14. Using Web Proxies

Intro to Web Proxies

Today, most modern web and mobile applications work by continuously connecting to back-end servers to send and receive data and then processing this data on the user's device, like their web browsers or mobile phones. With most applications heavily relying on back-end servers to process data, testing and securing the back-end servers is quickly becoming more important.

Testing web requests to back-end servers make up the bulk of Web Application Penetration Testing, which includes concepts that apply to both web and mobile applications. To capture the requests and traffic passing between applications and back-end servers and manipulate these types of requests for testing purposes, we need to use `Web Proxies`.

What Are Web Proxies?

Web proxies are specialized tools that can be set up between a browser/mobile application and a back-end server to capture and view all the web requests being sent between both ends, essentially acting as man-in-the-middle (MITM) tools. While other `Network Sniffing` applications, like Wireshark, operate by analyzing all local traffic to see what is passing through a network, Web Proxies mainly work with web ports such as, but not limited to, `HTTP/80` and `HTTPS/443`.

Web proxies are considered among the most essential tools for any web pentester. They significantly simplify the process of capturing and replaying web requests compared to earlier CLI-based tools. Once a web proxy is set up, we can see all HTTP requests made by an application and all of the responses sent by the back-end server. Furthermore, we can intercept a specific request to modify its data and see how the back-end server handles them, which is an essential part of any web penetration test.

Uses of Web Proxies

While the primary use of web proxies is to capture and replay HTTP requests, they have many other features that enable different uses for web proxies. The following list shows some of the other tasks we may use web proxies for:

- Web application vulnerability scanning
- Web fuzzing
- Web crawling
- Web application mapping
- Web request analysis
- Web configuration testing
- Code reviews

In this module, we will not discuss any specific web attacks, as other HTB Academy web modules cover various web attacks. However, we will thoroughly cover how to use web proxies and their various features and mention which type of web attacks require which feature. We will be covering the two most common web proxy tools: `Burp Suite` and `ZAP`.

Burp Suite

Burp Suite (Burp) -pronounced Burp Sweet- is the most common web proxy for web penetration testing. It has an excellent user interface for its various features and even provides a built-in Chromium browser to test web applications. Certain Burp features are only available in the commercial version `Burp Pro/Enterprise`, but even the free version is an extremely powerful testing tool to keep in our arsenal.

Some of the `paid-only` features are:

- Active web app scanner
- Fast Burp Intruder
- The ability to load certain Burp Extensions

The community `free` version of Burp Suite should be enough for most penetration testers. Once we start more advanced web application penetration testing, the `pro` features may become handy. Most of the features we will cover in this module are available in the community `free` version of Burp Suite, but we will also touch upon some of the `pro` features, like the Active Web App Scanner.

Tip: If you have an educational or business email address, then you can apply for a free trial of Burp Pro at this link to be able to follow along with some of the Burp Pro only features showcased later in this module.

OWASP Zed Attack Proxy (ZAP)

OWASP Zed Attack Proxy (ZAP) is another common web proxy tool for web penetration testing. ZAP is a free and open-source project initiated by the Open Web Application

Security Project (OWASP) and maintained by the community, so it has no paid-only features as Burp does. It has grown significantly over the past few years and is quickly gaining market recognition as the leading open-source web proxy tool.

Just like Burp, ZAP provides various basic and advanced features that can be utilized for web pentesting. ZAP also has certain strengths over Burp, which we will cover throughout this module. The main advantage of ZAP over Burp is being a free, open-source project, which means that we will not face any throttling or limitations in our scans that are only lifted with a paid subscription. Furthermore, with a growing community of contributors, ZAP is gaining many of the paid-only Burp features for free.

In the end, learning both tools can be quite similar and will provide us with options for every situation through a web pentest, and we can choose to use whichever one we find more suitable for our needs. In some instances, we may not see enough value to justify a paid Burp subscription, and we may switch to ZAP to have a completely open and free experience. In other situations where we want a more mature solution for advanced pentests or corporate pentesting, we may find the value provided by Burp Pro to be justified and may switch to Burp for these features.

Setting Up

Both Burp and ZAP are available for Windows, macOS, and any Linux distribution. Both are already installed on your PwnBox instance and can be accessed from the bottom dock or top bar menu. Both tools are pre-installed on common Penetration Testing Linux distributions like Parrot or Kali. We will cover the installation and setup process for Burp and Zap in this section which will be helpful if we want to install the tools on our own VM.

Burp Suite

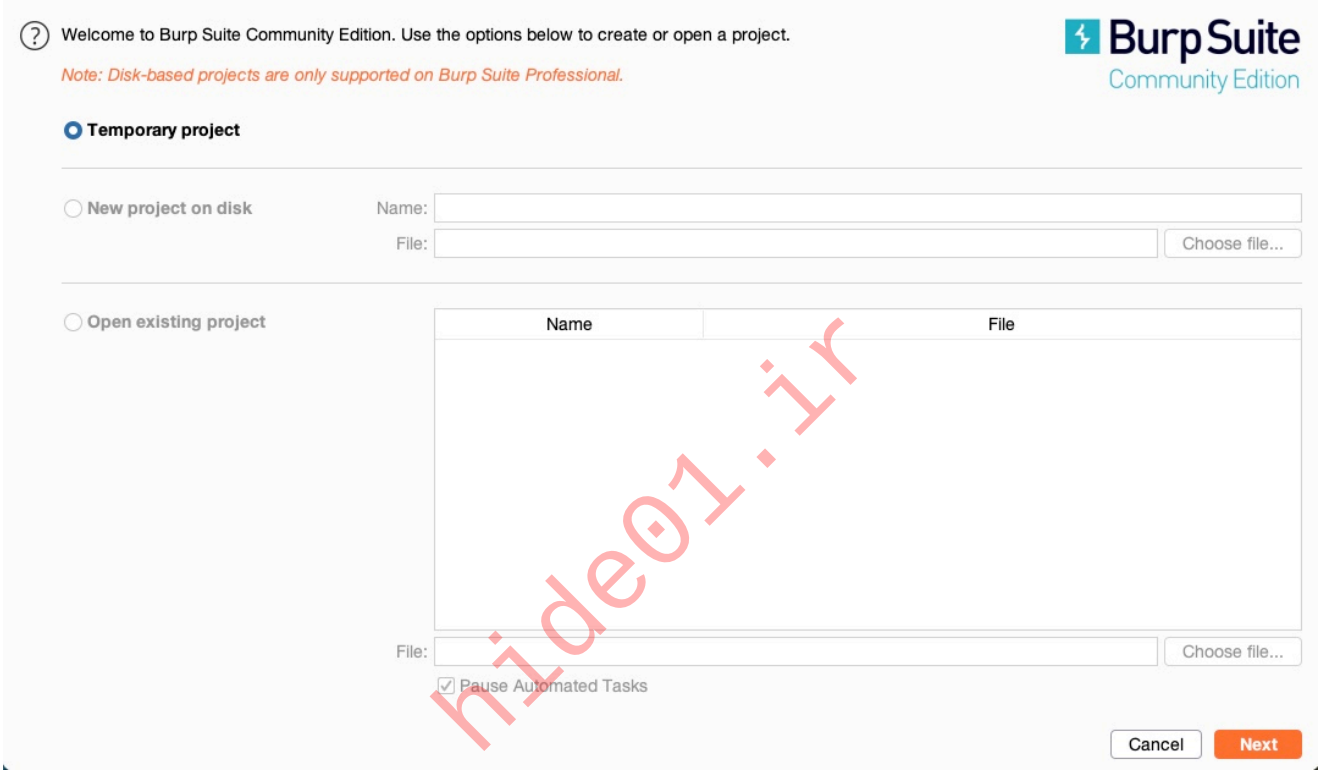
If Burp is not pre-installed in our VM, we can start by downloading it from Burp's Download Page. Once downloaded, we can run the installer and follow the instructions, which vary from one operating system to another, but should be pretty straightforward. There are installers for Windows, Linux, and macOS.

Once installed, Burp can either be launched from the terminal by typing `burpsuite`, or from the application menu as previously mentioned. Another option is to download the `JAR` file (which can be used on all operating systems with a Java Runtime Environment (JRE) installed) from the above downloads page. We can run it with the following command line or by double-clicking it:

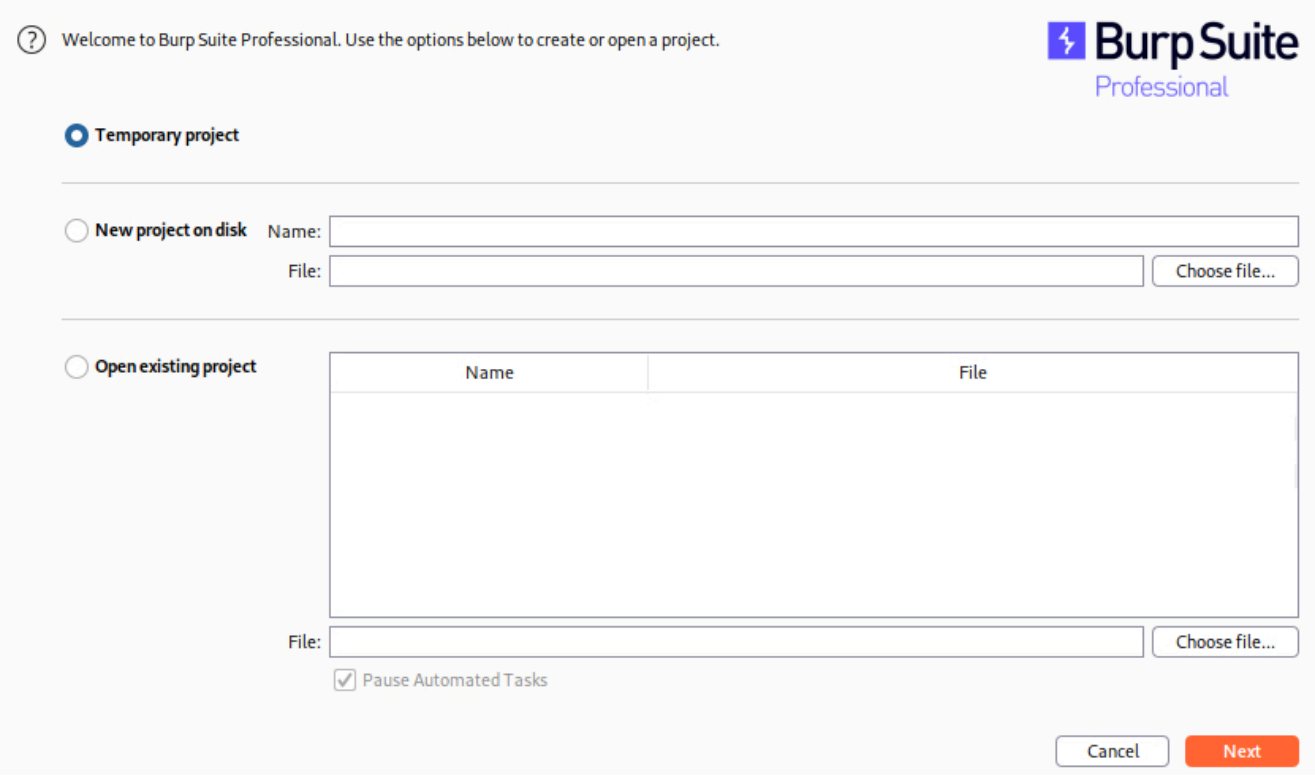
```
java -jar </path/to/burpsuite.jar>
```

Note: Both Burp and ZAP rely on Java Runtime Environment to run, but this package should be included in the installer by default. If not, we can follow the instructions found on this page.

Once we start up Burp, we are prompted to create a new project. If we are running the community version, we would only be able to use temporary projects without the ability to save our progress and carry on later:

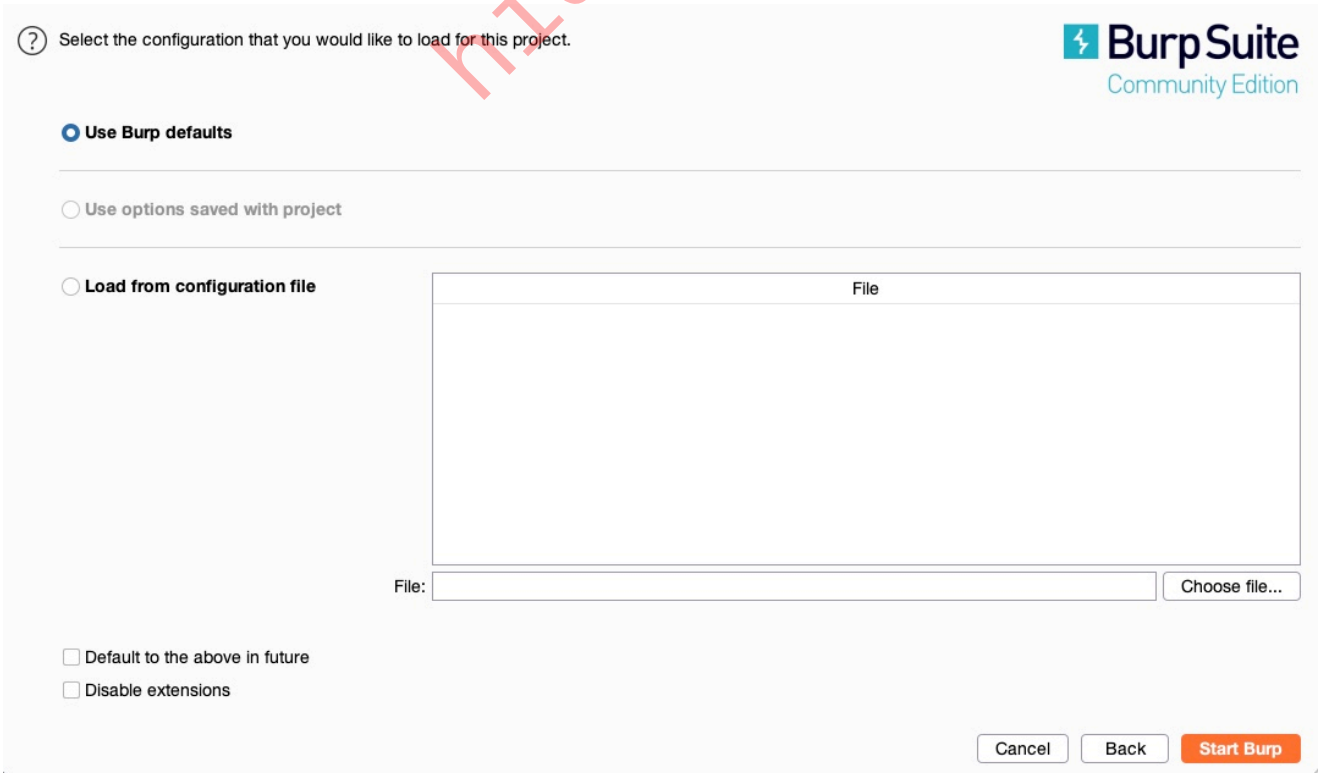


If we are using the pro/enterprise version, we will have the option to either start a new project or open an existing project.



We may need to save our progress if we were pentesting huge web applications or running an `Active Web Scan`. However, we may not need to save our progress and, in many cases, can start a `temporary` project every time.

So, let's select `temporary project`, and click continue. Once we do, we will be prompted to either use `Burp Default Configurations`, or to `Load a Configuration File`, and we'll choose the first option:



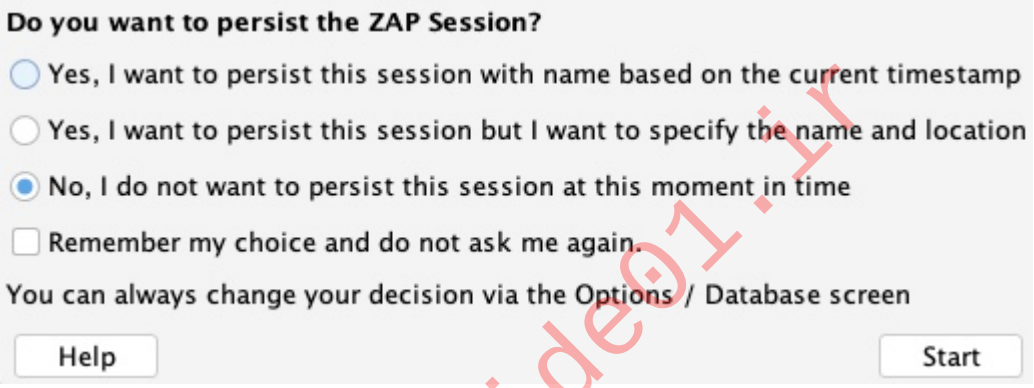
Once we start heavily utilizing Burp's features, we may want to customize our configurations and load them when starting Burp. For now, we can keep `Use Burp Defaults`, and `Start`

`Burp`. Once all of this is done, we should be ready to start using Burp.

ZAP

We can download ZAP from its download page, choose the installer that fits our operating system, and follow the basic installation instructions to get it installed. ZAP can also be downloaded as a cross-platform JAR file and launched with the `java -jar` command or by double-clicking on it, similarly to Burp.

To get started with ZAP, we can launch it from the terminal with the `zaproxy` command or access it from the application menu like Burp. Once ZAP starts up, unlike the free version of Burp, we will be prompted to either create a new project or a temporary project. Let's use a temporary project by choosing `no`, as we will not be working on a big project that we will need to persist for several days:



After that, we will have ZAP running, and we can continue the proxy setup process, as we will discuss in the next section.

Tip: If you prefer to use to a dark theme, you may do so in Burp by going to (`User Options>Display`) and selecting "dark" under (`theme`), and in ZAP by going to (`Tools>Options>Display`) and selecting "Flat Dark" in (`Look and Feel`).

Proxy Setup

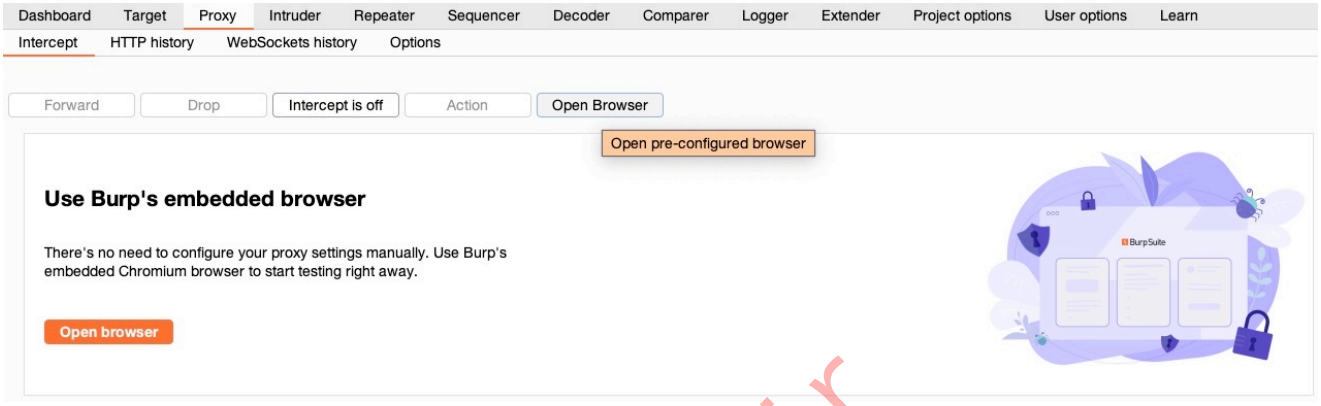
Now that we have installed and started both tools, we'll learn how to use the most commonly used feature; `Web Proxy`.

We can set up these tools as a proxy for any application, such that all web requests would be routed through them so that we can manually examine what web requests an application is sending and receiving. This will enable us to understand better what the application is doing in the background and allows us to intercept and change these requests or reuse them with various changes to see how the application responds.

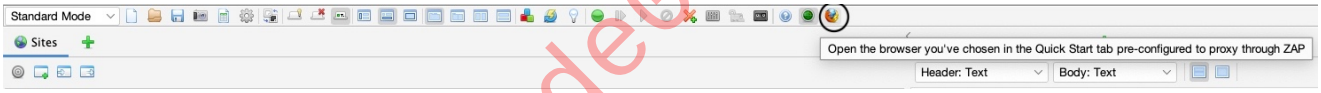
Pre-Configured Browser

To use the tools as web proxies, we must configure our browser proxy settings to use them as the proxy or use the pre-configured browser. Both tools have a pre-configured browser that comes with pre-configured proxy settings and the CA certificates pre-installed, making starting a web penetration test very quick and easy.

In Burp's (`Proxy>Intercept`), we can click on `Open Browser`, which will open Burp's pre-configured browser, and automatically route all web traffic through Burp:



In ZAP, we can click on the Firefox browser icon at the end of the top bar, and it will open the pre-configured browser:



For our uses in this module, using the pre-configured browser should be enough.

Proxy Setup

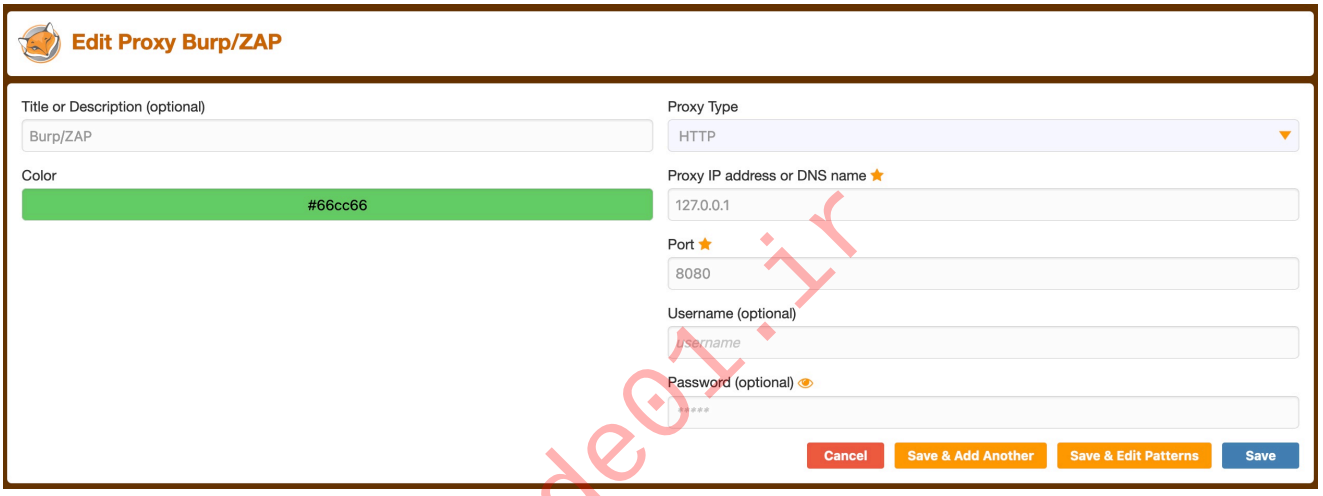
In many cases, we may want to use a real browser for pentesting, like Firefox. To use Firefox with our web proxy tools, we must first configure it to use them as the proxy. We can manually go to Firefox preferences and set up the proxy to use the web proxy listening port. Both Burp and ZAP use port `8080` by default, but we can use any available port. If we choose a port that is in use, the proxy will fail to start, and we will receive an error message.

Note: In case we wanted to serve the web proxy on a different port, we can do that in Burp under (`Proxy>Options`), or in ZAP under (`Tools>Options>Local Proxies`). In both cases, we must ensure that the proxy configured in Firefox uses the same port.

Instead of manually switching the proxy, we can utilize the Firefox extension Foxy Proxy to easily and quickly change the Firefox proxy. This extension is pre-installed in your PwnBox instance and can be installed to your own Firefox browser by visiting the Firefox Extensions Page and clicking `Add to Firefox` to install it.

Once we have the extension added, we can configure the web proxy on it by clicking on its icon on Firefox top bar and then choosing `options`:

Once we're on the `options` page, we can click on `add` on the left pane, and then use `127.0.0.1` as the IP, and `8080` as the port, and name it `Burp` or `ZAP`:



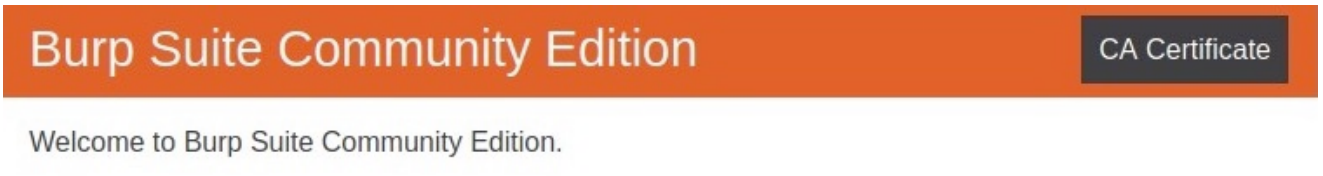
Note: This configuration is already added to Foxy Proxy in PwnBox, so you don't have to do this step if you are using PwnBox.

