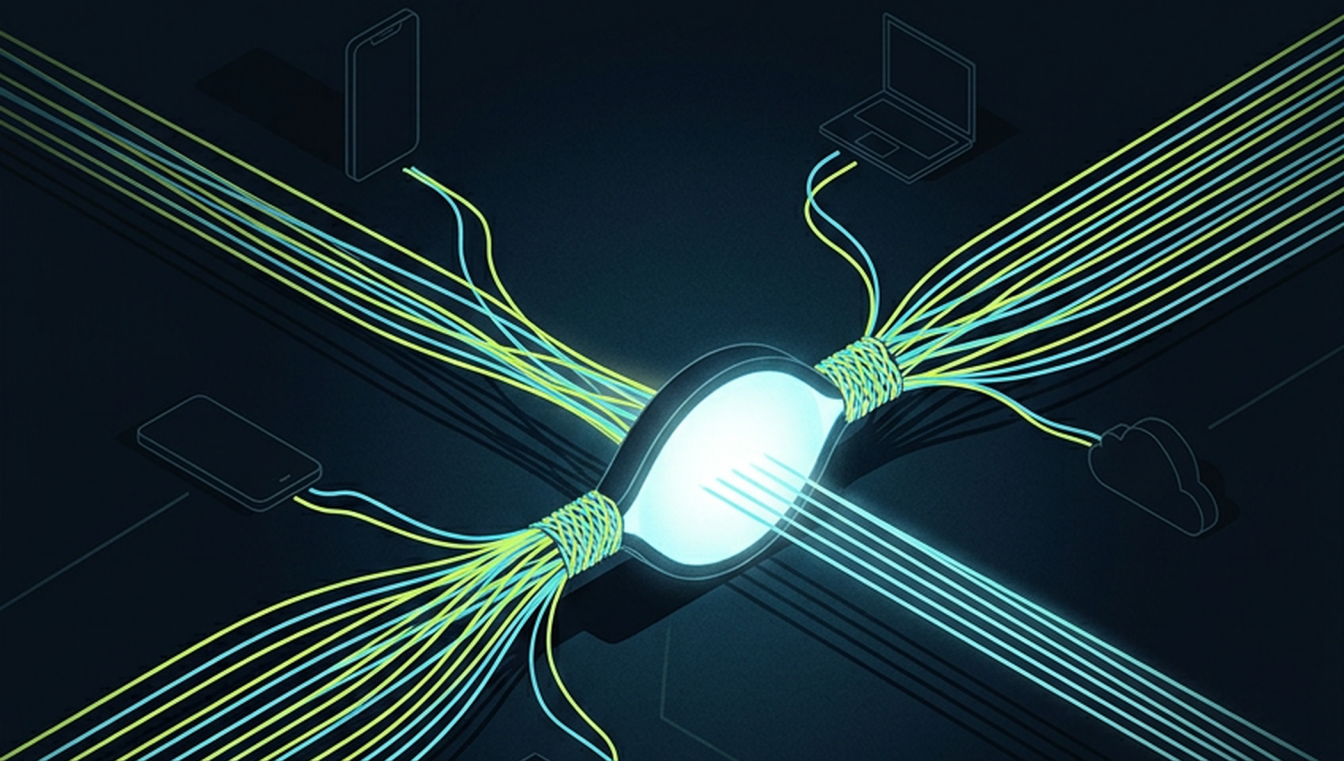




PRACTICAL MITMPROXY

Mastering mitmproxy 12 for Security Researchers,
Mobile App Testers, and API Developers

INTERCEPT, EDIT AND REPLAY LIVE TRAFFIC
BYPASS CERTIFICATE PINNING ON ANDROID AND iOS
WRITE ADDONS AND AUTOMATE API SECURITY TESTS

GROUND TRUTH BOOKS

Practical mitmproxy

Mastering mitmproxy 12 for Security Researchers,
Mobile App Testers, and API Developers

Ground Truth Books

This book is available at <https://leanpub.com/practical-mitmproxy>

This version was published on 2026-10-12



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Ground Truth Books

Contents

Chapter 1: Introduction	1
What this book covers	1
Who this book is for	4
How to use this book	5
A note on versions and commands	6
 Getting Started	 8
Chapter 2: Installing mitmproxy 12.2.3	9
Installation on Linux, macOS, and Windows	9
Verifying the installation	9
First run: what lands in ~/.mitmproxy	9
Upgrading and uninstalling	10
 Chapter 3: The Three Frontends	 11
mitmproxy: the interactive TUI	11
mitmweb: the web interface	11
mitmdump: the scriptable CLI	11
Shared command-line options	11
 Interception and Inspection	 12
Chapter 4: How mitmproxy Intercepts HTTPS	13
The proxy core and sans-I/O design	13
TLS termination and certificate generation	14
The mitmproxy CA and trust stores	16
What interception cannot see	17
 Chapter 5: Trusting the mitmproxy CA	 20
Installing the CA on Linux	20

