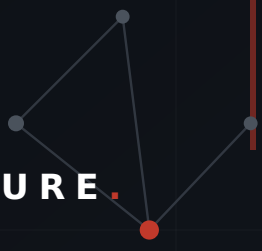
CODELIVLY

Vulnerability Research

A Practical Guide to Finding, Analyzing, Reproducing & Reporting Software Vulnerabilities



FIND. ANALYZE. PROVE. SECURE.

CODELIVLY · TECHNICAL PUBLICATION · 2026

Vulnerability Research

A Practical Guide to Finding, Analyzing, Reproducing & Reporting Software Vulnerabilities

PUBLISHED BY CODELIVLY

Vulnerability Research: A Practical Guide to Finding, Analyzing, Reproducing & Reporting Software Vulnerabilities

Published by Codelivly.

Copyright © 2026 Codelivly. All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

Educational purpose. This book is provided for educational and professional-development purposes. It teaches vulnerability research methodology. the disciplined process of finding, analyzing, reproducing, understanding, and responsibly reporting software vulnerabilities. for the purpose of improving software security.

Authorized research only. Every technique in this book is intended to be practiced only against software and systems that you own, that are explicitly designed for security training, or that you have clear, written authorization to test. Testing systems without authorization is illegal in most jurisdictions and is contrary to the ethics of the profession this book

describes. The reader is solely responsible for ensuring that any research they perform is lawful and authorized.

Responsible disclosure. This book advocates coordinated, responsible disclosure. Discovering a vulnerability creates a responsibility to handle it carefully. Readers are encouraged to follow the disclosure practices described in Chapters 17 and 18 and to respect the policies of the vendors and projects whose software they study.

Limitation of liability. The information in this book is provided on an "as is" basis, without warranty of any kind, express or implied. Neither the author nor Codelivly shall have any liability to any person or entity with respect to any loss or damage caused, or alleged to be caused, directly or indirectly, by the information contained in this book or by the use or misuse of any technique, tool, or code it describes.

No guarantee of results. Vulnerability research is a skill developed through practice over time. This book provides methodology, examples, and guidance, but it does not guarantee that any reader will discover vulnerabilities, obtain a CVE identifier, earn bug-bounty rewards, or achieve any particular employment or career outcome.

Third-party trademarks. All product names, logos, standards, and brands mentioned in this book are the property of their respective owners. Names such as OWASP, MITRE, CWE, CVE, NIST, CVSS, Ghidra, and others are used for identification and educational reference only. Use of these names does not imply endorsement. Codelivly is not affiliated with these organizations unless explicitly stated.

Fictional scenarios. The applications, services, companies, crash logs, diffs, and vulnerability reports used as examples in this book are fictional unless a real, verifiable source is explicitly cited. Fictional examples are clearly labeled. They are constructed to teach methodology and do not describe real security incidents, real products, or real organizations. Any resemblance to a real product or company is coincidental.

Tool behavior. Software tools change between versions. Commands, output, and behavior described in this book reflect general concepts and typical behavior at the time of writing and may differ in your environment. Always consult the official documentation for the version of a tool you are using.

First edition · 2026
Codelivly · *Find. Analyze. Prove. Secure.*

For the researchers who refuse to accept "it just crashed" as an answer,

for the reverse engineers who read what the compiler wrote, the developers who ask what their code does on its worst day, the open-source contributors who fix the bug and then the whole class of bugs, and the curious learners who keep asking ***why*** *until the software finally tells them.*

This book is for everyone who investigates why software fails, and then does the harder work of proving it, understanding it, and reporting it so that it gets fixed.

Acknowledgements

A book about vulnerability research is only possible because of the communities that built the discipline in the open. This book stands on the accumulated knowledge of people who chose to document how software breaks rather than keep it to themselves.

We are grateful to the **security research communities** who publish write-ups, give talks, and answer questions from strangers. The culture of sharing root-cause analysis, not just the fact that a bug exists, but the precise reason it exists, is what turns vulnerability research from folklore into a teachable craft.

We thank the **open-source developers and maintainers** whose projects serve as the training ground for nearly every researcher. Most people learn this work by reading real code, reproducing real fixes, and studying real patches. That is only possible because maintainers develop in public, write honest commit messages, and treat security reports with seriousness and grace, often as unpaid volunteers.

We thank the **vulnerability researchers, bug-bounty researchers, and CVE researchers** whose disclosures, advisories, and regression tests form a living curriculum. Every carefully written advisory teaches the next generation how to think.

