

Welcome to SANS SEC522!

SEC522 Lab Workbook Version G01_01

The goal of the SEC522 Electronic Workbook is to increase the **in-class**, and, most importantly, **after-class** value of the course material. This electronic workbook is, and likely always will be, very much a work in progress.

Contained in the wiki, you will find:

- Electronic Copies of the Lab Guides (**copy and paste, FTW!!!**)
- Course Index
- Tools, cheat sheets, and white papers

Updating the Wiki

Note: To ensure you are seeing the latest version of the wiki, please run the following commands in the Terminal.

```
cd ~/
sans-update-workbook
```

The output should look similar to the following snippet.

```
f8e4886 (grafted, HEAD, origin/F01_01, F01_01) Lab 3.3: Fixed typo in lab
```


objectives.

Errata Patches

On occasion, errata for the VM may be published to update command or lab files. To apply the errata patches, open Firefox inside the virtual machine to view the online Wiki. On the homepage, locate the Errata Patches section of the Wiki and click the link to view the patches for the various versions of the VM.

To find the current version, look on the front page of the course Wiki and locate the "Course Version X##_##". Click the associated link under the Errata menu to view the patches.

Course, Workbook, or Lab Bugs

Please let us know if you find any bugs in the courseware, workbook, or in the labs that we need to squash. Also, reach out if you have suggestions to improve the course (e.g. content/labs/tools that should be added, removed, or updated).

SANS DevSecOps Curriculum

SANS Software Security seeks to ingrain security into the minds of every developer in the world by providing world-class educational resources to design, develop, procure, deploy, and manage secure software.

Looking for a soft copy of the DevSecOps Practices poster? You can download the DevSecOps Practices poster on the SANS site.

Interested in contributing your DevSecOps success stories to the SANS DevSecOps Blog? Let us know and we'll share them with the community.

Common Errors

Hardware

"The host supports Intel VT-x, but Intel VT-x is disabled"

One reason starting the virtual machine may fail is having Intel VT-x disabled. If this is the case, you will see an error message about "The host supports Intel VT-x, but Intel VT-x is disabled". VT-x has to be enabled in your system BIOS, which slightly differs from system to system. You may need the instructor's help to find out how to fix this.

Newer Systems (UEFI, not traditional BIOS)

1. Hold the SHIFT key while restarting your Windows System.
2. The system will restart and end up on a blue screen with various options. Select "Troubleshoot".
3. From the "Troubleshoot" menu, select "Advanced Options".
4. Select "UEFI Firmware Settings".
5. Select "Restart".
6. The system will restart again and give you the option to change your BIOS settings, usually by pressing F10. But watch the startup screen carefully for instructions.
7. Continue at Step 3 below.

Older System (traditional BIOS, not UEFI)

1. Reboot the system.
2. During boot, watch for which key to use to enter the BIOS settings. Common keys include F1, F10, and Delete.
3. Once in the BIOS settings, look for "Virtualization Technology", "VT-x", or similar options and enable them. You can often find them on the "System Configuration" or "Security" page.
4. Exit and save your changes.

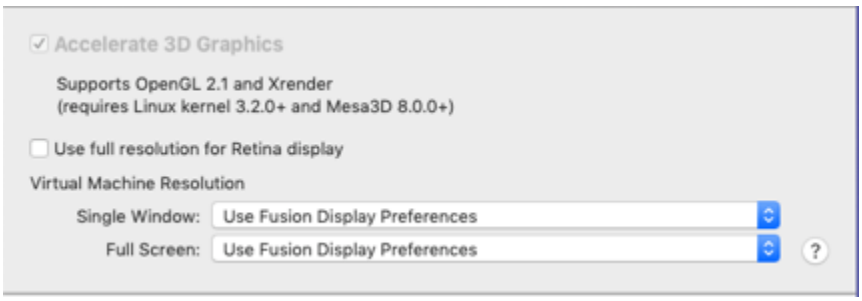
The exact wording and instructions may change from system to system. You will need to have access to your BIOS settings. Some corporate configurations password-protect the settings. You will need to obtain the password to change them.

High-Resolution Screens

Some modern laptops use very high-resolution screens ("Retina Screens"). This can lead to hard-to-read fonts within the virtual machine. The simplest way to make the screen readable is by reducing the resolution of your screen in Windows, which will essentially turn off the high-resolution feature.

Apple laptops with high-resolution screens running MacOS and VMware Fusion do not suffer from this problem. If you do experience the issue, double-check your display settings:

1. Select "Virtual Machine" -> "Settings..." from the VMware Fusion menu.
2. Select "Display".
3. Verify that your settings match the settings in the image below.



VMware Fusion Display Settings

Windows

Windows Credential/Device Guard

With Windows Credential Guard enabled, Windows will run in a Hyper-V virtual machine. This isn't obvious but will prevent VMware from running (VMware cannot run inside a virtual machine). You will see an error like:

VMware Workstation and Device/Credential Guard are not compatible. VMware Workstation can be run after disabling Device/Credential Guard.

To disable credential guard, follow these steps (copied from Microsoft's knowledge base article at https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard-manage.

1. From the Group Policy Management Console, go to "Computer Configuration" -> "Administrative Templates" -> System -> "Device Guard".
2. Double-click "Turn On Virtualization Based Security", and then click the "Enabled" option.
3. In the Select Platform Security Level box, choose "Secure Boot" or "Secure Boot and DMA Protection".
4. In the Credential Guard Configuration box, click "Enabled with UEFI lock", and then click OK.
5. Close the Group Policy Management Console.

   To enforce processing of the group policy, you can run

   ```
   gpupdate /force.
   ```

Linux Cheat Sheet

To navigate the virtual machine, you will need some basic Linux skills.

The main user used for most exercises is "sec522". Dev522's home directory is /home/sec522.

To open a terminal, click on the Terminal icon on the left. This area on the left is also referred to as the "Favorites" area.

1. Terminal Icon

   Once a terminal opens, you may navigate directories using the "cd" command.

   To edit a file, you can just use "gedit" followed by the filename. For example,

   ```
   gedit /tmp/testing.txt
   ```


will edit the file "testing.txt" in the /tmp directory. Gedit is similar to Windows Notepad.

For a more full-featured editor, we have Microsoft's Visual Studio Code available. You can start it by typing "code" on the command line, or by clicking on the icon on the left.

2. Microsoft Visual Studio Code Icon

Preparing the Virtual Machine

The exercises for this course use a virtual machine. In order to install the virtual machine, you will need to copy the "**SEC522VM**" directory from the provided USB drive to your system. This directory is large (about 20GB) and copying will take a while. Make sure you have enough free space on your hard disk. You will need about 40GB for this course to work well.

It does not matter where you place this directory; however, we recommend placing it on the desktop to make it easy to find later.

Please unplug the USB drive after you have copied the files. Your USB drive can be used to restore the virtual machine in case something goes wrong later.

IMPORTANT: We avoid the use of what is commonly referred to as "Hacking Tools" or "Malware". However, some of the tools that we do use may not be allowed by your organization's policy. Please refer to your organization's policy before using these tools outside of class. During our testing, the virtual machine files did not trigger anti-malware software. However, this can change at any time due to ongoing signature updates. You may need to disable anti-malware to allow the virtual machine to function.

No network connectivity is required for class. All exercises are self-contained within the virtual machine.

1. Copy the "**SEC522VM"** directory from the USB drive to your Desktop.
2. Inside the "**SEC522VM**" directory, double-click **ClickHere.vmx**. The virtual machine will start and automatically log you in. You may not see the .vmx extension if you configured Windows not to show extensions.
3. Click "Power on this Virtual Machine".

   ! DO NOT APPLY ANY UPDATES !

4. After starting the virtual machine, you will be logged in automatically. In case the screen saver locks the machine, or if you are prompted for a username and password, use:

Account	Password
student	StartTheLabs

SEC522 G01_01 Errata Patches

HTTP Basics

Objectives

Estimated Time: 20 minutes

PART 0: ***(IF YOU ARE NOT ALREADY FAMILIAR WITH BURP)***

In this exercise, we will use the Burp proxy to intercept HTTP requests and responses. The goal of the exercise is to become familiar with Burp.

PART 1:

Learn how to manipulate requests and responses with Burp and how invalid requests and responses are affecting the browser and the server.

PART 2:

Understand how HTTPS requests can be intercepted with Burp and what limitations may exist.

Requirements

- Lab VM

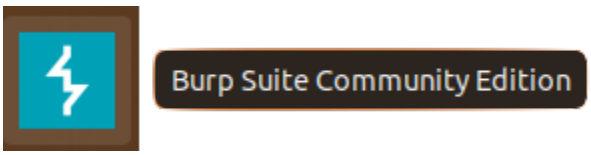
Part 0: Introduction to Burp *(If You Are Not Already Familiar with Burp)*

Step by Step

Burp Community Edition is a free interception proxy created by PortSwigger. It can be used to inspect and manipulate HTTP requests, as well as HTTP responses. Any requests being sent by the browser can be intercepted by Burp. You will be able to edit requests before they are passed to the server. The reverse is also true: Any response from the server can be edited before the browser receives it.

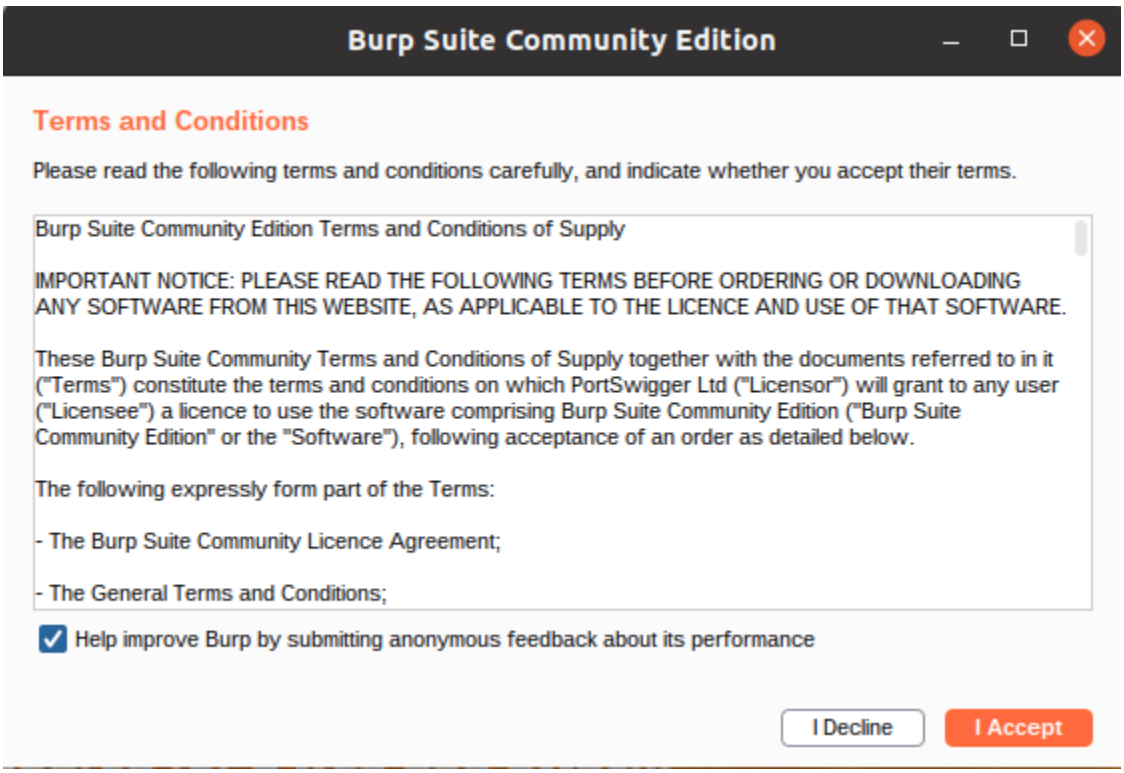
Burp Overview

1. Start **Burp** by clicking on the icon on the left.



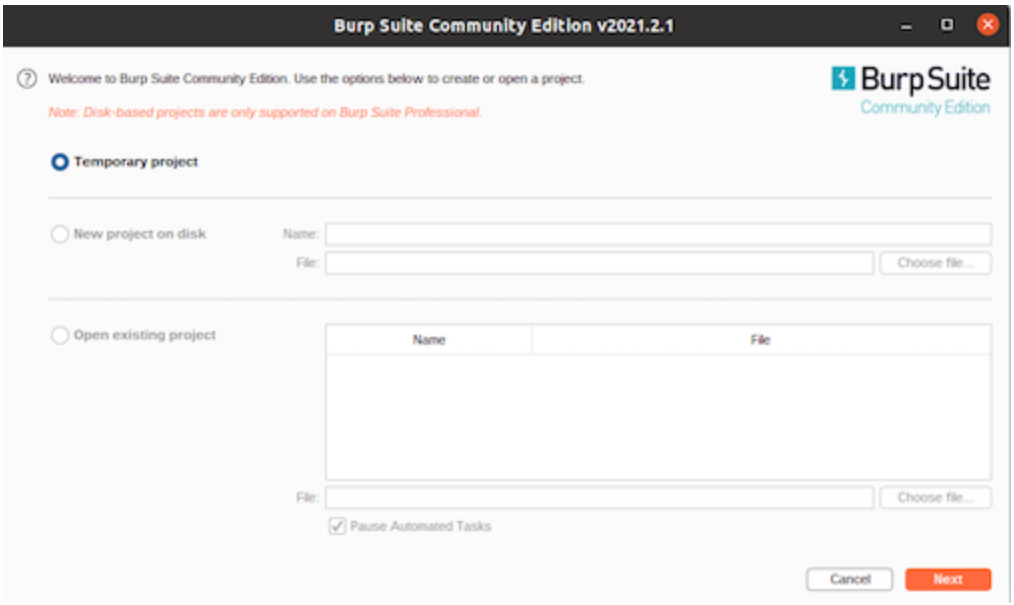
2. Accept License

   Click on "I Accept" on the Terms and Conditions prompt.

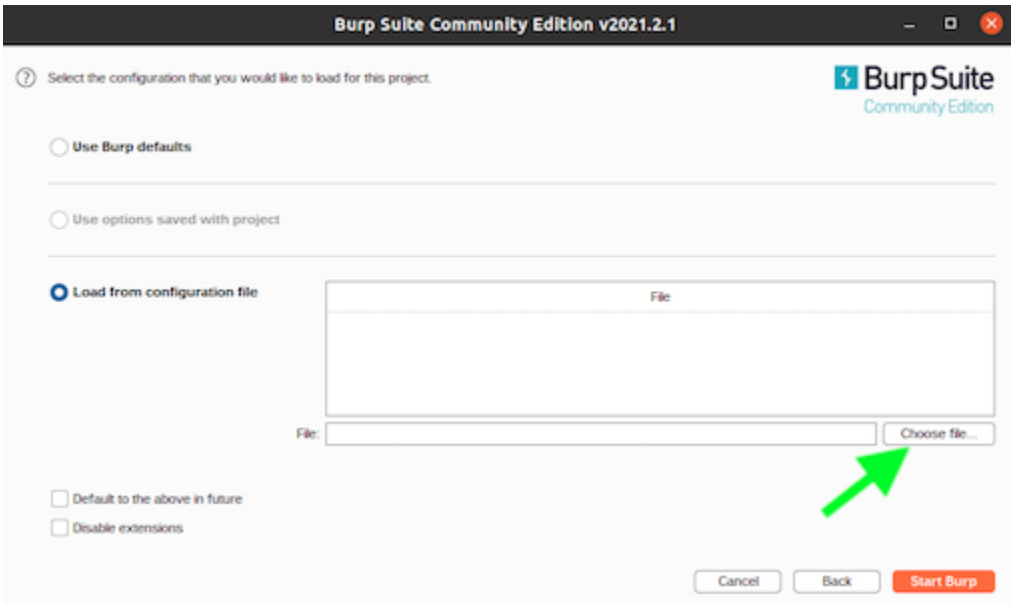


> ***DO NOT INSTALL ANY UPDATES!***
>
> Please do not install any updates if prompted.

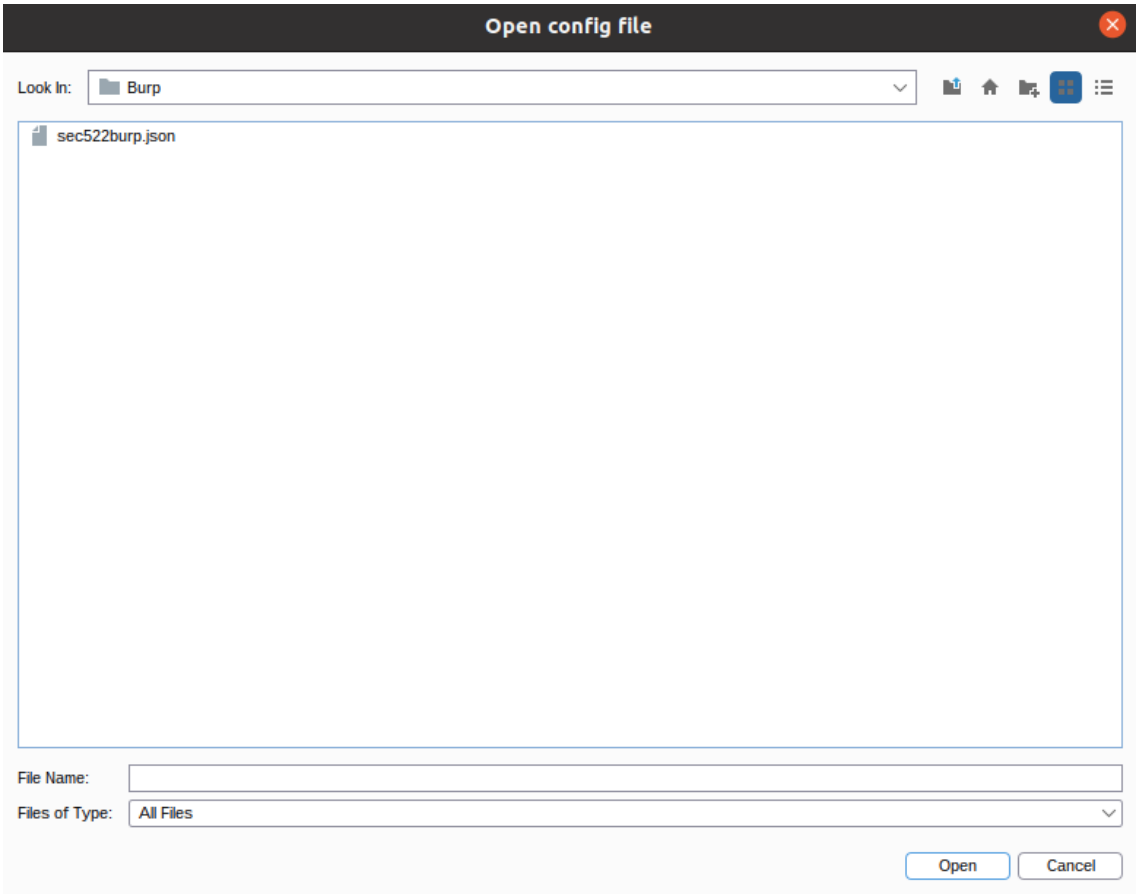
The Burp Community Edition only allows for temporary projects. Click "Next" on the Welcome screen.



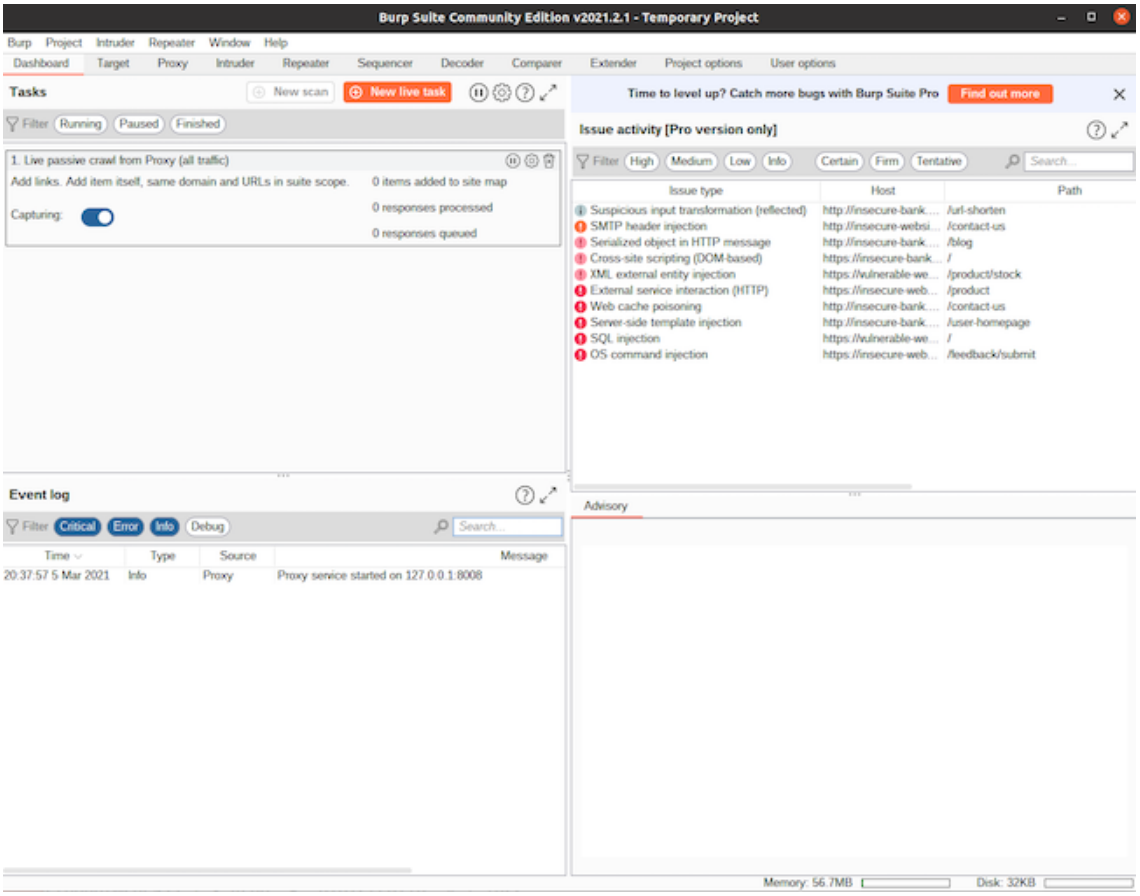
3. Next, select "Load from configuration file" and then select "Choose File" on the right hand side.



The file is located in the "Documents" directory and then "Burp" directory. The prepared configuration file is "sec522burp.json". Select to open the file and then "Start Burp" (the orange button on the lower right hand corner)

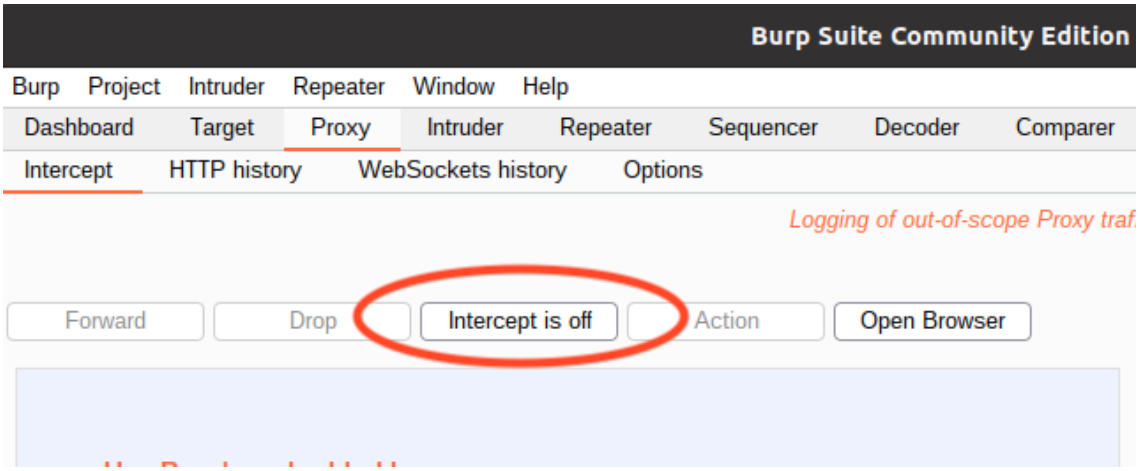


Eventually, you will see the main Burp window.



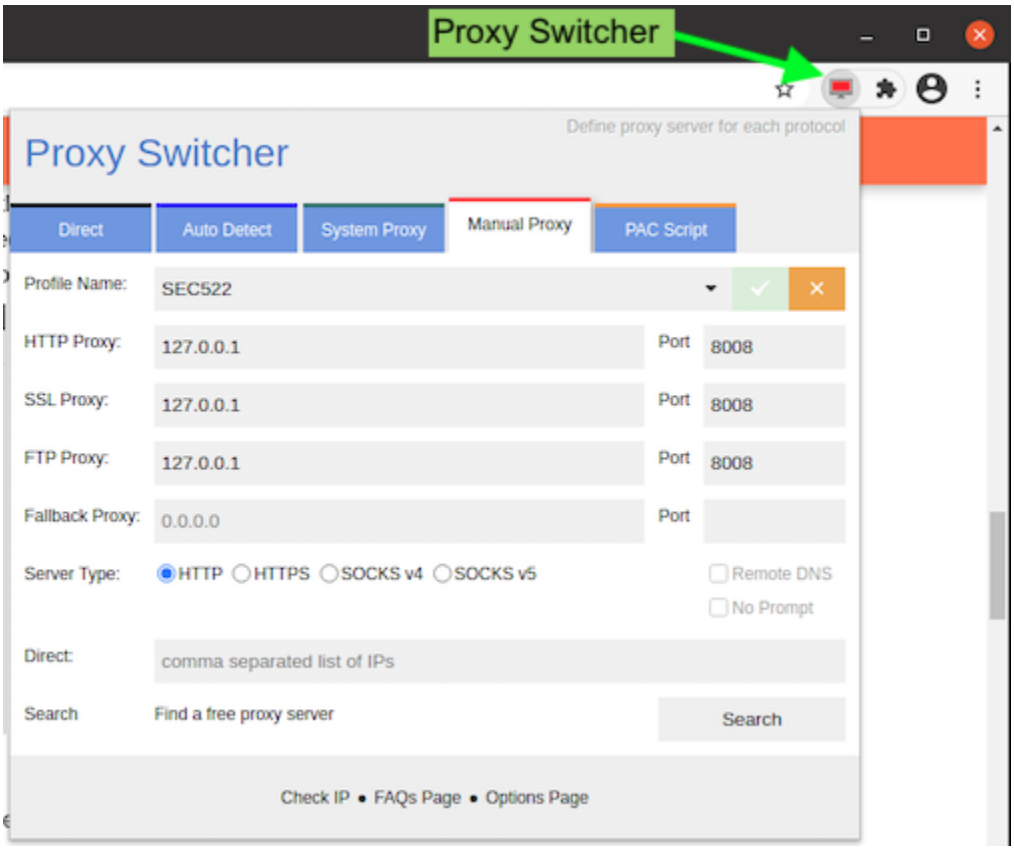
Main Burp Window

4. For now, we want to disable interception via Burp. Instead, we would like to have Burp record requests and responses without holding them. Select the "Proxy" tab and make sure the "Intercept" button states "Intercept is off". Click the button to toggle it from "on" to "off".



5. Start the **Chrome** web browser by clicking the Chrome icon on the left.

6. After Chrome starts, use the Proxy Switcher plugin to select Burp as a proxy. The Proxy Switcher plugin can be found in the top right corner of Chrome. The icon is gray to indicate that no proxy is selected, and Chrome connects directly. Once you select a proxy, the icon will be red. Click on the icon to open the settings. Select "Manual Proxy" to activate the proxy. "Burp" should be the active selection in the "Profile" dropdown.