Installing the CA on macOS and Windows	20
Browsers and runtimes with their own trust store	21
Troubleshooting CA trust	21
Chapter 6: Filter Expressions and Flow Inspection	22
Filter expression syntax	22
Common filter patterns	22
Inspecting flows in the TUI	22
Inspecting flows in mitmweb	22
Modification and Replay	23
Chapter 7: Modifying Requests and Responses	24
Interactive modification in the TUI	24
modify_headers, modify_body, map_local and map_remote	24
Modification with addons	24
Common modification recipes	24
Chapter 8: Replaying Requests	25
Client-side replay	25
Server-side replay	25
Replay in mitmweb	25
Replay caveats and pitfalls	25
Addon Development	26
Chapter 9: Addon Architecture and Events	27
Anatomy of an addon	27
Event hooks and their order	27
Loading and running addons	27
The abbreviated scripting syntax	27
Chapter 10: Options, Commands, and Hooks	28
Declaring addon options	28
Defining custom commands	28
Custom contentviews	28
Exposing addons in the UI	28
Chapter 11: Working with Flows and Messages	29

The flow object model	29
HTTP request and response objects	29
Headers, cookies and bodies	29
Reading flow files from Python	29
Deployment Modes	30
Chapter 12: Choosing a Mode: Regular, Transparent and Reverse . . .	31
The mode specification syntax	31
Choosing a mode for your task	31
Regular proxy mode	31
Transparent mode on Linux	31
Transparent mode on macOS	31
Reverse proxy mode	31
Chapter 13: Upstream, SOCKS, and WireGuard Modes	33
Upstream proxy mode	33
SOCKS proxy mode	33
WireGuard mode	33
Combining modes	33
Chapter 14: Local Capture Mode	34
Local redirect on Linux	34
Local redirect on macOS	34
Local redirect on Windows	34
Capturing Docker containers	34
Mobile and Advanced Protocols	35
Chapter 15: Intercepting Android and iOS Apps	36
Android emulator setup	36
Installing the CA as a system certificate	36
Android 14 and newer CA installation	36
Physical Android devices	36
WireGuard for Android apps	36
Emulator networking and localhost	36
iOS proxy and CA setup	37
Full trust for TLS on iOS	37

Chapter 16: Bypassing Certificate Pinning	38
What certificate pinning is	38
Frida and objection basics	38
Universal unpinning scripts	38
Pinning on iOS	38
Pinning bypass caveats	38
Chapter 17: HTTP/2, HTTP/3, and WebSockets	39
HTTP/2 interception	39
HTTP/3 and QUIC	39
WebSocket message handling	39
Protocol-specific addon events	39
Automation and Reference	40
Chapter 18: Exporting Flows: HAR, curl and OpenAPI	41
Saving and loading flow files	41
Streaming large transfers	41
Exporting HAR	41
Copying requests as curl and raw	41
From flows to an OpenAPI spec	41
Chapter 19: Automating API Security Testing	42
A harness you drive with addons	42
Building a fuzzing addon	42
Automated replay and regression	42
Auth token rotation	42
Reporting what the tests found	42
Chapter 20: Conclusion	43
What you can now do	43
Where to go next	43
Keeping current with mitmproxy	43
Appendix A: Command-line options reference	44
How options reach the tools	44
Invocation and file I/O	44
Proxy modes and listeners	44
TLS and certificates	44

Modification and replay	44
Protocol and bodies	44
Frontend-specific flags	45
Environment variables	45
Appendix B: Filter expression reference	46
Operators	46
Combining expressions	47
Regex arguments	48
Flow selectors	48
Where filters apply	49
Appendix C: Addon event reference	51
Addon event reference	51
Appendix D: Flow export formats	53
At a glance	53
The .mitm flow file	53
HAR (HTTP Archive)	53
Export commands: curl, HTTPie, raw	53
OpenAPI 3.0 through mitmproxy2swagger	53
Glossary	54
References	55

Chapter 1: Introduction

What this book covers

A mobile app talks to its backend over HTTPS, and so do a single-page app and an API client. To a packet capture, that conversation shows endpoints and ciphertext. The payload, from the request line down to the body, stays hidden until someone sits in the middle of the TLS connection and terminates it.

mitmproxy is that someone. It is an interactive proxy that can terminate the TLS on a client connection, at which point the request and response become a single flow you can inspect or modify [1, 2]. The same engine runs behind three faces, from a terminal UI to a browser UI to a scriptable command line [1]. Its addon system hooks one Python API into each stage of a flow's life [3, 4]. A set of deployment modes decides how the client's traffic reaches the proxy in the first place [5].

This book turns that into a working mental model. When you finish, you will have installed mitmproxy 12.2.3 and trusted its CA. You will have inspected and modified live HTTPS traffic and written addons that automate security tests, and you will have deployed the proxy in whatever mode a client forces on you.

The tool at a glance

The project ships three frontends over one core, and the docs are explicit about the relationship: mitmproxy, mitmweb, and mitmdump are “different frontends to the same core proxy” [1, 4]. From flow filtering to replay to addons, behavior is identical across the three, so skills transfer when you switch faces [4].

mitmproxy is the terminal UI. It keeps all flows in memory, which makes it right for small-ish samples that you want to inspect and manipulate by hand [1]. Pressing ? opens context-sensitive help from any screen [1]. When you are mid-session and need to know what a request actually contained, the console gets you there faster than a browser [4].

`mitmweb` is the browser face. It serves the same data as a web application, at `http://localhost:8081` by default, in the spirit of Chrome's DevTools [6]. Pick it when you want a GUI or want to hand a capture to a colleague who does not live in a terminal [4].

`mitmdump` is the command-line face, which the docs describe as “tcpdump for HTTP” [1, 7]. It is non-interactive, so it drives scripting and automation, including offline analysis of saved captures [6, 4]. You will see it most in this book, because it is the face you can run against files and program against.

