Tips and Scripts for Reconnaissance and Scanning

Zoltan Panczel

Tips and scripts for reconnaissance and scanning

GIAC (GWAPT) Gold Certification

Author: Zoltán Pánczél, panczelz@gmail.com
Advisor: Christopher Walker, CISSP
Accepted: December 1, 2019

Abstract

Nowadays, information is the key to success. Pentesters' and bounty hunters' first step is to collect information about the target. The crucial part of the recon process is to identify as many hosts as possible. There are plenty of useful, necessary tools to conduct this searching but with limited automated capabilities. The recon and scanning procedures are repetitive; hence, automating these is effective to minimize the effort. Testers can make a customized framework if they combine the primary tools with scripting. Based on the discovered vulnerabilities and work experience, a little tuning or modification of tools might open new opportunities to find bugs.

1. Introduction

Web-based bug bounty hunting and penetration testing are deliberate pursuits. The success rate is relatively low without systematic planning. The assessment involves many repetitive, monotonous, and time-consuming tasks (SACHIN GROVER, 2019). For the sake of efficiency, automation is recommended. However, manual checking is a must. The goal is to maximize useful activity.

There are good “howto”-s and presentations about bug bounty hunting (Jason Haddix, 2018). These cover methods, tools, and tips. Testers have to make their testing environment. Many tools are available to perform different tasks like directory or subdomain brute force. Essentially, there is no guideline on how to establish these into a complex framework. A proper framework can save lots of time, collect and organize relevant information. Most of the testers use VPS systems; hence, console and web-based appropriate, unlike GUI.

After selecting the appropriate necessary tools, shell scripts are used to build the initial framework. In this case, the acquired information is stored at file level. In the second phase of development, a web application is created based on shell scripts developed in the first stage. Most of the data is stored here at the database level, making it easier to search and filter. Accurate parameterization of the tools will be described, which significantly reduces the testing time and false-positive results.

Author Name, email@address

2. Main Part

2.1. Basic tools

One of the recommended testing system in the pentest world is Kali Linux (Offensive Security, 2019). This distribution contains lots of preinstalled security tools and libraries. Additionally, the following software programs are needed for making the testing framework:

1. Dirsearch
   Multithreaded, python developed web path scanner. This command-line tool is suitable for brute-forcing files and directories on web servers.

   Download from: https://github.com/maurosoria/dirsearch

2. Massdns
   Fast and distributed DNS resolver to collect target subdomains. This tool uses the brute force method and certificate transparency logs.

   Download from: https://github.com/blechschmidt/massdns

3. Amass
   Amass is a sophisticated external asset discovery tool that uses open-source information and active reconnaissance. The main testing modules are DNS, Scraping, Certificates, APIs, and web archives.

   Download from: https://github.com/OWASP/Amass

4. Subfinder
   Another subdomain discovery tool uses passive and online resources. Subfinder has permutation capability to discover additional targets.

   Download from: https://github.com/subfinder/subfinder

5. Httprobe
   This tool discovers HTTP/HTTPS services on user-defined ports.

   Download from: https://github.com/tomnomnom/httprobe

Author Name, email@address

6. Subjack
   Subjack is a subdomain takeover tool that supports mass testing fastly and effectively developed in Go. This tool checks for subdomains attached to another domain that expired, and anyone can register it.

   Download from: https://github.com/haccer/subjack

7. Eyewitness
   This tool is taking screenshots from websites and tries both HTTP/HTTPS protocols, therefore, better than other tools. Additionally, checks default credentials on login pages.
   Download from: https://github.com/FortyNorthSecurity/EyeWitness

8. Waybackurls
   Fetch the URLS from archive.org about the target domain.

   Download from: https://github.com/tomnomnom/waybackurls

9. Meg
   The meg application is fetching URLs from servers to avoid flooding the targets. Enhanced replacement of using curl or wget.

   Download from: https://github.com/tomnomnom/meg

10. Linkfinder
    This tool is collecting URLs, API endpoints from the javascript files to identify possible new attack surfaces.

    Download from: https://github.com/GerbenJavado/LinkFinder

In this framework, all tools are installed in the /opt directory.