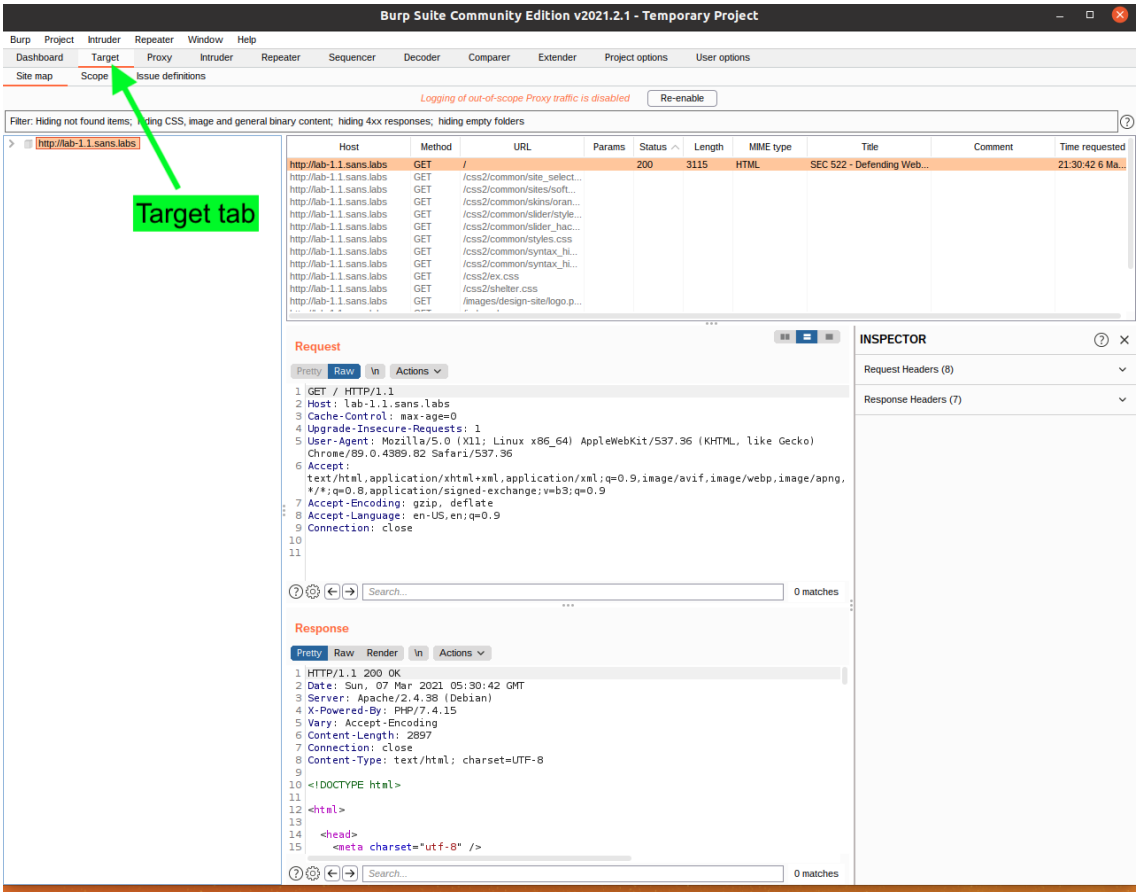
7. Proxy Switcher Configuration

   If you wish to turn off interception later, just select "Direct" from the Proxy Switcher dialog.

Proxy Switcher

Proxy switcher gives you an easy means to toggle between traffic directly going from browser to website or routing through the Burp proxy. When the Proxy Switcher icon is red, you know the traffic is going through the Burp proxy. If you toggle the Proxy Switcher to route traffic to Burp and Burp is not running, that could lead to the browser traffic to time out.

8. In Chrome, browse to http://lab-1.1.sans.labs/ (expect a small delay on this first page load)
9. You are now presented with an Animal Shelter web application.
10. If you pass these tests, switch back to Burp by clicking on the Burp icon **in** the "Favorites" list on the left (the bottom one). At this point, Burp is recording requests and responses, but not allowing you to alter them. You can inspect the requests and responses by selecting the "Target" tab.



11. Burp Target Tab

    Highlight the request for "/" (likely at the top).

Use the information from the target tab to answer the following questions:

What method was used to request the page?

GET

What HTTP version did the browser use?

1.1

What browser (and version) was used?

Chrome 89.0.4389.90

What language does the browser prefer?

en-US

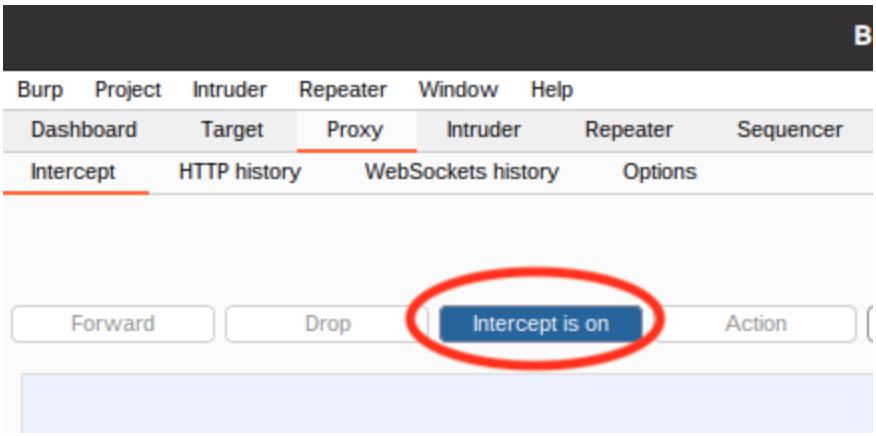
Which header is used specifically to tell the server that this browser prefers HTTPS?
(extra credit)

Upgrade-Insecure-Requests: 1

Part 1: Manipulating HTTP Requests

Step by Step

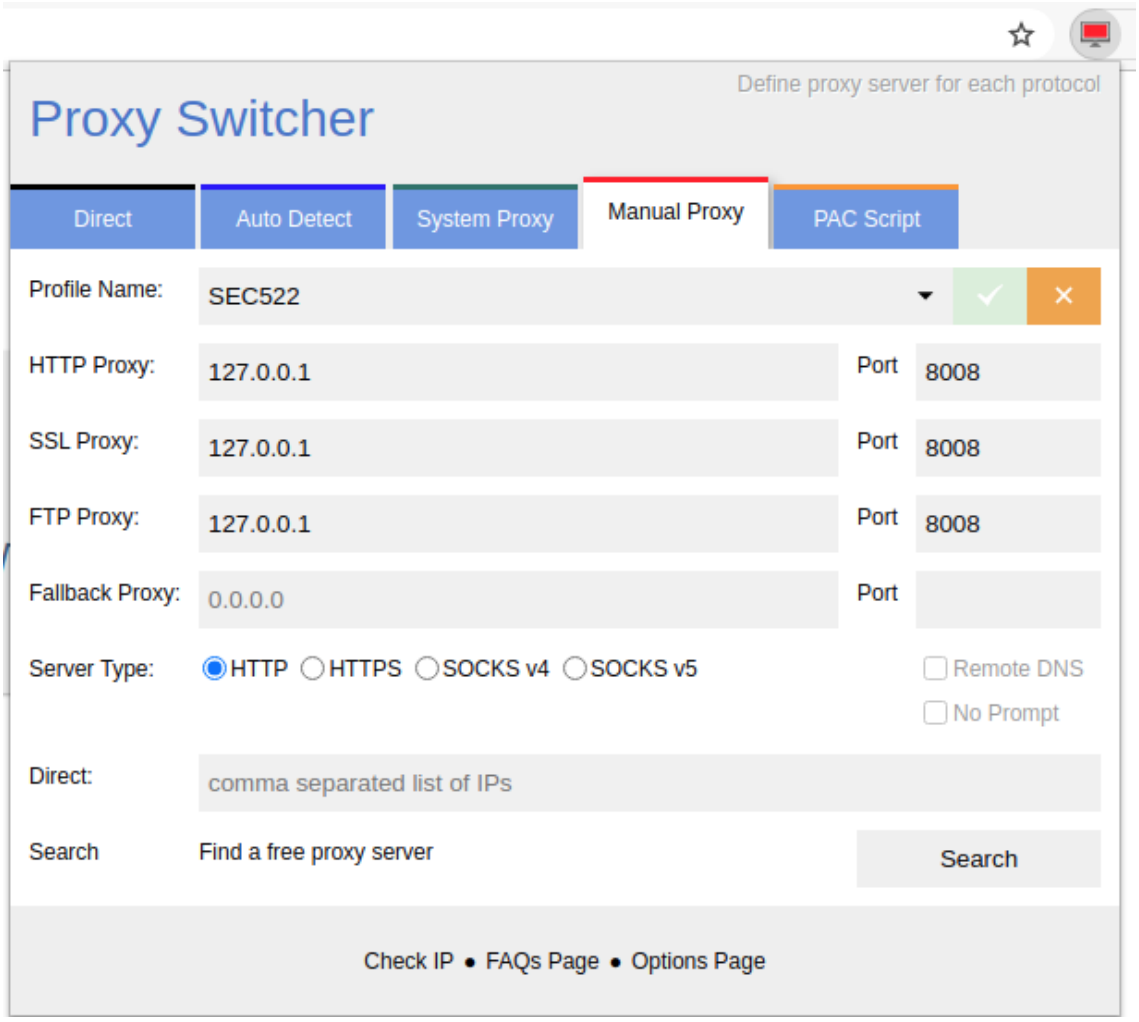
1. Load http://lab-1.1.sans.labs/inspect.php in Chrome. This page is also accessible by clicking on the link (Inspect Headers) at upper right hand corner of the Animal Shelter main page.
2. Make sure Burp is running
3. In Burp, select the "Proxy" tab, and make sure "Intercept" is "on". If not, click on the button to enable it.



Enabling Interception in Burp

> **Warning**
>
> After interception is enabled in Burp, and Burp is enabled as a proxy using Proxy Switcher in Chrome, all requests will be held by Burp. If you refresh your browser, the connection will appear to "hang" until you forward the request in Burp.

4. Enable Burp as a proxy in Chrome via Proxy Switcher.



5. Enabling Burp as a Proxy in Chrome via Proxy Switcher
6. Reload the page in Chrome by clicking the

7. Switch to Burp (click the Burp icon at the bottom of the left "Favorites" bar)

   You will now see the request held by Burp. For now, we just want to make sure everything is working right.
8. Click "Forward".
9. Switch back to Chrome.
10. Reload the page again.
11. Switch to Burp. This time remove the "Host" header from the request. Make sure you remove the entire line. There should be no empty line left in the header.

12. Click "Forward".

What error message does Chrome display?

400 Bad Request

13. In Burp, select the "HTTP history" from the "Proxy" tab and select the last line to answer the next few questions. The last line should represent the request you just edited. You will find the answers by selecting the "Response" tab in the middle of the Burp window.

What status code did the server use to respond?

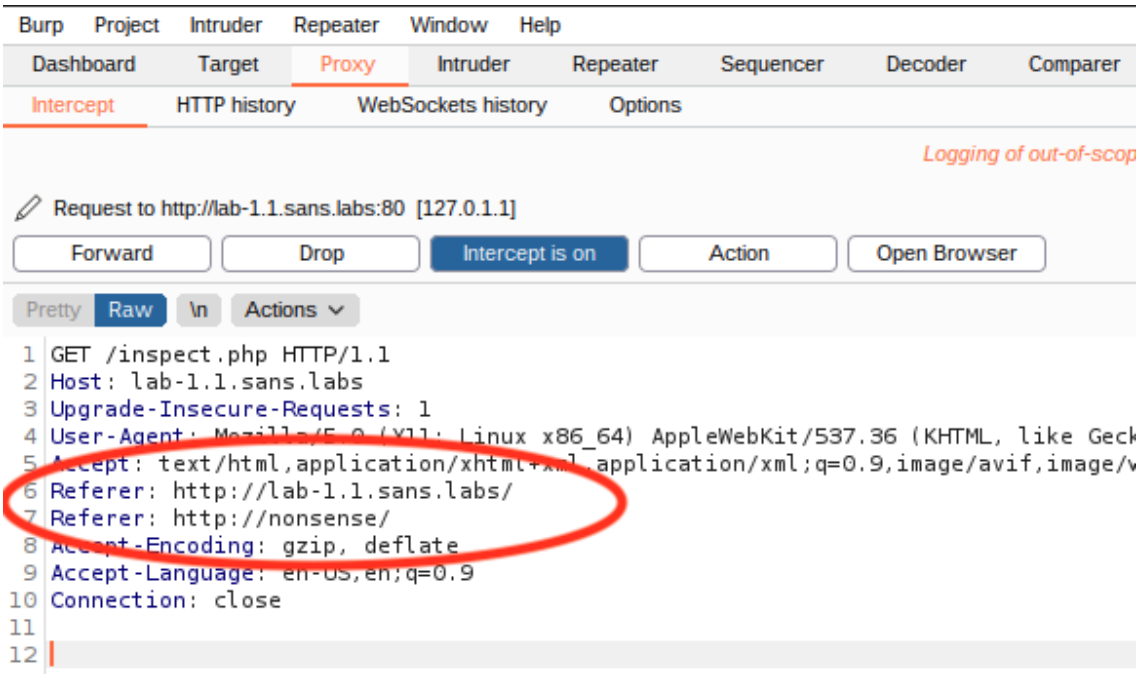
400

What server version was used?

Apache/2.4.38 (Debian)

14. Reload the page once more.
15. Switch back to the "Intercept" tab in Burp.
16. In Burp, add a second "Referer" header with arbitrary content. (If there is no Referer header, then add two "Referer" headers to the request). For example:

```
Referer: http://www.test.com
Referer: Some random nonsense
```

Request in Burp with two "Referer" headers prior to clicking "Forward".

17. Once the response is returned, the page will display the Referer header as the web application received it. What do you see? The content of the first, second, or both headers?

> **Which data does the application receive for the 'Referer' header?**
>
> Both, separated by a comma (,)

Part 2: Intercepting HTTPS Requests

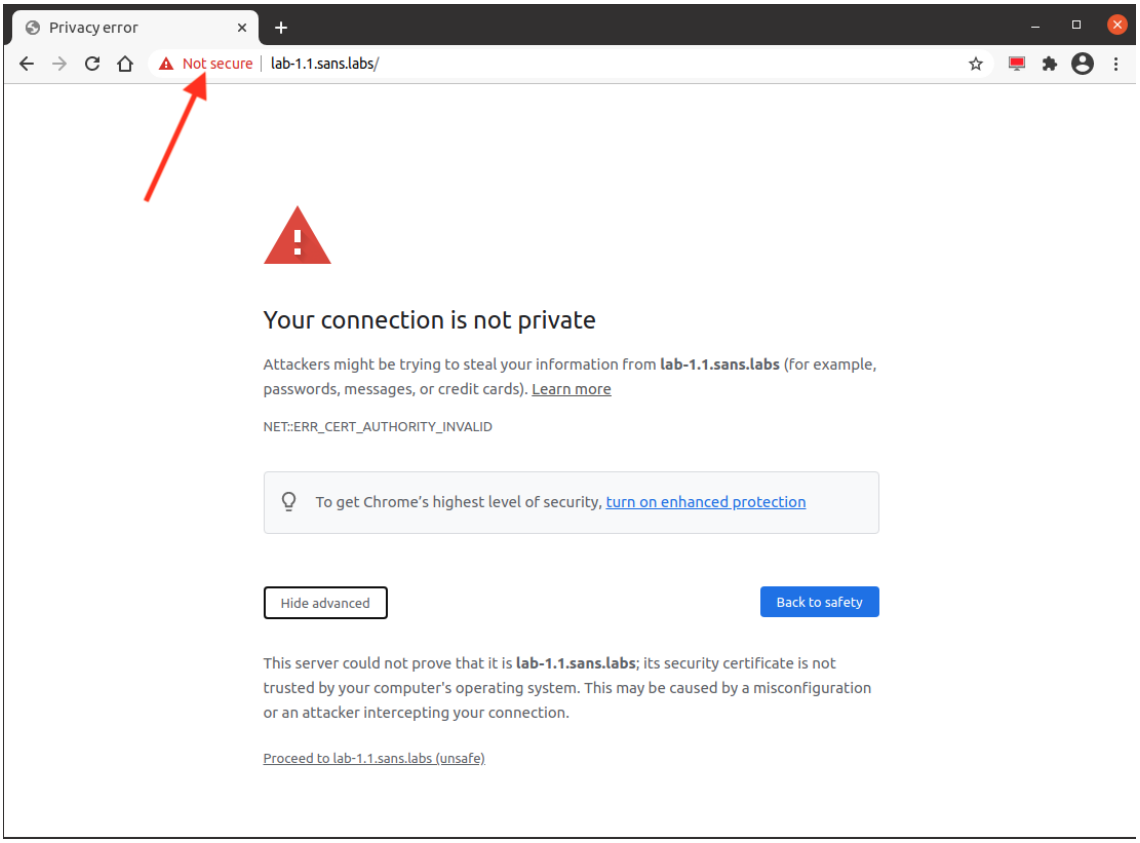
Step by Step

Load an HTTPS page via Burp. Inspect the certificate and explain why a warning is displayed.

1. Keeping Burp enabled from the prior exercise, but disable "Interception" (in the "Proxy" menu check the "Intercept" tab and toggle the "Intercept on/off" button) and visit any HTTPS page. For example, https://lab-1.1.sans.labs/.

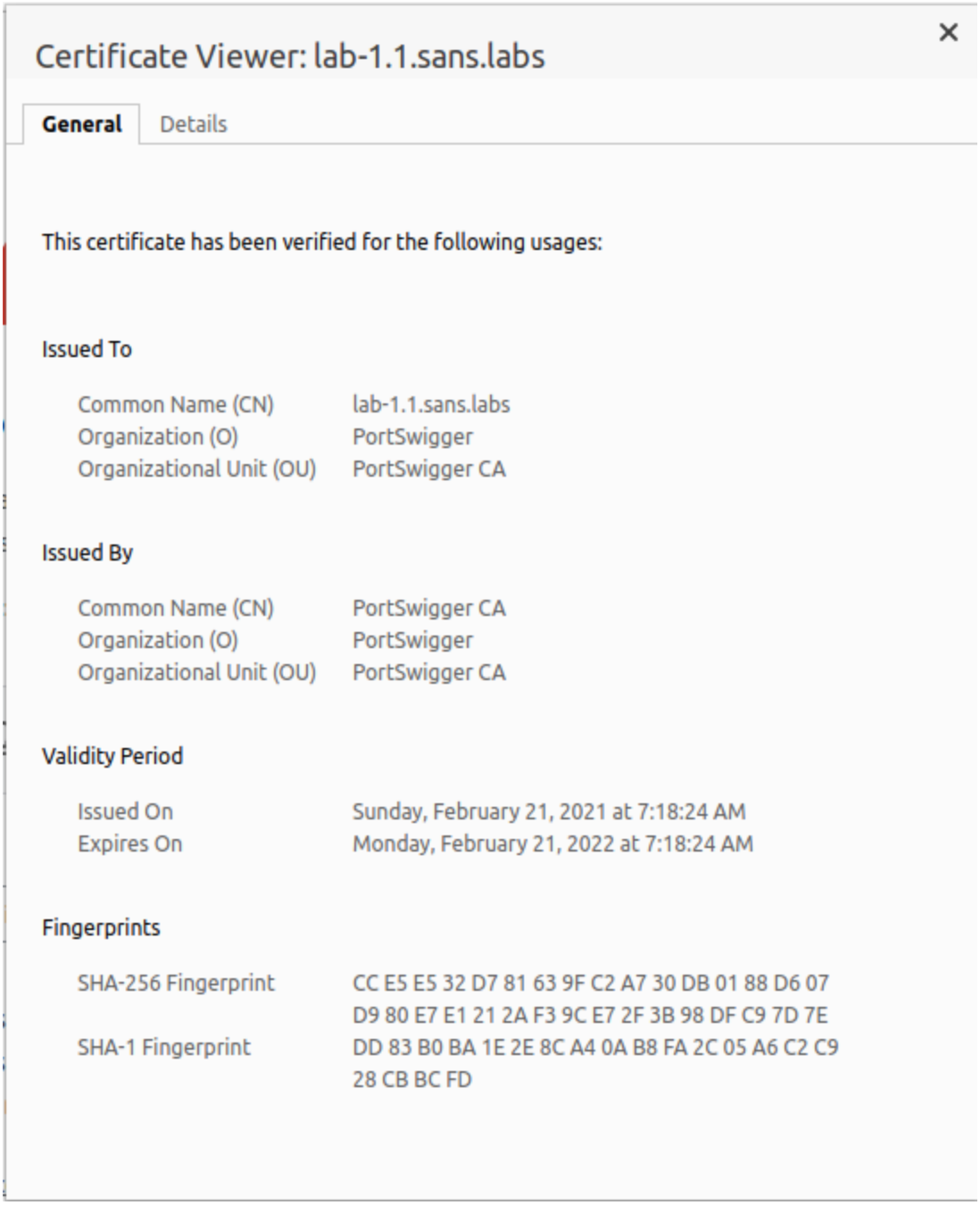
> **You will receive a warning. Why?**

2. Click on "Advanced".
3. Inspect the certificate by clicking on "Not secure" to the left of the URL.



TLS Security Warning.

4. Click on "Certificate (invalid)". This will display the certificate information:



Burp Certificate

Who issued the certificate?

PortSwigger CA

Which hostname is covered by the certificate?

lab-1.1.sans.labs

Conclusion

- Burp is a man-in-the middle proxy, allow you to intercept and manipulate requests prior to sending the request to server and before the response is received by the browser.
- Browser handles the request and response header, generally without the user being aware of their existance.
- The request headers have an impact on how the server process the request. The reponse headers also have an impact on the browser's behavior as well.

Explore Further

- What is HTTP response code 418?
- With interception on, try the following links https://lab-1.1.sans.labs/images/ and https://lab-1.1.sans.labs/images. What are the differences?

Inspecting HTTP/2 Traffic and Crafting Requests

Objectives

Estimated Time: 10 minutes

PART 1:

In this exercise, you will inspect HTTP/2 traffic to better understand how it differs from HTTP/1.1

PART 2:

First, you will try to use an authentication token retrieved in one request to send a second request. The section half of this exercise will teach you how to impersonate a valid browser using simple tools like "curl". You will first collect a request from a regular browser to create a "fingerprint" of the browser. Next, you will use "curl" to create a request that matches the fingerprint.

PART 3:

This exercise is an extension to the previous two exercises where the crafting of headers have to be exact.

Requirements

- Lab VM

Part 1: Inspecting HTTP/2 Traffic

No Hints

You will be using Wireshark to inspect an HTTP/2 packet capture. The packet capture can be found in /home/sec522/Documents/pcaps/http2.pcap. Find out which headers you can identify in the HTTP/2 request and response.

Step by Step

1. Open Wireshark by clicking on the blue "fin" on the left.

Wireshark icon

2. Open the packet capture using either Ctrl+O or "File" > "Open" from the Wireshark menu. Navigate to /home/sec522/Documents/pcaps and open the file "http2.pcap" (you may find the file still listed as a "Recent" file, which may make things easier).

3. Select packet #15 and investigate the middle pane of Wireshark. Expand the "Hypertext Transfer Protocol 2" section by clicking on the triangles to the left until you see the headers:

```
▾ HyperText Transfer Protocol 2
  ▾ Stream: HEADERS, Stream ID: 1, Length 40, GET /index.php
      Length: 40
      Type: HEADERS (1)
    ▸ Flags: 0x05
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identif
      [Pad Length: 0]
      Header Block Fragment: 82048860d5485f2bd73d7f86418bf1e3c
      [Header Length: 138]
      [Header Count: 6]
    ▸ Header: :method: GET
    ▸ Header: :path: /index.php
    ▸ Header: :scheme: http
    ▸ Header: :authority: www.sec522.org
    ▸ Header: user-agent: curl/7.58.0
    ▸ Header: accept: */*
```

Wireshark Details

4. Using Wireshark, answer these questions about the request:

Which user agent was used to collect the data?

curl/7.58.0

Which hostname did the client connect to?

www.sec522.org

5. Now select packet #21 and again inspect the HTTP/2 part in more detail. This packet contains the response.

What status code did the server return?

200

Which server version did we use?

Apache 2.4.29

Feel free to inspect any additional items in this sample.

Part 2: Acquire and play back token

No Hints

The CSRF/ proof service at http://lab-1.2.sans.labs/authservice/index.php gives you a special token

Step by Step

1. Open a terminal (click the "Terminal" icon on the left)
2. Use "curl" to send a PUT request to http://lab-1.2.sans.labs/authservice/index.php

```
curl -iX PUT http://lab-1.2.sans.labs/authservice/index.php
```

The return will include an "Auth-Token" header:

```
HTTP/1.1 200 OK
Date: Thu, 11 Mar 2021 02:56:38 GMT
Server: Apache/2.4.38 (Debian)
X-Powered-By: PHP/7.4.15
Auth-Token: [use this value for the next step]
Vary: Accept-Encoding
Content-Length: 89
Content-Type: text/html; charset=UTF-8
```

3. Now create a request to http://lab-1.2.sans.labs/authservice/test.php using this Auth-Token header (replace "TOKEN" with the value you received in step 2.

```
curl -i -H 'Auth-Token: <Token you got earlier>' http://lab-1.2.sans.labs/authservice/
test.php
```

   The return will let you know if you got it right

Part 3: Spoofing an HTTP Request

No Hints

The exercise starts at http://lab-1.2.sans.labs/spoofchallenge/index.html . The page includes brief instructions showing you how to proceed.

1. Open http://lab-1.2.sans.labs/spoofchallenge/index.html in Chrome.
2. Click on the "Step 1" link. This will open the page fingerprint.php and display your browser fingerprint.
3. Click "Save Fingerprint". You will now see the saved fingerprint in the lower half of the page.
4. Open a terminal window (for example by clicking on the "Terminal" icon on the left)
5. Enter:

```
curl http://lab-1.2.sans.labs/spoofchallenge/fingerprinttest.php
```

   You will see a number of errors due to missing or incomplete headers.

6. Now add individual headers using the curl "-H" option. For example to add the User-Agent header use (the URL in the end is abbreviated. Use the same URL as in step 5).

```
curl -H "User-Agent: [some user agent]" http://www...
```

7. Add more headers. Read the error messages for any missing headers, or values that you do not have right yet.
8. As you add the "Accept:" header, the output may be compressed and you will get a warning indicating that the output is binary. Pipe the output to "zcat" if you see the warning. For example like in the abbreviated command below:

```
curl -H [... any headers... ] http://[url] | zcat
```

9. For a solution, see http://lab-1.2.sans.labs/spoofchallenge/solution.php

Conclusion

- HTTP 2 is the current version of HTTP standard which has the original protocol on much more optimized basis.
- HTTP Request headers manipulation is a big part of security defense and offense. HTTP headers in both request and response affects the functions of browser or server. In modern day applications, the headers are where authentication tokens and security decisions are transported.

Explore Further

- Can you use HTTP 2 over cleartext?

Service Isolation

Objectives

Estimated Time: 15 minutes

PART 1:

In this exercise, you will learn about the importance of isolating services like FTP and HTTP. We will also experiment with web application firewalls to demonstrate how they can prevent sensitive data leakage.

PART 2:

Many organizations deploy web applications in docker containers. Docker containers are administered via an API. Exposing the API, or tools that use it, will expose your Docker containers to attack. In this exercise, we will experiment with a common Docker administration tool called "Portainer".