One consequence of the single core matters early. Built-in features are themselves addons [4, 8], so flow filtering and content-view plugins run on the same Python API you write against [4]. Chapters 9 and 10 build on that. There you write the same kind of addon objects the tool ships with [8, 3], and you can load them over the captures this chapter reads.

The trust model

The whole scheme rests on a small piece of home-grown PKI. The first time you start any frontend, `mitmproxy` creates a certificate authority (CA) in the config directory, `~/.mitmproxy` by default [9]. For each domain a client visits it generates a certificate on the fly and signs it with that CA [9, 1]. A client that has not installed the CA fights a certificate warning on each new domain. A client that trusts it lets the proxy read and rewrite the traffic [10, 9]. Chapters 4 and 5 cover the mechanics and the one-time trust step. Chapters 15 and 16 return to it on a device, where you install the CA and switch on full trust before interception works.

Deployment modes

Interception also depends on getting the client's traffic to the proxy. The default is regular mode, an HTTP(S) proxy on port 8080 that the client is configured to use [10, 5]. Out of the box the proxy binds every interface, so `mitmdump` logs listening at `*:8080` and the proxy answers from the LAN; `--listen-host 127.0.0.1` keeps it on the local machine [11, 12]. From there the docs branch into eight other modes: local capture for applications on the same machine, WireGuard for whole devices, reverse mode in front of a server, and transparent, upstream, SOCKS, TUN, and DNS modes for the special cases [5]. Chapters 12 through 14 pick the right mode for a given client.

What mitmproxy 12.2.3 adds

The book targets the 12.x generation, and one feature separates it from earlier releases. Version 12.0.0 shipped in April 2025 [13] with interactive contentviews. You can now edit the prettified, human-readable rendering of a binary body, and mitmproxy re-encodes it back into the original binary format [14]. Tinkering with a binary payload, a Protobuf or MsgPack body for instance, no longer means hunting through raw bytes, and the content views for those formats are now written in Rust [15, 8]. Version 12 also added TLS 1.3 post-handshake authentication support and multi-flow selection in mitmweb [15].

Newest in the series is 12.2.3, released 12 May 2026 [16]. The 12.2 line reads like a polish release, and the 12.2.3 changelog fits that bill: it shortens the leaf certificates mitmproxy generates to 197 days, keeping them under Chromium’s 200-day limit, and fixes a few bugs, among them one in mitmweb and one in HAR export [8]. You will use the new machinery later: chapter 11 edits a binary body through the interactive content views, and chapter 18 ships your recorded flows out as HAR.

Two earlier milestones matter for what you can capture. mitmproxy 11 shipped full HTTP/3 support in transparent and reverse proxy modes, and WireGuard and local modes support it too [17, 18], so QUIC-only services stayed visible. The tool speaks HTTP/1, HTTP/2, HTTP/3, and WebSockets out of the box [4].

How the chapters fit together

This is a 20-chapter book, and the route map is easiest to hold as six groups.

Chapters 2 through 5 lay the foundation. You install mitmproxy 12.2.3 and meet the three frontends, then see how the proxy terminates TLS and why trusting its CA unlocks interception [9, 4].

Chapters 6 through 8 are the day-to-day. Filter expressions pick flows out of a capture by host or status code, and they can also match on content [12]. The interactive views show each byte of a request and response, and you can change what crosses the line interactively or in a script [1]. Replaying requests (client-side) or recorded responses (server-side) is how you test a fix without its server [19].

Chapters 9 through 11 open the engine room. An addon is a Python object whose methods fire by name as events happen [3, 20]. Options and commands

make addons first-class citizens of the UI, and the flow and message objects are the data model the addon API reads and writes [2].

Chapters 12 through 14 deploy the proxy. Regular, transparent, and reverse mode cover most work. Upstream and SOCKS modes chain mitmproxy behind another proxy, while WireGuard tunnels a whole device and local capture redirects one application's traffic on the same machine [5, 21].

Chapters 15 through 17 take interception to phones and modern protocols. They cover trusting the CA on Android and iOS and defeating certificate pinning, then the wire protocols that modern apps speak [18, 21].

Chapters 18 and 19 get the traffic out and put it to work. Flow export covers HAR and curl, while an OpenAPI spec can be generated from a captured session with the mitmproxy2swagger tool [22]. The same addon API drives an API security testing harness [4].

Who this book is for

The book exists for three readers who share one problem. They must see inside HTTPS traffic that is not theirs. Security researchers test APIs and web applications with it [7], and so do mobile app testers watching what a phone actually sends and API developers inspecting their own endpoints [4]. None of these jobs ends at “look at the request”. Each wants to modify a request or replay it in volume, which is where mitmproxy's addons and modes come in [3, 5].

Prerequisites are light. You should know what a method, a status code, and a header are, and you should be comfortable in a terminal. Chapters 9 through 11 and 19 use Python, and the examples stay small enough that a little experience suffices [20]. You do not need to know TLS or X.509 beforehand, and certificate pinning gets its own chapter. The book treats the certificate machinery as a black box you install once and trust [9]. When that box refuses to cooperate, the TLS handshake fails and the proxy logs why; chapter 5 shows that exact log line [11].

The examples run against a lab built for this book, labshop, a small shop API with deliberate flaws. Each capture in the book was recorded on this machine against labshop, and no third-party traffic appears anywhere. That keeps the listings reproducible and the security bugs in the lab.

