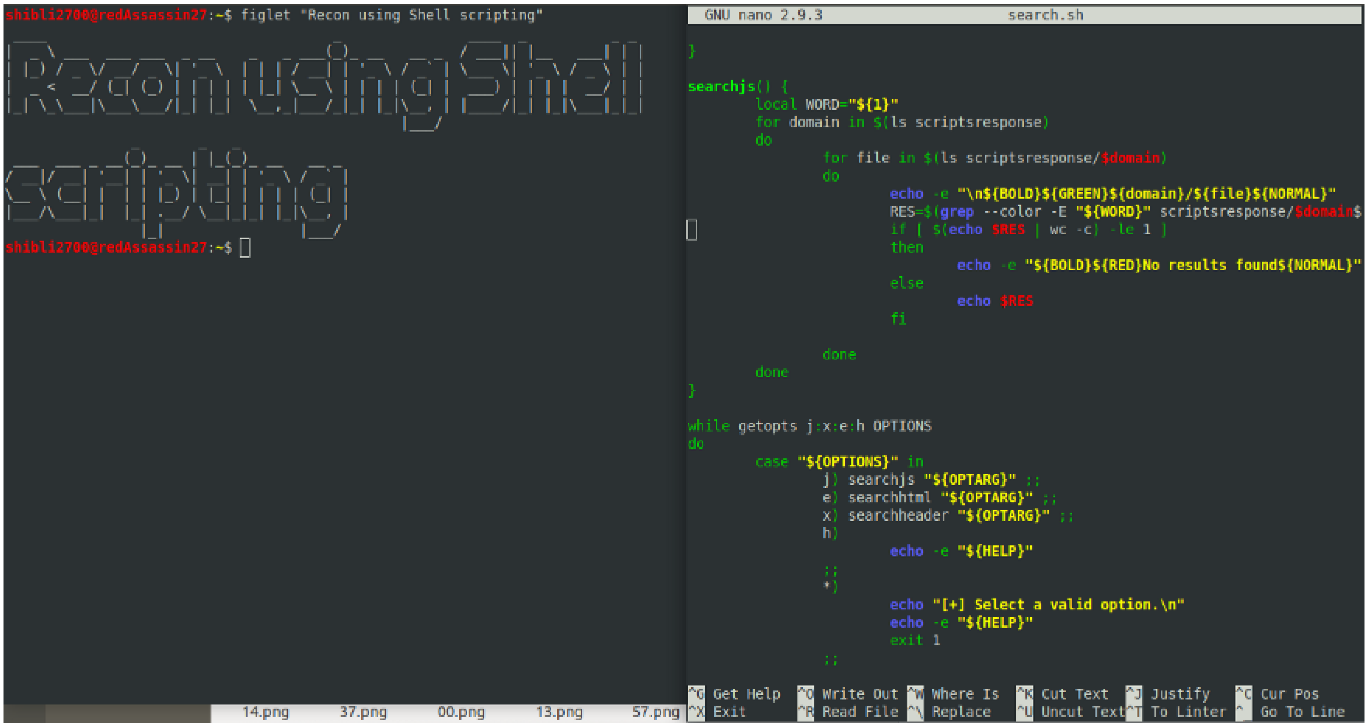
Fasten your Recon process using Shell Scripting

Mohd Shibli Follow
Nov 16, 2019 · 10 min read

> *A lot of new hackers do not utilize the power of automation but once you get used to it there is no turning back.*



```
shibli2700@redAssassin27:~$ figlet "Recon using Shell scripting"
shibli2700@redAssassin27:~$
```

```
  GNU nano 2.9.3                    search.sh

}

searchjs() {
        local WORD="${1}"
        for domain in $(ls scriptsresponse)
        do
                for file in $(ls scriptsresponse/$domain)
                do
                        echo -e "\n${BOLD}${GREEN}${domain}/${file}${NORMAL}"
                        RES=$(grep --color -E "${WORD}" scriptsresponse/$domain$
                        if [ $(echo $RES | wc -c) -le 1 ]
                        then
                                echo -e "${BOLD}${RED}No results found${NORMAL}"
                        else
                                echo $RES
                        fi

                done
        done
}

while getopts j:x:e:h OPTIONS
do
        case "${OPTIONS}" in
                j) searchjs "${OPTARG}" ;;
                e) searchhtml "${OPTARG}" ;;
                x) searchheader "${OPTARG}" ;;
                h)
                        echo -e "${HELP}"
                ;;
                *)
                        echo "[+] Select a valid option.\n"
                        echo -e "${HELP}"
                        exit 1
                ;;
```