PART 3:

In this part of the exercise, you will experiment the Web application firewall (mod_security) and it's capability to detect data leak post-compromise and view the logs captured by the Web Application Firewall

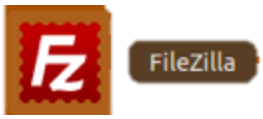
Requirements

- Lab VM

Part 1: FTP/HTTP Server Isolation

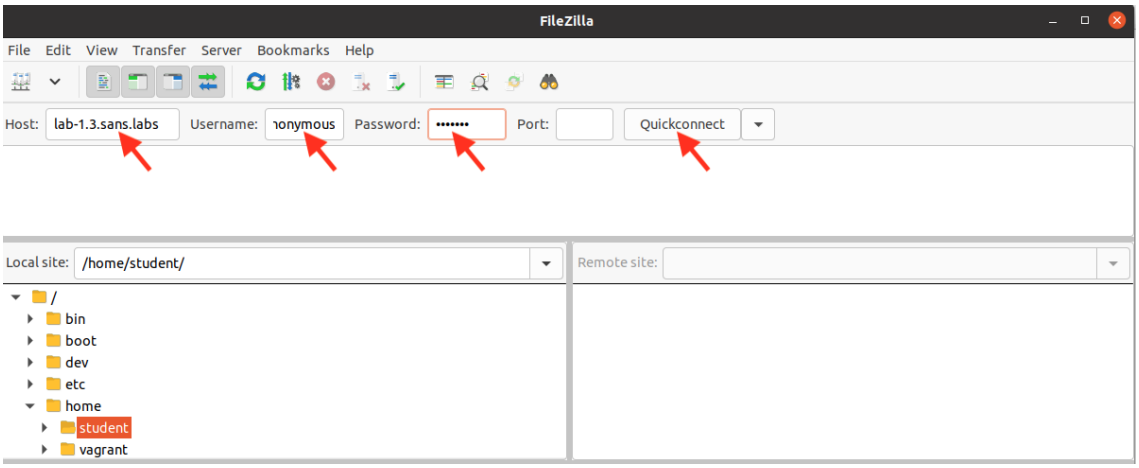
Step-by-Step Instructions

1. In Chrome, load the URL - http://lab-1.3.sans.labs/ftp/. This will activate the exercise components.
2. Click on the **"FileZilla"** icon on the left side of the screen (Favorites bar) to start the FTP client.

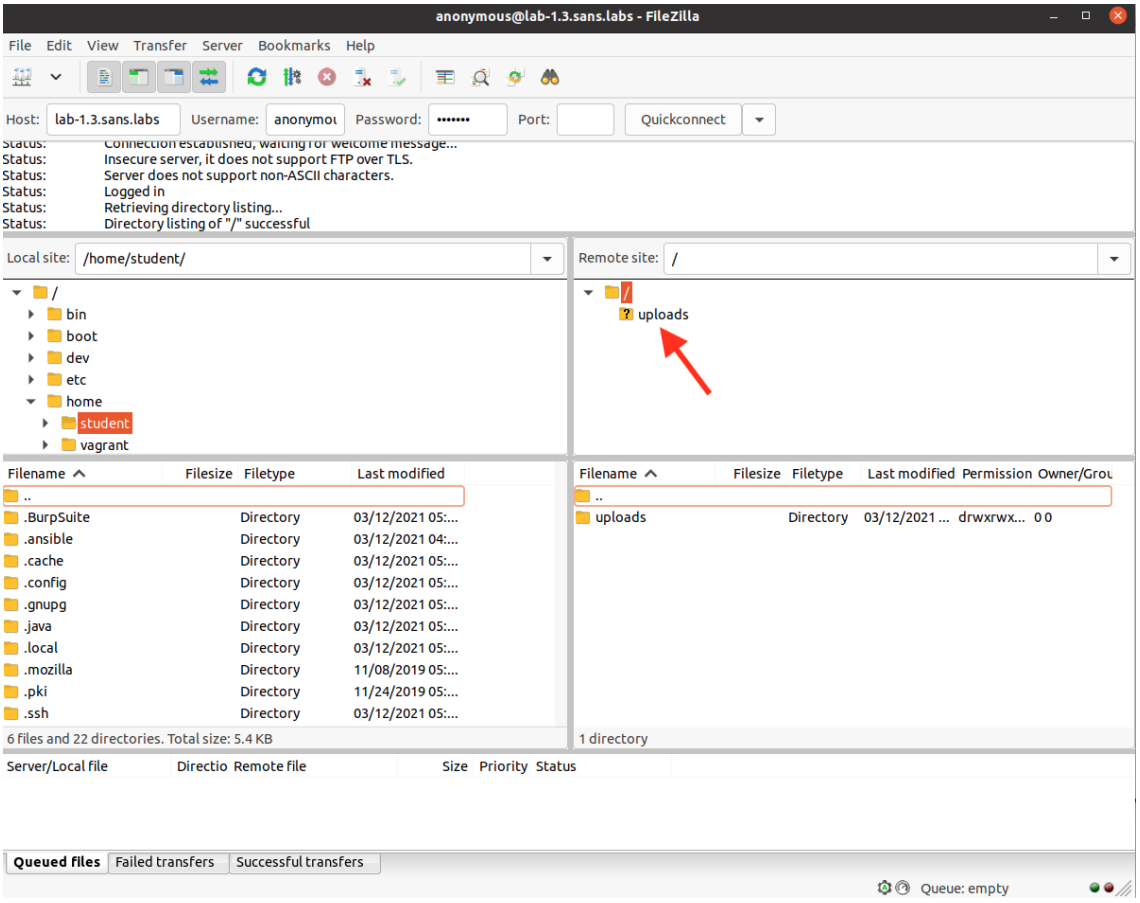


FileZilla Icon

3. Once FileZilla is launched, enter **lab-1.3.sans.labs** as the host, the username as **anonymous** and password as **testing**. Click "Quickconnect" to connect.

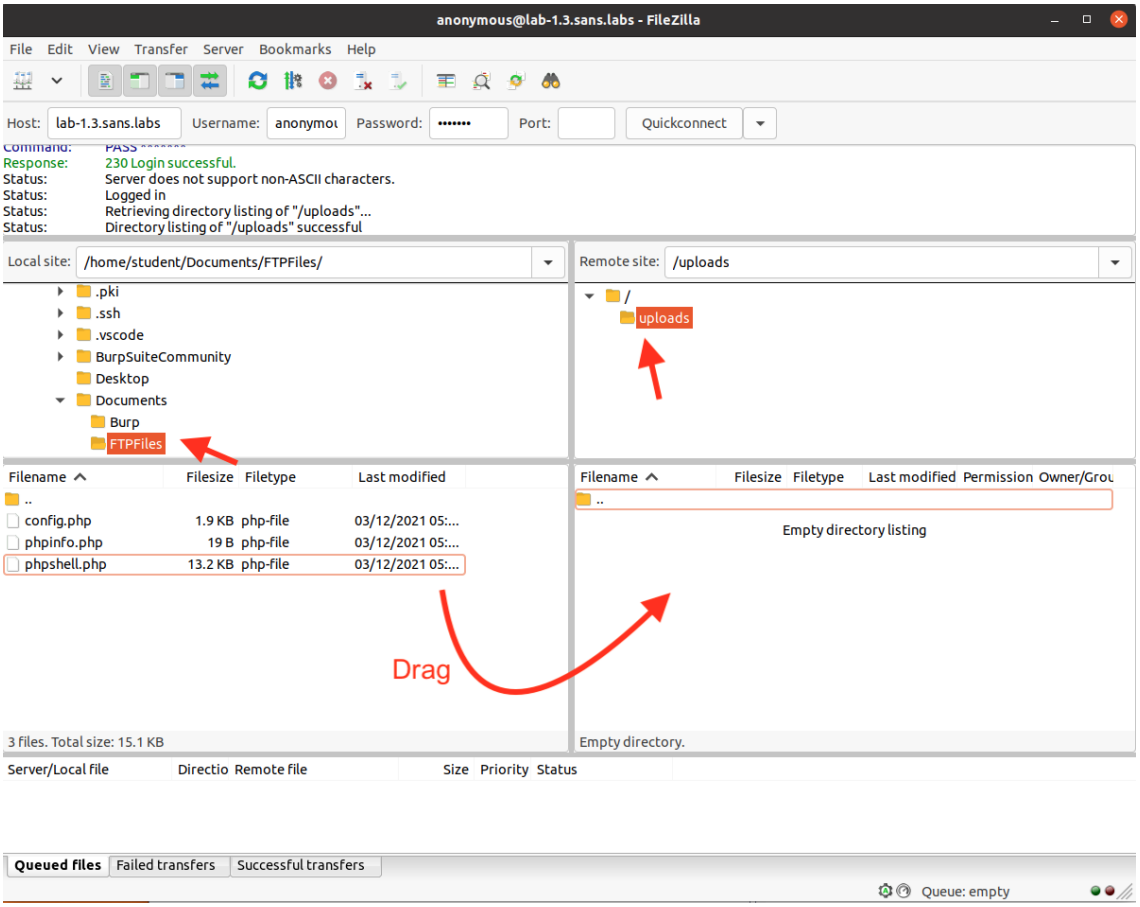


4. After FileZilla is connected, select the "uploads" directory in the lower right-hand pane and set the "Local Site" to "/home/student/Documents/FTPFiles/".

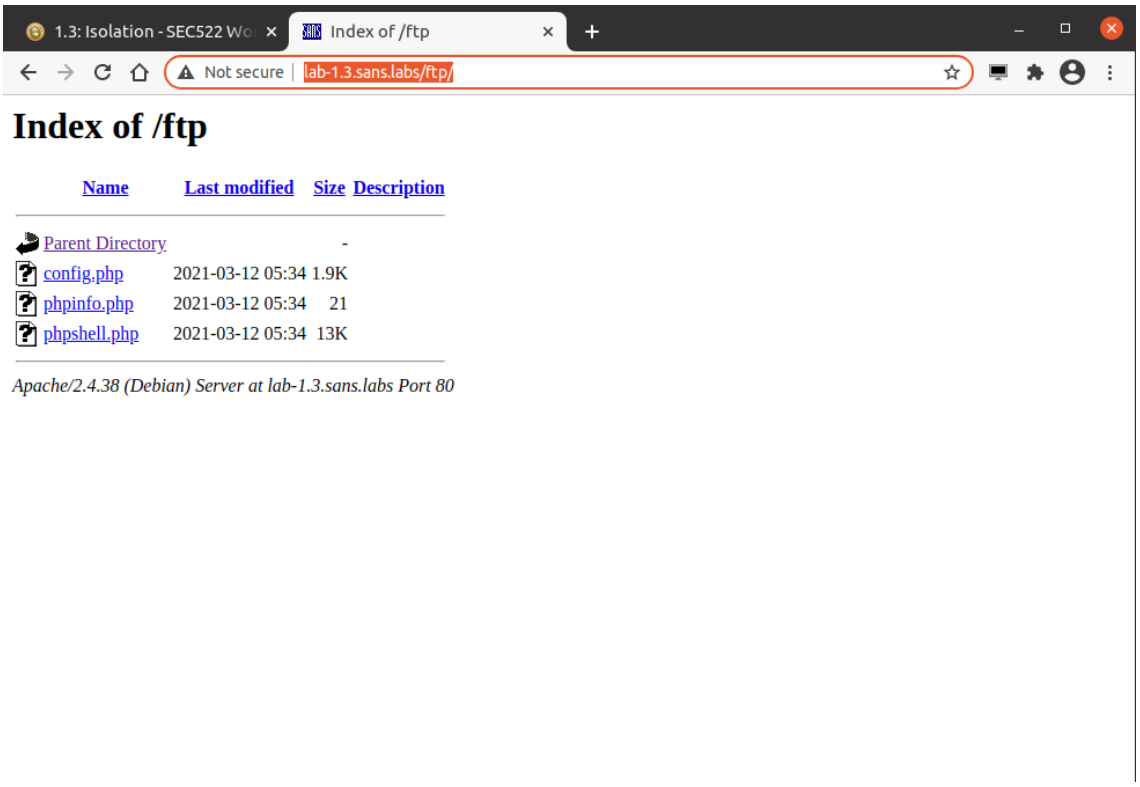


FileZilla Main Screen

5. Highlight the three files (config.php, phpinfo.php, phpshell.php) on the left and drag them to the "Remote Site" pane (lower right, where you clicked on "uploads"). The files are now uploaded.



6. FileZilla uploading files
7. Open Chrome and make sure no proxy is selected.
8. Proceed to **http://lab-1.3.sans.labs/ftp** in Chrome. You will see a directory listing showing the three files we just uploaded.



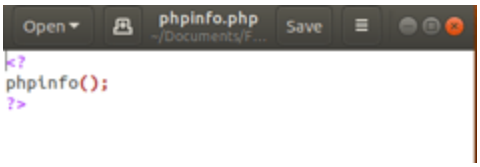
FTP Directory in Chrome

9. Click on **phpinfo.php**. This will execute the phpinfo script we just uploaded. It uses the phpinfo() command to list various php configuration parameters.

> **Note**
>
> The files that we uploaded earlier contained code. These code are executed by the webserver.

10. The PHP configuration is displayed on-screen. Let's inspect what we uploaded via FTP earlier.
11. Open "Files" in the upper left-hand corner of the screen (in the Favorites bar) and navigate to Documents->FTPFiles. Double-click on "phpinfo.php". This will open the file in a text editor.

File



phpinfo.php content

The file contains a single PHP command, "phpinfo()". This command, if executed, creates the output we saw earlier. This proves that the server executed our code.

1. Go back to **http://lab-1.3.sans.labs/ftp** in Chrome and click on **"phpshell.php"**.
2. Log in with the username **sec522** and the password **training**.
3. Once logged in, users can run any shell command. The operating system in this instance is Unix. You can try normal Unix shell commands such as "ls" to show a listing of files or "uname -a" to show details about the operating system. Feel free to explore the interface and see what evil things can be done using this tool. You are essentially controlling the computer through this remote web shell interface and you are able to run any command the web server is permitted to run.

Part 2: Docker Administration Tools

No Hints

Portainer is running on a container within the course environmnent, and you can connect to it via Chrome (http://portainer.sans.labs/). Try to guess the login credentials and add a rogue admin user.

Step by Step

1. Open Google Chrome and visit http://portainer.sans.labs
2. Log in using the username "sec522" and the password "training".
3. Click on "Users" on the left.
4. Review the currently registered users in the lower part of the User Management interface on the right.

How many users are present?

2

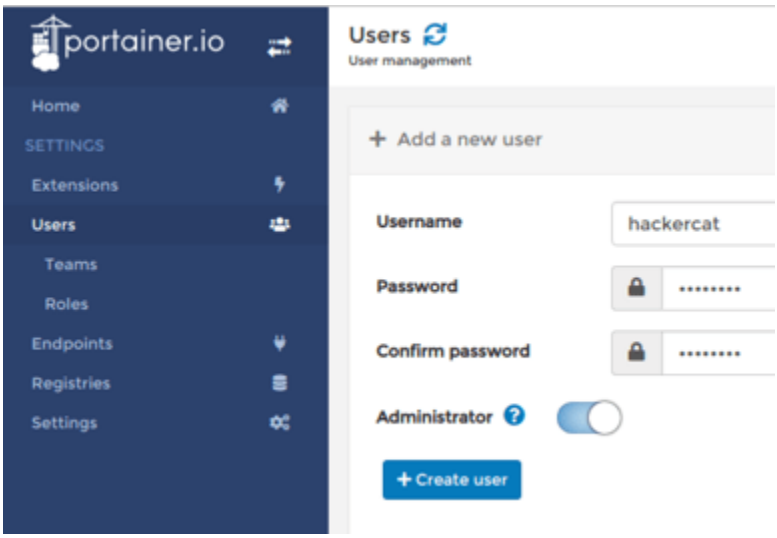
What are their usernames?

admin, sec522

What are their roles?

administrator

5. Enter "hackercat" as the username and "training" as the password. Don't forget to enter "training" again to confirm the password.
6. Check the "Administrator" option.



User Management Dialog in Portainer

7. Click "Create user".

   You now created a new administrative user. Optionally, you may test this user by logging out of Portainer and logging in using the new credentials. An attacker could use this new user to access the system even after the administrator changed the password for the sec522 user.

   !!! question "What measures can an administrator take to prevent this exploit?

Part 3: ModSecurity Web Application Firewall

Step by Step

We will experiment with the mod_security web application firewall in the following exercise.

1. In Chrome, go to **http://normalsecurity.sans.labs** and click on the two files "**Credit Card Data (cc.html)**" and "**Source Code (trans.txt)**".

   What do these two files show?

   cc.html shows transaction data including credit card numbers.

   trans.txt leaks source code.

   Why is it a problem that our application leaks data like this?

   cc.html - leaking credit card data is, of course, a problem.

   trans.txt - an attacker could use this source code to find vulnerabilities.

2. Try http://normalsecurity.sans.labs/form.php. You will be able to submit the string "cats are evil" or other strings for that matter.

   Switch

   We are now switching to the mod_security protected version of the same website

3. Revisit the two URLs above (under **http://modsecurity.sans.labs**). What do you see now? (You may need to reload the page if you still see the original text.)
4. Revisit **http://modsecurity.sans.labs/form.php** and see if you can still enter "Cats are evil". Try "Cats are cute" instead.
5. Review the web application firewall's log file at **http://modsecurity.sans.labs/modsecurity.log**

   Try to identify the log entries that indicate which URL was blocked and why?

6. Next, we are going to examine the ruleset to see what rules are triggered to block access to the file. Open the configuration file at **http://modsecurity.sans.labs/modsecurity.rules**.

> **Try to understand what the rule is matching for.**

Conclusion

- Architecture supporting the application has a huge impact on the overall security. Isolation and segmentation are common approaches to secure the application.
- Web Application Firewall are versatile security tools to implement security at the HTTP level without a lot of programming or coding.

Explore Further

- Think about how to implement Web Application Firewall to a production application.

SSRF and Credentials Stealing Services

Objectives

Estimated Time: 10 minutes

This exercise simulates a cloud compromise due to SSRF vulnerability. You will first exploit the SSRF vulnerability, and then use the retrieved data to access a cloud storage service. You should understand how SSRF can be used to expose internal services and how that information can be used for additional compromise.

Requirements

- Lab VM

No Hints

Start at http://lab-1.4.sans.labs/ssrf.php. The page will provide additional hints.

Step by Step

1. Open http://lab-1.4.sans.labs/ssrf.php in a browser. The page will allow you to add RSS feed URLs. The server will retrieve the feed without validating the source.
2. To test it, enter http://lab-1.4.sans.labs/ as a URL in the Feed URL. You will see the HTML content of the page being retrieved and displayed.
3. There is a meta-data server at 169.254.169.254. Retrieve its index by using http://169.254.169.254 as "Feed URL".
4. Explore the content of the service by retrieving the content of directories (e.g. http://169.254.169.254/latest).
5. The server uses a special local URL for meta data like authentication tokens. Try this URL (important: do not forget the slash in the end).

   http://169.254.169.254/latest/meta-data/iam/security-credentials/cg-ec2-role-cgidmxg5qsmw9b/

6. Copy the entire response (including the enclosing { }) to a file called "/home/student/.sawssecret". You can do this either with the terminal or with Visual Studio Code.
7. Open a terminal and run "s-aws" to gain access to the storage repository. "s-aws" reads credentials from the /home/student/.sawssecret file.

Conclusion

- SSRF can lead to the web server becoming a relay point for sending a HTTP request on behalf of the client. Since the request is sent by the Web server, it could access resources that are only accessible to the Web server such as DMZ or internal network resources.
- In public Cloud environment, specifically the Infrastructure as a Service Computing, the orign of the HTTP request matters a great deal. Through SSRF attacks, it is possible to acquire credentials for the Virtual Machine that the web application is running on and then pivot over to attack/access other Cloud based components.

Explore Further

- Look up the IMDS (Instance Metadata Service) in Cloud Security Provider and determine if they are vulnerable to similar type of flaws.

SQL Injection

Objectives

Estimated Time: 15 minutes

PART 1:

You will familiarize yourself with the basic principles of SQL injection.

PART 2:

You will experiment with an actual application. This will provide us with a realistic example of how SQL injection works and how an attacker may exploit it.

Requirements

- Lab VM

Part 1: Basic SQL Injection

No Hints

We are again using our "Animal Shelter" application. This time, we use a modified page that includes a search feature. The search feature allows for SQL injection. You can check the "Debug" checkbox to see the SQL query. The page can be found at http://lab-2.1.sans.labs/.

Step by Step

1. Open http://lab-2.1.sans.labs/ in Google Chrome.
2. Enter one of the animal names (for example, "Stash") in the search box and click "Submit". Only this animal should now be visible.
3. Enter a single quote (') in the search box and click "Submit". This should trigger an SQL error. The SQL error message will include the query. Try to identify the vulnerability.

Digging Deeper

Critical: The next few commands require you to include a space after the final '--'.

1. Enter the string below in the search field and click submit. You will not see an error now, but all animals are visible.

```
' OR 1=1 --
```

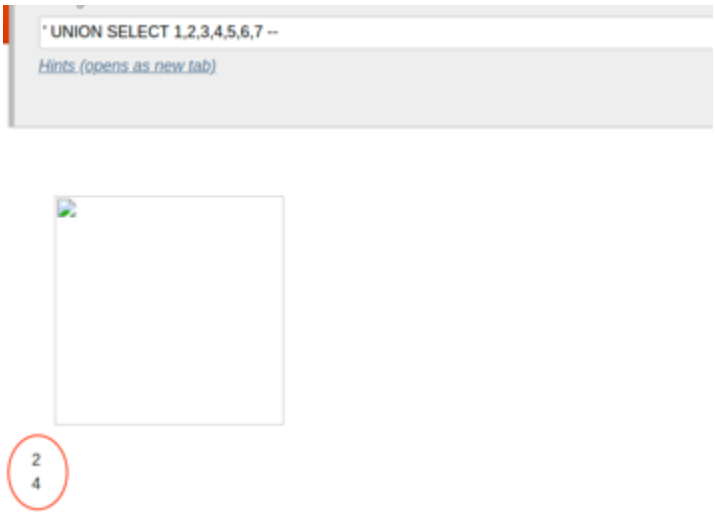
2. Enter the below string and click "Submit".

```
' UNION SELECT 1,2,3 --
```

   The last step will again result in an error message. The error message will state that both SELECT statements in a UNION query need to return the same number of columns. Count the columns returned by the first statement and rewrite the second statement using the right number of columns.

3. Enter the below string and hit submit. You will now get a broken image and the numbers 2 and 4 will be visible below the broken image.

```
' UNION SELECT 1,2,3,4,5,6,7 --
```



Result of Entering the Correct UNION Query

4. Replace the SELECT query with one selecting the username and password column from the animalshelter.users table

```
' UNION SELECT 1,username,3,password,5,6,7 FROM animalshelter.users --
```

Do not forget the space after the -- in the end

You will now see two broken images (one for each user in the database). The usernames and passwords are displayed where you saw the numbers 2 and 4 in Step 6.

5. View the page source (CTRL-U) to see the password hashes more clearly.

   You may try other queries now.

Can you create a query without the '--' in the end?

Answer: **' OR 1='1**

Part 2: phpBB SQL Injection

No Hints

The old version of phpBB we have installed here suffers from multiple SQL injection flaws. In this exercise, we explore an SQL injection flaw in the messaging interface. To trigger the flaw, log in at http://phpbb.sans.labs (username: sec522, password: training) and then use one of the hints to get you started (http://phpbb.sans.labs). The injection will only work if you are logged in. You may also want to explore the login form for additional opportunities to experiment with SQL injection.

Step by Step

We have a URL at hand that we know is vulnerable to SQL injection. The first step to test for SQL injection is to insert a quote into each field.

The following steps will inspect a parameter used in the phpBB private messaging system.

1. Use Chrome to visit **http://phpbb.sans.labs**.
2. Log in using the username "sec522" and the password "training".

3. Click on the "You have no new messages" link at the top of the page.
4. Click on the "Savebox" link.
5. We are going to exploit this page. Open the following URL: **http://phpbb.sans.labs/hints.html**

   We will use the links on this page to explore and exploit the SQL injection vulnerability. Note how exploit URLs on this page will include a URL encoded single quote (%27) as part of the pm_sql_user parameter.
6. We would want to know where in the statement the user input is located. Click on the second link (Exploit 1). Exploit 1 injects HTML into the SQL statement. This will make the injectable location visible in red.
7. Return back to the hints page (**http://phpbb.sans.labs/hints.html**)
8. Read Exploit 2 to see how a union statement is used to join two distinct queries together. Click on the "Exploit 2" link. There is no error in the statement. We created a valid statement. The main purpose of this statement is to ensure the statement runs error free. If the number of fields is different between the two distinct statements, they cannot be "unioned" together.
9. Return back to the hints page (**http://phpbb.sans.labs/hints.html**).

   Exploit 3 simply makes the statement return \'aaa\', \'bbb\', and so on just to see which field can be returned on the screen, as some of the fields are not being returned on screen.

   Following Exploit 3, you will find a group of statements that use SQL injection to enumerate the table structure (infomation_schema holds all this information). One query at a time, information about the structure of the database is being returned; information such as table names and column names is gathered.
10. The "Administrator Exploit" at the end of the page returns the administrator\'s username and password. Please click on this statement. This is dependent on the results of enumerating the table and column names in Step 11.

    We will dive into the source code of privmsg.php and investigate the reasons for the SQL injection. For the non-PHP-coders, do not worry; the code is really simple.
11. Browse to **http://phpbb.sans.labs/privmsg.phps** (note the "s" at the end of the URL) to review the source code.
12. Scroll to line 240 of the source code and review the variable $sql. Pay special attention to lines 226 and 229.

$pm_sql_user on line 240 consists of data from line 226. Line 226 concatenates data from user input in $pm_sql_user with other data on lines 226 to 229. The .= operator means -- leave whatever is already in $pm_sql_user; in addition, also put the following data into $pm_sql_user. The $pm_sql_user variable isn\'t filtered or sanitized; it came straight from the user.

Line 244 is where the user input gets concatenated with other static SQL strings to make the dynamic SQL statement. The resulting statement is passed to the database for execution on line 253.

When raw user input gets to become part of a dynamic SQL query without validation, we know what the story looks like---SQL injection.

Conclusion

Explore Further

-

Guestbook CSRF

Objectives

Estimated Time: 10 minutes

In this exercise, we will investigate the impact of CSRF on a simple guestbook application. We will learn how to identify and exploit CSRF vulnerabilities.

Requirements

Lab VM

Identifying & Exploiting CSRF

No Hints

The guestbook requires users to log in to post. But due to a CSRF vulnerability, it is possible to trick users into posting without their knowledge as long as they are logged in. Explore the guestbook at http://lab-2.2.sans.labs/login.php (user/pass - sec522/training). Try to exploit the vulnerability and suggest a fix.

Step by Step

1. Using Chrome, browse to http://lab-2.2.sans.labs/login.php.
2. Log in. Username: **"sec522"**, Password: **"training"**.
   - Feel free to explore the guestbook and attempt to add some entries into the guestbook. Remember, at this point, you are logged in to the guestbook.
3. Open a second browser window (Ctrl-N), and visit http://lab-2.2.sans.labs/attack.php. Alternatively, a new browser tab works too.
4. Switch back to the other browser window (loaded with the guestbook page) and refresh the page.

   On the guestbook page, there is an extra guestbook entry that was never entered. What is happening here?

5. Change back to the window with the attack.php opened. Press Ctrl-U to view the source code of the page.
6. Read the page source carefully; pay close attention to the IFrame.

   This self-signing guestbook phenomenon is caused by CSRF (Cross-Site Request Forgery). The following line of HTML code is in the attack.php that was visited earlier.