Author Name, email@address

2.2. Reconnaissance

2.2.1. Preface

During the reconnaissance and scanning phases, there are some recurrent commands like killing or viewing processes as well as tuning the scripts. Making essential auxiliary functions is a must have to speed up the work. All of the scripts below should be in the /root/.bashrc file:

The "k" function has one parameter, which is the name of the killing process. This function tries more aggressively terminate the victim process because killall command is not always working:

```
k(){
    if [ $# -ne 1 ]
    then
        echo "${FUNCNAME[0]} <processname>"
        return 0
    fi
    ps auxw | grep $1 | grep -v grep | awk '{print $2}' | xargs -n1 kill -9
}
```

Figure 1. - k function

The “v” function is a quick edit command to modify the bashrc file, changing vim to other editor is possible based on the user decision:

```
v(){
    vim /root/.bashrc
}
```

Figure 2. - v function

The “s” function applies the modifications which are made in the bashrc file:

```
s(){
    source /root/.bashrc
}
```

Figure 3. - s function

The “p” function shows all the information about the given process name:

```
p(){
    if [ $# -ne 1 ]
    then
        echo "${FUNCNAME[0]} process-name"
        return 0
    fi

    ps auxw | grep -v $1 | grep $1
}
```

Figure 4. - p function

2.2.2. Major modules

The initial information of reconnaissance is a domain/subdomain name or network range. The crucial phase is the subdomain discovery, as many as possible. Hence the core of the recon phase is the **mdns** function:

```
mdns(){
    # mdns hostname wordlist
    if [ $# -ne 2 ]
    then
        echo "${FUNCNAME[0]} <hostname> <wordlist>"
        return 0
    fi
    DIR=$( pwd )
    pushd .
    cd /opt/massdns
    ./scripts/subbrute.py $2 $1 | ./bin/massdns -r lists/resolvers.txt -t A -o S -w $DIR/$1-dns-results.txt
    ./scripts/ct.py $1 | ./bin/massdns -r lists/resolvers.txt -t A -o S -w $DIR/$1-dns-cert-results.txt

    popd

    amass enum -passive -d $1 | tee $1-passive-dns-results.txt
    #amass enum -active -config /root/configs/amass.ini -d $1 | tee $1-active-dns-results.txt
    assetfinder -subs-only $1 | tee $1-assetfinder-dns-results.txt
    findomain -t $1 -o txt
    subfinder -d $1 -silent | tee $1-subfinder-results.txt
    cat *-results.txt *_*.txt | awk '{print $1}' | tr -d ' ' | sed -e 's/\.$//g' |  grep $1 | sort -u >d1.txt
    cat *-results.txt | grep CNAME | awk '{print $3}' | sed -e 's/\.$//g' |  grep $1 | sort -u >>d1.txt
    cat d1.txt | sort -u > $1-domains.txt
    rm d1.txt
}
```

Figure 5. - mdns function

The function has two parameters, the target hostname, and a wordlist file. A good source is the Seclist (Daniel Miessler, 2019) Github repository for making the initial wordlist, and this should expand with the discovered subdomains. This combines with using the scripts of the massdns app to brute force subdomains and collects from certificate transparency logs used by crt.sh. The massdns uses a resolvers.txt file containing DNS servers to change the list only to trustworthy ones. The other enumeration tools (amass, assetfinder, findomain, subfinder) use more than 20 sources to

gather domain information mixed with, for example, wordlist permutation. The result of running the function is a file that contains one domain per line. This list is not 100% correct due to some of the mentioned sources (e.g., archive.org) that have obsolete information.

At this point, there are at least two different approaches to continue the recon phase. Primarily, the following **subextract** script helps to create additional target subdomains to recursively brute force them:

```
subextract(){
    # extract the subdomains from the discovered pool
    if [ $# -ne 1 ]
    then
        echo "${FUNCNAME[0]} <discovered-domains-file.txt>"
        return 0
    fi
    for x in $( cat $1)
    do
        echo $x | awk -F\. 'NF>2' | grep -v ^www\. | awk -F\. '{for (i=NF;i>0;i--){ t=$i"."t; if (i<NF-1){print t} }}' | sed 's/\.$//g' | tee -a subdomains.txt
    done
}
```

Figure 6. - subextract function

The function works with the previously discovered domains list by the **mdns** script.