We thank the **security educators** who insist that methodology matters more than any single tool, and the **tool developers** who build the disassemblers, fuzzers, debuggers, and sanitizers that make modern research possible, and who document them well enough for newcomers to begin.

We thank the organizations that maintain the public infrastructure of this field (the standards, taxonomies, and databases referenced throughout these pages) for making rigorous, shared vocabulary available to everyone.

Finally, we thank the **Codelivly learners and community**. This book was written for you, shaped by your questions, and tested against the thing that matters most: whether a motivated beginner can actually use it to become a researcher. Your persistence is the reason this book exists.

Foreword

Every piece of software is a theory about how the world will behave, written down in a language a machine can execute. The programmer imagines the inputs that will arrive, the order in which events will happen, the sizes of things, the states the system can be in. Then the software ships, and the real world arrives, larger, stranger, more hostile, and more creative than any theory anticipated. Vulnerability research is the discipline of finding the exact places where the theory and the world disagree, and where that disagreement has consequences for security.

It is tempting to imagine this work as a matter of running the right tool against the right target and reading off the answer. It is not. Tools are indispensable, and this book will teach you to use whole categories of them, but a tool only ever produces an observation. A fuzzer produces a crash. A scanner produces a warning. A disassembler produces a listing. None of those things is a vulnerability. A vulnerability is a *claim*, a claim that under specified conditions, a specific flaw in specific software lets an attacker cross a specific security boundary, and a claim is only worth anything when it is supported by evidence that another person can reproduce. The distance between "the program crashed" and "here is a reproducible, root-caused, responsibly reported security vulnerability" is where the entire profession lives.

That distance is crossed by a method, and the method is older than computing. Observe something unexpected. Form a hypothesis about why it happened. Design an experiment that would confirm or refute the hypothesis. Run it. Look honestly at the evidence, including the evidence that you were wrong. Revise. Repeat until you understand the mechanism well enough to explain it to someone else and have them reproduce it on the first try. This is the scientific method, applied to artifacts that human beings built and can therefore fully understand. The researchers who last in this field are not the ones with the most tools or the most aggressive instincts. They are the ones who are honest about what their evidence does and does not show, and patient enough to keep going after the first ten hypotheses turn out to be wrong.

Software complexity is why the work will never run out. A modern application is a tower of dependencies, each written by different people at

different times under different assumptions, communicating across interfaces that no single person fully understands. Every interface is a place where one component's assumptions meet another component's behavior, and every mismatch is a candidate. The attack surface of real software is not a wall to be scaled; it is a coastline, fractal and effectively endless, and the researcher's skill is in knowing where to look and how to look closely.

This book takes a deliberate position: that the ability to *explain* a vulnerability is as important as the ability to find one. A finding that cannot be reproduced is a rumor. A finding that cannot be explained is a liability, to the vendor who must fix it, to the users who depend on the software, and to the researcher whose credibility rests on being right. The chapters that follow spend as much effort on reproduction, root-cause analysis, and reporting as they do on discovery, because that is the correct ratio for real work.

If you read this book the way it is meant to be read, with a lab open, a research journal beside you, and a willingness to record your failed experiments as carefully as your successful ones, you will come out the other side thinking differently about software. You will stop seeing programs as black boxes that either work or do not, and start seeing them as mechanisms whose behavior can be investigated, predicted, and proven. That shift in perception is the real beginning of a research career.

, The Codelivly Editorial Team

Preface

Why this book exists

There is a great deal of cybersecurity material in the world, and very little of it is about vulnerability research specifically. Most of it teaches you to *use* the output of research, to run a scanner that detects known vulnerabilities, to fire a published exploit at a lab machine, to follow a checklist of common web flaws. Those are useful skills, and some of them overlap with research, but they are downstream of it. Someone has to find the vulnerability in the first place, understand it well enough to explain it, and report it well enough that it gets fixed. That someone is a vulnerability researcher, and the path to becoming one is poorly documented.

This book exists to document that path. It is not a catalog of tricks and it is not a tour of tools. It is an attempt to teach the *method* of vulnerability research, a repeatable way of moving from an unexpected software behavior to a reproducible, technically understood, responsibly reported vulnerability, and to teach it deeply enough that you can apply it to targets this book has never seen.

Who should read it

This book is written for someone who already has a working foundation and wants to specialize. We assume you are comfortable at a Linux command line, that you can read and write code in at least one language, that you understand the basics of how networks and operating systems work, and that you know what a web request is. We do not assume you have ever performed professional vulnerability research, reverse-engineered a binary, written a fuzzing harness, or triaged a crash. Everything specific to research, we teach from the ground up.