   ```
   <iframe style="visibility:hidden" src="http://lab-2.2.sans.labs/guestbook.php?
   f=add&name=Evil+Cat&email=kitten@evilexample.com&url=http://
   evilexample.com&comments=I+need+more+wetfood+or+I+will_hack_you!+The+Cat" />
   ```

   It contains a link to the guestbook, asking the guestbook to add an entry. Since this IFrame is set to be invisible, the user does not know of its presence.
7. Change back to the guestbook window and **click on the logout link** to log out of the guestbook.
8. Switch back to the attack window. Refresh the page on **http://lab-2.2.sans.labs/attack.php**. This is to attempt the CSRF attack once more.
9. Switch back once again to the other window with **http://lab-2.2.sans.labs/guestbook.php** and log in with "sec522" and "training" as username/password.

How come no extra entry is added this time?

If the session is not active (the user logged out), the attack simply doesn't work.

Conclusion

- CSRF only works while the user is logged into the application with active session.
- Attacker steer the browser to submit request on behalf of the user. The request has the credential of the user attached.

Cross-Site Scripting

Objectives

Estimated Time: 10 minutes

PART 1:

You will learn how to identify XSS and how important it is to check all input fields for XSS. We will also learn how XSS can be abused against a guestbook web application and how XSS may result in CSRF.

PART 2:

Uses "Juice Shop", a pre-built web application with known vulnerabilities. Juice Shop is a modern Node.js based application. The search feature is vulnerable to a DOM-based XSS attack.

Requirements

- Lab VM

Part 1: Guestbook XSS with CSRF

No Hints

Also, for the guestbook at http://lab-2.3.sans.labs/login.php, we experimented with this guestbook before. This time find the XSS vulnerability and exploit it to automatically approve posts that an administrator views. (Logins available - sec522/training and admin/admin)

Step by Step

For this part, we will revisit the guestbook system. The administrator of the site noticed the CSRF attack and decided to manually approve every single guestbook entry.

1. Using Chrome, browse to **http://lab-2.3.sans.labs/login.php**.

2. Log in to the guestbook by using **"sec522"** and **"training"** as username and password.
   a. Attempt to input two guest entries into the guestbook. The new guestbook requires the administrator to approve the entries before they are posted on the page.
   b. Log out of the guestbook by clicking on the **"Logout"** link.
3. Log in with username **"admin"** and password **"admin"**. This allows you to log in as an administrator.

   Administrative functions are now available. This includes the ability to delete comments and to approve messages.
   a. There should be two comments awaiting approval. Please click delete on one of the comments. That comment should then be deleted.
   b. There should be one comment left; click the **"Approve All Posts"** button to approve the comment to be displayed to the public.
4. Log out of the guestbook by clicking on the **"Logout"** link.

The CSRF attack towards other users is obviously no longer possible, as it would likely not get past the administrator, who will review the comments before allowing the public to see them.

Now that we understand how this guestbook system works, let's explore if it is vulnerable to attacks.

1. Log in to the guestbook by using **"sec522"** and **"training"** as username and password.
2. Attempt to add one new comment. Put anything as name, email, and URL. For comments, enter the following string:

```
XSS<script>alert('xss');</script>
```

3. Press the **"Add"** button. The message is awaiting for the administrator to approve as indicated by the message in the response page.
4. Click **"Logout"** then log in with username **"admin"** and password **"admin"**. A pop-up box should appear with **"xss"** in it. The input from the user became part of the page; the script input becomes rendered in the browser as script. This is a classic case of stored XSS vulnerability.

5. *(Optional)* Feel free to view the source of the page to look for the XSS alert box and see why it became part of the scripting content in the page.
6. Delete the XSS comment in the **"waiting for approval"** section, as this can be annoying since the pop-up box shows up in every page load. Click the **"Delete Message"** link in the XSS comment.
7. Log out of the guestbook by clicking on the **"Logout"** link.

We have now proved the comment field to be vulnerable to XSS attack. Let's see how we can leverage this vulnerability.

We know there is an XSS vulnerability in the comment field. Anyone looking at that comment will have their browser hijacked to run scripts. Hence, these browsers can be controlled. That is very useful: what if we can control the administrator's browser so that the comments could automatically be approved as soon as an administrator looks at them? Let's try this concept.

1. Log in to the guestbook by using **"sec522"** and **"training"** as username and password.
   a. Attempt to add one new comment. Put anything as name, email, and URL. For comments, put the following string:

      ```
      You will approve this<img src="guestbook.php?approve=Y">.
      ```

      approve=Y is to trigger the **"Approve all Posts"** button we saw in the admin function.

   b. Click on **"Add"**.

      No action is triggered at this point. The CSRF string is waiting on the server to be retrieved by the administrator (as the admin logs in); the delivery mechanism from the attacker (sec522) to the administrator is XSS.

   c. Log out of the guestbook by clicking on the **"Logout"** link.

> **Info**
>
> At this point, the sec522 user had submitted an attack into the system awaiting for the administrator to stumble upon it.

2. Log in with **admin / admin** as the credential. This allows you to log in as an administrator.

We see that the comment is waiting for approval. It even contains a broken image icon in the comment.

a. Without doing any delete or approval, click **"Logout"**. At this point, the administrator already unwillingly clicked on the **Approve All Posts** link to approve the comment. The browser did the approval on behalf of the administrator; the instruction came from the user's comment.

3. Log in to the guestbook by using **"sec522"** and **"training"** as username and password.

   The comment is already shown on the screen. The administrator never intentionally approved the comments. XSS and CSRF approved the comment.

Let's go over the whole attack scenario one more time. There are a couple of vulnerabilities at play in this exercise.

User sec522 logs in to post a comment on the guest book. With the text of the comment, the user sec522 also puts in an extra <IMG> tag, which links to the approval function and looks like an image that the browser should automatically load. Once this comment has been submitted, sec522 simply sits and waits till the administrator logs in.

The administrator logs in and, upon logging in, the comments for approval automatically show up on the screen. The <IMG> tag appears to be an image to the administrator's browser, the < and the > tell the browser that these are HTML codes, and the browser happily obeys these instructions. The browser visits the link in the <IMG>, which leads to approving all of the comments.

Then when sec522 user logs in again, the comment that sec522 user posted originally is showing up as approved.

To demonstrate a fix for the XSS vulnerability, we have prepared a \"fixed\" version of the guestbook: "guestbook-xss.php". This file is essentially the same as the guestbook.php we have seen, except the XSS in the comment field is fixed.

The htmlentities function in PHP is to used to encode characters such as < >, /, &, \', or \" to HTML entities. For example, "\< becomes "<" and "\>" becomes ">" . When "\< gets sent to the browser, the browser will interpret it as HTML. But when "<" gets sent to the browser, the browser will display the "\< symbol.

One line of example code is below:

```
print htmlentities($r[3]);
```

Before $r[3] is displayed on-screen, it is encoded.

Feel free to explore the fixed code at **http://lab-2.3.sans.labs/guestbook-xss.phps** (line 117).

1. Browse to http://lab-2.3.sans.labs/guestbook-xss.php.
2. Look at the comments and explore why there is no longer a broken image sign but the entire XSS and CSRF string are simply displayed on the screen. Viewing the source of this page gives insight as to why this is no longer treated as an <IMG> tag but simply text on-screen.

Part 2: DOM-Based XSS in Juice Shop

1. Juice Shop is implemented in Docker. To start it, type at the command line.

   ```
   juiceshop start
   ```

2. Open the browser and visit http://juiceshop:3000. Juice Shop can take a bit of time to load.
3. Explore the search field and attempt to identify the cross-site scripting vulnerability. Try test strings like:

   ```
   ABC<> " '
   ```

   ```
   <script>
   ```

   ```
   <script>alert('text')</script>
   ```

   What works or doesn't work?
4. Try this exploit:

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


5. Open the Chrome debugger. (Click on the three vertical dots in the top right. Then select "Developer Tools" from "More tools".)
6. Select the "Network" tab.

> **Do you see any requests when you submit the search? Why not?**
>
> The exploit in Step 4 works. This is DOM-based XSS that does not require data to be transferred to the server (Step 6: you will not see a network connection).

Conclusion

Explore Further

-

Unicode and File Upload

Objectives

Estimated Time: 10 minutes

PART 1:

In this exercise, you will experience some of the visual problems associated with international domain names.

PART 2:

In this exercise, we will explore the XSS that can be induced with injecting unexpected character sets input.

PART 3:

In this exercise, we will evaluate the threats of handling file upload from users.

Requirements

- Lab VM

Part 1: International Domain Names

Step by Step

1. Browse to http://lab-2.4.sans.labs/unicode.html.
2. The task is to determine which of the domain names listed on this page do not contain any international characters. Can you visually determine that?
3. To test your answer, open the Unicode Converter in a new browser tab.

4. Copy/paste each of the domain names into the converter (CTRL-C: copy, CTRL-V: paste).
5. Paste the domain name into the green "Mixed Input" field. The output of the "Unicode U+hex" field usually gives away any non-ASCII characters.

   The "IDN Converter", linked from the question page, can also help identify Unicode domain names.

Solutions

You can find the solutions at this location http://lab-2.4.sans.labs/unicode-solutions.htm

Part 2: XSS via Inappropriate Character Set Conversion

1. Go to http://lab-2.4.sans.labs/unicode.txt. This file is UTF-16 encoded and displays some "Chinese" characters. The characters themselves don't make any sense at this point.
2. Go to http://lab-2.4.sans.labs/unicodexss.php. This page will load the file we looked at above and display it using a UTF-8 character set. In this context, you will see an "XSS" popup.

3. Let's check the source code for unicodexss.php by going to http://lab-2.4.sans.labs/unicodexss.phps. You will see that the file calls file_get_contents (line 9), a simple function that will read a file and send its contents to the browser.
4. Open a terminal and run:

```
cat /etc/docker/compose/2.4/web/html/unicode.txt
```

This will display the "script" in the OS context, it's not showing the Chinese characters we saw earlier.

Part 3: File Upload

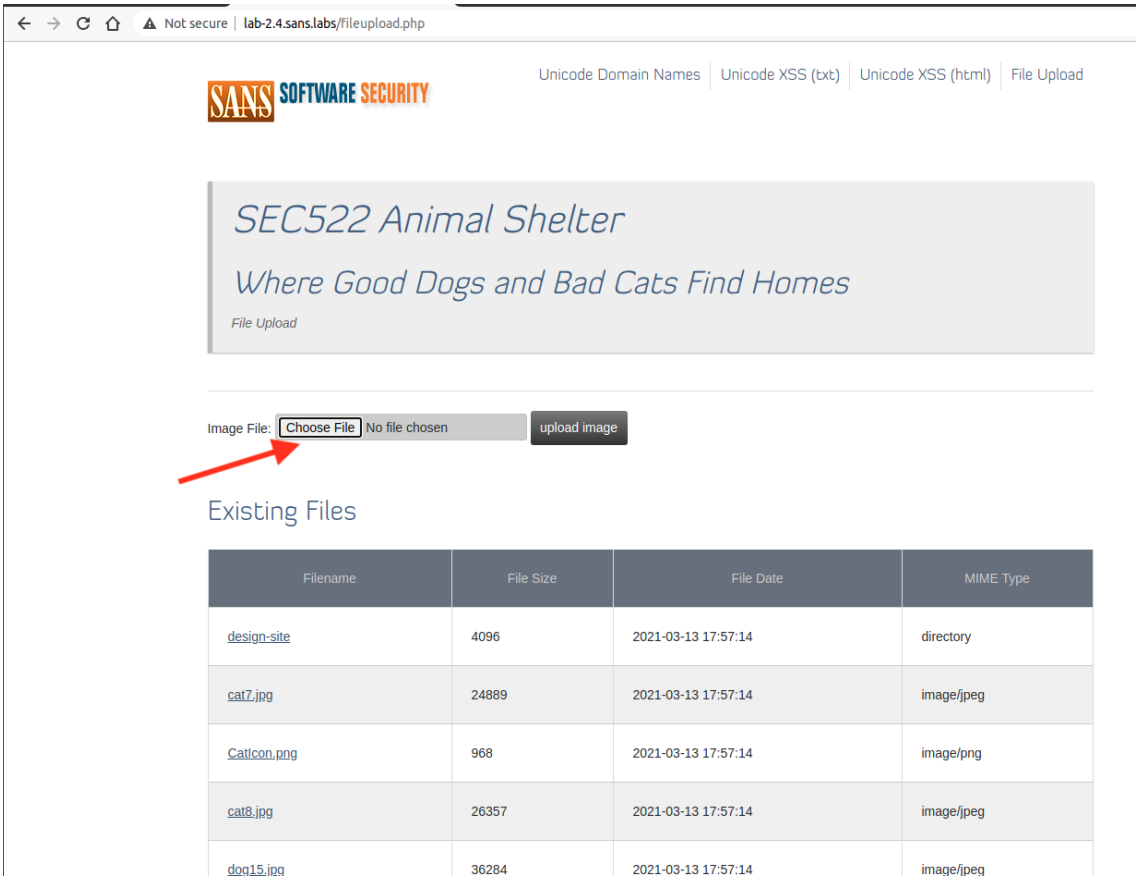
No Hints

The upload function of the TWiki system is our test subject in this exercise. Browse to **http://lab-2.4.sans.labs/fileupload.php** and experiment with the upload function, then browse through the "Documents" directory on the desktop and see how you can leverage the upload feature to compromise the upload system.

Step by Step

1. With Chrome, browse to http://lab-2.4.sans.labs/fileupload.php

   In this system, users can upload an image or document.
2. Click the button labeled "Choose File", then navigate to **Documents** (left tab) then select the **hackercat.php.1** file. Click on **open**.



3. You should now see the local file selected. Click on **"Upload Image"** at the to the right side of the file selection.

   The file is then uploaded to the server and is subsequently displayed on the main page as part of the attachment to the page (new files at the top).

4. Click the link for "hackercat.php.1".

> **What is being displayed on-screen?**
>
> At the top of the page, there are some corrupted characters and then followed by the PHP environment details. To understand the content displayed on the screen and what is going on behind the scene, we need to review the file we uploaded to the server.

5. Use Text Editor or Visual Studio Code (see icon on the left favorite bar) to open the **hackercat.php.1** file. It is located within the **Documents** directory.

The editor may complain that there are invalid characters in the file. You can click on **"Edit Anyway"** to ignore the warning. Within the first three lines of the file, you should find "<?php phpinfo() ?>"; it is a single PHP programming command. phpinfo(); is a call to a function that displays the status of the current PHP environment to the user. What we have seen in the browser when we click on the uploaded file is actually the result of this command executing. Although the rest of the content in the file are binary characters forming a gif graphic file, the PHP interpreter picks out the content it understands and attempts to execute the command.

We have successfully uploaded the file and executed the commands within the file. Execution of scripting command is a serious compromise of server security.

In most modern Linux distributions, PHP is often bundled and enabled by default. Any file with a ".php" extension is executed by PHP interpreter.

In the /etc/docker/compose/2.4/vhost.conf configuration file, which controls both the Apache and the PHP instance, there is the following configuration line:

```
<FilesMatch ".+.ph(ar|p|tml)">
   SetHandler application/x-httpd-php
</FilesMatch>
```

This tells the Apache server to pass any file that **contains** .php to the PHP interpreter for execution. This means not just .php but also .php.1 or .php.bak as well. This Apache configuration is the default in mutiple Linux distributions.

The upload code in TWiki had anticipated someone might upload PHP code, so numberous filters are implemented. You can inspect them directly at **http://lab-2.4.sans.labs/fileupload.phps** and the code are below as well

```
# retrieve file parameters

$sTmpFile=$_FILES['file']['tmp_name'];
$sFileName=$_FILES['file']['name'];
$nFileSize=$_FILES['file']['size'];
$aCheck=getimagesize($sTmpFile);

# check if it is an image

if ( $aCheck !== false ) {
   $sMessage .= 'File is an image.<br/>';
} else {
   $sError .= 'File is not an image.<br/>';
}
```

```
# check file size

if ( file_exists("images/".$sFileName) ) {
    $sError .= 'File already exists.<br/>';
}

# check if the extension is .php

if ( preg_match('/\.php$/',$sFileName) ) {
   $sError .= 'Filename has the wrong extension.<br/>';
}

# check mime type

if ( mime_content_type($sTmpFile)==='image/gif' ||
     mime_content_type($sTmpFile)==='image/png' ||
     mime_content_type($sTmpFile)==='image/jpg' ||
     mime_content_type($sTmpFile)==='image/jpeg' ) {
   $sMessage.='File type: '.mime_content_type($sTmpFile);
} else {
   $sError.='Wrong File type: '.mime_content_type($sTmpFile);
}
```

This only blocks extensions like ".php" but not .php.1. Double extensions like ".php.1" can be used to bypass this filter.

How can this type of vulnerability be avoided? There are multiple mitigation avenues.

As soon as the server receives the file, it can change the file extension to make sure it cannot be executed. For example, changing everything with a ".txt" extension will make any script interpreted as text format.

If the file does not need to be downloaded by the user in normal operation, do not store the file in web directories. Hide the file on the server in a directory outside of the web-accessible files. If the files must be downloaded by other users, ensure there are no execution rights on the directory or the file itself. Various platforms have different ways to do this. Essentially, the file or the whole directory storing uploaded content should be denied any execution of code.

In the case of double extension files, an Apache config may be used to disallow any use of double extension ".php.xxx" files.

```
<FilesMatch ".php.$">
   Order deny, allow
   Allow from all
</FilesMatch>
```

Conclusion

Explore Further

-

Authentication Exercise

Objectives

Estimated Time: 22 minutes

PART 1:

In this exercise, we will experiment with different examples of weak authentication and demonstrate how to break them.

PART 2:

In this exercise, we will experiment with different examples of weak authentication and demonstrate how to break them.

PART 3:

In this exercise, you will walk through the use of U2F (Universal Two-Factor) tokens for two factor authentication. We use a software simulation of a token (softu2f). After registering and testing the token, we will experiment with a method to bypass the use of the token.

Requirements

- Lab VM

Part 0: *(Optional)* Simple User-Agent-Based Authentication

No Hints

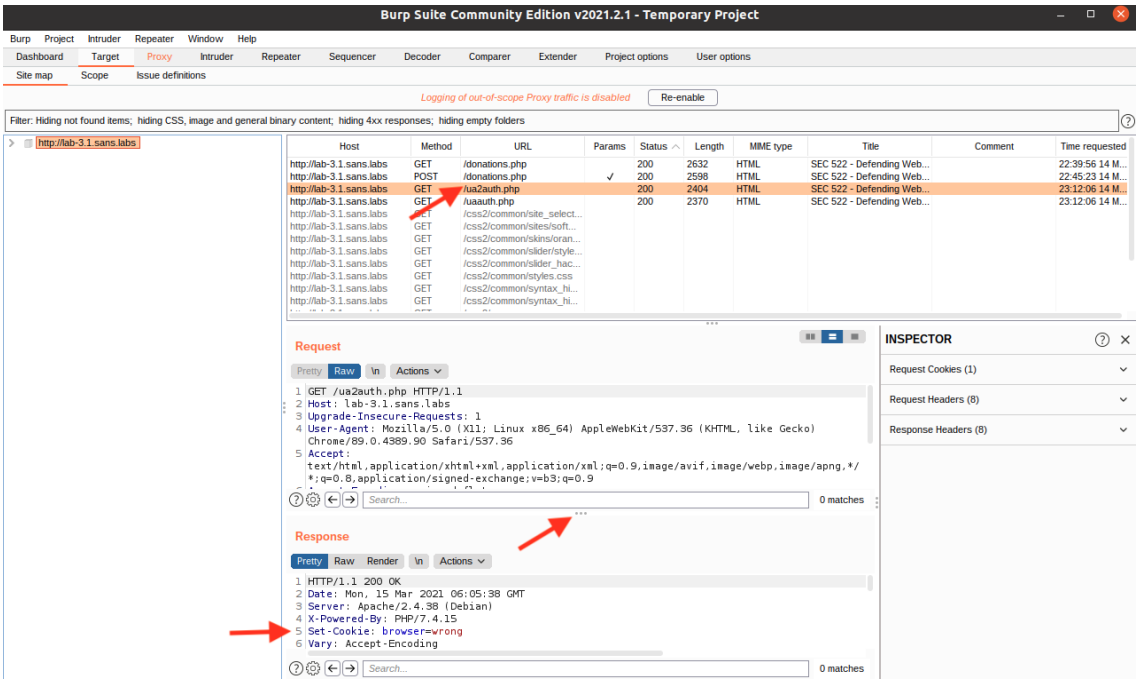
Step-by-Step Instructions

1. Use Chrome to proceed to **http://lab-3.1.sans.labs/uaauth.php**. The page will indicate that you are using the wrong browser.
2. Start Burp and configure Chrome to use Burp via the Proxy Switcher plugin.

3. Check to make sure interception is on in Burp ("Proxy" ->"Intercept" -> "Intercept is on").
4. Reload the page in the browser.
5. The request will now be held by Burp. Use Burp to modify the request and change the User-Agent to "CatBrowser".
6. Click "Forward" in Burp. The modified request will now be sent, and Chrome should display the page with a thank you statement for using the SANS browser.

Part 2: More Complex, But Still Weak, Authentication

1. Ensure that the Chrome is set to sending all browser traffic through Burp - the ProxySwitcher in Chome should be "red" in color.
2. Open http://lab-3.1.sans.labs/ua2auth.php in Chrome.
3. Switch over to Burp and allow all traffic to pass without modification. Keep clicking on "**Forward**" in Burp's Proxy Intercept. This page looks just like the page we examined in the exercise above but there seems to be some subtle difference.
4. To inspect this previous request, in Burp, we change over to the "**Target**" tab.
5. Look for the most request to "ua2auth.php" and click on it. At the bottom pane of the Burp window is the response details. Due to the window sizing, you might have to adjust the pane to let the response be visible. There is the "three dots symbol" that allow you to adjust the pane sizes. In the response section, you will see the browser cookie being set by the server. We will need to get rid of this cookie in order to get to the website.



6. Switch back to Chrome and reload the page.
7. The request will now be held by Burp. Use Burp to modify the request and change the User-Agent to "DOGBrowser" and also to delete the cookie line. Make sure there are no blank lines in between the HTTP headers.
8. Click "Forward" for all the following requests. You will then see the protected page.

What trick did the developer play to harden the page? Was it sufficient?

The developer coded the authnetication such that when the "wrong" browser connect to the webpage, the application provides a cookie to the browser so that the application can reference the use of "wrong" browser in subsequent requests.

Part 3: Brute Forcing

This is an additional challenge exercise on authentication building on the knowledge you acquired so far. This exercise is to simulate a registration system where there is personal information being stored.

1. Configure Chrome to bypass the Burp proxy via the Proxy Switcher plugin. The icon should be black in color.

2. Browse to http://lab-3.1.sans.labs/donations.php

   The page has a lookup system where the user can review their own respective donation information. The objective is to acquire as much information from this system as possible.

   Use the following hints to help you:

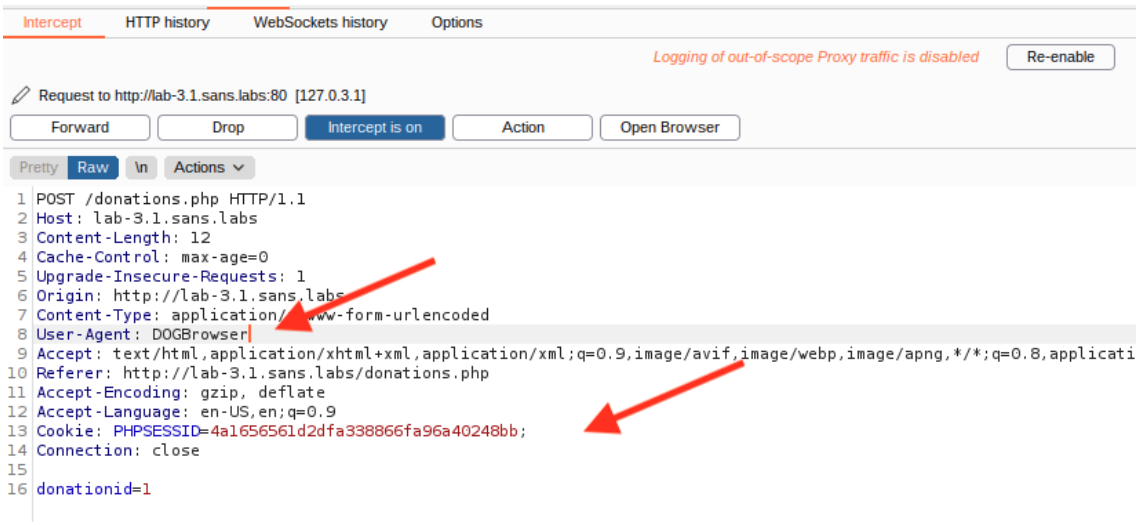
   > **Hint #1**
   >
   > What's the likely user-agent for "DOGBrowser"?

   > **Hint #1**
   >
   > What's the purpose of the cookie "browser"?

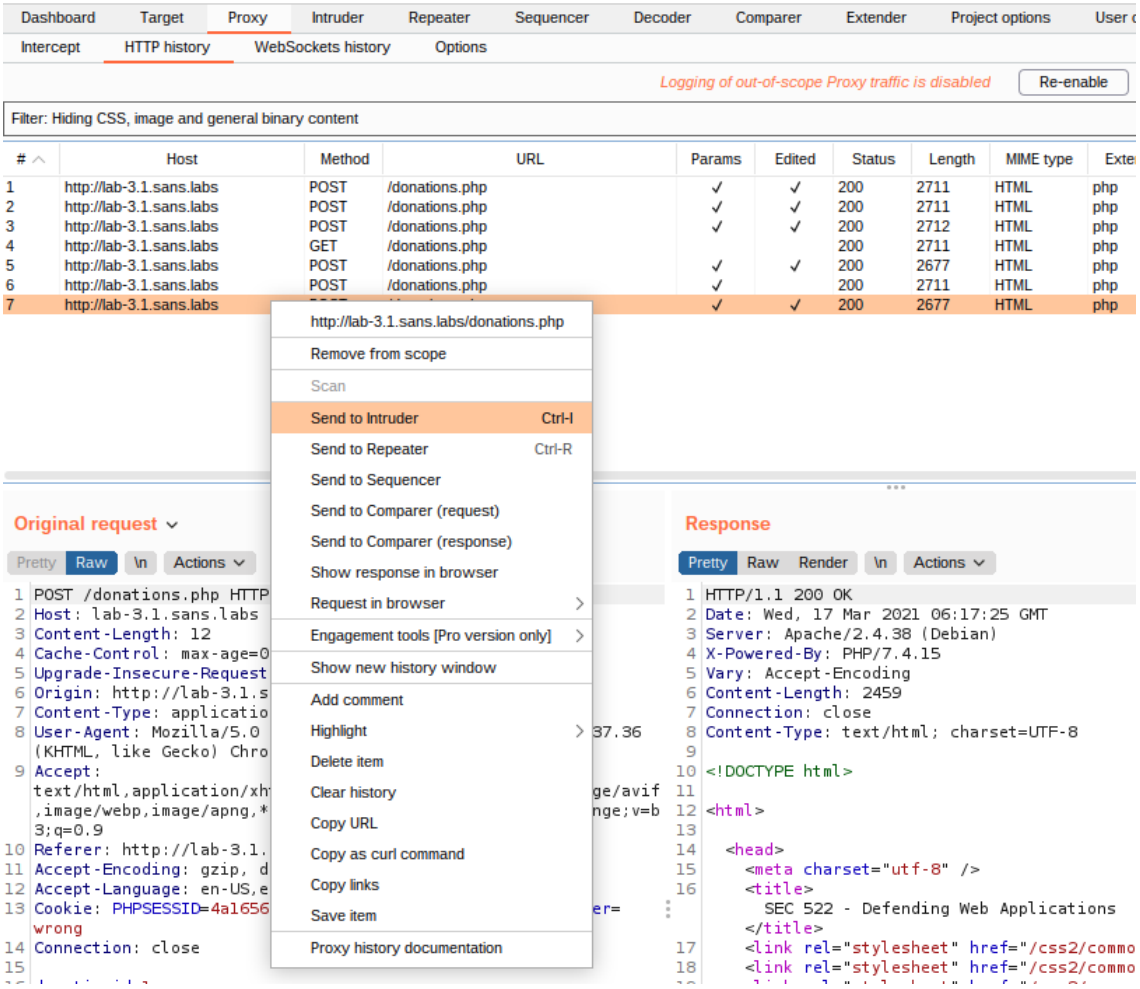
   > **Hint #2**
   >
   > Can you brute force the donation number? Scripting may help.