Finally, we can click on the `Foxy Proxy` icon and select `Burp`/ `ZAP`.

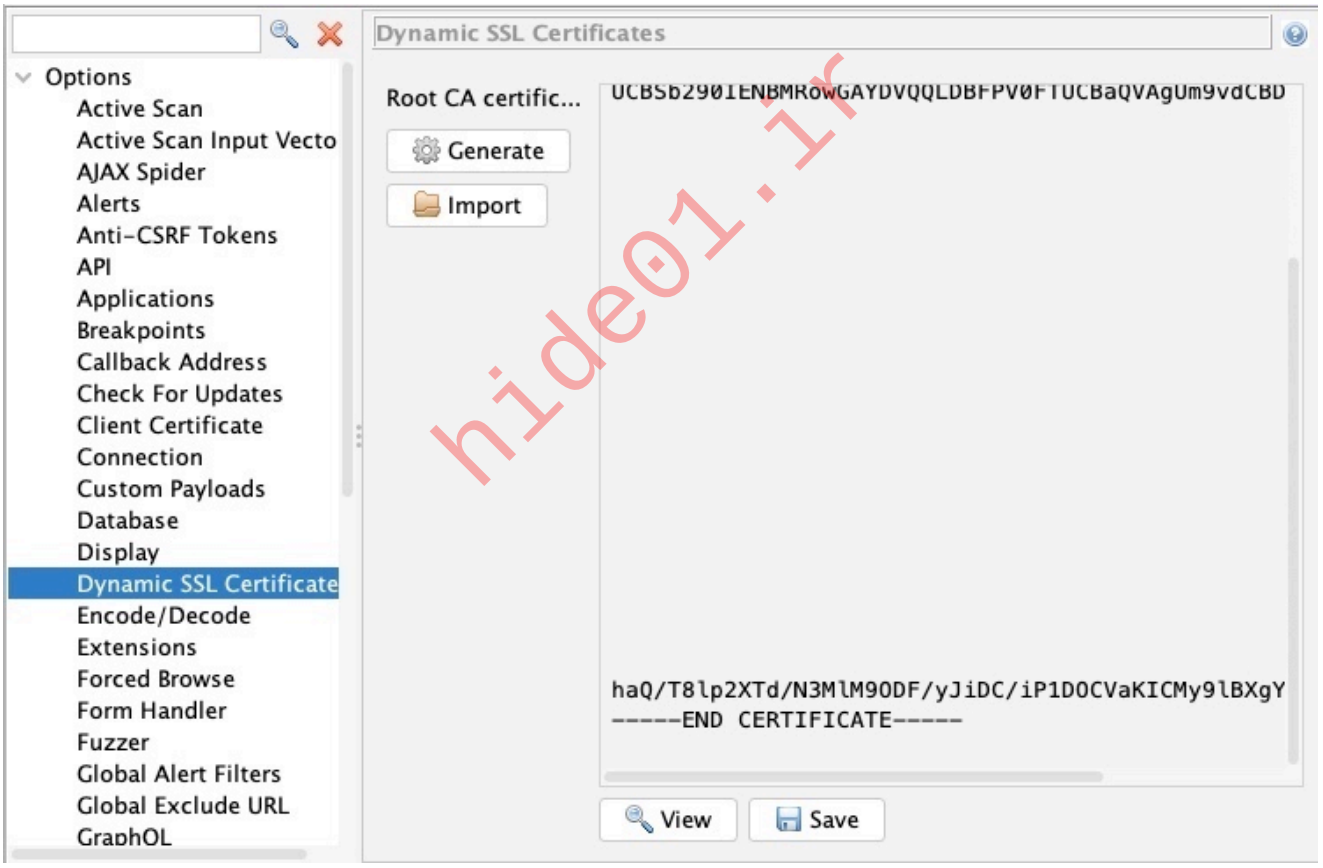
Installing CA Certificate

Another important step when using Burp Proxy/ZAP with our browser is to install the web proxy's CA Certificates. If we don't do this step, some HTTPS traffic may not get properly routed, or we may need to click `accept` every time Firefox needs to send an HTTPS request.

We can install Burp's certificate once we select Burp as our proxy in `Foxy Proxy`, by browsing to `http://burp`, and download the certificate from there by clicking on `CA Certificate`:

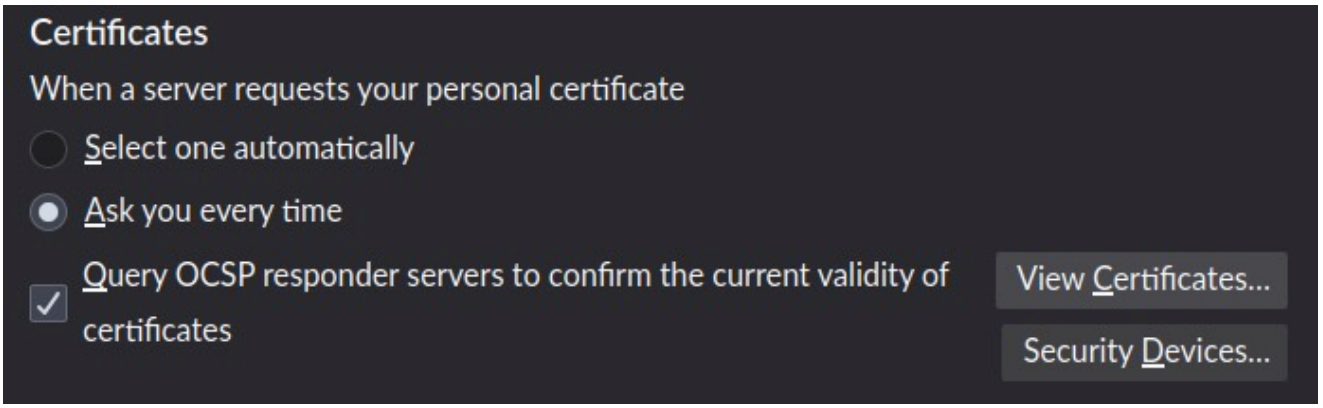


To get ZAP's certificate, we can go to (`Tools>Options>Dynamic SSL Certificate`), then click on `Save`:

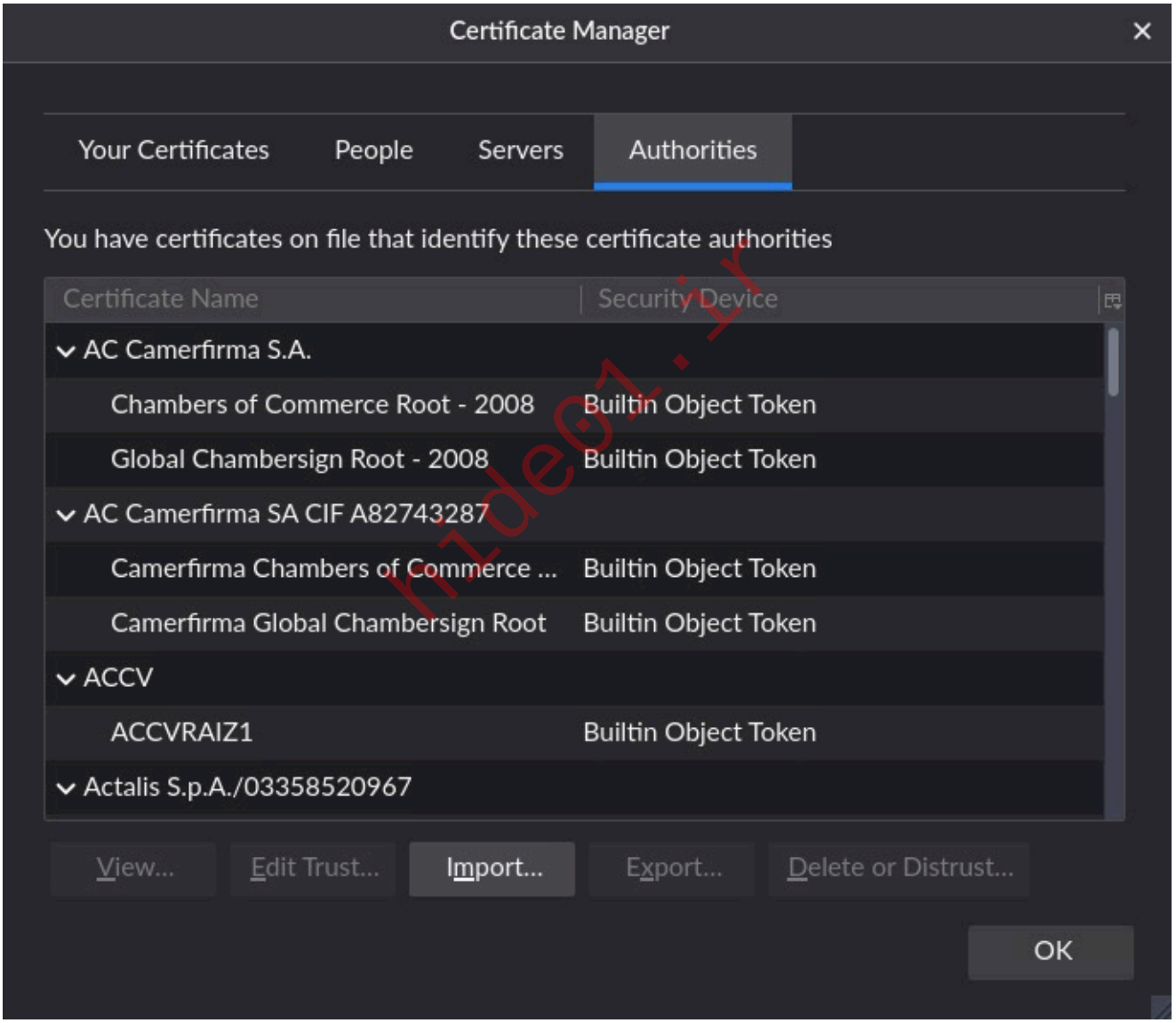


We can also change our certificate by generating a new one with the `Generate` button.

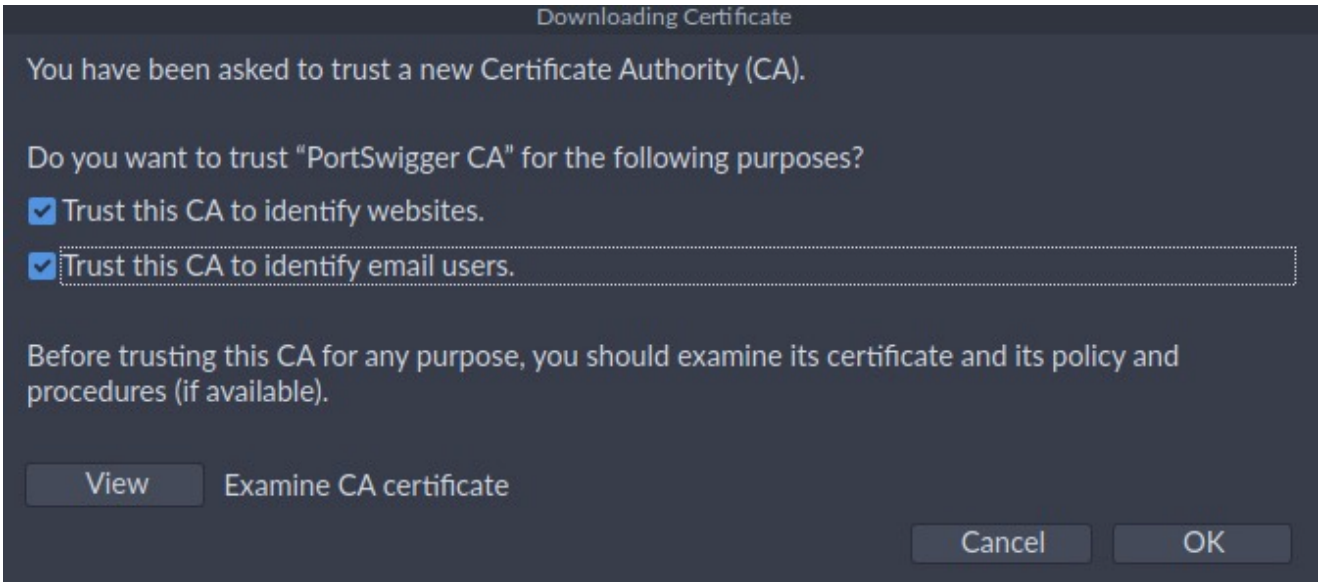
Once we have our certificates, we can install them within Firefox by browsing to about:preferences#privacy, scrolling to the bottom, and clicking `View Certificates`:



After that, we can select the `Authorities` tab, and then click on `import`, and select the downloaded CA certificate:



Finally, we must select `Trust this CA to identify websites` and `Trust this CA to identify email users`, and then click OK:



Once we install the certificate and configure the Firefox proxy, all Firefox web traffic will start routing through our web proxy.

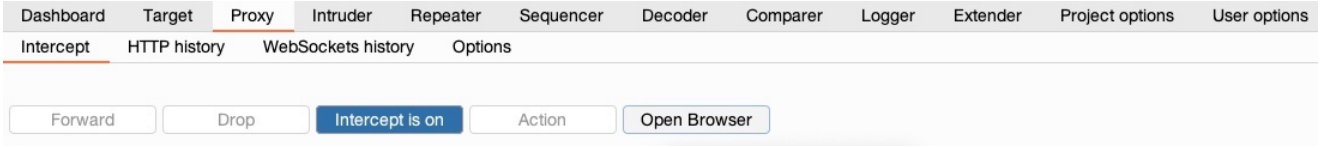
Intercepting Web Requests

Now that we have set up our proxy, we can use it to intercept and manipulate various HTTP requests sent by the web application we are testing. We'll start by learning how to intercept web requests, change them, and then send them through to their intended destination.

Intercepting Requests

Burp

In Burp, we can navigate to the `Proxy` tab, and request interception should be on by default. If we want to turn request interception on or off, we may go to the `Intercept` sub-tab and click on `Intercept is on/off` button to do so:

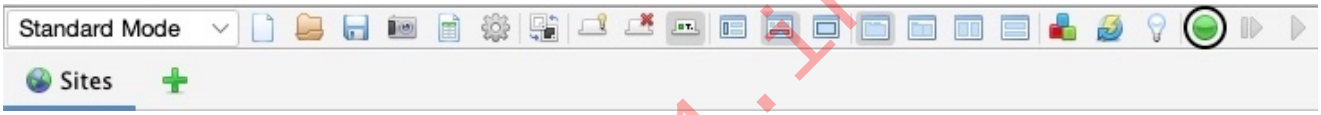


Once we turn request interception on, we can start up the pre-configured browser and then visit our target website after spawning it from the exercise at the end of this section. Then, once we go back to Burp, we will see the intercepted request awaiting our action, and we can click on `forward` to forward the request:

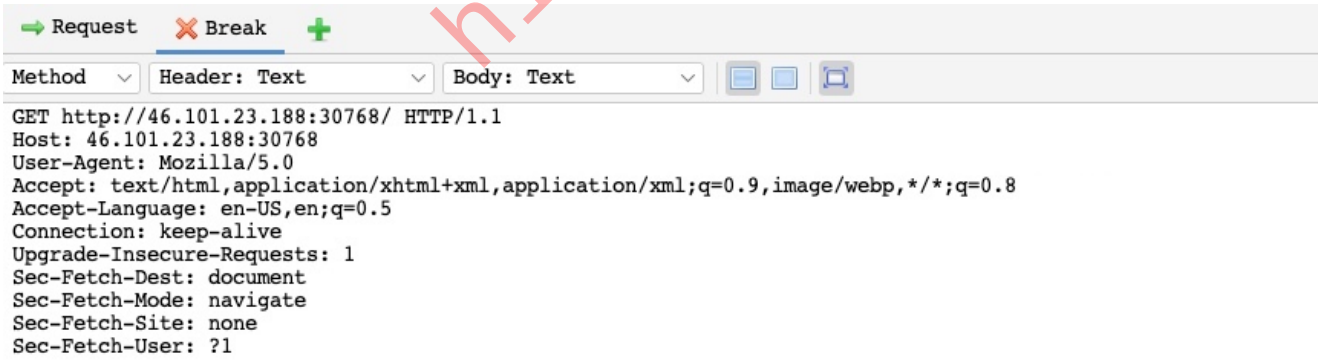
Note: as all Firefox traffic will be intercepted in this case, we may see another request has been intercepted before this one. If this happens, click 'Forward', until we get the request to our target IP, as shown above.

ZAP

In ZAP, interception is off by default, as shown by the green button on the top bar (green indicates that requests can pass and not be intercepted). We can click on this button to turn the Request Interception on or off, or we can use the shortcut [`CTRL+B`] to toggle it on or off:



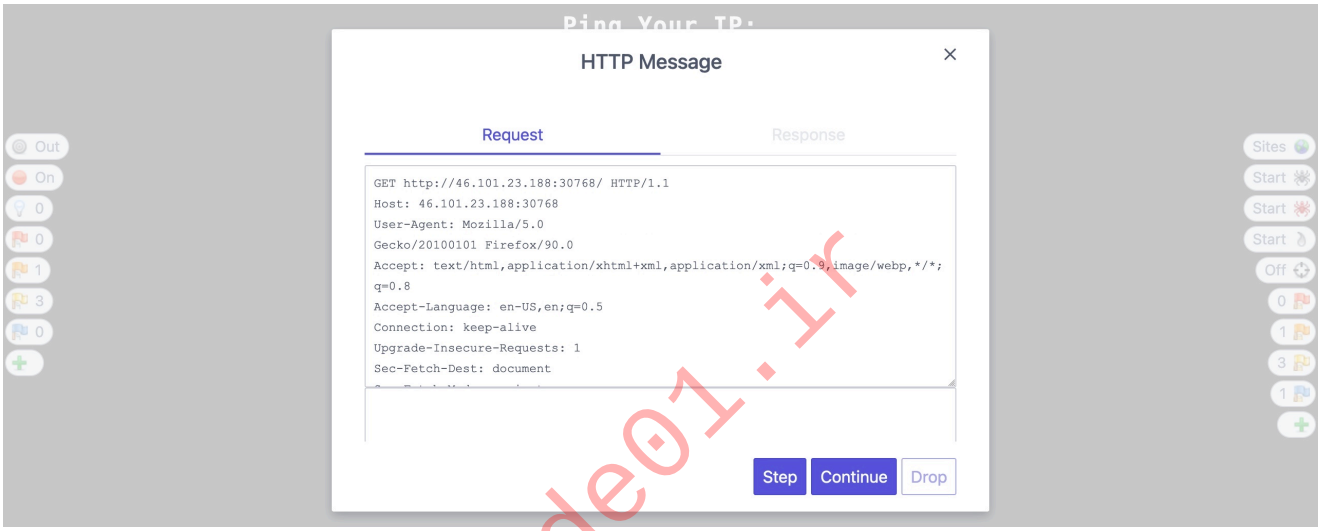
Then, we can start the pre-configured browser and revisit the exercise webpage. We will see the intercepted request in the top-right pane, and we can click on the step (right to the red `break` button) to forward the request:



ZAP also has a powerful feature called `Heads Up Display (HUD)`, which allows us to control most of the main ZAP features from right within the pre-configured browser. We can enable the `HUD` by clicking its button at the end of the top menu bar:

The HUD has many features that we will cover as we go through the module. For intercepting requests, we can click on the second button from the top on the left pane to turn request interception on:

Now, once we refresh the page or send another request, the HUD will intercept the request and will present it to us for action:



We can choose to `step` to send the request and examine its response and break any further requests, or we can choose to `continue` and let the page send the remaining requests. The `step` button is helpful when we want to examine every step of the page's functionality, while `continue` is useful when we are only interested in a single request and can forward the remaining requests once we reach our target request.

Tip: The first time you use the pre-configured ZAP browser you will be presented with the HUD tutorial. You may consider taking this tutorial after this section, as it will teach you the basics of the HUD. Even if you do not grasp everything, the upcoming sections should cover whatever you missed. If you do not get the tutorial, you can click on the configuration button on the bottom right and choose "Take the HUD tutorial".

Manipulating Intercepted Requests

Once we intercept the request, it will remain hanging until we forward it, as we did above. We can examine the request, manipulate it to make any changes we want, and then send it to its destination. This helps us better understand what information a particular web

application is sending in its web requests and how it may respond to any changes we make in that request.

There are numerous applications for this in Web Penetration Testing, such as testing for:

1. SQL injections
2. Command injections
3. Upload bypass
4. Authentication bypass
5. XSS
6. XXE
7. Error handling
8. Deserialization

And many other potential web vulnerabilities, as we will see in other web modules in HTB Academy. So, let's show this with a basic example to demonstrate intercepting and manipulating web requests.

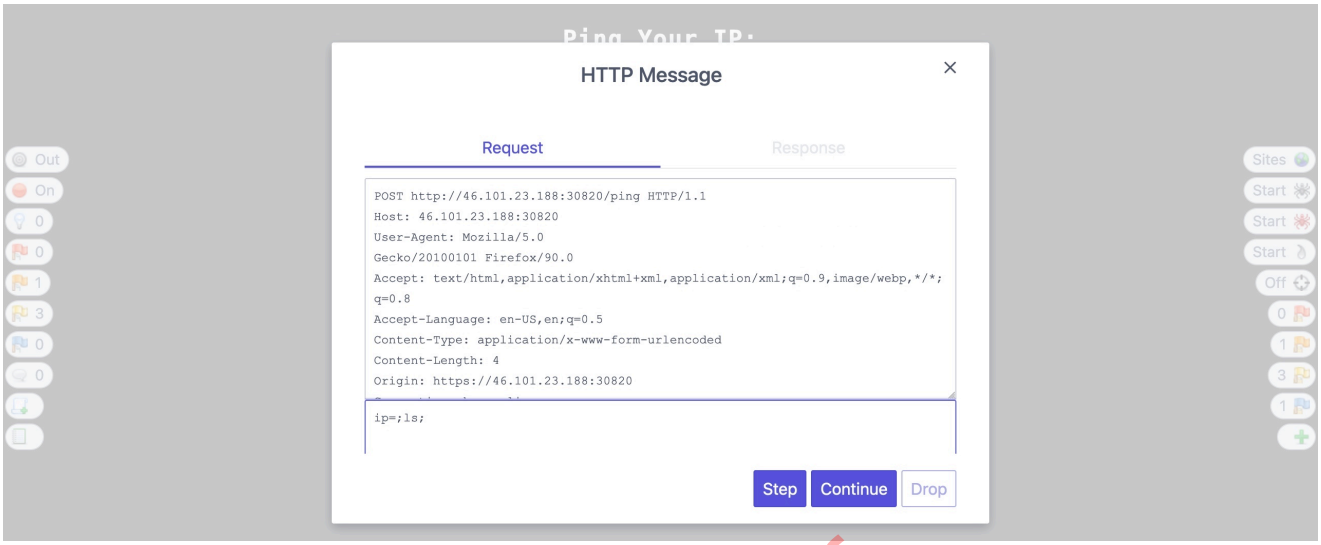
Let us turn request interception back on in the tool of our choosing, set the `IP` value on the page, then click on the `Ping` button. Once our request is intercepted, we should get a similar HTTP request to the following :

```
POST /ping HTTP/1.1
Host: 46.101.23.188:30820
Content-Length: 4
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://46.101.23.188:30820
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
p,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://46.101.23.188:30820/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

ip=1
```

Typically, we can only specify numbers in the `IP` field using the browser, as the web page prevents us from sending any non-numeric characters using front-end JavaScript. However, with the power of intercepting and manipulating HTTP requests, we can try using other characters to "break" the application ("breaking" the request/response flow by manipulating

the target parameter, not damaging the target web application). If the web application does not verify and validate the HTTP requests on the back-end, we may be able to manipulate it and exploit it.

So, let us change the `ip` parameter's value from `1` to `;ls;` and see how the web application handles our input:



Once we click continue/forward, we will see that the response changed from the default ping output to the `ls` output, meaning that we successfully manipulated the request to inject our command:

```
flag.txt
index.html
node_modules
package-lock.json
public
server.js
```


This demonstrates a basic example of how request interception and manipulation can help with testing web applications for various vulnerabilities, which is considered an essential tool to be able to test different web applications effectively.

Note: As previously mentioned, we will not be covering specific web attacks in this module, but rather how Web Proxies can facilitate various types of attacks. Other web modules in HTB Academy cover these types of attacks in depth.

