Burp Suite: Other Modules

Introduction

Welcome to the Burp Suite Other Modules room!
In addition to the widely recognized Repeater and Intruder rooms, Burp Suite incorporates several lesser-known modules. These will form the focus of this room's exploration.

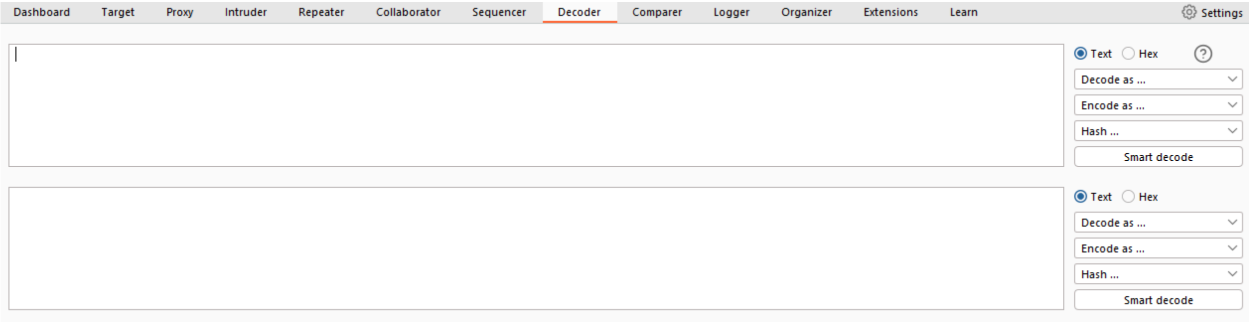
The spotlight will be on the Decoder, Comparer, Sequencer, and Organizer tools. They facilitate operations with encoded text, enable comparison of data sets, allow the analysis of randomness within captured tokens, and help you store and annotate copies of HTTP messages that you may want to revisit later. Although these tasks appear straightforward, accomplishing them within Burp Suite can substantially save time, thus emphasizing the importance of learning to use these modules effectively.

Without further ado, let's delve into the first tool, the Decoder.

Decoder: Overview

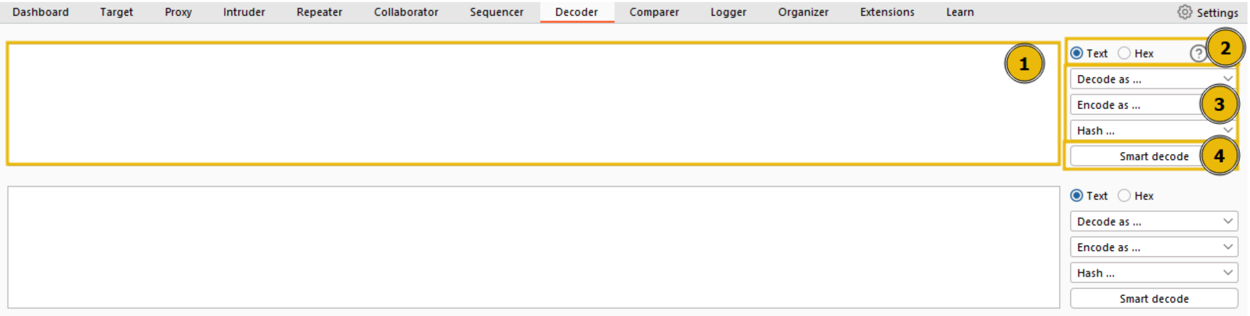
The Decoder module of Burp Suite gives user data manipulation capabilities. As implied by its name, it not only decodes data intercepted during an attack but also provides the function to encode our own data, prepping it for transmission to the target. Decoder also allows us to create hashsums of data, as well as providing a Smart Decode feature, which attempts to decode provided data recursively until it is back to being plaintext (like the "Magic" function of Cyberchef).

To access the Decoder, navigate to the Decoder tab from the top menu to view the available options:

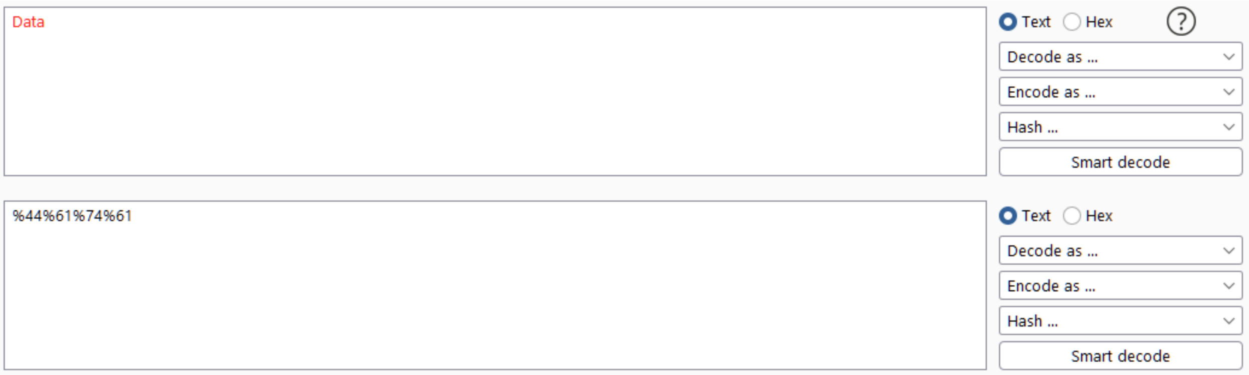


This interface lays out a multitude of options.

1. This box serves as the workspace for entering or pasting data that requires encoding or decoding. Consistent with other modules of Burp Suite, data can be moved to this area from different parts of the framework via the Send to Decoder option upon right-clicking.
2. At the top of the list on the right, there's an option to treat the input as either text or hexadecimal byte values.
3. As we move down the list, dropdown menus are present to encode, decode, or hash the input.
4. The Smart Decode feature, located at the end, attempts to auto-decode the input.



Upon entering data into the input field, the interface replicates itself to present the output of our operation. We can then choose to apply further transformations using the same options:



```
Data
```

```
%44%61%74%61
```

****

Answer the questions below:

Which feature attempts auto-decode of the input?

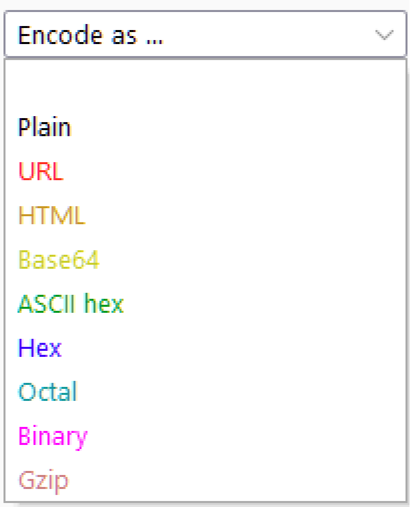
Answer: Smart decode

****

Decoder: Encoding/Decoding

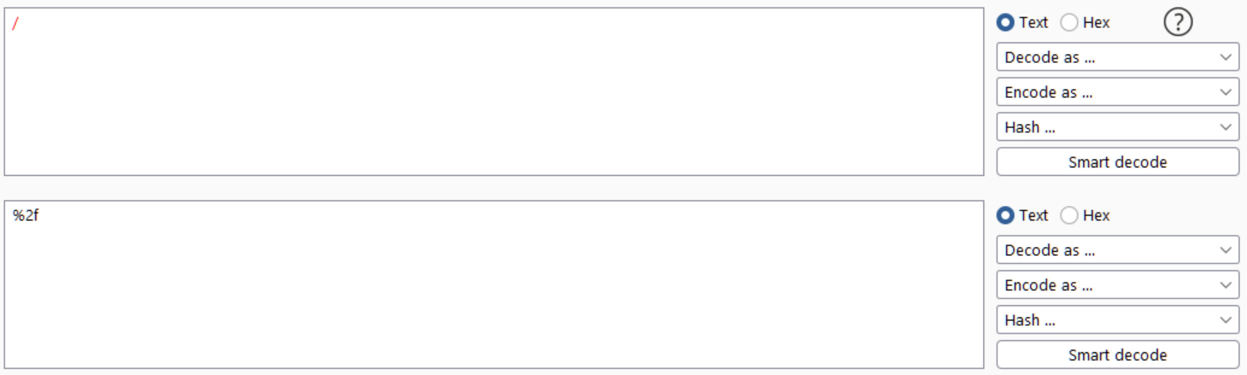
Encoding and Decoding with Decoder

Now, let's examine the manual encoding and decoding options in detail. These are identical whether the decoding or encoding menu is chosen:



- Plain: This refers to the raw text before any transformations are applied.
- URL: URL encoding is utilized to ensure the safe transfer of data in the URL of a web request. It involves substituting characters for their ASCII character code in hexadecimal format, preceded by a percentage symbol (%). This method is vital for any type of web application testing.

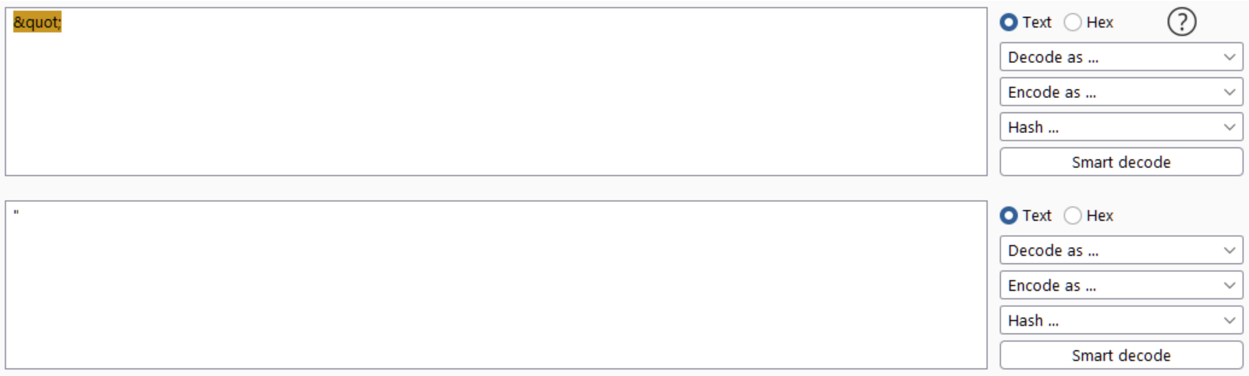
  For instance, encoding the forward-slash character (/), whose ASCII character code is 47, converts it to 2F in hexadecimal, thus becoming %2F in URL encoding. The Decoder can be used to verify this by typing a forward slash in the input box, then selectingEncode as -> URL :



```
/
```

```
%2f
```

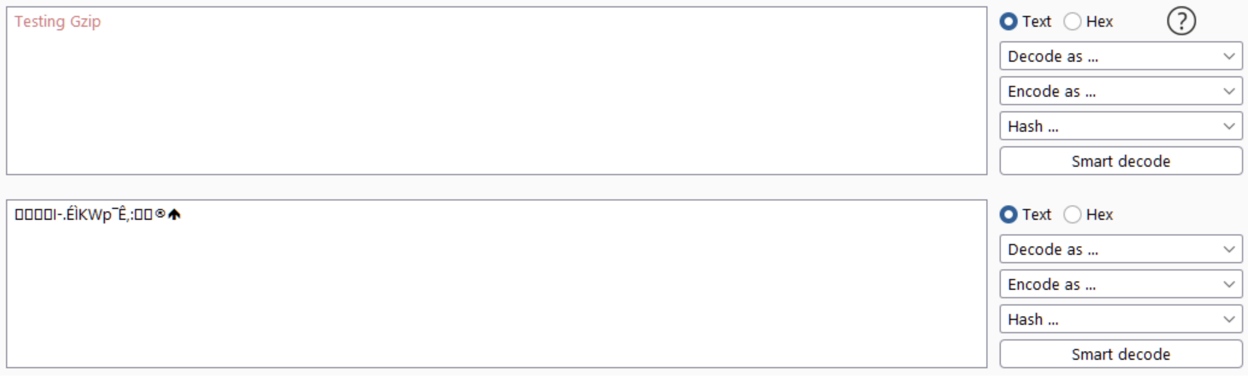
- HTML: HTML Entities encoding replaces special characters with an ampersand (&), followed by either a hexadecimal number or a reference to the character being escaped, and ending with a semicolon (;). This method ensures the safe rendering of special characters in HTML and helps prevent attacks such as XSS. The HTML option in Decoder allows any character to be encoded into its HTML escaped format or decode captured HTML entities. For instance, to decode a previously discussed quotation mark, input the encoded version and choose Decode as -> HTML:



```
&quot;
```

```
"
```

- Base64: Base64, a commonly used encoding method, converts any data into an ASCII-compatible format. The under-the-hood functioning isn't crucial at this stage; however, interested individuals can find the underlying mathematics here.
- ASCII Hex: This option transitions data between ASCII and hexadecimal representations. For instance, the word "ASCII" can be converted into the hexadecimal number "4153434949". Each character is converted from its numeric ASCII representation into hexadecimal.
- Hex, Octal, and Binary: These encoding methods apply solely to numeric inputs, converting between decimal, hexadecimal, octal (base eight), and binary representations.
- Gzip: Gzip compresses data, reducing file and page sizes before browser transmission. Faster load times are highly desirable for developers looking to enhance their SEO score and avoid user inconvenience. Decoder facilitates the manual encoding and decoding of gzip data, although it often isn't valid ASCII/Unicode. For instance:



```
Testing Gzip
```

```
□□□□I-.ÉÌKWp¯Ê,:□□®🡅
```

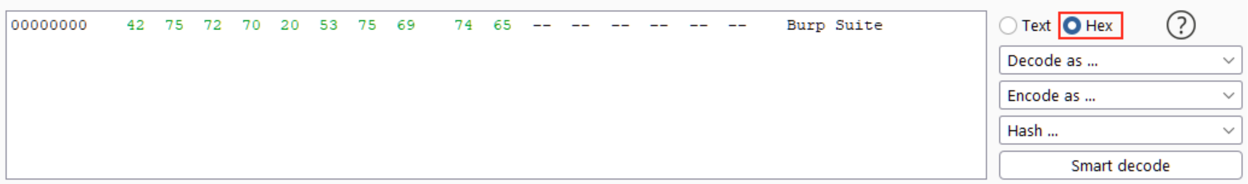
These methods can be stacked. For example, a phrase ("Burp Suite Decoder") could be converted to ASCII Hex and then to octal.

In combination, these methods grant us substantial control over the data we are encoding or decoding.

Each encoding/decoding method is color-coded, enabling swift identification of the applied transformation.

Hex Format

While inputting data in ASCII format is beneficial, there are times when byte-by-byte input editing is necessary. This is where "Hex View" proves useful, selectable above the decoding options:



```
00000000    42  75  72  70  20  53  75  69    74  65  --  --  --  --  --  --      Burp Suite
```

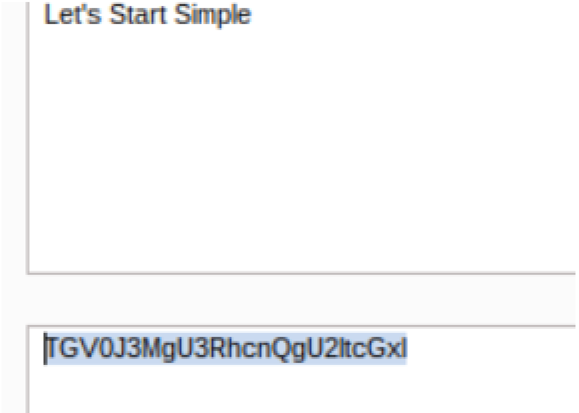
This feature enables us to view and alter our data in hexadecimal byte format, a vital tool when working with binary files or other non-ASCII data.

Smart Decode

Lastly, we have the Smart decode option. This feature tries to auto-decode encoded text. For instance, &#x42;&#x75;&#x72;&#x70;&#x20;&#x53;&#x75;&#x69;&#x74;&#x65; is automatically recognized as HTML encoded and is accordingly decoded.
While not perfect, this feature can be a quick solution for decoding unknown data chunks.
****

Answer the questions below:

Base64 encode the phrase: Let's Start Simple.
What is the base64 encoded version of this text?

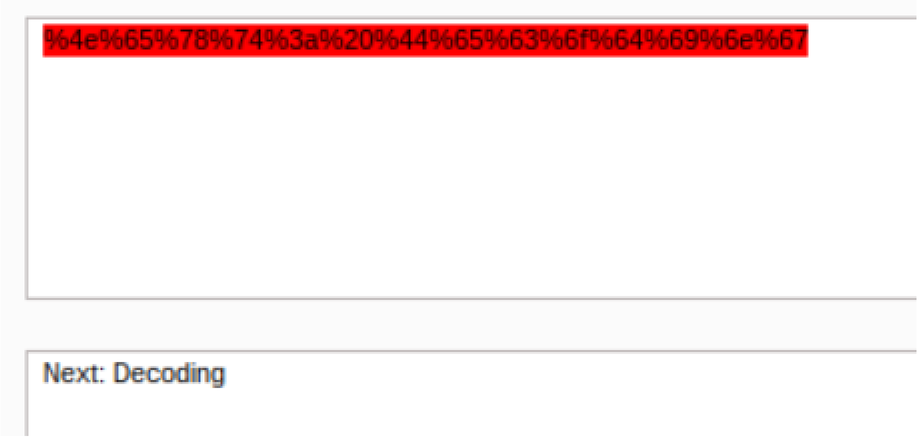


```
Let's Start Simple
```

```
TGV0J3MgU3RhcnQgU2ltcGxl
```

Answer: TGV0J3MgU3RhcnQgU2ltcGxl

URL Decode this data:
%4e%65%78%74%3a%20%44%65%63%6f%64%69%6e%67.
What is the plaintext returned?

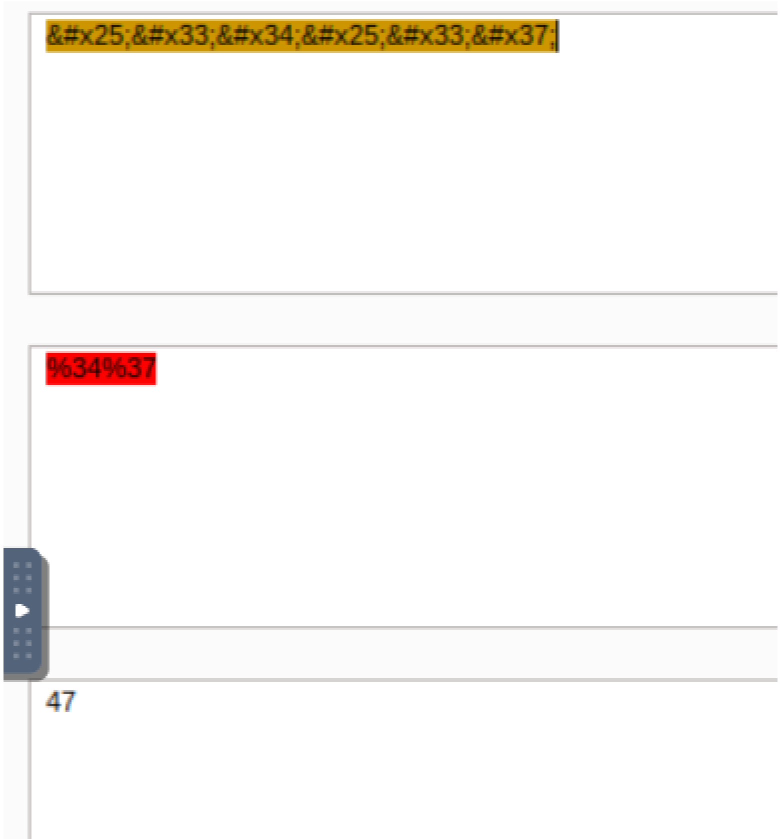


```
%4e%65%78%74%3a%20%44%65%63%6f%64%69%6e%67
```

```
Next: Decoding
```

Answer: Next: Decoding

Use Smart decode to decode this data: &#x25;&#x33;&#x34;&#x25;&#x33;&#x37;.
What is the decoded text?



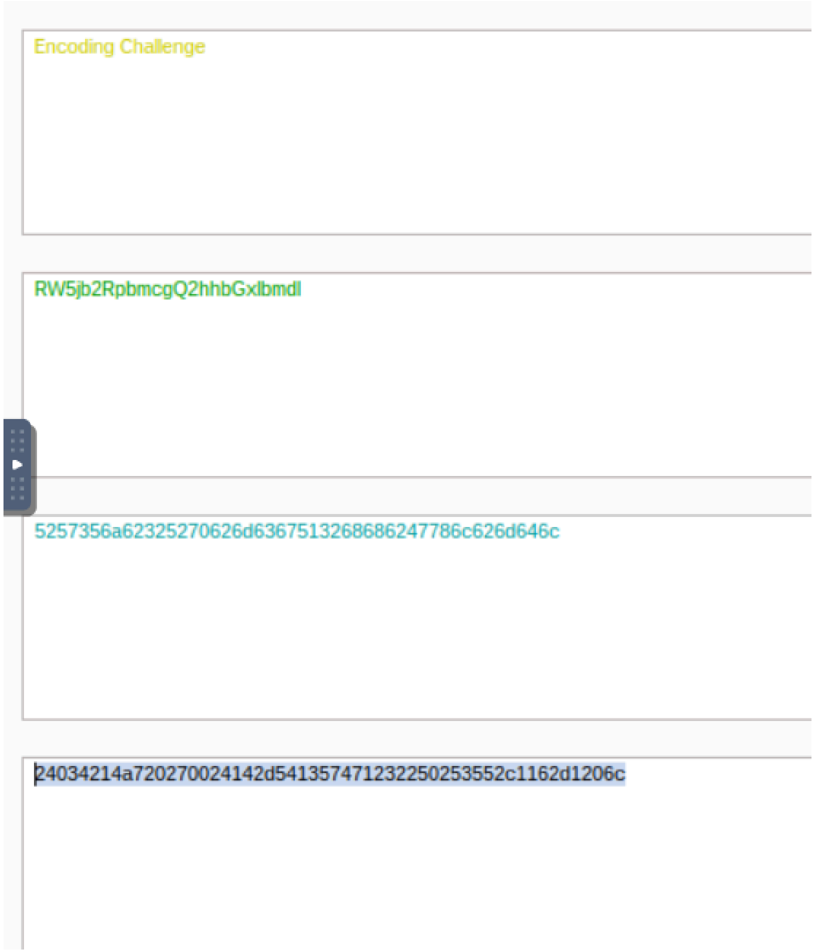
```
&#x25;&#x33;&#x34;&#x25;&#x33;&#x37;
```

```
%34%37
```

```
47
```

Answer: 47

Encode this phrase: Encoding Challenge.
Start with base64 encoding. Take the output of this and convert it into ASCII Hex.
Finally, encode the hex string into octal.
What is the final string?



```
Encoding Challenge

RW5jb2RpbmcgQ2hhbGxlbmdl

5257356a62325270626d636751326868624778626d646c

24034214a720270024142d5413574712322502533552c1162d1206c
```

Answer: 24034214a720270024142d5413574712322502533552c1162d1206c
****

Decoder: Hashing

In addition to its Encoding/Decoding functionality, Decoder also offers the ability to generate hashsums for our data.

Theory

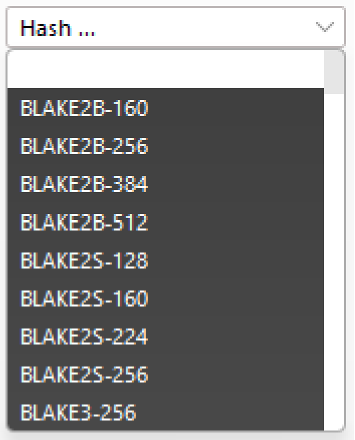
Hashing is a one-way process that transforms data into a unique signature. For a function to qualify as a hashing algorithm, the output it generates must be irreversible. A proficient hashing algorithm ensures that every data input will generate a completely unique hash. For instance, using the MD5 algorithm to produce a hashsum for the text "MD5sum" returns 4ae1a02de5bd02a5515f583f4fca5e8c. Using the same algorithm for "MD5SUM" yields an entirely different hash despite the close resemblance of the input: 13b436b09172400c9eb2f69fbd20adad. Therefore, hashes are commonly used to verify the integrity of files and documents, as even a tiny alteration to the file significantly changes the hashsum.

Note: The MD5 algorithm is deprecated and should not be used for contemporary applications.

Moreover, hashes are used to securely store passwords since the one-way hashing process makes the passwords relatively secure, even if the database is compromised. When a user creates a password, the application hashes and stores it. During login, the application hashes the submitted password and compares it against the stored hash; if they match, the password is correct. Using this method, an application never needs to store the original (plaintext) password.

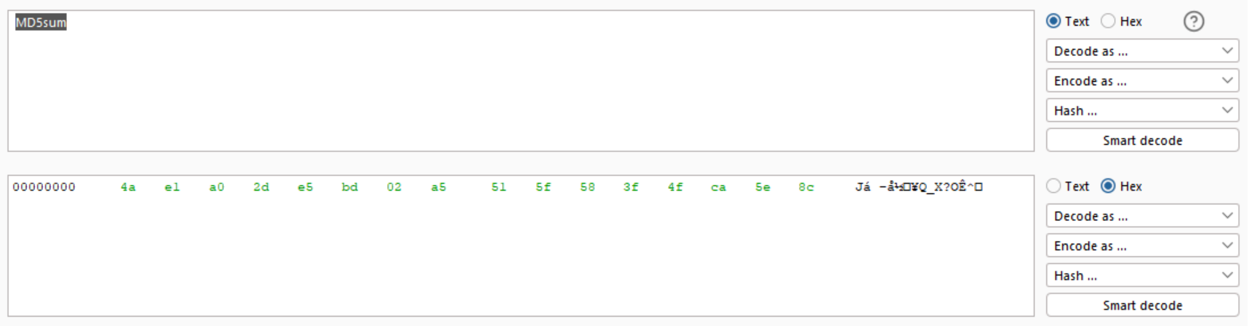
Hashing in Decoder

Decoder allows us to create hashsums for data directly within Burp Suite; it operates similarly to the encoding/decoding options we discussed earlier. Specifically, we click on the Hash dropdown menu and select an algorithm from the list:



Note: This list is significantly longer than the encoding/decoding algorithms – it's worth scrolling through to see the many available hashing algorithms.

Continuing with our earlier example, let's enter "MD5sum" into the input box, then scroll down the list until we find "MD5". Applying this automatically takes us into the Hex view:



MD5sum

```
00000000     4a  e1  a0  2d  e5  bd  02  a5   51  5f  58  3f  4f  ca  5e  8c    Já -å½□¥Q_X?OÊ^□
```

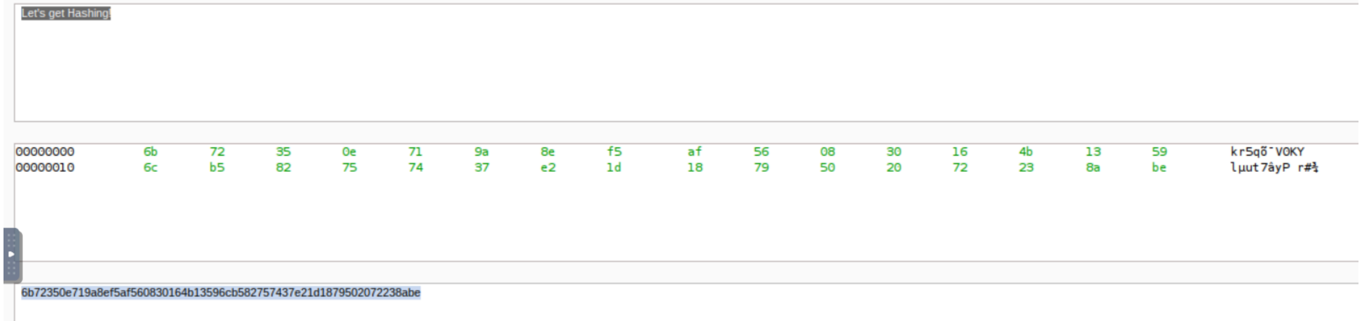
A hashing algorithm's output does not yield pure ASCII/Unicode text. Hence, it's customary to convert the algorithm's output into a hexadecimal string; this is the "hash" form you might be familiar with.

Let's complete this by applying an "ASCII Hex" encoding to the hashsum to create the neat hex string from our initial example.

****

Answer the questions below:

Using Decoder, what is the SHA-256 hashsum of the phrase: Let's get Hashing!? Convert this into an ASCII Hex string for the answer to this question.



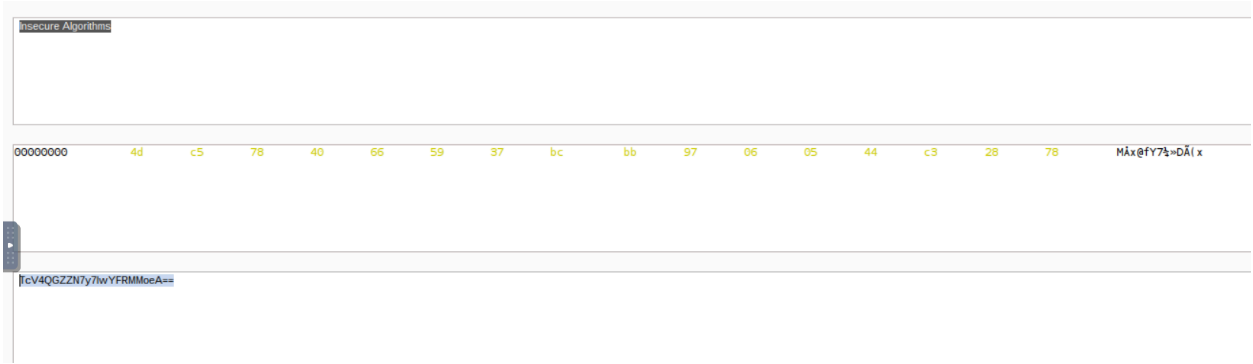
Let's get Hashing!

```
00000000     6b   72   35   0e   71   9a   8e   f5     af   56   08   30   16   4b   13   59     kr5qõ¯V0KY
00000010     6c   b5   82   75   74   37   e2   1d     18   79   50   20   72   23   8a   be     lµut7âyP r#¾
```

6b72350e719a8ef5af560830164b13596cb582757437e21d1879502072238abe

Answer: 6b72350e719a8ef5af560830164b13596cb582757437e21d1879502072238abe

Generate an MD4 hashsum of the phrase: Insecure Algorithms.
Encode this as base64 (not ASCII Hex) before submitting.



```
Insecure Algorithms
```

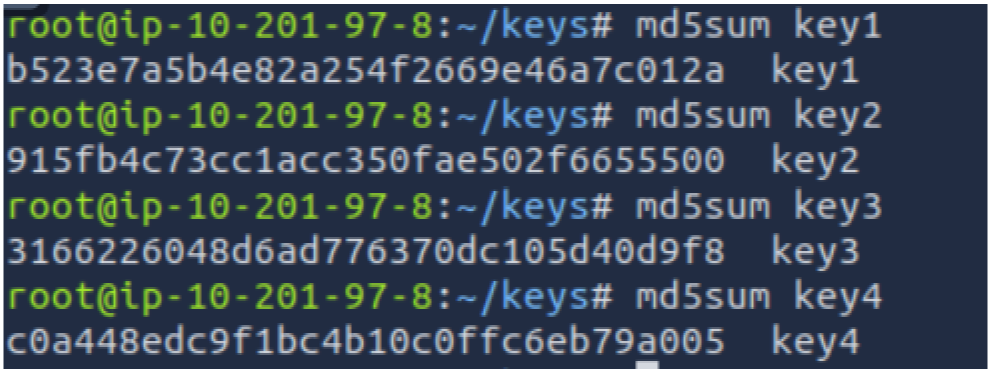
```
00000000    4d    c5    78    40    66    59    37    bc    bb    97    06    05    44    c3    28    78    MÅx@fY7¼»DÃ( x
```

```
TcV4QGZZN7y7lwYFRMMoeA==
```

Answer: TcV4QGZZN7y7lwYFRMMoeA==

Let's look at an in-context example:
First, download the file attached to this task.
Note: This file can also be downloaded from the deployed VM with wget http://10.201.21.46:9999/AlteredKeys.zip — you may find this helpful if you are using the AttackBox.

Now read the problem specification below:
"Some joker has messed with my SSH key! There are four keys in the directory, and I have no idea which is the real one. The MD5 hashsum for my key is 3166226048d6ad776370dc105d40d9f8 — could you find it for me?"
What is the correct key name?



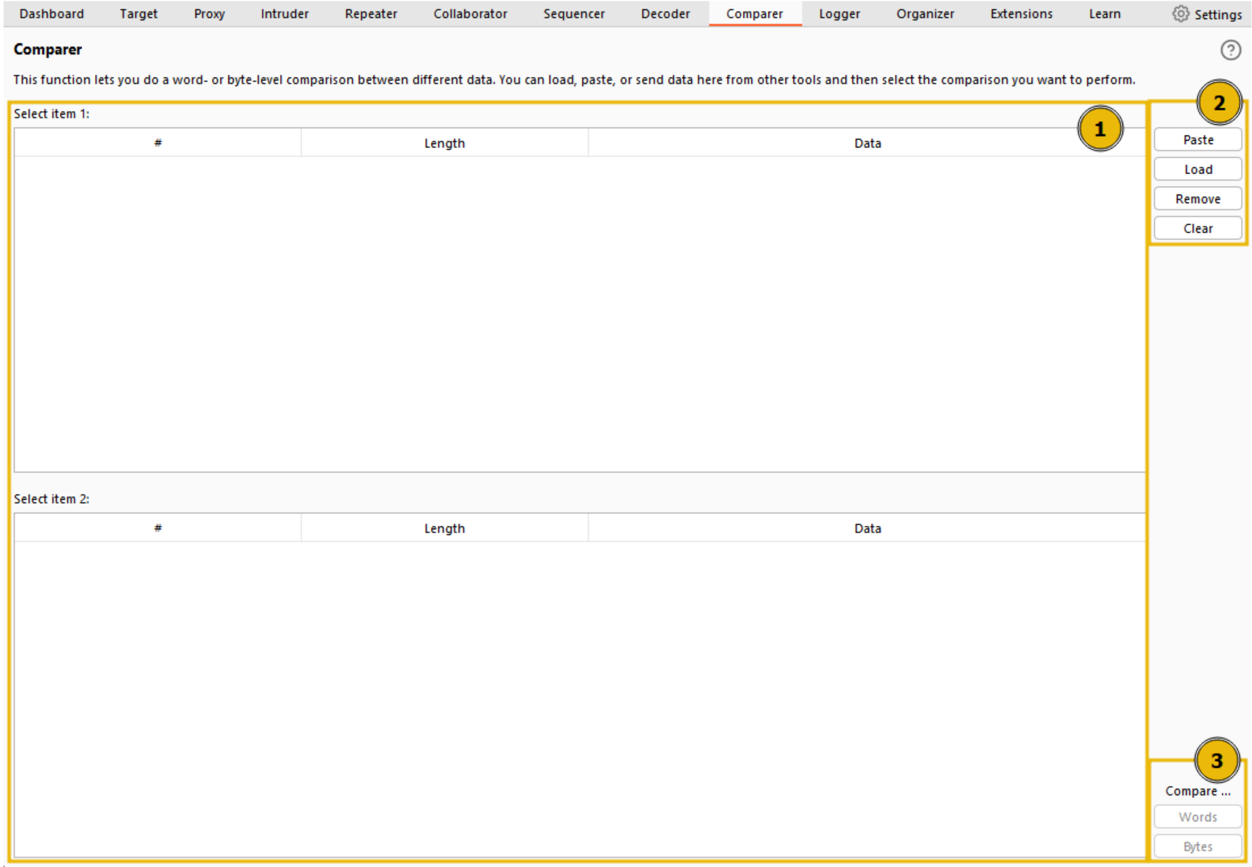
```
root@ip-10-201-97-8:~/keys# md5sum key1
b523e7a5b4e82a254f2669e46a7c012a  key1
root@ip-10-201-97-8:~/keys# md5sum key2
915fb4c73cc1acc350fae502f6655500  key2
root@ip-10-201-97-8:~/keys# md5sum key3
3166226048d6ad776370dc105d40d9f8  key3
root@ip-10-201-97-8:~/keys# md5sum key4
c0a448edc9f1bc4b10c0ffc6eb79a005  key4
```

Answer: key3
****

Comparer: Overview

Comparer, as the name implies, lets us compare two pieces of data, either by ASCII words or by bytes.

Let's first check out the interface:

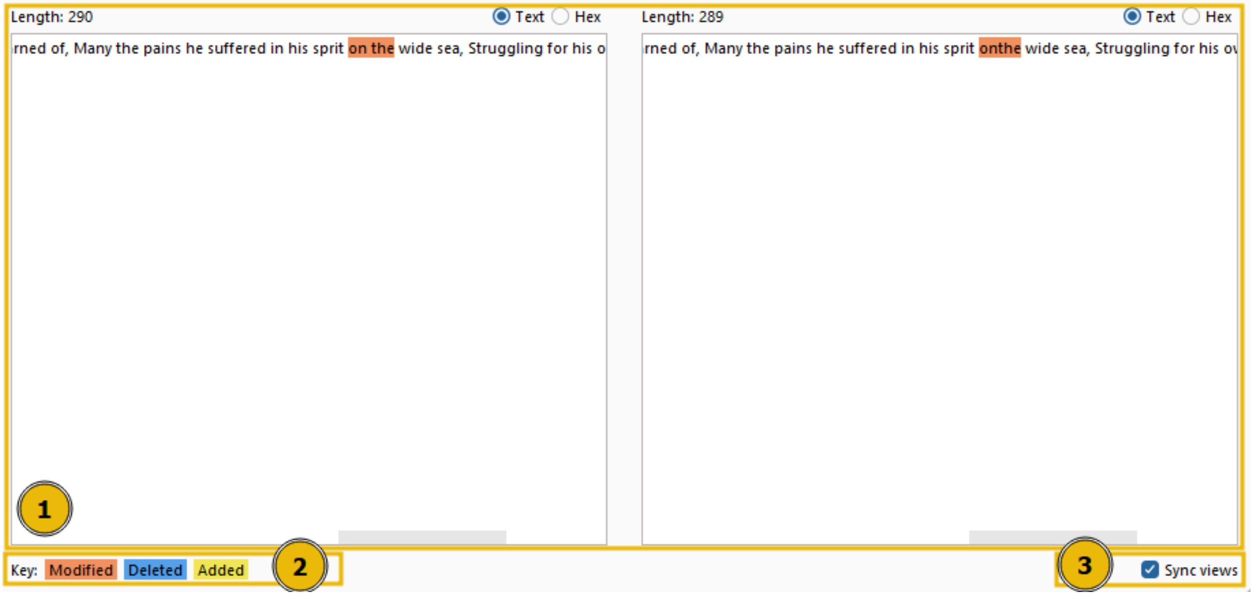


The interface can be divided into three main sections:

1. On the left, we see the items to be compared. When we load data into Comparer, it appears as rows in these tables. We select two datasets to compare.
2. On the upper right, we have options for pasting data from the clipboard (Paste), loading data from a file (Load), removing the current row (Remove), and clearing all datasets (Clear).
3. Lastly, on the lower right, we can choose to compare our datasets by either words or bytes. It doesn't matter which of these buttons you select initially because this can be changed later. These are the buttons we click when we're ready to compare the selected data.

Just like most Burp Suite modules, we can also load data into Comparer from other modules by right-clicking and choosing Send to Comparer.

Once we've added at least 2 datasets to compare and press on either Words or Bytes, a pop-up window shows us the comparison:



Length: 290 ● Text ○ Hex

rned of, Many the pains he suffered in his sprit on the wide sea, Struggling for his o

Length: 289 ● Text ○ Hex

rned of, Many the pains he suffered in his sprit onthe wide sea, Struggling for his ov

Key: Modified Deleted Added

☑ Sync views

This window also has three distinct sections:

1. The compared data occupies most of the window; it can be viewed in either text or hex format. The initial format depends on whether we chose to compare by words or bytes in the previous window, but this can be overridden by using the buttons above the comparison boxes.
2. The comparison key is at the bottom left, showing which colors represent modified, deleted, and added data between the two datasets.
3. The Sync views checkbox is at the bottom right of the window. When selected, it ensures that both sets of data will sync formats. In other words, if you change one of them into Hex view, the other will adjust to match.

The window title displays the total number of differences found.

****

Answer the questions below:

Click me to proceed to the next task.

No Answer Needed

****

Comparer: Example

We now know what Comparer does, but how is this useful?

There are many situations where being able to quickly compare two (potentially very large) pieces of data can come in handy.

For example, when performing a login bruteforce or credential stuffing attack with Intruder, you may wish to compare two responses with different lengths to see where the differences lie and whether the differences indicate a successful login.

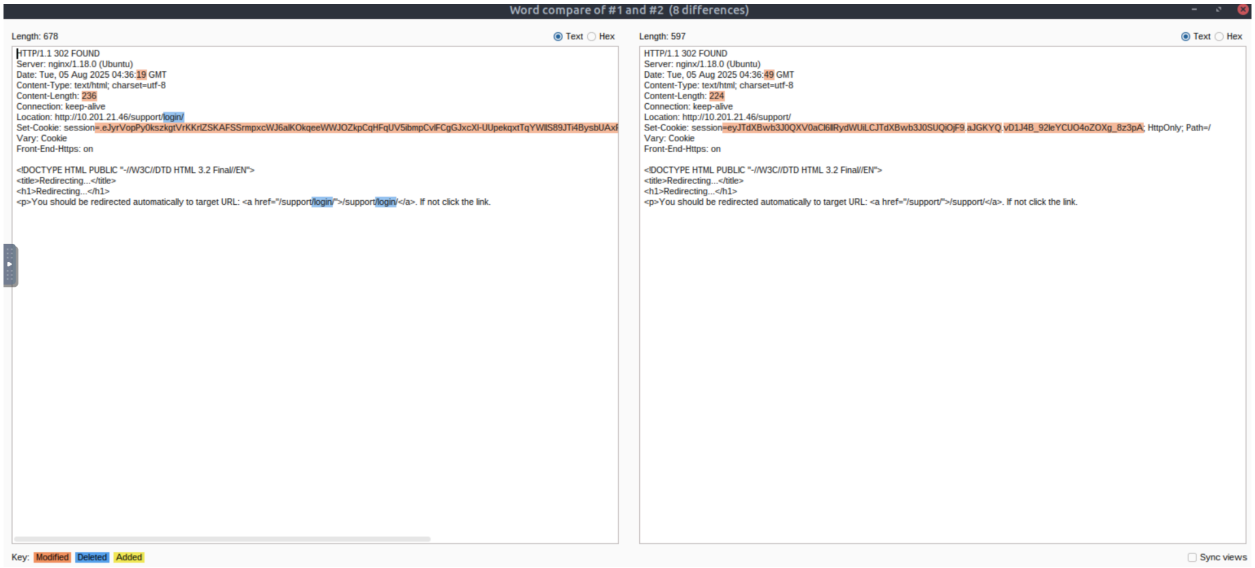
Practical Example

1. Navigate to http://10.201.21.46/support/login
2. Try to log in with an invalid username and password – capture the request in the Burp Proxy.
3. Send the request to Repeater with Ctrl + R (or Mac equivalent) or by right-clicking on the request in the Proxy module and choosing Send to Repeater.
4. Send the request, then right-click on the response and choose Send to Comparer.
5. In the Repeater tab, change the credentials to:
    - Username: support_admin
    - Password: w58ySK4W

Send the request again, then pass the new response to Comparer.
****

Answer the questions below:

Compare the two responses by word. Can you identify the main differences?



Length: 678 ● Text ○ Hex

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 05 Aug 2025 04:36:19 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 236
Connection: keep-alive
Location: http://10.201.21.46/support/login/
Set-Cookie: session=.eJyrVopPy0kszkgtVrKKrlZSKAFSSrmpxcWJ6alKOkqeeWWJOZkpCqHFqUV5ibmpCvlFCgGJxcXl-UUpekqxtTqYWllS89JTi4BysbUAx
Vary: Cookie
Front-End-Https: on

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="/support/login/">/support/login/</a>. If not click the link.
```

Length: 597 ● Text ○ Hex

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 05 Aug 2025 04:36:49 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: keep-alive
Location: http://10.201.21.46/support/
Set-Cookie: session=eyJTdXBwb3J0QXV0aCI6IlRydWUiLCJTdXBwb3J0SUQiOjF9.aJGKYQ.vD1J4B_92leYCUO4oZOXg_8z3pA; HttpOnly; Path=/
Vary: Cookie
Front-End-Https: on

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="/support/">/support/</a>. If not click the link.
```

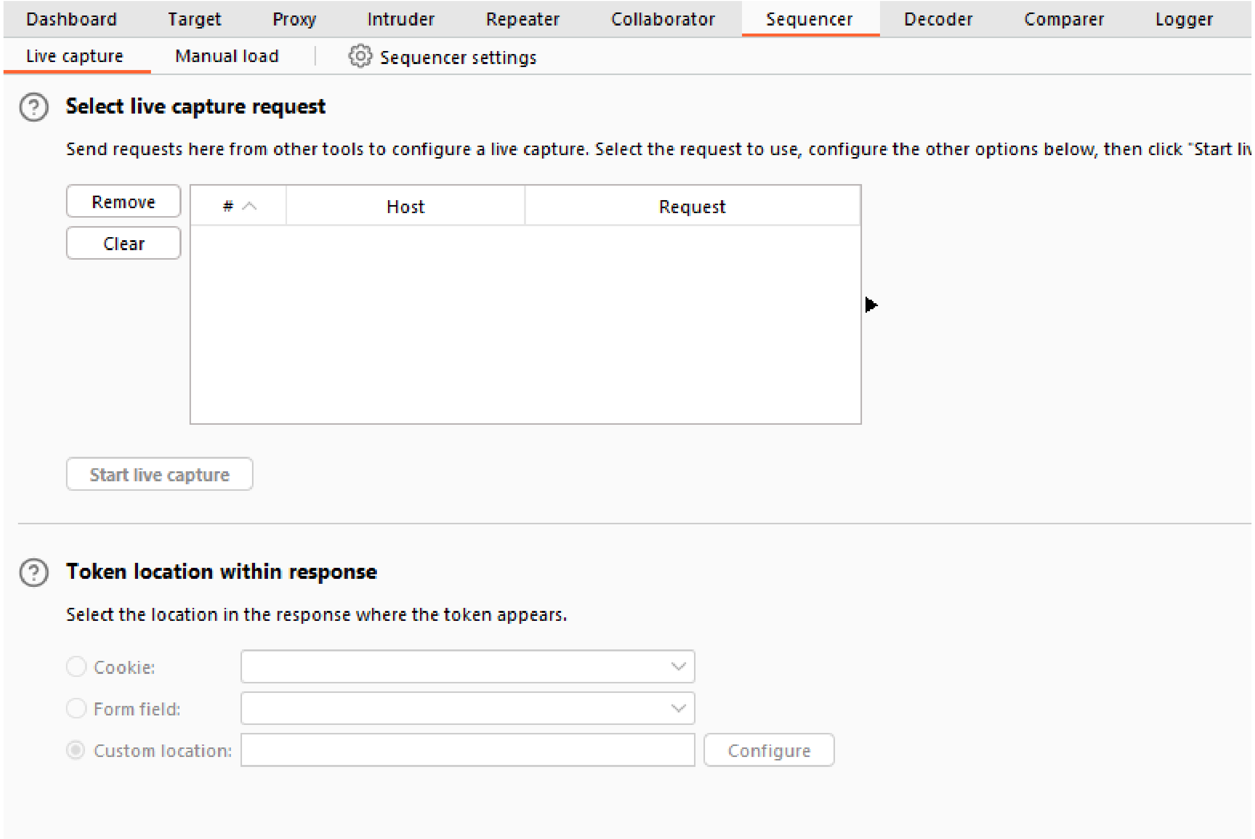
Key: Modified Deleted Added ☐ Sync views

No Answer Needed
****

Sequencer: Overview

Sequencer allows us to evaluate the entropy, or randomness, of "tokens". Tokens are strings used to identify something and should ideally be generated in a cryptographically secure manner. These tokens could be session cookies or Cross-Site Request Forgery (CSRF) tokens used to protect form submissions. If these tokens aren't generated securely, then, in theory, we could predict upcoming token values. The implications could be substantial, for instance, if the token in question is used for password resets.

Let's start by looking at the Sequencer interface:



Select live capture request

Send requests here from other tools to configure a live capture. Select the request to use, configure the other options below, then click "Start liv

Token location within response

Select the location in the response where the token appears.

We have two main ways to perform token analysis with Sequencer:

- Live Capture: This is the more common method and is the default sub-tab for Sequencer. Live capture lets us pass a request that will generate a token to Sequencer for analysis. For instance, we might want to pass a POST request to a login endpoint to Sequencer, knowing that the server will respond with a cookie. With the request passed in, we can instruct Sequencer to start a live capture. It will then automatically make the same request thousands of times, storing the generated token samples for analysis. After collecting enough samples, we stop the Sequencer and allow it to analyze the captured tokens.
- Manual Load: This allows us to load a list of pre-generated token samples directly into Sequencer for analysis. Using Manual Load means we don't need to make thousands of requests to our target, which can be noisy and resource-intensive. However, it does require that we have a large list of pre-generated tokens.

We will be focusing on live captures in this room.
****

Answer the questions below:

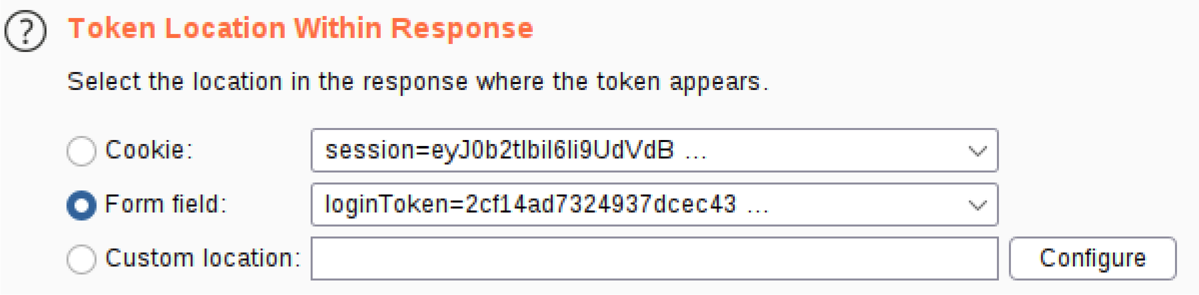
What does Sequencer allow us to evaluate?
Answer: Entropy
****

Sequencer: Live Capture

Great, let's dive into the process of using the Sequencer's live capture for entropy analysis on the anti-bruteforce token used in the admin login form.

First, capture a request to http://10.201.21.46/admin/login/ in the Proxy. Right-click on the request and select Send to Sequencer.

In the "Token Location Within Response" section, we can select between Cookie, Form field, and Custom location. Since we're testing the loginToken in this case, select the "Form field" radio button and choose the loginToken from the dropdown menu:



Token Location Within Response

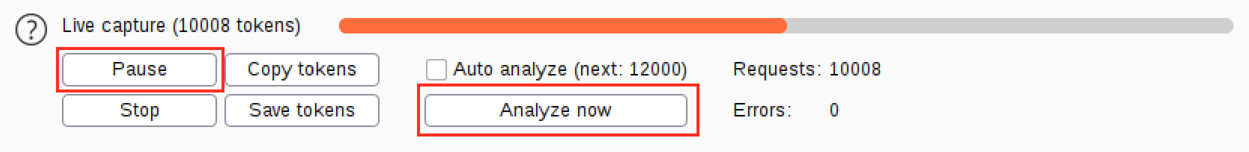
Select the location in the response where the token appears.

○ Cookie: session=eyJ0b2tlbiI6Ii9UdVdB ...
● Form field: loginToken=2cf14ad7324937dcec43 ...
○ Custom location:

In this situation, we can safely leave all other options at their default values. So, click on the Start live capture button.

A new window will pop up indicating that a live capture is in progress and displaying the number of tokens captured so far. Wait until a sufficient number of tokens are captured (approximately 10,000 should suffice); the more tokens we have, the more precise our analysis will be.

Once around 10,000 tokens are captured, click on Pause and then select the Analyze now button:



Live capture (10008 tokens)

□ Auto analyze (next: 12000)

Requests: 10008
Errors: 0

It's important to note that we could have also chosen to Stop the capture. However, by opting to pause, we keep the option to resume the capture later if the report doesn't have enough samples to accurately calculate the token's entropy.

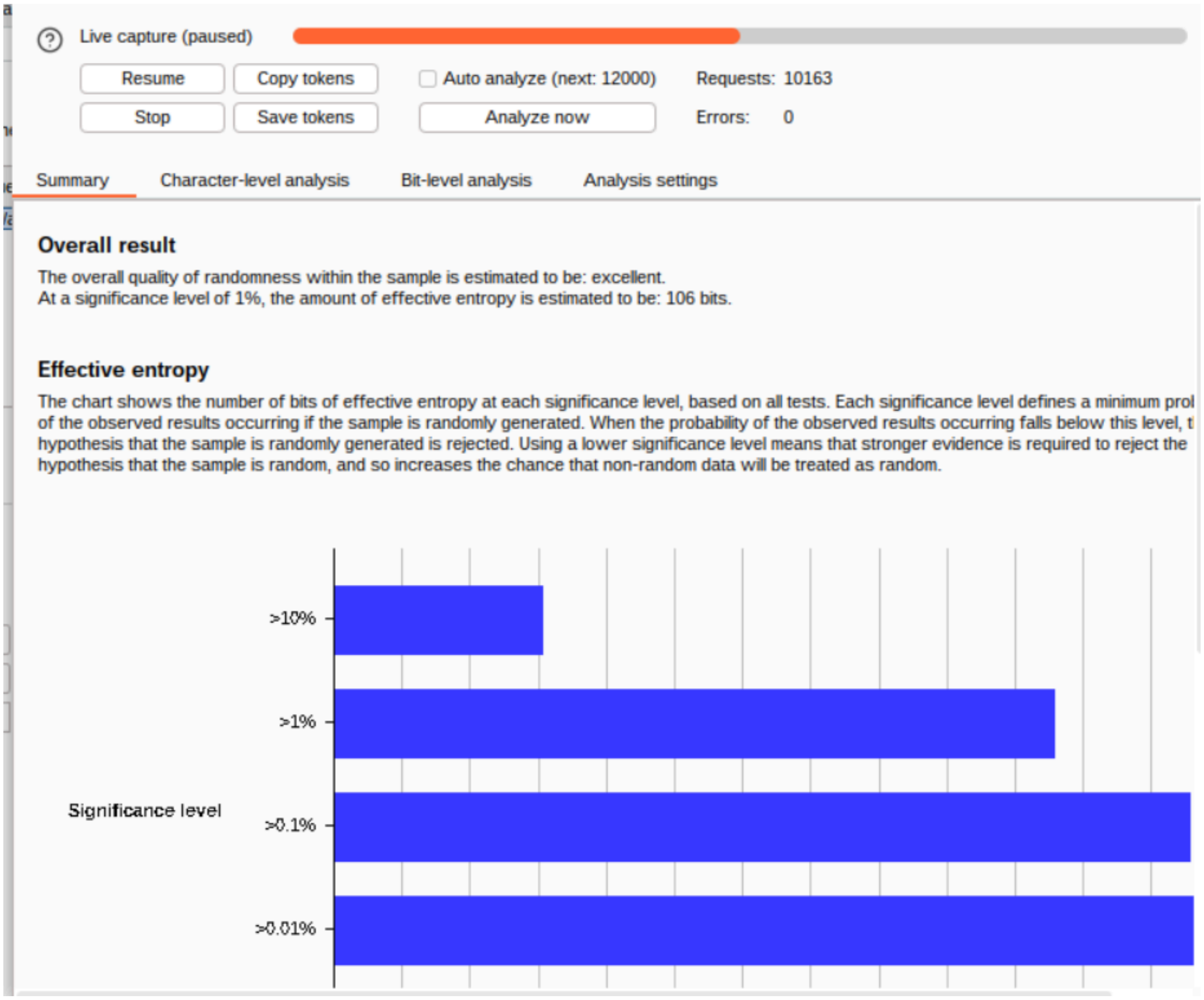
If we wished for periodic updates on the analysis, we could have also selected the "Auto analyze" checkbox. This option tells Burp to perform the entropy analysis after every 2000 requests, providing frequent updates that will become increasingly accurate as more samples are loaded into Sequencer.

At this point, it's also worth noting that we could choose to copy or save the captured tokens for further analysis at a later time.

Upon clicking the Analyze now button, Burp will analyze the token's entropy and generate a report.
****

Answer the questions below:

What is the overall quality of randomness estimated to be?



Overall result

The overall quality of randomness within the sample is estimated to be: excellent.
At a significance level of 1%, the amount of effective entropy is estimated to be: 106 bits.

Effective entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum prol of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, t hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

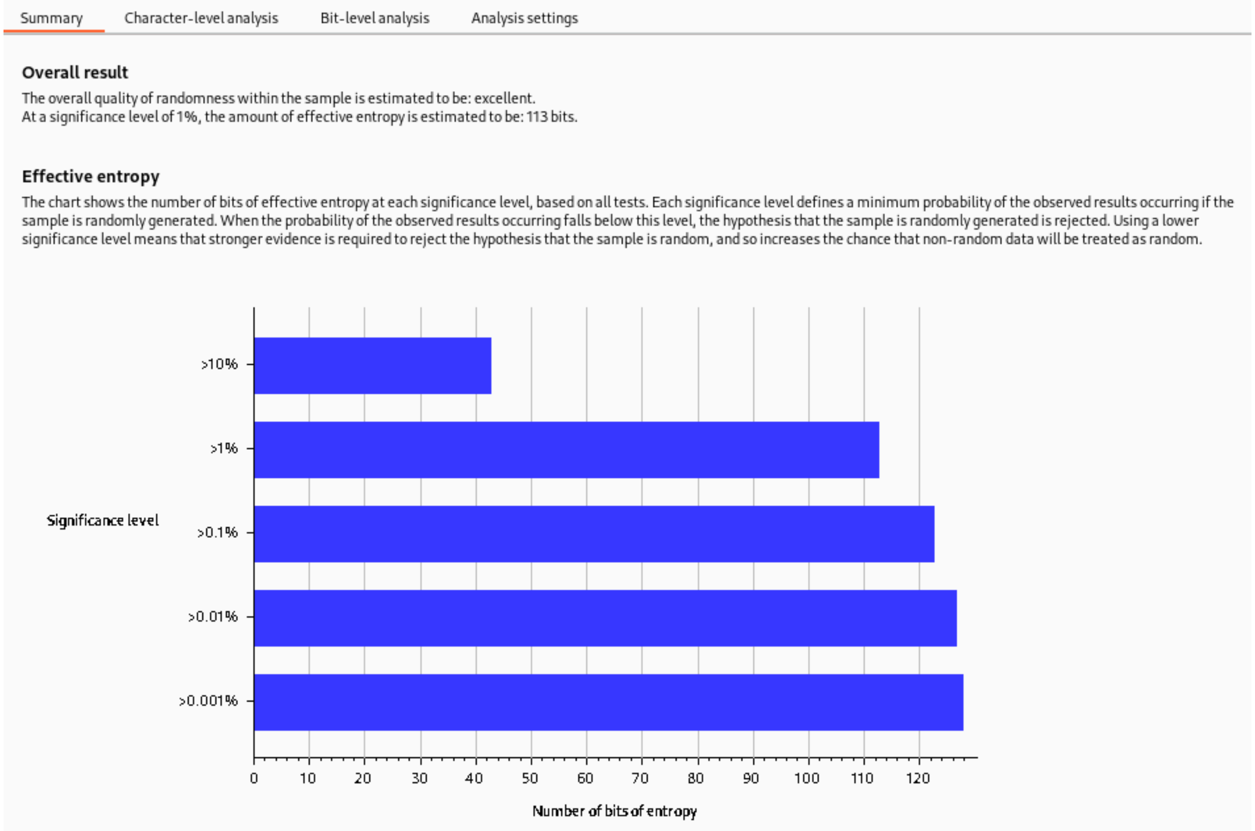
Answer: excellent

****

Sequencer: Analysis

Now that we have a report for the entropy analysis of our token, it's time to analyze it!

The generated entropy analysis report is split into four primary sections. The first of these is the Summary of the results. The summary gives us the following:



Overall result

The overall quality of randomness within the sample is estimated to be: excellent.
At a significance level of 1%, the amount of effective entropy is estimated to be: 113 bits.

Effective entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

- Overall result: This gives a broad assessment of the security of the token generation mechanism. In this case, the level of entropy indicates that the tokens are likely securely generated.
- Effective entropy: This measures the randomness of the tokens. The effective entropy of 117 bits is relatively high, indicating that the tokens are sufficiently random and, therefore, secure against prediction or brute force attacks.
- Reliability: The significance level of 1% implies that there is 99% confidence in the accuracy of the results. This level of confidence is quite high, providing assurance in the accuracy of the effective entropy estimation.
- Sample: This provides details about the token samples analyzed during the entropy testing process, including the number of tokens and their characteristics.

While the summary report often provides enough information to assess the security of the token generation process, it's important to remember that further investigation may be necessary in some cases. The character-level and bit-level analysis can provide more detailed insights into the randomness of the tokens, especially when the summary results raise potential concerns.

While the entropy report can provide a strong indicator of the security of the token generation mechanism, there needs to be more definitive proof. Other factors could also

impact the security of the tokens, and the nature of probability and statistics means there's always a degree of uncertainty. That said, an effective entropy of 117 bits with a significance level of 1% suggests a robustly secure token generation process.
****

Answer the questions below:

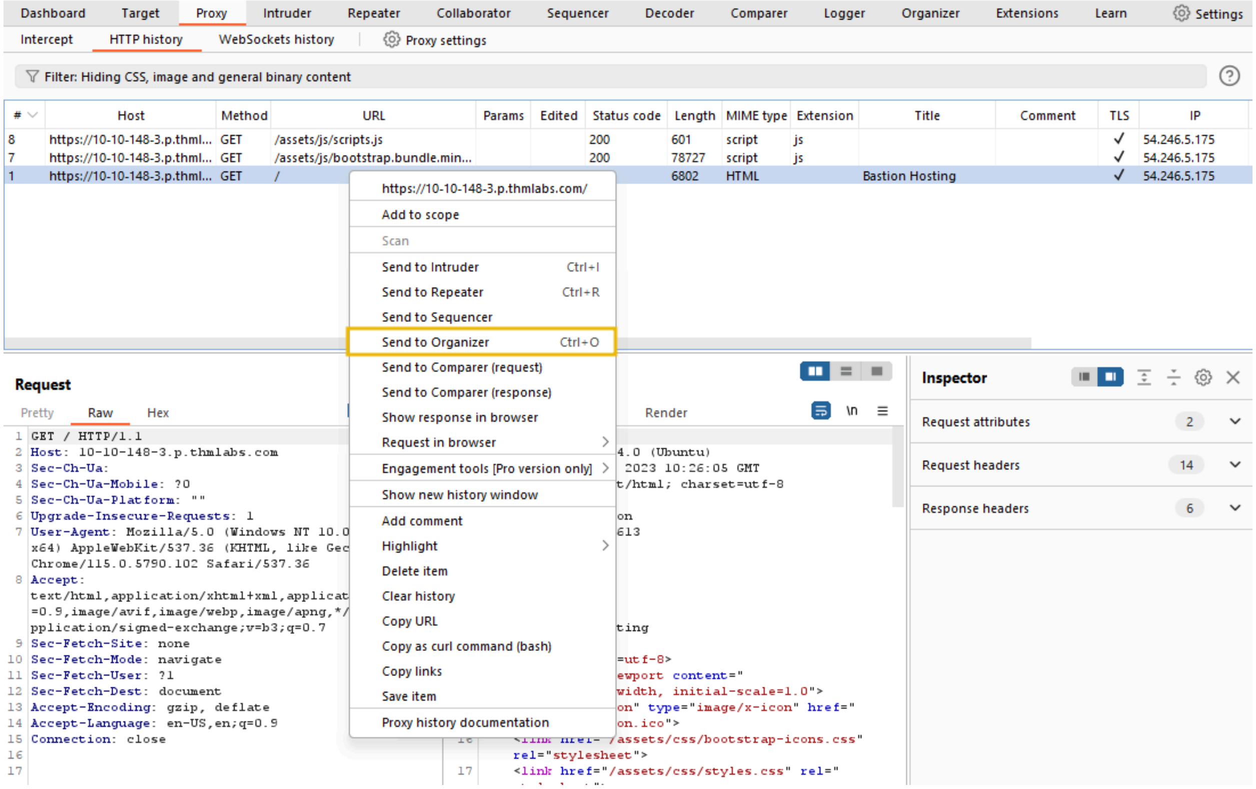
Click me to proceed to the next task.

No Answer Needed
****

Organizer: Overview

The Organizer module of Burp Suite is designed to help you store and annotate copies of HTTP requests that you may want to revisit later. This tool can be particularly useful for organizing your penetration testing workflow. Here are some of its key features:

- You can store requests that you want to investigate later, save requests that you've already identified as interesting, or save requests that you want to add to a report later.
- You can send HTTP requests to Burp Organizer from other Burp Modules such as Proxy or Repeater. You can do this by right-clicking the request and selecting Send to Organizer or using the default hotkey Ctrl + O. Each HTTP request that you send to Organizer is a read-only copy of the original request saved at the point you sent it to Organizer.



#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Comment	TLS	IP
8	https://10-10-148-3.p.thml...	GET	/assets/js/scripts.js			200	601	script	js			✓	54.246.5.175
7	https://10-10-148-3.p.thml...	GET	/assets/js/bootstrap.bundle.min...			200	78727	script	js			✓	54.246.5.175
1	https://10-10-148-3.p.thml...	GET	/				6802	HTML		Bastion Hosting		✓	54.246.5.175

```
GET / HTTP/1.1
Host: 10-10-148-3.p.thmlabs.com
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0
x64) AppleWebKit/537.36 (KHTML, like Gec
Chrome/115.0.5790.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,applicat
=0.9,image/avif,image/webp,image/apng,*/
pplication/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

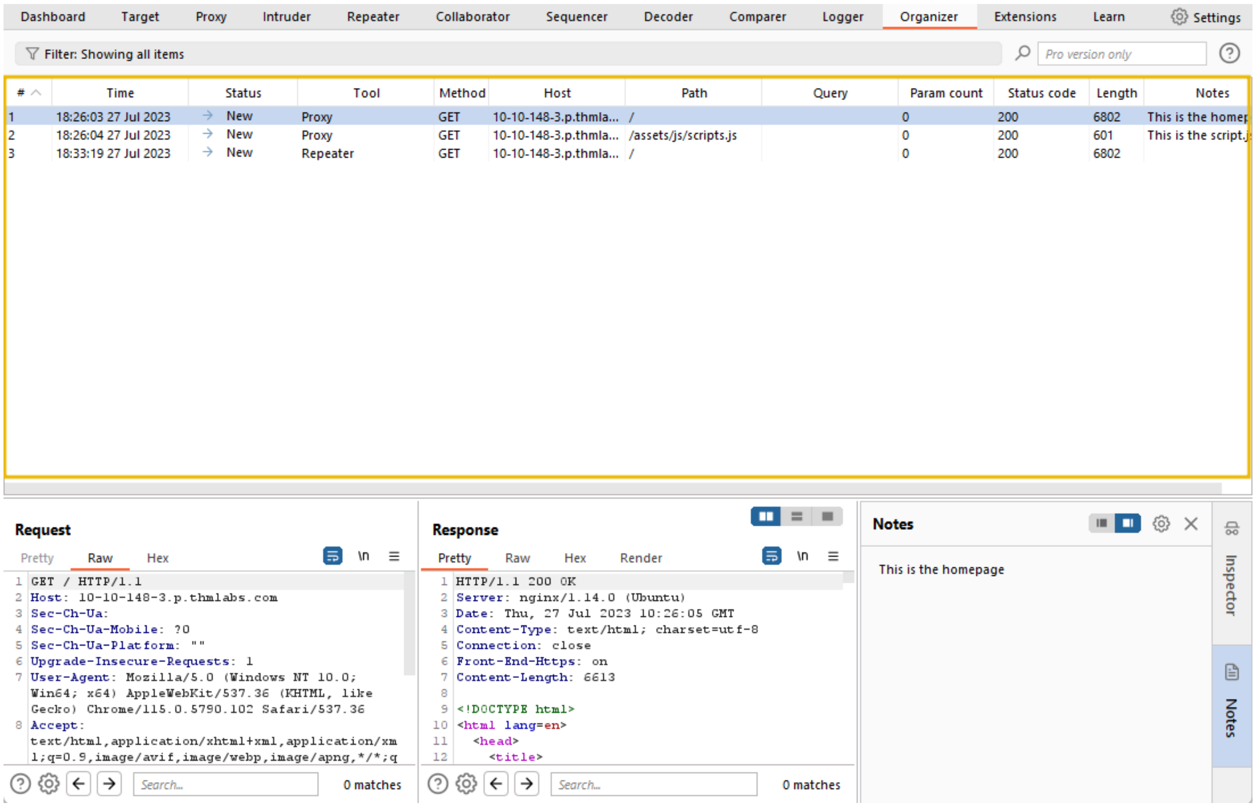
```
4.0 (Ubuntu)
 2023 10:26:05 GMT
t/html; charset=utf-8

on
613

ting

=utf-8>
ewport content="
width, initial-scale=1.0">
on" type="image/x-icon" href="
on.ico">
    <link href="/assets/css/bootstrap-icons.css"
    rel="stylesheet">
    <link href="/assets/css/styles.css" rel="
```

- Requests are stored in a table, which contains columns such as the request index number, the time the request was made, workflow status, Burp tool that the request was sent from, HTTP method, server hostname, URL file path, URL query string, number of parameters in the request, HTTP status code of the response, length of the response in bytes, and any notes that you have made.



#	Time	Status	Tool	Method	Host	Path	Query	Param count	Status code	Length	Notes
1	18:26:03 27 Jul 2023	→ New	Proxy	GET	10-10-148-3.p.thmla...	/		0	200	6802	This is the homep
2	18:26:04 27 Jul 2023	→ New	Proxy	GET	10-10-148-3.p.thmla...	/assets/js/scripts.js		0	200	601	This is the script.j
3	18:33:19 27 Jul 2023	→ New	Repeater	GET	10-10-148-3.p.thmla...	/		0	200	6802	

```
GET / HTTP/1.1
Host: 10-10-148-3.p.thmlabs.com
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/115.0.5790.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xm
l;q=0.9,image/avif,image/webp,image/apng,*/*;q
```

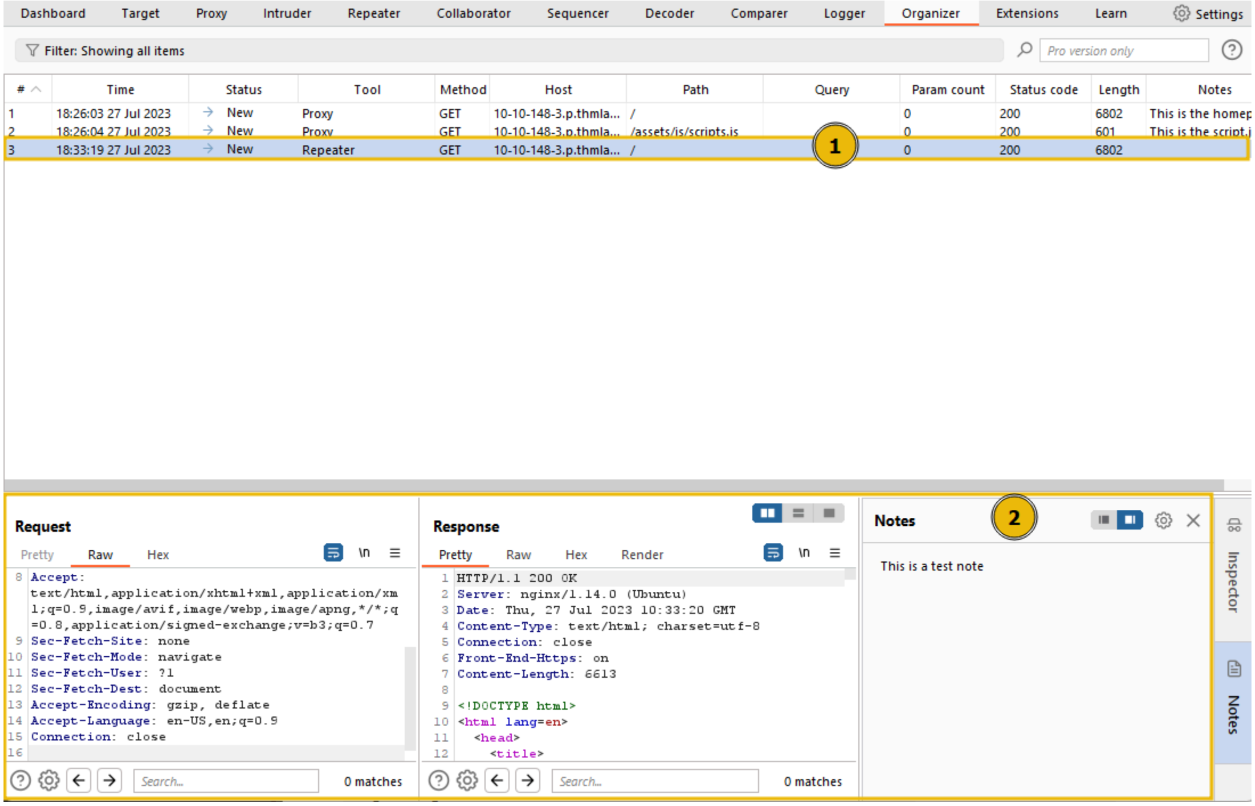
```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Thu, 27 Jul 2023 10:26:05 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Front-End-Https: on
Content-Length: 6613

<!DOCTYPE html>
<html lang=en>
  <head>
    <title>
```

This is the homepage

To view the request and response:

1. Click on any Organizer item.
2. The request and response are both read-only. You can search within the request or response, select the request, and then use the search bar below the request.



#	Time	Status	Tool	Method	Host	Path	Query	Param count	Status code	Length	Notes
1	18:26:03 27 Jul 2023	→ New	Proxy	GET	10-10-148-3.p.thmla...	/		0	200	6802	This is the homep
2	18:26:04 27 Jul 2023	→ New	Proxy	GET	10-10-148-3.p.thmla...	/assets/js/scripts.js		0	200	601	This is the script.j
3	18:33:19 27 Jul 2023	→ New	Repeater	GET	10-10-148-3.p.thmla...	/		0	200	6802	

Request

```
Accept:
text/html,application/xhtml+xml,application/xm
l;q=0.9,image/avif,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Thu, 27 Jul 2023 10:33:20 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Front-End-Https: on
Content-Length: 6613

<!DOCTYPE html>
<html lang=en>
  <head>
    <title>
```

Notes

This is a test note

****

Answer the questions below:

Are saved requests read-only? (yea/nay)

Answer: yea

****

Conclusion

Congratulations on completing the Burp Suite Other Modules room!

To summarize, Decoder allows you to encode and decode data, making it easier to read and understand the information being transferred. Comparer enables you to spot differences between two datasets, which can be pivotal in identifying vulnerabilities or anomalies. Sequencer helps in performing entropy analysis on tokens, providing insights into the randomness of their generation and, consequently, their security level. Organizer enables you to store and annotate copies of HTTP requests that you may want to revisit later.

Having a fundamental understanding of these tools equips you with a broader skillset when it comes to web application penetration testing.

In the next and final room of this module, you will be exploring the Burp Suite Extensions tool.