

MASTERING THE METASPLOIT FRAMEWORK

PRACTICAL METASPLOIT

From first exploit to advanced
post-exploitation

RECON · EXPLOIT · POST-EXPLOIT

CUSTOM MODULES · AUTOMATION · EVASION

GROUND TRUTH BOOKS

Practical Metasploit

Mastering the Metasploit Framework from First Exploit
to Advanced Post-Exploitation

Ground Truth Books

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

This version was published on 2026-10-09



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

Foundations	1
What Is Metasploit	2
Why Metasploit for penetration testers	3
How Metasploit works	4
The seven components	5
Glossary	6
Modules and Framework Architecture	9
Understanding the Metasploit module directory structure	9
Auxiliary modules	10
Payloads	11
Filesystem and libraries	12
Encoders	13
NOPs	14
ATT&CK metadata	14
Building Your Attack Lab	16
Installing the Metasploit Framework	16
Installing on Kali Linux	17
Launch msfconsole in Kali	19
DNS configuration	22
Keeping the lab current	23
The msfconsole Workflow	25
Metasploit commands	25
Getting help for msfconsole commands	25
Searching for exploits with keywords	25
Datastore	25
Finding modules	25
Understanding module options	25
Use, back, and exit commands	26

The Metasploit Database	27
Initialize the Metasploit PostgreSQL database	27
Database support	27
Database workspaces	27
Example workflow	27
Keep your Metasploit data safe	27
Using msfdb	27
 Exploitation	 29
Service Discovery and Scanning	30
Port scanning	30
Scanner HTTP auxiliary modules	30
Auxiliary module examples	30
Vulnerability scanning with Metasploit	30
A TCP port scanner	30
The Metasploitable scanning lab	30
 Staged versus Stageless Payloads	 32
How payloads work	32
What are staged payloads?	32
What are stageless payloads?	32
Staged versus stageless handlers	32
Connecting stageless payloads to staged handlers	32
Stageless mode	32
Working with payloads	33
 Generating Payloads with msfvenom	 34
Introducing msfvenom	34
How to use msfvenom	34
Output formats	34
Handlers	34
Advanced msfvenom payload generation	34
A worked example	34
 Exploitation and Session Management	 36
Working with exploits	36
Working with sessions	36
Checking target availability	36

CONTENTS

Manage Meterpreter and shell sessions	36
Setting up a payload	36
Basic workflow	36
Meterpreter and Post-Exploitation	38
Meterpreter Fundamentals	39
What is Metasploit Meterpreter?	39
How Meterpreter works	39
Meterpreter flavors	39
Core commands	39
File system commands	39
Help	39
Meterpreter transport control	40
The lab	40
Migration and Privilege Escalation	41
Meterpreter getsystem	41
Migrate	41
priv_migrate	41
System commands	41
Other commands	41
The escalation lab	41
Pivoting and Routing	43
How pivoting works	43
Route-based pivoting	43
Port forwarding	43
Credentials, Persistence, and Lateral Movement	44
About post-exploitation	44
hashdump	44
Post-exploitation modules by category	44
Lateral movement and persistence	44
Extending the Framework	46
Resource Scripts and Automation	47
Resource scripts	47

CONTENTS

Six ways to automate Metasploit	47
Automating the Metasploit console	47
Basic workflow	47
Writing Custom Auxiliary Modules	48
Writing an auxiliary module	48
How to add a third-party module	48
Running private modules	48
Navigating the codebase	48
Using Rex::Proto::Http::Client	48
The Exploit::Remote::Tcp mixin	48
Module documentation	49
Writing Your First Exploit	50
Building your first Metasploit exploit	50
Setting up a Metasploit development environment	50
Writing module documentation	50
Railgun and the Windows API	52
Using Railgun for Windows post-exploitation	52
Defining a DLL and its functions	52
MSF Meterpreter and Railgun	52
Evasion and Modern Context	53
Encoders and Evasion Limits	54
Encoders	54
AV bypass with Metasploit templates	54
A defender's checklist	54
Detecting Metasploit attacks	54
Hardening and defensive takeaways	54
Keeping Modules Current	55
Metasploit 6 in development	55
Metasploit Framework 6.0 release notes	55
Metasploit Framework 6.4	55
The 6.5 MCP server	55
When the APT repo falls behind	55
Reading the release notes	55

Future trends and emerging considerations	56
Appendix A: Command and module cheat sheet	57
Console startup	57
Navigation and workflow	57
Module layout	57
Payloads and msfvenom	57
Database commands	58
Sessions, jobs, handlers	58
Meterpreter commands	58
Pivoting	58
Post module pick list	58
Ranks and check codes	58
Resource scripts	58
Appendix B: Framework release history	60
Major releases	60
Version numbering and release cadence	60
Module inventory over time	60
Compatibility breaks	60
Editions and licensing	60
Appendix C: Auxiliary and post module index	61
Auxiliary modules	61
Post modules	61
Glossary	62
References	63

Foundations

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

What Is Metasploit

Metasploit is an open-source penetration testing framework written in Ruby, one that provides the infrastructure to develop and run exploit code against remote targets [1], [2]. The project's README describes it as an open-source tool released under a BSD-style license [3]. H. D. Moore built it because hand-validating public exploit code was eating the time he had for testing [4].

He released the first version in October 2003: a Perl framework with 11 exploits [4]. Exploits circulated on IRC channels and mailing lists in their own formats, and each had to be re-verified before it was safe to point at a target. A framework that normalized exploit code into a common structure, with a shared payload layer, fixed that once instead of on every engagement [4]. The project moved to Ruby in 2007, a rewrite that took about 18 months and added 150,000-plus lines of code, and Rapid7 acquired it in 2009 [4], [5], [6].

