



THE COMPLETE CVE HUNTING GUIDE

By Delta Obscura

THIS IS WHERE YOU BECOME AN OBSCURA

A step-by-step guide from Delta Obscura members, built on proven training methodologies, to teach you the practical ins and outs of CVE hunting.

Table of Contents

Changelog.....	3
Foreword.....	4
Author(s).....	6
Contributors	6
What is CVE Hunting?.....	7
CVSS Score	7
Bug Hunting	9
Bug Bounties or Bug Bounty Hunting	9
Types of hacking.....	10
Web hacking.....	10
API Hacking	10
IoT Hacking.....	11
Binary exploitation	11
Radio hacking	11
How to find targets?	13
Google.....	13
GitHub Topics.....	13
GitHub GHSA	14
GitHub Search.....	14
GitHub Organizations	15
Buildwith	15
DockerHub	16
What to do after finding a target?.....	16
How to determine impact of a target?.....	17
Builtwith	17
GitHub repository	18
Product's installation tracker	18
Ask them	18

Training resources	19
Free resources	20
PortSwigger Academy	20
Flask tutorials by Miguel Grinberg	21
Hamy’s CVE Hunting video demonstration	21
OverTheWire – Natas	21
Paid resources	22
HackTheBox subscription	22
Offensive Bug bounty hunter 2.0	22
Where should you start?	22
Tools	24
Bug hunting tips	25
Hamed Kohi (0xHamy)	25
Tip #1	25
Report writing	26
CVE Numbering Authorities (CNAs)	27
MITRE	27
VulDB	27
Trend Micro’s ZDI	27
GHSA	28
Other CNAs	28
CVE Alternatives	28
Which one should you use?	28
Responsible disclosure	29
How the 90-day rule works	29
How to put your skills to the test?	30
Conclusion	31

Changelog

This document was first released on August 23rd, 2025, it gets updated frequently due to new information being added. We don't keep an extensive changelog but every time this manual is updated, we mention it here with highlights of what topics were added.

August 23rd, 2025

Initial release.

Foreword

CVE hunting gets a bad rep. Most people think it's some exclusive boys' club where you need expensive toys, deep connections, and secret handshakes to get anywhere. That's complete nonsense. The reality? It's messier, grittier, and way more accessible than anyone wants to admit. If you're curious enough to poke around and stubborn enough to stick with it when things get weird, you can absolutely find stuff that matters.

I wrote this book because I got tired of seeing good people convinced they weren't "technical enough" or didn't have the "right background" to contribute. That's garbage. The security community loves to mystify this work, but honestly? Most of the best researchers I know started as complete outsiders who just refused to accept "that's how it works" as an answer.

Here's what you're actually going to discover:

1. Targets are everywhere once you know how to spot them.
2. Your curiosity matters infinitely more than whatever certifications are gathering dust on your wall.
3. The tools that find the really interesting bugs usually cost nothing.
4. The difference between beginners and experts isn't some magical knowledge, it's just being willing to keep digging when everyone else gives up.
5. Reading boring documentation will teach you more than any flashy exploit framework ever will.

Look, this isn't about collecting CVEs like Pokemon cards or building some kind of hacker reputation. It's about learning to see systems the way attackers do, so you can actually make them safer.

The problem-solving skills you'll develop, the respect you'll earn from people who know what they're talking about, and yeah, the satisfaction of knowing you've helped protect actual humans from actual bad guys, that's what makes this worth doing.

If you're brand new to this, I'm going to show you exactly how to get your first CVE without any of the usual gatekeeping nonsense. If you've been at this for a while, I'll help you get better at picking your battles, understanding what actually matters, and communicating your findings so people listen. Either way, you'll stop being a spectator and start being someone who actually moves the needle on security.

The Delta Obscura crew put this together because we believe the community gets stronger when more people contribute. There aren't any secret handshakes here, just work that needs doing and people willing to do it.

Consider this your invitation to get started.