Intercepting Responses

In some instances, we may need to intercept the HTTP responses from the server before they reach the browser. This can be useful when we want to change how a specific web

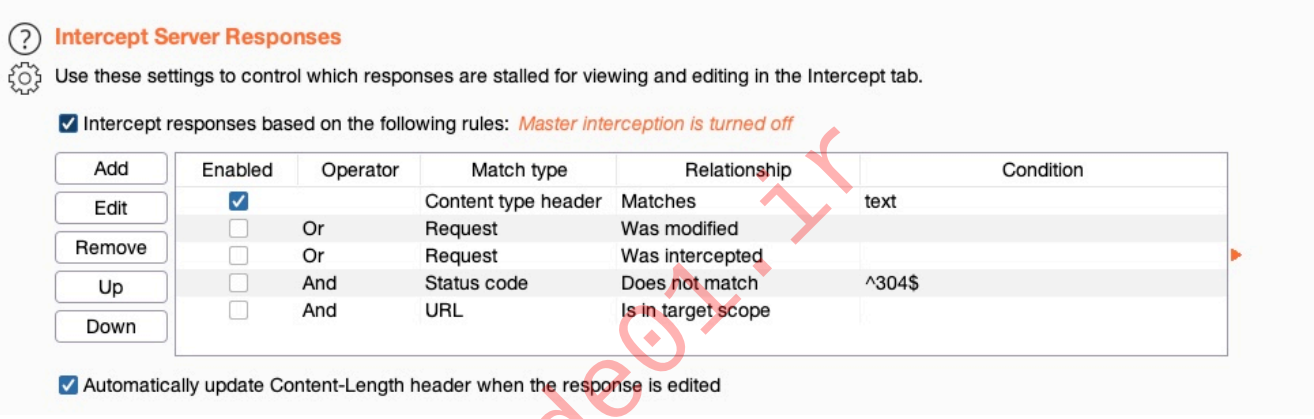
page looks, like enabling certain disabled fields or showing certain hidden fields, which may help us in our penetration testing activities.

So, let's see how we can achieve that with the exercise we tested in the previous section.

In our previous exercise, the `IP` field only allowed us to input numeric values. If we intercept the response before it reaches our browser, we can edit it to accept any value, which would enable us to input the payload we used last time directly.

Burp

In Burp, we can enable response interception by going to (`Proxy>Options`) and enabling `Intercept Response` under `Intercept Server Responses` :



After that, we can enable request interception once more and refresh the page with [`CTRL+SHIFT+R`] in our browser (to force a full refresh). When we go back to Burp, we should see the intercepted request, and we can click on `forward` . Once we forward the request, we'll see our intercepted response:

```
21  <body>
22    <form name='ping' class='form' method='post' id='form1' action='/ping'>
23      <center>
24        <h1>
25          <label for="ip">
              Ping Your IP:
            </label>
26          <center>
              127.0.0.
27            <input type="number" id="ip" name="ip" min="1" max="255" maxlength="3"
28            oninput="javascript: if (this.value.length > this.maxLength) this.value = this.value.slice(0, this.maxLength);"
29            required>
```

Let's try changing the `type="number"` on line 27 to `type="text"` , which should enable us to write any value we want. We will also change the `maxlength="3"` to `maxlength="100"` so we can enter longer input:

```
<input type="text" id="ip" name="ip" min="1" max="255" maxlength="100"
    oninput="javascript: if (this.value.length > this.maxLength)
this.value = this.value.slice(0, this.maxLength);"
```

```
    required>
```

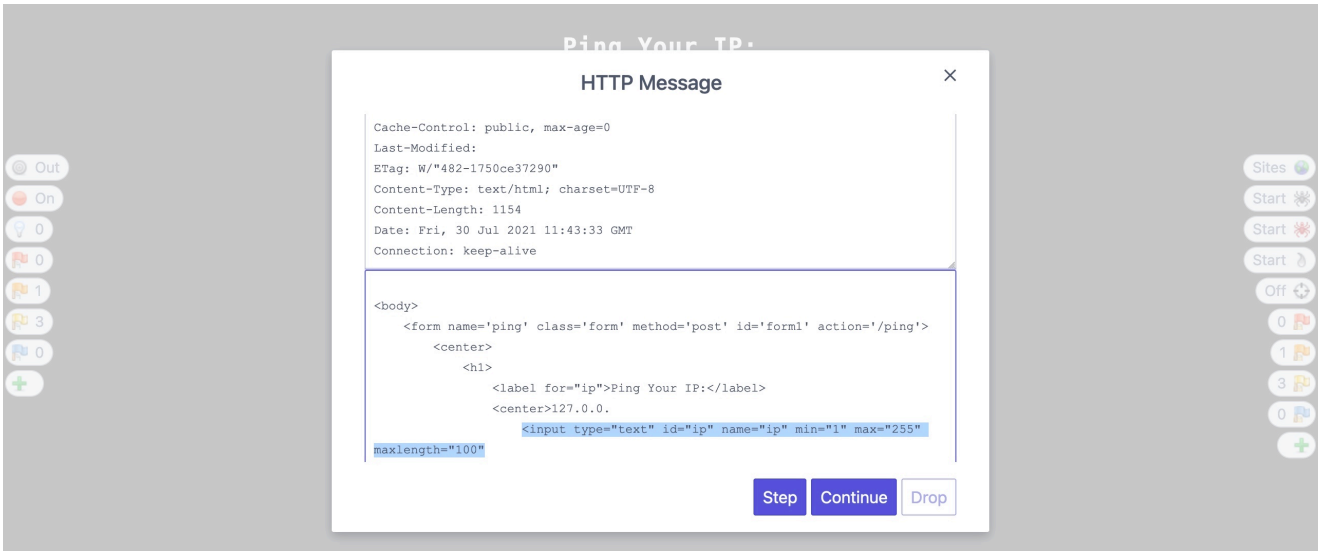
Now, once we click on `forward` again, we can go back to Firefox to examine the edited response:

As we can see, we could change the way the page is rendered by the browser and can now input any value we want. We may use the same technique to persistently enable any disabled HTML buttons by modifying their HTML code.

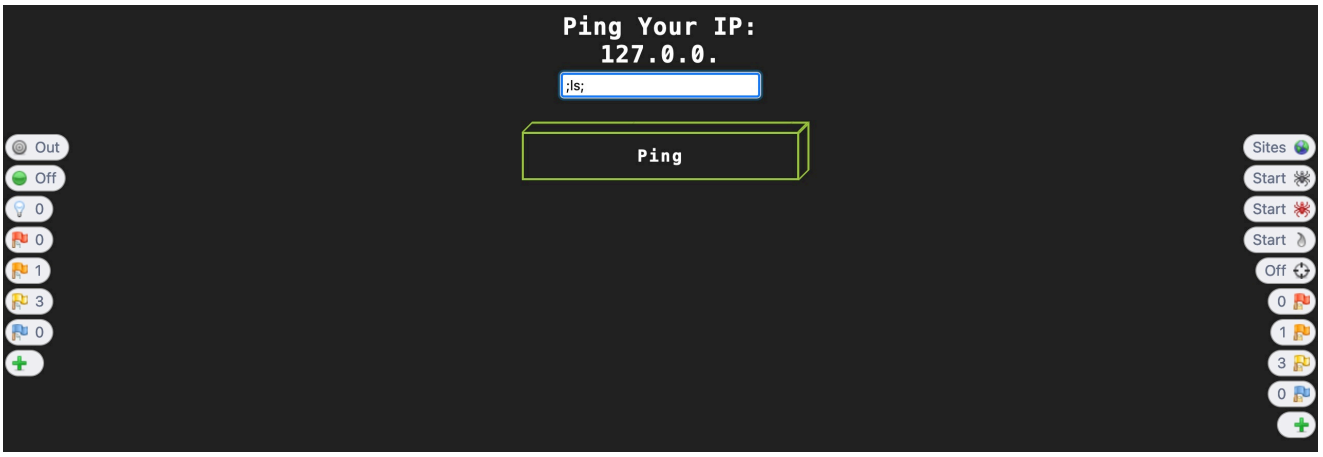
Exercise: Try using the payload we used last time directly within the browser, to test how intercepting responses can make web application penetration testing easier.

ZAP

Let's try to see how we can do the same with ZAP. As we saw in the previous section, when our requests are intercepted by ZAP, we can click on `Step`, and it will send the request and automatically intercept the response:



Once we make the same changes we previously did and click on `Continue`, we will see that we can also use any input value:



However, ZAP HUD also has another powerful feature that can help us in cases like this. While in many instances we may need to intercept the response to make custom changes, if all we wanted was to enable disabled input fields or show hidden input fields, then we can click on the third button on the left (the light bulb icon), and it will enable/show these fields without us having to intercept the response or refresh the page.

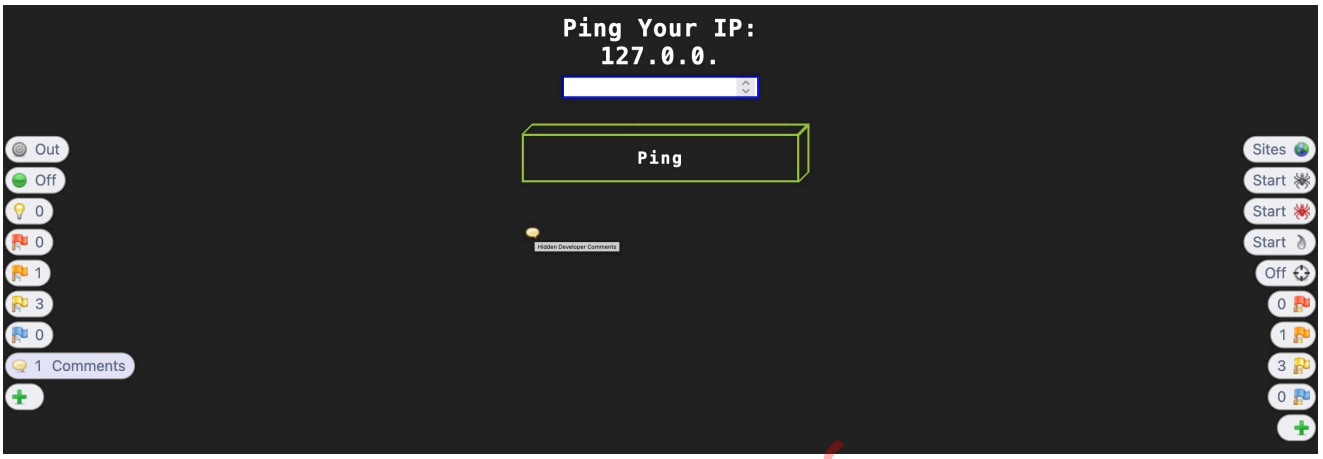
For example, the below web application has the `IP` input field as disabled:

In these cases, we can click on the `Show/Enable` button, and it will enable the button for us, and we can interact with it to add our input:

We can similarly use this feature to show all hidden fields or buttons. `Burp` also has a similar feature, which we can enable under `Proxy>Options>Response Modification`, then

select one of the options, like `Unhide hidden form fields`.

Another similar feature is the `Comments` button, which will indicate the positions where there are HTML comments that are usually only visible in the source code. We can click on the `+` button on the left pane and select `Comments` to add the `Comments` button, and once we click on it, the `Comments` indicators should be shown. For example, the below screenshot shows an indicator for a position that has a comment, and hovering over it with our cursor shows the comment's content:



Being able to modify how the web page looks makes it much easier for us to perform web application penetration testing in certain scenarios instead of having to send our input through an intercepted request. Next, we will see how we can automate this process to modify our changes in the response automatically so we don't have to keep intercepting and manually changing the responses.

Automatic Modification

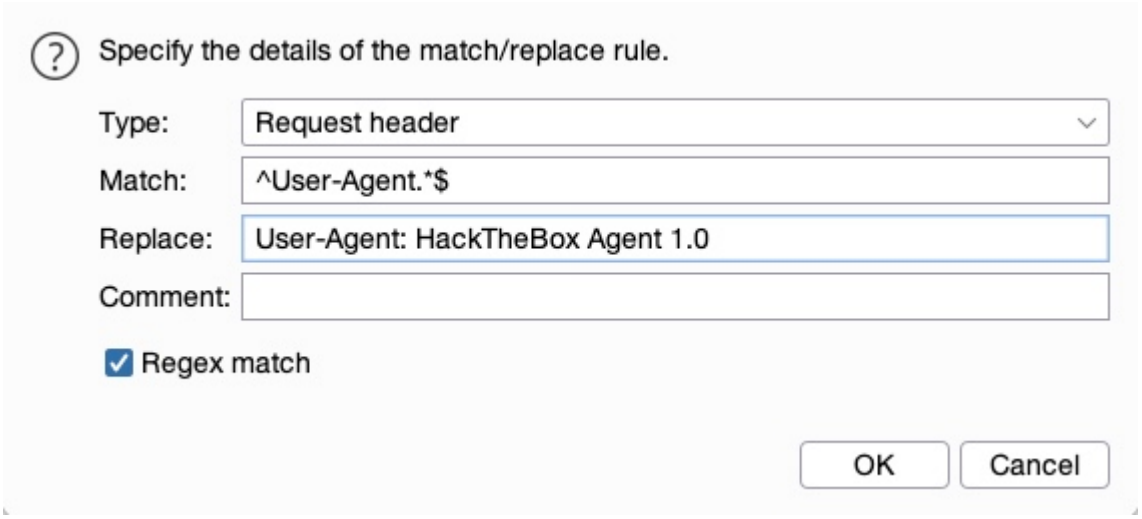
We may want to apply certain modifications to all outgoing HTTP requests or all incoming HTTP responses in certain situations. In these cases, we can utilize automatic modifications based on rules we set, so the web proxy tools will automatically apply them.

Automatic Request Modification

Let us start with an example of automatic request modification. We can choose to match any text within our requests, either in the request header or request body, and then replace them with different text. For the sake of demonstration, let's replace our `User-Agent` with `HackTheBox Agent 1.0`, which may be handy in cases where we may be dealing with filters that block certain User-Agents.

Burp Match and Replace

We can go to (`Proxy>Options>Match and Replace`) and click on `Add` in Burp. As the below screenshot shows, we will set the following options:

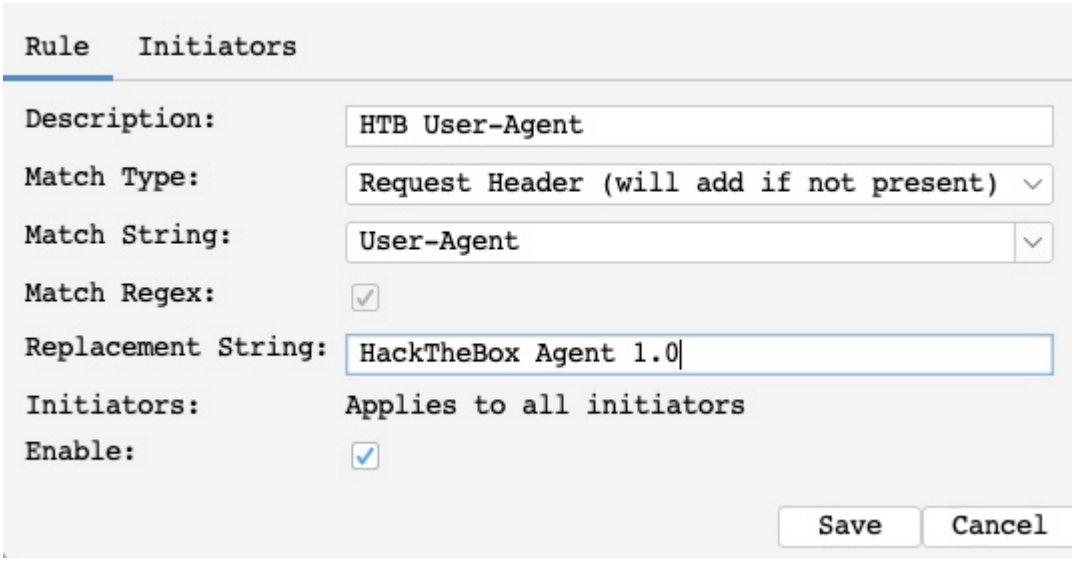


`Type` : `Request header`	Since the change we want to make will be in the request header and not in its body.
`Match` : `^User-Agent.*$`	The regex pattern that matches the entire line with `User-Agent` in it.
`Replace` : `User-Agent: HackTheBox Agent 1.0`	This is the value that will replace the line we matched above.
`Regex match` : True	We don't know the exact User-Agent string we want to replace, so we'll use regex to match any value that matches the pattern we specified above.

Once we enter the above options and click `Ok` , our new Match and Replace option will be added and enabled and will start automatically replacing the `User-Agent` header in our requests with our new User-Agent. We can verify that by visiting any website using the pre-configured Burp browser and reviewing the intercepted request. We will see that our User-Agent has indeed been automatically replaced:

ZAP Replacer

ZAP has a similar feature called `Replacer`, which we can access by pressing [`CTRL+R`] or clicking on `Replacer` in ZAP's options menu. It is fairly similar to what we did above, so we can click on `Add` and add the same options we user earlier:



- `Description`: `HTB User-Agent`.
- `Match Type`: `Request Header (will add if not present)`.
- `Match String`: `User-Agent`. We can select the header we want from the drop-down menu, and ZAP will replace its value.
- `Replacement String`: `HackTheBox Agent 1.0`.
- `Enable`: True.

ZAP also has the `Request Header String` that we can use with a Regex pattern. `Try using this option with the same values we used for Burp to see how it works.`

ZAP also provides the option to set the `Initiators`, which we can access by clicking on the other tab in the windows shown above. Initiators enable us to select where our `Replacer` option will be applied. We will keep the default option of `Apply to all HTTP(S) messages` to apply everywhere.

We can now enable request interception by pressing [`CTRL+B`], then can visit any page in the pre-configured ZAP browser:

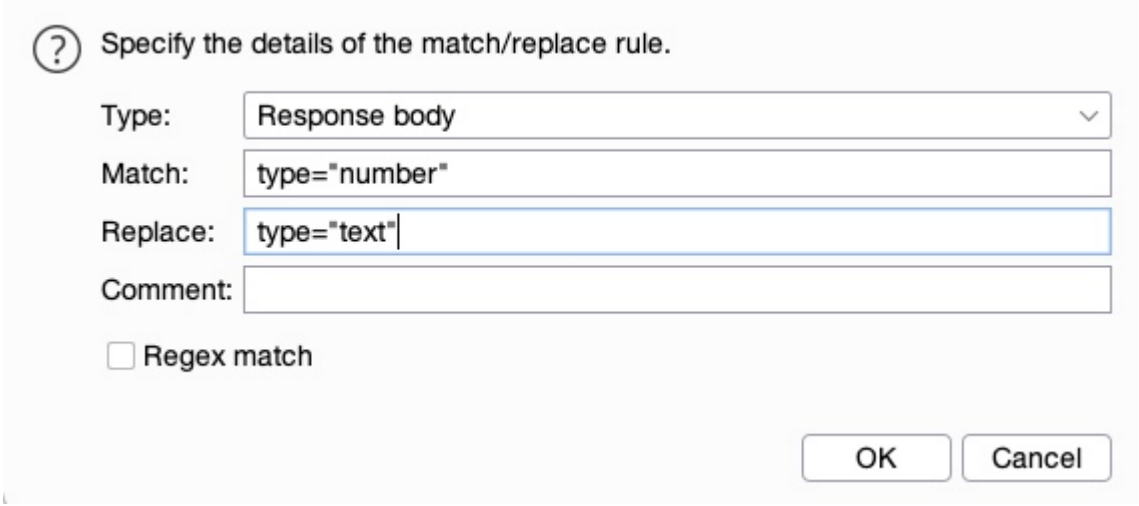
Request Response

```
GET http://46.101.23.188:31342/ HTTP/1.1
Host: 46.101.23.188:31342
User-Agent: HackTheBox Agent 1.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
```

Automatic Response Modification

The same concept can be used with HTTP responses as well. In the previous section, you may have noticed when we intercepted the response that the modifications we made to the `IP` field were temporary and were not applied when we refreshed the page unless we intercepted the response and added them again. To solve this, we can automate response modification similarly to what we did above to automatically enable any characters in the `IP` field for easier command injection.

Let us go back to (`Proxy>Options>Match and Replace`) in Burp to add another rule. This time we will use the type of `Response body` since the change we want to make exists in the response's body and not in its headers. In this case, we do not have to use regex as we know the exact string we want to replace, though it is possible to use regex to do the same thing if we prefer.

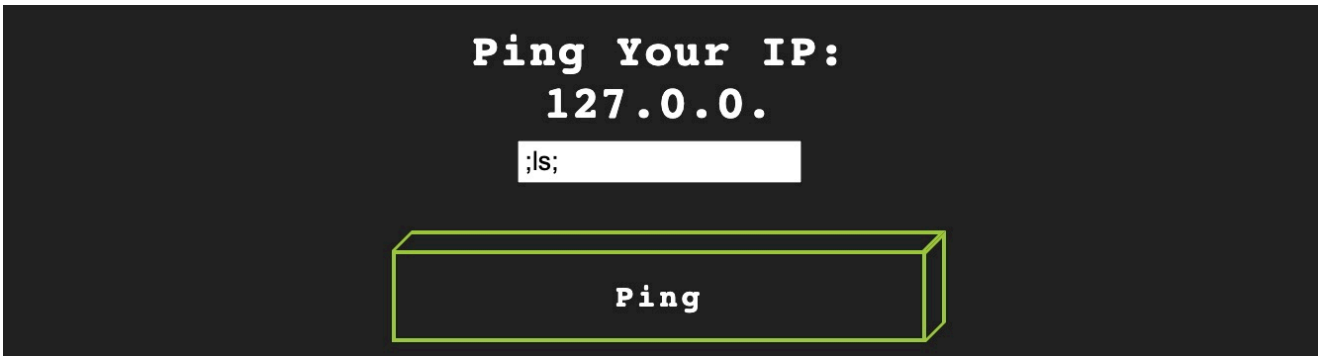


- `Type` : `Response body` .
- `Match` : `type="number"` .

- `Replace`: `type="text"`.
- `Regex match`: False.

Try adding another rule to change `maxlength="3"` to `maxlength="100"`.

Now, once we refresh the page with [`CTRL+SHIFT+R`], we'll see that we can add any input to the input field, and this should persist between page refreshes as well:



We can now click on `Ping`, and our command injection should work without intercepting and modifying the request.

Exercise 1: Try applying the same rules with ZAP Replacer. You can click on the tab below to show the correct options.

Click to show the answer

Change input type to text:

- `Match Type`: `Response Body String`.

- `Match Regex`: `False`.

- `Match String`: `type="number"`.

- `Replacement String`: `type="text"`.

- `Enable`: `True`.

Change max length to 100:

- `Match Type`: `Response Body String`.

- `Match Regex`: `False`.

- `Match String`: `maxlength="3"`.

- `Replacement String`: `maxlength="100"`.

- `Enable`: `True`.

Exercise 2: Try adding a rule that automatically adds `;ls;` when we click on `Ping`, by matching and replace the request body of the `Ping` request.

Repeating Requests

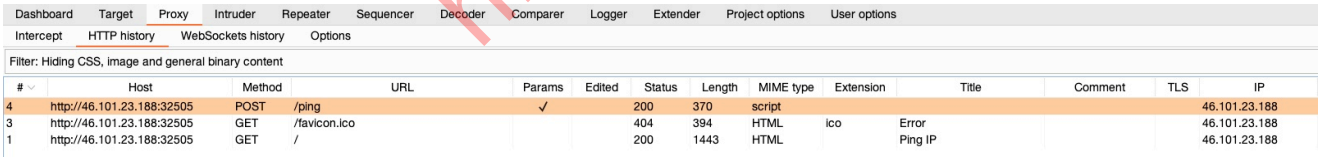
In the previous sections, we successfully bypassed the input validation to use a non-numeric input to reach command injection on the remote server. If we want to repeat the same process with a different command, we would have to intercept the request again, provide a different payload, forward it again, and finally check our browser to get the final result.

As you can imagine, if we would do this for each command, it would take us forever to enumerate a system, as each command would require 5-6 steps to get executed. However, for such repetitive tasks, we can utilize request repeating to make this process significantly easier.

Request repeating allows us to resend any web request that has previously gone through the web proxy. This allows us to make quick changes to any request before we send it, then get the response within our tools without intercepting and modifying each request.

Proxy History

To start, we can view the HTTP requests history in `Burp` at (`Proxy>HTTP History`):



In `ZAP` HUD, we can find it in the bottom History pane or ZAP's main UI at the bottom `History` tab as well:

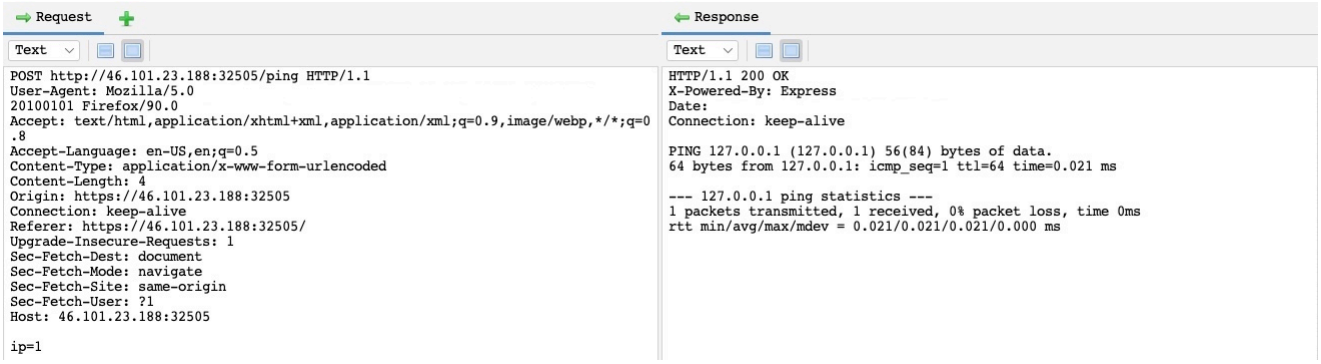
Both tools also provide filtering and sorting options for requests history, which may be helpful if we deal with a huge number of requests and want to locate a specific request. `Try to see how filters work on both tools.`

Note: Both tools also maintain WebSockets history, which shows all connections initiated by the web application even after being loaded, like asynchronous updates and data fetching. WebSockets can be useful when performing advanced web penetration testing, and are out of the scope of this module.