   > **Hint #3**
   >
   > You can use the "Intruder" feature in Burp to assist with brute forcing the solution.

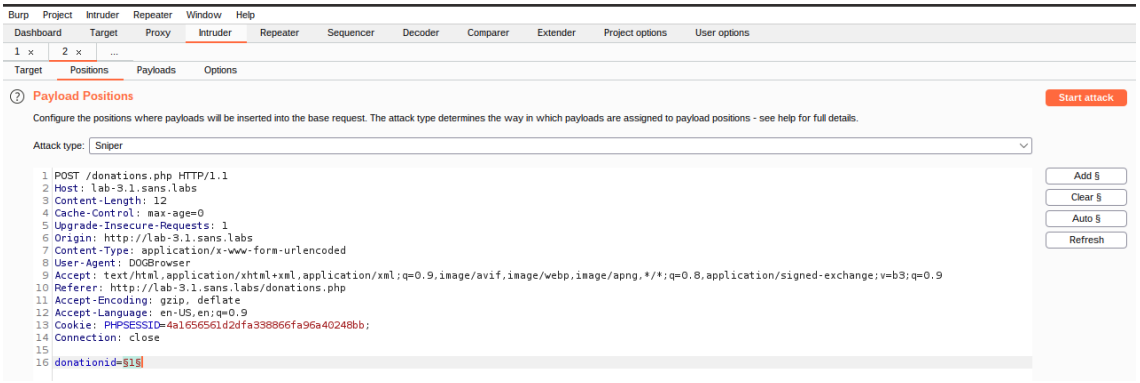
3. The page asks for a donation ID. Guess one (maybe 1) and click "Send". The server does not seem to respond to this.
4. Let's inspect it in details. Configure Chrome to use Burp proxy via the Proxy Switcher and then reload the page.
5. In the Burp intercept window, edit the request so that the User-Agent in the request is "DOGBrowser". Also eliminate the **browser=wrong** cookie by deleting it.



6. The system should indicatet that the ID does not exist. You still need to guess a donation ID. There are different ways to do this. We will use Burp in this example.
7. Select the Proxy->HTTP history tab in Burp and right-click on the last request for donations.php

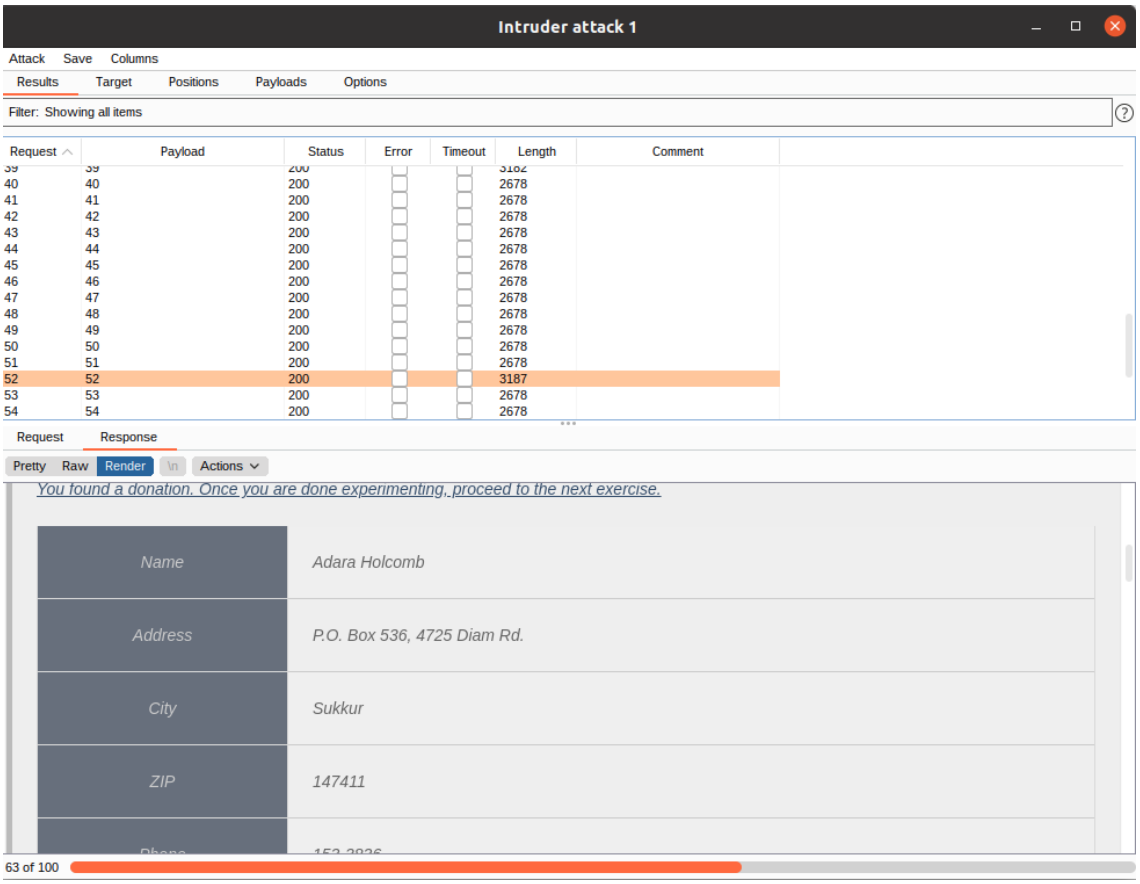


8. Send Request to Intruder
9. Select "Send to Intruder".
10. Click on the "Intruder" tab (it should now be highlighted in orange).
11. Select the "Positions" tab.
12. Click the "Clear §" button. This will clear the auto-selected payload positions.
13. Highlight the digit after **donation=**, and click the "Add §" button. This will mark the digits for brute forcing. Also, verify the User-Agent ("DOGBrowser") and make sure the "Browser=wrong" cookie is removed.



Marking Position for Intruder

14. Select the "Payloads" tab.
15. Select "Numbers" from the Payload type dropdown (you may have to scroll down).
16. Select "1" as "From", 100 as "To" , "1" as "Step" and 0 as maximum fraction digits (do not select a larger range, as the Burp Community Edition does rate-limit these requests).
17. Click "Start attack" and acknowledge the restrictions of the community edition.
18. You do not have to wait for the attack to finish. Look for requests with unusual values in the "Length" column and click on them. The larger responses are the ones we are interested in. Select the "Response" and "Render" tab at the bottom to review the response. For successful requests, you should see personal information displayed as part of the response. Request 6 and 17 (and many others) will show positive results. You may need to scroll up in the result window to see the early responses.



Intruder Result

Part 4: U2F And Bypassing 2nd Factor

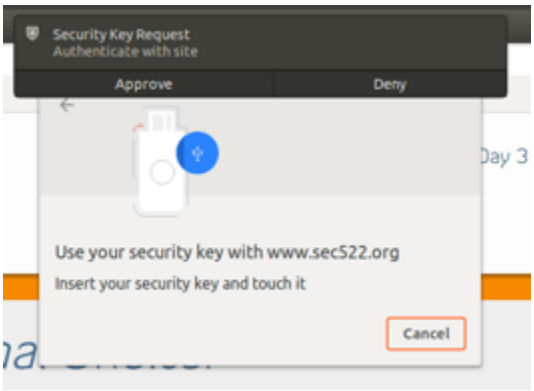
No Hints

Start at https://lab-3.1.sans.labs/login.php . Click "Reset U2F Data". Next log in and follow the instructions to setup the token. Once you setup and verified that the token is working: Try to reach the "myaccount.php" without using the token (just use the username and password).

Step by Step

1. Open https://lab-3.1.sans.labs/login.php in Google Chrome.
2. Click on "Reset U2F Data" (lower left corner of the page).

3. Login using the username "sec522" and the password "training". You will see a popup at the top of the browser asking you to "Approve" the use of the virtual U2F token.



U2F Dialog

4. Click "approve". You will now be redirected to the "My Account" page at https://lab-3.1.sans.labs/myaccount.php
5. Click on "Log Out". You will be redirected to the login page.

> **Information**
>
> At this point, we have registered a token with the server. There is a key pair generated and the server is storing the key information so that it can be used in the futue to authenticate the user when the user login.
>
> You may have noticed that you have to click on the token approval earlier. This is a design of Fido2 compatible tokens (whether it's the Android OS, iOS, keyfobs) they all require explicit user approval to avoid accidental and unintentional approvals.

6. Login again using username "sec522" and password "training".
7. Again, you will be asked to approve the use of the U2F token
8. You are now reaching the "My Account" page.

   So far, you learned how to use a U2F token. For a physical token, you would have to pres a button instead of clicking "Approve". Next, we will learn how to bypass the token request.
9. Click "Log Out" again.
10. Verify that you are actually logged out: Try to open https://lab-3.1.sans.labs/myaccount.php in the browser. You will be redirected to the login page.

11. Login using the username "sec522" and the password "training". You will be prompted for the U2F token again.
12. Decline the U2F token request, deny the U2F token and cancel the browser's authentication prompt.
13. Open https://lab-3.1.sans.labs/myaccount.php in Chrome. You will be able to access the page. You just accessed the "My Account" page without using the U2F token, bypassing the two-factor authentication requirement.

    The application logged you in after you entered the username and password. This often happens if the U2F authentication is added later. A better solution would not associate the users user id with the session until the U2F process is complete, or set a flag with the session that the U2F authentication has not yet completed.

> **If all else fails**
>
> This exercise required the use of a Linux U2F token software for authentication, this is used in lieu of a real Fido2 token (Fido2 is backward compatible with U2F). If resetting of the token software is need, please run the following command in the terminal window.
>
> ```
> systemctl --user restart softu2f
> ```

Conclusion

Explore Further

-

Session Fixation and Session Breaking

Objectives

Estimated Time: 12 minutes

PART 1:

In the first part, you will learn how to identify and fix session fixation vulnerabilities.

PART 2:

In second part, you will experience weak session IDs and learn how an attacker may exploit them.

Requirements

- Lab VM

Part 1: Session Fixation

No Hints

EDIT

Load http://lab-3.2.sans.labs/login.php in Chrome. The first two forms are susceptible to session fixation. Why? (In particular, the second one is not that obvious.) The third form is not susceptible.

Step by Step

This exercise will ask you to run Firefox in addition to Chrome. To start Firefox when asked to, click on the Firefox icon at the left side of the screen.

IMPORTANT: Do not "Refresh Firefox" or update Firefox. Click "Cancel" if a dialog box appears offering to refresh Firefox.

1. Load http://www.sec522.org/ex2/2/index.php in the browser. You will be redirected to a URL that includes a session ID as a get parameter. The session ID is random. It should look like: PHPSESID=gn5blighc8a)o5ks45k3pu81l
2. Copy the session ID and paste it into an empty file (you can use the text editor on the left). We will call this session ID the "original session"
3. Log in using the first form, "Step 1", using the username "sec522" and the password "training". Did the session ID change? The form is vulnerable if it did not change.
4. To simulate the attacker, open a second browser. (Firefox is installed in the virtual machine. Search for it by clicking on "Activities".) Copy and paste the URL and check if you are indeed logged in.
5. Close Firefox and close the logged in tab.
6. Click on "Clear Session Data" button to obtain a new session ID.
7. Again, copy paste the session ID from the browser URL.

   Log in using the "Step 2" form.

   Did the session ID change?

8. Open Firefox again and copy the original session ID to the URL bar.

   Does it still work?

9. Now try the third and final form ("Step 3"). Log in again and compare the session IDs.
10. Again, copy the original session ID to the URL bar in Firefox.

    Does the page work?

11. Compare the source code:

    http://www.sec522org/ex2/2/fix2.phps
    http://www.sec522org/ex2/2/fix3.phps

There are subtle differences in the location of session_regenerate_id. See lines 14 and 16 of the code.

What are the differences?

Answers

Did the session ID change?

3 & 5: The session ID did not change and can be used in a different browser.

7: The session ID did change.

Does it still work?

8: The old session ID still works, and it is now associated with the user.

Does the page work?

10: The old session ID no longer works and you end up with a redirect loop---not elegant, but "secure".

Does the page work?

11: "fix2" regenerates the session ID after the login was verified. "fix3" regenerates the session ID first, so the user ID is never associated with the old session. Also, the old session is destroyed (the "true" parameter).

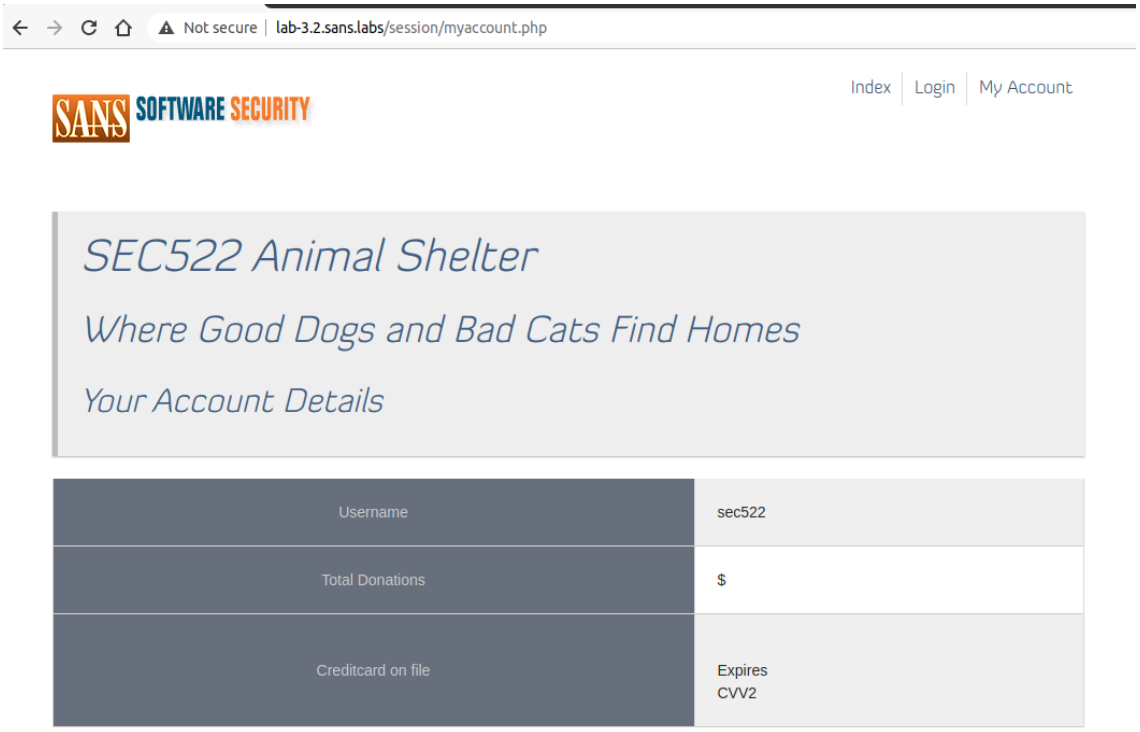
Part 2: Breaking Sessions

No Hints

Go to http://lab-3.2.sans.labs/session/login.php. Log in using the username "sec522" and the password "training". Try to switch to a different user's account after logging in.

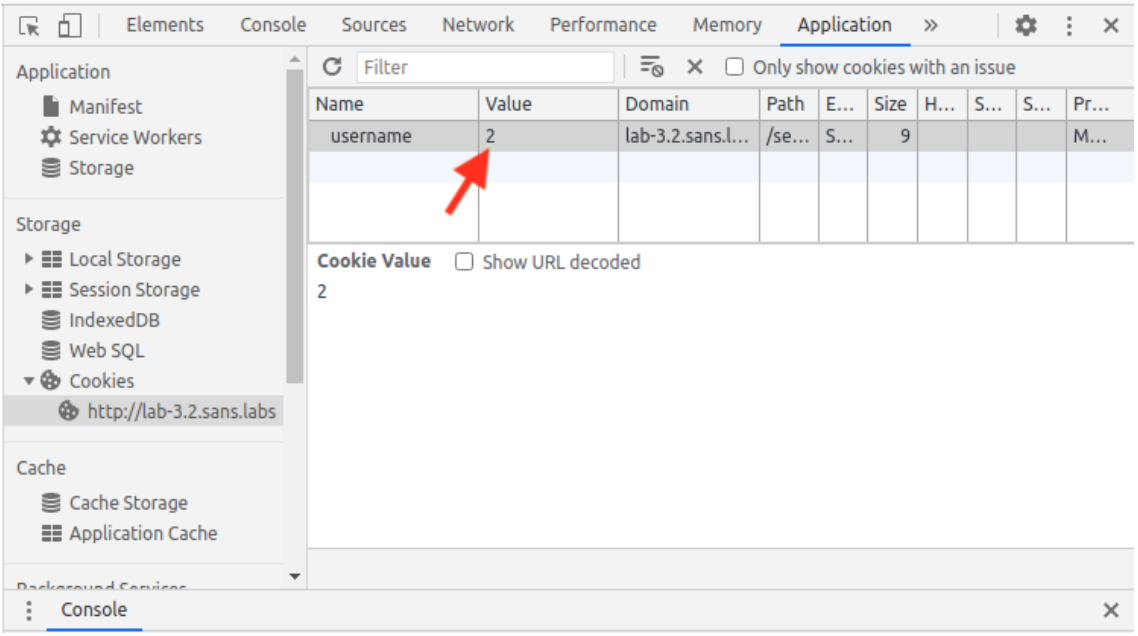
Step by Step

1. In Google Chrome, proceed to **http://lab-3.2.sans.labs/session/login.php** and log in using the username **sec522** and the password **training**.



Account Balance Screen

2. Press SHIFT-CTRL-I to open the developer tools.
3. Select the "Application" tab and select http://lab-3.2.sans.labs under Cookies.
4. Double-click on the cookie value (value is 1) and change it to 2 . Reload the page.



Chrome Developer Tools Cookies

You will now see a different account. Experiment with different values and try to find the account with the most donations.

Which account has the most donations?

tdeerness18@wordpress.com of value 45 has the most donation at $997.74

Conclusion

Explore Further

-

OAuth and Access Control

Objectives

Estimated Time: 15 minutes

PART 1:

We learned that for web applications, it is important to apply access control consistently throughout the entire application. In this exercise, we are examining a flaw that allows regular users to perform administrative functions. To explore this flaw, we first execute these actions as administrator to learn what is possible and then later try to replay these actions as a normal user.

PART 2:

In this exercise, you will walk through an OAuth authentication and later demonstrate how this application does not secure its OAuth tokens successfully

Requirements

- Lab VM

Part 1: Access Control

No Hints

Open http://lab-3.3.sans.labs/index.php in a browser. Feel free to explore on your own at this point. An administrator is able to adjust the status of an animal in the shelter (for example, flip the "Cat Aggressive" status) by clicking on the respective icon in the page describing an individual animal. Try to perform this action as a normal user. Admin username/pass is sec522/training and regular user is hackercat/training

Step by Step

1. Open http://lab-3.3.sans.labs/index.php in Chrome.

2. Click the "Login" button (upper right hand corner). This will direct you to our animal shelter login page.
3. Log in as an administrator (username: sec522, password: training). You will see a list of animals up for adoption.
4. Click on an animal's image (for example, the first cat, "Cinnamon"). This will display a page with details about this particular animal.
5. Bellow the image, you will see three icons. Click on one of the icons and it will change color from green to red (or red to green if it was red).

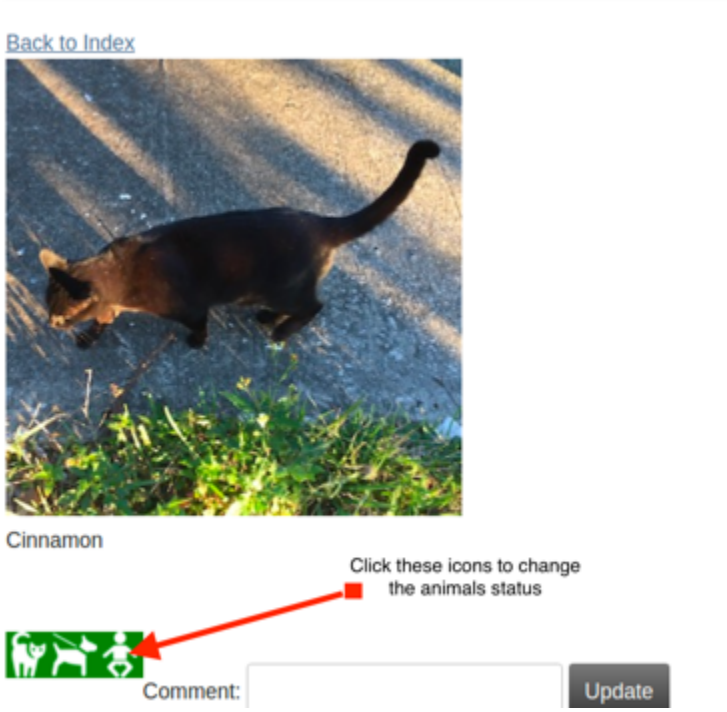
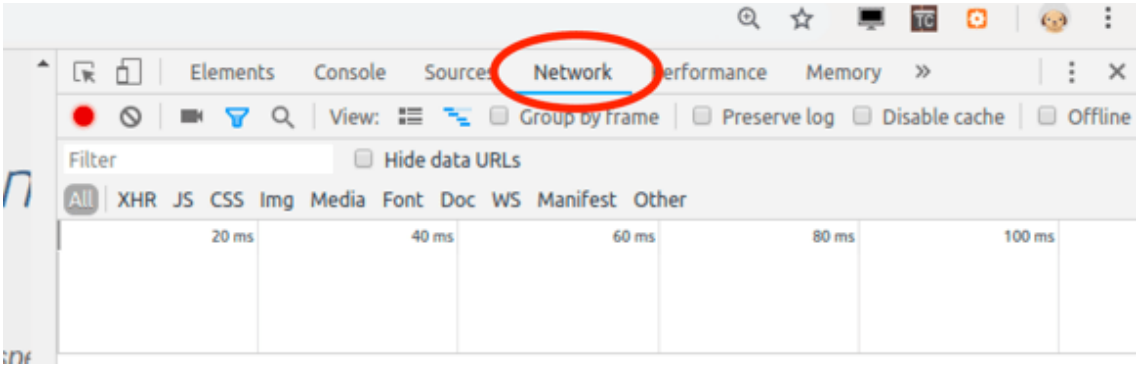


Image of Individual Animal with Icons Indicating Status

6. Click on the "Log Out" link (upper right hand corner).
7. Click on the "Login again" link.
8. Log in as a normal user (username: hackercat, password: training).
9. Click on an animal.
10. Verify that clicking on the animal's "status icons" does nothing.
11. Click on the "Log Out" link (upper right hand corner).
12. Click on the "Login again" link.
13. Log in as an administrator (username: sec522, password: training).

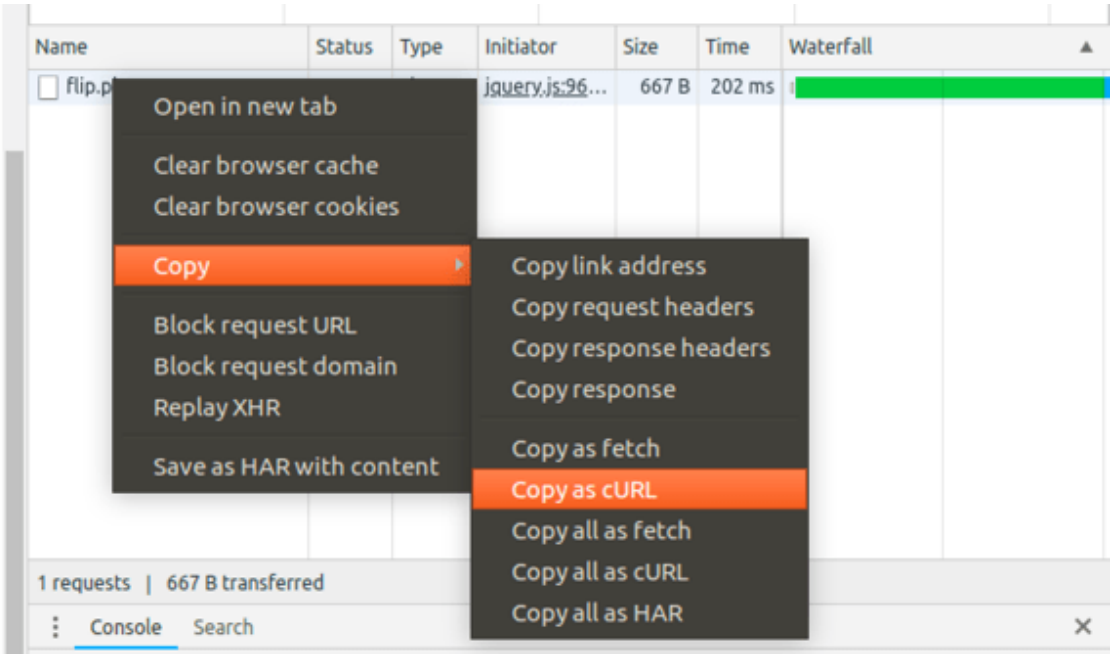
Now we will explore what happens when the administrator clicks on one of the status icons.

14. Click on an animal's picture again to end up on the animal's individual page.
15. Open the Google Chrome developer tools by pressing CTRL-Shift-I.
16. Select the "Network" tab in the developer tools.



Google Chrome Developer Tools With Network Tab Selected

17. Click on the "Dog" icon to change the animal's "Dog Aggressive" status. This will create a new entry in the developer tool window for "flip.php".
18. Right-click on the "flip.php" entry and select "Copy" -> "Copy as cURL".



Copy "flip.php" Request as cURL

19. Open an editor (for example, Visual Studio Code) and paste the request into an empty file.

20. Save the file (recommended name: "test.sh" in the user's home directory, /home/student).
21. Open a terminal.
22. Execute the file:

```
source /home/student/test.sh
```

Your output will look like.

```
student@sec522 ~ $ source /home/student/test.sh
{"catagressive":"0","0":"0","dogagressive":"0","1":"0","kidagressive":"0","2":"0","status":"OK"}
student@sec522 ~ $ _
```

Output of test.sh

> **Note**
>
> The highlighted part in the end **"status":"OK"**. This indicates that the script ran successfully. At this point, you are still logged in as an administrator, and a successful script is what we expect.