Author(s)

Hamed Kohi (0xHamy)

Hamed Kohi, also known as [0xHamy](#), is a distinguished vulnerability researcher whose expertise has been recognized through 18 CVEs acknowledged by MITRE, VulDB, GitHub, and Apache. With specialized skills in vulnerability research and development and application security, he holds VHL+ and VHL Advanced+ certifications and has completed advanced coursework in CRTO and CPTS.

Beginning his cybersecurity journey in Kabul in 2020, where he developed foundational expertise in ethical hacking, programming, and malware research, Kohi immigrated to Canada in 2022 and has since expanded his focus to vulnerability analysis and exploit development.

As a CTF designer for events including iHack and HackFest Canada, he contributes to creating challenging security scenarios for the cybersecurity community. Currently serving as Founder & Director of [Cyber Mounties Canada](#), Security Researcher at [Delta Obscura](#), and Content Creator & Training Provider at [Cyber Mounties Academy](#), Kohi continues to advance the field through open-source security contributions, technical blogging at hkohi.ca, and mentoring emerging researchers.

Contributors

Thanks to the following people for helping me write this book.

- ❖ Alexandru Ionut Raducu ([@Xoriath](#))

What is CVE Hunting?

A CVE (Common Vulnerabilities and Exposures) is simply a publicly disclosed vulnerability in software or hardware. If you discover a vulnerability, especially in open-source code, you can potentially be assigned to your own CVE, provided you report your findings through the appropriate channels.

CVE hunting is just bug hunting but with a different purpose, CVE hunting involves hunting for bugs or vulnerabilities in open-source software because finding vulnerabilities here will often result in a CVE. CVE IDs are used to keep track of vulnerabilities.

Here is how a CVE looks like:

CVE-2025-9001

You can search for it on [cve.org](https://www.cve.org) and get information about it such as what software it affects:

<https://www.cve.org/CVERecord?id=CVE-2025-9001>

Acquiring a CVE ID for a vulnerability proves that you can identify vulnerabilities in real-world environments.

CVSS Score

The **CVSS score** (Common Vulnerability Scoring System) is an open standard used to rate the severity of security vulnerabilities in software, hardware, or firmware. It gives a numerical value (0.0 to 10.0) and a qualitative label (e.g., *Low*, *Medium*, *High*, *Critical*) to describe how dangerous a vulnerability is.

It's often used as a way of describing severity of CVEs but can be used outside CVEs as well. Besides severity, we also have **CVSS Vector String** which is a way of describing how a vulnerability is exploited.

Here is an example of what a CVSS string looks like:

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:L/SC:N/SI:N/SA:N/E:P

This implies a 6.9 score in CVSS version 4.0, there are multiple versions of CVSS with varying calculations, CVSS version 3.1 is widely used. The following is an interpretation of that vector string.

Exploitability metrics:

- **AV:N** – *Attack Vector: Network*
→ Vulnerability can be exploited remotely over a network.
- **AC:L** – *Attack Complexity: Low*
→ No special conditions needed; straightforward to exploit.
- **AT:N** – *Attack Requirements: None* (new in v4.0)
→ Attacker doesn't need to collect additional info or perform setup steps.
- **PR:N** – *Privileges Required: None*
→ Attacker doesn't need any prior access.
- **UI:N** – *User Interaction: None*
→ No user needs to click/approve anything for the exploit to succeed.

Impact metrics:

- **VC:N** – *Vulnerable system Confidentiality: None*
→ Exploit doesn't directly expose sensitive data on the vulnerable system.
- **VI:N** – *Vulnerable system Integrity: None*
→ Exploit doesn't allow attacker to alter data on the vulnerable system.
- **VA:L** – *Vulnerable system Availability: Low*
→ Exploit causes a partial performance degradation or small downtime.
- **SC:N** – *Subsequent system Confidentiality: None*
- **SI:N** – *Subsequent system Integrity: None*
- **SA:N** – *Subsequent system Availability: None*
→ No cascading impact on downstream systems (important in v4.0, which distinguishes between *vulnerable system* and *subsequent system* impacts).