That acquisition changed less than you might expect. Rapid7 did not fork the framework into a closed product. The BSD license stayed in force on the open-source line, and Moore kept leading the project, now with a team that worked on it full time [4]. The company announced it would “not only maintain, but accelerate the open source framework Metasploit with dedicated resources and contributions” [7]. Metasploit Pro wraps the open core rather than replacing it, and a new exploit module lands in `modules/exploits/` once for both [4]. Pro layers a web UI and guided MetaModules on that same base, with social engineering campaigns and dynamic payloads for antivirus evasion on top [8], [4].

Version 6, announced in August 2020, reset the payload layer around a “secure by default” theme. It brought end-to-end encryption of Meterpreter communications across all five implementations, plus SMBv3 client support and a polymorphic routine that randomizes Windows shellcode on each invocation [9]. The encryption uses AES with keys negotiated over RSA, and Meterpreter commands were re-encoded as integers so that names like `core_migrate` no longer appear as strings in the binaries [10]. The release also dropped Meterpreter support for Windows versions older than XP SP2 and made SMB3 encryption the negotiated default [10]. Later 6.x releases kept the same line. Version 6.4 added interactive database and SMB session types

[11] and Kerberos diamond and sapphire ticket forging [12]. The 6.4 line later added the PoolParty process-injection technique [11]. Version 6.5 added MITRE ATT&CK tagging to module search, a response to a module library that has passed roughly 5,000 entries on a 6.5 build [4].

The 2007 Ruby rewrite explains the framework's internals. Ruby's object model let the project express "an exploit is a subclass of a module, which has a datastore, which has options" as real inheritance rather than convention, and the framework's internals all descend from that decision [4]. When you load a module in this book, you are loading one instance of that hierarchy.



Metasploit is a dual-use tool: the same modules that secure a network can break it. Only run it against systems you own, or against systems a signed scope document covers. Beyond that line, your work and an unauthorized intrusion look identical to a court [13]. Practice only inside a lab you control, and when an engagement has a scope document, keep a copy to hand [14]. Every later chapter assumes you work within that authorization.

Why Metasploit for penetration testers

The case for the framework is easiest to see against what came before it. Before Metasploit, testers probed with a set of tools that may or may not have supported the platform they were testing. They wrote their own exploit code by hand and carried it onto the network manually, and remote testing was largely out of reach [15]. Metasploit turned that into the repeatable loop you will use throughout this book [16].

The modular design is why the loop holds. Because you can pair any exploit with any payload, a tester who understands one module's options understands the shape of the next, and the delivery code is written once in the framework rather than on every engagement [5]. The module library is also a shared asset that grows because contributors add to it. In 2024 the framework added 165 new modules from 62 developers, 39 of them first-time contributors [11]. Vendors frequently ship a Metasploit module alongside a vulnerability advisory, making the framework a de facto clearing house for exploit development [5]. The acquisition accelerated the flow: Rapid7 reported the framework's code base grew by 156% in the two years after it took over [17].

The second reason is validation. An automated scanner tells you a vulnerability might exist. Metasploit tells you whether it can actually be exploited, by attempting a real attack in a controlled setting [11]. The gap between those two answers carries most of a report's value. A critical CVE can be unexploitable because of how the environment is configured, while a moderate-severity issue can hand over a direct path to compromise [11].

A third reason is that the tool is expected. Metasploit is core curriculum for the OSCP and the CEH as well as Rapid7's own Metasploit Pro Certified Specialist, so fluency in its functions as a hiring signal as well as a skill [11]. The same visibility has a defensive side: MITRE ATT&CK records adversaries obtaining Metasploit under technique T1588.002 and names groups such as CopyKittens and Magic Hound, which is why defenders study the framework too [11].

How Metasploit works

The framework runs an engagement through a small loop of commands. You search for a module, load it with `use`, review what it needs with `show options`, set the target and payload values, run `check` if the module supports it, and launch with `exploit` or `run` [16], [13]. The values you set most often are `RHOSTS` and `RPORT` for the service, and `LHOST` and `LPORT` for the listener a reverse payload calls back to [18], [13].

Use `check` before you `exploit` whenever the module offers it. A successful check does not guarantee the exploit will land, but it tells you the module believes the target is vulnerable without running the full attack path, which reduces disruption on a live assessment [13], [19].

Four kinds of component do the work of an engagement, and the next section gives each its own entry. Only encoders need a caveat here. They encode the payload so it arrives intact, removing characters that would break the exploit [1]. That can fool some signature-based antivirus, but it is not a reliable bypass for modern endpoint defenses [20], [21].

Payload names encode the whole plan. Take `windows/x64/meterpreter/reverse_tcp`: `windows` is the platform, `x64` the architecture, `meterpreter` the stage, and `reverse_tcp` the stager that delivers it [22]. The punctuation tells you whether a payload is staged. `windows/shell_bind_tcp` is a single, self-contained payload, while `windows/shell/bind_tcp` is a stager (`bind_tcp`)

plus a stage (`shell`) [23], [24]. Staging exists for tight exploits. The stager is deliberately small and dumb: it opens a connection and reads a four-byte stage size, then allocates memory and hands execution to the stage, which can then be arbitrarily large [25], [22]. Meterpreter is the advanced stage most operators reach for, an in-memory payload whose interactive shell handles everything from file transfer to pivoting [26], [11].

When an exploit succeeds, Metasploit opens a session, the communication channel between your console and the target [27]. You list sessions and connect to one with `sessions -i <id>`, then background it and work on another target [16], [27].

Beneath the console sit three libraries. Rex handles the low-level socket and protocol work, from basic data transformations to full protocol implementations such as SMB and HTTP. MSF Core provides the common API that defines the framework. MSF Base wraps that in a friendlier user-oriented interface [28], [6].

The seven components

The framework is easier to hold in your head as seven components. Six are things you use. The seventh is the configuration layer that ties them together [29].

MSFconsole is the main command-line interface [29]. It is the most commonly used way to work with the framework, and it exposes every command an engagement needs [1].

Exploit modules target a specific, known vulnerability. The library spans buffer overflows and web application exploits, and each module carries a payload you can run against the target [29], [1].