23. Back on Chrome, click the "Log Out" link.
24. Run the script again, just like above:

```
source /home/sec522/test.sh
```

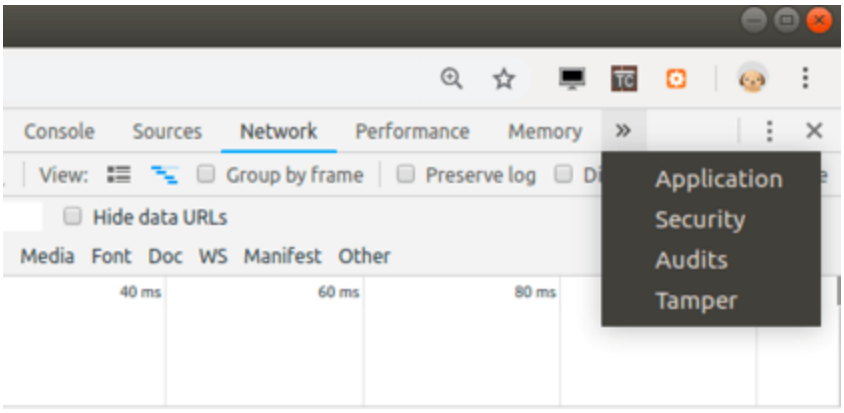
This time, the output will be an error:

```
student@sec522 ~ $ source /home/student/test.sh
{"status":"FAIL"}
```

Terminal Output After Running the Script a Second Time After Logging Out

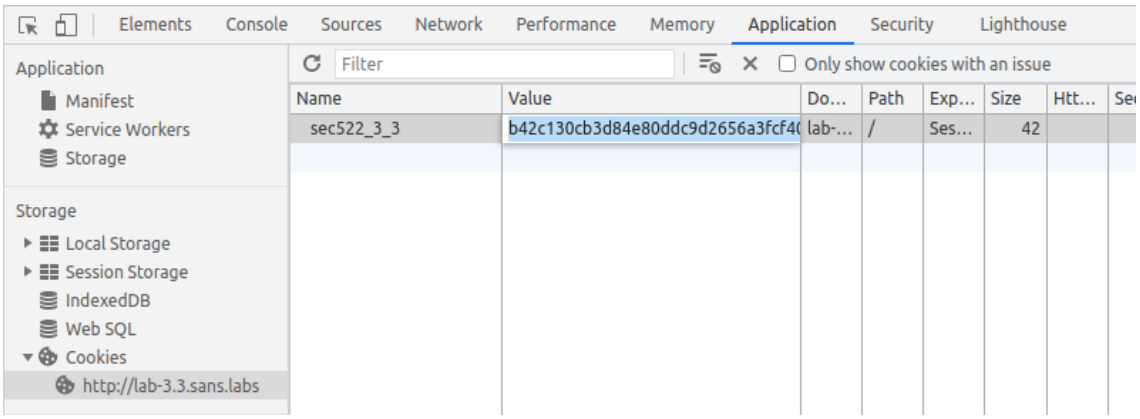
The script failed this time because "sec522" is no longer logged in, and the cookie included in the prior request in test.sh is no longer valid.

25. Log in as a normal user again (username: hackercat, password: training).
26. Open the "Application" tab in Chrome's Developer Tool. Note that, depending on your screen resolution, this option may be available by clicking on the '>>' symbol in the top right area of the developer tools (see picture below).



Opening the Application Tab

27. Next, click on "Cookies" in the left part of the developer menu.
28. Click on "http://lab-3.3.sans.labs" under "Cookies".
29. Double-click on the "Value" to the right of "sec522_3_3". It will turn blue.



Highlighting the Cookie Value

30. Right-click on the highlighted value and select "Copy".
31. Open the "test.sh" script you created earlier in your favorite editor. If you are using Visual Studio Code, all you need to do is open the editor and it should automatically open the last edited file.
32. Find the cookie value. It will be the random string after "sec522_3_3=" and replace it with the cookie value you copied from Chrome. Just highlight the old value and "Paste" to overwrite it. If there is a single quote at the end of the session ID, do not delete it.

```
9   -H 'Accept-Language: en-US,en;q=0.9' \
10  -H 'Cookie: sec522_3_3=b42c130cb3d84e80ddc9d2656a3fcf40' \
11  --data-raw 'flip=dogagressive&animalid=1' \
```

Highlighted Cookie Value

33. Exit the editor.
34. Switch back to the terminal.
35. Run the script again. This time, the request should succeed. The figure below shows the terminal with all three requests: first, the one using the administrator's cookie; next, the failed request after the administrator logged out; and finally, the request we just sent with the regular user's cookie, which succeeded as well.

The final request should not have succeeded. This proves that a normal user can execute an administrative request as long as the user knows what request to send.

You may want to now return to the browser to check if the information was changed. You can also make additional edits to the script. Close to the end of the request you should see an "animalid" parameter that can be used to pick different animals. Experiment with it.

Please suggest ways to fix the issues in the applications. You can review the source code of the script at http://lab-3.3.sans.labs/api/flip.phps .

Part 2: OAuth

No Hints

The exercise starts at http://lab-3.3.sans.labs/calendar/index.php. The basic steps are explained as part of the website.

Step by Step

1. The exercise starts at http://lab-3.3.sans.labs/calendar/index.php. The page implements a simple calendar display. But to use it, you need to allow the calendar page to retrieve your calendar from the animal shelter site. Click on the link on the page to be sent to the animal shelter.
2. At the animal shelter, you will need to log in (if you are not already logged in). The username is "sec522", the password is "training".
3. Next, a simple page will ask you for permission to use the calendar. Inspect the URL. It will contain the client_id, which identifies the application requesting access. The "state" variable is used to identify the response returned to the client. Allow access by clicking "yes".

4. You will now be redirected back to the volunteer calendar app and your schedule should appear.
5. Next, click on the "problem" link to learn more about the issue causing authentication tokens to leak.
6. The link already SQL injected the target page. You will now see the vulnerable statement. The "animalname" parameter is injectable, and we can use a union statement to retrieve additional data. The union statement needs to retrieve the same number of columns as the original statement (7). The second column ("animalname") is easily displayed and we will use it for our exploit. Enter as a search:

   Important: there is a space after the two dashes '--'.

   ```
   ' UNION SELECT 1,access_token,2,3,4,5,6 FROM oauth.oauth_access_tokens LIMIT 1 --
   ```

   The token for OAuth authentication (Refresh token) will be displayed below a broken picture on the result page.

Conclusion

- While OAuth can offer secure and reliable authorization between two web sites, the token handling needs to be performed with extreme care. Compromises of tokens can be through other attacks such as SQL injection. Once an attacker acquire the refresh token, the attacker can use the token to get new Access Tokens for further access.
- Access control flaws can occur to the most modern applications. Keeping the access controls consistent is the key to successfully defend your application.

Explore Further

- Using the acquired refresh token to acquire new access tokens
- Are there other refresh tokens inside the database?

Inspecting SSL Traffic with Wireshark

Objectives

Estimated Time: 15 minutes

PART 1:

In this exercise, the objective is to learn about how SSL actually works. There have been a lot of misconceptions about how SSL works and what features it provides. Everyone seems to understand that SSL makes things secure, but exactly how is this done? We will try to demystify SSL in this exercise.

Requirements

Lab VM

Inspecting SSL Traffic

No Hints

Step by Step

We will use Wireshark to inspect a packet capture. Wireshark is a full-featured packet analysis tool. It includes the ability to decrypt SSL traffic as long as we provide the secret key or the session key negotiated between server and browser. In our example, we will use two files:

- ~/Documents/SSL/ssl.pcap -- a packet capture of https traffic
- ~/Documents/SSL/server.key -- the secret key used by the server in the above sample captures.
- Click on the **Wireshark** icon on the left of the virtual machine desktop.

Wireshark Icon

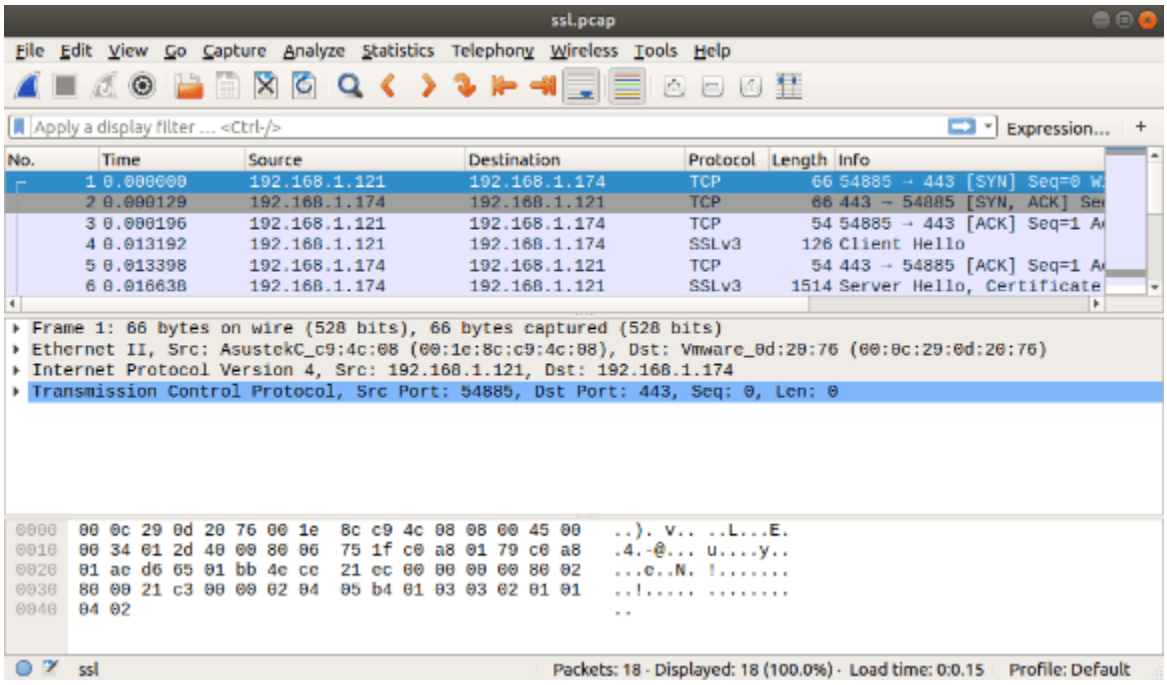
- After Wireshark starts up, click on the top menu **"File"** and then **"Open"**. Select the following file: **/home/student/Documents/SSL/ssl.pcap**

Wireshark will now display its familiar default "3-pane" layout. At the top,

you will see a list of captured packets, one per line.

The second pane will "explain" the currently highlighted packet.

At the bottom, the third pane will display the entire packet in hexadecimal "as captured".



Wireshark Default Display

The first three packets (Nos. 1--3) hold the TCP three-way handshake. They establish the TCP connection and do not indicate that we are going to use SSL.

Packet 4, sent by the client (192.168.1.121) to the server (192.168.1.174), represents an "SSL Client Hello". With this packet, the client indicates that it would like to talk SSL to the server. It does provide some basic parameters that are used to negotiate the SSL connection.

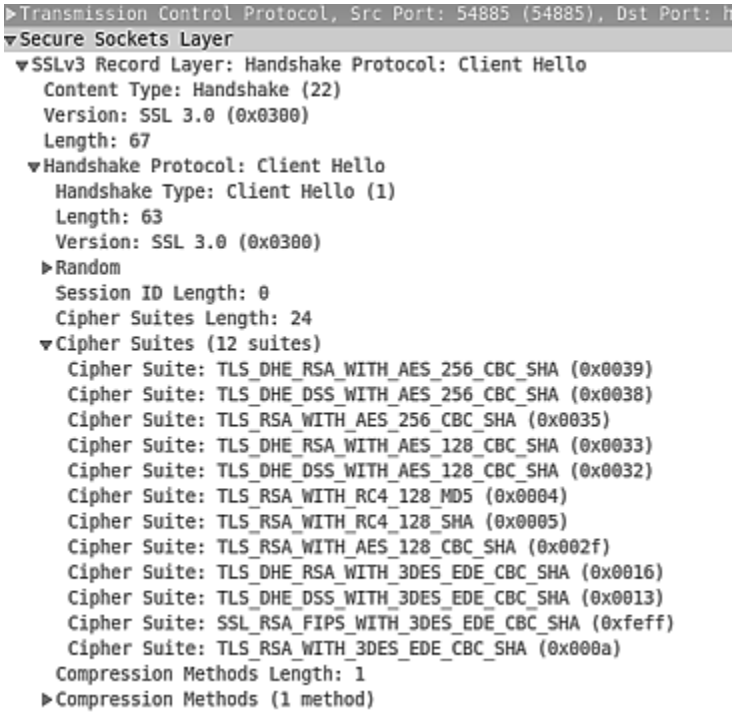
By inspecting the middle pane, try to identify the following parameters:

What SSL version is the client using?

SSL 3.0

How many cipher suites does the client support?

12



Client Hello Details

In packet #5, the server acknowledges receipt of the client hello.

Packet 6 contains the server's response, a "Server Hello". This packet also includes the server's certificate. Just as for packet 4, identify the following parameters:

Which SSL version is the server accepting?

SSL 3.0

Which cipher suite did the server select?

TLS_RSA_WITH_RC4_128_MD5

Who issued the certificate?

SANS CA

What is the hostname (common name) of the server according to the certificate?

sanshost

```
▾ Secure Sockets Layer
  ▾ SSLv3 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 74
    ▾ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 70
        Version: SSL 3.0 (0x0300)
      ▸ Random: 483a29e965197c85ae89cd035d1de03fb996b3a6e1b23547...
        Session ID Length: 32
        Session ID: 65bace29d987cac9844035ebf2f7e389e4fe8f465ab0acff...
        Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
        Compression Method: null (0)
  ▾ SSLv3 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 1368
    ▾ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 1364
        Certificates Length: 1361
      ▾ Certificates (1361 bytes)
          Certificate Length: 1358
        ▾ Certificate: 3082054a30820332020101300d06092a864886f70d010105... (id-at-commonName=sanshost,id…
          ▾ signedCertificate
              serialNumber: 1
            ▸ signature (sha1WithRSAEncryption)
            ▾ issuer: rdnSequence (0)
              ▸ rdnSequence: 5 items (id-at-commonName=SANS CA,id-at-organizationName=SANS Web App CA O…
            ▸ validity
            ▸ subject: rdnSequence (0)
            ▸ subjectPublicKeyInfo
          ▸ algorithmIdentifier (sha1WithRSAEncryption)
            Padding: 0
            encrypted: 14976dc5da83e275930f756dc938af9eda3dceaec7ce69b9...
```

Server Hello Details

Packet 12 is the first encrypted data packet.

How many bytes of encrypted data does it contain?

434

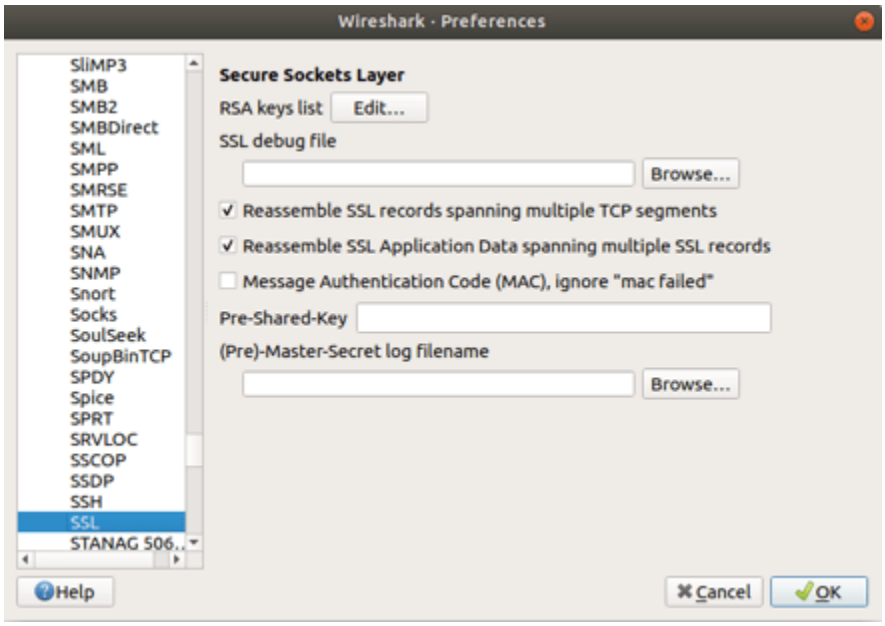
```
▶Frame 12: 493 bytes on wire (3944 bits), 493 bytes captured (3944 bits)
▶Ethernet II, Src: AsustekC_c9:4c:08 (00:1e:8c:c9:4c:08), Dst: Vmware_0d:20:76 (00
▶Internet Protocol Version 4, Src: 192.168.1.121 (192.168.1.121), Dst: 192.168.1.1
▶Transmission Control Protocol, Src Port: 54885 (54885), Dst Port: https (443), Se
▼Secure Sockets Layer
 ▼SSLv3 Record Layer: Application Data Protocol: http
    Content Type: Application Data (23)
    Version: SSL 3.0 (0x0300)
    Length: 434
    Encrypted Application Data: f55cdbc52752532de3ce6d7ff48ac1381fe75e8c268bad51...
```

Encrypted Packets

Next, we will try to decrypt the packets.

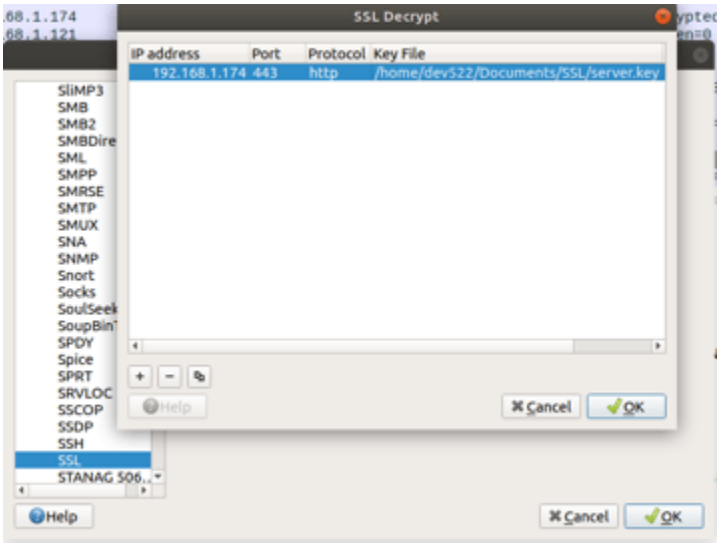
To do so, we need to provide Wireshark with the secret key.

1. Go to **Edit->Preferences**, then click on **"Protocols"** at the left-hand side of the Preferences window. Expand the list of protocols and find **"TLS"** (just type "TLS").



SSL Configuration Dialog

1. Click on the **"Edit"** button to the right of "RSA keys list:" and click the "+" button in the lower left-hand corner to add a new key.
2. You will now be able to enter the key parameters: IP Address: 192.168.1.174, Port: 443, Protocol: http. For the key file, please select **"Documents"** directory and then **"SSL"** where you can select the **"/home/student/Documents/SSL/server.key"**.



SSL Key Configuration

Leave the "Password" field empty and click "OK" and close the "Preferences" Dialog by clicking OK. As soon as you close the dialogs, you should see decrypted traffic.

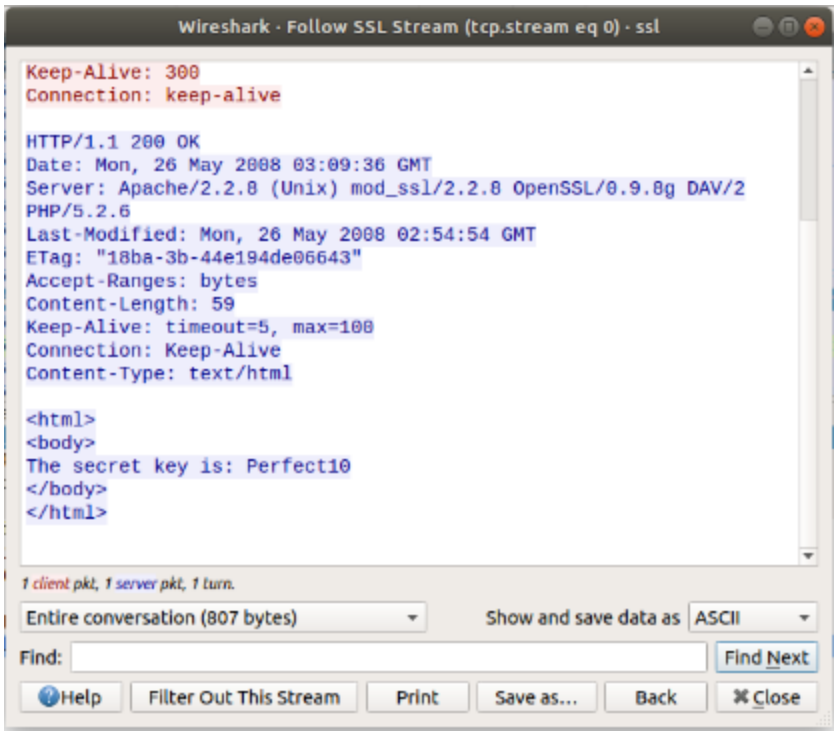
Packets 12 and 14 are now decrypted.

```
11 7.297009   192.168.1.174   192.168.1.121   SSLv3   121 Change Cipher Spec, Finished
12 7.298537   192.168.1.121   192.168.1.174   HTTP    493 GET /secret.html HTTP/1.1
13 7.299118   192.168.1.174   192.168.1.121   TCP      54 443 → 54885 [ACK] Seq=1529 Ack
14 7.311684   192.168.1.174   192.168.1.121   HTTP    485 HTTP/1.1 200 OK  (text/html)
15 7.502215   192.168.1.121   192.168.1.174   TCP      54 54885 → 443 [ACK] Seq=1100 Ack
```

3. Decrypted Packets in Wireshark
4. Right-click on packet #12, then select "Follow -> TLS Stream" to reconstruct the payload of the HTTP request and response.

What is the content of the payload of the response?

HTTP Request - The page has HTML content of a message "The Secret key is: Perfect 10"



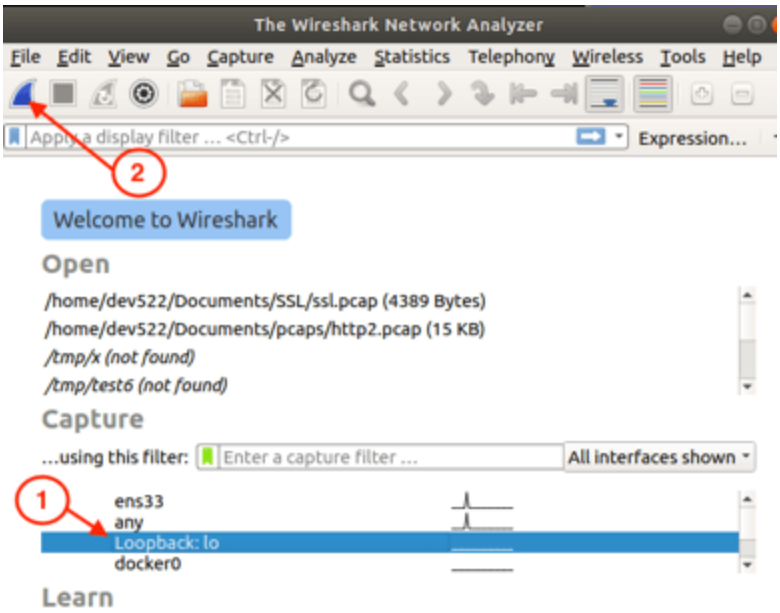
Decrypted SSL Stream

Advanced: SSL Master Key

In this exercise, we will decrypt SSL without knowing the server's private key. After all, the client has to be able to decrypt the content without the server's keys. Instead, the symmetric key that is negotiated during the SSL handshake is needed. If Diffie-Hellman ciphers are used, then this is the only method to decrypt traffic.

Chrome (and Firefox) are able to log the symmetric master keys to a file, and Wireshark is able to use this file to decrypt the content.

1. Start Wireshark.
2. Select the "Loopback: lo" interface and click the blue "fin" on the far left of the toolbar. Wireshark will now start to collect data.



Starting Packet Capture in Wireshark

3. Close Chrome first if it is running.
4. Open Chrome FROM THE COMMAND LINE (not by clicking on the icon). To do this, open a terminal and then enter "google-chrome". This will apply the necessary bash environment variables to Chrome to log the master keys.

```
google-chrome
```

5. Visit https://wiki.sans.labs/ (it is a TLS protected site)
6. Close Google Chrome.
7. Stop the acquisition in Wireshark by clicking on the red square.
8. To display only the SSL traffic, enter "tls" as a filter and click "Apply". You will recognize the now-familiar packets like "Client Hello" and "Server Hello".
9. Just like before, we enter the RSA key configuration dialog. Select Edit->Preferences from the Wireshark menu and proceed to the TLS protocol configuration. But instead of editing the RSA keys list, we just select a (Pre)-Master-Secret log filename at the bottom of the dialog.
10. Use /home/student/Documents/sslkeylogfile.log
11. Just like before, the HTTPS traffic will now be decrypted. But this time, we used symmetric keys captured by the client.

Information

The reason why we are launching Chrome from the shell is because of the environment variable in the shell environment called "SSLKEYLOGFILE". This environment variable tells the browser to capture a copy of the SSL key into the disk drive for debugging purpose. This method illustrated in part 2 of the exercise is exactly how anyone can debug their traffic with a TLS website. If you ever need to get to decipher the TLS protected traffic with a website, let's say, Google or Facebook. This method allows you to decrypt the traffic from the workstation.

Conclusion

- While TLS/SSL is known for protecting traffic between two network hosts, the traffic protected by it can be reverted to cleartext given the right circumstances.
- To debug your own SSL/TLS traffic, use the SSLKEYLOGFILE environment variable to enable the debugging mode in the browser so that encryption key can be captured.

Explore Further

- Visit an Internet site with SSL and see if you can capture the TLS key on your SSL session.

WSDL Enumeration and Parameter Tampering

Objectives

Estimated Time: 10 minutes

In this exercise, we will experiment with the SOAP Web Services injection and request manipulation techniques. The emphasis is on the fact that Web Services can be attacked just like a standard HTTP request in the case of a normal Web Application.

Requirements

- Lab VM

No Hints

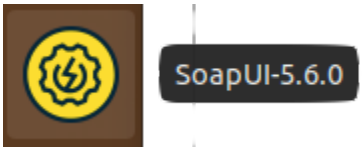
Go to http://lab-4.1.sans.labs/soapapi/?wsdl. Inspect the WSDL (this will be a lot easier with a tool like SOAPUI. See the left "Favorite" bar). There are two requests that do not require authentication. Identify the one that is sensitive and try to retrieve data.

Step by Step

1. Go to **http://lab-4.1.sans.labs/soapapi/?wsdl**. Your browser is a very good viewer for XML content, such as the WSDL file. With the WSDL content now on the screen, we can look for operations.
2. Look for operations listed in the WSDL file; specifically, look for the **operation** tag. You should be able to find it inside the **portType** element. If it is too hard to find the operation tag, press **Ctrl-F** to find the string on-screen.

   The operations in this WSDL are as follows: GetDayofWeek, GetAnimal, GetUser and GetUserName. So there are four operations listed by this WSDL.
3. Open SoapUI by clicking on the SoapUI icon on the left side of the screen. This application will take a full minute to load. Once loaded, you need need to skip

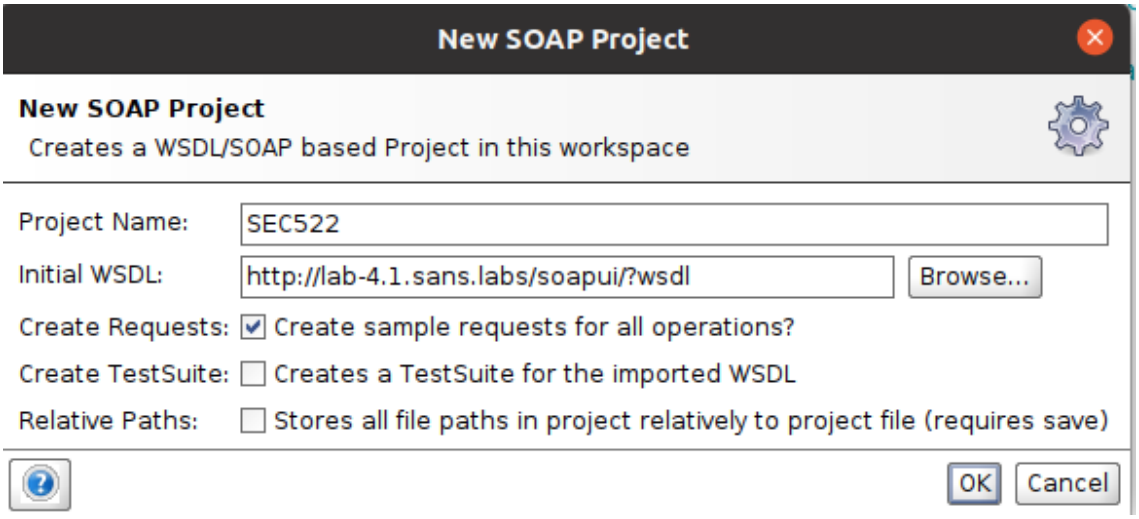
through the offer to collect your personal information. Also close the Endpoint Explorer that pops up.