Threat Metric

- **E:P** – *Exploit Maturity: Proof-of-Concept*
→ A proof-of-concept exploit exists, but no widely available weaponized exploit yet.

Human translation:

This vulnerability is easy to exploit over the network without any privileges or user action. However, it only has a low impact on system availability and no impact on confidentiality or integrity. A proof-of-concept exploit exists, but it's not fully weaponized.

Bug Hunting

Bug hunting is the practice of actively searching for vulnerabilities, coding errors, or unexpected behaviors in applications, systems, or networks, usually to identify and fix them before attackers can exploit them. This can be done by security researchers, developers, or even hobbyists, and it's a cornerstone of modern cybersecurity.

It's important to note that **bug hunting can lead to earning a CVE**, but not always. **CVE hunting** is a more specific subset of bug hunting, where researchers focus on vulnerabilities in software that qualifies for a CVE (Common Vulnerabilities and Exposures) identifier, often open-source projects or widely used software components.

Think of it like this:

- **Finding a flaw on Facebook?** That's bug hunting, but it likely won't get you a CVE because Facebook doesn't assign them.
- **Finding a flaw in an open-source library?** That's bug hunting too, but because open-source software usually participates in CVE assignment, it's also CVE hunting.

In short, all CVE hunting is bug hunting, but not all bug hunting is CVE hunting.

Bug Bounties or Bug Bounty Hunting

Bug bounties are formal programs that organizations set up to reward people who report security vulnerabilities or critical software bugs. Companies or platforms (like HackerOne, Bugcrowd, or Intigriti) publish clear rules such as what systems are in scope for every bug bounty program. They also specify what types of bugs qualify, and how much they'll pay based on severity.

The goal is to crowdsource security testing: rather than relying only on internal teams, organizations invite external researchers to help find flaws before malicious actors do. A bug bounty program is essentially *the marketplace* where vulnerabilities are responsibly traded for recognition or money.

Bug bounty hunting, on the other hand, is the *activity* of participating in these programs, the practice of actively seeking bugs for rewards. A bug bounty hunter is usually a security researcher or ethical hacker who scans websites, apps, APIs, or networks looking for weaknesses that meet program criteria.

Types of hacking

There are various types of fields within cybersecurity and the offensive side of cybersecurity has multiple other sub-sections such as web application penetration testing, network penetration testing, reverse engineering and binary exploitation. Some are easier than others to break into.

Web hacking

Web hacking targets websites, web applications, and their underlying infrastructure. Attackers look for flaws such as Server-side request forgery (SSRF), cross-site scripting (XSS) and authentication bypasses.

Because websites are highly exposed and relatively easy to test, many researchers acquire CVEs (Common Vulnerabilities and Exposures) by reporting issues in popular content management systems, plugins, or frameworks.

In terms of **ease of entry**, **web hacking** is generally the most accessible, you can start with just a browser, proxy tools like Burp Suite or Caido, and public documentation. The other forms (especially binary exploitation and radio hacking) tend to have steeper learning curves, but they also yield more rare and valuable CVEs.

The following is an example of a CVE found on a web application by Alasdair Gorniak, a member of Delta Obscura:

<https://www.cve.org/CVERecord?id=CVE-2025-45892>

API Hacking

API hacking focuses on Application Programming Interface (APIs) that power mobile apps, services, or backend integrations.

Vulnerabilities include broken authentication, excessive data exposure, and insecure endpoints. Successful API attacks often lead to CVEs when they affect widely used platforms or libraries.

API issues can be harder to find than web app bugs because they require understanding API routes, how data flows and how authorization is enforced. Still, many organizations publish public APIs, making them attractive targets for researchers.

IoT Hacking

IoT hacking examines Internet of Things devices such as smart cameras, routers, or industrial sensors. Vulnerabilities may stem from weak default credentials, outdated firmware, or unsafe update mechanisms. These flaws frequently earn CVEs because IoT products are mass-produced, meaning a single bug affects thousands of devices.