`mitmdump -nr flows/basic-http.mitm` reads the shipped file without starting a proxy (`-n`) and prints a request line and a response line per flow [23]. That file holds seven plain HTTP/1.1 flows, from a form login that sets a session cookie and redirects to a failed login answered with 401; chapter 2 runs the same command and shows the full listing [23]. A later chapter's capture, `flows/shop-api-https.mitm`, records the same shop over HTTPS and HTTP/2 and contains the bugs the book exploits later: an IDOR on user profiles and an order shipped at a tampered price, plus an admin endpoint that trusts a forged cookie. Because the file sits on disk, each step of that analysis replays on your machine [23, 24].

How to use this book

Read chapters 2 through 9 in order on a first pass. They build one stack, and the later chapters assume it [1, 20]. The deployment chapters, 12 through 14, and the mobile chapters, 15 and 16, each answer one question, so read them when the question comes up.

Each chapter opens with a problem you can reproduce and shows the commands or add-on code that solve it. It closes on what the output looks like and why it works. Output blocks come from one of two places: `mitmdump 12.2.3` run against the shipped captures offline with `-n`, or a recorded terminal session of the same tool against the lab [24, 23]. Where a topic cannot be shown that way, installing a CA on a phone, for example, the chapter gives the documented commands and tells you what to look for instead [10]. Code blocks are runnable unless the sentence that introduces them says they are fragments.

Two conventions keep the examples honest. Omitted lines are marked with `...` on their own line [23]. The flow files in this lab are read-only, so commands that write fail. The examples do not depend on writing.



The sharpest trap in this book is `-w`. It feels natural to read a capture, run an add-on over it, and write the result back, the way the docs show for live use [1]. This lab's files are read-only, so that command errors out. Read with `-n`, and you are clear.

Filter expressions appear from Chapter 6 onward. `~u` and `~c` match on the URL and the status code, while boolean operators like `!` and `&` combine them

[12]. They work the same in `mitmdump`'s arguments and in the TUI's interactive view filter [12, 1]. Commands in this book avoid pipes and shell redirection, so you can run them in a plain terminal.

A note on versions and commands

Version numbers matter in a book about a moving tool, so this section pins down the contract. The book was checked against `mitmproxy` 12.2.3, released 12 May 2026 [16, 24]. The captures were taken with the standalone binary, which reports itself as `Mitmproxy: 12.2.3 binary` [24]. Chapter 2 prints the full `mitmdump --version` banner.

Installation gets its own chapter, but the one-line version you will type most often is `pipx install mitmproxy`, which keeps the tool and its dependencies away from your system Python [4]. The package requires Python 3.12 or newer [4], and the current binary releases ship with Python 3.14 [8, 24]. One install puts three frontend commands on your path [4, 1].

The three frontends share the core options, and most command-line flags are aliases for them [21]. The ones this book leans on, `-n` (do not start a proxy server), `-r` (read flows from a file), `-w` (write flows to a file), and `-s` (run an addon script), work the same in each; the docs say the `-s` flag is identical for all tools [1, 20]. Some flags stay per frontend: only `mitmdump` takes a trailing filter expression as a positional argument and understands `--flow-detail` [25], while `--intercept` and `--view-filter` belong to `mitmproxy` and `mitmweb` [21]. By default the regular proxy listens on every interface, reachable at `*:8080` from the LAN (see “Deployment modes” above) [11, 12]. `mitmweb` serves its interface on port 8081 [21].

The certificate files live in `~/ .mitmproxy`; chapter 2 inventories the four CA files and which platform each serves [9]. The CA is unique per installation, a deliberate security choice, because a shared one would let another `mitmproxy` user intercept your traffic [9].

The addon API changes between majors, and the project tracks those breaks in a dedicated changelog [26]. `mitmproxy` 12, for example, drastically simplified the `contentview` API and renamed `mitmproxy.dns.Message` to `mitmproxy.dns.DNSMessage` [26]. This book targets 12.2.3. If you run a later major, check that page before porting an addon [26].

After chapter 2 installs the tool, run `mitmdump --version` and `mitmdump -nr flows/basic-http.mitm` to see both outputs in full on your own machine [24, 23]. Chapter 2 also explains what that first-run message about a new CA means.

Getting Started

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 2: Installing mitmproxy 12.2.3

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Installation on Linux, macOS, and Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Linux

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

macOS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Docker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Verifying the installation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

First run: what lands in ~/.mitmproxy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Upgrading and uninstalling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 3: The Three Frontends

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

mitmproxy: the interactive TUI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

mitmweb: the web interface

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

mitmdump: the scriptable CLI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Shared command-line options

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Interception and Inspection

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 4: How mitmproxy Intercepts HTTPS

The proxy core and sans-I/O design

Open `flows/shop-api-https.mitm` with `mitmdump -nr` and the flows read as plain HTTP, because each one crossed two TLS sessions. The client's session terminates at the proxy, and the proxy's session runs to the server. What joins them is the proxy core, and its design explains both why interception works and why a paused flow does not deadlock the proxy.

mitmproxy's pre-7 core gave each connection its own thread that blocked on socket reads [55]. HTTP/2 broke that model, because it multiplexes many requests over one connection and a thread blocked on one request stalls the others. So did mitmproxy itself, which blocks whenever an addon intercepts a request and waits for the user to resume it [55]. The old core patched over both with hand-rolled multiplexing, which produced a steady stream of race conditions [55].

The 7.0 core, which 12.2.3 still runs, moved all protocol logic into synchronous code that never waits on a socket, the design called sans-I/O [55, 8]. An I/O server per proxy mode owns the listeners and hands each new connection to a `ProxyConnectionHandler` [56]. The handler turns socket activity into events (`Start`, `DataReceived`, `ConnectionClosed`) and feeds them to the protocol logic, which answers with commands: network instructions such as `SendData` or `OpenConnection`, or a hook that hands control to the addons [55]. When a blocking hook finishes, the server feeds a `HookCompleted` event back in, and the layer resumes as it would after any other event [57, 58]. The handler's `handle_hook` runs the addons and then waits for the flow to be resumed, which is how an intercepted flow pauses without stalling the proxy [56].

Protocol logic lives in layers, and events travel from the outermost layer inward [55]. The 7.0 design post shows one request round-trip as a test playbook, trimmed here [55]:

```

1  assert (
2      Playbook(http_layer)
3      >> Start()
4      >> DataReceived(client, b"GET http://example.com/foo?hello=1
   ↪ HTTP/1.1\r\nHost: example.com\r\n\r\n")
5      << HttpRequestHeadersHook(flow)
6      >> reply()
7      << OpenConnection(server)
8      ...
9  )

```

Two of those layers are TLS layers. The client-side TLS layer terminates the client's handshake and decrypts, while the server-side TLS layer opens the upstream handshake and encrypts. Between them the HTTP layer works on plaintext [55].

TLS termination and certificate generation

In an explicit proxy setup, the first message the client sends for an HTTPS target is a `CONNECT` request, not a request line. From the client's side, curl sees the proxy's `200 Connection established` before the server's own response [11]:

```

1  $ curl -sS -I -x http://127.0.0.1:8080 --cacert
   ↪ ~/.mitmproxy/mitmproxy-ca-cert.pem https://labshop.test:8443/catalog
2  HTTP/1.1 200 Connection established
3
4  HTTP/2 301
5  content-length: 0
6  location: /api/products
7  ...

```

A conventional proxy answers the `CONNECT` and then acts as a blind pipe. It forwards data in both directions without knowing anything about the contents, so the TLS negotiation happens over that pipe and the requests and responses that follow stay opaque to the proxy [59]. mitmproxy instead terminates both sides of the pipe, which is the MITM in its name.

The full explicitly proxied HTTPS sequence runs like this [59]:

1. The client connects to the proxy and sends `CONNECT example.com:443 HTTP/1.1`.

2. The proxy replies `200 Connection Established`, as if it had opened the pipe.
3. The client starts a TLS handshake and reveals the hostname in its Server Name Indication (SNI).
4. The proxy pauses the client's handshake and opens its own connection to the server, completing a TLS handshake using that SNI hostname.
5. The server responds with its certificate, and the proxy reads the Common Name and the Subject Alternative Names from it, a step the docs call upstream certificate sniffing [59, 9].
6. The proxy forges an interception certificate for that hostname, signed by its own CA, and resumes the paused client handshake with it.
7. The client sends its request over the client-side TLS session.
8. The proxy forwards the request over the server-side TLS session and relays the response back.

Step 5 is why the interception certificate matches the site. The client verifies that the certificate names the domain it is connecting to and aborts otherwise, so the name must be right [59]. The `CONNECT` target usually supplies it, but the client may `CONNECT` to a bare IP address that names no host. Upstream sniffing recovers the CN from the real certificate [59]. SNI complicates the sniffing in turn. Connecting upstream without SNI returns a default certificate that may have nothing to do with what the client expects, so the proxy lets the client's handshake run just past the SNI value before pausing it [59]. Upstream sniffing is on by default and can be disabled with the `upstream_cert` option [9]. In transparent mode the proxy has no `CONNECT`, so the name comes from SNI and, with `upstream_cert` on, from the certificate sniffed off the original destination [59, 9]. Without a custom certificate, a `ClientHello` with no SNI may leave the proxy unable to forge a certificate the client trusts [18].

The name feeds the forgery. The proxy draws on its CA to mint “dummy certificates for each visited website” on the fly [9]. Between steps 4 and 6 it generates a per-host leaf, so the client's certificate chain terminates at your CA while the upstream chain terminates at the server's real CA.

Here is what the proxy sees on its side after termination, from the `shop-api-https.mitm` capture that later chapters exploit. This is `mitmdump` rendering one of the 11 flows, and the second order posts a tampered price [60]:

```

1 127.0.0.1:36282: POST https://shop.localhost:8443/api/orders HTTP/2.0
2   ...
3   content-type: application/json
4   content-length: 51
5
6   {
7       "items": [
8           {
9               "product_id": 102,
10              "qty": 2,
11              "price": 0.01
12          }
13      ]
14  }
15
16 << HTTP/2.0 201 Created 42b
17   ...
18   {
19       "order_id": 5001,
20       "user_id": 1,
21       "total": 0.02
22   }

```

None of that JSON ever crossed the network in the clear. It exists only in the plaintext leg between the two TLS layers. The capture file stores the same decrypted form, which is exactly why you can read and edit HTTPS flows offline with `-nr`.

Two options let you steer the forgery. `--certs [domain=]path` replaces the generated leaf with your own PEM certificate for a domain, instead of one signed by the mitmproxy CA [9]. `tls_version_client_min` (default `TLS1_2`) and the related `tls_*` options set the protocol floor on each leg independently [21]. If you also want to decrypt the packets on the wire in Wireshark, set `SSLKEYLOGFILE` to a writable file before starting mitmproxy. It logs the master secrets for both legs [61].

The mitmproxy CA and trust stores

The first time mitmproxy runs, it mints its certificate authority in the config directory, `~/.mitmproxy` by default [9, 40]. The CA is generated uniquely per installation and is never shared between installations, so other mitmproxy users cannot use it to intercept your traffic [9]. Two of the files in that directory matter here. `mitmproxy-ca.pem` bundles the certificate and the private key

that sign each leaf, and `mitmproxy-ca-cert.pem` is the certificate-only file you hand to clients. The `.cer` and `.p12` names in the same directory are that certificate wrapped for Android and Windows, inventoried in chapter 2 [9].

The client must trust this CA, because the leaf certificate it is handed on each domain is signed by the CA and the client verifies that signature. The CA system exists to stop exactly what mitmproxy does. “If this signature doesn’t match or is from a non-trusted party, a secure client will simply drop the connection and refuse to proceed” [59]. Since no browser trusts the mitmproxy CA out of the box, you either click through a certificate warning on every domain or install the CA once [9]. Without the CA, curl aborts with exit code 60 and mitmdump logs that the client does not trust its certificate, the failure chapter 5 shows from both sides. With the CA trusted, the same command succeeds and the request shows up as a normal HTTP/2 flow [11]. How you earn that trust (the operating system’s certificate store, `--cacert` for curl, or the `mitm.it` onboarding app that serves the right file for your platform [40]) is the one-time trust step, so the next chapter walks through each route.

The proxy maintains a second, separate trust store for the other leg. mitmproxy verifies the upstream server’s certificate too, against its own CA bundle, with `ssl_verify_upstream_trusted_ca` naming a PEM file and `ssl_verify_upstream_trusted_confdir` a `c_rehash` directory of anchors [42]. `--ssl-insecure` (or `-k`) skips that validation, and the option’s own help text warns that with it enabled “mitmproxy itself will be vulnerable to TLS interception” [42]. In the mitmdump log of that failed request, the line `server connect labshop.test:8443` still appears, because it records only the upstream connection being established. What shows the upstream handshake passed is the GET `https://labshop.test:8443/api/products` HTTP/2.0 flow that returns 200 once the client trusts the CA [11]. In this book’s lab, labshop’s certificate is signed by a lab CA that was placed in the upstream trust store. That is lab setup, and you should never read it as mitmproxy accepting self-signed server certificates on its own.

You can swap in your own CA with `--set confdir=DIRECTORY`, which makes mitmproxy look for `mitmproxy-ca.pem` there, generating it if missing. For the swap to work, your own `mitmproxy-ca.pem` must bundle the private key and the certificate, and the certificate must carry the `CA:TRUE` basic constraint and the `Certificate Sign` key usage [9].

What interception cannot see

Interception is conditional on three things: the traffic must reach the proxy, the proxy must choose to process it, and the client must accept the certificate the proxy presents. Each condition has traffic that defeats it in the field.

Traffic that never reaches the proxy stays invisible no matter what you do. Apps that ignore the device's proxy configuration simply do not send their traffic to it [62]. That is a deployment problem, and transparent mode is the standard answer, but it changes which leg supplies the hostname, as above.

Traffic you exempt keeps flowing, unmodified. `ignore_hosts` matches a regex against the `host:port` of each connection, and “matching hosts are excluded from interception, and passed on unmodified” [63]. In regular mode only SSL traffic is ignored [21]. The related `tcp_hosts` and `udp_hosts` options opt raw TCP and UDP connections into interception explicitly, and even then the TCP proxy will not detect a plaintext protocol that upgrades to TLS via STARTTLS. DNS-over-TLS, DNS-over-HTTPS, and DNS-over-QUIC are not implemented at all [64].

Then there is the traffic that refuses your certificate. Certificate pinning is the direct consequence of this chapter's mechanism. The application hard-codes which certificate or CA it will accept, so “mitmproxy's certificates will not be accepted by these applications without modifying them” [9]. If the pinned contents do not matter, the docs recommend `ignore_hosts` for those domains. If they do, “you need to patch the application manually,” and the certificates page lists `apk-mitm`, `objection`, `ssl-kill-switch2`, and `android-unpinner` for Android and iOS [9]. Expect both outcomes. For apps that use the well-known APIs, a pinning bypass “can be bypassed within seconds,” but apps with a custom framework or library need manual patching [65]. Either way, the app refuses your intercepting certificate, so installing the CA changes nothing and the handshake still fails, typically with an alert of the `unknown-ca` family [9, 65].

Two newer mechanisms widen that gap. Encrypted Client Hello encrypts the SNI itself, which would leave mitmproxy without a hostname to forge. mitmproxy counters by stripping ECH keys from DNS HTTPS records, so the client has no key to encrypt its ClientHello with and the proxy still learns the domain [18]. The `strip_ech` option controls this and defaults to `True` [21]. HTTP/3 rides on QUIC over UDP, and the regular proxy listener is TCP, so QUIC

traffic never arrives at it. mitmproxy handles HTTP/3 in reverse, transparent, local, and WireGuard modes [64, 18]. Chrome, for example, does not trust user-added CAs for QUIC and falls back to HTTP/2 when the proxy cannot present a publicly trusted certificate [18]. Mobile interception tooling exploits the same fallback. The Frida scripts bundled with HTTP Toolkit block all UDP to port 443 so apps fall back to HTTP/2 or HTTP/1 [66].

Finally, mutual TLS restricts who may speak at all. If the upstream server demands a client certificate, mitmproxy presents one from `client_certs`, a PEM file or a directory selected by hostname [9]. The client leg works the same way by configuration. `--set request_client_cert=true` makes mitmproxy send the `CertificateRequest` TLS message to the client, establishing mTLS without validating the client's identity [9].

The interception certificate is a per-host leaf, dynamically generated and signed by your CA. It succeeds exactly when the client trusts that CA, and pinning is the mechanism an app uses to withdraw that trust for a specific domain. Where pinning is absent, you see the same decrypted JSON that the server's operator sees. Where pinning is present, neither a CA install nor the trust step in chapter 5 helps; defeating it is the reversing work of the mobile chapters (15–16). Before moving on, send one request through the proxy without `--cacert` and one with it, and read the mitmdump log lines that accompany each.

Chapter 5: Trusting the mitmproxy CA

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Installing the CA on Linux

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Debian and Ubuntu

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

RHEL, Fedora, CentOS, and Oracle Linux

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Arch and containers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Installing the CA on macOS and Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

macOS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Browsers and runtimes with their own trust store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Firefox

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chrome and Edge

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Java

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Python

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Node.js

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Troubleshooting CA trust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 6: Filter Expressions and Flow Inspection

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Filter expression syntax

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Common filter patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Inspecting flows in the TUI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Inspecting flows in mitmweb

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Modification and Replay

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 7: Modifying Requests and Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Interactive modification in the TUI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

`modify_headers`, `modify_body`, `map_local` and `map_remote`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Modification with addons

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Common modification recipes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 8: Replaying Requests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Client-side replay

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Server-side replay

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Replay in mitmweb

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Replay caveats and pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Addon Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 9: Addon Architecture and Events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Anatomy of an addon

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Event hooks and their order

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Loading and running addons

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

The abbreviated scripting syntax

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 10: Options, Commands, and Hooks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Declaring addon options

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Defining custom commands

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Custom contentviews

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Exposing addons in the UI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 11: Working with Flows and Messages

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

The flow object model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

HTTP request and response objects

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Headers, cookies and bodies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Reading flow files from Python

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Deployment Modes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 12: Choosing a Mode: Regular, Transparent and Reverse

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

The mode specification syntax

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Choosing a mode for your task

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Regular proxy mode

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Transparent mode on Linux

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Transparent mode on macOS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Reverse proxy mode

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 13: Upstream, SOCKS, and WireGuard Modes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Upstream proxy mode

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

SOCKS proxy mode

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

WireGuard mode

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Combining modes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 14: Local Capture Mode

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Local redirect on Linux

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Local redirect on macOS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Local redirect on Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Capturing Docker containers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Mobile and Advanced Protocols

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 15: Intercepting Android and iOS Apps

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Android emulator setup

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Installing the CA as a system certificate

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Android 14 and newer CA installation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Physical Android devices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

WireGuard for Android apps

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Emulator networking and localhost

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

iOS proxy and CA setup

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Full trust for TLS on iOS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 16: Bypassing Certificate Pinning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

What certificate pinning is

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Frida and objection basics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Universal unpinning scripts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Pinning on iOS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Pinning bypass caveats

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 17: HTTP/2, HTTP/3, and WebSockets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

HTTP/2 interception

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

HTTP/3 and QUIC

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

WebSocket message handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Protocol-specific addon events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Automation and Reference

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 18: Exporting Flows: HAR, curl and OpenAPI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Saving and loading flow files

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Streaming large transfers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Exporting HAR

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Copying requests as curl and raw

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

From flows to an OpenAPI spec

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 19: Automating API Security Testing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

A harness you drive with addons

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Building a fuzzing addon

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Automated replay and regression

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Auth token rotation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Reporting what the tests found

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Chapter 20: Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

What you can now do

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Where to go next

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Keeping current with mitmproxy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Appendix A: Command-line options reference

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

How options reach the tools

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Invocation and file I/O

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Proxy modes and listeners

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

TLS and certificates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Modification and replay

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Protocol and bodies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Frontend-specific flags

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Environment variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Appendix B: Filter expression reference

Operators

Every operator below takes a Python-style regex unless it is a flag or a combinator; ~c is the exception and takes an integer. The docs' generated filter table lists this set [83].

Operator	Argument	Matches
~a	none	asset in a response: CSS, JavaScript, images, fonts
~all	none	all flows
~b	regex	body, request or response
~bq	regex	request body
~bs	regex	response body
~c	integer	HTTP response status code, exact value
~comment	regex	flow comment
~d	regex	domain (hostname)
~dns	none	DNS flows
~dst	regex	destination address
~e	none	flows with a connection or protocol error
~h	regex	any header, request or response
~hq	regex	request header
~hs	regex	response header
~http	none	HTTP flows
~m	regex	HTTP method
~marked	none	flows marked by the user
~marker	regex	marked flows whose marker matches
~meta	regex	flow metadata
~q	none	request with no response
~replay	none	flows replayed in either direction

Operator	Argument	Matches
<code>~replayq</code>	none	replayed client request
<code>~replays</code>	none	replayed server response
<code>~s</code>	none	flows with a response
<code>~src</code>	regex	source address
<code>~t</code>	regex	content-type header
<code>~tcp</code>	none	TCP flows
<code>~tq</code>	regex	request content-type header
<code>~ts</code>	regex	response content-type header
<code>~u</code>	regex	request URL
<code>~udp</code>	none	UDP flows
<code>~websocket</code>	none	WebSocket flows, including the HTTP upgrade handshake [252]
<code>!</code>	expression	negates what follows
<code>&</code>	expression	and
<code> </code>	expression	or
<code>(...)</code>	expression	grouping

The four flag operators `~q`, `~s`, `~e`, `~a` and the flow-type tests answer yes or no and take no argument, so they appear bare. The request/response variants (`~bq/~bs`, `~hq/~hs`, `~tq/~ts`, `~replayq/~replays`) scope a search to one side of the exchange [83].