If we click on any request in the history in either tool, its details will be shown:

`Burp`:

`ZAP`:



Tip: While ZAP only shows the final/modified request that was sent, Burp provides the ability to examine both the original request and the modified request. If a request was edited, the pane header would say `Original Request`, and we can click on it and select `Edited Request` to examine the final request that was sent.

Repeating Requests

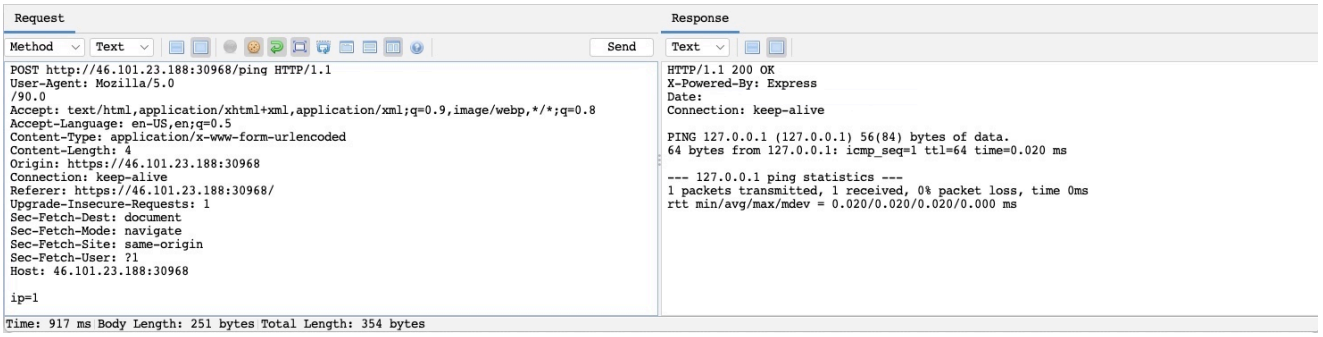
Burp

Once we locate the request we want to repeat, we can click [`CTRL+R`] in Burp to send it to the `Repeater` tab, and then we can either navigate to the `Repeater` tab or click [`CTRL+SHIFT+R`] to go to it directly. Once in `Repeater`, we can click on `Send` to send the request:

Tip: We can also right-click on the request and select `Change Request Method` to change the HTTP method between POST/GET without having to rewrite the entire request.

ZAP

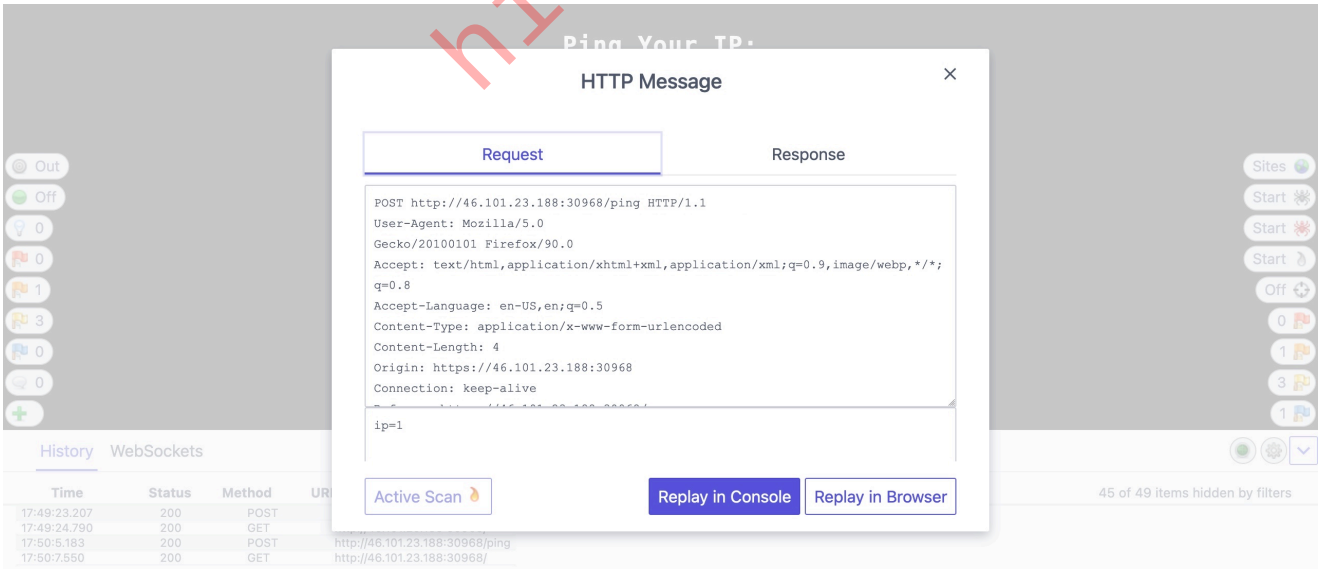
In ZAP, once we locate our request, we can right-click on it and select `Open/Resend with Request Editor`, which would open the request editor window, and allow us to resend the request with the `Send` button to send our request:



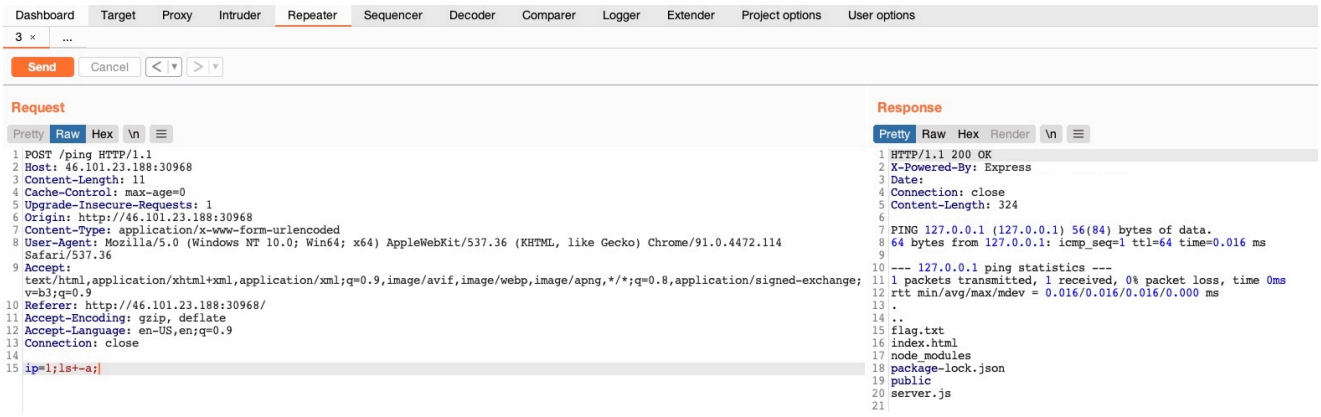
We can also see the `Method` drop-down menu, allowing us to quickly switch the request method to any other HTTP method.

Tip: By default, the Request Editor window in ZAP has the Request/Response in different tabs. You can click on the display buttons to change how they are organized. To match the above look choose the same display options shown in the screenshot.

We can achieve the same result within the pre-configured browser with `ZAP HUD`. We can locate the request in the bottom History pane, and once we click on it, the `Request Editor` window will show, allowing us to resend it. We can select `Replay in Console` to get the response in the same `HUD` window, or select `Replay in Browser` to see the response rendered in the browser:



So, let us try to modify our request and send it. In all three options (`Burp Repeater`, `ZAP Request Editor`, and `ZAP HUD`), we see that the requests are modifiable, and we can select the text we want to change and replace it with whatever we want, and then click the `Send` button to send it again:



As we can see, we could easily modify the command and instantly get its output by using Burp `Repeater`. Try doing the same in `ZAP Request Editor` and `ZAP HUD` to see how they work.

Finally, we can see in our previous POST request that the data is URL-encoded. This is an essential part of sending custom HTTP requests, which we will discuss in the next section.

Encoding/Decoding

As we modify and send custom HTTP requests, we may have to perform various types of encoding and decoding to interact with the webserver properly. Both tools have built-in encoders that can help us in quickly encoding and decoding various types of text.

URL Encoding

It is essential to ensure that our request data is URL-encoded and our request headers are correctly set. Otherwise, we may get a server error in the response. This is why encoding and decoding data becomes essential as we modify and repeat web requests. Some of the key characters we need to encode are:

- `Spaces`: May indicate the end of request data if not encoded
- `&`: Otherwise interpreted as a parameter delimiter
- `#`: Otherwise interpreted as a fragment identifier

To URL-encode text in Burp Repeater, we can select that text and right-click on it, then select (`Convert Selection>URL>URL encode key characters`), or by selecting the text and clicking [`CTRL+U`]. Burp also supports URL-encoding as we type if we right-click and enable that option, which will encode all of our text as we type it. On the other hand, ZAP should automatically URL-encode all of our request data in the background before sending the request, though we may not see that explicitly.

There are other types of URL-encoding, like `Full URL-Encoding` or `Unicode URL` encoding, which may also be helpful for requests with many special characters.

Decoding

While URL-encoding is key to HTTP requests, it is not the only type of encoding we will encounter. It is very common for web applications to encode their data, so we should be able to quickly decode that data to examine the original text. On the other hand, back-end servers may expect data to be encoded in a particular format or with a specific encoder, so we need to be able to quickly encode our data before we send it.

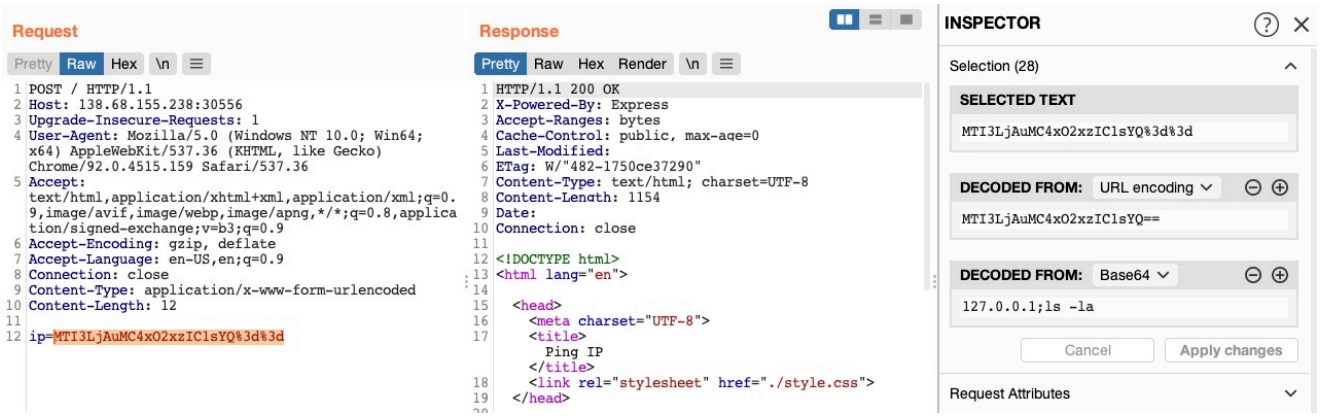
The following are some of the other types of encoders supported by both tools:

- HTML
- Unicode
- Base64
- ASCII hex

To access the full encoder in Burp, we can go to the `Decoder` tab. In ZAP, we can use the `Encoder/Decoder/Hash` by clicking [`CTRL+E`]. With these encoders, we can input any text and have it quickly encoded or decoded. For example, perhaps we came across the following cookie that is base64 encoded, and we need to decode it:
`eyJ1c2VybmFtZSI6Imd1ZXN0IiwgImlzX2FkbWluIjpmYWxzZX0=`

We can input the above string in Burp Decoder and select `Decode as > Base64`, and we'll get the decoded value:

In recent versions of Burp, we can also use the `Burp Inspector` tool to perform encoding and decoding (among other things), which can be found in various places like `Burp Proxy` or `Burp Repeater`:



In ZAP, we can use the `Encoder/Decoder/Hash` tool, which will automatically decode strings using various decoders in the `Decode` tab:

Tip: We can create customized tabs in ZAP's Encoder/Decoder/Hash with the "Add New Tab" button, and then we can add any type of encoder/decoder we want the text to be shown in. Try to create your own tab with a few encoders/decoders.

Encoding

As we can see, the text holds the value `{"username":"guest", "is_admin":false}`. So, if we were performing a penetration test on a web application and find that the cookie holds this value, we may want to test modifying it to see whether it changes our user privileges. So, we can copy the above value, change `guest` to `admin` and `false` to `true`, and try to encode it again using its original encoding method (`base64`):

Tip: Burp Decoder output can be directly encoded/decoded with a different encoder. Select the new encoder method in the output pane at the bottom, and it will be encoded/decoded again. In ZAP, we can copy the output text and paste it in the input field above.

We can then copy the base64 encoded string and use it with our request in Burp `Repeater` or ZAP `Request Editor`. The same concept can be used to encode and decode various types of encoded text to perform effective web penetration testing without utilizing other tools to do the encoding.

Proxying Tools

An important aspect of using web proxies is enabling the interception of web requests made by command-line tools and thick client applications. This gives us transparency into the web requests made by these applications and allows us to utilize all of the different proxy features we have used with web applications.

To route all web requests made by a specific tool through our web proxy tools, we have to set them up as the tool's proxy (i.e. `http://127.0.0.1:8080`), similarly to what we did with our browsers. Each tool may have a different method for setting its proxy, so we may have to investigate how to do so for each one.

This section will cover a few examples of how to use web proxies to intercept web requests made by such tools. You may use either Burp or ZAP, as the setup process is the same.

Note: Proxying tools usually slows them down, therefore, only proxy tools when you need to investigate their requests, and not for normal usage.

Proxychains

One very useful tool in Linux is proxychains, which routes all traffic coming from any command-line tool to any proxy we specify. `Proxychains` adds a proxy to any command-line tool and is hence the simplest and easiest method to route web traffic of command-line tools through our web proxies.

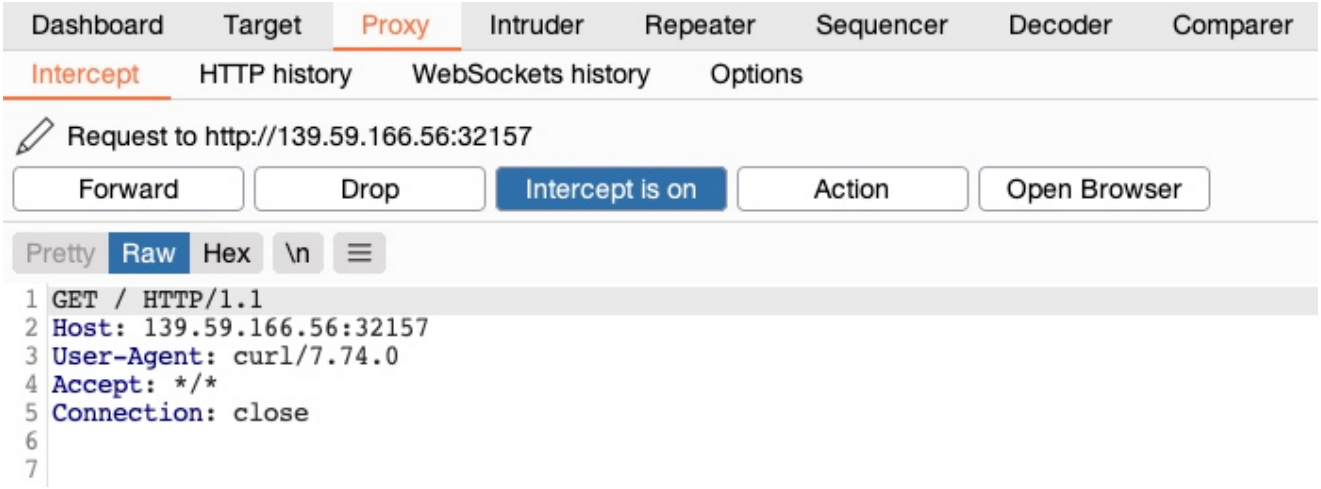
To use `proxychains`, we first have to edit `/etc/proxychains.conf`, comment out the final line and add the following line at the end of it:

```
#socks4         127.0.0.1 9050
http 127.0.0.1 8080
```

We should also enable `Quiet Mode` to reduce noise by un-commenting `quiet_mode`. Once that's done, we can prepend `proxychains` to any command, and the traffic of that command should be routed through `proxychains` (i.e., our web proxy). For example, let's try using `cURL` on one of our previous exercises:

```
proxychains curl http://SERVER_IP:PORT

ProxyChains-3.1 (http://proxychains.sf.net)
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Ping IP</title>
    <link rel="stylesheet" href="./style.css">
</head>
...SNIP...
</html>
```


We see that it worked just as it normally would, with the additional `ProxyChains-3.1` line at the beginning, to note that it is being routed through `ProxyChains`. If we go back to our web proxy (Burp in this case), we will see that the request has indeed gone through it:



Nmap

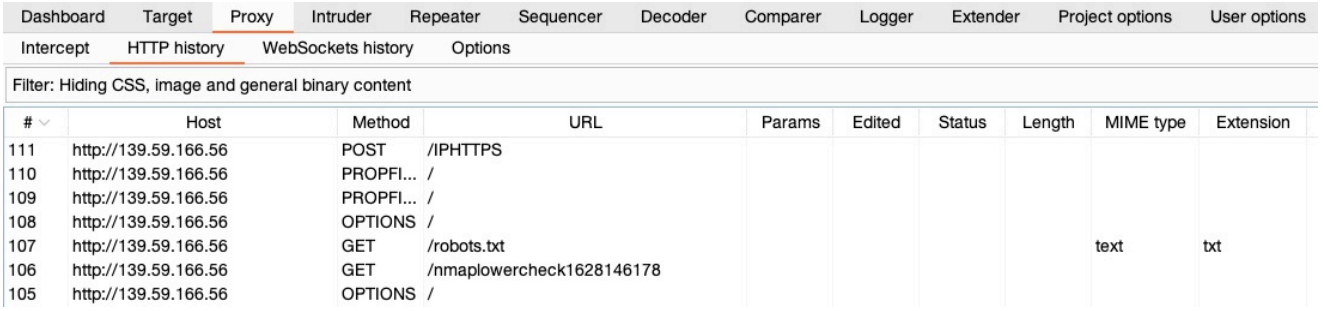
Next, let's try to proxy `nmap` through our web proxy. To find out how to use the proxy configurations for any tool, we can view its manual with `man nmap`, or its help page with `nmap -h`:

```
nmap -h | grep -i prox

  --proxies <url1,[url2],...>: Relay connections through HTTP/SOCKS4
proxies
```

As we can see, we can use the `--proxies` flag. We should also add the `-Pn` flag to skip host discovery (as recommended on the man page). Finally, we'll also use the `-sC` flag to examine what an nmap script scan does:

```
nmap --proxies http://127.0.0.1:8080 SERVER_IP -pPORT -Pn -sC

Starting Nmap 7.91 ( https://nmap.org )
Nmap scan report for SERVER_IP
Host is up (0.11s latency).

PORT      STATE SERVICE
PORT/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.49 seconds
```


Once again, if we go to our web proxy tool, we will see all of the requests made by nmap in the proxy history:



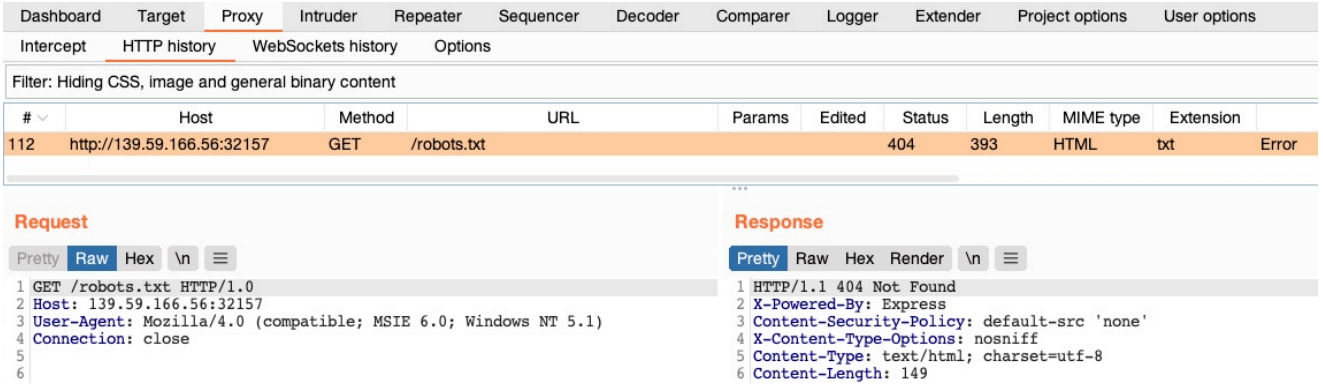
Note: Nmap's built-in proxy is still in its experimental phase, as mentioned by its manual (`man nmap`), so not all functions or traffic may be routed through the proxy. In these cases, we can simply resort to `proxychains`, as we did earlier.