However, testing IoT hardware can be trickier than web hacking since it sometimes requires physical access or specialized tools to analyze firmware.

Binary exploitation

Binary exploitation digs into compiled applications or operating systems at a low level, finding memory corruption bugs such as buffer overflows or use-after-free errors.

These vulnerabilities often lead to high-impact CVEs because they allow remote code execution or privilege escalation. This area is harder than web hacking because it demands knowledge of assembly, debugging, and exploit development but it produces some of the most valuable CVEs, especially in critical software.

The following is an example of a buffer overflow vulnerability found by OXHamy, a member of Delta Obscura:

<https://www.cve.org/CVERecord?id=CVE-2025-9001>

Radio hacking

Radio hacking targets wireless communications such as Wi-Fi, Bluetooth, RFID, or proprietary radio protocols. Attacks may involve sniffing, jamming, or injecting malicious packets to compromise devices.

When these weaknesses are found in popular chipsets or wireless standards, they can lead to high-profile CVEs affecting many products at once. Radio hacking requires specialized equipment (software-defined radios or SDR, antennas) and protocol expertise, making it more challenging than simply testing web apps.

The following is an example of a vulnerability involving radio hacking:

<https://www.cve.org/CVERecord?id=CVE-2025-1727>

How to find targets?

People often get confused when they've mastered all the "necessary" skills, collected shiny certifications, and still can't find a single target, despite the internet looking like an all-you-can-hack buffet. I've met folks with CPTS, CBBH, OSCP, and CRTO who stare at open-source web apps like they're ancient hieroglyphics, convinced there's nothing to hack.

I learned most of what I know from CPTS myself, so when I see people struggling, I have to wonder: did they learn the material to become actual penetration testers, or just to decorate their LinkedIn profiles?

CPTS isn't about memorizing flags, it's about building the mindset. They give you all those labs to wire your brain into connecting dots.

Mind you, 0xHamy doesn't have any of those fancy certs. He found 18x CVEs using a creaky laptop he literally bought from Skopje's scrap market, if that's not proof that mindset beats equipment, I don't know what is.

So when people say they "can't find targets," I can't help but chuckle. Finding them is usually the easiest part. Fifteen minutes of effort and you've got something to poke at.

Google

Google is, shockingly, still the best way to find targets. How, you ask? Brace yourself for some advanced wizardry: open the search engine, type "content management system," hit Enter, and... download something. Yes, it's really that complicated.

Of course, not every target will spin up with a single click, some actually come with documentation, installation guides, and manuals. Don't panic, though. These mystical scrolls contain words and sentences that can be read and followed... assuming you've unlocked the ancient skill of basic literacy somewhere between kindergarten and now.

GitHub Topics

One of the best ways to find open-source targets is using GitHub topics, topics on GitHub are like tags, all repositories can use one or multiple tags, a web application can use tags

like `api`, `web`, `cms`, `lms` and others to indicate what the application is about. This allows other people to discover your software.

For example, here is how I found over 1,000 Learning Management Software (LMS) by using `lms` topic:

<https://github.com/topics/lms>

If you open a LMS like `frappe`, you may notice that it contains several topics that can be used to find more software.

GitHub GHSA

Another way to find targets is by using [GitHub Security Advisory](#) or GHSA. GHSA is used to keep track of all vulnerabilities that were reported through GitHub's vulnerability reporting system.

You can filter the results by CWE to find vulnerabilities found on specific repositories, for example CWE-79 indicates Cross-Site Scripting, this is a common vulnerability often found on web applications. Searching for this CWE will bring up all repositories where this vulnerability was reported to:

<https://github.com/advisories?query=cwe%3A79>

Every vulnerability affects a repository; some repositories aren't necessarily accessible through GitHub so you may not be able to see the specific software a vulnerability affects.

If you are looking for targets with active maintainers to assign CVEs for the vulnerabilities you find, you should use GHSA.

