[Skip to main content](https://lms.alnafi.com/xblock/block-v1:alnafi+DCCS102+2025_DCCS+type@vertical+block@e2563247955c4f47ae011784c3cf05d1?exam_access=&recheck_access=1&show_bookmark=0&show_title=0&view=student_view#main)

**Play Around With Burpsuite**

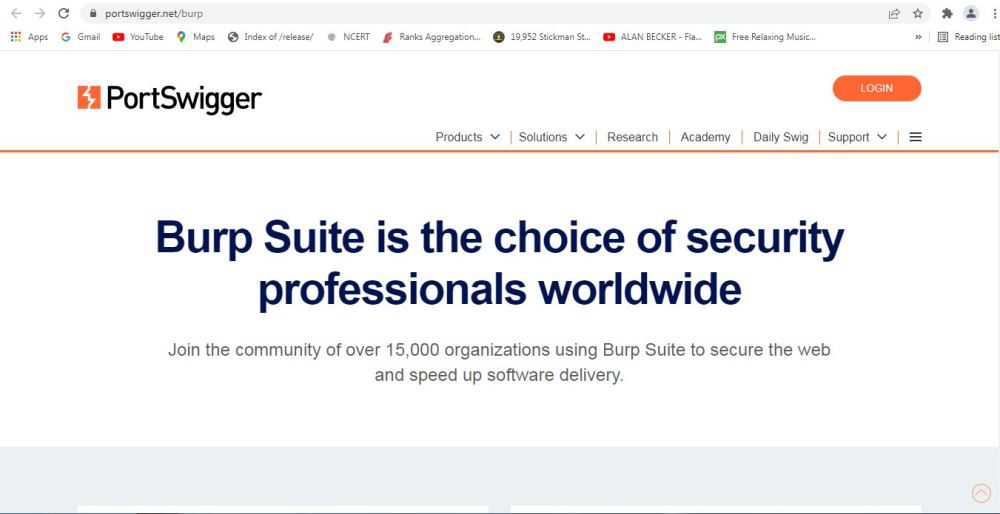
1: Download Burpsuite Pro

Burp Suite is a very powerful cyber security tool that is used for scanning vulnerabilities from different applications like web applications. It is available for different operating systems like macOS, Windows, Linux, etc. It is available in two versions one is Community edition and the other is Professional. It is available for free only for a time period of 30 days after which the trial period ends and subscription is compulsory.

Installing Burp Suite Professional on Windows:

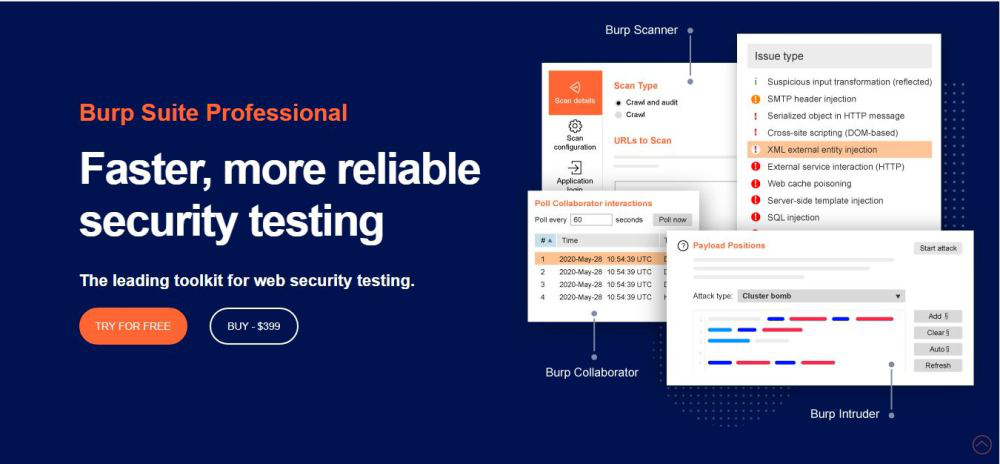
Follow the below steps to install Burp Suite Professional on Windows:

Step 1: Visit the official Burp Suite website using any web browser.



Step 2: Click on Products list choose Burp suite Professional and click on it.

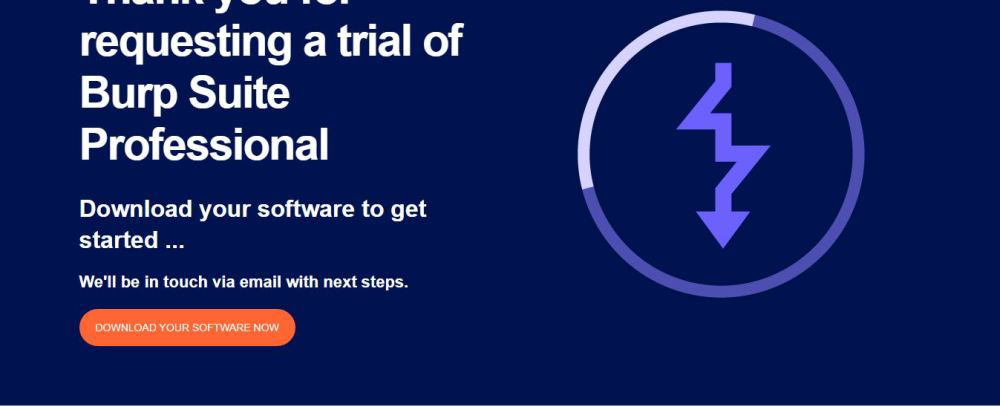
Step 3: On the next web page click on the TRY FOR FREE button.



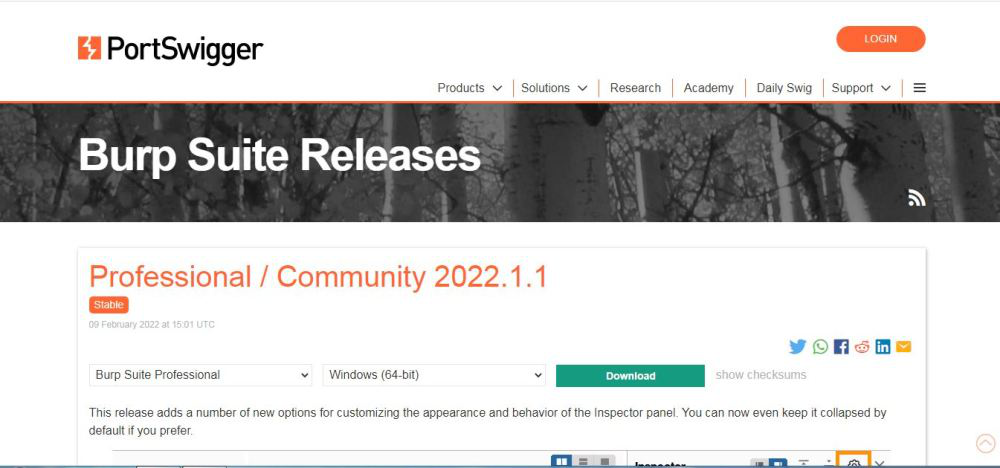
Step 4: New web page will open, which will ask for an email id, fill in the email id, and then click on the TRY FOR FREE button.



Step 5: On the next page click on Download your software now button.



Step 6: Now choose Burp suite Professional along with Windows (64-bit)and then click on the download button, to start downloading the executable file.



Step 7: Now check for the executable file in downloads in your system and open it.

Step 8: Loading of Installation Wizard will appear which will take a few seconds.

Step 9: After this Setup screen will appear, now click on the Next button.

Step 10: The next screen will be of installing location so choose the drive which will have sufficient memory space for installation process.

Step 11: Next screen is for selecting associate files so check the box and then press the Next button.

Step 12: Next screen will be of choosing the Start menu folder so don’t do anything just click on the Next Button.

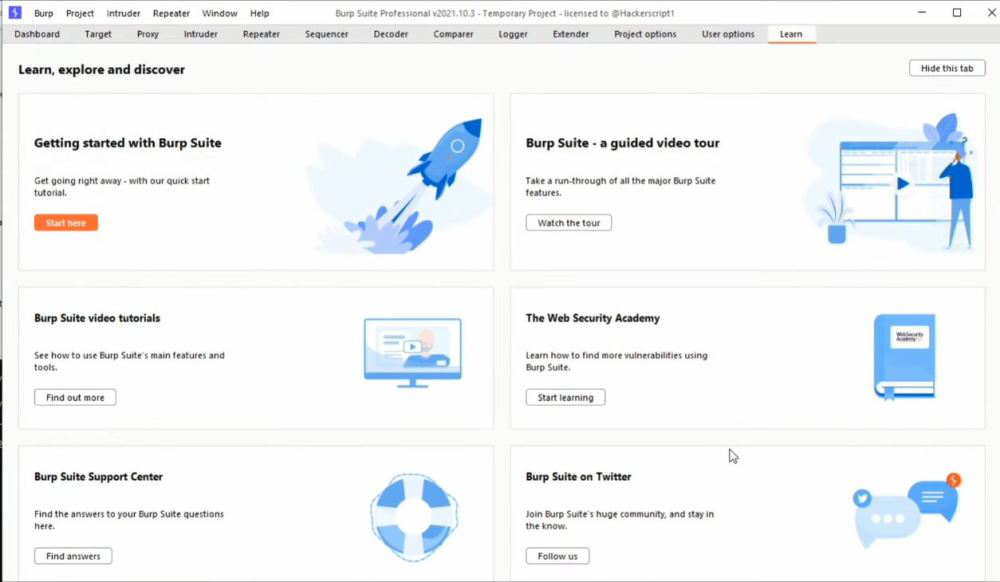
Step 13: After this installation process will start and will hardly take a minute to complete the installation.

Step 14: Click on the Finish button after the installation process is complete.

Step 15: Burp suite is successfully installed on the system and an icon is created on the desktop.

Step 16: Run the software, a screen containing terms and conditions will appear Click on I Accept.

Step 17: Finally the interface will initialize.



Congratulations!! At this point, you have successfully installed Burp Suite Professional on your windows system.

http://172.30.2.67:8000

http://172.30.2.67:8001

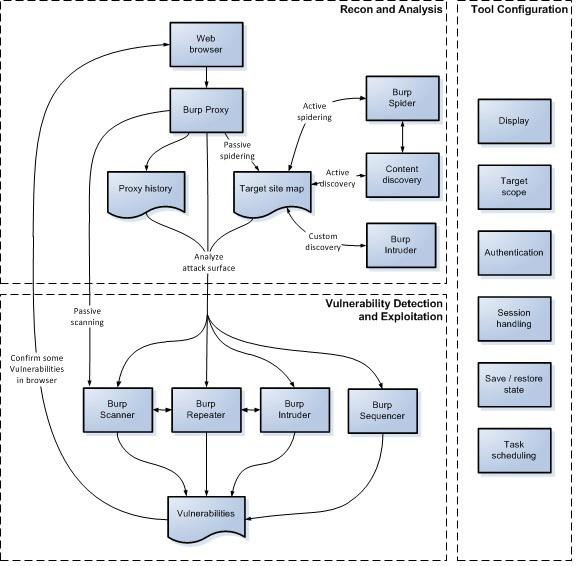
http://172.30.2.67:8003

2: Burpsuite Installation & Walkthrough

Burp lets you combine manual and automated techniques effectively, gives you complete control over all of the actions that Burp performs, and provides detailed information and analysis about the applications you are testing.

Some users may not wish to use Burp in this way, and only want to perform a quick and easy vulnerability scan of their application. If this is what you need, please refer to Scanning web sites.

The diagram below is a high-level overview of the key parts of Burp's penetration testing workflow:



**Recon and analysis**

The Proxy tool lies at the heart of Burp's workflow. It lets you use Burp's browser to navigate the application, while Burp captures all relevant information and lets you easily initiate further actions. In a typical test, the recon and analysis phase involves the tasks described below.

**Manually map the application**

Using Burp's browser while proxying traffic through Burp, manually map the application by following links, submitting forms, and stepping through multi-step processes. This process will populate the Proxy history and Target site map with all of the content requested, and (via a live task) will add to the site map any further content that can be inferred from application responses (via links, forms, etc.). You should then review any unrequested items (shown in gray in the site map), and request these using the browser.

http://172.30.2.67:8000

http://172.30.2.67:8001

http://172.30.2.67:8003

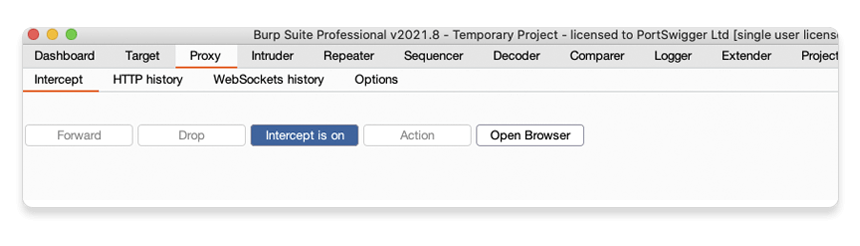
3: Intercept Request Using Burp Proxy

Burp Proxy lets you intercept HTTP requests and responses sent between Burp's browser and the target server. This enables you to study how the website behaves when you perform different actions.

Step 1: Launch Burp's browser

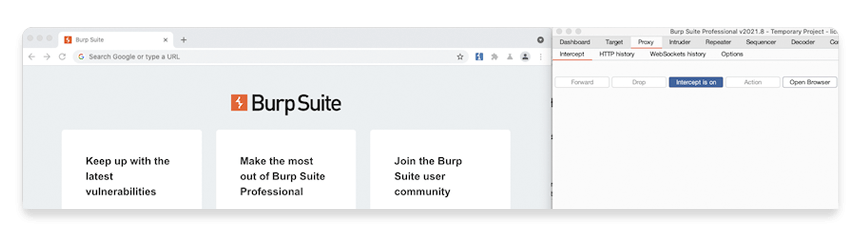
Go to the Proxy > Intercept tab.

Click the Intercept is off button, so it toggles to Intercept is on.



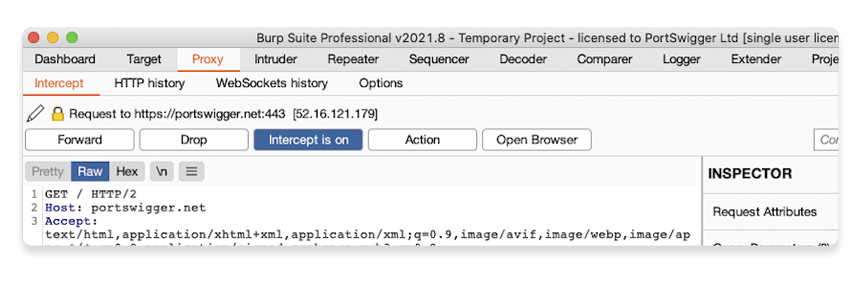
Click Open Browser. This launches Burp's browser, which is preconfigured to work with Burp right out of the box.

Position the windows so that you can see both Burp and Burp's browser.



Step 2: Intercept a request

Using Burp's browser, try to visit https://portswigger.net and observe that the site doesn't load. Burp Proxy has intercepted the HTTP request that was issued by the browser before it could reach the server. You can see this intercepted request on the Proxy > Intercept tab.



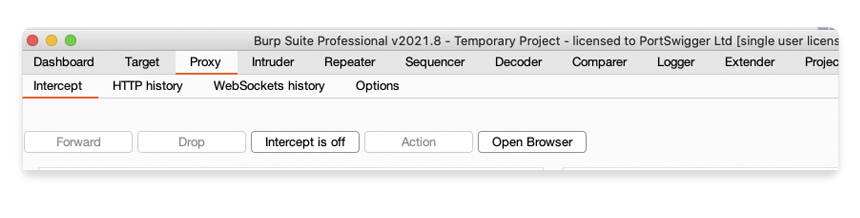
The request is held here so that you can study it, and even modify it, before forwarding it to the target server.

Step 3: Forward the request

Click the Forward button several times to send the intercepted request, and any subsequent ones, until the page loads in Burp's browser.

Step 4: Switch off interception

Due to the number of requests browsers typically send, you often won't want to intercept every single one of them. Click the Intercept is on button so that it now says Intercept is off.

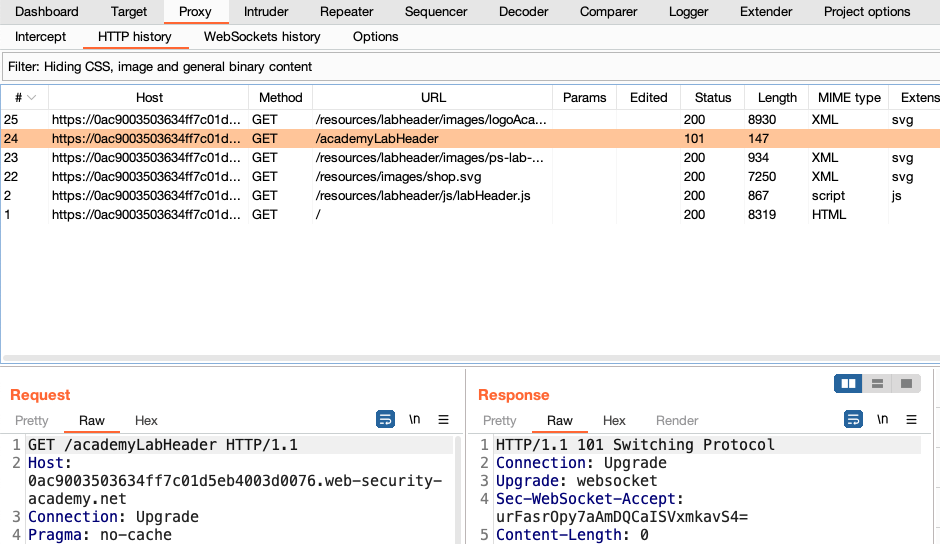


Go back to the browser and confirm that you can now interact with the site as normal.

Step 5: View the HTTP history

In Burp, go to the Proxy > HTTP history tab. Here, you can see the history of all HTTP traffic that has passed through Burp Proxy, even while interception was switched off.

Click on any entry in the history to view the raw HTTP request, along with the corresponding response from the server.



This lets you explore the website as normal and study the interactions between Burp's browser and the server afterward, which is more convenient in many cases.

http://172.30.2.67:8000

http://172.30.2.67:8001

http://172.30.2.67:8003

4: Burpsuite Intruder DeepDive

Burp Intruder is a tool for automating customized attacks against web applications. It is extremely powerful and configurable, and can be used to perform a huge range of tasks, from simple brute-force guessing of web directories through to active exploitation of complex blind SQL injection vulnerabilities.

**How Intruder works**

Burp Intruder works by taking an HTTP request (called the "base request"), modifying the request in various systematic ways, issuing each modified version of the request, and analyzing the application's responses to identify interesting features.

For each attack, you must specify one or more sets of payloads, and the positions in the base request where the payloads are to be placed. Numerous methods of generating payloads are available (including simple lists of strings, numbers, dates, brute force, bit flipping, and many others). Payloads can be placed into payload positions using different algorithms. Various tools are available to help analyze the results and identify interesting items for further investigation.

**Saving an attack**

Intruder attacks are not saved to a project file by default. If you are using a project file, you can save attacks by doing any of the following:

* From the Intruder attack window, click on the Save tab. Here, you can save to a project file, or save the results table, server responses or configuration of the attack.
* From the Intruder attack window, click on the Options tab. Under Save Options, select the check box to save the attack to a project file. This can be done before or during an attack.
* From the Dashboard, scroll to the attack in the task list and click on the save icon. This can be done before or during an attack.
* In Intruder, select the Options tab. Under Save Options, select the check box to save the attack to a project file.
* When an attack has finished, Burp will give you the option of saving the attack to a project file if you close the attack window.

Once an attack is saved to a project file, the state of the attack is constantly saved from that point on. Saved attacks can be closed, and re-opened later from the task list of the Dashboard.

Intruder does not save attacks to project files by default, as saving many attacks can result in large project files. We recommend that you only save attacks to project files once you have found something interesting. Note that this opt-in saving is unique to Intruder: other tasks (such as scans) have a smaller effect on project file size and are saved to project files by default.

Note

Intruder attacks can no longer be saved to state files. Legacy state files can still be loaded, however. To load a legacy state file, Select the top level Intruder tab and click on Open saved attack.

**Closing an attack window**

If you close an attack window while an attack is in progress, you will be prompted as to whether you wish to let the attack carry on in the background or discard the attack. If you close an attack window once the attack is finished, you will be prompted as to whether you wish to discard the attack, keep it in memory, or save it to a project file.

If you don't want to be asked each time, you can set a default answer to these prompts. Go to the top-level Intruder menu and select Close attack results preferences to set the default results for closing attack windows.

Note

Missing information in a row on the attack results page may mean that you shut down Burp Suite while an attack was in progress, and one of the requests was not sent.

**Typical uses**

Burp Intruder is a very flexible tool and can help automate all kinds of tasks when testing web applications. The most common use cases for Intruder fall into the following categories:

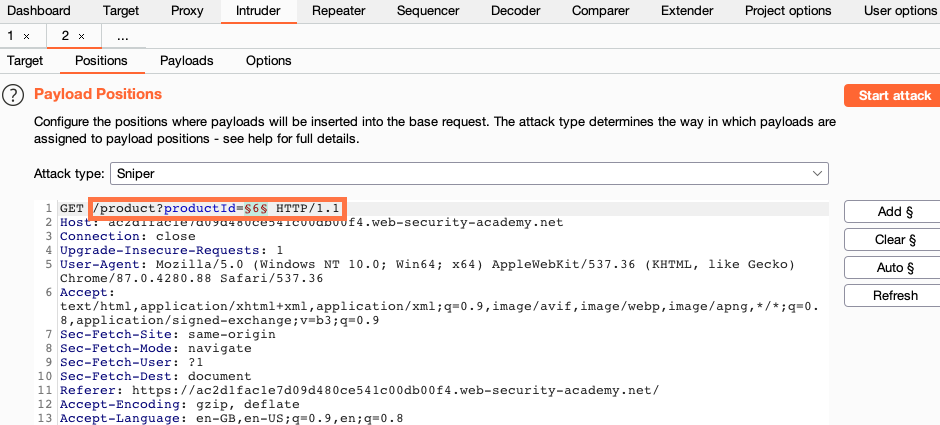
* Enumerating identifiers
* Harvesting useful data
* Fuzzing for vulnerabilities

**Enumerating identifiers**

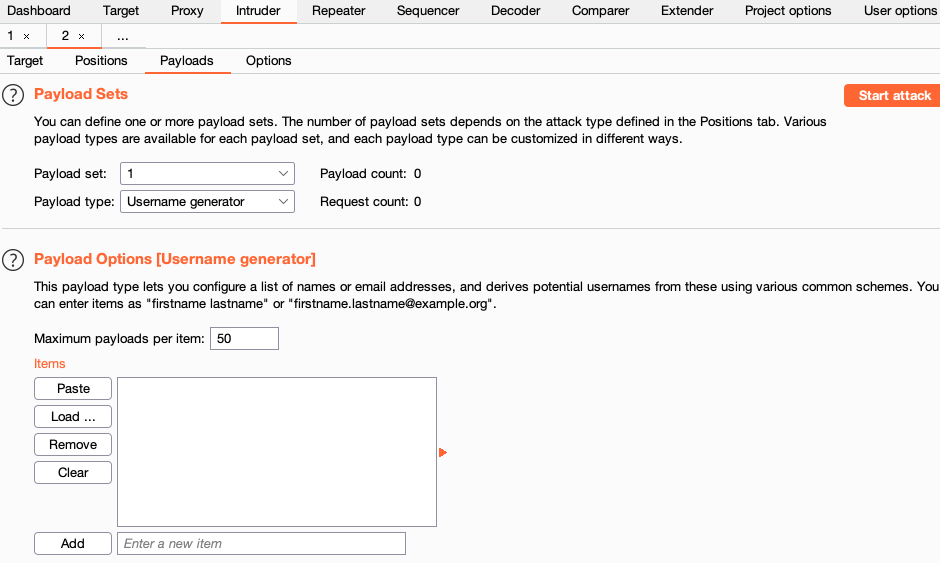
Web applications frequently use identifiers to refer to items of data and resources; for example, usernames, document IDs, and account numbers. Often, you will need to cycle through a large number of potential identifiers to enumerate which ones are valid or worthy of further investigation. To do this in Burp Intruder, you can perform the following steps:

Find an application request that contains the identifier in a parameter, and where the response indicates whether the identifier is valid.

Configure a single payload position at the parameter's value.

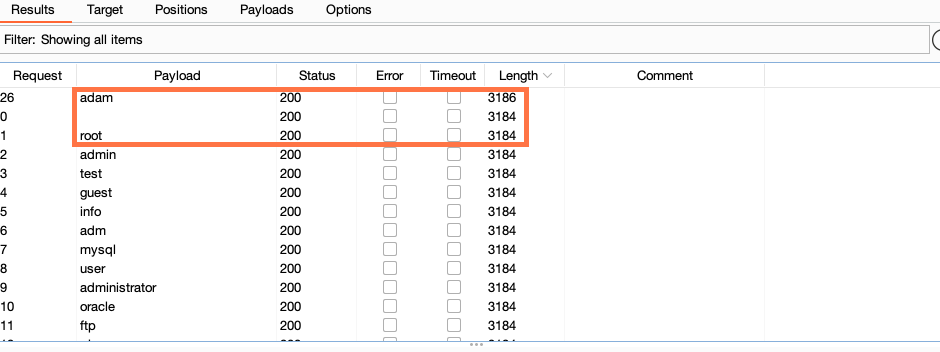


Use a suitable payload type to generate potential identifiers to test, using the correct format or scheme.



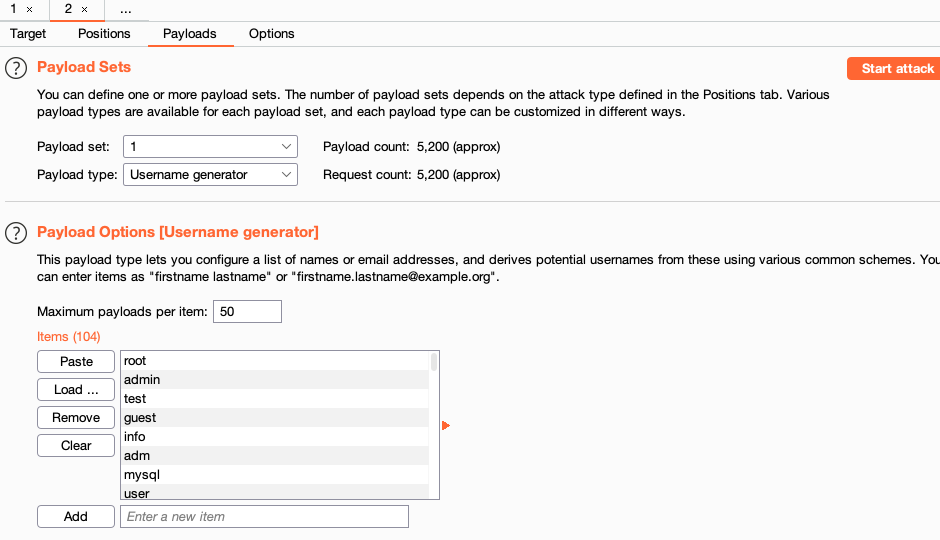
Identify a feature of the response from which valid identifiers can be reliably inferred, and configure Burp accordingly.

For example, if a valid identifier returns a different HTTP status code or response length, you can sort the attack results on this attribute. Or if a valid identifier returns a response containing a specific expression, you can define a match grep item to pick out responses that match this expression.



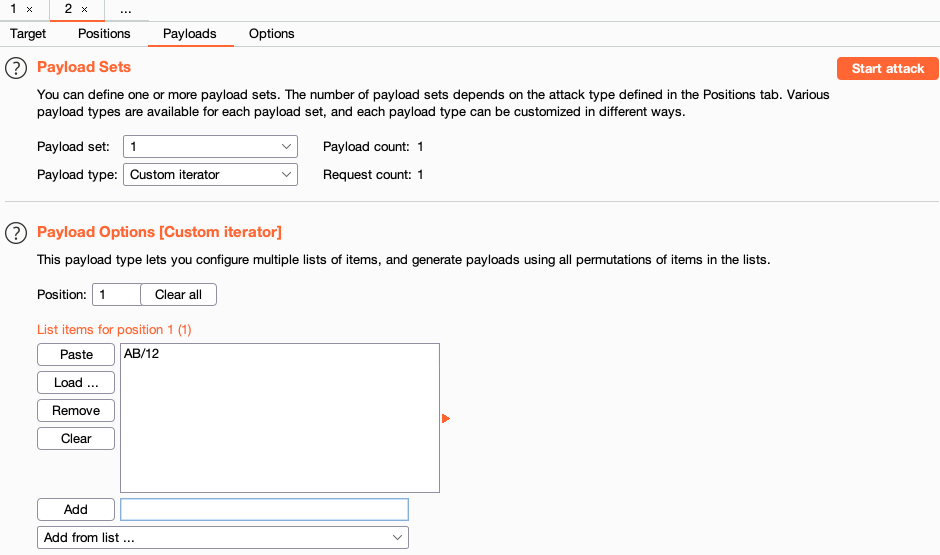
If the application's login failure messages let you enumerate valid usernames, use the username generator payload type to cycle through a long list of possible usernames and identify valid ones.

Having identified a list of valid usernames, you can use the simple list payload type with a set of common passwords to attempt to guess user's passwords.



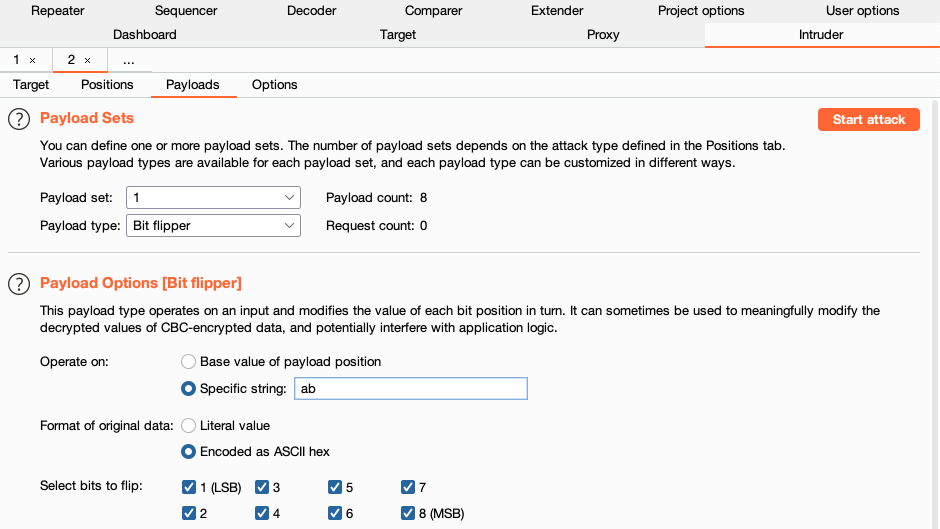
If an order processing application function lets you view details of any order by submitting a valid order ID, you can use the custom iterator payload type to generate potential order IDs in the correct format, and trawl for other users' orders.

This payload type lets you configure multiple lists of items, and generate payloads using all permutations of items in the lists. It provides a powerful way to generate custom permutations of characters or other items according to a given template. For example, a payroll application may identify individuals using a personnel number of the form AB/12; you may need to iterate through all possible personnel numbers to obtain the details of all individuals.



If an application uses meaningful structured session tokens that are encrypted using a CBC cipher, you can use the bit flipper payload type to systematically modify a valid token to try to meaningfully tamper with its decrypted value.

This payload type operates on an input and modifies the value of each bit position in turn. It can operate on the existing base value of each payload position, or on a specified string. It cycles through the base string one character at a time, flipping each (specified) bit in turn.

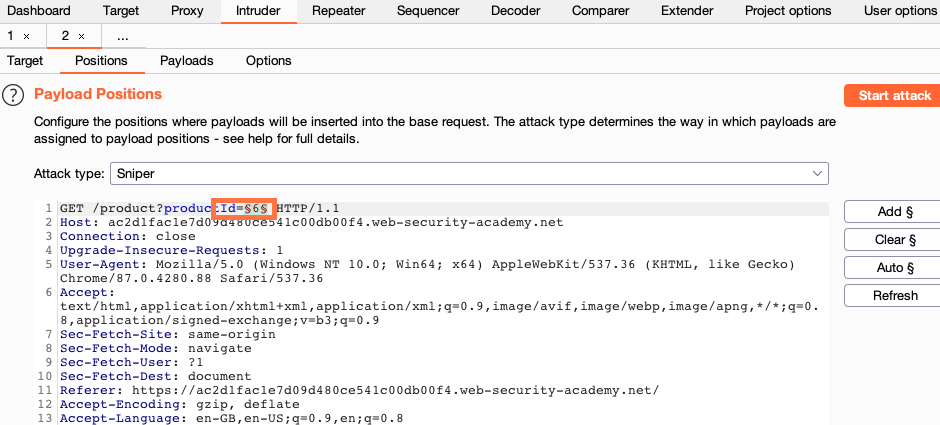


**Harvesting useful data**

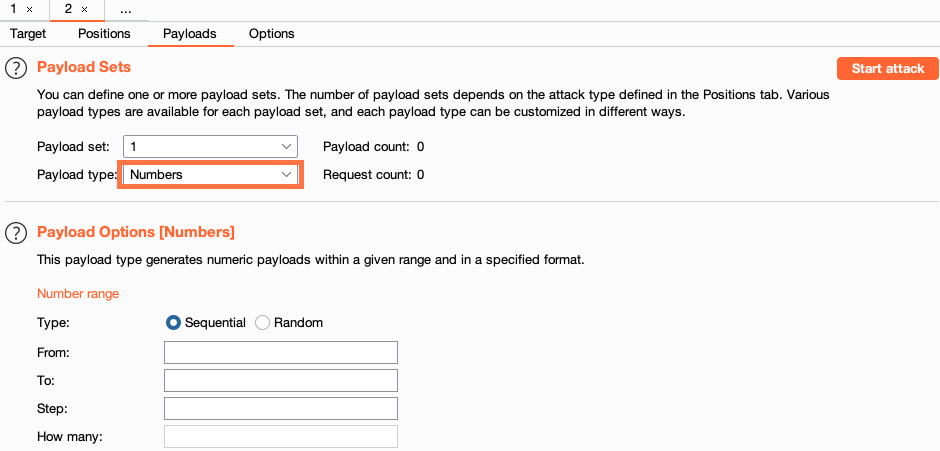
In many situations, rather than simply identifying valid identifiers, you need to extract some interesting data about each item, to help you focus your efforts on the most critical items, or to feed in to other attacks. To do this in Burp Intruder, you can perform the following steps:

Find an request that contains an identifier in a parameter, and where the response contains the interesting data about the requested item.

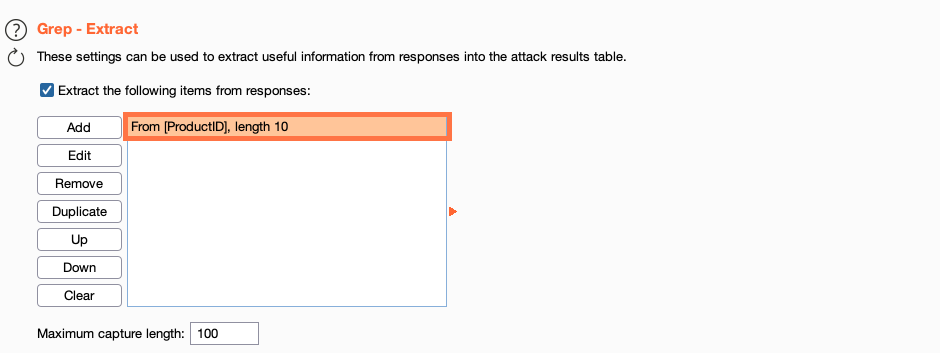
Configure a single payload position at the parameter's value.



Use a suitable payload type to generate potential identifiers to test, using the correct format or scheme.

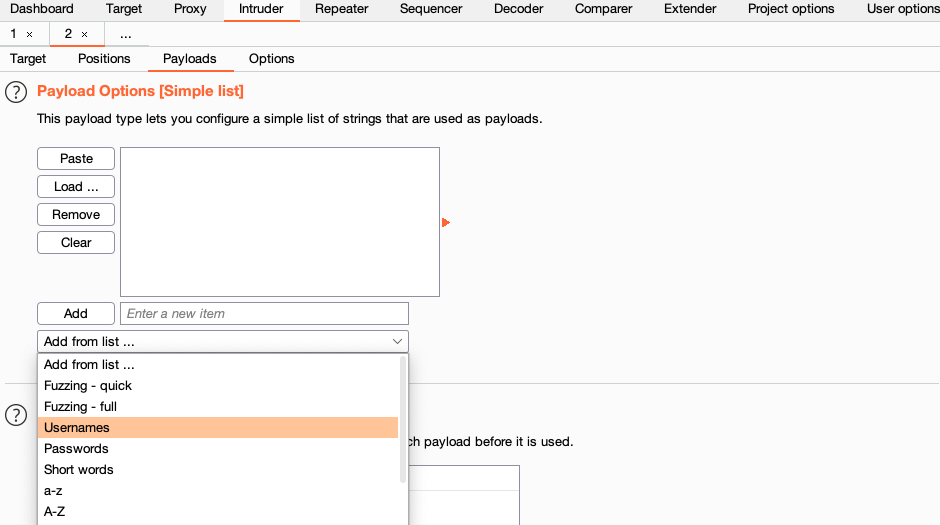


Configure an extract grep item to retrieve the relevant data from each response, and list this in the attack results.

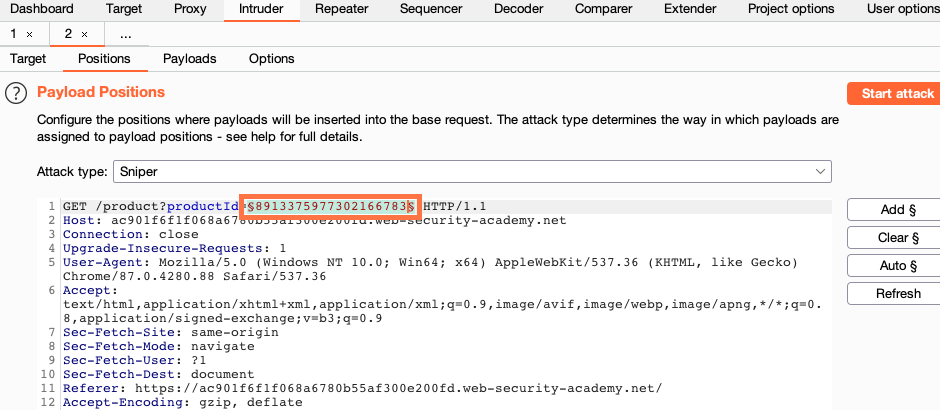


If the application has a "Forgotten password" feature that takes a username as a parameter and displays a password hint that was set by that user, you can cycle through a simple list of common usernames, and extract the password hint for each valid user.

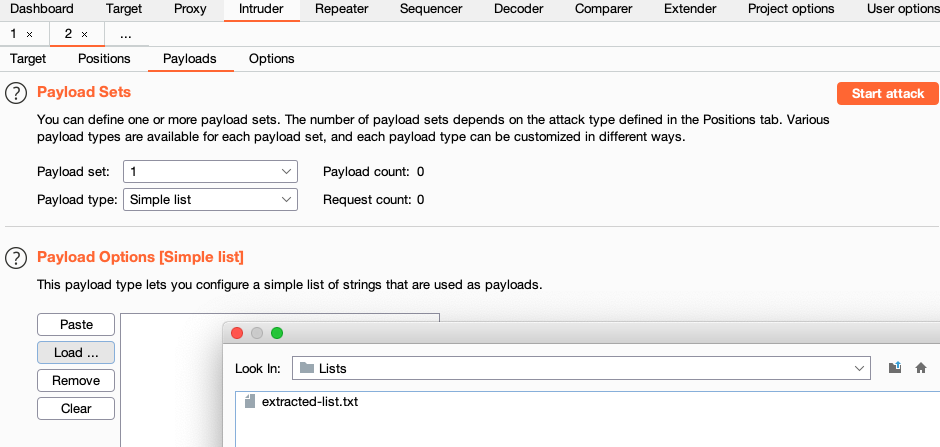
You can then quickly scan the listing of retrieved hints to locate ones that are easily guessed.



If the application returns some content dynamically, via a single URL that contains a numeric page ID parameter, you can use the numbers payload type to cycle through all possible identifiers and retrieve the HTML title tag for each page. You can then quickly review the list of available pages to identify any that are particularly interesting or which you should not be allowed to access.



If application has a "User profile" page containing information about each user, including their role in the application, you can cycle through an already extracted list of usernames, and retrieve the role for each user, allowing you to quickly identify administrative accounts for further targeted attacks.



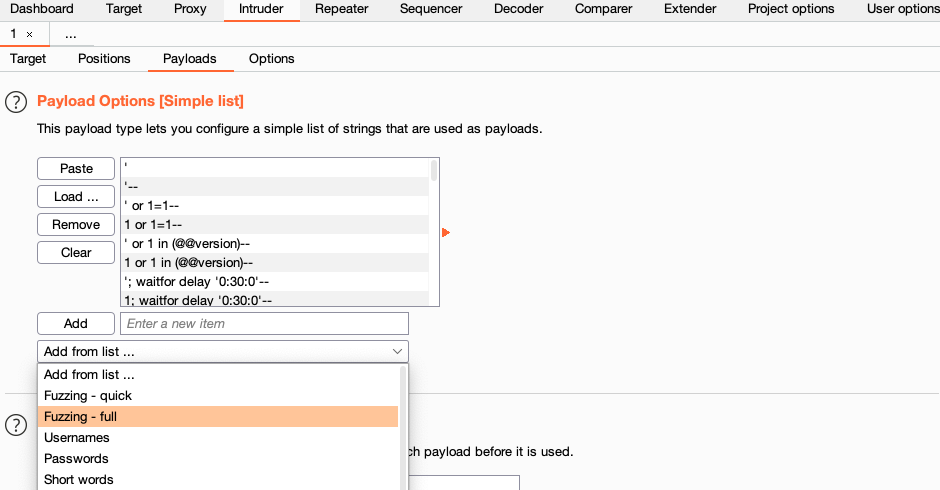
**Fuzzing for vulnerabilities**

Many input-based vulnerabilities, such SQL injection, cross-site scripting, and file path traversal can be detected by submitting various test strings in request parameters, and analyzing the application's responses for error messages and other anomalies. Given the size and complexity of today's applications, performing this testing manually is a time consuming and tedious process.

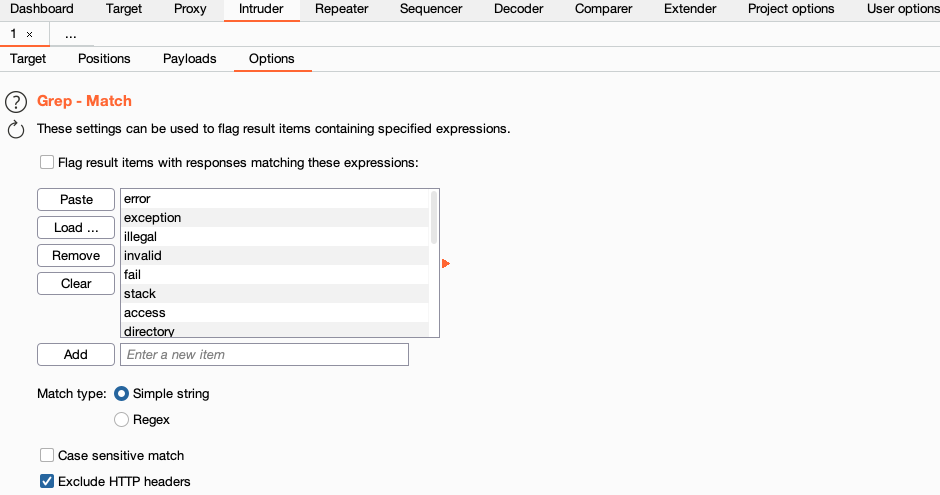
You can automate web application fuzzing with Burp Intruder by performing the following steps:

First, configure payload positions at the values of all request parameters. Then use the simple list payload type.

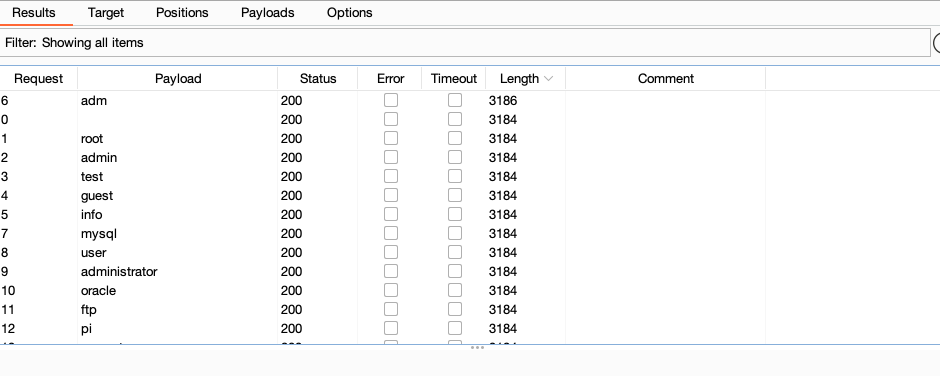
Configure the payload list using one of Burp's predefined payload lists containing common fuzz strings, or your own list of attack strings.



Configure match grep items with various common error message strings. The default options in the match grep UI include a list of useful strings for this purpose.



After launching the attack review the attack results to identify interesting errors and other anomalies. You should sort the results table on each of the match grep columns, and also on other relevant columns such as response length, HTTP status code, response timers, and so on.



When fuzzing, you will typically want to test a large number of requests using the same Intruder payloads and match grep configuration. To facilitate this, you can use the Intruder menu to configure the New tab behavior option to Copy configuration from last tab. Then, when you have configured your payloads and grep strings for one request, subsequent requests that you send to Intruder will pick up the same configuration options within their tab. To fuzz multiple requests, you then simply need to send each one to Intruder, and choose Start attack from the Intruder menu.

http://172.30.2.67:8000

http://172.30.2.67:8001

http://172.30.2.67:8003

5 ; VA Using Burpsuite

The audit phase of a scan involves analyzing the application's traffic and behavior to identify security vulnerabilities and other issues. Burp Scanner employs a wide range of techniques to deliver a high-coverage, dead-accurate audit of the application being scanned.

**Audit phases**

Burp Scanner carries out several distinct audit phases. These are divided into three areas:

* Passive phases
* Active phases
* JavaScript analysis phases

Performing multiple phases within each area allows Burp to:

* Effectively find and exploit functions that store and return user input.
* Avoid duplication by handling frequently occurring issues and insertion points in an optimal manner.
* Execute applicable work in parallel to make most efficient use of system resources.

**Issue types**

Burp is able to detect a huge variety of issues, including security vulnerabilities and other items of informational interest. For a full list, see Vulnerabilities detected by Burp Scanner.

Issues can be divided into different types according to the nature of the audit activity that is involved in detecting them:

* Passive - These are issues that can be detected purely by inspecting the application's normal requests and responses. For example, serialized objects in HTTP messages.
* Light active - These are issues that can be detected by making a small number of benign additional requests. For example, cross-origin resource sharing (CORS) that trusts arbitrary origins.
* Medium active - These are issues that can be detected by making requests that the application might reasonably view as malicious. For example, OS command injection.
* Intrusive active - These are issues that can be detected by making requests that carry a higher risk of damaging the application or its data. For example, SQL injection.
* JavaScript analysis - These are issues that can be detected by analyzing the JavaScript that the application executes on the client side. For example, DOM-based cross-site scripting. Detecting these issues is often resource-intensive on the machine that Burp is running on. These issues can also be categorized as "passive" (for self-contained DOM-based issues) or "medium active" (for reflected and stored variants).

Issues can also be divided into different types according to the level at which they are found:

* Host level - These are issues that arise at the level of the host HTTP service that the application is running on. For example, a permissive flash cross-domain policy.
* Request level - These are issues that arise at the level of an individual request. For example, cross-site request forgery.
* Insertion point level - These are issues that arise at the level of an insertion point within a request. For example, file path traversal.

**Insertion points**

Burp Scanner uses the concept of insertion points to place payloads into different locations within requests. In general, an insertion point represents a piece of data within a request that might be specifically processed by the server-side application. The following example shows a request with some common types of insertion points highlighted:

Burp Scanner audits insertion points individually, sending payloads to each insertion point in turn to test the application's handling of that input.

**Encoding data within insertion points**

Different locations within an HTTP request typically need to have their data encoded in different ways. Burp Scanner automatically applies suitable encoding to payloads based on the type of the insertion point, to ensure that the raw payloads reach the relevant application functions. For example, different encoding is applied to standard body parameters:

to parameters within JSON data:

and to parameters within XML data:

Burp Scanner also detects when an application is using other types of encoding that are not tied to the insertion point type, such as Base64:

**Nested insertion points**

Some applications apply multiple layers of encoding to the same data, effectively nesting one format within another. Burp Scanner is able to detect this behavior, and automatically applies the same layers of encoding to payloads:

**Modifying parameter locations**

Some applications place a certain piece of input into one type of parameter, but will in fact process the input if it is submitted in a different type of parameter. This happens because some platform APIs that applications use to retrieve input from requests are agnostic as to the type of parameter that is holding the input. However, some protections in place around an application, such as web application firewalls, might only apply to the original parameter type.

Burp can optionally exploit this behavior by changing the parameter types of insertion points, to create requests that might bypass protections and reach vulnerable application functionality. For example, if a payload is submitted within a URL query string parameter, Burp might also submit corresponding payloads within a body parameter and a cookie:

**Automatic session handling**

When Burp Scanner's audit follows on from an automated crawl, it is able to make use of the crawl results to automatically maintain session during the audit, with zero configuration by the user.

When Burp performs an audit of an individual request, it begins by identifying the shortest path to reach that request from the starting location of the crawl:

Burp then determines the most efficient way to deliver that same request repeatedly within a valid session. It does this by first re-walking the path to obtain a fresh sample of any session tokens, and then testing various simplifications of the path to see if the session is correctly maintained.

In many cases, it is possible to simply reissue the final request over and over. This can happen because the request doesn't actually contain any session tokens at all:

Or because the only session tokens are cookies, which can typically be used multiple times:

Or because, although the request contains both cookies and CSRF tokens, the CSRF tokens can be used repeatedly:

In some cases, it is necessary to issue the preceding request on each occasion prior to issuing the request that is being audited. This normally happens because the application uses single-use CSRF tokens. Because the tokens are single-use, it is necessary to reissue the preceding request on each occasion, to obtain a fresh token.

In extreme cases, every transition between requests is protected by a single-use token. This happens occasionally in high-security applications where navigation is tightly controlled. In this situation, the most reliable way to repeatedly issue the request to be audited is to always return to the starting location and walk the full path to that request:

Once Burp has determined the most efficient way to repeatedly issue the request that is to be audited, it carries out the audit. While carrying out its various audit checks, Burp periodically monitors the application's responses to ensure that a valid session is maintained. If Burp positively confirms the validity of the session, then it sets a checkpoint on the audit checks that have been fully completed. If Burp identifies that the session is no longer valid, it rolls back to the latest checkpoint and resumes from there. This logic is carried out in a way that minimizes the overhead of session management and avoids indefinite loops if sessions are frequently lost. For example:

**Avoiding duplication**

Burp Scanner uses various techniques to minimize duplicated effort and reporting of duplicate issues.

**Consolidation of frequently occurring passive issues**

Some passively-detected issues are liable to exist at many different locations within an application, due to a chosen approach to development or a reused page template (for example, cross-site request forgery or cross-domain script include). Some issues will even exist across the entire application due to a platform-level configuration (for example, strict transport security not enforced). In this situation, Burp Scanner will by default avoid generating duplicated issues by aggregating these and reporting a single issue at the applicable level, which might be the web root of the host or a particular folder beneath which all of the issues are found.

**Handling of frequently occurring insertion points**

Some insertion points are liable to exist within many or all requests used by the application, but might not represent interesting attack surface. Examples of this are cookies which, once set, are submitted within every subsequent request, and cachebuster parameters which are placed into URL query strings to prevent caching but are not processed by the server-side application. Performing a full audit of these insertion points in every request could represent a considerable overhead of redundant work. In this situation, Burp Scanner will by default identify insertion points that have proven to be uninteresting (occurring frequently without any issues generated) and will drop to performing a more lightweight audit of those insertion points. If the lightweight audit identifies any interesting behavior that indicates server-side processing, then Burp will continue and perform a full audit of the insertion point as normal.

**JavaScript analysis**

Burp Scanner analyzes JavaScript contained within application responses, to identify a wide range of DOM-based vulnerabilities. To do this, it uses a combination of static and dynamic analysis:

* Static analysis - This parses the JavaScript code to construct an abstract syntax tree (AST). It identifies the tainted sources that are potentially controllable by an attacker, and the dangerous sinks that could be used to perform an attack. It analyzes the possible data flows through the code to identify potential paths via which malicious data could be propagated from a tainted source to a dangerous sink.
* Dynamic analysis - This loads the response into an embedded headless browser. It injects payloads into the DOM at locations that are potentially controllable by an attacker, and executes the JavaScript within the response. It also interacts with the page by creating mouse events to achieve as much code coverage as possible (in onclick event handlers etc.). It monitors the dangerous sinks that could be used to perform an attack, to identify any injected payloads that reach those sinks.

The static and dynamic approaches have different inherent strengths and weaknesses:

* Static analysis is able to find some vulnerabilities that dynamic analysis misses, because static analysis can identify code paths that could possibly be executed in the right circumstances, but which are not in fact executed during the dynamic analysis. For example, a branch in execution might be controlled by a number of parameters that the attacker controls. The static analysis is able to identify and analyze this branch and find taint paths within it, while the dynamic analysis might not trigger the relevant execution due to the actual combination of parameters that it uses. However, static analysis is inherently prone to false positive results, because it might see some combinations of code branches as executable when they are in fact not, and because it fails to understand custom data validation logic that means taint paths from sources to sinks are not in fact exploitable.
* Dynamic analysis has the opposite characteristics. It is much less prone to false positives because if it actually observes suitable tainted data being propagated from source to sink during execution, then this behavior is concrete evidence for a vulnerability. However, it can suffer from false negatives in situations where the tainted data that it injects doesn't reach a sink due to the current state of the application or the values of other data, both of which an attacker might in fact be able to control.

Burp Scanner harnesses the joint benefits of the static and dynamic approaches. Where possible, it correlates the results of the two techniques, and reports issues with evidence obtained using both. These issues may be regarded as rock-solid findings, and are reported as certain. In cases where only static analysis can detect a potential issue, Burp downgrades the confidence with which the issue is reported. This integrated approach to JavaScript analysis greatly assists a tester who is reviewing the results to find the most important issues.

**Handling application errors**

Performing a full audit of a web application can be an invasive process, and it is common to encounter problems like connection failures, transmission timeouts, or outages of back-end components while a scan is in progress. Additionally, protections like web application firewalls might selectively drop connections based on specific payloads, or even based on any unexpected values in certain parameter values.

During the audit process, Burp tracks the occurrence of error conditions in as granular a way as possible. If an individual action causes an error, Burp marks that action as failed and moves on to the next action. Optionally, if repeated actions fail at the same level of activity, then that whole level is marked as failed. So Burp will progressively mark as failed: individual audit checks, then individual insertion points, then the entire request being audited, and ultimately the entire scan.

Since it is common to encounter isolated errors with specific audit requests, Burp will initially capture details of the error and continue scanning. When the whole audit is complete, Burp can optionally perform a number of follow-up passes to retry failed operations. This can be useful in cases where a particular application component (such as a back-end database) experienced a problem for part of the scan. Burp can also optionally pause or abort the scan if too many errors are observed, so that the user can investigate the problem and resume or repeat the scan when the application is stabilized.

http://172.30.2.67:8000

http://172.30.2.67:8001

http://172.30.2.67:8003

6 : Authenticated Scan With Burpsuite

When testing for web security vulnerabilities, it's vital to cover as much of an application's attack surface as possible. But what if that attack surface is partially hidden within a privileged area (e.g. a user dashboard) that requires a login? Authenticated scanning allows a web vulnerability scanner to log in to search for vulnerabilities inside such areas.

**How you can enable Burp Scanner to authenticate itself**

With simple login functions, authenticating Burp Scanner is as easy as supplying it with a valid set of credentials (e.g. username and password). Burp Scanner will then identify HTML login forms, and use your data to authenticate itself when crawling and scanning.

But with more complex login sequences like single-sign on (SSO), automation isn't so straightforward. Such systems often make heavy use of JavaScript, meaning that they must be rendered in a browser before being interacted with. Fortunately, Burp Scanner can execute JavaScript through its embedded Chromium browser - making it possible to automate many of these complex login processes.

Using the Burp Suite Navigation Recorder Chromium extension, users can record paths through complex login systems for future use. Use it in your own Chrome installation, or Burp Suite's embedded Chromium browser (Burp Suite Professional), and Burp Suite will store the path as JSON. Whether it's external SSO, a multi-step form, or another more arbitrary process, Burp Suite's browser-powered scanner can (with a few exceptions) authenticate itself

http://172.30.2.67:8000

[http://172.30.2.67:8001](http://172.30.2.67:8001/)

http://172.30.2.67:8003