Auxiliary modules do everything that is not exploitation, and they never execute a payload [29], [1]. On an assessment you use them for reconnaissance and service enumeration before you reach for an exploit [11].

Post-exploitation modules run after you have access. They enumerate applications and services and deepen access to the target and its network, dumping password hashes along the way [29], [1].

Payload modules supply the shellcode that runs after a successful exploit. A payload can open a command shell or the advanced Meterpreter payload,

which lets you load new capabilities as DLLs on the fly [29], [1].

NOP generators produce random bytes used to pad buffers, with the goal of bypassing NOP-sled signatures in intrusion detection and prevention systems [29], [1].

The datastore is the central configuration that governs how the other components behave. It is a table of named values that every component reads and writes, carrying options between modules [1], [29]. Two levels exist: the global datastore, set with `setg` and visible to all modules, and the module datastore, set with `set` and visible only to the current module [30], [1].

Those seven components sit on a base of seven module types, the categories the framework itself uses. Its module tree lists auxiliary, encoders, evasion, exploits, NOPs, payloads, and post [31].

The reason modules of any type look familiar to write is the mixin system. A module is a Ruby class that inherits from a base such as `Msf::Exploit::Remote` for remote exploits or `Msf::Post` for post modules, and its `initialize` method declares the module's name and description, then its options [32], [33]. A mixin adds behavior and with it options. Include `Msf::Auxiliary::Scanner` and your module gains an `RHOSTS` option for multiple hosts and a `THREADS` option, with scan progress tracking and a `run_host(ip)` entry point instead of `run` [34]. Include `Msf::Exploit::Remote::Tcp` and you gain a `connect` method plus SSL and proxy options [35].

Those mixins reach down into Rex, the Ruby Exploitation library that lives in `lib/rex` [2], [36]. TCP connections, for instance, are `Rex::Socket::Tcp` objects that extend Ruby's own `Socket` class. You use the Rex socket rather than the native one so the framework can manage the connection, which is exactly what makes features like pivoting possible [35].

Glossary

The working terms this book uses throughout. Definitions follow Rapid7's framework documentation and glossary [26], [37].

Term	Meaning
Auxiliary module	A module that does not execute a payload; used for scanners and fuzzers [26], [1]
Bind shell	A shell that opens a listener on the exploited system and waits for the attacking machine to connect [26]
Exploit	A program that takes advantage of a specific vulnerability and gives access to the target, typically carrying a payload [26]
Exploit module	A module that runs a sequence of commands to exploit a vulnerability; in short, it creates a session [37]
Listener	A component that waits for and manages an incoming connection, from either the target or the attacking machine [26]
Meterpreter	An advanced multi-function payload providing an interactive shell; it runs in memory and handles everything from file transfer to pivoting [26]
Module	A standalone piece of code that extends the framework; its type determines its purpose [26]
Payload	The shellcode that runs after an exploit succeeds; it can open a Meterpreter or command shell [26]
Post-exploitation module	A module that gathers more information or gains further access after a target is exploited [26]
Reverse shell	A shell that connects back to the attacking machine as a command prompt [26]
Session	The communication channel Metasploit establishes with a target after a successful exploit [27]
Shellcode	The set of instructions an exploit uses as its payload [26]

Term	Meaning
Vulnerability	A security hole in software, hardware, or an operating system that provides a potential angle to attack [26]
Workspace	The framework's name for a project: a container for the targets, hosts, and data of one engagement [26]

The next chapter takes this vocabulary apart. It walks the module hierarchy and the mixin system you met here, and msfconsole itself opens in Chapter 4.

Modules and Framework Architecture

Understanding the Metasploit module directory structure

Each command you ran in Chapter 1, from `search` to `exploit`, operated on one of the module types you met there. This section shows where those modules live on disk and what their file paths encode.

From inside `msfconsole`, the `edit` command answers the first question. It opens the current module's source in your editor, and the path it prints tells you exactly which file the framework loaded [38]:

```
1  [*] Launching /usr/bin/vim /usr/share/metasploit-framework/modules/exploits/windows/smb/ms10_061_spoolss.rb
```

That path sits in the first of two locations the framework scans. The install tree at `/usr/share/metasploit-framework/modules/` holds the shipped modules, and `~/.msf4/modules/` holds yours [39]. The top level carries one directory per module type: `auxiliary`, `encoders`, `exploits`, `nops`, `payloads`, `post`, and `evasion` [40, 39].

Beneath the type, the second level is often described as `protocol`, but the tree varies by type [30]. Exploits break down by platform first, so you find directories such as `aix` and `windows` [39]. Auxiliaries break down by job, with `scanner` and `gather` among the groups [41]. Encoders and NOP generators separate by target architecture, one directory per family [39]. Payloads separate by shape, which the payloads section takes apart [39].

Follow the convention down and a module's full name is its path. `auxiliary/scanner/http/options` reads “auxiliary type, scanner job, HTTP protocol, options module”, and the same pattern gives `exploit/windows/smb/ms08_067_netapi`. You type that path to use, and `info` echoes it in the `Module` field [42].

Msfconsole does not rescan these directories as you type. It reads a prebuilt index, `db/modules_metadata_base.json`, which updates whenever a module lands or changes and backs module-name autocompletion [40].

Drop a custom module into `~/.msf4/modules/` and three mistakes will keep it from loading. The type directories are plural for exploits and payloads, but singular for auxiliary and post. The file needs at least one category level under the type, such as `auxiliary/scanner/`. And the filename must match the class name declared inside the file [33, 43]. Permissions matter too. A file your user cannot read never loads [33].