GitHub Search

GitHub search is another way of finding vulnerabilities, the difference between GHSA and search is that with search, you can find any repositories even the ones without a vulnerability reporting program. With search and topics you can find targets with no CVE assignment program, these are usually less crowded repositories because there is no guaranteed incentive (e.g. CVE assignment) to hunt for bugs.

These programs aren't necessarily easy to hack, Backdrop CMS for example is a widely used CMS that has no CVE assignment program on GitHub but they still assign CVEs outside GitHub.

As you know GitHub topics are mostly made up of English words, but what if you wanted to find Chinese content management systems?

In that case, you can use the search as such:

<https://github.com/search?q=%E5%86%85%E5%AE%B9%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F&type=repositories>

I have done this previously to find country & industry-specific targets.

GitHub Organizations

GitHub organizations are like pages; these are where you can find dozens of repositories by one organization or company.

For example, the Apache Software Foundation has their own CNA so if you want to find all open-source software developed by Apache, you can look up their GitHub organization:

<https://github.com/orgs/apache/repositories>

Buildwith

Buildwith offers tools that can help you to identify what kind of software is used on millions of websites out there.

If you want to find content management systems that are used by real websites, here is how you can do it:

<https://trends.builtwith.com/cms>

If you want to find only open-source content management systems, here is how you can do it:

<https://trends.builtwith.com/cms/open-source>

To see what content management systems are used in a country, click here:

<https://trends.builtwith.com/cms/open-source/country/Germany>

DockerHub

Perhaps one of the easiest ways to find a target is by using DockerHub.

DockerHub is a cloud-based repository for Docker container images, essentially a library where developers can share and distribute pre-built applications, services, and environments.

It allows you to pull and run complex web apps or services with a single command, without worrying about installing dependencies, configuring databases, or managing runtimes, everything comes packaged inside the container.

For CVE hunting, DockerHub is extremely convenient because it provides ready-to-use instances of popular software, frameworks, and CMS platforms, which you can deploy locally or in a lab to test for vulnerabilities.

What to do after finding a target?

Once you find a target, depending on where it is, it might take you anywhere between 5 minutes to 5 hours to install and run it. If your target is on DockerHub, it takes just a command, if it's on GitHub and comes with `Dockerfile` or `docker-compose.yaml` files, it takes 5-10 minutes to download and run.

Not all software has Docker, for example when I was testing Fuel CMS, I had to install it and ask ChatGPT to help me out with all the errors. When I see people asking me how to install something without asking ChatGPT first, I usually assume that they are lazy and aren't putting enough effort. I have achieved world peace with AI, there is nothing AI can't do.

How to determine impact of a target?

Determining impact means understanding what real-world consequences would occur if someone exploited the vulnerability you've found. In other words, you're asking: *Who or what would actually be affected, and how serious would it be?*

For example, consider a vulnerability in [Stirling-PDF](#), which is typically a self-hosted application. Because it's rarely exposed directly to the internet, exploiting it in practice would be difficult, there just aren't many accessible targets. Even if you discovered an unauthenticated remote code execution (RCE) bug, it wouldn't spark a global cybersecurity crisis because very few systems would be at risk.

This is why impact matters so much. Vulnerability research isn't just about finding bugs for the sake of it, it's about identifying issues that could genuinely help secure people, systems, and infrastructure. If a vulnerability can't realistically be exploited against real targets, then its value, at least from an attacker's perspective, is low.

In short, determining impact helps you understand whether your work is truly making people safer, or just checking boxes.

Builtwith

Using BuiltWith to determine impact of a target is the best way to see how many targets finding a vulnerability on a software can impact.

For example, to check usage for a headless CMS called Strape, here is what you can search for:

<https://trends.builtwith.com/cms/Strapi>

You can also find websites that are running Strapi:

<https://trends.builtwith.com/websitelist/Strapi>

GitHub repository

If your target has a GitHub repository, you can figure out how many people are using it based on the amount of forks. This is not guaranteed because most developers use bots to amplify stars and forks on GitHub repositories.