Secondly, skip the recursive recon and check the remotely accessible web servers. Indeed, after the recursive subdomain recon, this has to be the next step. Keep in mind; sometimes, there is a country restriction in place; for example, the target web server is open from the US but not from Spain. Using VPN/VPS is a proper solution for this case and also prevents possible permanent banning. The **wo** script probes the open HTTP/HTTPS services on the target hosts:

```
wo(){
    if [ $# -ne 1 ]
    then
        echo "${FUNCNAME[0]} <domain-names-file>"
        return 0
    fi
    cat $1 | sort -u | httprobe -c 100 -t 5000 -s -p http:80 -p https:443 -p http:8080 -p http:8888 -p https:8443 -p http:8000 -p http:8081
}
```

Figure 7. - wo function

The **wo** function checks the standard ports of the web servers, but the port list should be based on a port scan result.

Using the **eyewitness** program is an opportunity to grab screenshots from the main page of the discovered web servers. Pictures can be useful to prioritize the target web applications rapidly. The Eyewitness uses a headless browser that is not bulletproof

to capture every webpage. It is recommended to adjust the timeout value to various environments. The **grab** function saves the pictures from open web servers list in the screens subdirectory:

```
grab(){
    if [ $# -ne 1 ]
    then
        echo "${FUNCNAME[0]} <http[s]://open-webservers-list-file>"
        return 0
    fi

    eyewitness --threads 50 --active-scan --web --cycle Crawler -f `pwd`/$1 -d `pwd`/screens
}
```

Figure 8. - grab function

This version of the **wb** function is simple, iterates through the discovered open web servers list and mirrors the archive.org findings by **waybackurls** application:

```
wb(){
    if [ $# -ne 1 ]
    then
        echo "${FUNCNAME[0]} <http[s]://open-webservers-list-file>"
        return 0
    fi

    for x in $( cat $1)
    do
        echo $x | waybackurls | tee -a $x-waybackurls.txt
    done
}
```

Figure 9. - wb function

The tester should make an exclusion list to maximize valuable content. This list can contain pictures (gif, png, jpg), font files (woff, woff2), documents (doc, pdf, ppt). Nevertheless, excessive exclusion can omit potentially vulnerable endpoints like "download.aspx?f=document.pdf". On the other hand, the inclusive approach can skip uncommon file extensions or interfaces. The tester should be prudent when making the filters. The possible direction of development is the parameterization of this function.

Nowadays, more and more companies use cloud infrastructure, and this opens new vulnerabilities. One of the attack methods is the subdomain takeover (Patrik Hudak, 2018). This technique is based on the expired and available domains for registration by

anybody. The **subtake** script checks the discovered subdomains, and the result is reported in a text file:

```
subtake(){
    # subdomain takeover tester
    if [ $# -ne 1 ]
    then
        echo "${FUNCNAME[0]} <subdomain-list-file>"
        return 0
    fi

    #SubOver -l $1 -timeout 5 -o subdomain-takover-output.txt
    # tko-subs
    subjack -w $1 -t 10 -timeout 3 -o subdomain-takover-output.txt -ssl -c /root/go/src/github.com/haccer/subjack/fingerprints.json -a
}
```

Figure 10. - subtake function

LinkFinder is another excellent tool to collect new information about the target(s). The **lf** script can spider a domain and discover javascript files or parse a single URL. There are two main approaches to use this function. One of the possible inputs is the file, which contains the identified web servers. Further inputs could be chained with the **wb** script output from archive.org. The freshly discovered API endpoints should be added to the target list.

```
lf(){
    #
    if [ $# -ne 2 ]
    then
        echo "${FUNCNAME[0]} <http[s]://hostname|http[s]://hostname/script.js> <domain|url>"
        return 0
    fi

    pushd .
    DIR=$( pwd )
    cd /opt/LinkFinder
    HOST=$( echo $1 | tr '/:' '-' )
    FILE=$DIR/$HOST

    case $2 in
    domain)
            ./linkfinder.py -i $1 -d -o cli > "$FILE"-linkfinder.txt
            ;;
    url)
            ./linkfinder.py -i $1 -o cli > "$FILE"-linkfinder.txt
            ;;
    esac

    popd
}
```