Metasploit

Finally, let's try to proxy web traffic made by Metasploit modules to better investigate and debug them. We should begin by starting Metasploit with `msfconsole`. Then, to set a proxy for any exploit within Metasploit, we can use the `set PROXIES` flag. Let's try the `robots_txt` scanner as an example and run it against one of our previous exercises:

```
msfconsole

msf6 > use auxiliary/scanner/http/robots_txt
msf6 auxiliary(scanner/http/robots_txt) > set PROXIES HTTP:127.0.0.1:8080

PROXIES => HTTP:127.0.0.1:8080

msf6 auxiliary(scanner/http/robots_txt) > set RHOST SERVER_IP

RHOST => SERVER_IP

msf6 auxiliary(scanner/http/robots_txt) > set RPORT PORT

RPORT => PORT

msf6 auxiliary(scanner/http/robots_txt) > run

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```


Once again, we can go back to our web proxy tool of choice and examine the proxy history to view all sent requests:



We see that the request has indeed gone through our web proxy. The same method can be used with other scanners, exploits, and other features in Metasploit.

We can similarly use our web proxies with other tools and applications, including scripts and thick clients. All we have to do is set the proxy of each tool to use our web proxy. This allows us to examine exactly what these tools are sending and receiving and potentially repeat and modify their requests while performing web application penetration testing.

Burp Intruder

Both Burp and ZAP provide additional features other than the default web proxy, which are essential for web application penetration testing. Two of the most important extra features are `web fuzzers` and `web scanners`. The built-in web fuzzers are powerful tools that act as web fuzzing, enumeration, and brute-forcing tools. This may also act as an alternative for many of the CLI-based fuzzers we use, like `ffuf`, `dirbuster`, `gobuster`, `wfuzz`, among others.

Burp's web fuzzer is called `Burp Intruder`, and can be used to fuzz pages, directories, sub-domains, parameters, parameters values, and many other things. Though it is much more advanced than most CLI-based web fuzzing tools, the free `Burp Community` version is throttled at a speed of 1 request per second, making it extremely slow compared to CLI-based web fuzzing tools, which can usually read up to 10k requests per second. This is why we would only use the free version of Burp Intruder for short queries. The `Pro` version has unlimited speed, which can rival common web fuzzing tools, in addition to the very useful features of Burp Intruder. This makes it one of the best web fuzzing and brute-forcing tools.

In this section, we will demonstrate the various uses of Burp Intruder for web fuzzing and enumeration.

Target

As usual, we'll start up Burp and its pre-configured browser and then visit the web application from the exercise at the end of this section. Once we do, we can go to the Proxy History, locate our request, then right-click on the request and select `Send to Intruder`, or use the shortcut [`CTRL+I`] to send it to `Intruder`.

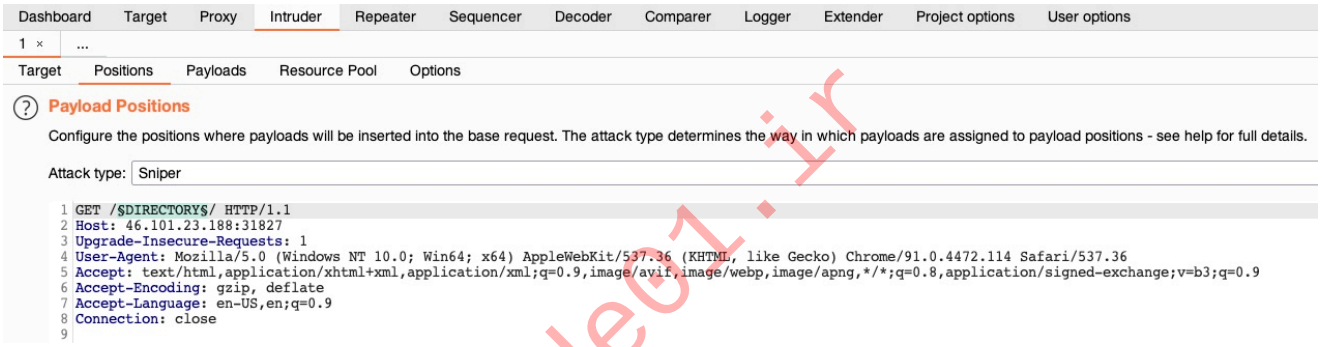
We can then go to `Intruder` by clicking on its tab or with the shortcut [`CTRL+SHIFT+I`], which takes us right to `Burp Intruder`:

On the first tab, '`Target`', we see the details of the target we will be fuzzing, which is fed from the request we sent to `Intruder`.

Positions

The second tab, '`Positions`', is where we place the payload position pointer, which is the point where words from our wordlist will be placed and iterated over. We will be demonstrating how to fuzz web directories, which is similar to what's done by tools like `ffuf` or `gobuster`.

To check whether a web directory exists, our fuzzing should be in '`GET /DIRECTORY/`', such that existing pages would return `200 OK`, otherwise we'd get `404 NOT FOUND`. So, we will need to select `DIRECTORY` as the payload position, by either wrapping it with `§` or by selecting the word `DIRECTORY` and clicking on the `Add §` button:



Tip: the `DIRECTORY` in this case is the pointer's name, which can be anything, and can be used to refer to each pointer, in case we are using more than position with different wordlists for each.

The final thing to select in the target tab is the `Attack Type`. The attack type defines how many payload pointers are used and determines which payload is assigned to which position. For simplicity, we'll stick to the first type, `Sniper`, which uses only one position. Try clicking on the `?` at the top of the window to read more about attack types, or check out this link.

Note: Be sure to leave the extra two lines at the end of the request, otherwise we may get an error response from the server.

Payloads

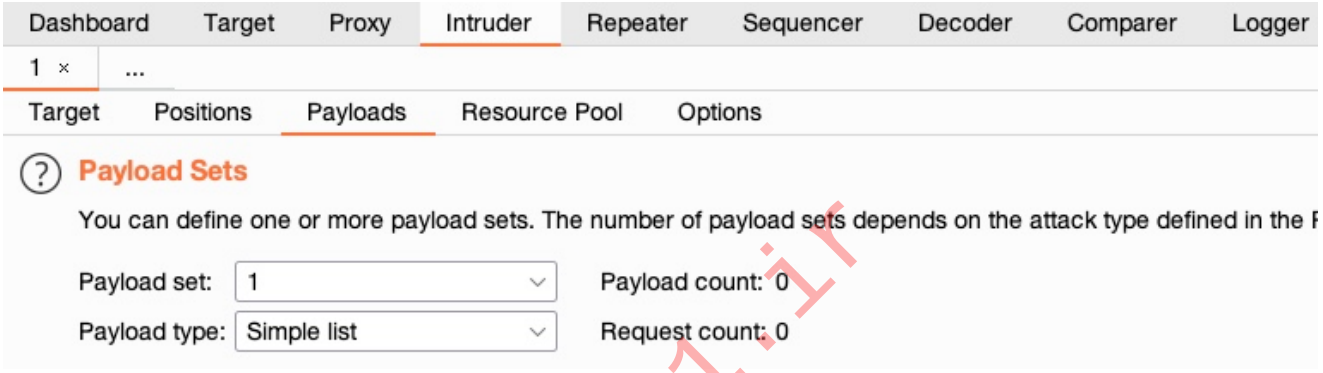
On the third tab, '`Payloads`', we get to choose and customize our payloads/wordlists. This payload/wordlist is what would be iterated over, and each element/line of it would be placed

and tested one by one in the Payload Position we chose earlier. There are four main things we need to configure:

- Payload Sets
- Payload Options
- Payload Processing
- Payload Encoding

Payload Sets

The first thing we must configure is the `Payload Set`. The payload set identifies the Payload number, depending on the attack type and number of Payloads we used in the Payload Position Pointers:



In this case, we only have one Payload Set, as we chose the '`Sniper`' Attack type with only one payload position. If we have chosen the '`Cluster Bomb`' attack type, for example, and added several payload positions, we would get more payload sets to choose from and choose different options for each. In our case, we'll select `1` for the payload set.

Next, we need to select the `Payload Type`, which is the type of payloads/wordlists we will be using. Burp provides a variety of Payload Types, each of which acts in a certain way. For example:

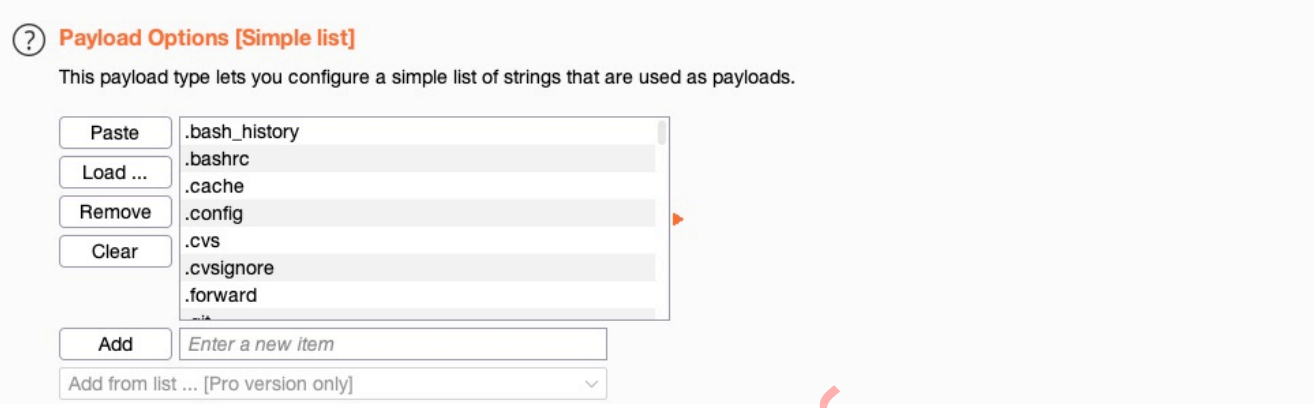
- `Simple List`: The basic and most fundamental type. We provide a wordlist, and Intruder iterates over each line in it.
- `Runtime file`: Similar to `Simple List`, but loads line-by-line as the scan runs to avoid excessive memory usage by Burp.
- `Character Substitution`: Lets us specify a list of characters and their replacements, and Burp Intruder tries all potential permutations.

There are many other Payload Types, each with its own options, and many of which can build custom wordlists for each attack. Try clicking on the `?` next to `Payload Sets`, and then click on `Payload Type`, to learn more about each Payload Type. In our case, we'll be going with a basic `Simple List`.

Payload Options

Next, we must specify the Payload Options, which is different for each Payload Type we select in `Payload Sets`. For a `Simple List`, we have to create or load a wordlist. To do so, we can input each item manually by clicking `Add`, which would build our wordlist on the fly. The other more common option is to click on `Load`, and then select a file to load into Burp Intruder.

We will select `/opt/useful/seclists/Discovery/Web-Content/common.txt` as our wordlist. We can see that Burp Intruder loads all lines of our wordlist into the Payload Options table:



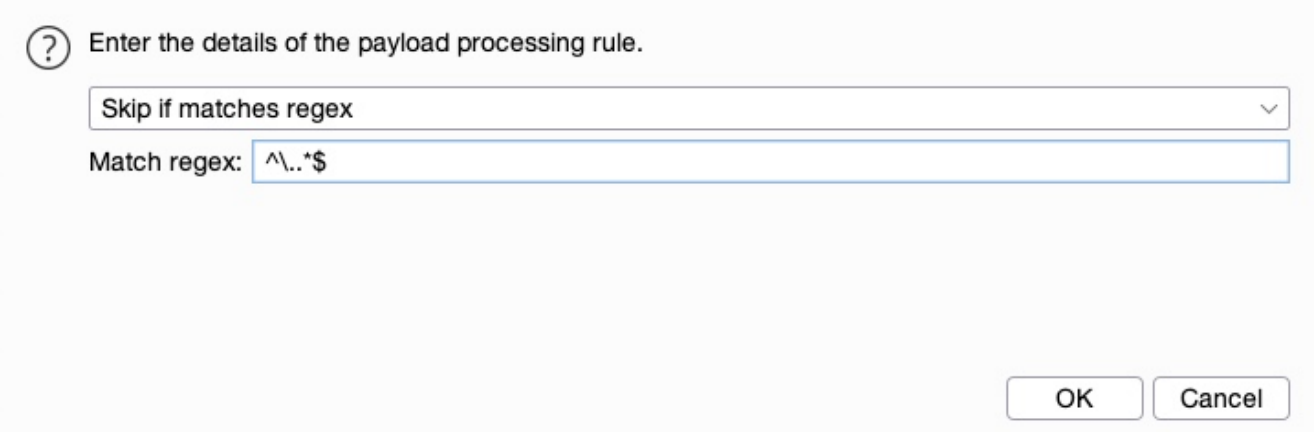
We can add another wordlist or manually add a few items, and they would be appended to the same list of items. We can use this to combine multiple wordlists or create customized wordlists. In Burp Pro, we also can select from a list of existing wordlists contained within Burp by choosing from the `Add from list` menu option.

Tip: In case you wanted to use a very large wordlist, it's best to use `Runtime file` as the Payload Type instead of `Simple List`, so that Burp Intruder won't have to load the entire wordlist in advance, which may throttle memory usage.

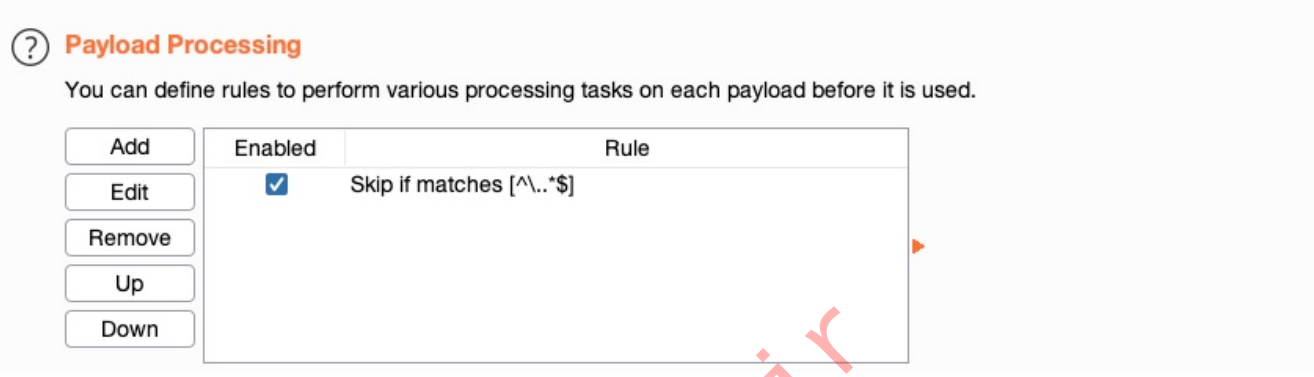
Payload Processing

Another option we can apply is `Payload Processing`, which allows us to determine fuzzing rules over the loaded wordlist. For example, if we wanted to add an extension after our payload item, or if we wanted to filter the wordlist based on specific criteria, we can do so with payload processing.

Let's try adding a rule that skips any lines that start with a `.` (as shown in the wordlist screenshot earlier). We can do that by clicking on the `Add` button and then selecting `Skip if matches regex`, which allows us to provide a regex pattern for items we want to skip. Then, we can provide a regex pattern that matches lines starting with `.`, which is: `^\..*$`:

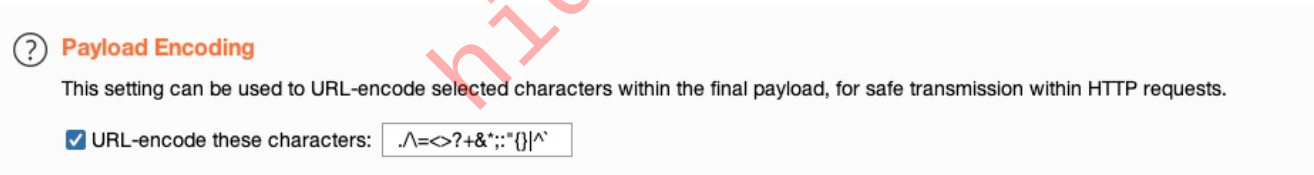


We can see that our rule gets added and enabled:



Payload Encoding

The fourth and final option we can apply is `Payload Encoding`, enabling us to enable or disable Payload URL-encoding.



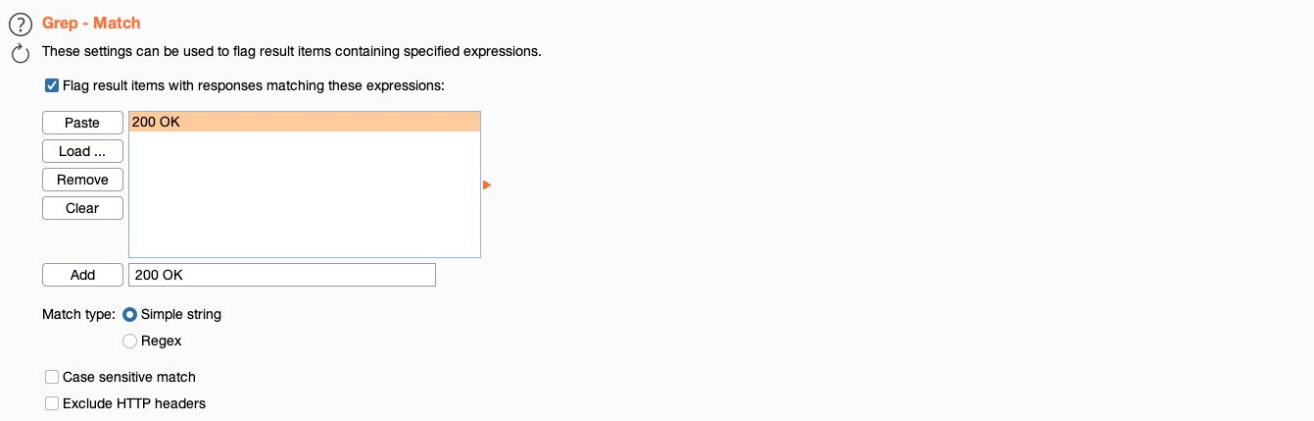
We'll leave it enabled.

Options

Finally, we can customize our attack options from the `Options` tab. There are many options we can customize (or leave at default) for our attack. For example, we can set the number of `retried on failure` and `pause before retry` to 0.

Another useful option is the `Grep - Match`, which enables us to flag specific requests depending on their responses. As we are fuzzing web directories, we are only interested in responses with HTTP code `200 OK`. So, we'll first enable it and then click `Clear` to clear the current list. After that, we can type `200 OK` to match any requests with this string and click

`Add` to add the new rule. Finally, we'll also disable `Exclude HTTP Headers`, as what we are looking for is in the HTTP header:



We may also utilize the `Grep - Extract` option, which is useful if the HTTP responses are lengthy, and we're only interested in a certain part of the response. So, this helps us in only showing a specific part of the response. We are only looking for responses with HTTP Code `200 OK`, regardless of their content, so we will not opt for this option.

Try other `Intruder` options, and use Burp help by clicking on `?` next to each one to learn more about each option.

Note: We may also use the `Resource Pool` tab to specify how much network resources Intruder will use, which may be useful for very large attacks. For our example, we'll leave it at its default values.

