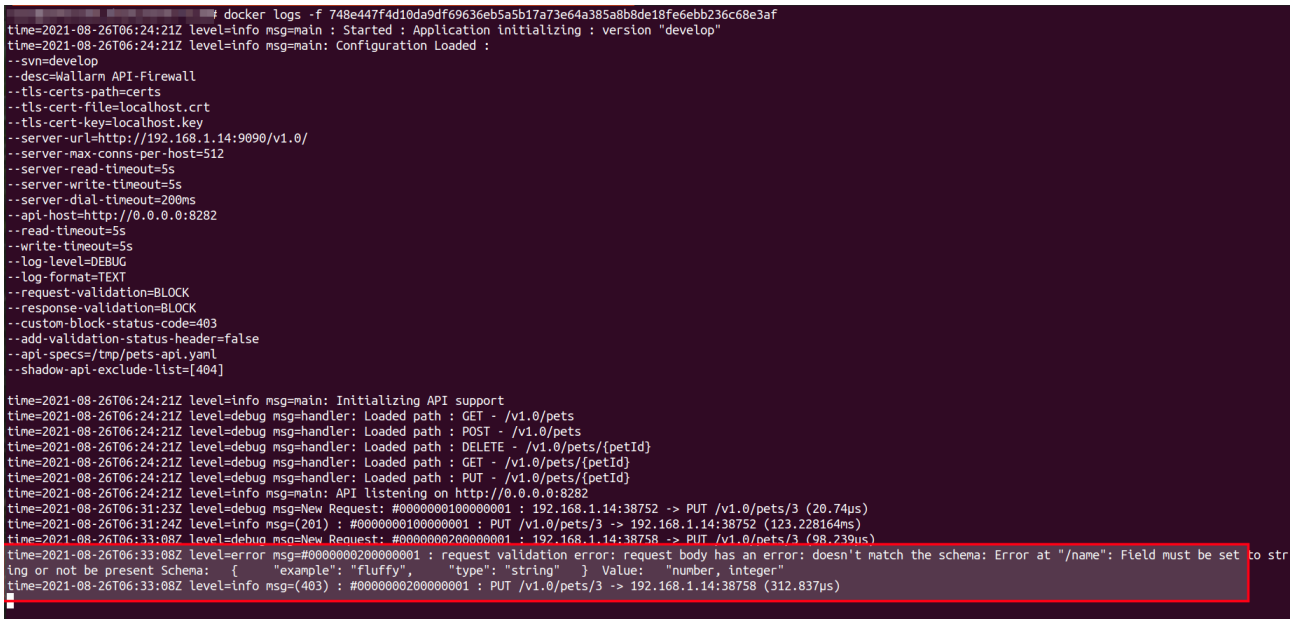
```
docker logs -f 748e447f4d10da9df69636eb5a5b17a73e64a385a8b8de18fe6ebb236c68e3af
time=2021-08-26T06:24:21Z level=info msg=main : Started : Application initializing : version "develop"
time=2021-08-26T06:24:21Z level=info msg=main: Configuration Loaded :
--svn=develop
--desc=Wallarm API-Firewall
--tls-certs-path=certs
--tls-cert-file=localhost.crt
--tls-cert-key=localhost.key
--server-url=http://192.168.1.14:9090/v1.0/
--server-max-conns-per-host=512
--server-read-timeout=5s
--server-write-timeout=5s
--server-dial-timeout=200ms
--api-host=http://0.0.0.0:8282
--read-timeout=5s
--write-timeout=5s
--log-level=DEBUG
--log-format=TEXT
--request-validation=BLOCK
--response-validation=BLOCK
--custom-block-status-code=403
--add-validation-status-header=false
--api-specs=/tmp/pets-api.yaml
--shadow-api-exclude-list=[404]

time=2021-08-26T06:24:21Z level=info msg=main: Initializing API support
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : GET - /v1.0/pets
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : POST - /v1.0/pets
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : DELETE - /v1.0/pets/{petId}
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : GET - /v1.0/pets/{petId}
time=2021-08-26T06:24:21Z level=debug msg=handler: Loaded path : PUT - /v1.0/pets/{petId}
time=2021-08-26T06:24:21Z level=info msg=main: API listening on http://0.0.0.0:8282
time=2021-08-26T06:31:23Z level=debug msg=New Request: #0000000100000001 : 192.168.1.14:38752 -> PUT /v1.0/pets/3 (20.74µs)
time=2021-08-26T06:31:24Z level=info msg=(201) : #0000000100000001 : PUT /v1.0/pets/3 -> 192.168.1.14:38752 (123.228164ms)
```

7. From the vm send the request

curl -X PUT -H 'content-type: application/json' -d '{"name":123123, "tag":"aa"}' http://192.168.1.14:8282/v1.0/pets/3

for which the logs are



In Short – The above is a sample scenario of rules which tells what to & what not to communicate to our application with the use of an api-firewall.