Figure 11. - lf function

2.3. Scanning

The tester's next step is to discover remotely accessible services after the recon process. The **portscan** function uses two port scanner tools; the parameters are based on a comparative analysis (Capt. Meelo, 2019). The masscan is fast, but insufficient Nmap is slower but accurate hence, the **portscan** function uses both. **Portscan** function uses scanner application according to the second parameter:

```
portscan(){
    if [ $# -ne 2 ]
    then
        echo "${FUNCNAME[0]} <hostname> <masscan|nmap>"
        exit
    fi

    HOST=$( ping -c 1 $1 -W 1 | head -1 | awk '{print $3}' | tr -d \(\)) # Check host file first, after resolv!

    case $2 in
        masscan)
            masscan -p 1-10000 --rate 5000 --wait 0 --banners --open $HOST -oG $HOST-portscan
            ;;

        nmap)
            nmap --max-rtt-timeout=1250ms --min-rtt-timeout=100ms --initial-rtt-timeout=500ms --max-retries=3 --max-scan-delay=10ms -n -sV --script=banner -T4 -Pn $HOST -oA $
                HOST-portscan -p 1-10000
            ;;

        *)
            echo "Wrong scanning method!"
            ;;
    esac
}
```

Figure 12. - portscan function

The masscan works with IP addresses; thus, the script has to convert the discovered hostname into IP address. The **portscan** function uses ping command and shell script-fu to convert hostname into IP address considering the /etc/hosts file.

One of the most important and complex tasks is the directories and files discovering. This scanning framework uses fine-tuned **dirsearch**:

```
dirsearch(){
    #
    if [ $# -lt 2 ]
    then
        echo "${FUNCNAME[0]} <http[s]://hostname> <extensions> [wordlist] [dir]"
        return 0
    fi

    pushd .
    DIR=$( pwd )
    cd /opt/dirsearch
    HOST=$( echo $1 | tr '/:' '-' )
    FILE=$DIR/$HOST
    case $# in
    2)
        python3 dirsearch.py -f -u $1 -e $2 -t 100 -b -H 'X-FORWARDED-FOR: 127.0.0.1' -H 'X-Bug-Bounty: redacted' --ua='Mozilla/5.0 (compatible; Googlebot/2.1;
            +http://www.google.com/bot.html)' --plain-text-report="$FILE"-found.txt --timeout=3 -x 302,400,429,503
        ;;
    3)
        python3 dirsearch.py -f -u $1 -e $2 -t 100 -b -H 'X-FORWARDED-FOR: 127.0.0.1' -H 'X-Bug-Bounty: redacted' -w $3 --ua='Mozilla/5.0 (compatible; Googlebot/2.1;
            +http://www.google.com/bot.html)' --plain-text-report="$FILE"-found.txt --timeout=3 -x 302,400,429,503
        ;;
    4)
        python3 dirsearch.py -f -u $1 -e $2 -t 100 -b -H 'X-FORWARDED-FOR: 127.0.0.1' -H 'X-Bug-Bounty: redacted' -w $3 --ua='Mozilla/5.0 (compatible; Googlebot/2.1;
            +http://www.google.com/bot.html)' --plain-text-report="$FILE"-found.txt --scan-subdir=$4  --timeout=3 -x 302,400,429,503
        ;;
    esac

    popd
}
```

Figure 13. - dirsearch function

The dirsearch.py has two mandatory parameters: host and extension(s). The **dirsearch** function has two additional parameters: the wordlist file and searches dir. According to the command line parameters, the shell script sets up the dirsearch.py application with the same conditions. This setup uses 100 threads and forces requests by hostname. Some of the web pages/WAFs do not send a proper response if the user agent is not generally known. The scripts set the user-agent HTTP header to Googlebot. This header setting gets access to protected contents in special cases, inappropriate configuration. The X-Forwarded-For header is representing the original IP of the client connecting to a web server through a load balancer, proxy server. The "127.0.0.1" value of the header is able to bypass ACLs. Before starting the manual testing, write a simple Burp Suite Pro extension to add all the "X-" headers to every request. The wordlist file is the default, but the Seclists (Daniel Miessler, 2019), fuzzdb(FuzzDB Project, 2019) are good sources to create a new extended list. The tricky part of the settings is the excluded status codes. If the tester does not use exclusion, he gets too much garbage. In this case, that enormous exclusion misses potential reachable and vulnerable interfaces. The 30X response codes are some type of redirections. The dirsearch.py adds a slash to every possible directory by default. Removing this setting takes advantage of the redirections. The following example shows the importance of the redirect codes:

```
/ws  > 301/302 HTTP to /ws/
/ws/ > 404 HTTP
/ws/services > 200 HTTP
```

Figure 14. - Directory discovering with HTTP response codes

In most environments, this technique is useful to discover more content. The 401 and 403 HTTP responses are also decent targets mixed with the presented **wb** function. There is a chance that the archive.org contains unreferenced or left behind interfaces in subdirectories that are accessible despite the 401/403 response codes.