Attack

Now that everything is properly set up, we can click on the `Start Attack` button and wait for our attack to finish. Once again, in the free `Community Version`, these attacks would be very slow and take a considerable amount of time for longer wordlists.

The first thing we will notice is that all lines starting with `.` were skipped, and we directly started with the lines after them:

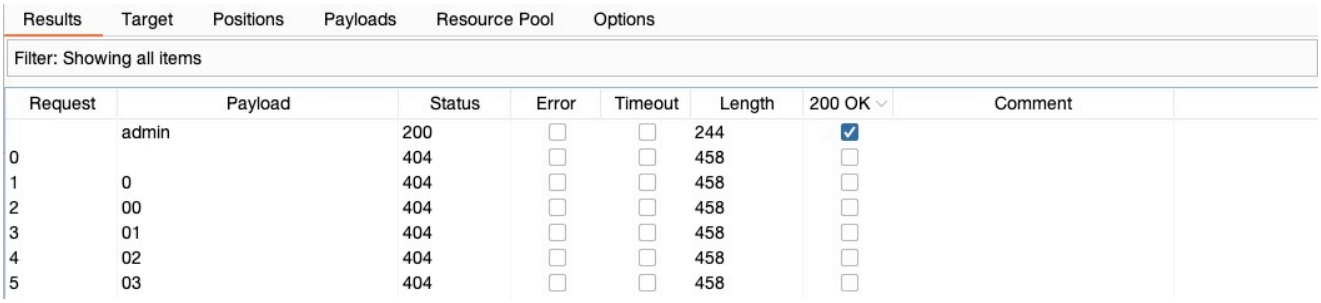
Results | Target | Positions | Payloads | Resource Pool | Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	200 OK	Comment
0		404	☐	☐	458	☐	
1	0	404	☐	☐	458	☐	
2	00	404	☐	☐	458	☐	
3	01	404	☐	☐	458	☐	
4	02	404	☐	☐	458	☐	
5	03	404	☐	☐	458	☐	
6	04	404	☐	☐	458	☐	

We can also see the `200 OK` column, which shows requests that match the `200 OK` grep value we specified in the Options tab. We can click on it to sort by it, such that we'll have

matching results at the top. Otherwise, we can sort by `status` or by `Length`. Once our scan is done, we see that we get one hit `/admin`:



We may now manually visit the page `<http://SERVER_IP:PORT/admin/>`, to make sure that it does exist.

Similarly, we can use `Burp Intruder` to do any type of web fuzzing and brute-forcing, including brute forcing for passwords, or fuzzing for certain PHP parameters, and so on. We can even use `Intruder` to perform password spraying against applications that use Active Directory (AD) authentication such as Outlook Web Access (OWA), SSL VPN portals, Remote Desktop Services (RDS), Citrix, custom web applications that use AD authentication, and more. However, as the free version of `Intruder` is extremely throttled, in the next section, we will see ZAP's fuzzer and its various options, which do not have a paid tier.

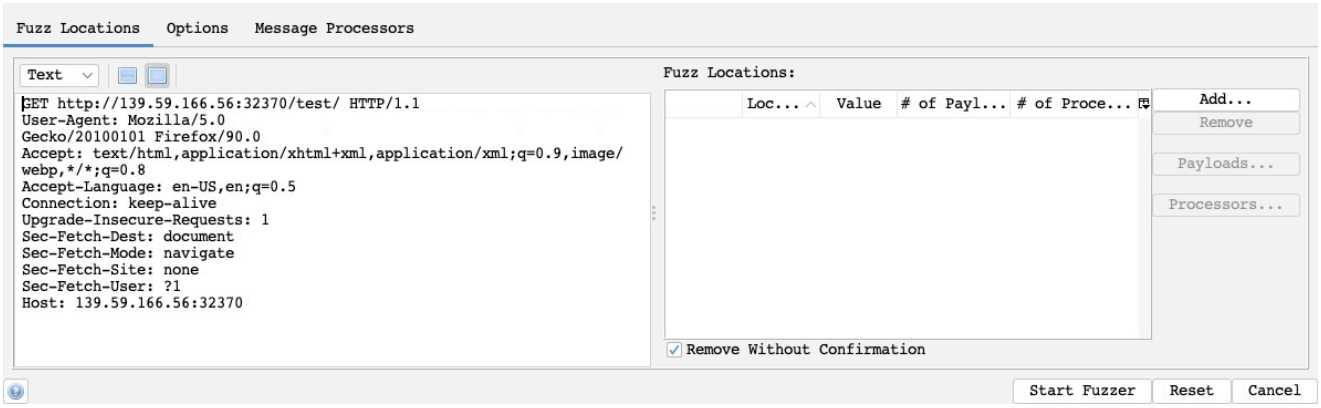
ZAP Fuzzer

ZAP's Fuzzer is called (`ZAP Fuzzer`). It can be very powerful for fuzzing various web end-points, though it is missing some of the features provided by Burp Intruder. ZAP Fuzzer, however, does not throttle the fuzzing speed, which makes it much more useful than Burp's free Intruder.

In this section, we will try to replicate what we did in the previous section using ZAP Fuzzer to have an "apples to apples" comparison and decide which one we like best.

Fuzz

To start our fuzzing, we will visit the URL from the exercise at the end of this section to capture a sample request. As we will be fuzzing for directories, let's visit `<http://SERVER_IP:PORT/test/>` to place our fuzzing location on `test` later on. Once we locate our request in the proxy history, we will right-click on it and select (`Attack>Fuzz`), which will open the `Fuzzer` window:



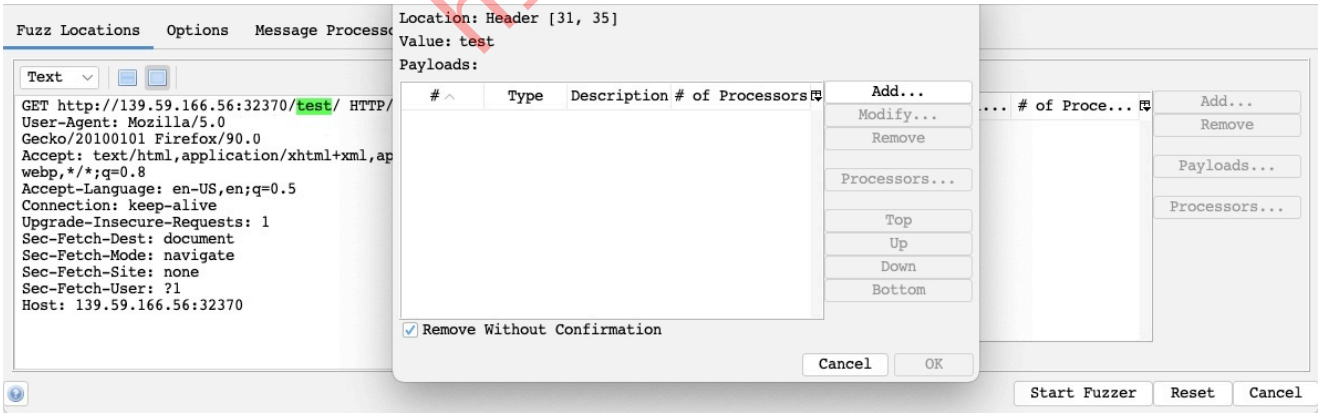
The main options we need to configure for our Fuzzer attack are:

- Fuzz Location
- Payloads
- Processors
- Options

Let's try to configure them for our web directory fuzzing attack.

Locations

The `Fuzz Location` is very similar to `Intruder Payload Position`, where our payloads will be placed. To place our location on a certain word, we can select it and click on the `Add` button on the right pane. So, let's select `test` and click on `Add`:



As we can see, this placed a `green` marker on our selected location and opened the `Payloads` window for us to configure our attack payloads.

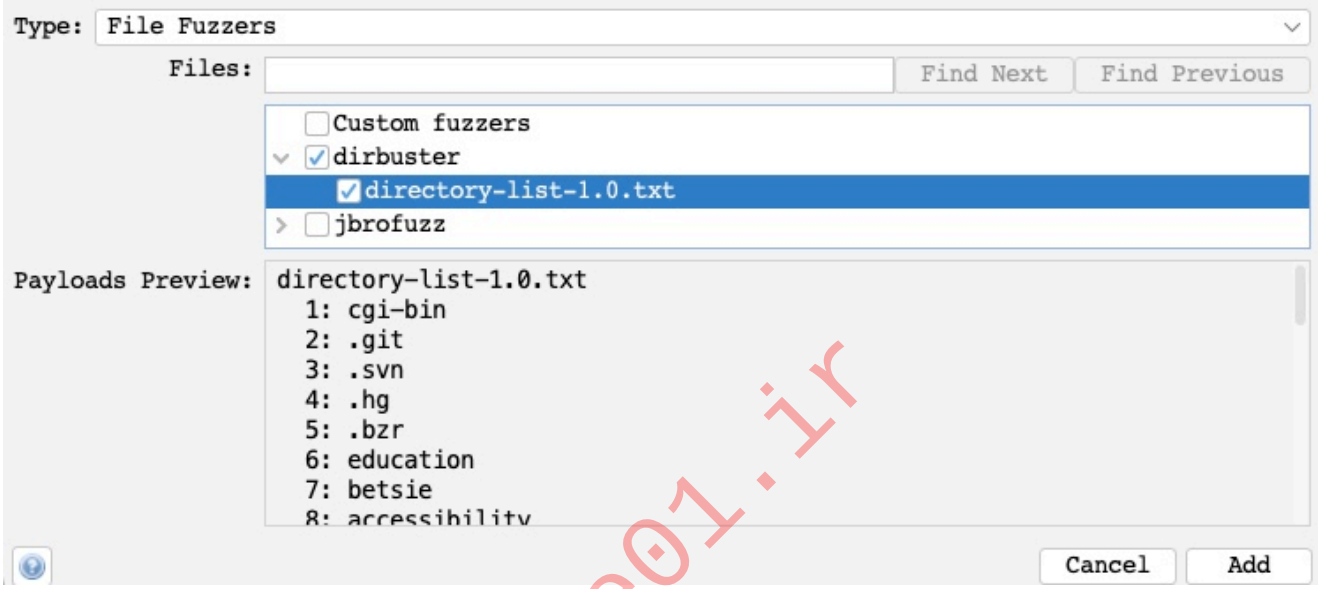
Payloads

The attack payloads in ZAP's Fuzzer are similar in concept to Intruder's Payloads, though they are not as advanced as Intruder's. We can click on the `Add` button to add our payloads

and select from 8 different payload types. The following are some of them:

- `File`: This allows us to select a payload wordlist from a file.
- `File Fuzzers`: This allows us to select wordlists from built-in databases of wordlists.
- `Numberzz`: Generates sequences of numbers with custom increments.

One of the advantages of ZAP Fuzzer is having built-in wordlists we can choose from so that we do not have to provide our own wordlist. More databases can be installed from the ZAP Marketplace, as we will see in a later section. So, we can select `File Fuzzers` as the `Type`, and then we will select the first wordlist from `dirbuster`:



Once we click the `Add` button, our payload wordlist will get added, and we can examine it with the `Modify` button.

Processors

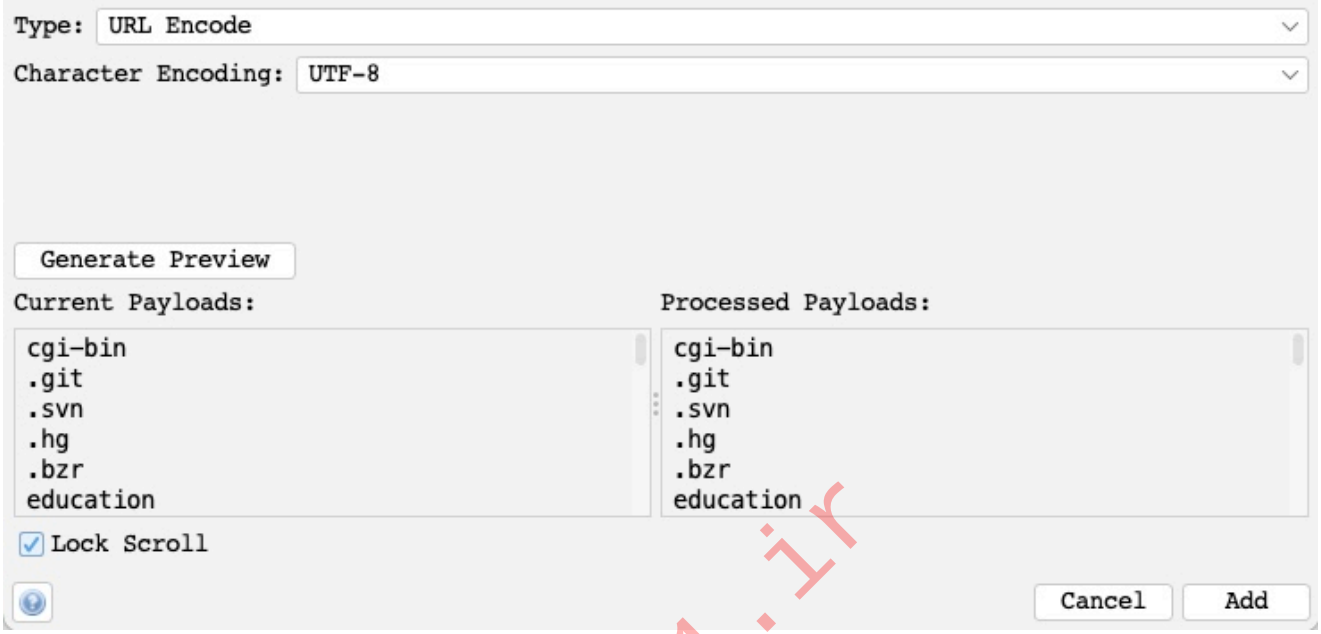
We may also want to perform some processing on each word in our payload wordlist. The following are some of the payload processors we can use:

- Base64 Decode/Encode
- MD5 Hash
- Postfix String
- Prefix String
- SHA-1/256/512 Hash
- URL Decode/Encode
- Script

As we can see, we have a variety of encoders and hashing algorithms to select from. We can also add a custom string before the payload with `Prefix String` or a custom string

with `Postfix String`. Finally, the `Script` type allows us to select a custom script that we built and run on every payload before using it in the attack.

We will select the `URL Encode` processor for our exercise to ensure that our payload gets properly encoded and avoid server errors if our payload contains any special characters. We can click on the `Generate Preview` button to preview how our final payload will look in the request:



Once that's done, we can click on `Add` to add the processor and click on `Ok` in the processors and payloads windows to close them.

Options

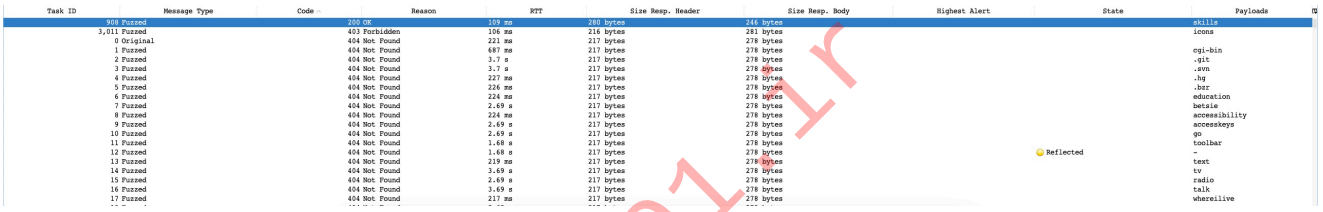
Finally, we can set a few options for our fuzzers, similar to what we did with Burp Intruder. For example, we can set the `Concurrent threads per scan` to `20`, so our scan runs very quickly:

The number of threads we set may be limited by how much computer processing power we want to use or how many connections the server allows us to establish.

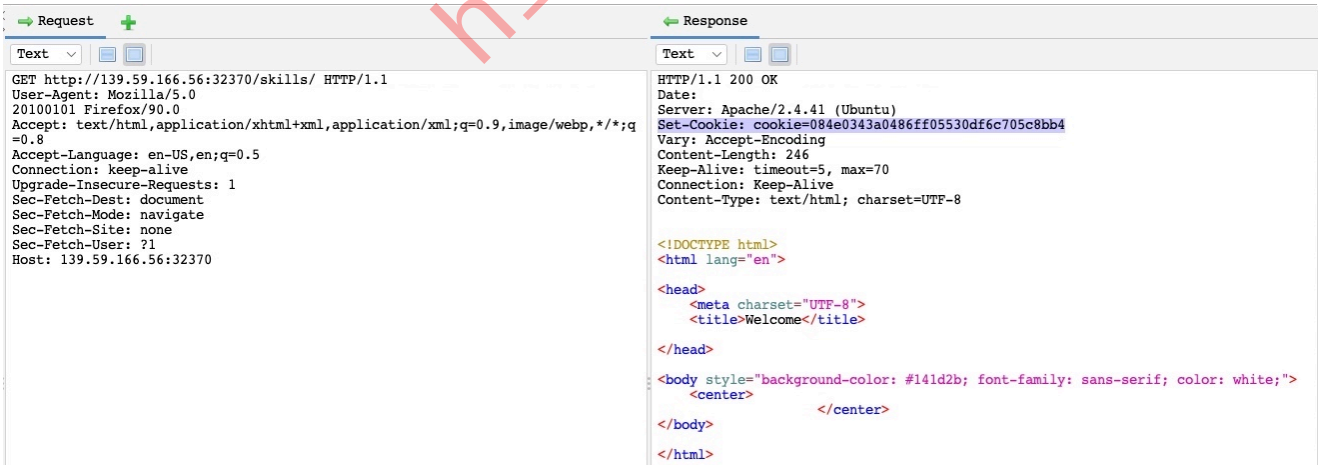
We may also choose to run through the payloads `Depth first`, which would attempt all words from the wordlist on a single payload position before moving to the next (e.g., try all passwords for a single user before brute-forcing the following user). We could also use `Breadth first`, which would run every word from the wordlist on all payload positions before moving to the next word (e.g., attempt every password for all users before moving to the following password).

Start

With all of our options configured, we can finally click on the `Start Fuzzer` button to start our attack. Once our attack is started, we can sort the results by the `Response` code, as we are only interested in responses with code `200`:



As we can see, we got one hit with code `200` with the `skills` payload, meaning that the `/skills/` directory exists on the server and is accessible. We can click on the request in the results window to view its details:



We can see from the response that this page is indeed accessible by us. There are other fields that may indicate a successful hit depending on the attack scenario, like `Size Resp. Body` which may indicate that we got a different page if its size was different than other responses, or `RTT` for attacks like `time-based SQL injections`, which are detected by a time delay in the server response.

Burp Scanner

An essential feature of web proxy tools is their web scanners. Burp Suite comes with `Burp Scanner`, a powerful scanner for various types of web vulnerabilities, using a `Crawler` for building the website structure, and `Scanner` for passive and active scanning.

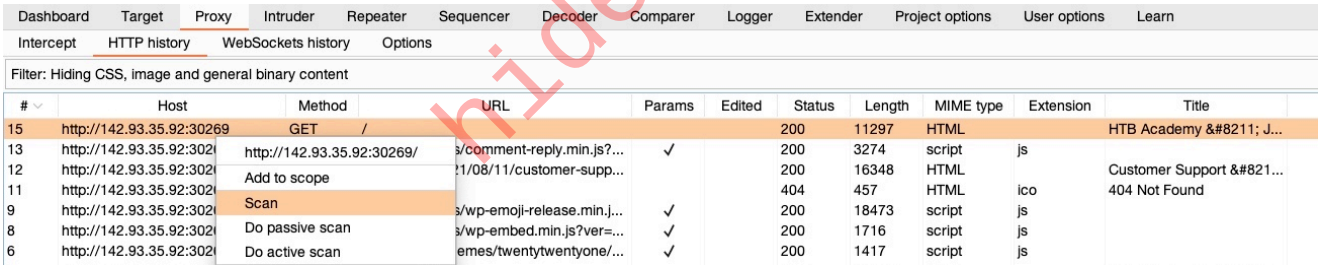
Burp Scanner is a Pro-Only feature, and it is not available in the free Community version of Burp Suite. However, given the wide scope that Burp Scanner covers and the advanced features it includes, it makes it an enterprise-level tool, and as such, it is expected to be a paid feature.

Target Scope

To start a scan in Burp Suite, we have the following options:

1. Start scan on a specific request from Proxy History
2. Start a new scan on a set of targets
3. Start a scan on items in-scope

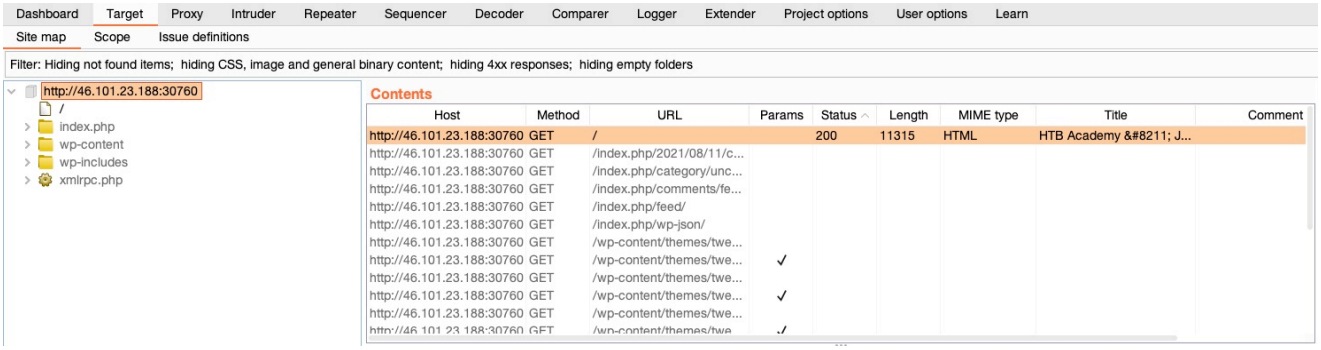
To start a scan on a specific request from Proxy History, we can right-click on it once we locate it in the history, and then select `Scan` to be able to configure the scan before we run it, or select `Passive/Active Scan` to quickly start a scan with the default configurations:



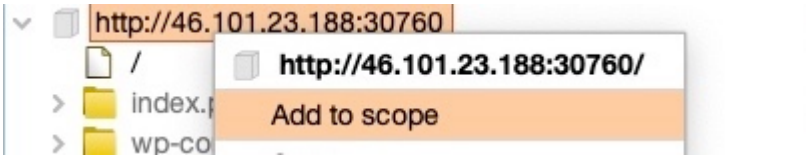
We may also click on the `New Scan` button on the `Dashboard` tab, which would open the `New Scan` configuration window to configure a scan on a set of custom targets. Instead of creating a custom scan from scratch, let's see how we can utilize the scope to properly define what's included/excluded from our scans using the `Target Scope`. The `Target Scope` can be utilized with all Burp features to define a custom set of targets that will be processed. Burp also allows us to limit Burp to in-scope items to save resources by ignoring any out-of-scope URLs.

Note: We will be scanning the web application from the exercise found at the end of the next section. If you obtain a license to use Burp Pro, you may spawn the target at the end of the next section and follow along here.

If we go to (`Target>Site map`), it will show a listing of all directories and files burp has detected in various requests that went through its proxy:

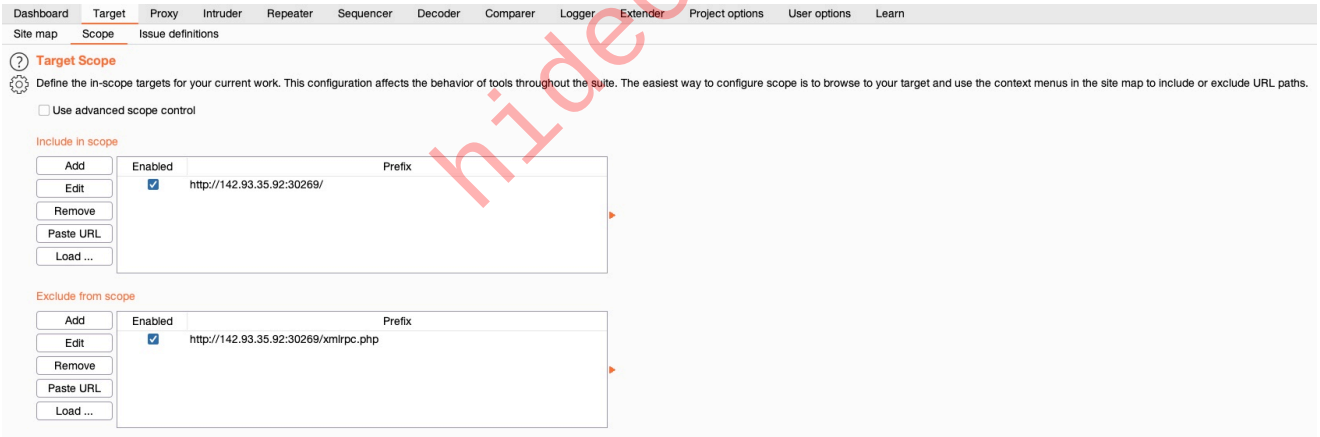


To add an item to our scope, we can right-click on it and select `Add to scope`:



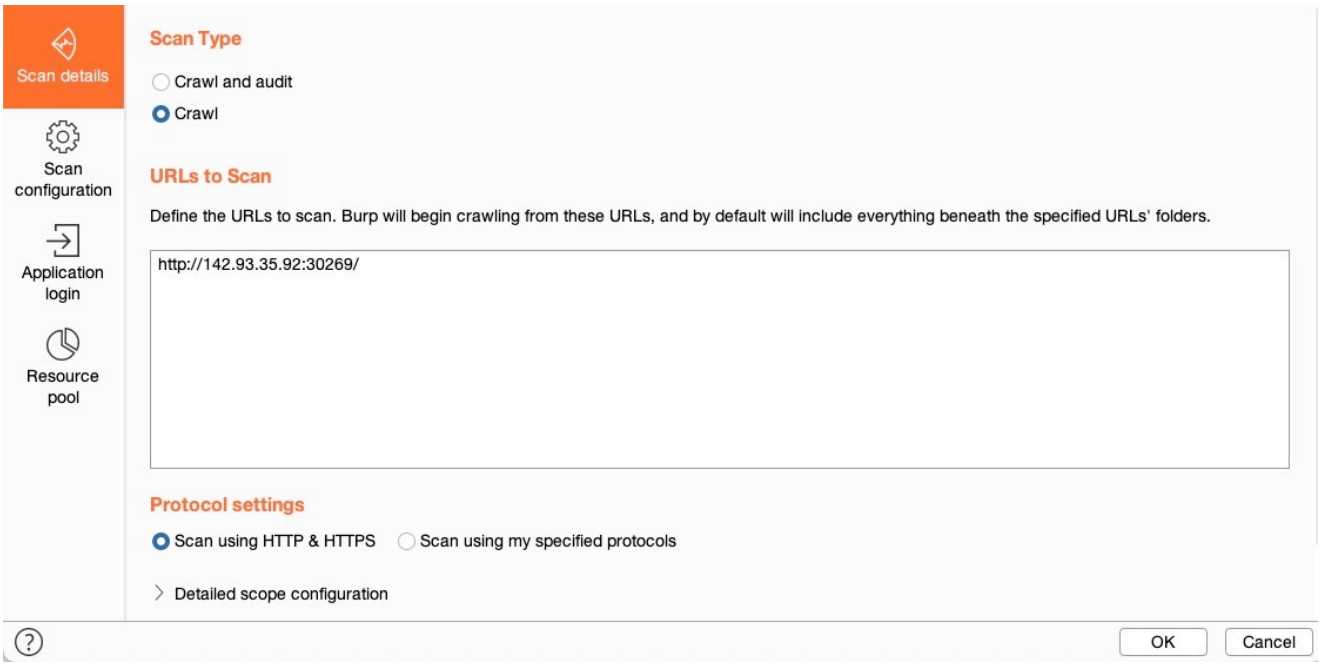
Note: When you add the first item to your scope, Burp will give you the option to restrict its features to in-scope items only, and ignore any out-of-scope items.

We may also need to exclude a few items from scope if scanning them may be dangerous or may end our session 'like a logout function'. To exclude an item from our scope, we can right-click on any in-scope item and select `Remove from scope`. Finally, we can go to (`Target>Scope`) to view the details of our scope. Here, we may also add/remove other items and use advanced scope control to specify regex patterns to be included/excluded.



Crawler

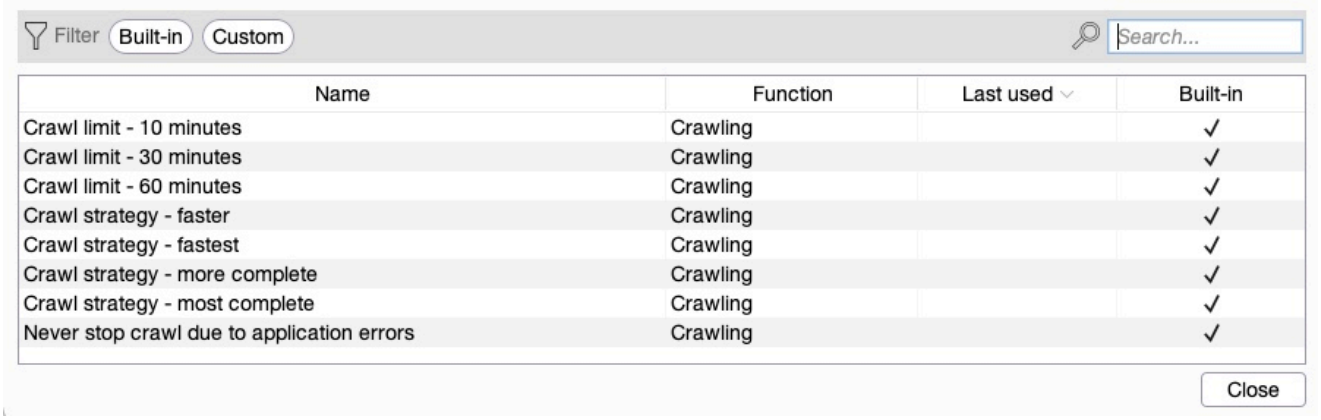
Once we have our scope ready, we can go to the `Dashboard` tab and click on `New Scan` to configure our scan, which would be automatically populated with our in-scope items:



We see that Burp gives us two scanning options: `Crawl and Audit` and `Crawl`. A Web Crawler navigates a website by accessing any links found in its pages, accessing any forms, and examining any requests it makes to build a comprehensive map of the website. In the end, Burp Scanner presents us with a map of the target, showing all publicly accessible data in a single place. If we select `Crawl and Audit`, Burp will run its scanner after its Crawler (as we will see later).

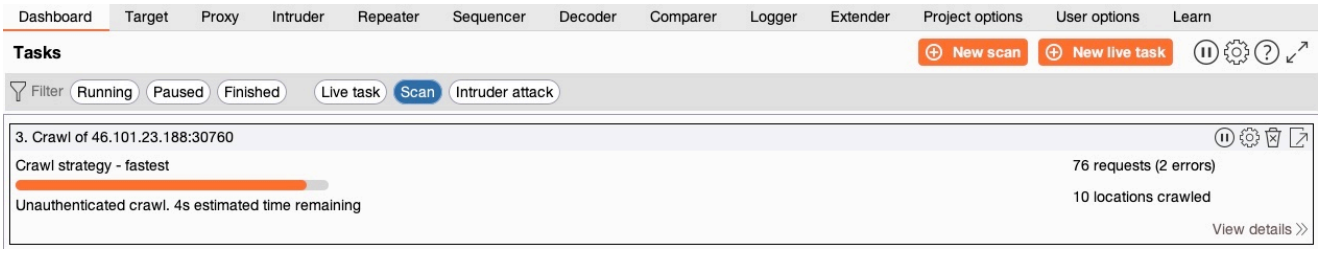
Note: A Crawl scan only follows and maps links found in the page we specified, and any pages found on it. It does not perform a fuzzing scan to identify pages that are never referenced, like what dirbuster or ffuf would do. This can be done with Burp Intruder or Content Discovery, and then added to scope, if needed.

Let us select `Crawl` as a start and go to the `Scan configuration` tab to configure our scan. From here, we may choose to click on `New` to build a custom configuration, which would allow us to set the configurations like the crawling speed or limit, whether Burp will attempt to log in to any login forms, and a few other configurations. For the sake of simplicity, we will click on the `Select from library` button, which gives us a few preset configurations we can pick from (or custom configurations we previously defined):

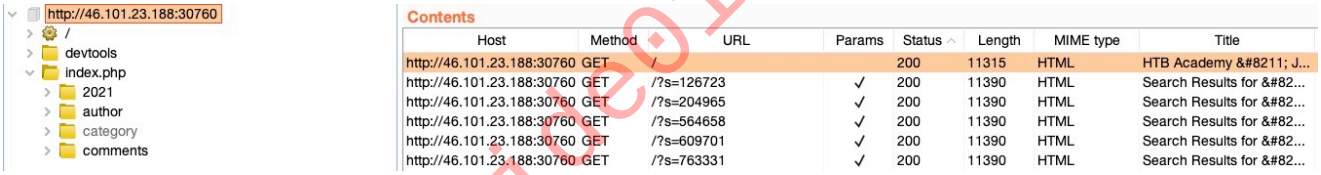


We will select the `Crawl strategy - fastest` option and continue to the `Application login` tab. In this tab, we can add a set of credentials for Burp to attempt in any Login forms/fields it can find. We may also record a set of steps by performing a manual login in the pre-configured browser, such that Burp knows what steps to follow to gain a login session. This can be essential if we were running our scan using an authenticated user, which would allow us to cover parts of the web application that Burp may otherwise not have access to. As we do not have any credentials, we'll leave it empty.

With that, we can click on the `Ok` button to start our Crawl scan. Once our scan starts, we can see its progress in the `Dashboard` tab under `Tasks`:



We may also click on the `View details` button on the tasks to see more details about the running scan or click on the gear icon to customize our scan configurations further. Finally, once our scan is complete, we'll see `Crawl Finished` in the task info, and then we can go back to (`Target>Site map`) to view the updated site map:

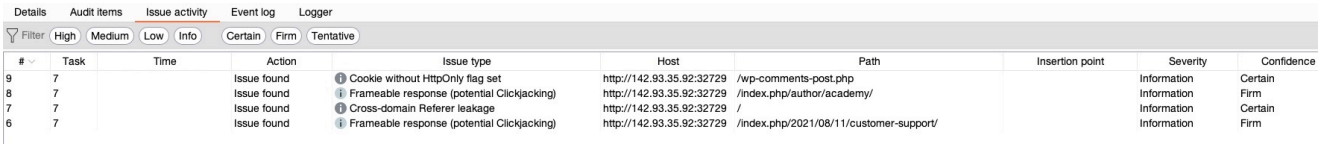


Passive Scanner

Now that the site map is fully built, we may select to scan this target for potential vulnerabilities. When we choose the `Crawl and Audit` option in the `New Scan` dialog, Burp will perform two types of scans: A `Passive Vulnerability Scan` and an `Active Vulnerability Scan`.

Unlike an Active Scan, a Passive Scan does not send any new requests but analyzes the source of pages already visited in the target/scope and then tries to identify `potential` vulnerabilities. This is very useful for a quick analysis of a specific target, like missing HTML tags or potential DOM-based XSS vulnerabilities. However, without sending any requests to test and verify these vulnerabilities, a Passive Scan can only suggest a list of potential vulnerabilities. Still, Burp Passive Scanner does provide a level of `Confidence` for each identified vulnerability, which is also helpful for prioritizing potential vulnerabilities.

Let's start by trying to perform a Passive Scan only. To do so, we can once again select the target in (`Target>Site map`) or a request in Burp Proxy History, then right-click on it and select `Do passive scan` or `Passively scan this target`. The Passive Scan will start running, and its task can be seen in the `Dashboard` tab as well. Once the scan finishes, we can click on `View Details` to review identified vulnerabilities and then select the `Issue activity` tab:



Alternately, we can view all identified issues in the `Issue activity` pane on the `Dashboard` tab. As we can see, it shows the list of potential vulnerabilities, their severity, and their confidence. Usually, we want to look for vulnerabilities with `High` severity and `Certain` confidence. However, we should include all levels of severity and confidence for very sensitive web applications, with a special focus on `High` severity and `Confident/Firm` confidence.

Active Scanner

We finally reach the most powerful part of Burp Scanner, which is its Active Vulnerability Scanner. An active scan runs a more comprehensive scan than a Passive Scan, as follows:

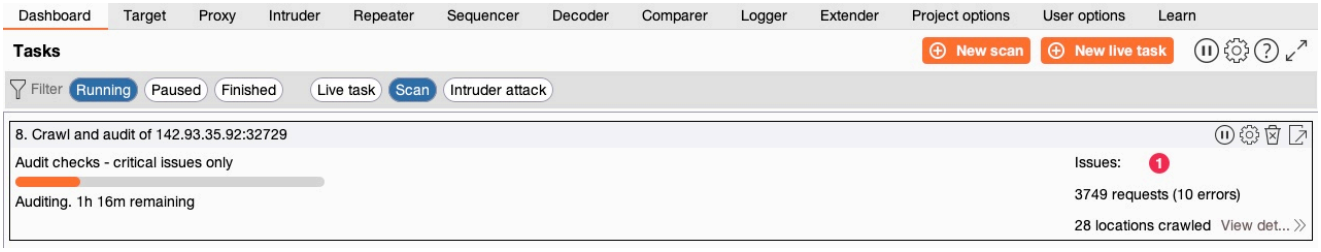
1. It starts by running a Crawl and a web fuzzer (like dirbuster/ffuf) to identify all possible pages
2. It runs a Passive Scan on all identified pages
3. It checks each of the identified vulnerabilities from the Passive Scan and sends requests to verify them
4. It performs a JavaScript analysis to identify further potential vulnerabilities
5. It fuzzes various identified insertion points and parameters to look for common vulnerabilities like XSS, Command Injection, SQL Injection, and other common web vulnerabilities

The Burp Active scanner is considered one of the best tools in that field and is frequently updated to scan for newly identified web vulnerabilities by the Burp research team.

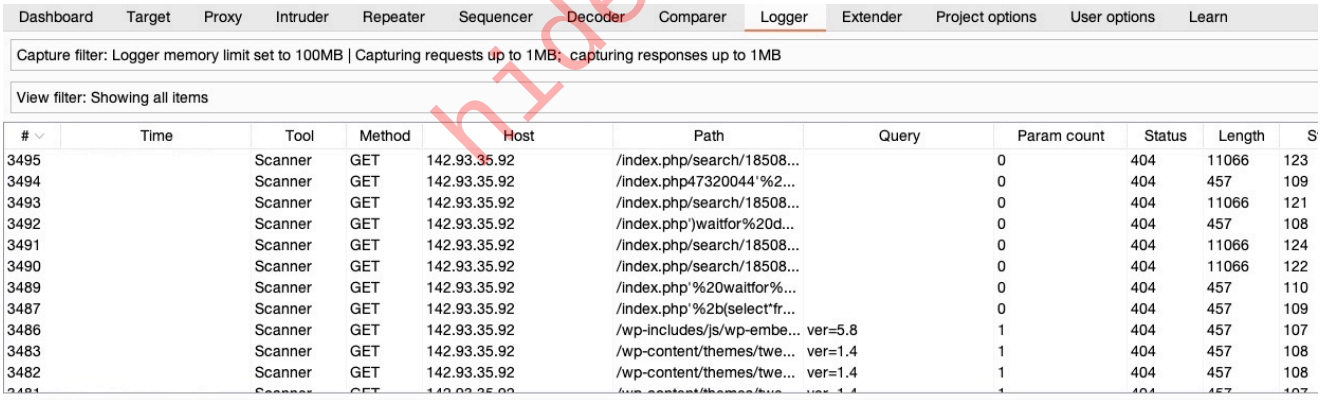
We can start an Active Scan similarly to how we began a Passive Scan by selecting the `Do active scan` from the right-click menu on a request in Burp Proxy History. Alternatively, we can run a scan on our scope with the `New Scan` button in the `Dashboard` tab, which would allow us to configure our active scan. This time, we will select the `Crawl and Audit` option, which would perform all of the above points and everything we have discussed so far.

We may also set the Crawl configurations (as we discussed earlier) and the Audit configurations. The Audit configurations enable us to select what type of vulnerabilities we want to scan (defaults to all), where the scanner would attempt inserting its payloads, in addition to many other useful configurations. Once again, we can select a configuration preset with the `Select from library` button. For our test, as we are interested in `High` vulnerabilities that may allow us to gain control over the backend server, we will select the `Audit checks - critical issues only` option. Finally, we may add login details, as we previously saw with the Crawl configurations.

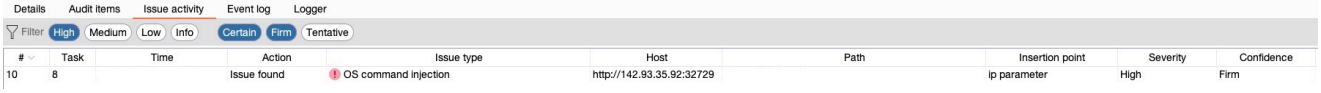
Once we select our configurations, we can click on the `Ok` button to start the scan, and the active scan task should be added in the `Tasks` pane in the `Dashboard` tab:



The scan will run all of the steps mentioned above, which is why it will take significantly longer to finish than our earlier scans depending on the configurations we selected. As the scan is running, we can view the various requests it is making by clicking on the `View details` button and selecting the `Logger` tab, or by going to the `Logger` tab in Burp, which shows all requests that went through or were made by Burp:

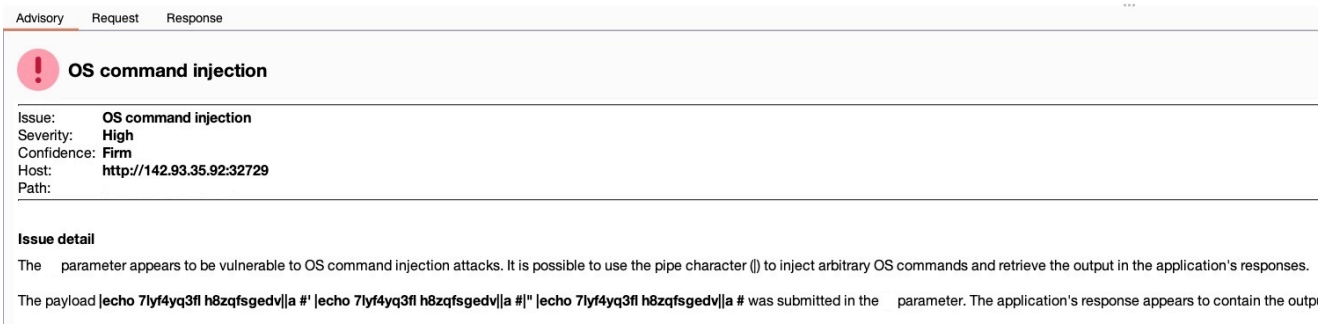


Once the scan is done, we can look at the `Issue activity` pane in the `Dashboard` tab to view and filter all of the issues identified so far. From the filter above the results, let's select `High` and `Certain` and see our filtered results:



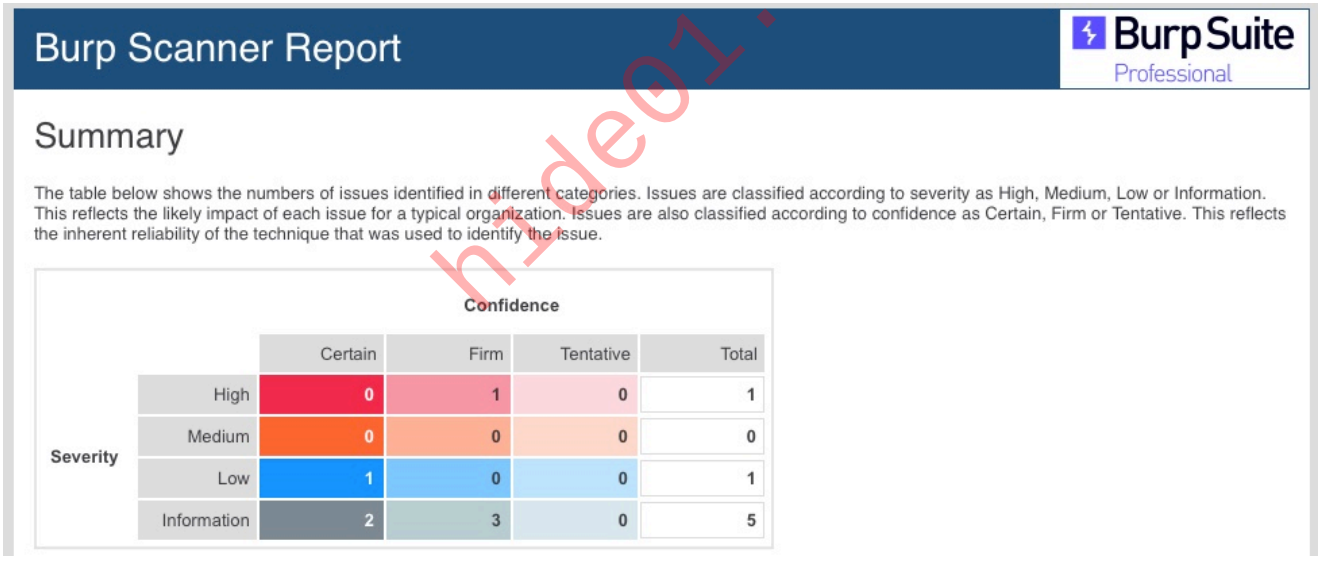
We see that Burp identified an `OS command injection` vulnerability, which is ranked with a `High` severity and `Firm` confidence. As Burp is firmly confident that this severe vulnerability exists, we can read about it by clicking on it and reading the advisory shown and view the

sent request and received response, to be able to know whether the vulnerability can be exploited or how it poses a threat on the webserver:



Reporting

Finally, once all of our scans are completed, and all potential issues have been identified, we can go to (`Target>Site map`), right-click on our target, and select (`Issue>Report issues for this host`). We will get prompted to select the export type for the report and what information we would like to include in the report. Once we export the report, we can open it in any web browser to view its details:



As we can see, Burp's report is very organized and can be customized to only include select issues by severity/confidence. It also shows proof-of-concept details of how to exploit the vulnerability and information on how to remediate it. These reports may be used as supplementary data for the detailed reports that we prepare for our clients or the web application developers when performing a web penetration test or can be stored for our future reference. We should never merely export a report from any penetration tool and submit it to a client as the final deliverable. Instead, the reports and data generated by tools can be helpful as appendix data for clients who may need the raw scan data for remediation efforts or to import into a tracking dashboard.

ZAP Scanner

ZAP also comes bundled with a Web Scanner similar to Burp Scanner. ZAP Scanner is capable of building site maps using ZAP Spider and performing both passive and active scans to look for various types of vulnerabilities.

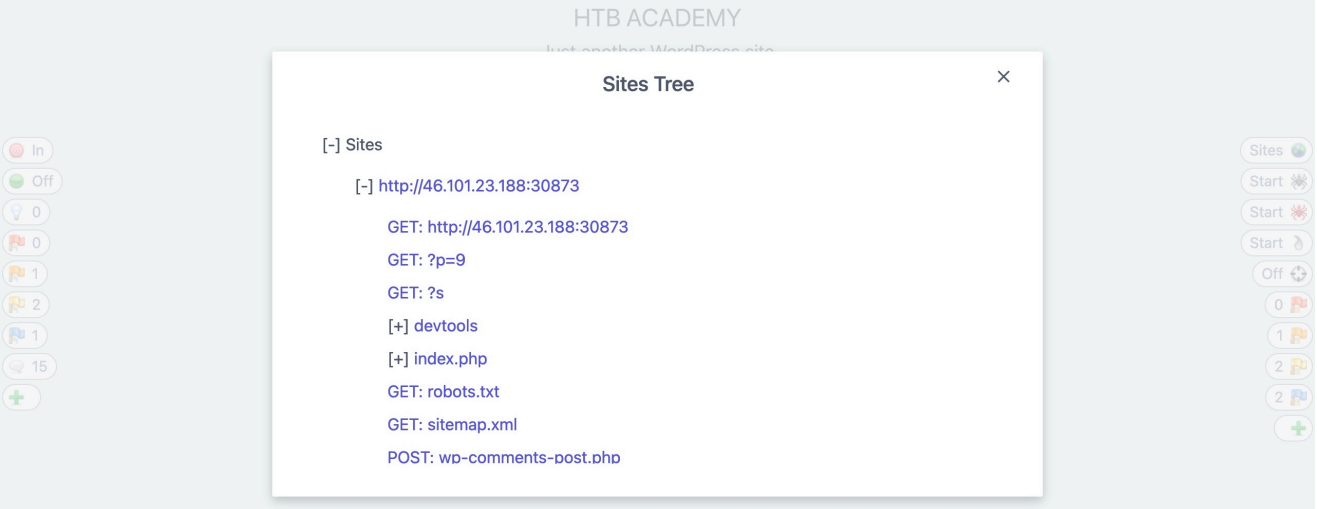
Spider

Let's start with `ZAP Spider`, which is similar to the Crawler feature in Burp. To start a Spider scan on any website, we can locate a request from our History tab and select (`Attack>Spider`) from the right-click menu. Another option is to use the HUD in the pre-configured browser. Once we visit the page or website we want to start our Spider scan on, we can click on the second button on the right pane (`Spider Start`), which would prompt us to start the scan:

Note: When we click on the Spider button, ZAP may tell us that the current website is not in our scope, and will ask us to automatically add it to the scope before starting the scan, to which we can say 'Yes'. The Scope is the set of URLs ZAP will test if we start a generic scan, and it can be customized by us to scan multiple websites and URLs. Try to add multiple targets to the scope to see how the scan would run differently.