After you place a file, run `reload_all` to rescan all module paths [33, 43]. Alternatively, point msfconsole at an extra tree at startup with `-m`, or load one live from inside msfconsole with `loadpath`. Run it bare to see the usage message, then give it a path [39]:

```
1 msfconsole -m ~/secret-modules/

1 msf > loadpath
2 Usage: loadpath </path/to/modules>
3 Loads modules from the given directory which should contain subdirectories for
4 module types, e.g. /path/to/modules/exploits
5 msf > loadpath ~/secret-modules/
6 Loaded 1 modules:
7     1 auxiliary
```

Auxiliary modules

Auxiliary modules do most of the reconnaissance in the framework, thanks to their scanner mixin. Metasploit draws the line by what a module ships: exploit modules carry a payload, and among the module types that do not, those that are not post-exploitation modules are auxiliary [41]. The same boundary reads from the other side. If a module executes a payload, it is an exploit. If it does not, it stays auxiliary even when it takes advantage of a vulnerability [34]. Port scanners and sniffers land here [30].

Because they run no payloads, auxiliaries carry most of the reconnaissance work. A current 6.5 checkout holds 1,325 auxiliary modules [4]. They prove useful both before and after exploitation, since gathering and enumeration

tasks appear on both sides of that line [28]. Many of them are scanners that test a range of hosts. The `Msf::Auxiliary::Scanner` mixin does the heavy lifting there. Including it registers a `RHOSTS` option for the target range and a `THREADS` option for concurrency, and it changes the entry point from `run` to `run_host(ip)`, which the framework invokes once per host [34].

Here is the option surface a real scanner exposes [44]:

```

1  msf auxiliary(options) > show options
2  Module options (auxiliary/scanner/http/options):
3      Name      Current Setting  Required  Description
4      ----      -
5      Proxies                no        A proxy chain of format
        ↳ type:host:port[,type:host:port][...]
6      RHOSTS                yes       The target address range or CIDR identifier
7      RPORT      80         yes       The target port (TCP)
8      SSL        false      no        Negotiate SSL/TLS for outgoing connections
9      THREADS    1         yes       The number of concurrent threads
10     VHOST                no        HTTP server virtual host

```

The `RHOSTS` line takes a single address or a CIDR block. A range such as `192.168.1.200-210` covers many hosts at once, and `msfconsole` reports progress as `run_host` fires per host [44].

The boundary blurs in one direction worth knowing. An auxiliary module able to run operating-system commands becomes an exploit the moment you deliver it a `cmd*` payload or wrap it with a command stager [34]. The same mixins serve both families, which is why the module system you are mapping here underlies the rest of the framework.

Payloads

Payloads come in three structural variants. A single is self-contained. It carries all the code it needs in one blob, and it can add a user or launch `notepad.exe` [24]. A stager does one narrow job. It sets up the connection channel between framework and target, then downloads the rest of the payload, the stage [24]. Because the stager is small, staged payloads keep the initial footprint low while the stage can carry a full Meterpreter or command shell [24].

The payload name marks the variant in its punctuation, as Chapter 1 showed; Chapter 7 takes staged versus stageless naming apart.

The directory tree mirrors the taxonomy. `modules/payloads` holds the three variants as sibling directories [39].

A payload is a module like any other, so you can load one directly and ask for output in a given format. The Modules guide walks the loop on a Linux reverse-shell payload: `use` points at the module, then `generate -f c` prints the shellcode as C source [45]. `generate -f elf -o linux_shell` writes an executable you can drop on a Linux test box [45].

On an exploit module, the payload is a datastore option named `PAYLOAD`, and the module's metadata lists which payloads fit so you avoid blind trial-and-error [13, 16]. Later chapters return to choosing a transport. That choice rides on the same module machinery as every other type.

Filesystem and libraries

The install root `/usr/share/metasploit-framework` organizes the whole program by function. `data` holds wordlists and binaries, `documentation` holds module write-ups, `modules` holds the module tree, `plugins` holds console plugins, `scripts` holds Meterpreter and console scripts, and `tools` holds command-line utilities [40, 46].

The `lib` directory is where the machinery lives, in three layers [46]. `Rex`, the Ruby Exploitation library, handles the low-level work. It manages sockets and protocols, with implementations for SSL and SMB [46]. `MSF Core` supplies the basic API and defines the framework. `MSF Base` supplies the friendlier API the console interfaces use [46]. A separate `lib/metasploit` namespace rounds out the layout [2].

`Rex` is the most fundamental piece of the architecture, yet module authors rarely touch it directly. Modules depend on the framework core and its mixins, which wrap `Rex` for them [47]. The mixins under `lib/msf/core/exploit` bundle protocol behavior, with files like `tcp.rb` and `smb.rb` ready to include [48].

A module declares what it needs by inheritance plus inclusion. The class inherits a base such as `Msf::Exploit::Remote` or `Msf::Auxiliary`, and each include pulls in a mixin's methods and datastore options. For TCP work, include `Msf::Exploit::Remote::Tcp`. The mixin provides the `connect` method and registers the SSL and proxy options [35]. For HTTP, include `Msf::Exploit::Remote::HttpClient` [47]. Include `Msf::Auxiliary::Scanner` instead,

and you get the RHOSTS and THREADS options with the run_host contract from the auxiliary section [34].

The contributor guide fixes the include order: protocol mixins first, utility mixins second, reporting mixins third, then prepend Msf::Exploit::Remote::AutoCheck last [32]. Condensed to its bones, the canonical exploit template looks like this [32]:

```
1 class MetasploitModule < Msf::Exploit::Remote
2   include Msf::Exploit::Remote::HttpClient
3   include Msf::Exploit::FileDropper
4   prepend Msf::Exploit::Remote::AutoCheck
5
6   def initialize(info = {})
7     super(update_info(info, 'Name' => 'Vendor Product Vulnerability', 'License'
8       => MSF_LICENSE))
9   end
end
```

Mixins are why the module files stay small. Reusable behavior lives once in the shared libraries. The guidance to contributors is explicit. Keep shared constants and helpers in one authoritative place rather than copying them into each module [32]. Something that affects multiple modules belongs in lib/, not in the module file [40].