The **dirsearch** function forces the extensions for every wordlist entries. The current setup uses the following extensions: ashx, asmx, json, xml, svc, php, jsp ,aspx , html, wadl, do, cgi. The tester should extend this list or adapt to the environment as usual.

The dirsearch.py application does not check the possible backup, temp versions of the discovered files therefor the following modification is necessary:

```
./lib/core/Fuzzer.py:

def thread_proc(self):
    self.playEvent.wait()
    backupexts = ['.tar','.zip','.tar.gz','~','_orig','_ORIG','.old','.1','.txt','.save']
    try:
        path = next(self.dictionary)
        while path is not None:
            try:
                status, response = self.scan(path)
                previous = len(response)
                result = Path(path=path, status=status, response=response)

                if status is not None:
                    self.matches.append(result)
                    for callback in self.matchCallbacks:
                        callback(result)
                else:
                    for callback in self.notFoundCallbacks:
                        callback(result)

                if status == 200 and path.find(".") != -1:
                    for i in backupexts:
                        backup = path + i
                        status, response = self.scan(backup)
                        result = Path(path=backup, status=status, response=response)
                        if previous != len(response):
                            self.matches.append(response)
                            for callback in self.matchCallbacks:
                                callback(result)

                del status
                del response
```

Figure 15. - dirsearch backup checker

If the **dirsearch** discovers a file with 200 HTTP response code, the backup scanner makes new HTTP requests with the predefined backup extensions. This list should be updated based on the project experience. The backup or temp files may contain sensitive information and help the vulnerability scanning process.

The discovered URL list is useful for updating the used wordlist. The bounty hunters usually share their own, and these are great sources to make or use new wordlists.

3. Web-based framework

Console based frameworks that generate lots of information are not easy to handle. Therefore, adding database support and web frontends is the next development step. The main properties of the web-based framework are simple, quick, and do not waste resources. Considering the simplicity, Flask, Python, Bash, and SQLite are chosen for the development environment. The web-based framework is currently under development. After creating a new project, a mandatory option is a domain/subdomain name. There is only one function, the subdomain collector, in this screen below the "**Command**" column. A simple status indicator shows the actual state of the running background commands:

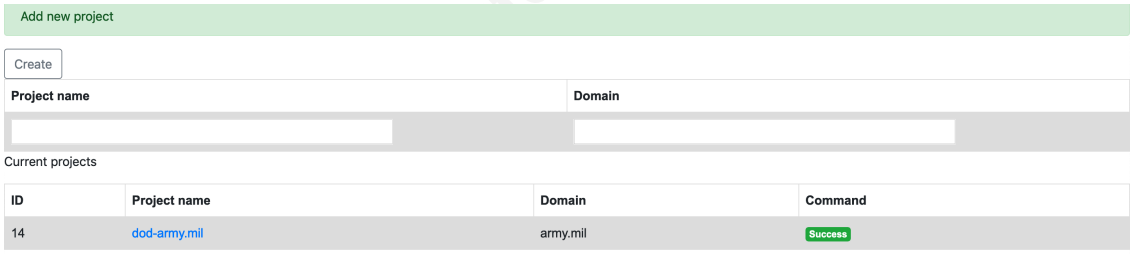

Figure 16. - Main page (Add new projects)

The command-line functions are the right initials to make new shell scripts with database support. Before this, there is a zero-step, comprehensive database designing:

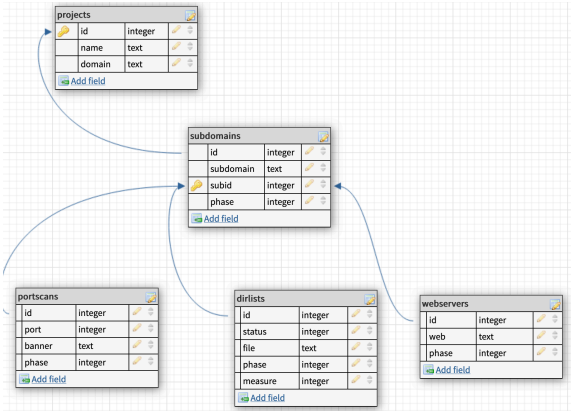

Figure 17. - Database schema design during development

The following screenshot demonstrates the changes in the **mdns** function:

```
#!/bin/bash
if [ $# -ne 3 ]
then
    echo "${FUNCNAME[0]} <hostname> <wordlist> <project_id>"
    exit
fi

sqlite3 /root/recenumscan/sqlite.db 'update projects set phase=1 where id='$3''

DIR=$( pwd )
pushd .
cd /opt/massdns
./scripts/subbrute.py $2 $1 | ./bin/massdns -r lists/resolvers.txt -t A -o S -w $DIR/$1-dns-results.txt
./scripts/ct.py $1 | ./bin/massdns -r lists/resolvers.txt -t A -o S -w $DIR/$1-dns-cert-results.txt
popd

amass enum -passive -d $1 | tee $1-passive-dns-results.txt
assetfinder -subs-only $1 | tee $1-assetfinder-dns-results.txt
findomain -t $1 -o txt
subfinder -d $1 -silent | tee $1-subfinder-results.txt
cat *-results.txt *_*.txt | awk '{print $1}' | tr -d ' ' | sed -e 's/\.$//g' |  grep $1 | sort -u >$1-domains.txt
for i in $( cat $1-domains.txt)
do
    sqlite3 /root/recenumscan/sqlite.db 'insert into subdomains (id, subdomain, phase) values ('$3','\"$i\"',2)'
    sqlite3 /root/recenumscan/sqlite.db 'insert into portscans (id,phase) values ((select subid from subdomains order by subid desc limit 1),0)'
    sqlite3 /root/recenumscan/sqlite.db 'insert into webservers (id,phase) values ((select subid from subdomains order by subid desc limit 1),0)'
    sqlite3 /root/recenumscan/sqlite.db 'insert into dirlists (id,phase) values ((select subid from subdomains order by subid desc limit 1),0)'
    sqlite3 /root/recenumscan/sqlite.db 'insert into waybackurls (id,phase) values ((select subid from subdomains order by subid desc limit 1),0)'
    sqlite3 /root/recenumscan/sqlite.db 'insert into linkfinders (id,phase) values ((select subid from subdomains order by subid desc limit 1),0)'
    sqlite3 /root/recenumscan/sqlite.db 'insert into grabs (id,phase) values ((select subid from subdomains order by subid desc limit 1),0)'
    sqlite3 /root/recenumscan/sqlite.db 'insert into subtakes (id,phase) values ((select subid from subdomains order by subid desc limit 1),0)'
done
sqlite3 /root/recenumscan/sqlite.db 'update projects set phase=2 where id='$3''
rm -f *-*.txt
```

Figure 18. - Modified mdns function

The core structure of the new function is the same – collection domain information - only the results loading into the database is the improvement. The following screenshot demonstrated the FLASK script to run the subdomain discovery process:

```
@app.route('/collect/<int:id>', methods=['GET', 'POST'])
def list(id):

    if request.method == 'GET':

        #1. update phase value
        db = get_db()
        db.execute('update projects set phase = 1 where id=?', (id,))
        db.commit()

        #2. run shell script
        hostname = db.execute('select domain from projects where id = ?',(id,)).fetchone()

        cmd = "./scripts/subdomains.sh {} /tmp/a.txt {}".format(hostname[0], int(id,))
        os.system(cmd)

        #3. from shell script update the tables
        #4. update the phase value

        cmd = "./scripts/webservers-full.sh {}".format(int(id,))
        os.system(cmd)

        cmd = "./scripts/portscan-full.sh {}".format(int(id,))
        os.system(cmd)

    return redirect('/')
```

Figure 19. - FLASK script to run the subdomain discovery

Author Name, email@address

JINJA template is the technique for rendering the results of the discovery process. Templates can contain static and dynamic data, thus rendering the status, and the new functions are easy to implement. The following screenshot shows the possible recon and scanning functions for every discovered subdomain:

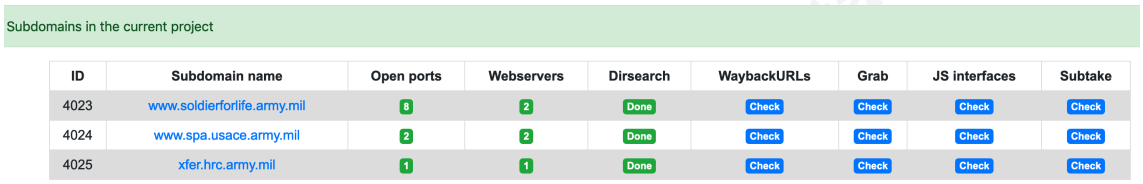

Figure 20. – Recon and scanning functions during the development

The visual representation is based on **bootstrap**, the frontend component library. Behind the subdomain links there is the detailed information as the following screenshot demonstrates:

Figure 21. - Detailed information about a subdomain

The collapsed menu helps focusing on only one of the areas:

Author Name, email@address

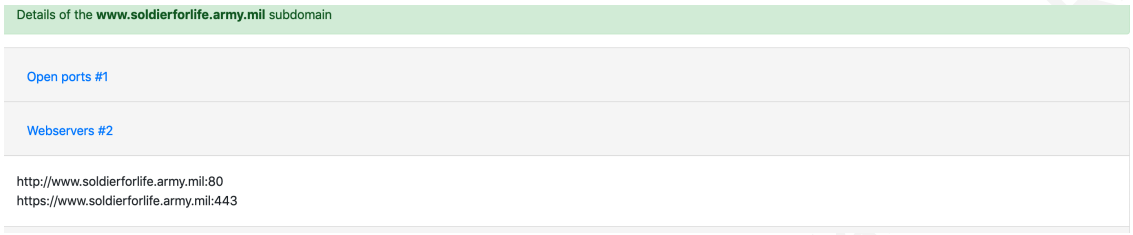

Figure 22. - Collapsed menu structure

The first step of the future development is to migrate all of the functions into the web framework with JavaScript based compound data filtration. Making a system panel that is capable of controlling the background tasks, e.g., kill. Additionally, modifies the core settings like changing the used wordlists. Nevertheless, writing notes function from the discovered details is a must-have because of the future recon or a wide scope. Every host has a checklist which method was tested, for example, SQLi, XXE by the discovered URLs.

4. Conclusion

The demonstrated framework helps speed up the recon phase. The collected information is easily filterable but does not substitute the analysis knowledge. The tester should implement new methods and techniques after a project. Lots of publicly disclosed bugs contain useful techniques (Teddy Katz, 2019). Collect the web interfaces, endpoints from newly discovered vulnerabilities, and implement it into the recon phase is the right direction of development.

Security professionals make better and faster tools from day to day. Replacing **dirsearch.py** with **ffuf** can dramatically decrease the recon time (Joona Hoikkala, 2019). The additional functions, like virtual host discovery, can reveal potential new targets.

Online resources that are good for recon provide API endpoints and facilitate easy implementation into the framework (SecurityTrails, 2019). If this is not possible, making an additional shell function could be the solution.

There is no finished framework; testers have to improve the settings and functions according to the target environment, new techniques, and methods. A wrong tool with a good wordlist is a significant advantage compared to an excellent tool with an average list.

Process documentation is a crucial task. It is recommended to write side notes to the finished tasks to be able to see the big picture or to be prepared for possible future reaudit. These notes should contain every little detail, like request/response pairs, screenshots, and ideas.

Author Name, email@address

References

Offensive Security. (2019). *Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution*. Retrieved 1 November, 2019, from https://www.kali.org/

Daniel Miessler. (2019). *SecLists*. Retrieved 20 October, 2019, from https://github.com/danielmiessler/SecLists

Capt. Meelo. (2019). *Finding the Balance Between Speed & Accuracy During an Internet-wide Port Scanning*. Retrieved 4 October, 2019, from https://captmeelo.com/pentest/2019/07/29/port-scanning.html

Patrik Hudak. (2018). *Subdomain Takeover: Basics*. Retrieved 24 September, 2019, from https://0xpatrik.com/subdomain-takeover-basics/