Once we click on `Start` on the pop-up window, our Spider scan should start spidering the website by looking for links and validating them, very similar to how Burp Crawler works. We can see the progress of the spider scan both in the HUD on the `Spider` button or in the main ZAP UI, which should automatically switch to the current Spider tab to show the progress and sent requests. When our scan is complete, we can check the Sites tab on the main ZAP UI, or we can click on the first button on the right pane (`Sites Tree`), which should show us an expandable tree-list view of all identified websites and their sub-

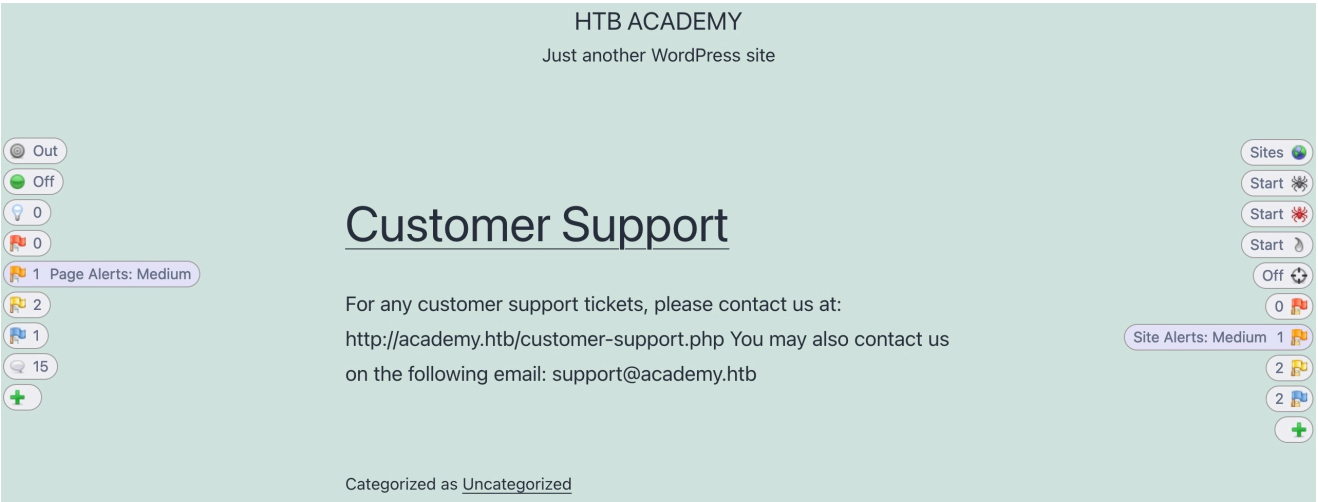
directories:



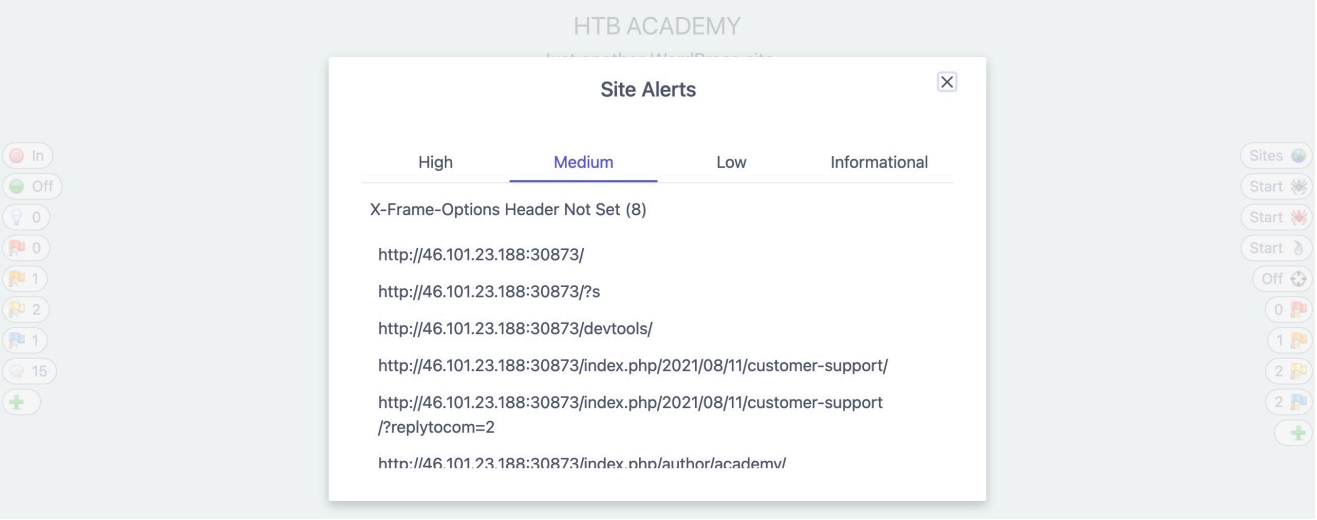
Tip: ZAP also has a different type of Spider called `Ajax Spider`, which can be started from the third button on the right pane. The difference between this and the normal scanner is that Ajax Spider also tries to identify links requested through JavaScript AJAX requests, which may be running on the page even after it loads. Try running it after the normal Spider finishes its scan, as this may give a better output and add a few links the normal Spider may have missed, though it may take a little bit longer to finish.

Passive Scanner

As ZAP Spider runs and makes requests to various end-points, it is automatically running its passive scanner on each response to see if it can identify potential issues from the source code, like missing security headers or DOM-based XSS vulnerabilities. This is why even before running the Active Scanner, we may see the alerts button start to get populated with a few identified issues. The alerts on the left pane shows us issues identified in the current page we are visiting, while the right pane shows us the overall alerts on this web application, which includes alerts found on other pages:

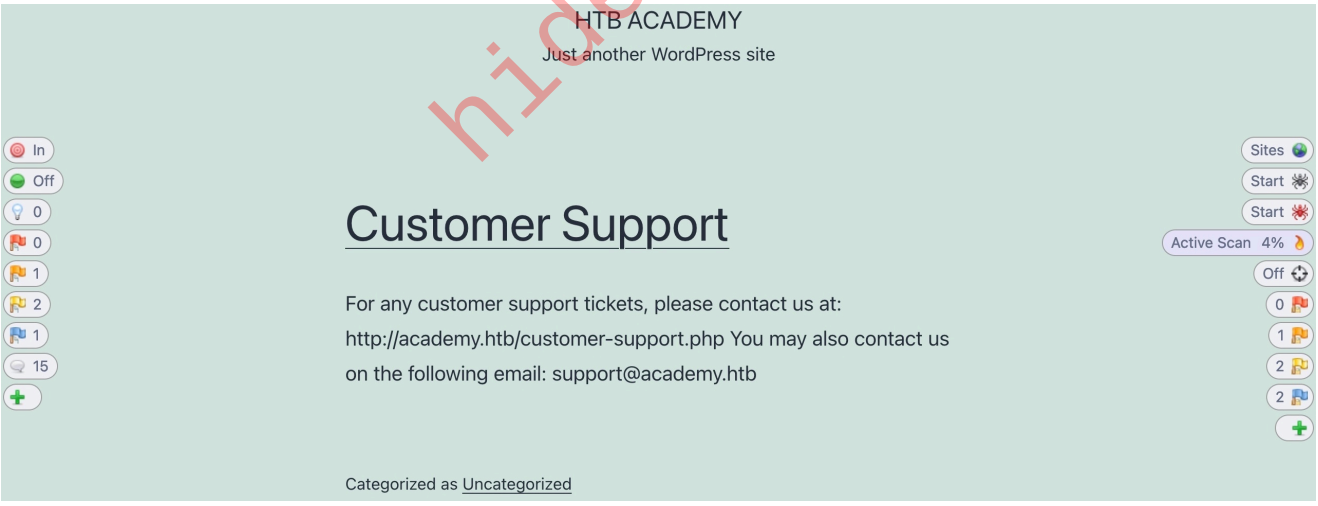


We can also check the `Alerts` tab on the main ZAP UI to see all identified issues. If we click on any alert, ZAP will show us its details and the pages it was found on:

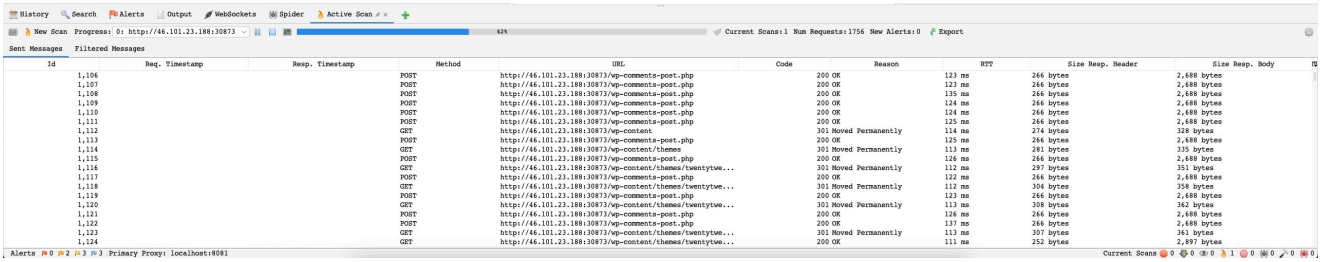


Active Scanner

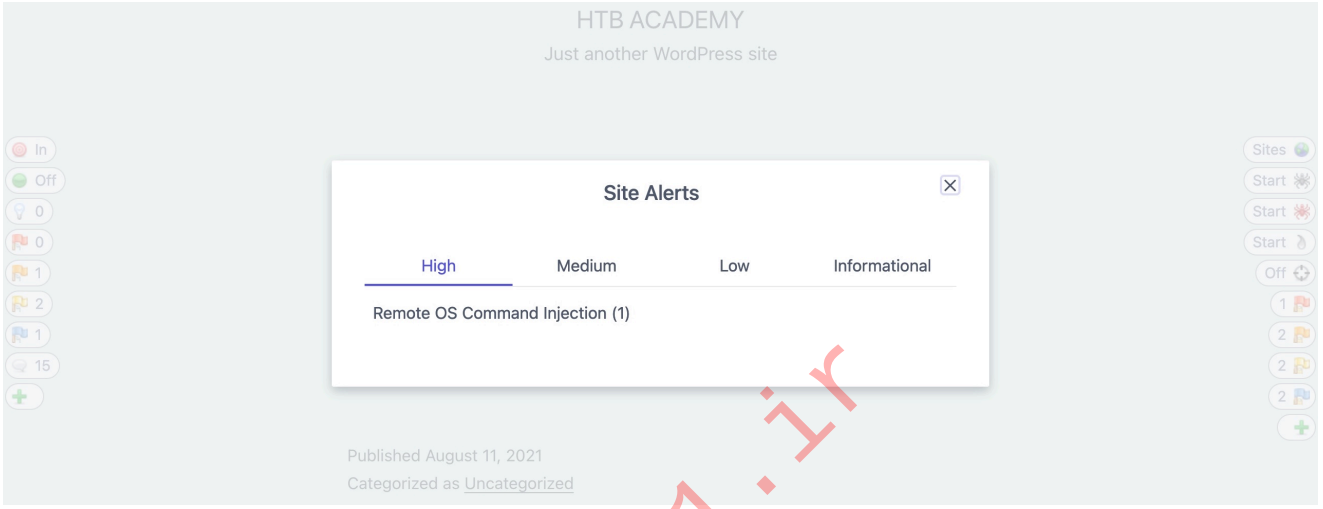
Once our site's tree is populated, we can click on the `Active Scan` button on the right pane to start an active scan on all identified pages. If we have not yet run a Spider Scan on the web application, ZAP will automatically run it to build a site tree as a scan target. Once the Active Scan starts, we can see its progress similarly to how we did with the Spider Scan:



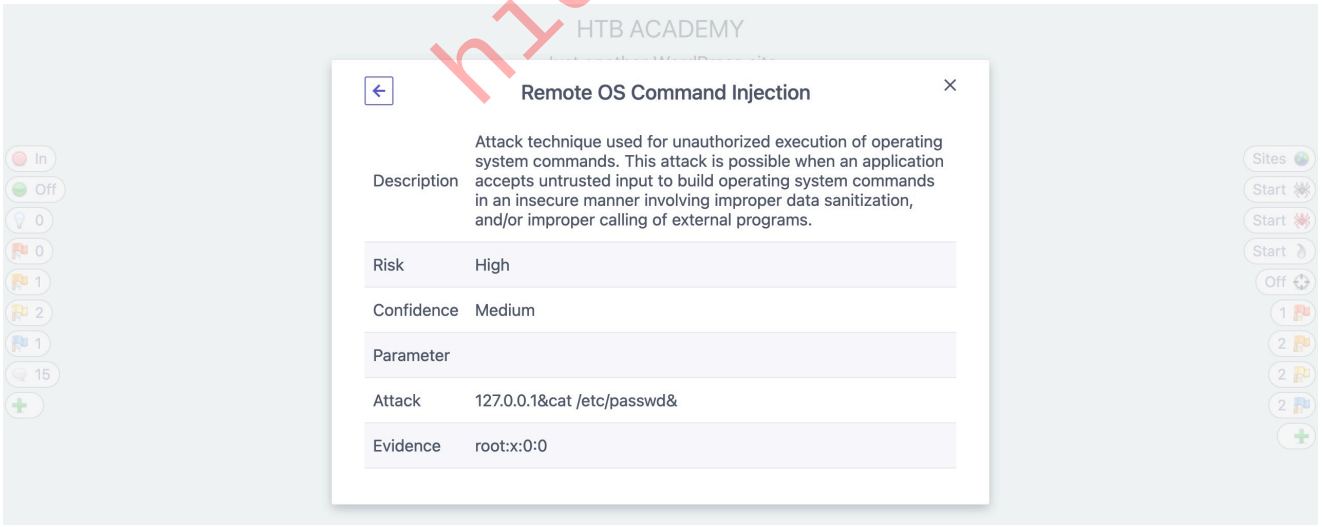
The Active Scanner will try various types of attacks against all identified pages and HTTP parameters to identify as many vulnerabilities as it can. This is why the Active Scanner will take longer to complete. As the Active Scan runs, we will see the alerts button start to get populated with more alerts as ZAP uncovers more issues. Furthermore, we can check the main ZAP UI for more details on the running scan and can view the various requests sent by ZAP:



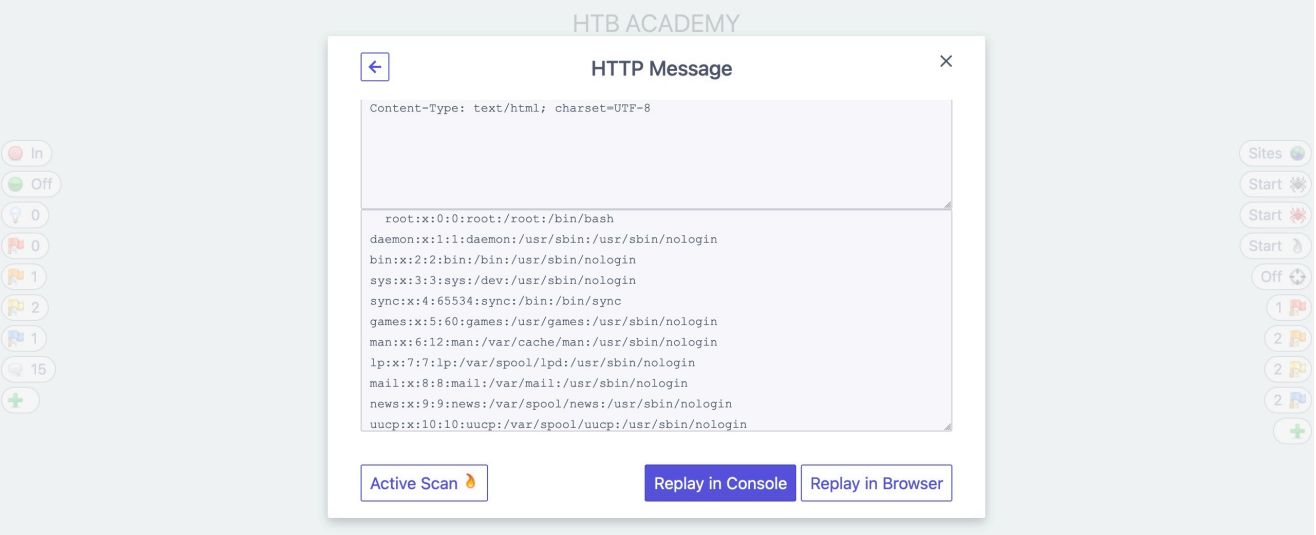
Once the Active Scan finishes, we can view the alerts to see which ones to follow up on. While all alerts should be reported and taken into consideration, the `High` alerts are the ones that usually lead to directly compromising the web application or the back-end server. If we click on the `High Alerts` button, it will show us the identified High Alert:



We can also click on it to see more details about it and see how we may replicate and patch this vulnerability:



In the alert details window, we can also click on the URL to see the request and response details that ZAP used to identify this vulnerability, and we may also repeat the request through ZAP HUD or ZAP Request Editor:



Reporting

Finally, we can generate a report with all of the findings identified by ZAP through its various scans. To do so, we can select (`Report>Generate HTML Report`) from the top bar, which would prompt us for the save location to save the report. We may also export the report in other formats like `XML` or `Markdown`. Once we generate our report, we can open it in any browser to view it:

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	3
Low	8
Informational	6

Alerts

Name	Risk Level	Number of Instances
Remote OS Command Injection	High	1
Cross-Domain Misconfiguration	Medium	1
Directory Browsing	Medium	9
X-Frame-Options Header Not Set	Medium	8
Absence of Anti-CSRF Tokens	Low	11
Incomplete or No Cache-control Header Set	Low	37
X-Content-Type-Options Header Missing	Low	44
Charset Mismatch	Informational	1
Information Disclosure - Suspicious Comments	Informational	7
Timestamp Disclosure - Unix	Informational	144

As we can see, the report shows all identified details in an organized manner, which may be helpful to keep as a log for various web applications we run our scans on during a penetration test.

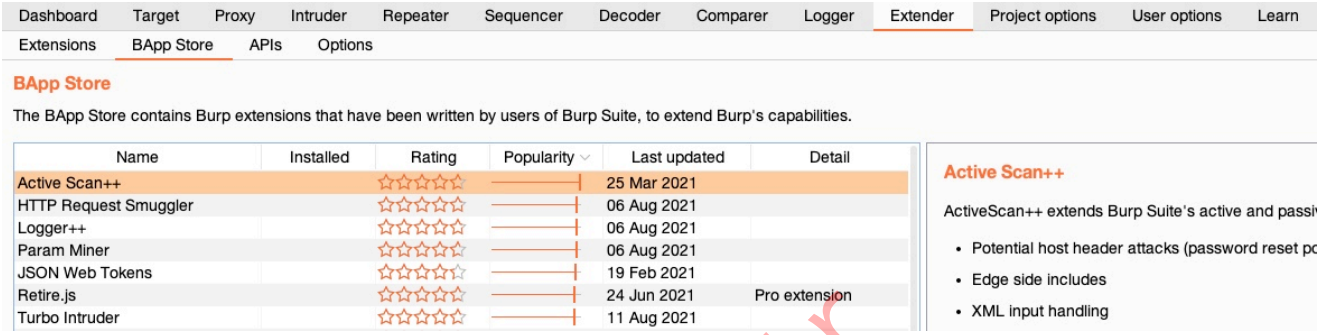
Extensions

Both Burp and ZAP have extension capabilities, such that the community of Burp users can develop extensions for Burp for everyone to use. Such extensions can perform specific

actions on any captured requests, for example, or add new features, like decoding and beautifying code. Burp allows extensibility through its `Extender` feature and its BApp Store, while ZAP has its ZAP Marketplace to install new plugins.

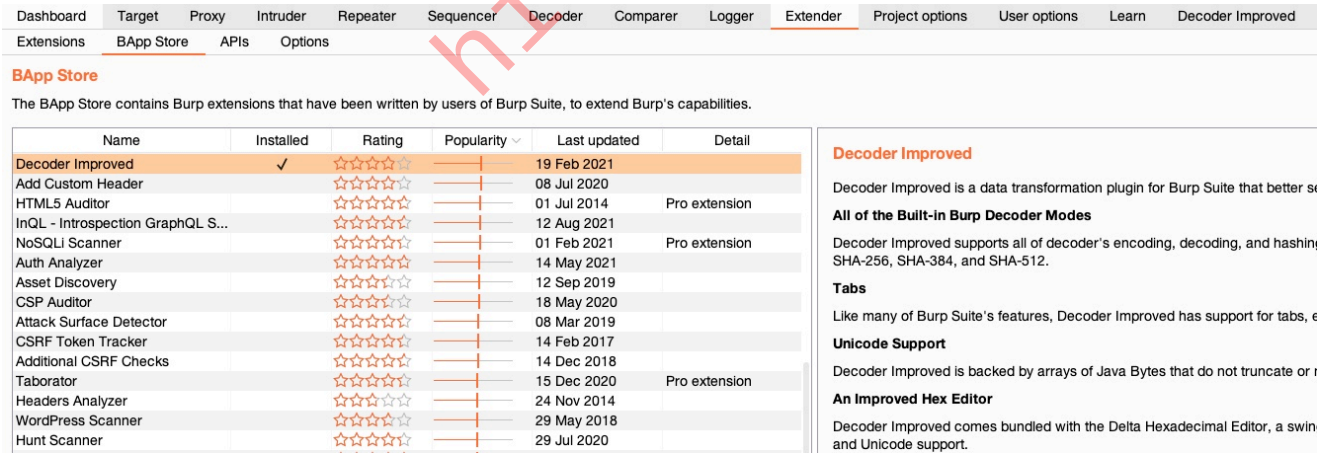
BApp Store

To find all available extensions, we can click on the `Extender` tab within Burp and select the `BApp Store` sub-tab. Once we do this, we will see a host of extensions. We can sort them by `Popularity` so that we know which ones users are finding most useful:



Note: Some extensions are for Pro users only, while most others are available to everyone.

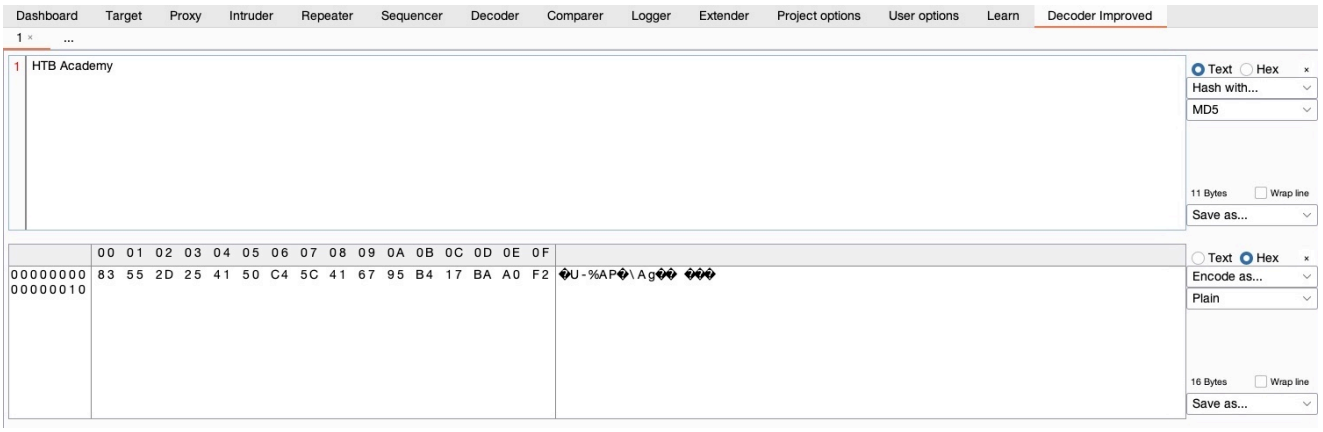
We see many useful extensions, take some time to go through them and see which are most useful to you, and then try installing and testing them. Let's try installing the `Decoder Improved` extension:



Note: Some extensions have requirements that are not usually installed on Linux/macOS/Windows by default, like `Jython`, so you have to install them before being able to install the extension.

Once we install `Decoder Improved`, we will see its new tab added to Burp. Each extension has a different usage, so we may click on any extension's documentation in `BApp Store` to read more about it or visit its GitHub page for more information about its usage. We can use this extension just as we would use Burp's Decoder, with the benefit of having many

additional encoders included. For example, we can input text we want to be hashed with `MD5`, and select `Hash With>MD5`:



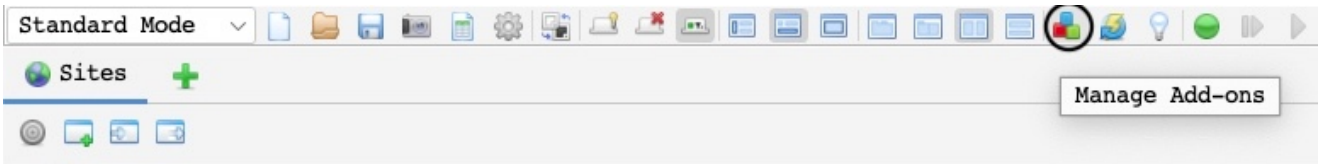
Similarly, we can perform other types of encoding and hashing. There are many other Burp Extensions that can be utilized to further extend the functionality of Burp.

Some extensions worth checking out include, but are not limited to:

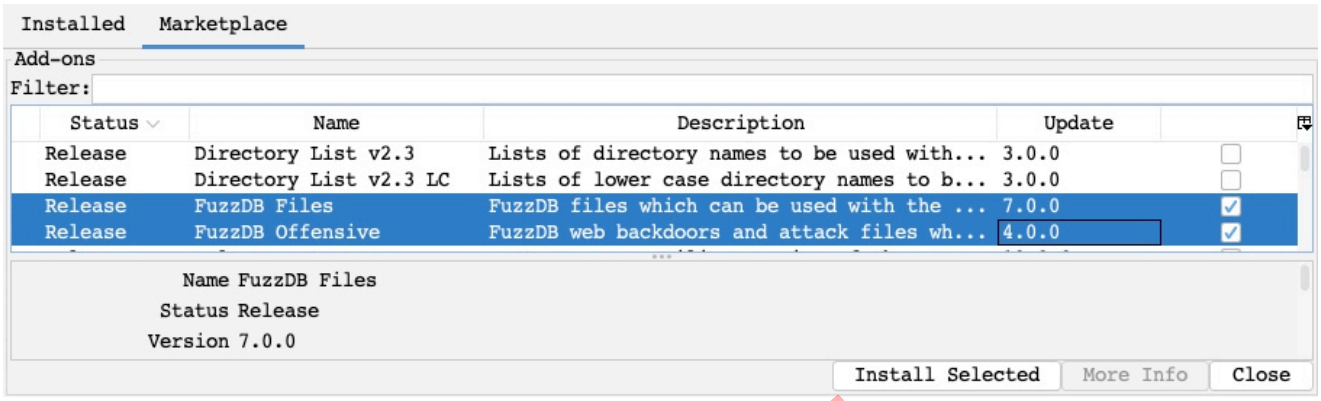
.NET beautifier	J2EEScan	Software Vulnerability Scanner
Software Version Reporter	Active Scan++	Additional Scanner Checks
AWS Security Checks	Backslash Powered Scanner	Wsdler
Java Deserialization Scanner	C02	Cloud Storage Tester
CMS Scanner	Error Message Checks	Detect Dynamic JS
Headers Analyzer	HTML5 Auditor	PHP Object Injection Check
JavaScript Security	Retire.JS	CSP Auditor
Random IP Address Header	Autorize	CSRF Scanner
JS Link Finder		

ZAP Marketplace

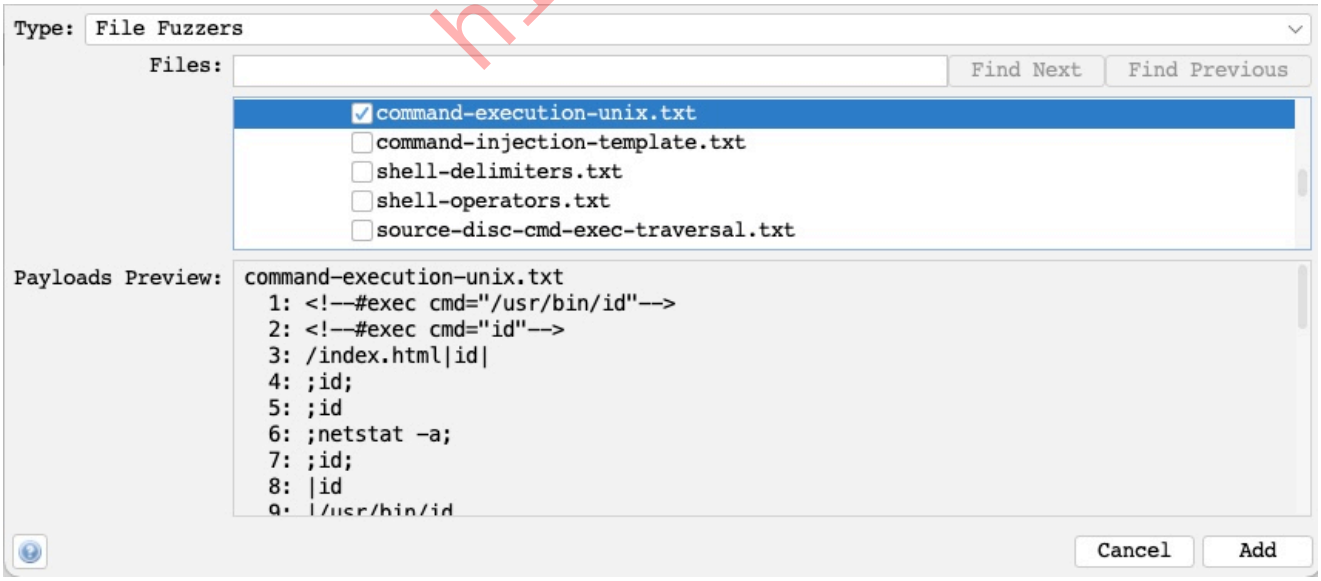
ZAP also has its own extensibility feature with the `Marketplace` that allows us to install various types of community-developed add-ons. To access ZAP's marketplace, we can click on the `Manage Add-ons` button and then select the `Marketplace` tab:



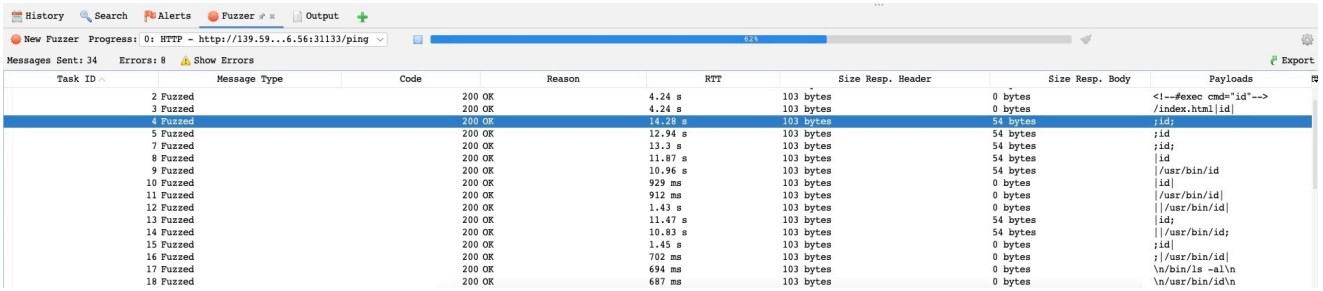
In this tab, we can see the different available add-ons for ZAP. Some add-ons may be in their `Release` build, meaning that they should be stable to be used, while others are in their `Beta/Alpha` builds, which means that they may experience some issues in their use. Let's try installing the `FuzzDB Files` and `FuzzDB Offensive` add-ons, which adds new wordlists to be used in ZAP's fuzzer:



Now, we will have the option to pick from the various wordlists and payloads provided by FuzzDB when performing an attack. For example, suppose we were to perform a Command Injection fuzzing attack on one of the exercises we previously used in this module. In that case, we will see that we have more options in the `File Fuzzers` wordlists, including an OS Command Injection wordlist under (`fuzzdb>attack>os-cmd-execution`), which would be perfect for this attack:



Now, if we run the fuzzer on our exercise using the above wordlist, we will see that it was able to exploit it in various ways, which would be very helpful if we were dealing with a WAF protected web application:



Try to repeat the above with the first exercise in this module to see how add-ons can help in making your penetration test easier.

Closing Thoughts

Throughout this module, we have demonstrated the power of both Burp Suite and ZAP proxies and analyzed the differences and similarities between the free and pro versions of Burp and the free and open-source ZAP proxy. These tools are essential for penetration testers focused on web application security assessments but have many applications for all offensive security practitioners as well blue team practitioners and developers. After working through each of the examples and exercises in this module, attempt some web attack-focused boxes on the main Hack The Box platform and other web application security-related modules within HTB Academy to strengthen your skillsets around both of these tools. They are must-haves in your toolbox alongside Nmap, Hashcat, Wireshark, tcpdump, sqlmap, Ffuf, Gobuster, etc.

Skills Assessment - Using Web Proxies

We are performing internal penetration testing for a local company. As you come across their internal web applications, you are presented with different situations where Burp/ZAP may be helpful. Read each of the scenarios in the questions below, and determine the features that would be the most useful for each case. Then, use it to help you in reaching the specified goal.