If you are a developer who wants to understand how your code gets attacked, a penetration tester who wants to go deeper than running other people's exploits, a student aiming at a security-research career, or a self-taught enthusiast who keeps hitting the ceiling of checklist-based security, this book is for you.

What vulnerability research actually involves

It involves far more reading than writing, far more analysis than exploitation, and far more failed hypotheses than successful ones. A realistic research session looks like this: you pick a target, you spend hours building a mental model of how it processes input, you form a guess about where it might be fragile, you build a small experiment to test that guess, and the guess is wrong. So you revise it and try again. Most of the work is understanding. The dramatic moment of discovery, when it comes, is the short payoff at the end of a long, patient, methodical process. This book tries to show you that process honestly, including the parts that are slow and the parts that do not work.

How this book differs from a penetration-testing book

A penetration test is a time-boxed, goal-oriented assessment of a system's security, usually against a defined scope and usually using a mixture of known techniques. Its deliverable is an assessment: here is what we found, here is the risk, here is what to fix. Vulnerability research is open-ended and discovery-oriented. Its deliverable is *new knowledge*: here is a flaw that was not previously known, here is exactly why it exists, here is how to reproduce it, here is how to fix it. A penetration tester asks "is this system secure enough?" A vulnerability researcher asks "what is wrong with this software that nobody has noticed yet, and can I prove it?" The skills overlap, but the mindset, the timescale, and the output are different, and this book is firmly about the second discipline.

How to use the exercises

Every chapter contains exercises and projects that are meant to be done, not read. Vulnerability research is a motor skill as much as an intellectual one; you cannot acquire it by understanding it in the abstract. The projects are designed around targets you can safely and legally set up yourself: software you own, intentionally vulnerable training applications, open-source code you study locally, and capture-the-flag environments. None of them require, or permit, testing systems you do not have authorization to test.

Building a research lab, keeping notes, building a portfolio

Three practices run through this entire book and are worth establishing before you reach Chapter 1. First, build a **lab**, an isolated environment

where you can run, break, and observe software without risk to yourself or anyone else. Chapter 2 walks you through this in detail. Second, keep a **research journal**. Vulnerability research generates a large amount of state (hypotheses, commands, outputs, dead ends) and a researcher who does not write it down will lose it and repeat work. The journal is not optional; it is the instrument that makes your work reproducible. Third, turn your work into a **portfolio**. Every project in this book is designed to produce an artifact (a harness, a write-up, a report) that you can clean up and publish. A credible portfolio of research write-ups is the single most effective way to demonstrate this skill to the world, and you should start building it on your first day, not your last.

How to Use This Book

Recommended reading order

The book is built to be read straight through the first time. The chapters are ordered to build capability in the sequence a researcher actually develops it: first you learn to see software as an attacker sees it (Chapters 1, 3), then to analyze it statically and dynamically (Chapters 4, 7), then to make it fail on purpose and understand the failures (Chapters 8, 10), then to apply those skills to specific domains (Chapters 11, 14), then to perform the advanced work that separates researchers from testers (Chapters 15, 19), and finally to build a career from the skill (Chapter 20).

That said, the book is also a reference. Once you have read it once, the chapters stand alone well enough to return to. If you already reverse-engineer comfortably, you can skim Chapter 6 and spend your time on fuzzing and patch analysis. If you work on the web, Chapters 3, 4, 10, and 11 form a coherent track. The appendices (checklists, templates, taxonomy, and glossary) are designed to be used during real work, not just read once.

Lab methodology

Do every practical exercise in an isolated lab, never against systems you do not own or have explicit authorization to test. Chapter 2 specifies the lab in detail, but the governing rule is simple and absolute: **if you do not own it and nobody with the authority to grant permission has given you written permission to test it, it is off limits.** The entire profession depends on researchers who honor this line. Your lab is where you are free; the rest of the world is not your lab.

The research journal

Keep a journal from the first exercise. A simple, consistent structure beats an elaborate one you will not maintain. The template in Appendix C and the notebook system described at the end of this front matter give you a starting point. Record what you tried, what you expected, what actually happened, and what you concluded, especially when the conclusion is "that hypothesis was wrong."

Evidence collection and reproducibility

Treat every finding as something another person will try to reproduce, because in real research they will. Capture the exact input, the exact version of the software, the exact environment, and the exact steps. A finding you cannot reproduce on demand is not yet a finding. Reproducibility is not a formality you perform at the end; it is a property you build in from the start by being disciplined about recording state.