Mariem. (2018). *Conference notes: The Bug Hunters Methodology v3(ish) (LevelUp 0x02 / 2018)*. Retrieved 20 September, 2019, from https://pentester.land/conference-notes/2018/08/02/levelup-2018-the-bug-hunters-methodology-v3.html

James Kettle. (2019). *Turbo Intruder: Embracing the billion-request attack*. Retrieved 17 September, 2019, from https://portswigger.net/research/turbo-intruder-embracing-the-billion-request-attack

FuzzDB Project. (2019). *Official FuzzDB Project Repo*. Retrieved 5 November, 2019, from https://github.com/fuzzdb-project/fuzzdb

Teddy Katz. (2019). *Bypassing GitHub's OAuth flow*. Retrieved 12 November, 2019, from https://blog.teddykatz.com/2019/11/05/github-oauth-bypass.html

Joona Hoikkala. (2019). *ffuf - Fuzz Faster U Fool*. Retrieved from 12 November, 2019, from https://github.com/ffuf/ffuf

SecurityTrails. (2019). *SecurityTrails*. Retrieved from 12 November, 2019, from https://docs.securitytrails.com/docs

Jason Haddix. (2018). The Bug Hunters Methodology v3(ish) (LevelUp 0x02 / 2018). Retrieved from 15 November, 2019, from https://www.youtube.com/watch?v=Qw1nNPiH_Go

SACHIN GROVER. (2019). Recon Everything. Retrieved from 17 October, 2019, from https://medium.com/@maverickNerd/recon-everything-48aafbb8987

Author Name, email@address

Last Updated: March 14th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS London April 2020	London, GB	Apr 20, 2020 - Apr 25, 2020	Live Event
SANS SEC440 Copenhagen April 2020	Copenhagen, DK	Apr 27, 2020 - Apr 28, 2020	Live Event
SANS Bucharest May 2020	Bucharest, RO	May 04, 2020 - May 09, 2020	Live Event
SANS Milan May 2020	Milan, IT	May 04, 2020 - May 09, 2020	Live Event
SANS Amsterdam May 2020	Amsterdam, NL	May 11, 2020 - May 18, 2020	Live Event
SANS Hong Kong 2020	Hong Kong, HK	May 11, 2020 - May 16, 2020	Live Event
SANS London May 2020	London, GB	May 18, 2020 - May 23, 2020	Live Event
SANS Autumn Sydney 2020	Sydney, AU	May 18, 2020 - May 23, 2020	Live Event
SANS Dublin May 2020	Dublin, IE	May 25, 2020 - May 30, 2020	Live Event
SANS Krakow May 2020	Krakow, PL	May 25, 2020 - May 30, 2020	Live Event
SANS Stockholm May 2020	Stockholm, SE	May 25, 2020 - May 30, 2020	Live Event
SANS FOR508 Madrid May 2020 (In Spanish)	Madrid, ES	May 25, 2020 - May 30, 2020	Live Event
SANS London June 2020	London, GB	Jun 01, 2020 - Jun 06, 2020	Live Event
SANS Chicago Spring 2020	Chicago, ILUS	Jun 01, 2020 - Jun 06, 2020	Live Event
Rocky Mountain HackFest Summit & Training 2020	Denver, COUS	Jun 01, 2020 - Jun 08, 2020	Live Event
SANS ICS Europe Summit & Training 2020	Munich, DE	Jun 08, 2020 - Jun 13, 2020	Live Event
SANS FOR508 Milan June 2020 (in Italian)	Milan, IT	Jun 08, 2020 - Jun 13, 2020	Live Event
SANS New Orleans 2020	New Orleans, LAUS	Jun 08, 2020 - Jun 13, 2020	Live Event
SANS Las Vegas Summer 2020	Las Vegas, NVUS	Jun 08, 2020 - Jun 13, 2020	Live Event
SANS Paris June 2020	Paris, FR	Jun 08, 2020 - Jun 13, 2020	Live Event
SANS Budapest June 2020	Budapest, HU	Jun 08, 2020 - Jun 13, 2020	Live Event
SANSFIRE 2020	Washington, DCUS	Jun 13, 2020 - Jun 20, 2020	Live Event
SANS Brussels April 2020	OnlineBE	Apr 20, 2020 - Apr 25, 2020	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced