Burp suite components

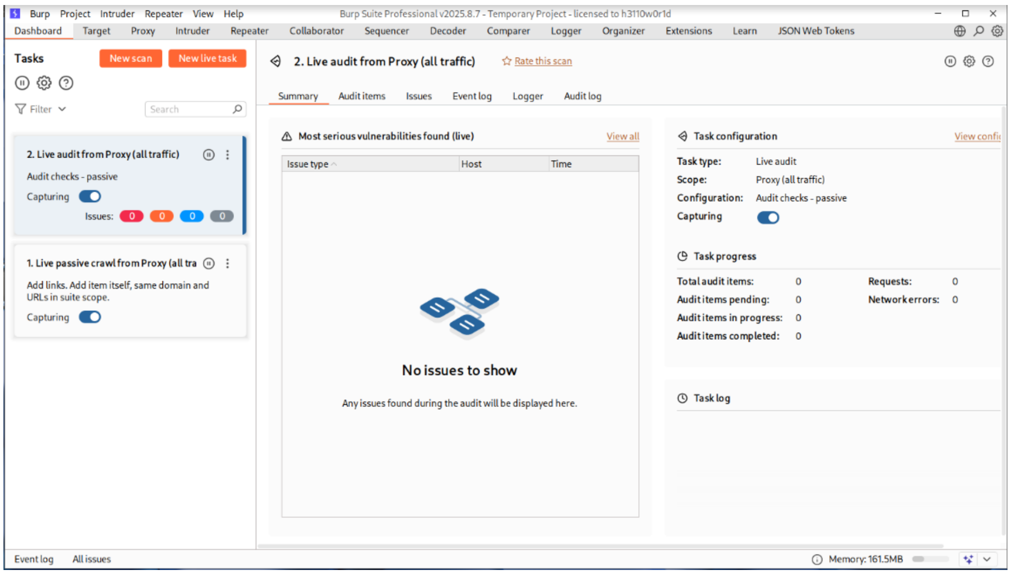
In **Burp Suite**, tools like **Proxy**, **Repeater**, **Intruder**, **Decoder**, and others are known as **components** or **modules**. Each component has a specific role in testing web applications for vulnerabilities. For example, the **Proxy** captures and inspects web traffic, while the **Repeater** is used to resend and modify requests manually. Together, these components work as an integrated toolkit that helps security testers analyze, exploit, and secure web applications effectively.

Dashboard

The Dashboard in Burp Suite is the central control panel where you can monitor all the activities, tasks, and background processes happening within the tool. It provides a quick overview of your current testing session and helps you manage scans, view issues, and keep track of what Burp is doing behind the scene.

1. Overview

The Dashboard acts as a real-time monitoring center. When you open Burp Suite, it's usually the first tab you see. It summarizes your active scans, issues found, and background tasks. This helps you understand your testing progress without jumping between tools.


Burp Project Intruder Repeater View Help
Burp Suite Professional v2025.8.7 - Temporary Project - licensed to h3110w0r1d
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn JSON Web Tokens
Tasks
New scan
New live task
Filter
Search
2. Live audit from Proxy (all traffic)
Rate this scan
Summary Audit items Issues Event log Logger Audit log
Most serious vulnerabilities found (live)
View all
Issue type
Host
Time
No issues to show
Task configuration
View confi
Task progress
Task log
Event log All issues
Memory: 161.5MB

2. Live audit from Proxy (all traffic)
Audit checks - passive
Capturing
Issues: 0 0 0 0

1. Live passive crawl from Proxy (all tra
Add links. Add item itself, same domain and URLs in suite scope.
Capturing

Any issues found during the audit will be displayed here.

Task type: Live audit
Scope: Proxy (all traffic)
Configuration: Audit checks - passive
Capturing

Total audit items: 0
Audit items pending: 0
Audit items in progress: 0
Audit items completed: 0
Requests: 0
Network errors: 0

It shows information such as:

The number of HTTP requests made.

The active scanning status.

Any errors or warnings.

Key results from passive and active scans.

2. Activity and Issue Tracking

Activity Log: Displays a continuous feed of Burp's activities — such as intercepted requests, analyzed responses, and background scans. You can see what Burp is currently doing in real time.

Issue Activity: Shows vulnerabilities and findings that Burp's scanner has detected. Each issue includes severity levels (High, Medium, Low, or Informational) and details about where the issue was found.

Event Log: Lists system events like scan starts, proxy connections, or errors, helping you troubleshoot problems.

3. Tasks and Scans

Tasks Panel: Lists all running and completed tasks, such as scans or crawls. You can pause, resume, or cancel these tasks.

New Scan Button: Lets you start a new scan directly from the Dashboard by specifying a target scope or URL.

Task Details: For each task, you can view progress, request/response counts, and identified issues.

4. Background Tasks

Burp continuously runs background processes such as:

Passive scanning of captured traffic.

Live issue analysis.

Extensions running through Burp Extender.

The Dashboard shows these in a Background Tasks section with progress indicators.

5. Customization and Usefulness

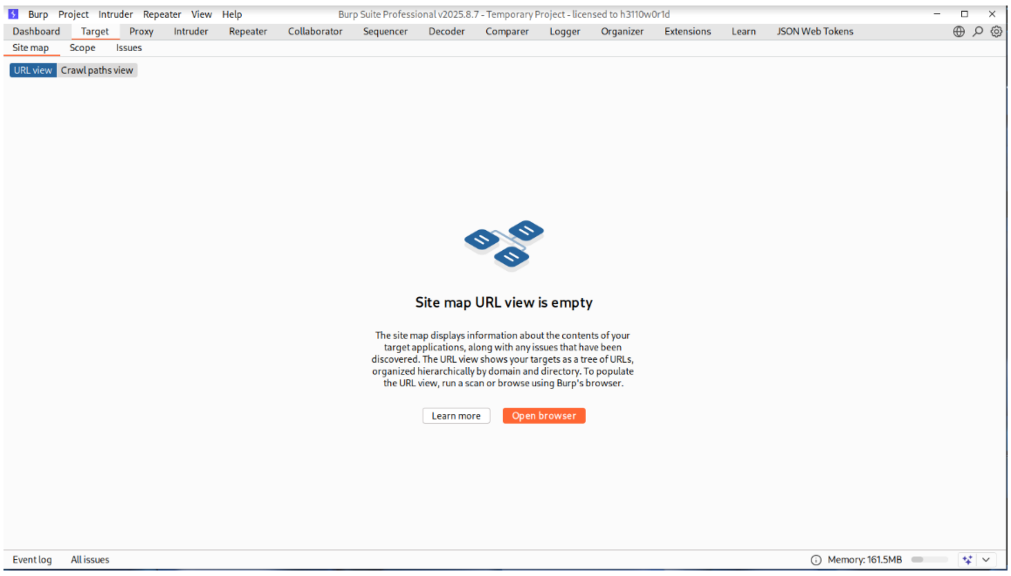
You can filter or expand panels according to your preference — for example, focusing only on issue activity or hiding less relevant logs.

The Dashboard helps you maintain situational awareness during penetration testing by showing what Burp has found so far, what it's currently analyzing, and any actions that need your attention.

Target

The **Target tab** in Burp Suite is essentially the starting point for understanding the scope and structure of the application you are testing. It provides a detailed map of all the endpoints, URLs, and content that Burp Suite has discovered, helping you focus your testing and organize your attack surface. This tab is crucial because it gives you visibility over what is in scope and what isn't, ensuring your testing remains targeted and efficient.

Here's a breakdown of the **Target tab** features and usage:


Burp Project Intruder Repeater View Help
Burp Suite Professional v2025.8.7 - Temporary Project - licensed to h3110w0r1d
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn JSON Web Tokens
Site map Scope Issues
URL view Crawl paths view
Learn more
Open browser
Event log All issues
Memory: 161.5MB

Site map URL view is empty

The site map displays information about the contents of your target applications, along with any issues that have been discovered. The URL view shows your targets as a tree of URLs, organized hierarchically by domain and directory. To populate the URL view, run a scan or browse using Burp's browser.

1. Site Map

- The **Site Map** is a hierarchical tree that displays all the domains, subdomains, and pages that Burp has discovered either through crawling or manual exploration.

- It organizes content by domain, directories, and endpoints.
- You can expand or collapse nodes to explore specific sections of the application.
- Right-clicking items allows actions like sending requests to other Burp tools (Repeater, Intruder, Scanner), or marking them as in-scope or out-of-scope.

2. Scope Management

- Burp lets you define a **scope** for your testing, which can include or exclude specific domains, directories, or file types.
- Requests and scans can be restricted to this scope to avoid testing unintended targets.
- Items outside the scope appear differently in the Site Map, helping you avoid accidental attacks on external systems.

3. Detailed Information Panel

- Clicking a node in the Site Map shows details such as HTTP methods supported, parameters, request/response snippets, and status codes.
- This helps in identifying potential areas for testing, like endpoints with GET/POST parameters or hidden forms.

4. Contextual Actions

- From the Target tab, you can:

    Send requests to other Burp tools (Repeater, Intruder, Scanner, Sequencer).

    Perform manual tests on specific endpoints.

    Filter the view by content type (HTML, JavaScript, JSON, etc.) to focus on areas of interest.

5. Usage Tips

- Regularly update your Site Map by browsing through the app while Proxy captures traffic.
- Use the **filter and search functions** to quickly find specific endpoints.
- Keep the scope clean to avoid unnecessary noise from external or irrelevant URLs.

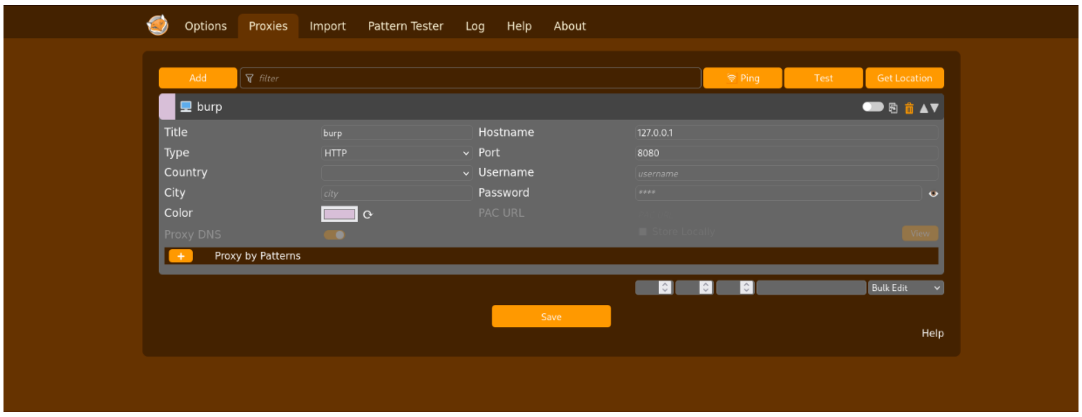
<u>Proxy</u>

Introduction: A proxy in Burp Suite functions as the central interception point that mediates all HTTP(S) and WebSocket traffic between your browser (or other HTTP client) and the target server, enabling you to observe, pause, modify, and forward messages in real time. By routing traffic through Burp you gain a live, editable record of every request and response, which is stored in the HTTP history and contributes to the Target site map used by other Burp tools; this makes the proxy both the natural starting point for manual exploration and the hub that feeds Repeater, Intruder, Scanner, Sequencer and extensions. Because the proxy performs TLS interception for encrypted traffic, it presents its own CA certificate to the client and opens a separate TLS session to the server, so you can inspect encrypted payloads — but that also means you must only use the proxy in controlled, authorized environments and install Burp's CA into the client trust store to avoid certificate warnings.

Setting up FoxyProxy

- Install FoxyProxy (Firefox/Chrome extension).
- Create a new proxy profile: host 127.0.0.1, port 8080 (or your Burp listener).

- Configure pattern-matching rules to only proxy lab/test domains (e.g., *.lab.local, localhost) so production traffic isn't proxied.
- Enable the profile when you want Burp to see traffic; switch back to "Disabled" or "Direct" when finished.
- Export Burp's CA and import it into the browser/OS trust store so HTTPS interception doesn't trigger certificate warnings.
- Test with a known lab URL and verify requests show up in Burp's HTTP history.

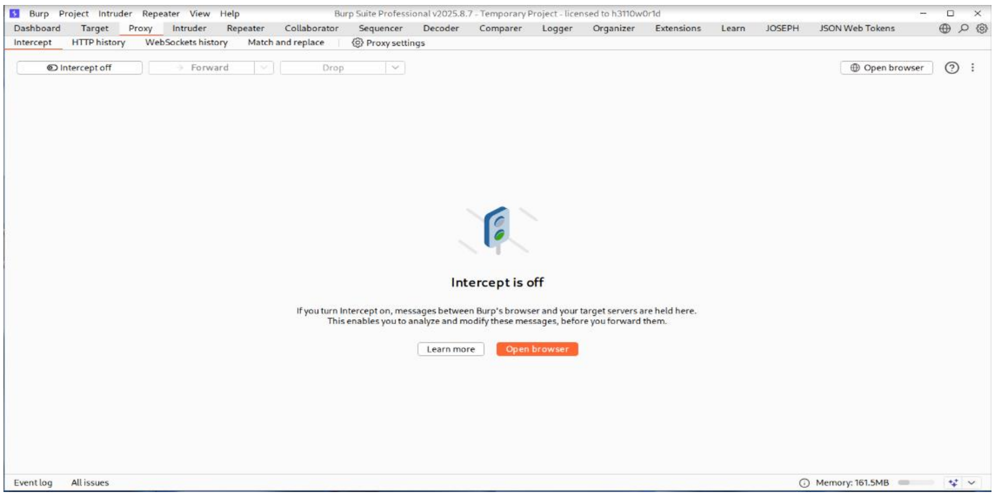


burp

Title	burp	Hostname	127.0.0.1
Type	HTTP	Port	8080
Country		Username	username
City	city	Password	****
Color		PAC URL	
Proxy DNS			

Components in the Proxy tab

- **Intercept pane:** paused request/response view for manual inspection and edits (Forward / Drop).
- **HTTP history:** chronological log of proxied requests/responses with filters and right-click actions (Send to Repeater/Intruder/etc.).
- **WebSockets pane:** live WebSocket frames and history for socket-based comms.
- **Options area:** listener settings, SSL/TLS (CA/client cert) controls, upstream proxy chaining, invisible proxy mode, and match-and-replace rules.
- **Filters & breakpoints:** rules to limit which traffic is intercepted or automatically modified.
- **Utilities:** quick send-to-tool shortcuts and context menu actions for exporting or saving requests.



Burp Suite Professional v2025.8.7 - Temporary Project - licensed to h3110w0r1d

Intercept is off

If you turn Intercept on, messages between Burp's browser and your target servers are held here.
This enables you to analyze and modify these messages, before you forward them.

How to capture packets

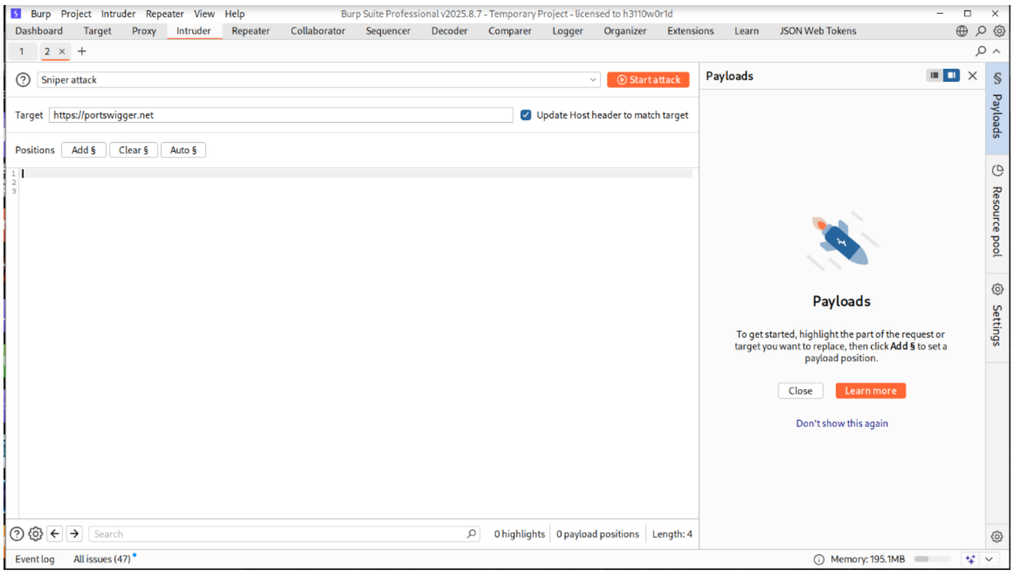
- Start Burp and ensure the proxy listener is running on the chosen port.

- Point your browser (or FoxyProxy) to the Burp proxy profile and confirm the CA is trusted.
- Decide capture mode: turn **Intercept ON** to pause messages for live edits, or **Intercept OFF** to pass traffic while logging HTTP history.
- Browse or perform the workflow you need; watch HTTP history and Target site map populate.
- Use scope settings and intercept filters to restrict capture to only relevant hosts, methods, or content types.
- For mobile: set the device Wi-Fi proxy to your Burp host/port and install the CA on the device. For non-proxy clients, use invisible proxy mode or upstream chaining.
- Verify captured items appear in HTTP history, then send selected requests to Repeater/Intruder/Decoder as needed.
- When done, disable proxying and (optionally) remove the CA from any device not used for testing.

Intruder

The **Intruder** tool in Burp Suite is a powerful, automated request-fuzzing engine. It takes one or more HTTP requests you've captured and systematically mutates parts of them (the *payload positions*) to find vulnerabilities like broken access controls, parameter manipulation flaws, SQL injection, XSS, and weak authentication. Think of it as an automated, configurable hammer for trying many inputs against a target endpoint.

Intruder lets you mark positions in a captured request where payloads should be inserted, choose one or more payload sets, and pick an attack strategy that controls how payloads are combined and sent. It reports responses (status, length, matches) and can highlight responses that look interesting, so you can triage likely vulnerabilities quickly instead of manually trying thousands of inputs.


Burp Suite Professional v2025.8.7 - Temporary Project - licensed to h3110w0r1d

Sniper attack

Target: https://portswigger.net

☑ Update Host header to match target

Payloads

To get started, highlight the part of the request or target you want to replace, then click **Add §** to set a payload position.

0 highlights | 0 payload positions | Length: 4

Memory: 195.1MB

Attack types

- **Sniper** — single position attack. Good for testing one parameter at a time with many payloads (e.g., testing a single parameter for SQLi).
- **Battering Ram** — same payload applied to all marked positions simultaneously. Useful when the same value should be submitted to multiple fields.
- **Pitchfork** — parallel lists: it takes N payload sets and inserts payload[i] into position[i] for each i. Useful when testing correlated inputs.

- **Cluster Bomb** — Cartesian product of payload sets (all combinations). Very powerful but can explode in size — use carefully.

How to configure an Intruder attack

1. **Send request** from Proxy/Repeater to Intruder.
2. **Positions tab** — Burp will auto-mark positions, but you should edit them. Use clear, minimal positions (mark only the parameter or value you want to mutate).
3. **Payloads tab** — choose payload set type (simple list, runtime file, numbers, dates, character floods, etc.), load lists or generate payloads, optionally add payload processing rules (prefix/suffix, encoders).
4. **Options tab** — configure thread count (concurrency), retry behavior, payload encoding, request throttling/delay, and response handling (grep for strings, match conditions).
5. **Start attack** — monitor responses, sort by length/status/time, and investigate anomalies.

Useful features

- **Payload processing**: add encoders (URL-encode, base64) or substitutions so payloads are formatted correctly.
- **Grep/Match rules**: highlight responses containing or lacking specific strings (e.g., “Welcome”, “SQL syntax error”). Useful to automatically flag interesting responses.
- **Response sorting & filters**: sort by response length, status, or error count to spot anomalies quickly.
- **Extensions support**: many Burp extensions enhance Intruder (wordlists, smart filters, rate-limit handling).
- **Pause/Resume & save results** for later triage.

Repeater

Burp Repeater is the manual request/response workbench inside Burp Suite. It’s the place you go when you want precise, hands-on control: edit a single HTTP request, re-send it, inspect the server’s response, tweak again, repeat — until you understand the behavior or craft a payload that works.

Main UI & features

- **Request pane** — full raw HTTP request you can edit (method, path, headers, body).
- **Response pane** — raw HTTP response, status, headers, and body; supports syntax highlighting and rendered HTML view.
- **Params/Headers view** — quick access to parsed parameters and headers for easy edits.
- **Hex view / Rendered view** — switch between raw hex and rendered HTML.
- **History / Tabs** — each saved request appears in its own tab (or multiple tabs) so you can keep different attempts separated.
- **Copy as curl / Copy full request** — export the request for scripts or external tools.
- **Save / Comments** — save interesting requests and add notes for later.
- **Search & compare** — search response text or visually compare requests/responses to spot differences.

Typical workflow

1. **Capture** a request via Proxy (or craft one in Repeater).
2. **Right-click → Send to Repeater** (or Ctrl+R).
3. **Switch to Repeater tab** and select the request you want to work on.
4. **Edit** the part you want to test (parameter value, cookie, header, HTTP verb, endpoint).
5. **Click "Go"** to send the request.
6. **Inspect response**: status code, response length, body, error messages, cookies, redirects, timings.
7. **Iterate** — tweak payloads, add encoders (manually or via macros/extensions), and resend until you get the behavior you want.
8. **Document** useful findings by saving the tab or copying the request/response.

Collaborator

Burp Collaborator is a service built into Burp Suite that helps you detect out-of-band (OOB) interactions — i.e., when a target server makes DNS/HTTP/SMTP (or other) requests to an external host as a result of your payload. It's essential for finding blind SSRF, blind SQL injection, blind command injection, and certain server-side request/lookup behaviors that don't return the result directly in the HTTP response.

How it works

When you enable Collaborator, Burp gives you a unique, random domain (and corresponding interaction endpoints). You include that domain in payloads you send to the target (for example as a URL, host, or header). If the target system resolves or connects to that domain, the DNS/HTTP/SMTP interaction is recorded by the Collaborator server and shown inside Burp so you can link the interaction back to the request that triggered it.

Types of interactions it records

- **DNS** lookups (very reliable for detecting server-side resolution).
- **HTTP(S)** requests (includes headers, path, query string, and sometimes body).
- **SMTP** (email delivery attempts).
- **Other TCP/UDP** interactions depending on server setup (less common).

Each interaction includes timestamps, remote IP (where available), protocol, the raw request/lookup data, and sometimes response metadata.

Built-in (public) vs private Collaborator server

- **Default (public) Collaborator**: Burp provides a public Collaborator server you can use immediately. Convenient for quick tests, but interactions go via Burp's public infrastructure — not ideal for sensitive targets, and may be rate-limited.
- **Private Collaborator server**: You can deploy your own Collaborator server (usually on a cloud VM with a public domain) and configure Burp to use it. This gives full control over logs, privacy, and reliability, and avoids sharing interaction data with third parties. For engagement against production or sensitive environments, prefer a private server (with authorization).

Typical workflow / examples

1. **Get a Collaborator payload**: In Burp, generate a unique collaborator URL/domain or use the Collaborator client.

2. **Inject it**: Place the domain in places that might cause external lookups: URLs, headers (Host, Referer, X-Forwarded-For), user-agent, CSS @import, XML external entities, LDAP/JNDI payloads, email fields, etc.

   Example payloads:

   - http://<random-id>.collaborator-domain/
   - ${jndi:ldap://<random-id>.collab-domain/a}

3. **Send request** via Proxy / Repeater / Intruder / Scanner.

4. **Monitor Collaborator**: Burp will show interactions in the Collaborator tab (or in issue reports from Scanner).

5. **Correlate** the interaction to the original request to confirm an OOB vulnerability.

Using Collaborator with other tools

- **Scanner (Pro)** uses it automatically to detect blind/OOB issues.
- **Intruder**: include collaborator payloads to fuzz for OOB triggers (use sparingly to avoid lots of external traffic).
- **Repeater / Manual tests**: very useful for validating suspected blind vulnerabilities—manually place a collaborator domain and observe any resulting interaction.

Interpreting results

- **DNS interaction present**: the target resolved the collaborator domain — indicates the target performed a name resolution (could be server-side code performing a hostname resolution).
- **HTTP request present**: the target connected to the collaborator URL — stronger evidence of SSRF, server-side fetching, or template inclusion. The request may include headers revealing an internal IP, server software, or other metadata.
- **Timing**: note timestamps and whether multiple interactions happen after repeated attempts — may indicate retries or asynchronous processing.

Practical tips & best practices

- **Use short, descriptive identifiers** in your collaborator subdomain if you're running many tests (helps correlation).
- **Prefer private Collaborator** for real engagements or sensitive targets.
- **Respect scope and authorization** — Collaborator interaction means you caused network calls from the target; always have permission.
- **Avoid noisy mass testing on production** — OOB tests can create unexpected outbound traffic. Throttle and coordinate with stakeholders.
- **Combine with other detection methods**: sometimes a DNS lookup occurs but no HTTP request — that still indicates interesting behavior.
- **Check for intermediate proxies or sanitization** — some apps will resolve DNS locally, others may proxy requests via web application firewalls or CDNs which may hide internal details.
- **Use Collaborator to detect blind data exfiltration** — crafted payloads can force a server to include secrets (like session IDs) in outgoing requests (be careful with sensitive data and legal rules).

Common pitfalls

- **No interaction ≠ no vulnerability** — absence of Collaborator hits doesn't prove the target is safe. The particular vector or context might not trigger external calls.
- **WAFs / network filters** may block outgoing requests or rewrite payloads, producing false negatives.
- **Rate limits or caching** on the target or DNS can cause missing interactions.
- **Public collaborator detections logged externally** — if using the public service, logs are not private.

Example use-cases

- Detect blind SSRF: inject collaborator URL into user-controlled URL parameter; if the server fetches it, you'll get an HTTP interaction.
- Detect blind command injection: craft a command that causes an HTTP/DNS lookup to collaborator (e.g., ; curl http://<id>.collab/), then watch for hits.
- Detect XXE / XML external entity resolution: include a collaborator domain as the entity system id; a resolver will fetch it if vulnerable.
- Confirm JNDI/Ldap injection (e.g., Log4Shell style): insert a JNDI payload referencing the collaborator domain and await LDAP/DNS/HTTP interactions

Sequencer

Burp Sequencer is a statistical analysis tool for evaluating the *randomness* (unpredictability) of tokens and session identifiers issued by a web application. Use it when you want to know whether session IDs, anti-CSRF tokens, password reset tokens, or other secret values are strong enough to resist guessing or prediction.

When to use Sequencer

- Testing session identifiers (session cookies, auth tokens).
- Testing CSRF tokens, password reset tokens, one-time tokens.
- Validating tokens after fixes or library upgrades.
- Triaging suspected weak randomness in blind attacks (if you can collect many tokens).

Main UI & outputs (what you'll see)

- **Capture / Samples pane** — a list of token samples you've captured or uploaded.
- **Configure sample extraction** — tell Sequencer which portion of the request/response is the token (it can auto-detect or you can define a custom extractor/regex).
- **Sample size setting** — how many samples to collect/analyze.
- **Statistical results** — numeric metrics such as:

    Entropy (bits) — how many bits of unpredictability the token carries. Higher is better.

    Chi-square — measures how the token's byte/character frequencies deviate from uniform distribution.

    Mean value — average byte/character value (used for some tests).

    Serial correlation — checks whether successive characters/bits are correlated (non-random sequences often show correlation).

    Monte Carlo Pi estimate — an unconventional but useful randomness check derived from sampling.

- **Visualizations**:

  Character frequency distribution (which characters/bytes are common).

  Bit distribution (0 vs 1 distribution across bit positions).

  Runs / repeats (shows duplicate tokens / repeated patterns).

- **Verdict / comments** — Sequencer will flag obvious problems and let you inspect areas of concern.

Typical workflow

1. **Capture tokens**:

   Interact with the app (login, request password reset, refresh session) while Burp Proxy records requests/responses.

   Or use a scripted flow (e.g., with Repeater, a macro, or an API client) to fetch many tokens quickly and reliably.

2. **Send samples to Sequencer**:

   Right-click the captured response(s) and choose **Send to Sequencer**, or manually paste/upload a list of tokens.

3. **Define the token extractor**:

   Let Burp auto-detect or set a regex/position to extract the token string from the response.

4. **Set sample size**:

   Choose a meaningful sample size (hundreds to thousands of tokens). Larger = more reliable.

5. **Start analysis**:

   Click **Start**. Sequencer will compute metrics and display graphs.

6. **Interpret results**:

   Look at entropy first; then check chi-square, serial correlation, and distributions. Investigate any suspicious patterns (e.g., predictable prefixes, repeated values, low entropy).

7. **Follow up**:

   If weak randomness is indicated, try to reproduce, capture tokens from different accounts/requests, and explore whether partial prediction or enumeration is possible in practice.

How to interpret key metrics

- **Entropy**: measured in bits. Rough guideline:

  < 32 bits — *very weak* (easy to brute-force / guess).

  32–64 bits — *weak/moderate* (may be risky depending on context).

  128 bits — *strong* for most purposes.
  (Exact acceptable values depend on token usage and attacker capability; prefer ≥128 bits for session secrets.)

- **Chi-square**: large deviations from expected values indicate non-uniform distribution — watch for obvious peaks in certain characters/bytes.
- **Serial correlation**: values near 0 indicate low correlation (good). Significant non-zero values indicate predictable relationships between successive characters.
- **Frequency charts**: repeated characters, limited alphabet (e.g., only hex digits with low variability), or structured formats (timestamps, counters, user IDs embedded) are red flags.
- **Duplicates**: any repeated tokens in the sample are a critical problem (token reuse).

Practical tips & best practices

- **Collect enough samples** — 1000+ is a good goal for decent statistical confidence; if you can't collect that many, be cautious interpreting results.
- **Remove fixed prefixes/suffixes** before analysis — if tokens have a static prefix (e.g., sess_), strip it out so Sequencer analyzes only the random part.
- **Test tokens from real issuance flows** — single-sample tokens issued under different conditions (e.g., after login vs API) may differ.
- **Automate safe collection** — use scripts that request tokens while respecting rate limits and authorization; don't flood production.
- **Combine with practical tests** — statistical weakness doesn't always translate to practical exploitation; try targeted guessing/enumeration to validate risk.
- **Watch for structured tokens** — e.g., JWTs contain three base64 parts; analyze the signature/nonce part, not the human-readable header/payload.
- **Consider token lifetime** — short-lived tokens may tolerate fewer bits of entropy depending on context, but proper entropy is still vital for safety.

Limitations of Sequencer

- **Statistics ≠ absolute proof** — Sequencer shows evidence of weak randomness, but a "pass" doesn't guarantee perfect security; a "fail" requires further investigation.
- **Small samples cause false results** — too few samples produce unreliable statistics.
- **Structured tokens can confuse tests** — tokens containing timestamps, counters, or encoded metadata might appear non-random even when the unpredictable portion is fine. You need to isolate the actual randomness.
- **Context matters** — how a token is used (single-use, scoped, tied to IP) affects the real-world risk of lower entropy.

Decoder

Burp Decoder is a small, very useful workbench inside Burp Suite for converting and inspecting encoded data. It's the tool you grab when you see some obfuscated value in a request/response (Base64, hex, URL-encoding, HTML entities, Unicode escapes, etc.) and want to quickly decode, re-encode, or try several transforms in sequence to reveal the underlying data.

Main features & transforms you'll use most

- **URL encode / decode** — decode %xx escape sequences.
- **Base64 encode / decode** — handle standard and many common variants (with/without padding).

- **Hex (hexadecimal) encode / decode** — convert raw bytes ↔ hex strings.
- **HTML / XML entity encode / decode** — &lt;, &#x27;, etc.
- **Unicode / escaped sequences** — \uXXXX and \xNN style escapes.
- **Character set conversions** — view/convert between common encodings (UTF-8/16, ISO-8859-1, etc.).
- **Case transforms / URL-safe variants** — quick normalizations.
- **Multi-step (chained) transforms** — run several decodes in order (very handy for layered obfuscation).
- **Send to other Burp tools** — Repeater, Intruder, Comparer, or save for later analysis.
- **Raw vs rendered view** — see both the raw bytes and rendered text.

Typical workflow

1. **Capture** the interesting request/response in Proxy (or copy the suspicious value).
2. **Right-click → Send to Decoder** (or open Decoder and paste).
3. **Select a transform** (e.g., “Base64 decode”) and click **Decode**.
4. **If result is still encoded**, apply another transform (e.g., URL decode → Base64 decode).
5. **When satisfied**, send the plain text to Repeater/Intruder or copy it for a POC/report.

Small examples

- Base64 decode: aHR0cDovL2V4YW1wbGUuY29t → http://example.com
- URL decode: %3Cscript%3Ealert(1)%3C%2Fscript%3E → <script>alert(1)</script>
- Chained example: URL decode a value, then Base64 decode the result to reveal the payload.

Practical tips

- **Try chaining transforms** — many real-world payloads are layered (e.g., base64 inside URL-encoding).
- **Be careful with character encodings** — if decoded output looks garbled, try a different charset (UTF-8 vs ISO-8859-1).
- **Use “send to”** to quickly validate decoded payloads in Repeater or to fuzz them with Intruder.
- **Copy raw bytes** when analysing binary blobs or compressed data.
- **When unsure, try common transforms in sequence** (URL → Base64 → Hex) — it often reveals hidden data quickly.

Comparer

Burp Comparer is a lightweight but very handy tool inside Burp Suite for doing side-by-side diffs of two pieces of data (requests, responses, or any text/hex). Use it when you want to quickly spot small differences between two responses , for example before/after injection, two sessions, or an original vs manipulated request.

Main UI & features

- **Left / Right panes** — paste or send two items to compare.
- **Text vs Hex view** — switch to hex to inspect binary differences or hidden bytes.

- **Synchronized scrolling** — keep corresponding parts aligned while you inspect.
- **Highlighting** — differences are highlighted so you can jump between change points.
- **Send-to integration** — send requests/responses from Proxy, Repeater, Intruder, or Decoder directly to Comparer.
- **Copy / Save** — copy the diffed result or save either pane for reporting.

Typical workflow

1. **Capture** the two items you want to compare (e.g., original response and response after injecting payload).
2. **Right-click → Send to Comparer** (or open Comparer and paste).
3. **Switch views** if necessary (use hex to reveal non-printable differences).
4. **Inspect highlights** and use arrow/jump controls to move between differences.
5. **Extract useful info** (reflected payloads, changed cookies, new headers) and send interesting items to Repeater/Intruder for further testing.

Practical use-cases

- **Confirming reflection**: compare a response before and after an XSS payload to see exactly how your payload is echoed back.
- **Analyzing subtle SQLi/logic results**: compare pages returned for true/false payloads to identify differences you can use for exploitation.
- **Session/token comparison**: see what bits of a session cookie or token changed between requests.
- **Regression testing**: verify that a patch changed only intended parts of responses.
- **Binary analysis**: use hex view to detect byte-level changes in binary responses or encoded blobs.

Tips & best practices

- **Use hex view for hidden differences** — whitespace, null bytes, or non-printable characters are easier to catch in hex.
- **Compare just the relevant parts** — if a large HTML page hides small differences, copy only the fragment (e.g., body or parameter area) into Comparer to reduce noise.
- **Combine with other tools**: once you spot a difference, send the response to Repeater to manually probe the cause or to Intruder to automate variations.
- **Use for proof-of-concept**: include screenshots or the Comparer output in reports to clearly show exploited differences.
- **Synchronized scrolling helps** when comparing long documents — keep context while jumping between diffs.

Limitations

- **Not a full diff/merge tool** — it's lightweight and meant for quick manual inspection, not large code merges.
- **No advanced diff algorithms** — mainly character/byte differences; complex structural comparisons (XML/JSON semantic diffs) may require external tools.

- **Large files** — extremely large responses can be slow to load; trim to relevant sections where possible.

Logger

In Burp Suite, the **Logger** tab (often accessed via the Logger++ extension) is a centralized place to capture, view, filter, tag, and export HTTP requests and responses. While Burp's native Proxy History stores captured traffic, the Logger tab adds structured, persistent, and customizable logging for more efficient testing and reporting.

Purpose

The Logger tab is designed to:

- Maintain a centralized log of all requests/responses you care about.
- Filter out noise (static assets, irrelevant domains) so you focus only on in-scope traffic.
- Allow tagging, commenting, and categorization of entries for easy triage.
- Provide export options for CSV/JSON to create reports or import into external tools.
- Persist logs across sessions and projects for reproducibility.

Main Components / Features

1. Log List / Table

    Shows captured entries in rows with columns like timestamp, host, method, path, status, response length, and optional tags.

    Supports sorting and column customization.

2. Filters

    Filter traffic by domain, path, status, method, MIME type, or by custom regex rules.

    Helps focus on relevant traffic (API calls, dynamic pages, forms).

3. Entry Details Pane

    Clicking a log entry opens a pane showing full request and response.

    Can view headers, body, cookies, and metadata.

    Supports switching between Text / Raw / Hex views.

4. Tagging & Comments

    Add tags like XSS, SQLi, OOB-HIT, AUTH-FAIL to entries.

    Write comments for findings or notes for reports.

5. Export Options

    Export selected entries or entire log to CSV or JSON.

    Optionally redact sensitive fields (cookies, auth headers, personal data).

6. Integration / "Send to" Features

    Right-click entries to send them to Repeater, Intruder, Comparer, Decoder, or back to Proxy.

    Makes workflow seamless for manual testing or automation.

Typical Workflow

1. Capture traffic: Let Proxy or other Burp tools collect HTTP requests/responses.
2. Send to Logger tab: Either automatically (via Logger++ rules) or manually via right-click → "Send to Logger".
3. Apply filters: Narrow down to in-scope hosts, content types, or endpoints of interest.
4. Tag & comment: Label entries that indicate potential vulnerabilities or points of interest.
5. Inspect entries: Open individual log entries to review headers, body, or parameters.
6. Export or share: Generate CSV/JSON reports for audits, teammates, or documentation.

Practical Uses

- Track all requests and responses during a testing session for documentation and reproducibility.
- Quickly find interesting or suspicious requests via filters and tags.
- Validate payload reflections or changes using logging history.
- Hand-off logs to developers or management with clean, redacted CSV/JSON exports.
- Detect out-of-scope or noisy traffic and refine scope for Proxy and Scanner.

Organizer

The **Organizer tab** in **Burp Suite Professional** is a built-in project management area that helps penetration testers **plan, document, and manage their testing workflow**. It allows you to create and track **tasks, issues, and notes** directly inside Burp, keeping your assessment structured and well-documented without relying on external tools.

Purpose of the Organizer Tab

The Organizer tab acts as your **central workspace** for:

- Managing **testing activities** (like to-do lists and progress tracking).
- Recording **findings and vulnerabilities** discovered during testing.
- Writing **notes and observations** linked to specific requests or hosts.
- Keeping all your documentation **within the same Burp project**, making reporting easier later.

In short, the Organizer helps you **plan, track, and report** your penetration testing tasks effectively.

Main Components

1. Tasks

This section lets you manage your testing workflow.

- You can **create new tasks** such as "Check input validation on /login" or "Review cookies for Secure flag".
- Each task can include:

  Title – short description of the task.

  Details/Description – notes or steps for what to do.

  Priority – (Low, Medium, High).

Status – (Open, In Progress, Completed).

Linked items – attach HTTP requests, responses, or URLs related to that task.

- Tasks help ensure you don't miss any important checks during testing.

Example:
Task: Test for SQL Injection on /product?id=
Priority: High | Status: In Progress | Linked: Repeater request #12

2. Issues

The **Issues** section is used to document vulnerabilities or weaknesses you find during testing.

- Each issue entry includes:

  Title – e.g., "Reflected Cross-Site Scripting on /search".

  Severity level – Informational, Low, Medium, High, or Critical.

  Description – explain what the issue is.

  Impact – what could happen if it's exploited.

  Remediation advice – how to fix it.

  Linked items – attach related requests or responses for evidence.

- This makes it easy to later export or summarize findings for reports.

Example:
Issue: Missing HTTPOnly flag on session cookie
Severity: Medium | Linked request: GET /login
Remediation: Set the HTTPOnly attribute in Set-Cookie header

3. Notes

The **Notes** area works like a built-in notepad for your test.

- You can add **general notes**, payloads, test observations, or summaries of what you've tried.
- Notes can be linked to specific requests or hosts.
- Useful for recording **payloads that worked**, **endpoints of interest**, or **session details**.

Example:
Note: Tried XSS payload <script>alert(1)</script> on /feedback, reflected in response body.

Integration with Other Tabs

- You can **right-click** any HTTP request or response (from Proxy, Repeater, Intruder, etc.) → **"Add to Organizer → Task / Issue / Note"**.
- This automatically links that request to the new entry in the Organizer tab.
- Helps you maintain **traceability**, so every finding or note is backed by evidence from the captured traffic.

<u>Extensions</u>

The **Extensions** feature in **Burp Suite** allows you to **extend Burp's functionality** beyond its built-in tools. You can install, create, and manage **custom add-ons** (called *extensions*) that enhance automation, reporting, logging, scanning, and more.

These extensions can be downloaded from the **BApp Store** or developed manually using the **Burp Extender API**.

Purpose of Extensions

Extensions make Burp Suite more powerful and flexible by allowing you to:

- Add **new tools or tabs** inside Burp.
- Automate **custom tasks**, such as request modification or vulnerability detection.
- Integrate Burp with **external tools** like databases, browsers, or SIEMs.
- Enhance existing tools like **Proxy, Intruder, Scanner, or Logger**.
- Customize Burp according to your **testing style or project requirements**.

Main Components in the Extensions (Extender) Tab

The **Extender tab** in Burp Suite has several sub-sections:

1. Extensions

This is the main area where all installed extensions are listed.

- Shows details like **Name, Version, Author, Status, and Type (Python, Java, or JavaScript)**.
- You can **enable/disable, remove, or reload** extensions.
- Displays **output and errors** generated by each extension (useful for debugging).

2. BApp Store

- The **BApp Store** is Burp's official marketplace of tested, community-developed extensions.
- You can directly install extensions with one click.
- It includes both free and professional-grade add-ons.

Steps to install:

1. Open **Extender → BApp Store**.
2. Browse or search by name (e.g., "Logger++", "Autorize", "ActiveScan++").
3. Click **Install** → Burp downloads and activates it instantly.

3. Options

This section allows you to configure **Burp's API and language environments** for running custom extensions.

- **Supported Languages:**

  Java (default, runs via built-in Jython/JRuby)

  Python (using **Jython**)

  JavaScript (using **Rhino engine**)

- You can set the path to your **Jython or JRuby jar file** if you want to run Python or Ruby extensions.
- Configure extension logging and output display settings.

4. Output / Errors

These tabs show:

- **Output:** messages printed by extensions (e.g., logs, status updates).
- **Errors:** debugging information if an extension crashes or fails to run.

Useful for developers writing or debugging their own extensions.

Developing Custom Extensions

Burp provides the **Burp Extender API** that developers can use to create their own extensions.

Supported Languages:

- **Java** (most powerful and best integrated).
- **Python** (via Jython).
- **JavaScript** (via Rhino).

What You Can Do with the API:

- Intercept, modify, or drop HTTP requests/responses.
- Create new tabs or UI components inside Burp.
- Automate repetitive testing tasks (like payload insertion or comparison).
- Extend existing tools (Proxy, Intruder, Scanner).
- Build integrations (e.g., send findings to Splunk or Slack).

Example:
A custom Python extension that automatically checks responses for a specific keyword and highlights the matching requests.

Benefits of Using Extensions

Adds flexibility and power to Burp Suite.
Saves time through automation.
Enables deeper or more specialized security testing.
Allows integration with other cybersecurity tools.
Customizable — you can tailor Burp to your workflow.

Precautions

- Only install extensions from **trusted sources** (preferably BApp Store).
- Some extensions may slow down Burp if they process too many requests.
- Review extension permissions and output to avoid exposing sensitive data.
- Always test custom scripts in a **safe environment** first.