Combining expressions

- A token with no operator is a regex matched against the request URL [275]. In `basic-http.mitm`, `catalog` selects only the `/catalog` flow [276].
- Adjacent terms AND together; the default binary operator is `&` [275]. `~m POST ~c 302` means `~m POST & ~c 302` and selects the 302 login flow [277].
- `&` binds tighter than `|`. On `basic-http.mitm`, `~m POST | ~c 200 & ~u catalog` returns the same two POST flows as `~m POST | (~c 200 & ~u catalog)` [278, 279], and the other parenthesization (`~m POST | ~c 200`) `& ~u catalog` matches nothing [280].

```

1 $ mitmdump -nr flows/basic-http.mitm "~m POST | ~c 200 & ~u catalog"
2 [23:43:22.539] Only processing flows that match "~m POST | ~c 200 & ~u catalog"
3 127.0.0.1:59408: POST http://shop.localhost:8080/login
4         << 302 Found 0b
5 127.0.0.1:59444: POST http://shop.localhost:8080/login
6         << 401 Unauthorized 33b

```

- `!` negates a term or a parenthesized group; group with `(...)` whenever you mix `!` or `|` with `&` [275, 83]. The docs' example `!(~q & ~t "text/html")` matches anything but requests with a text/html content type [275].
- The flag operators depend on the capture. Every flow in basic-http.mitm carries a response and none has an error, so `~q` and `~e` match nothing there [23, 281]; keep those two for live interception or files that hold requests left unanswered.

Regex arguments

- Regexes are Python-style, unanchored, and case-insensitive by default; set `MITMPROXY_CASE_SENSITIVE_FILTERS=1` to make them case-sensitive [275].
- An argument containing reserved characters, meaning parentheses, whitespace, `~`, `'`, or `"`, must be quoted, or the expression parses differently or is rejected [275]. The docs' example is `~u "get(Info|Routers)"` [275].
- Header operators match against a serialized line of the form `name: value`, not the bare value [275]. `~hs set-cookie` therefore matches any response carrying a Set-Cookie header; on basic-http.mitm that is exactly the 302 login flow [282].
- `~c` does not take a regex. `~c '401|404'` fails the whole expression: `mitmdump: Invalid filter expression: "~c '401|404'"` [283]. Express a range by ORing exact codes, as in `~c 401 | ~c 403`.
- In a shell, wrap the expression in quotes so `|`, `!`, and spaces survive as one argument; every example in this book does so [108].

Flow selectors

Interactive tools accept a second vocabulary, the selectors, which operate on the current view. The docs scope them to commands and key bindings in interactive contexts [275]:

Selector	Selects
@all	all flows
@focus	the currently focused flow
@shown	all flows currently shown
@hidden	all flows hidden by the current view filter
@marked	all marked flows
@unmarked	all unmarked flows

Commands with flow-sequence arguments expand them, so `:replay.client @all` enqueues every viewed flow, and `@focus` names the flow under the cursor in every command that takes flow arguments [275]. Selectors are not filter operators; `mitmdump`'s positional argument is a filter, not a selector spec.

Where filters apply

Surface	Mechanism	Effect
<code>mitmdump</code> positional argument	<code>filter_args</code>	Sets both the <code>view_filter</code> and <code>save_stream_filter</code> options [284]
<code>TUI f</code> prompt	<code>view_filter</code>	Limits the list to matching flows; the option is shared by <code>mitmproxy</code> and <code>mitmweb</code> [21, 252]

Surface	Mechanism	Effect
mitmweb search box	view_filter	Same effect from the browser UI [84]
TUI i; -i/--intercept	intercept	Matched flows pause until resumed or edited [89, 285]
--stickycookie, --stickyauth	stickycookie, stickyauth	Filters matched against requests; matching requests get cookies or auth attached [284]
--modify-headers, --modify-body, --map-remote, --map-local, --set block_list=... -w writes	optional per-rule /flow-filter/ prefix save_stream_ filter	The rule touches only matching flows; without the prefix it applies to all HTTP flows [68, 21] Filters the flows written to the flow file; also selects the flows hardump saves [68]
-r reads	readfile_filter	Reads only matching flows instead of the whole file [68]
mitmdump output	dumper_filter	Limits which flows mitmdump dumps [68]

The rule-bearing options deserve one caution: their flow-filter prefix is optional, so a pattern like `| labshop.test:8443/catalog | labshop.test:8443/api/products` (no leading filter) rewrites those URLs for every flow, while a pattern that names a filter first, as in `|~m POST|/login|signin`, applies only to the matching subset [68].

Appendix C: Addon event reference

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Addon event reference

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Lifecycle and configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Connection events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

HTTP events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

TLS events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

WebSocket events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

TCP, UDP, DNS, QUIC, and SOCKS5 events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Advanced events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Appendix D: Flow export formats

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

At a glance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

The .mitm flow file

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

HAR (HTTP Archive)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Export commands: curl, HTTPie, raw

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

OpenAPI 3.0 through mitmproxy2swagger

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.

References

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/practical-mitmproxy>.