Building progressively harder projects

The projects escalate in difficulty on purpose. Early projects use small, well-understood targets so that the methodology is the only hard part. Later projects use larger, messier targets so that you practice the methodology under realistic conditions. Resist the urge to jump ahead to the hardest target; the point of the early projects is to make the method automatic so that your attention is free for the target when it gets genuinely difficult.

Recording failed experiments

This is worth stating on its own because beginners resist it. **Failed experiments are data.** When a hypothesis is refuted, you have learned something true about the target: it is *not* fragile in the way you guessed. That narrows the search. A research journal full of refuted hypotheses is not a record of wasted time; it is a map of the territory you have ruled out, and it is often the thing that makes the eventual discovery possible. Record your failures as carefully as your successes.

Turning research into portfolio material

At the end of every project, the book tells you what to add to your portfolio. Do it while the work is fresh. A research write-up written a week after the fact is a pale shadow of one written the same day. Over the course of this book you will accumulate a dozen or more portfolio pieces (harnesses, analyses, reports, and end-to-end case studies) that together demonstrate exactly the skill a prospective employer or collaborator is looking for.

A NOTE ON LEGALITY AND ETHICS

Throughout this book, every technique is taught for defensive and research purposes and framed around authorized targets. Finding a vulnerability is the beginning of a responsibility, not the end of a game. If you take one principle from these pages, take this: research only what you are authorized to research, and disclose what you find responsibly. The skills in this book are powerful; the discipline around them is what makes you a professional rather than a liability.

The Research Notebook System

Before Chapter 1, establish the single most important habit in this book: writing down what you do. The following template reappears, adapted, throughout the book and in Appendix C. Copy it into a plain-text file, a Markdown document, or a dedicated notes application, the format matters far less than the discipline of filling it in.

RESEARCH NOTEBOOK ENTRY TEMPLATE

```
RESEARCH QUESTION  : What am I trying to find out? (one sentence)
TARGET             : Software name
VERSION            : Exact version / commit hash
SCOPE              : What I am and am NOT authorized to touch
ATTACK SURFACE     : Which entry point / component is under study
INITIAL OBSERVATION: What unexpected thing did I notice?
HYPOTHESIS         : Why do I think it happens? (a testable claim)
EXPERIMENT         : The exact test I will run to confirm/refute it
RESULT             : What actually happened (paste real output)
EVIDENCE           : Artifacts captured (inputs, logs, screenshots,
hashes)
NEW HYPOTHESIS     : If refuted, the revised claim
ROOT CAUSE         : The precise mechanism, once understood
SECURITY IMPACT    : Which boundary is crossed, and what an
attacker gains
REPRODUCTION       : Minimal, exact steps another person can follow
POC                : Location of the minimal proof-of-concept
REMEDIATION        : What fix would address the root cause
REFERENCES         : Standards, docs, prior art consulted
FINAL CONCLUSION   : What I now know that I did not know before
```

You will not fill in every field on every entry, and that is expected. Early in an investigation you may have only a research question, a target, and an observation. The lower fields fill in as the work progresses, and many entries will end at `NEW HYPOTHESIS` because the experiment refuted your idea and sent you back to the beginning. That is normal. The notebook's job is to make your reasoning explicit and your results reproducible, so that future-you (and anyone you report to) can follow the chain from first observation to final conclusion.

The chapters ahead will refer back to this template constantly. The whole book, in a sense, is an extended lesson in filling it in well.

Table of Contents

FRONT MATTER

Foreword vi
Preface viii
How to Use This Book xi
The Research Notebook System xiv

PART I · FOUNDATIONS OF THE DISCIPLINE

1 Introduction to Vulnerability Research 2
2 Building a Vulnerability Research Lab 17
3 Understanding Software Attack Surfaces 32

PART II · ANALYSIS SKILLS

4 Source Code Review for Vulnerability Research 45
5 Program Analysis Fundamentals 60
6 Reverse Engineering for Vulnerability Researchers 72
7 Memory Safety and Memory Corruption 86

PART III · MAKING SOFTWARE FAIL

8 Fuzzing Fundamentals 101
9 Crash Triage and Root-Cause Analysis 114
10 Vulnerability Classes 126

PART IV · DOMAINS OF RESEARCH

11 API and Web Vulnerability Research 142
12 Mobile Vulnerability Research 153
13 Network Protocol and Service Research 162
14 Dependency, Supply Chain & Update Mechanism Research 172

PART V · ADVANCED RESEARCH & CAREER

15 Patch Analysis and Variant Hunting 182
16 Exploitability Analysis and Proof of Concept 194
17 Classification, Severity and CVE Research 204
18 Vulnerability Reporting and Responsible Disclosure 215