Encoders

An encoder transforms payload bytes without changing what they do, and its honest limits are worth stating. Its primary purpose is to strip out characters the target cannot accept, or to reshape the payload so the delivery channel takes it [49]. If a vulnerable service truncates on null bytes, you encode around them rather than redesigning the payload.

msfvenom is where you meet encoders in practice. The -b flag names the bad characters, and by default msfvenom picks a compatible encoder automatically when you use it [50]:

```
1 ./msfvenom -p windows/meterpreter/bind_tcp -b '\x00' -f raw
```

The -e flag forces a specific encoder, such as shikata_ga_nai [50].

shikata_ga_nai is the famous polymorphic XOR additive feedback encoder, and the family extends across architectures. The alpha_mixed and base64 encoders appear in `show encoders` output [38].

Encoders do not aim to bypass antivirus. They encode the payload [20]. They once served as a basic anti-signature trick, but modern security solutions handle encoded payloads far more effectively [49].

Encoder modules live in `modules/encoders`, sorted by target architecture with a directory per family [39].

NOPs

A NOP generator emits an instruction that performs no operation, so exploits can pad buffers and keep payload sizes consistent. In the Intel architecture that instruction is 0x90, and the processor spends one idle cycle on it [51]. The older function of that padding was practical cover. A NOP generator produces random bytes you can use to evade NOP-sled signatures in IDS and IPS, and you use them to pad buffers [30].

msfvenom exposes a NOP generator through `-n`, which prepends a nopsled of the given length to the payload [50]. NOP modules sort by architecture the way encoders do. `modules/nops` mirrors the layout with its own directory tree [39]. Inside the console, `show nops` lists what the running instance offers, with generators such as `opty2` and `single_byte` [38]. The type is small. A current 6.5 checkout carries 14 NOP modules against 2,677 exploits [4].

ATT&CK metadata

Modules carry tags naming the ATT&CK techniques they implement, and the console searches that metadata directly.

```
1 search att&ck:T1059.001
```

That query finds every module leveraging T1059.001, Command and Scripting Interpreter: PowerShell [52]. Search `att&ck:T1059` instead, and the hierarchy brings in sub-technique modules such as `exploit/windows/mysql/mysql_mof` [52]. The module library implements

techniques across all fourteen tactics [11]. Rapid7 added the tagging in 6.5, as Chapter 1 covered [52, 4].

Technique tags join the other metadata modules carry. Rapid7 maintains the tags, and the console exposes them through the `att&ck:` search modifier. References map to external identifiers. `['CVE', '2024-XXXXX']` and `['URL', 'https://...']` are the common rows [32]. The console search exposes `cve` and `edb` as keywords alongside `platform` and `rank` [53]. The Modules page in the Pro interface shows the same fields, including operating system and CVE [54].

Two more metadata blocks matter when you read a module listing. `Notes` declares operational characteristics: `Stability`, `SideEffects`, and `Reliability`, with valid values sourced from `lib/msf/core/constants.rb`. `Exploit`, `auxiliary`, and `post` modules must declare them; `payload`, `encoder`, `nop`, and `evasion` modules do not [32]. `Rank` describes exploit reliability, from `ManualRanking` to `ExcellentRanking`, and only exploits carry it [54, 32].

Threat-intelligence reports name techniques, so you can convert a technique straight into a module search instead of guessing module names [52]. When an engagement brief says the adversary used PowerShell, run `search att&ck:T1059.001` and let the metadata find the modules for you.

Building Your Attack Lab

This chapter stands up the lab that later chapters assume. You build a Kali attack box, a Metasploitable target, and Docker containers for web targets, all pinned to an isolated host-only network [55, 56]. Reproducibility is the point of the arrangement. You can repeat, break, and verify each exercise instead of repairing it, because a snapshot restores any damage and the network keeps it off your production traffic. The sections that follow install the framework, build the machines, start the console, and pin the lab down with hostnames and snapshots.

Installing the Metasploit Framework

You have three ways to get the Metasploit Framework, and they suit different readers. On Kali Linux, it is already installed [57, 58]. On other Linux distributions, Rapid7 ships a nightly installer script that builds a self-contained package for your distribution [57, 59]. If you plan to contribute modules or fixes, you clone the repository and run it from source [60, 61]. The rest of this chapter assumes Kali. The installer and the from-source path matter when a Kali VM is not an option.

The non-Kali path is the “omnibus” installer, the packaging system Rapid7 uses to bundle what the framework needs to run: dependent libraries, a specific Ruby version, and a built-in PostgreSQL database, wrapped into native .deb, .rpm, .msi, or .pkg packages [62, 63]. Rapid7 rebuilds the installers nightly, so `msfupdate` pulls in the newest exploits and payloads, and the installers themselves are open source [62, 63]. The package is not Metasploit Pro. It contains only the command-line interface and the associated tools and modules [62, 63]. It also installs companion tools such as John the Ripper and Nmap [57].

```
1 curl https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/templates/metasploit-framework-wrappers/msfupdate.erb > msfinstall &&
   → chmod 755 msfinstall && ./msfinstall
```

That line downloads Rapid7's `msfinstall` script and runs it with root privileges, and the script detects your distribution and installs the matching package [57, 59]. On Kali the distribution package, installed with `sudo apt install metasploit-framework`, takes about 600 MB of disk [58]. Whichever package you chose, confirm the install with `msfconsole --version`, which prints the framework build [59, 64]. Then start `./msfconsole` for the full console. The console asks whether you want to use and set up a new database, and answering `y` runs the initial configuration script [57]. The banner that follows lists this build's module counts, and the session ends at the `msf>` prompt [57]. On Kali the same setup happens later in this chapter with `msfdb init` instead of the prompt [65].



Antivirus software detects the Metasploit Framework as malicious, because the framework triggers the same signatures as the exploits it launches. Disable antivirus for the install or exclude the Metasploit directory from the scan, and remember that local firewalls can interrupt the download too [57].