SoapUI Icon

4. Create a new project by clicking on the link from the main screen or through **File > New SOAP Project** from the menu or the hotkey combination Ctrl + N. When the New SOAP Project dialog opens, give your new project a name (e.g., "SEC522"), and load the WSDL from the URL -

```
http://lab-4.1.sans.labs/soapapi/?wsdl
```



Creating a New SOAP Project in SoapUI.

5. Click "OK".

   This will load the WSDL and on the left-hand side you will see the "Operations". The operations in this WSDL are as follows: GetAnimal, GetDayofWeek, GetUser, and GetUserName, as we gathered in an earlier part of this exercise.

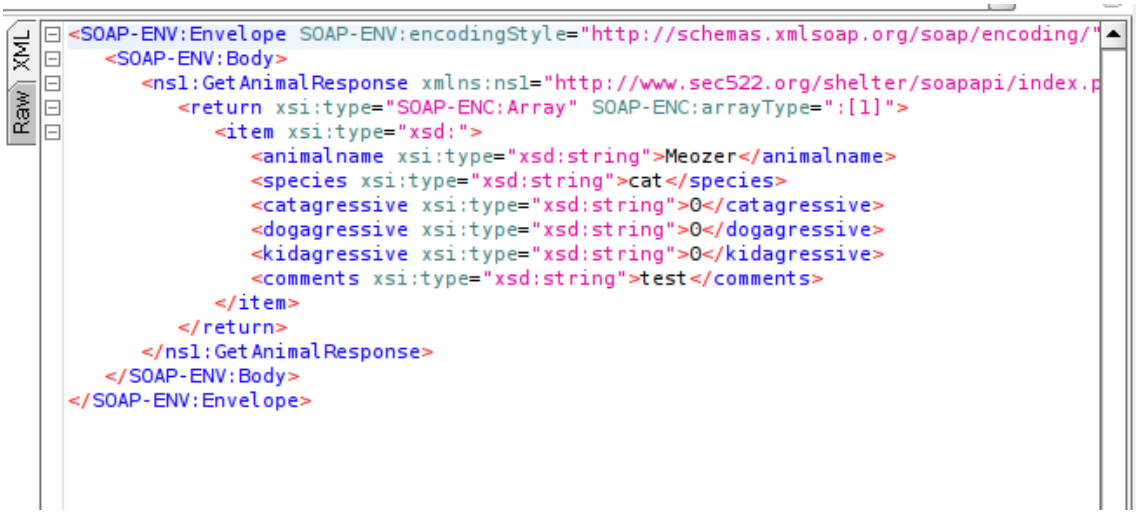
6. Click on the "+" to the left of "GetAnimal" and double-click on "Request 1".

7. In the following request window, enter an ID (e.g., 1) in the appropriate spot in the XML. SoapUI prefills the location with a "?". The API Key is **catsruledogsdrool**.

XML Snippet for SOAP Request

8. Next, click on the little "" ("Play") icon in the upper left-hand corner of the request window.

   You should now see the response in parsed XML.



9. XML Response in SoapUI.

   We have now triggered one operation; Inspect the remaining operations and find those that do not use an API key.

10. Just like in Step 6 and 7, create requests for GetDayofWeek, GetUser and GetUserName. For "GetDayofWeek", the date uses the YYYY-MM-DD format. For example 2021-04-01 for April 1st 2021.

11. GetDayofWeek and GetUser do not require an API key. GetDayofWeek has no sensitive information. But GetUserName has. Just like in steps 6 through 8, use SoapUI to retrieve data for user # 1.

Conclusion

- XML based Web Services are common and while they are intended to communicate from machine to machine, it is possible to use clients such as SoapUI to interact with the service and exploit any flaws that are in the system.

Explore Further

- Interact with public SOAP based web services such as http://www.dneonline.com/calculator.asmx

XMLHttpRequest and Same Origin Policy

Objectives

Estimated Time: 10 minutes

PART 1:

In this exercise, we will experiment with the JavaScript method XMLHttpRequest and the ability of this request to generate HTTP requests and fetch data within a JavaScript.

Requirements

- Lab VM

XMLHttpRequest and Same Origin Policy

Step by Step

1. Make sure Chrome is running.
2. Browse to **http://lab-4.2.sans.labs/sop.php**.
3. This page is assembled with contents from static HTML and also XMLHttpRequest. Let's explore more of the innerworkings of this page. Launch the Developer Tools (Ctrl-Shift-I) and select the **Sources** tab from the top and then select sop.php.
4. The source of the page including the JavaScript that loads all the content within the page. Please review the code from line 39-53.

```
<script>
    url = 'http://lab-4.2.sans.labs/content.php';
    req = new XMLHttpRequest();
    req.onreadystatechange = function() {
    if ( req.readyState == 4 ) {
        var response='waiting...';
        if ( req.status == 200 ) {
        response = req.responseText;
        } else {
        response="Sorry. The data could not be retrieved. Error:"+req.statusText;
```



```
        }
        document.getElementById('content').innerHTML=response;
    }
    }
    req.open('GET',url,true);
    req.send();
</script>
```

By reading the page source, answer these questions:

What value is the url variable holding?

The url variable is holding the URL of the resource to be fetched by XMLHTTPRequest. The content.txt file is in the lab-4.2.sans.labs server. This location will have a great effect on a later part of this lab.

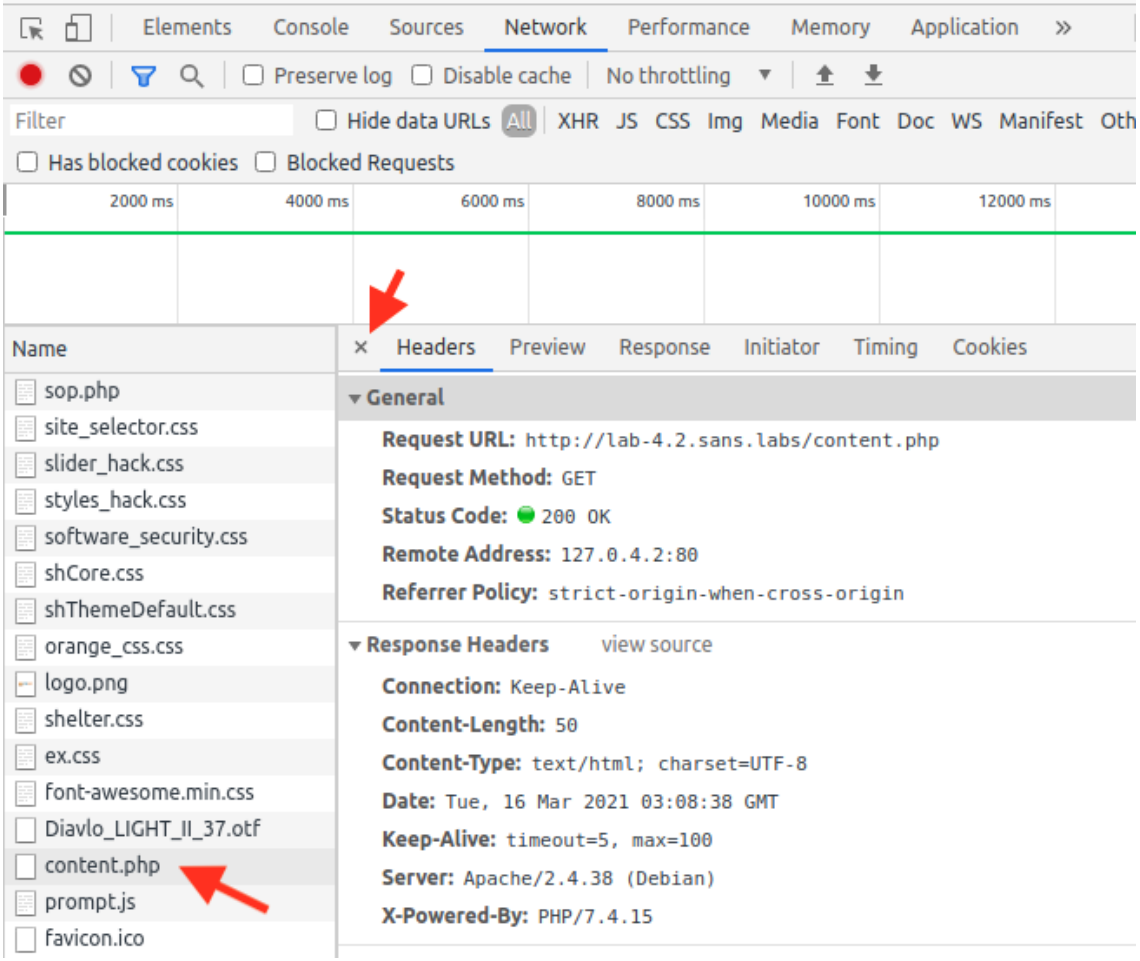
What does the *alert(response);* statement perform?

The response is a variable holding the content of an HTTP response; this data is returned by the XMLHTTPRequest. Alert is the function that generated the pop-up error message. The content of the pop-up box will be filled with the data returned by XMLHTTPRequest.

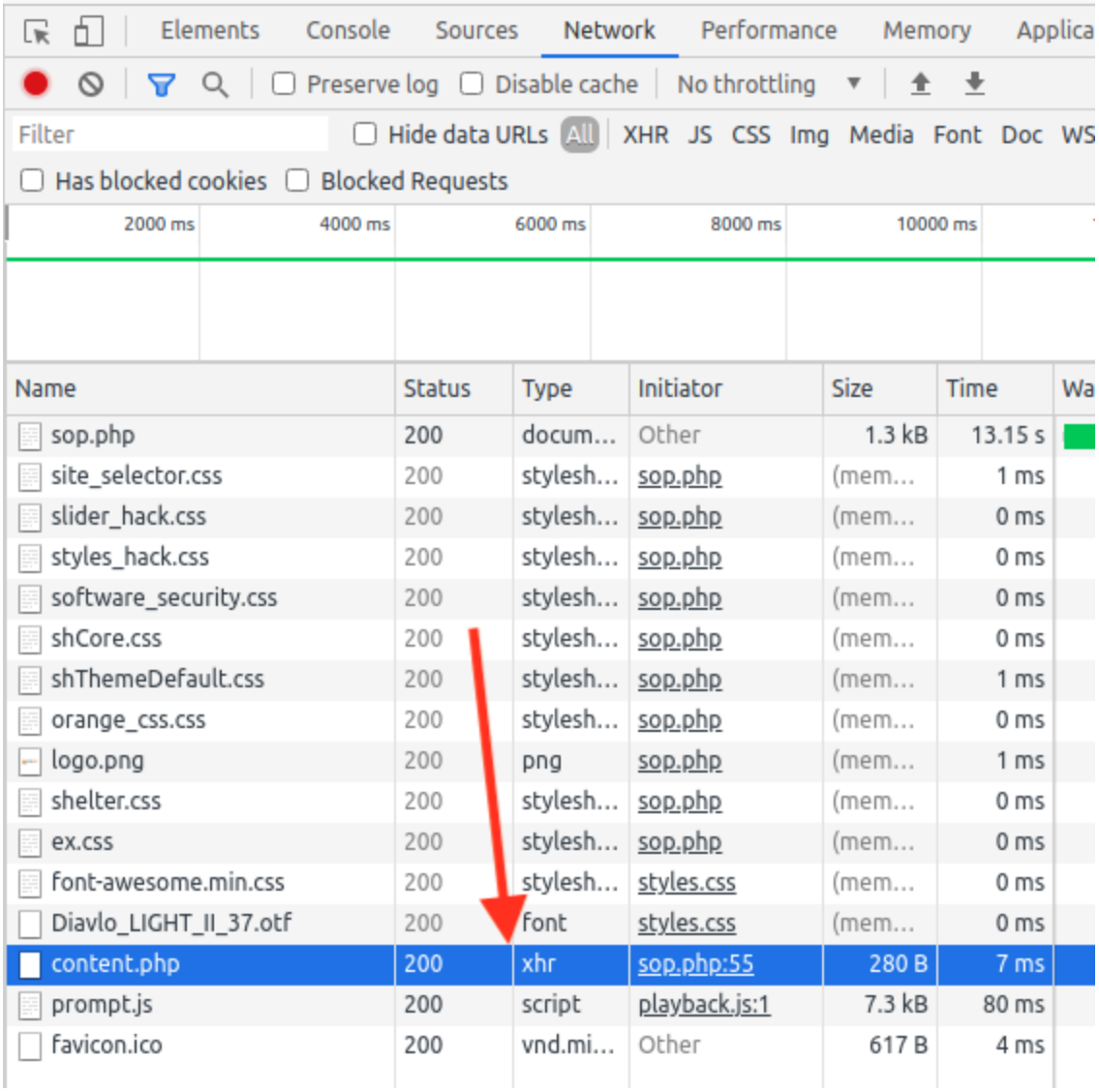
Which statement in the JavaScript code executes the sending of an HTTP request?

The "req.send();" statement sends the XMLHTTPRequest out to the web server.

5. Reload the page then switch over to the "Network" tab at the top. Look for **content.php** at the bottom tab. Close the bottom right tab (click the X next to Headers)



6. See the request type by the browser, it is XHR.



It is time to experience the restriction imposed by the Same Origin Policy.

7. Browse to **http://lab-4.2-evil.sans.labs/sop.php** (note the URL is different than the one we used before)
8. With the Developer Tools, inspect the "Console" tab from the top tab menu. Take a look at the red error messages. It's clear that this content.php is blocked.
9. Take a look at the source code by selecting the **Sources** tab from the top and then select **sop.php**. Please review line 39 onwards which is the portion of the code causing the browser violation.

```
<script>
    url = 'http://lab-4.2.sans.labs/content.php';
```

```
    req = new XMLHttpRequest();
    req.onreadystatechange = function() {
.....
```

XMLHTTPRequest targets "lab-4.2.sans.labs" under Same Origin Policy; if the URL is not "lab-4.2-**evil**.sans.labs", the browser will not continue with the request. As you can tell, XMLHTTPRequest happens in the background, silently making HTTP requests on your behalf. If it is allowed to request contents across domains, it would be very dangerous to the user. Think of the situation in which the user visits evil.com where there is JavaScript instructing the browser to attack another site using XMLHTTPRequest, sans.org. If the Same Origin Policy does not stop this, your browser will be attacking sans.org without your knowledge.

Conclusion

- Same Origin Policy restriction is embeded in the browser.
- Sam Origin Policy prevent content (scripts) from one origin to directly interact with the content from another origin.

Explore Further

- Find out how you can disable the Same Origin Policy in the browser

JavaScript Security Testing

Objectives

Estimated Time: 15 minutes

PART 1:

In this exercise, we will experiment with the built-in JavaScript debugger in Chrome and learn the basics of using this debugger for understanding the JavaScript running on a webpage.

Requirements

- Lab VM

Part 1: Accessing Arbitrary Shopping Carts

Step by Step

Chrome and many other browsers include capable JavaScript debuggers. They are commonly used by security professionals to understand the inner workings of a piece of JavaScript code.

This exercise uses Juice Shop.

1. Start Juice Shop. Open a terminal and type

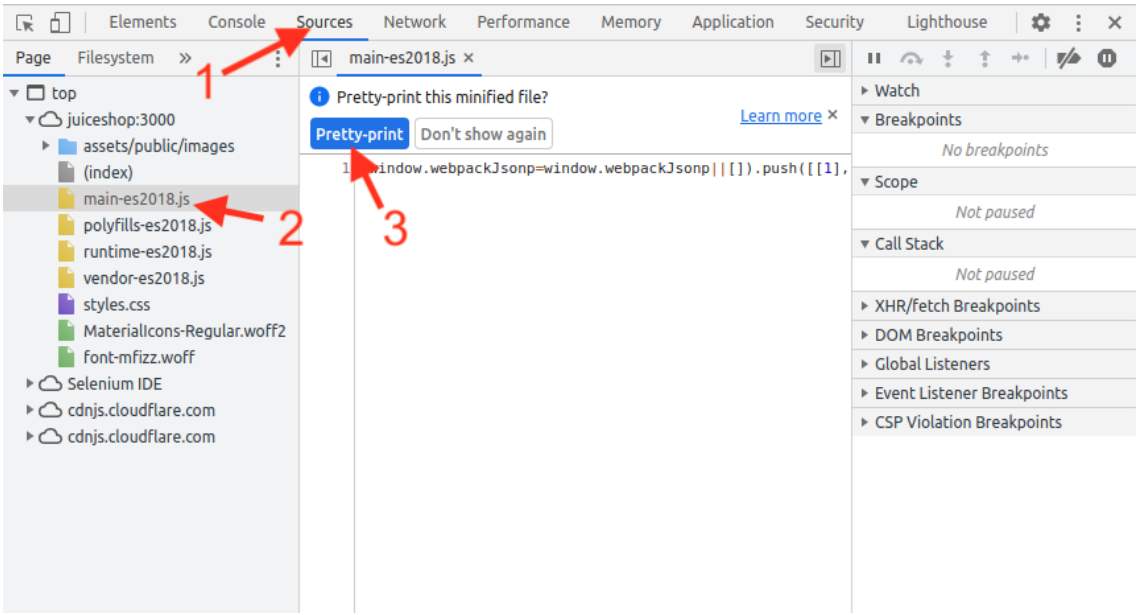
   ```
   juiceshop start
   ```

   You may see an error if Juice Shop is already running.
2. Go to Juice Shop in Chrome by visiting the URL http://juiceshop:3000.
3. Log in to Juice Shop using sec522@sec522.org and the password "training". Register a new account if the login fails (Juice Shop will reset its database whenever it is shut down).

4. Press CTRL-SHIFT-I to open the debugger in Chrome.
5. Select "Application" -> "Session Storage" and review the session storage for "juiceshop".
6. You will see a parameter called "bid". Change it to a different number (e.g., "3").
7. Review the shopping cart in Juice Shop ("Your Basket") by clicking on the shopping cart at the top of the page. You will see someone else's items.

Part 2: Analyzing Minimized JavaScript *(Advanced)*

1. In the Chrome Debugger, select "Sources" and open the "main-es2018.js" file.
2. Click the "{}" at the lower left-hand corner of the source code window. This will make the minimized code more readable.



JavaScript Debugger "main-es2018.js"

3. Line 12206-12218 contains obfuscated JavaScript code meant to hide a particular endpoint. Copy the code (Select the code, then Ctrl-C), starting with function(), all the way to "toLowerCase()" in line 12218.

```
12203 component: S
12204 [
12205 matcher: function(t) {
12206     return 0 === t.length ? null : t[0].toString().match(function(...t) {
12207         const e = Array.prototype.slice.call([25, 184, 174, 179, 182, 186])
12208           , a = e.shift();
12209         return e.reverse().map(function(t, e) {
12210             return String.fromCharCode(t - a - 45 - e)
12211         }).join("")
12212     }() + 36669..toString(36).toLowerCase() + function(...t) {
12213         const e = Array.prototype.slice.call(arguments)
12214           , a = e.shift();
12215         return e.reverse().map(function(t, e) {
12216             return String.fromCharCode(t - a - 24 - e)
12217         }).join("")
12218     }(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase()) ? {
12219         consumed: t
12220     } : null
12221 },
12222 component: v
```

Obfuscated JavaScript Code

4. Change to the **Console** tab (at the top). Type ""+ (empty quotes followed by "+") into the console and paste the code you copied right after the "+". Hit enter. You will see the deobfuscated result. (see answers at the end for the solution).

```
> ""+function(...t) {
                    const e = Array.prototype.slice.call([25, 184, 174, 179, 182, 186])
                      , a = e.shift();
                    return e.reverse().map(function(t, e) {
                        return String.fromCharCode(t - a - 45 - e)
                    }).join("")
                }() + 36669..toString(36).toLowerCase() + function(...t) {
                    const e = Array.prototype.slice.call(arguments)
                      , a = e.shift();
                    return e.reverse().map(function(t, e) {
                        return String.fromCharCode(t - a - 24 - e)
                    }).join("")
                }(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase()
```

Pasted Code in Console

Solution

The token is: "tokensale-ico-ea".

The hidden URL includes a white paper for a "Juicecoin" ICO.

5. Go to the hidden endpoint (http://juiceshop:3000/#/ [solution from step 4]).

Information

Despite the effort from the developer to obfuscate the "key" in the JavaScript code, the code is reversible with some level of coding skills. Lessons learned - do not hide keys in obfuscation.

Conclusion

- JavaScript loaded in the browser can be reverse engineered by the user easily. A debugger comes with every browser and can be used to understand the flow of the JavaScript programs.
- Obfuscation using JavaScript does not safeguard the program code or the logic/ data hidden by the script.

Explore Further

- Explore other flaws within Juice Shop

Content Security Policy

Objectives

Estimated Time: 15 minutes

Content Security Policy (CSP) is a great tool to limit JavaScript that an attacker may inject into a page. But if the CSP is too tight, then JavaScript code that was added by the developer may not run. In this exercise, you will learn how to debug CSPs.

PART 2:

Requirements

- Lab VM

No Hints

Proceed to http://lab-4.4.sans.labs/. The exercise will guide you through three parts. In each part, a CSP is shown that is too strict to display an image on the site. Modify the CSP to fix it. You can modify the CSP right on the page, and relevant CSP errors are displayed as well. You can also use the web console to learn more about the CSP issue.

Step-by-Step Instructions

1. Open **http://lab-4.4.sans.labs/** in your browser.
2. On that page, you will find a link to http://lab-4.4.sans.labs/csperror.html which is the log file. Open that link in a new window.
3. Click on the "CSP Error 1" link.
4. The page is missing an image. The "Last Error" section will show the last CSP error. You may have to reload the page if the section is empty. You can also refresh the error log page to get the latest errors, but that page may be difficult to read.
5. Inspect the blocked-uri section in the last error.

> **Does it match the img-src?**
>
> img-src lab-1.1.sans.labs

6. Edit the CSP to reflect the correct img-src.
7. Click "Update".
8. If your CSP is correct, then the image of a cat will appear, as well as a link to the next exercise.
9. For the next two pages, follow the same procedures as you did between Step 4 and Step 8.

> **Solutions**
>
> CSP1: img-src lab-4.4.sans.labs
>
> CSP2: script-src 'self' 'unsafe-inline'
>
> CSP3: script-src 'unsafe-inline' 'unsafe-eval'

Conclusion

- CSP is effective at protecting the web page against multiple types of attacks.
- CSP Requires some careful configuration, otherwise, it can backfire and render portions of the site not working.
- Using the CSP logging function (report-uri) is a must for troubleshooting

Explore Further

- Can you think of the scenarios where CSP can be bypassed?

DNS Rebinding

Objectives

Estimated Time: 15 minutes

PART 1:

In this exercise, you will learn how a modern browser deals with DNS rebinding issues and how an attacker may exploit this flaw.

PART 2:

In this exercise, you will see for yourself a practical way that attacker can use Javascript to get around the anti-DNS rebinding mechanism of the browsers.

Requirements

- Lab VM

Part 1: Simple DNS Rebinding

No Hints

There are two parts to this exercise. In the first part, you will experiment with DNS rebinding without taking advantage of JavaScript. The second exercise explores the use of JavaScript and XMLHTTPRequest to profile the anti-DNS rebinding done by the browser. Start at http://lab-5.1-evil.sans.labs/evil.php.

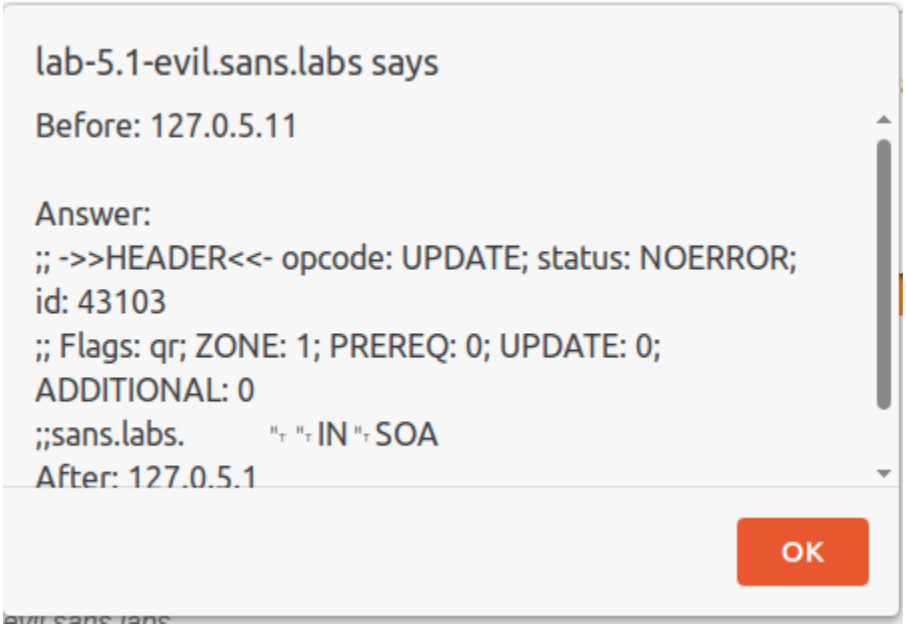
Step by Step

In this exercise, we will explore the possibilities of DNS rebinding (anti-DNS pinning) attacks firsthand. The attack demonstrated is dependent on the DNS record timing out within the browser, and then the attacker can rebind another host, making the victim\'s browser confused about the actual IP of the host it is trying to reach.

1. Using Chrome, browse to **http://lab-5.1.sans.labs/evil.php**. This is the "Good Dog Pics" page. The host you are connecting to is **lab-5.1.sans.labs**.

> **Note**
>
> The host name lab-5.1.sans.labs have dog picture. Watch how this fact is important later.

2. We are now going to the attacker's controled site. Using Chrome, browse to **http://lab-5.1-evil.sans.labs/evil.php**. The host name is similar to the one in previous step but this is a different site. The content are also different as well.
3. Click on **CLICK HERE TO SWITCH DNS**. This will start the process of DNS rebinding.
4. You will be presented with a Javascript prompt. This triggers a DNS changes on the server end as indicated by the content of the pop up box. The IP for the lab-5.1-evil.sans.labs was **127.0.5.11** and it is being changed to **127.0.5.1**. What that means is anyone asking for the IP of lab-5.1-evil.sans.labs will now be presented with 127.0.5.1.



5. After about 2 minutes, reload the page in the browser. Notice the URL? It is still http://lab-5.1-evil.sans.labs/evil.php but the content in the page are coming from the "Good Dog Pics" page which is at the IP address 127.0.5.1.

> **Note**
>
> The exercise starts off with the browser visiting lab-5.1-evil.sans.labs (127.0.5.11). At this time, the browser has the IP of the host pinned. After around 120 seconds when the DNS TTL expires, we reloaded the page. This time, the sans.labs DNS replies that lab-5.1-evil.sans.labs is located at 127.0.5.1 (not the same as before). This IP happens to be the server for lab-5.1.sans.labs (the Dog page). The Same Origin Policy has been violated with the time out and second DNS query since your browser URL is pointing to one domain and the content is coming from another location.

6. After this exercise, please click the Reset DNS and restart the browser to ensure all cached host records (DNS pin) is gone.

Part 2: JavaScript based Rebinding with XMLHTTPRequest

Similar to the DNS rebinding with the browser, the JavaScript engine can also be fooled as well. Its DNS pinning mechanism is actually isolated from the browsers.

1. Browse to http://lab-5.1-evil.sans.labs/eviljs.php
2. Similar to the previous exercise, we start off with the evil cat's page and Javascript immediately kick into action to change the DNS in the background. There is a long wait for the DNS pin to expire within the JavaScript engine. Review the change in the Javascript prompt and then click "OK".
3. The JavaScript automatically count down. In the background, every second there is a fetch request being sent to determine if the DNS pin has been lifted by the browser so the DNS would be re-queried and therefore change to the alternative host.
4. After about 100 seconds, the Chrome browser will lift the DNS pin and the JavaScript detects this and send the attack to the alternative host.

Conclusion

- DNS rebinding allows attacker to "confused" the browser into loading content from a different origin than what's specified by the host name and domain name.

Explore Further

- What would happen if the target site has TLS enabled?

Clickjacking

Objectives

Estimated Time: 10 minutes

PART 1:

In this exercise, you will experiment with a clickjacking exploit.

PART 2:

In this exercise, you will experiment with the defensive controls against the Clickjacking attack

Requirements

- Lab VM

Part 1: Clickjacking Attack

No Hints

The page http://lab-5.2.sans.labs/clickjack.php implements a clickjacking exploit. Explore how it works.

Which page is being 'clickjacked'?

Step by Step

1. Browse to **http://lab-5.2.sans.labs/vote.php**.
2. This is a very simple page where user can click on either red or blue box for voting and the vote is tallied.
3. Click the **"Dogs are the best!"** and **"Cats are the best!"** button a few times each to observe how this works.

4. Browse to **http://lab-5.2.sans.labs/clickjack.php**. (Clickjacked vote on the upper right hand corner) This is an evil page that exploits clickjacking vulnerability.
5. Try to click on the **"Dogs are the best"**. Although not much seems to be happening to the user, actions have already been taken in the background.
6. Click on the **"click here to reveal/hide iframe"** link on the page.
7. A translucent page is shown on the page, partially overlapping the existing web page. The translucent page contains content from **http://lab-5.2.sans.labs/vote.php**. This is exactly how clickjacking happens. The translucent portion of the page was "on top" of the visible page; when the user clicks on the page, the mouse clicks are sent to the page on top. The user effectively clicks on something that he/she cannot see, and the mouse click is redirected to the other page.

   Let's inspect the effect of the clickjacking attack.
8. Browse to **http://lab-5.2.sans.labs/vote.php**. Check the counter. Why is the count higher than last time you loaded this page?

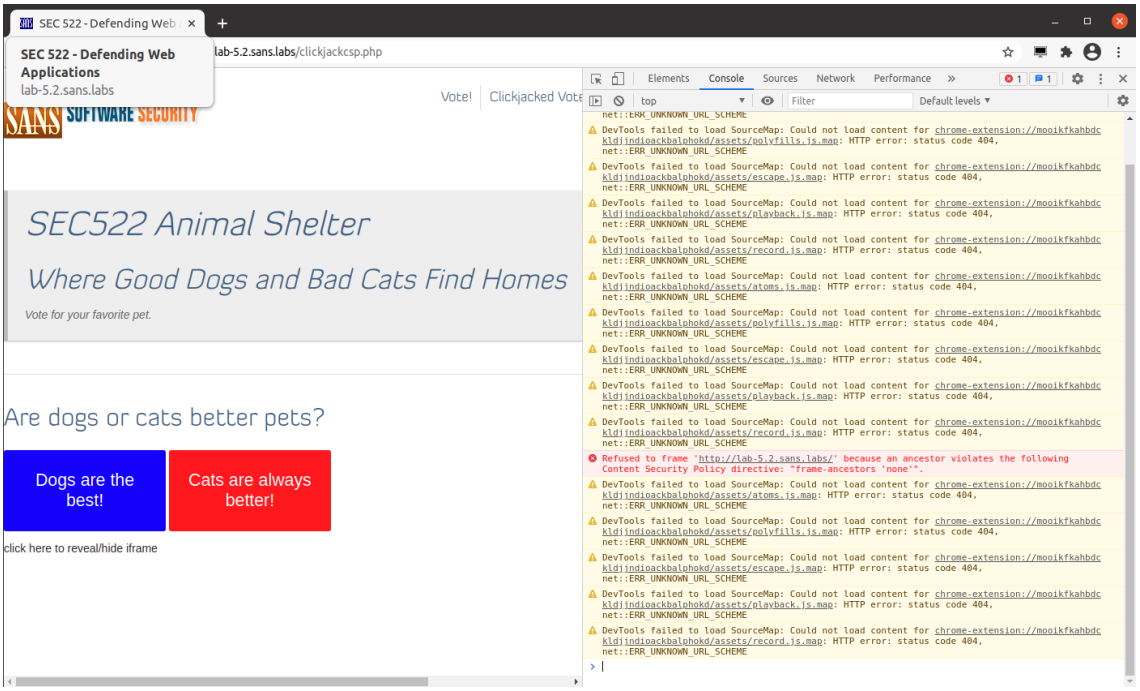
> **Info**
>
> In clickjacking, the attacker sets up pages that look harmless and entices the victim to click on some parts of the page. Hidden from the view of the victim, there is an invisible layer of another webpage which is receiving all the mouse clicks

Part 2: Defending Against ClickJacking with CSP

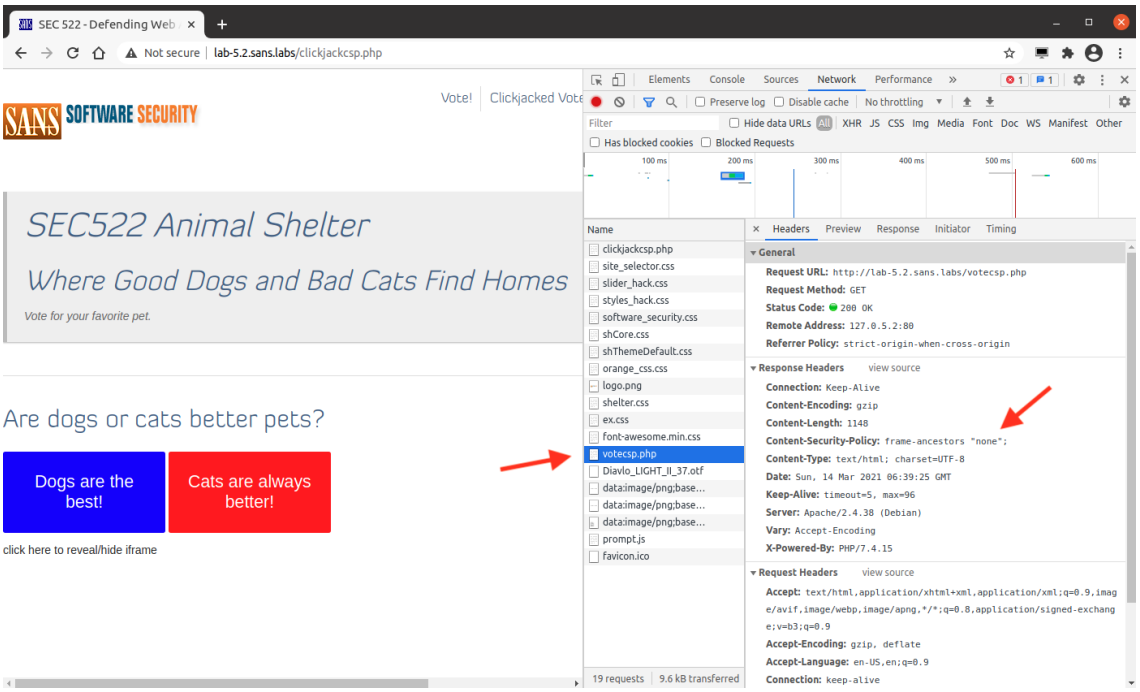
Clickjacking defense essentially relies on headers from Website to let the browser know whether the content can be inside an IFrame. Being loaded in IFrame essentially means that a malicious party can ClickJack the website.

1. Browse to http://lab-5.2.sans.labs/clickjackcsp.php. This page is very similar to the attack page earlier in that it loads a victim page so that mouse click from the user will land on the content inside the IFrame.

2. Click on the **"click here to reveal/hide iframe"** link on the page. You will notice that the victim's content are not loaded this time, instead a broken symbol is shown for the translucent content. See if you can determine what is causing this attack to fail?
3. Let us walk through the defense mechanics here. In Chrome tab with the page http://lab-5.2.sans.labs/clickjackcsp.php, launch the Developer Tools (Ctrl-Shift-I).
4. Refresh the page. This will let Developer Tools to reveal all the details of the page to you. Click on the **Console** tab in Developers Tools. There is one single error of the browser refusing to load the content of "http://lab-5.2.sans.labs/" due to the Content Security Policy.



5. Let's also inspect the exact Content Security Policy that was sent to the browser from the victim's page. Click on the **Network** tab in Developers Tools. Look for **votecsp.php** and click on it. Make sure the right hand side pane has "headers" selected at the top. This will show the headers that were exchanged. Look at Response Headers and evaluate the Content Security Policy. Can you determine why it was blocked from loading by the attacker's page?



Content Security Policy

The Content Security Policy loaded was

```
Content-Security-Policy: frame-ancestors "none";
```

This blocks the content within the HTTP response from being loaded into any Iframe.

Conclusion

- CSP with frame-ancestor is effective at preventing the clickjacking attack from happening.

Explore Further

- If the Content Security Policy conflicts with X-Frame-Options header, what happens? Does the browser complain about it?

HTML5

Objectives

Estimated Time: 10 minutes

PART 1:

In this exercise, you will become familiar with various HTML5 features and how they affect security. The first exercise will cover the difference of local and session storage. The second one will showcase security issues in cross-window messaging. The third and last exercise will cover HTML5 input validation.

PART 2:

In this exercise, we will explore the potential problems with unsecure file upload handling and see for ourselves the effect of improperly handling upload.

Requirements

- Lab VM

Part 1: Session vs. Local Storage

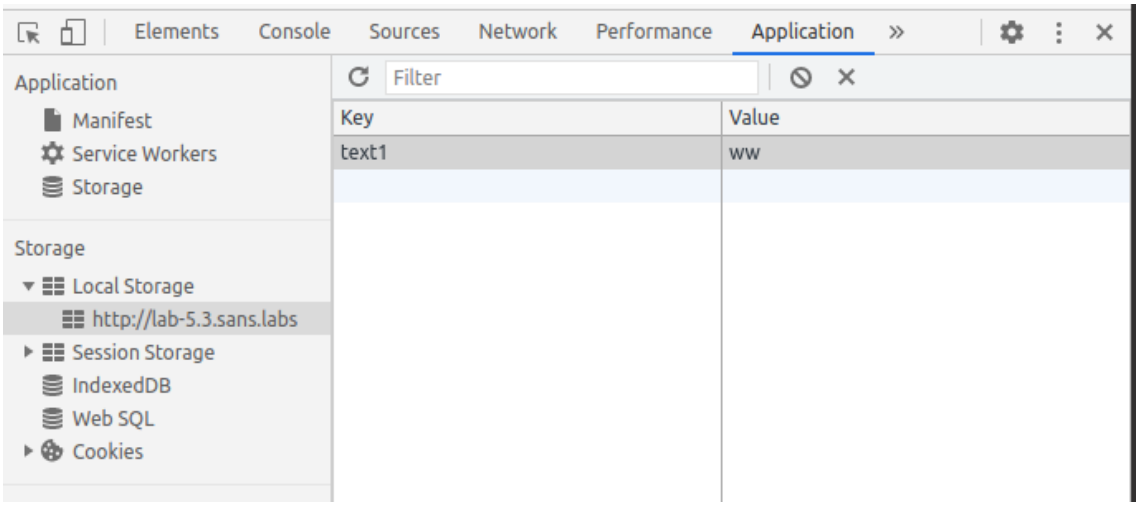
No Hints

Start at http://lab-5.3.sans.labs/html5.html. You will find links to the three exercises and basic instruction about what to do. Attempt to fix the security issue for the stock trading application in #2.

Step by Step

1. Browse to **http://lab-5.3.sans.labs/html5.html**.
2. Click on "1 -- Local Storage and Session Storage".
3. Enter some random text in both text areas (Storage #1 and Storage #2).
4. Click on both "Save" buttons.

5. Open the same page, **http://lab-5.3.sans.labs/storage.html**, in a new tab (press CTRL+T, or use the "File" menu in the browser).
6. In this new tab, to the right of the text areas, you will see "Saved Data". Only one of these fields is prefilled; the other field is empty. For one of the text areas, LocalStorage was used. For the other one, we used SessionStorage. Which one is which?
7. With both tabs open, enter different values in Storage #1 and Storage #2. Refresh the other tab to confirm your assumption.
8. In Chrome, open the developer tools (press SHIFT-CTRL-I) and select the "Application" tab. You will see options to inspect local and session storage on the left.



Adjusting Local Storage

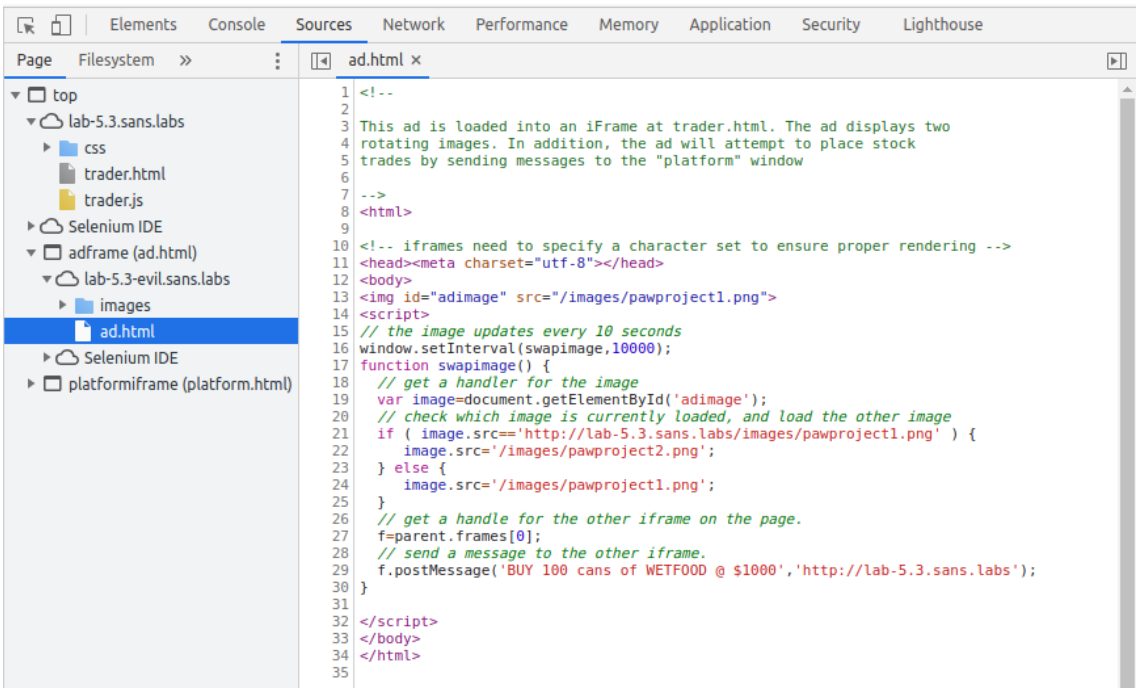
9. You will be able to review Local Storage and Session Storage. You will see the values you entered in the text area. For Session Storage, you will see different values in different tabs.

Answer

Storage #1 is kept in LocalStorage. Storage #2 is kept in SessionStorage. As a result, Storage #1 is the same for both tabs, while Storage #2 is different.

Part 2: Cross-Window Messaging

1. Open the "Cross-Domain Messaging" exercise at http://lab-5.3.sans.labs/trader.html
2. You will see a simple "Stock Trading" application. Place a trade in the green "Trade" window by selecting a stock symbol and entering a quantity.
3. After you click the "Execute" button, your trade will show up in the "Trade Status" frame. The stock trading application sends trades to the "Trade Status" frame, where they are then passed to the trading API.
4. Click on the ad on the left to enable ads.
5. After 10 seconds, the ad image will change. At the same time, a new trade will appear in the "Trade Status" window. Why?
6. Open the Chrome developer tools (SHIFT-CTRL-I) and select the "Sources" tab.
7. On the left, you will find "ad.html" after expanding "adframe (ad.html)" and "lab-5.3-evil.sans.labs". (If there are no content under ad.html, reload the web page) The "postMessage" function will be found on line 29.



8. Select "platform.html" on the left. This will bring up the trading platform's source code that receives the message.
9. Does the function "receiveMessage" (starting on line 6) check for the origin of the message?

10. **Extra Credit**

Can you fix the error?

You can find the source code for the files in /etc/docker/compose/5.3/web/html/platform.html.

Answer

platform.html never checks the origin of the message. This will allow any window to send messages if that window can obtain a handle to the "platform" IFrame. To

check for the origin, you could add this code.

```
If ( event.origin != 'http://lab-5.3.sans.labs') {
   return 0
}
```

In this example, checking the origin is, however, insufficient, as the ad is served from the same origin as the rest of the page. The ad should be loaded from a different origin.

Part 3: Input Validation

In this exercise, you will see four different input fields. Each field is supposed to validate the input to limit data submitted to the description to the left of the input field. For example, the user is not supposed to enter a number that exceeds 100 in the first input field. The actual validation is either too strict or too loose. Find out what needs to be improved to correct the regular expressions used to validate the data.

1. Open http://lab-5.3.sans.labs/valid.html in Chrome.
2. In the first input field, enter "1000" and click the "Submit" button. You will not receive an error message (but you should).
3. Right-click on the input field and select "Inspect". This will highlight the respective line in the page's source code:

```
<input type="text" pattern="[0-9]+">
```

4. Double-click on the regular expression. You will be able to edit it.
5. Change it to [0-9][0-9]. Now the large input will be rejected. Note that in this case, you do not need anchors for regular expressions.
6. Repeat Steps 2--5 for the other three input fields. Inspect the element to see the regular expression. Is it correct? If not, then try to enter input that would match the regular expression, but not the expected input. Also, verify if there is valid input that will be rejected.

Answer

The correct regular expressions are:

1

```
[0-9][0-9]
```

The original expression did not limit the length of the number. Other possible solutions: [0-9]{2}, \d\d, \d{2}

2

```
[0-9A-F]
```

The original expression allowed for letters up to H. G and H are not valid hexadecimal characters.

3

```
[0-9]{5}
```

Like the first input field, the length is not restricted. For extra credit, you could come up with a regular expression that allows for ZIP+4: [0-9]{5}(-[0-9]{4})?

4

For international phone numbers, it is hard to limit the length, as different lengths apply for different countries. But the country code may start with a '+', which is not allowed here. +?[0-9]+ is a better pattern. The ITU-T, which sets standards for phone systems, recommends that numbers should never exceed 15 digits, which includes the country code. Following this standard, the regular expression would be +?\d{1,15}

Conclusion

- Modern HTML interface has many different functions that can make the website more responsive. At the same time, some of the features also impact security.
- Local storage and other browser based storage need careful evaluation on privacy and security impact before use.

Sub Resource Integrity

Objectives

Estimated Time: 6 minutes

Understand how sub resources integrity can help prevent some attacks that swap components hosted by third parties for malicious components.

Requirements

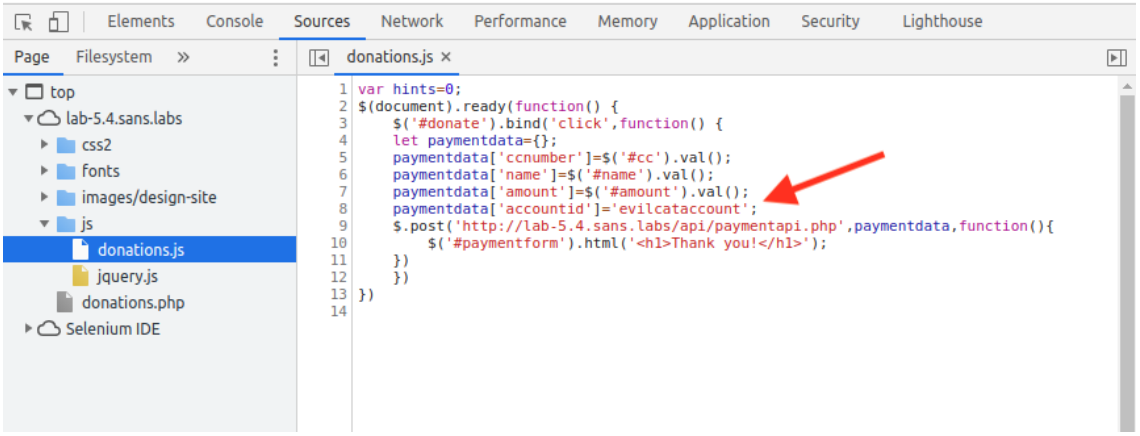
- Lab VM

No Hints

Start the exercise at http://lab-5.4.sans.labs/donations.php and follow the instructions provided.

Step by Step

1. Open http://lab-5.4.sans.labs/subresourceintegrity.php in Google Chrome.
2. Click on the link to donations.php - http://lab-5.4.sans.labs/donations.php.
3. Fill out the donation form and click "Submit." You will see a "Thank you" page, and all looks good.
4. Click on the http://lab-5.4.sans.labs/donations.php link again to restart the exercise.
5. Press Ctrl-Shift-I to open the Developers Tools.
6. Click on the "Sources" tab in the Developers Tools.
7. Under the js folder, open the "donations.js" file and inspect the "accountid". Note how the account ID has been altered to "evilcataccount" by an unknown attacker!



Content of donations.js

8. Open a terminal and enter

```
cd /etc/docker/compose/5.4/web/html/js
```

to change to the directory with the respective files.

9. Enter "ls" to obtain a directory listing. You will see a file called "dontations_for_stupid_dogs.js".
10. Use the "diff" command to compare the dontations_for_stupid_dogs.js file to the donations.js file. The files are identical with the exception of the account ID.

    In a terminal, run

```
diff donations_for_stupid_dogs.js  donations.js
```

to show the differences between two files.

11. Calculate the hash of the "good" file:

```
openssl dgst -sha384 -binary < donations_for_stupid_dogs.js | openssl base64 -A
```

You should get the following hash:
wZGpBo0UBEBZ2r+uvUPUVH8gvHqHHPJr5Yt9aK+W5D9cSxNhgvF6lAHxsf0gAXbw

12. Open **/etc/docker/compose/5.4/web/html/donations.php** in an editor (Visual Studio Code) and add the "integrity" attribute to the script tag for "donations.js" in line 17. (Solutions below) Save the file after you are done editing and keep the file open.
13. Reload the donations page. You will now see an error on the console tab in the Developer Tools.

Information

We have generated a hash (signature) of the legitimate JavaScript but the file being referenced remotely is a file that has been altered by the unknown attacker. As the hashes are not matching, the browser is not going to execute the JavaScript.

14. Change line 17 of the donations.php to use "donations_for_stupid_dogs.js" instead of donations.php (keep the integirty parameter intact).
15. Reload the donations page. The script will now load as the signature matches.

The "integrity" parameter will indicate to the browser that the file has changed. The browser is now able to block scripts if they changed.

Solution

Here is the JavaScript reference with integrity attributes embeded (line 17 of donations.php).

```
<script src="/js/donations.js" integrity="sha384-
wZGpBo0UBEBZ2r+uvUPUVH8gvHqHHPJr5Yt9aK+W5D9cSxNhgvF6lAHxsf0gAXbw"></script>
```

Conclusion

- Subresource integrity is useful for making sure that content being loaded from a remote location is of a specific version or that it hasn't been altered.
- When using Subresource integrity, the destination resource changes will cause the browser to stop loading the resource.
- Subresource integirty works for any <link> and <script> elements

Logging and Incident Response

Objectives

Estimated Time: 10 minutes

Use a standard web server log of an incident to figure out what happened.

Requirements

Lab VM

No Hints

The application under attack is the animal shelters "Citrus Gateway". The "Citrus Gateway" allow access to internal application, but the shelter never really used it, so it sits in its default configuration exposed to the internet. There are two attacks happening. The log does not have necessarily all the information. For the second attack, the "CGW-User" header was logged. To play with the application, see http://lab-5.5.sans.labs/citrusgw/index.php . The files for the application can be found in /etc/docker/compose/5.5/web/html/citrusgw

Step by Step

The logs use the following format (the last field is only present for the last few lines of the log):

[hostname] [attacker IP] - - [data/time] "METHOD URL HTTP/1.1" [Status Code] [Response Size] "-" "[User-Agent]" "CGW-User Header"

As an example, this line:

www.sec522.org:80 127.0.0.1 - - [26/Jan/2020:00:25:58 -0500] "POST /citrusgw/api/newbm.php HTTP/1.1" 200 207 "-" "l33t@tt@ck" "test"

Indicates:

Field	Value
Hostname	www.sec522.org
Client IP (ignore for this exercise)	127.0.0.1
Date/Time	26/Jan/2020:00:25:58 -0500
HTTP Method	POST
URL Requested	/citrusgw/api/newbm.php
Status Code	200 (OK)
Response Size	207 Bytes
User-Agent	l33t\@tt\@ck
CGW-User Header	test

1. Explore the attacked application to become familiar with it. Start by loading http://lab-5.5.sans.labs/citrusgw/index.php in Google Chrome
2. Log in using the credentials "sec522" and "training".
3. Enter a bookmark (any text will do).
4. Click on the "Files" icon on your desktop (Favorite menu on the left).
5. Open the "Documents" folder
6. Open the "Logs" folder
7. Double click on "attack.logs"
8. The file starts out with one type of attack. Review the first approximately 100 lines that are all accessing "index.php" and answer the following questions:

1. What type of attack do you think the attacker conducted?

2. Any indication if the attack succeeded?

3. Any simple signature you could use to block this attacker?

4. Any defensive technique you would recommend?

9. The second part of the exercise uses the log file you have already open, but the requests are now hitting newbm.php. Refer to the field list in the introduction above to identify which field changes for some of the requests.
10. You may want to open /etc/docker/compose/5.5/web/html/citrusgw/api/newbm.php first in a text editor (Visual Studio Code) to understand a bit more what is happening.

    Try to answer these questions:

5. What type of attack was the attacker launching?

6. Was it successful?

7. What other artifacts would you look for on this system to figure out if the attack was successful?

Answers

1. What type of attack do you think the attacker conducted?

The attacker likely launched a brute force attack. The login page (index.php) was hit very quickly with what looks like similar requests.

2. Any indication if the attack succeeded?

The last request returned a 302 status code (redirect) while the others returned a 200 code. This likely indicates that the last attempt succeeded.

3. Any simple signature you could use to block this attacker?

The attacker uses a very unique user agent. This user agent could easily be blocked with a firewall.

4. Any defensive technique you would recommend?

It may be a good idea to implement some kind of rate limiting and lock out for the login form.

5. What type of attack was the attacker launching?

The attacker attempted a directory traversal attack

6. Was it successful?

It may be successful as the response size changed for some of the request.

7. What other artifacts would you look for on this system to figure out if the attack was successful?

The attack attempts to write a file "test.txt" to a "tmp" directory, which may be found on the system.

Conclusion

- Log analysis can yield a lot of insights on the activities of the application. In incident response scenario, logs are often the main source of information to understand what had happened.

Explore Further

- How do you fix an application coding issue illustrated in this exercise?