Recon using shell scripting

Recon plays an important part while you are hacking into a system as it gives you the idea about the system and how much area you can cover while you will be hacking, sometimes you find a lot of cool vulnerabilities just by doing recon for example :-

- Sensitive information disclosure.
- Open S3 buckets.

- Subdomain takeovers.
- Usage of buggy applications.

So doing recon not only provides you with a bunch of **important** data but also helps you in finding **quick bugs**. In this article, we are going to cover some of the automation that I use while I do my recon, it not only saves time but also gives me a clear picture of all the parts of the system.

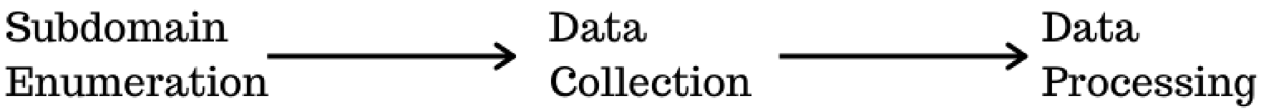
Getting Started

Before we get started I would recommend you to have a basic knowledge of the following topics:

- Understanding of Basic Linux commands and Shell Scripting.
- Basic Networking Knowledge and understanding of Client-Server Architecture.
- Understanding of different protocols http, ftp, ssh etc.

Approach

We are going to follow the below approach in our recon.



Recon Approach

- Subdomain Enumeration — We will be seeing how we can automate our subdomain enumeration process and can gather as many domains as possible.

- Data collection — This phase will focus on collecting different types of data about the hosts e.g open ports, JS files, technologies, header response etc.
- Data processing—In this phase, we will try to crawl different assets from the collected data and will see some techniques to look for quick bugs.

Subdomain Enumeration

There are lot of tools available for subdomain enumeration the one I use are listed below :-

- Sublist3r — https://github.com/aboul3la/Sublist3r
- Assetfinder by TomNomNom https://github.com/tomnomnom/assetfinder
- Google Dorks — While playing with google dorks its better to go for a manual approach.
- GitHub — Sometime GitHub also reveals some of the subdomain which are used internally by the organization.
- crt.sh — It allows you to use wildcards, this tool will help you to identify the domain structure of an organization.

Using all the above tools individually will take a lot of time and the data gathered needs to be formatted properly for further processing. So, let’s remove all the hassle and try to automate our subdomain enumeration process.

Before we start scripting, make sure you have downloaded and installed **Sublist3r**, **Assetfinder** and **httprobe** on your machine.

```
##!/bin/bash

#starting sublist3r

sublist3r -d $1 -v -o domains.txt

#running assetfinder

~/go/bin/assetfinder --subs-only $1 | tee -a domains.txt
```

```
#removing duplicate entries

sort -u domains.txt -o domains.txt

#checking for alive domains

echo "\n\n[+] Checking for alive domains..\n"
cat domains.txt | ~/go/bin/httprobe | tee -a alive.txt

#formatting the data to json

cat alive.txt | python -c "import sys; import json; print
(json.dumps({'domains':list(sys.stdin)}))" > alive.json

cat domains.txt | python -c "import sys; import json; print
(json.dumps({'domains':list(sys.stdin)}))" > domains.json
```

In the above script, we have used **Sublist3r** and **Assetfinder** for subdomain enumeration, then we used **sort** to remove the duplicate entries. One more step we added in the above script to check how many domains are actually alive, for that we have used a tool called **httprobe** by TomNomNom and saved the alive domains in a different file called **alive.txt**.

To run the script use the following commands

```
$ sudo chmod 755 enum.sh #setting file permissions
$ ./enum.sh example.com
```

Once you will run the script, you will get the following four files domains.txt, domains.json, alive.txt and alive.json containing all the subdomains in text and JSON format.

```
├── alive.json
├── alive.txt
├── domains.json
├── domains.txt
└── enum.sh

0 directories, 5 files
```

This is where our first step gets completed, if you want you can modify the script and add some more tools and modifications to it or you can add more domains manually by using **Google dorks** and **GitHub** (more domains means more data which means more bugs) but it's completely fine if you want to use it as it is.

Data Collection

Storing subdomain headers and response bodies

We got a bunch of subdomains now let's start working on them, In this step, we will catch all the response headers and response bodies of the subdomains stored in **alive.txt**, we will be using **cURL** as our primary tool.

```
#!/bin/bash

mkdir headers
mkdir responsebody

CURRENT_PATH=$(pwd)

for x in $(cat $1)
do
        NAME=$(echo $x | awk -F/ '{print $3}')
        curl -X GET -H "X-Forwarded-For: evil.com" $x -I >
"$CURRENT_PATH/headers/$NAME"
        curl -s -X GET -H "X-Forwarded-For: evil.com" -L $x >
"$CURRENT_PATH/responsebody/$NAME"
done
```

In the above script, we are looping through all the domains stored in **alive.txt** and sending cURL requests to fetch headers and response body and then storing them inside `headers` and `responsebody` directories.

Use the following commands to run the script

```
$ sudo chmod 755 response.sh
$ ./response.sh alive.txt
```

Collecting JavaScript files and Hidden Endpoints

Gathering data from JavaScript files is one of the most important steps in a Recon process. In this step, we are going to collect all the **JavaScript files** from the **response body text** which we collected in the previous step.

```
#!/bin/bash

mkdir scripts
mkdir scriptsresponse

RED='\033[0;31m'
NC='\033[0m'
CUR_PATH=$(pwd)

for x in $(ls "$CUR_PATH/responsebody")
do
        printf "\n\n${RED}$x${NC}\n\n"
        END_POINTS=$(cat "$CUR_PATH/responsebody/$x" | grep -Eoi "src=\"[^>]+></script>" | cut -d '"' -f 2)
        for end_point in $END_POINTS
        do
                len=$(echo $end_point | grep "http" | wc -c)
                mkdir "scriptsresponse/$x/"
                URL=$end_point
                if [ $len == 0 ]
                then
                        URL="https://$x$end_point"
                fi
                file=$(basename $end_point)
                curl -X GET $URL -L > "scriptsresponse/$x/$file"
                echo $URL >> "scripts/$x"
        done
done
```

Save the file as **jsfiles.sh** and run the following commands.

```
$ chmod 755 jsfiles.sh
$ ./jsfiles.sh
```

The above script will crawl all the absolute and relative JavaScript file paths from the response bodies and will segregate all the paths according to the sub-domains inside the `scripts` directory. So, if you want to see all the JavaScript files related to abc.example.com you can get it from the file `scripts/abc.example.com` .

The JavaScript URLs will get segregated with respect to the domain name under the scripts directory.

The script will also store the JavaScript file contents inside the `scriptresponse/{domainname}` directory e.g content of all the JavaScript files from abc.example.com will be stored under the directory `scriptresponse/abc.example.com/` .

We have the JavaScript URLs and their respective content the next step is to collect data from these files, the first thing we will try to collect are some hidden endpoints for this we are going to use a tool called relative-url-extractor by Jobert Abma, this tool will collect all the relative paths which are present in the JavaScript files the reason we are doing it is because we can end-up getting some interesting endpoints and configurations.

> *Make sure to clone relative-url-extractor tool in your home directory*

```
#!/bin/bash

#looping through the scriptsresponse directory

mkdir endpoints

CUR_DIR=$(pwd)

for domain in $(ls scriptsresponse)
do
        #looping through files in each domain
        mkdir endpoints/$domain
        for file in $(ls scriptsresponse/$domain)
        do
        ruby ~/relative-url-extractor/extract.rb
scriptsresponse/$domain/$file >> endpoints/$domain/$file
    done
done
```

Save the script as **endpoints.sh** and run the following commands.

```
$ chmod 755 endpoints.sh
$ ./endpoints.sh
```

The above script will loop through all the collected JavaScript files and pass it to our relative-url-extractor tool. Once you will run the above script you will be having `endpoints` directory containing all the endpoint related data. e.g if you want to know the endpoints present in the file `abc.js` from domain `abc.example.com` then it will be present inside the `endpoints/abc.example.com/abc.js` file.

Endpoints present in the JavaScript file.

Till now we have collected sufficient amount of data to work on but there are some few more things which we can collect and that is the number of open ports and services up and running on the hosts for this we are going to use **nmap** , we will be running **nmap** over all the subdomains we have collected so far and will store the results in `nmapscans` directory.

```
#!/bin/bash
mkdir nmapscans

for domain in $(cat $1)
do
        nmap -sC -sV $domain | tee nmapscans/$domain
done
```

Save the script as **nmap.sh** and run it using following commands.

```
$ chmod 755 nmap.sh
$ ./nmap.sh domains.txt
```

The above script is a simple one it will pass all the domains present in **domains.txt** to **nmap** and will store the result inside the `nmapscans` directory but will take considerate amount of time to get executed(only if you have bunch of domains to scan for) so leave the process for a while and grab a quick cup of coffee.

Now we have almost completed our data gathering phase the last thing remaining is **screenshotting,** we have a considerable amount of textual data with us, we also need some good visuals of our target, screenshotting sometimes can lead to quick bugs and find outs.

We are going to use aquatone to take the web screenshots, for this we don't need an actual script we just simply need to pass on our **alive.txt** domains to aquatone and it will generate the **screenshots** for us.

```
$ cat alive.txt | aquatone -out ~/example.com/screenshots/
```

This is the end of our **data-collection** phase. So far we have collected the following data:

- subdomains — More domains means more data to look at.
- Response headers and Response text — Can be used for fingerprinting and to run regex over the text to look for different types of data e.g s3 buckets, secret tokens etc.
- JavaScript files —To find hidden endpoints, sensitive data exposure, JavaScript hijacking and Manual Code testing.
- Nmap scans — Open ports and technologies running on the hosts.
- Web-Screenshots — For quick reviews and quick bugs.

The **data-collection** phase is not limited to the above mentioned techniques, you can add more techniques but for the sake of this article we will be ending our **data-collection** phase to make things a little simple.

Data Processing

This journey has almost come to an end(not really, recon is actually a never-ending process), we have collected a bunch of data now the next step is to process the data to find out vulnerabilities and other useful information.

We will finish off by writing a script to find particular strings in our collected data, this will not only help us to identify multiple assets at once but will also help us get our hands on to some sensitive data.

> ***Fun Fact:-*** *while writing this article I was able to find two info-disclosure bug using above methodology.*

```
#!/bin/bash

BOLD="\e[1m"
NORMAL="\e[0m"
GREEN="\e[32m"
RED="\e[30m"

HELP="
${BOLD}[+]USAGE:${NORMAL} ./search.sh  (OPTIONS)

-j (string) - search in javascript files
-x (string) - search in header files
-e (string) - search in  html files
```

```
-n (string) - search nmap scans
-h - help
"

#writing code to check for expressions in html
searchhtml() {
 local WORD="${1}"
 for domain in $(ls responsebody)
 do
  echo -e "\n${BOLD}${GREEN}${domain}${NORMAL}"
  RES=$(cat responsebody/$domain | grep -E "${WORD}")
  if [ $(echo $RES | wc -c) -le 1 ]
  then
   echo -e "${BOLD}${RED}No results found${NORMAL}"
  else
   echo $RES
  fi
 done
}

searchheader() {
 local WORD="${1}"
        for domain in $(ls headers)
        do
  echo -e "\n${BOLD}${GREEN}${domain}${NORMAL}"
                RES=$(cat headers/$domain | grep -E "${WORD}")
  if [ $(echo $RES | wc -c) -le 1 ]
                then
                        echo -e "${BOLD}${RED}No results
found${NORMAL}"
                else
                        echo $RES
                fi

done
}

searchjs() {
 local WORD="${1}"
        for domain in $(ls scriptsresponse)
        do
  for file in $(ls scriptsresponse/$domain)
  do
   echo -e "\n${BOLD}${GREEN}${domain}/${file}${NORMAL}"
                 RES=$(grep --color -E "${WORD}"
scriptsresponse/$domain/$file)
                 if [ $(echo $RES | wc -c) -le 1 ]
                 then
                         echo -e "${BOLD}${RED}No results
found${NORMAL}"
                 else
                         echo $RES
                 fi
done
 done
}
searchnmap() {
 local WORD="${1}"
        for domain in $(ls nmapscans)
        do
                echo -e "\n${BOLD}${GREEN}${domain}${NORMAL}"
                RES=$(cat nmapscans/$domain | grep -E "${WORD}")
                if [ $(echo $RES | wc -c) -le 1 ]
                then
                        echo -e "${BOLD}${RED}No results
found${NORMAL}"
                else
                        echo $RES
                fi

done

}

while getopts j:x:e:n:h OPTIONS
do
 case "${OPTIONS}" in
  j) searchjs "${OPTARG}" ;;
  e) searchhtml "${OPTARG}" ;;
  x) searchheader "${OPTARG}" ;;
  n) searchnmap "${OPTARG}" ;;
  h) echo -e "${HELP}" ;;
  *)
   echo "[+] Select a valid option.\n"
   echo -e "${HELP}"
   exit 1
  ;;
 esac
done
```

The above script uses command line options to search for the specified input inside the HTML, JavaScript, Nmap scans and header files. The script simply traverse through all the collected data and uses `grep` to find the matching keywords.

```
root@ubuntu:~/example.com$ ./search.sh -h

[+]USAGE: ./search.sh  (OPTIONS)

-j (string) - search in javascript files
-x (string) - search in header files
-e (string) - search in  html files
```

```
  -n (string) - search nmap scans
  -h - help
```

Some examples to use the above script are shown below:

```
  $ ./search.sh -j "admin"
  $ ./search.sh -x "nginx"
  $ ./search.sh -e "s3.amazonaws"
  $ ./search.sh -n "ssh" #searching nmap scans for the string ssh
```

In the first example, the string **“admin”** will be searched inside all the **JavaScript files**. In the second example, the string **“nginx”** will be searched inside all the **header responses** which we gathered in the data collection phase and the third example will look for the string **“s3.amazonaws”** in the **response bodies**.

Try it yourself: If you want you can create a custom **word-list** and can write a simple shell script to pass on the words from the word-list to the **search.sh** script for quick asset discovery.

This is the end of our **data processing** phase, there is a lot more you can add to this phase for example; **checking for s3 buckets using CNAME**, **Testing for PUT method against all the collected hosts**, automating the task to **check for open buckets** etc.

End notes :-

> I believe recon is a never-ending process, the more data you will collect the less it will be.

From the above methodology, we understood how shell scripting can speed up the process and can collect a huge amount of data in just a few minutes, The script you will write for your automation will enhance every time you will work with them and encounter new scenarios while bug hunting.

You are free to tweak with the above scripts and the methodology I used, you can add more phases and more tools in the above recon process for more enhancements, you can find all the above scripts at the following **Github Repository,** feel free to contribute :) .

shibli2700/Rekon

The project contains multiple shell scripts for automating the tasks which most of the…

github.com

Follow Infosec Write-ups for more such awesome write-ups.

InfoSec Write-ups

A collection of write-ups from the best hackers in the world on topics ranging from bug bounties and CTFs to vulnhub…

medium.com

Security Bug Bounty Infosec Cybersecurity Hacking