Set up a development environment only when you plan to contribute. For authorized testing, Rapid7's guide recommends the packaged installer or the pre-installed Kali version [60]. The dev path swaps the binaries for a git clone of the repository. You add a Ruby toolchain under a manager like RVM, plus PostgreSQL if you want the database and REST API, and then run `./msfconsole` from the checkout [60, 61]. The guide assumes an apt-based Linux and a non-root user [60], and it expects a GitHub account with an SSH key [60]. It spells out the sequence:

```
1 sudo apt update && sudo apt install -y git autoconf build-essential libpcap-dev
  ↳ libpq-dev zlib1g-dev libsqlite3-dev
2 git clone git@github.com:YOUR_USERNAME_FOR_GITHUB/metasploit-framework
3 cd metasploit-framework
4 gem install bundler
5 bundle install
6 ./msfconsole
```

Maintenance is the price. Updates are `git pull` plus `bundle`, a broken dependency is yours to fix [60, 59], and what you get in exchange is access to the newest features and fixes before the packages carry them [59]. For a lab whose purpose is repeatable exercises, the packaged framework is the right choice.

Installing on Kali Linux

Download the prebuilt VirtualBox image from kali.org/get-kali, a 3-4 GB OVA, and import it with File > Import Appliance, accepting the defaults [66]. Boot, log in with the default credentials `kali/kali`, and change the password immediately [66, 56]. Update before anything else with `sudo apt update && sudo apt upgrade -y` [66, 67]. In the VM settings, give the image at least 4 GB of RAM and 2 CPU cores, and install the VirtualBox Extension Pack from the same download page while you are at it [56].

How much machine you need scales with how many targets you run. A practical starting point is 12 GB of RAM, enough to run Kali and Metasploitable with DVWA in Docker alongside them [55]. With 8 GB, drop Metasploitable and keep the Docker target. At 16 GB you have room to add a first VulnHub machine [55].

The lab network

Keep the lab off your production traffic with a host-only adapter, a private segment shared by the host and its VMs with no route out to the internet or to your LAN [55, 68]. In VirtualBox, create it through File > Tools > Network Manager, and the host-only network usually lands on `192.168.56.0/24` [56]. Give each VM a static address on that segment, and record the plan:

```
1 Host-Only Network: 192.168.56.0/24
2 Kali Linux:        192.168.56.101
3 Metasploitable 2:  192.168.56.102
```

Kali takes `.101` and Metasploitable takes `.102`. A later target takes the next free address in the range. Each VM sets Adapter 1 to Host-Only and selects `vboxnet0` [56]. The attacker additionally needs an Adapter 2 in NAT mode for updates, because NAT lets a VM reach the internet through your host without exposing it to the LAN [69]. Vulnerable targets get host-only alone, so even a compromised target cannot reach your LAN [70, 56, 68].

The targets

Metasploitable 2 is your first target, an intentionally vulnerable Ubuntu VM built by Rapid7 for Metasploit practice [55]. It runs more than 30 vulnerable

services: FTP with the vsftpd 2.3.4 backdoor, SSH, Telnet, HTTP with DVWA and phpMyAdmin, and Samba, each with default credentials [55]. Download the image from SourceForge and create a VM that uses the extracted `.vmdk` as its disk, with 1 GB of RAM on `vboxnet0` [56]. Log in at the console with `msfadmin/msfadmin` [56]. Once it boots, scan it from Kali with `nmap -sV 192.168.56.102`, and the output fills a screen with open ports and service versions, which is the point of the box [71]. Docker-only setups get the same target as the `tleemcjr/metasploitable2` container image [71].



Never put Metasploitable on a bridged or NAT network. Its FTP service is backdoored and its credentials are public by design, so anyone who can reach the box can compromise it in minutes [55, 72].

DVWA (Damn Vulnerable Web Application) is the web target, a deliberately vulnerable PHP/MySQL application organized around the OWASP Top 10 with difficulty levels that run from low to impossible [56, 71]. The fastest path is Docker inside Kali:

```
1 sudo apt install docker.io -y
2 sudo docker run -d -p 80:80 vulnerables/web-dvwa
```

The first command installs Docker, and the second runs the DVWA image and publishes its web port on the Kali VM as port 80 [56]. Open `http://127.0.0.1` in the Kali browser. The first visit shows a setup page, so click **Create / Reset Database** to build the application database before the login form appears [66]. Then log in with `admin/password` [55, 71]. Because DVWA runs inside the attack VM, it needs no extra network configuration, and its traffic stays on the machine [55]. Containers also keep the lab light, because the same host that would struggle to hold three VMs will take ten containers, and a fresh image pulls in under a minute [66]. If you outgrow DVWA, the OWASP Broken Web Applications image bundles a collection of vulnerable web apps in one VM [66].



If your hypervisor cannot run x86 guests, VirtualBox on Apple Silicon for example, Metasploitable will not boot there, so run the targets as Docker containers on an internal network instead [66, 71].

Launch msfconsole in Kali

After installation, the framework is present but its database is not running. Kali's Network Services Policy starts no network services at boot, database services included [65]. The first launch takes two commands, and it begins with the database.

```
1 sudo msfdb init
```

`msfdb init` starts PostgreSQL and creates the framework database: the `msf` user, the `msf` and `msf_test` databases, the configuration file `/usr/share/metasploit-framework/config/database.yml`, and the initial schema [65]. Success looks like this:

```
1 [+] Starting database
2 [+] Creating database user 'msf'
3 [+] Creating databases 'msf'
4 [+] Creating databases 'msf_test'
5 [+] Creating configuration file
  ↳ '/usr/share/metasploit-framework/config/database.yml'
6 [+] Creating initial database schema
```



`sudo msfdb run` performs the same initialization and then launches `msfconsole` in one step [65].

Now start the console the way the Kali guide does, and verify the database:

```
1 $ msfconsole -q
2 msf6 > db_status
3 [*] Connected to msf. Connection type: postgresql.
```

The `-q` flag drops the banner so you land directly on the `msf6` prompt, and `db_status` confirms the connection to PostgreSQL [65]. The prompt number tracks your framework version, `msf6` on a 6.x build and `msf5` on an older one [73]. On a fresh install the first launch also builds the module cache in the background, a step that can take 2-5 minutes [57]. Verify that PostgreSQL is listening with `sudo msfdb status`, or by checking port 5432 in `ss -ant` [65]. The status output shows the service as `active` with a `postgres` listener on `localhost:5432`, the two lines that tell you the database is reachable [65]. Afterwards the same `msfdb` command manages the lifecycle:

```

1 kali@kali:~$ sudo msfdb
2 Manage the metasploit framework database
3   msfdb init      # start and initialize the database
4   msfdb reinit    # delete and reinitialize the database
5   msfdb delete    # delete database and stop using it
6   msfdb start     # start the database
7   msfdb stop      # stop the database
8   msfdb status    # check service status
9   msfdb run       # start the database and run msfconsole

```

That help block is the Kali packaging of `msfdb`, the tool this chapter reaches for whenever the database misbehaves [65, 58]. With the database connected, prove it does something useful. Scan Metasploitable straight from the console. The Metasploit cheatsheet documents the shape of that output, which for one vulnerable container looked like this [74]:

```

1 msf6 > db_nmap -sV 192.168.1.5
2 [*] Nmap: Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-08 03:47 UTC
3 [*] Nmap: Nmap scan report for 192.168.1.5 (192.168.1.5)
4 [*] Nmap: Host is up (0.12s latency).
5 [*] Nmap: rDNS record for 192.168.1.5: 192.168.1.5.vultr.com
6 [*] Nmap: Not shown: 991 closed ports
7 [*] Nmap: PORT      STATE      SERVICE      VERSION
8 [*] Nmap: 21/tcp    open      ftp          ProFTPD 1.3.5
9 [*] Nmap: 22/tcp    open      ssh          OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13
   ↳ (Ubuntu Linux; protocol 2.0)
10 [*] Nmap: 25/tcp    filtered  smtp
11 [*] Nmap: 80/tcp    open      http         Apache httpd 2.4.7 ((Ubuntu))
12 [*] Nmap: 139/tcp   open      netbios-ssn Samba smbd 3.X - 4.X (workgroup:
   ↳ WORKGROUP)
13 [*] Nmap: 445/tcp   open      netbios-ssn Samba smbd 3.X - 4.X (workgroup:
   ↳ WORKGROUP)
14 [*] Nmap: 631/tcp   open      ipp          CUPS 1.7
15 [*] Nmap: 6667/tcp  open      irc          UnrealIRCd
16 [*] Nmap: Service Info: Hosts: VULN, irc.TestIRC.net; OSs: Unix, Linux; CPE:
   ↳ cpe:/o:linux:linux_kernel
17 [*] Nmap: Service detection performed. Please report any incorrect results at
   ↳ https://nmap.org/submit/ .
18 [*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 18.18 seconds

```

Run the same command against 192.168.56.102 when Metasploitable is booted. The rows differ from the container's: Metasploitable serves vsftpd 2.3.4 on FTP rather than ProFTPD [55]. `db_nmap` wraps Nmap and writes each discovered host and service into the current workspace as the scan runs, so the hosts and services commands then answer from stored data [74, 75].

`hosts` lists the address and `services` lists what that scan stored, which means the next chapter starts against a populated database [74, 75].

Startup troubleshooting

Three failures cover most of what goes wrong. If `msfconsole` exits on launch before you have configured a database, start it with `msfconsole -n`, the flag that runs without database backing [58, 76]. Rapid7 documented `-n` as the workaround when a missing database crashed startup, and the fix landed in pull request #3666 [76]. Current builds skip that crash. On the omnibus path the console offers to set up a database on first run [57], and on Kali `msfdb init` covers the same ground [65]. Drop the flag once the database exists.

If `db_status` reports no database connection, PostgreSQL is probably not running. Start it with `sudo systemctl start postgresql` or `sudo msfdb start`, then rebuild the database with `sudo msfdb reinit` [59]. If the console aborts over the Bundler gem, the installation is missing its Ruby dependencies, so install bundler with `gem install bundler` and run `bundle install` inside `/opt/metasploit-framework` [59].

DNS configuration

Metasploit handles DNS itself. As modules run, the framework answers their hostname lookups directly, which keeps queries from leaking out of your segment and, later, lets you resolve internal names through a pivot session [12]. The console configures that resolver with three commands: `dns add` for rule-based resolution, `dns add-static` for fixed hostnames, and `dns print` to show the configuration [12].

```
1 dns add --rule *.lab.lan --session 1 --index 1 192.0.2.1
2 dns add --rule honeypot.lab.lan --index 2 black-hole
3 dns add-static example2.lab.lan 192.0.2.201
4 dns add --index 1 --rule * static system 192.0.2.1
```

The first resolves anything under `.lab.lan` through the nameserver 192.0.2.1, contacted via session 1, so the lookup originates from the compromised host rather than from your machine [12]. The second routes the honeypot hostname to `black-hole` [12]. The third pins a single hostname to

an address, and the fourth sends the remaining hostnames to your system's own resolver [12]. View the whole configuration with `dns print`:

```

1  msf6 > dns print
2  Default search domain: N/A
3  Default search list:
4    * tor.example.com
5    * localdomain
6  Current cache size:    0
7  Resolver rule entries
8  =====
9      #  Rule                Resolver    Comm channel
10     -  ----                -
11     1  *.lab.lan            192.0.2.1   Session 1
12     2  honeypot.lab.lan    black-hole  N/A
13     3  *
14     .   \_                  static       N/A
15     .   \_                  10.4.5.45
16     .   \_                  10.3.20.98
17  Static hostnames
18  =====
19      Hostname                IPv4 Address  IPv6 Address
20      -----                -
21      example.lab.lan         192.0.2.200
22      example2.lab.lan        192.0.2.201

```

The header lines above the rule table show the search domain and search list [12]. A new setup with no search domain prints N/A there [12].

For the lab, pin both machines beside the plan from “The lab network”. Run `dns add-static kali.lab.lan 192.168.56.101` and `dns add-static metasploitable.lab.lan 192.168.56.102`, so module output and loot read hostnames instead of IPs [12, 70]. Keep each machine's hostname in your lab notebook [70].

Keeping the lab current

Update the framework the way you installed it. On a package install, `msfupdate` pulls the latest release [77, 12], and on Kali the packaged framework updates through `apt` with `sudo apt upgrade metasploit-framework` [59]. Kali no longer supports `msfupdate`, because the package ships as part of the distribution, so the old update habit will fail there [78]. A fresh machine gets the same open-source nightly installers the first section used [77, 12]. Kali's

package tracks the distro's release cadence, so it can trail Rapid7's own releases by several patch versions [64].

Then make the lab reproducible, because that is the point of this chapter. Snapshot each VM after first boot and again before an exercise, then revert rather than repair after a test changes target state [70]. The payoff is that you can break a target badly and restore it in under 60 seconds, which is what makes repeated exercises cheap [55]. Keep the hostnames and the `dns add-static` lines from this chapter in a lab notebook, together with the host-only layout [70]. Create a workspace per exercise with `workspace -a`, so loot and findings stay isolated for each run, and switch back with `workspace <name>` when you revisit an exercise [75, 79].

Before each session, boot Kali and run `sudo msfdb run`, then settle in only when `db_status` reports a PostgreSQL connection [65]. Chapter 4 picks up at the `msf6` prompt and walks the `msfconsole` workflow against the Metasploitable target you just stood up.

The msfconsole Workflow

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

Metasploit commands

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

Getting help for msfconsole commands

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

Searching for exploits with keywords

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

Datastore

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

Finding modules

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

Understanding module options

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

Use, back, and exit commands

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

The Metasploit Database

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

Initialize the Metasploit PostgreSQL database

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

Database support

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

Database workspaces

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

Example workflow

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

Keep your Metasploit data safe

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

Using msfdb

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

Exploitation

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

Service Discovery and Scanning

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

Port scanning

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

Scanner HTTP auxiliary modules

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

Auxiliary module examples

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

Vulnerability scanning with Metasploit

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

A TCP port scanner

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

The Metasploitable scanning lab

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

Staged versus Stageless Payloads

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

How payloads work

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

What are staged payloads?

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

What are stageless payloads?

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

Staged versus stageless handlers

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

Connecting stageless payloads to staged handlers

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

Stageless mode

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

Working with payloads

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

Generating Payloads with msfvenom

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

Introducing msfvenom

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

How to use msfvenom

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

Output formats

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

Handlers

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

Advanced msfvenom payload generation

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

A worked example

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

Exploitation and Session Management

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

Working with exploits

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

Working with sessions

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

Checking target availability

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

Manage Meterpreter and shell sessions

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

Setting up a payload

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

Basic workflow

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

Meterpreter and Post-Exploitation

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

Meterpreter Fundamentals

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

What is Metasploit Meterpreter?

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

How Meterpreter works

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

Meterpreter flavors

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

Core commands

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

File system commands

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

Help

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

Meterpreter transport control

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

The lab

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

Migration and Privilege Escalation

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

Meterpreter getsystem

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

Migrate

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

priv_migrate

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

System commands

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

Other commands

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

The escalation lab

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

Pivoting and Routing

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

How pivoting works

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

Route-based pivoting

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

Port forwarding

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

Credentials, Persistence, and Lateral Movement

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

About post-exploitation

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

hashdump

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

Post-exploitation modules by category

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

Lateral movement and persistence

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

SMB and pass-the-hash

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

WinRM

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

SSH

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

Persistence

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

Cleanup, and the traces you cannot erase

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

Extending the Framework

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

Resource Scripts and Automation

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

Resource scripts

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

Six ways to automate Metasploit

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

Automating the Metasploit console

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

Basic workflow

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

Writing Custom Auxiliary Modules

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

Writing an auxiliary module

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

How to add a third-party module

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

Running private modules

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

Navigating the codebase

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

Using Rex::Proto::Http::Client

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

The `Exploit::Remote::Tcp` mixin

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

Module documentation

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

Writing Your First Exploit

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

Building your first Metasploit exploit

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

The check method

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

Targets, offsets, and bad characters

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

The exploit method

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

Setting up a Metasploit development environment

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

Writing module documentation

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

Capstone: your module in the psexec slot

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

Railgun and the Windows API

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

Using Railgun for Windows post-exploitation

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

Defining a DLL and its functions

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

MSF Meterpreter and Railgun

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

Evasion and Modern Context

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

Encoders and Evasion Limits

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

Encoders

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

AV bypass with Metasploit templates

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

A defender's checklist

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

Detecting Metasploit attacks

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

Hardening and defensive takeaways

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

Keeping Modules Current

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

Metasploit 6 in development

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

Metasploit Framework 6.0 release notes

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

Metasploit Framework 6.4

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

The 6.5 MCP server

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

When the APT repo falls behind

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

Reading the release notes

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

Future trends and emerging considerations

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

Appendix A: Command and module cheat sheet

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

Console startup

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

Navigation and workflow

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

search filters

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

Datastore scoping

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

Module layout

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

Payloads and msfvenom

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

Database commands

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

Sessions, jobs, handlers

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

Meterpreter commands

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

Pivoting

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

Post module pick list

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

Ranks and check codes

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

Resource scripts

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

Appendix B: Framework release history

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

Major releases

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

Version numbering and release cadence

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

Module inventory over time

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

Compatibility breaks

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

Editions and licensing

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

Appendix C: Auxiliary and post module index

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

Auxiliary modules

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

The Scanner mixin

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

Post modules

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

Windows gather and manage

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

Linux and multi-platform

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

Finding more

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

Glossary

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

References

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