Stars and forks usually imply that a repository is widely liked and used, it's basically the way that projects get noticed, but it can also make it difficult to properly analyze whether the stars and forks are genuine or amplified.

Regardless, when a project has 9,000 forks, it's safe to assume that at least 4,500 of those forks are by real people. Here is example of a repository with over 9000 forks:

<https://github.com/frappe/erpnext>

Product's installation tracker

Some software vendors include a beacon that calls back to an API each time their product is installed. However, this is one of the least reliable methods for assessing impact, because vendor-reported data cannot always be trusted. Many developers exaggerate customer numbers or inflate download statistics to make their products look more popular than they actually are.

Take Open Journal Systems (OJS) as an example, they do provide install tracking here: [OJS Usage Data](#). But relying on this methodology forces security researchers to take the vendor at their word, which is often easier said than done. Our team has had multiple vulnerabilities reported to OJS disputed, and we remain skeptical about the claimed user base.

But yeah, this is a common practice and sometimes you can trust vendors especially if they provide proper data and have a transparent vulnerability disclosure program.

Ask them

Another way to find out how many times the software is installed is to just ask the vendors.

Training resources

Before you worry about going bankrupt just to get into vulnerability research, let me assure you that you don't need fancy hardware or software to get started. In this section, I will walk you through free and paid resources that you can use to train yourself.

You may ask, what should you learn first?

This is often something a lot of people are confused about, I want to say that while you may succeed in finding one or two vulnerabilities by just watching a tutorial, it takes a lot of time to find unique vulnerabilities that haven't been found before. For example, the following is Apache JSPWiki's documentation:

<https://jspwiki-wiki.apache.org/Wiki.jsp?page=Image>

I had found an [XSS vulnerability on JSPWiki](#) by reading this documentation and without reading it, I wouldn't have been able to find it. Here is the payload I used for showing a SVG image that contained a malicious payload:

```
[{Image
src='data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC
9zdmciIHdpZHRoPSIyMDAiIGhlaWwdodD0iMjAwIiBvbmxvYWQ9ImFsZXJ0KCd4c3MnKSI+PHJlY3Q
gd2lkdGg9IjIwMCIgaGVpZ2h0PSIyMDAiIGZpbGw9ImxpZ2h0Ymx1ZSIgLz48L3N2Zz4='
caption='Testing Image' style='font-size: 120%; color: green;'}]
```

Now you may ask, how did I do that? Did I pay for some expensive mentorship or was it just my curiosity?

Truth is, it was free training and curiosity because my focus was on learning how to use the web application not exploit it, once I learned how to insert an image using the syntax laid in the documentation, I thought why not try XSS? I used ChatGPT and fed it the whole documentation and that's how I learned how to exploit it.

Again, what should you learn first?

Learn how to use the applications and then learn how to exploit them. There is no way that you can learn about ALL types of vulnerabilities in existence and then go out there with a cheat sheet and drop zerodays. That's not how this field works.

How much time will it take?

It will take anywhere between a few weeks to a few years depending on your exposure and experience in technology. I have seen people who claim to have learned everything about SSRF in just a weekend, but this surface-level understanding is exactly what separates script kiddies from legitimate security researchers.

True mastery of web application security vulnerabilities requires deep technical comprehension of underlying protocols, server architectures, and attack vectors that simply cannot be absorbed through a few YouTube tutorials or basic walkthrough guides. Understanding SSRF, for example, means grasping HTTP/HTTPS protocols, DNS resolution, network segmentation, cloud metadata services, URL parsing differences across languages, bypass techniques for various filters, and how different application frameworks handle requests - knowledge that takes months of hands-on research, experimentation, and real-world testing to internalize.

The difference between someone who has "learned" a vulnerability class in a weekend versus someone who has genuinely mastered it becomes apparent when they encounter non-standard implementations, complex filtering mechanisms, or need to chain multiple vulnerabilities together. Real expertise comes from understanding not just what works, but why it works, when it fails, and how to adapt your approach across different technologies and environments.

If you're serious about web application security, invest the time to truly understand each vulnerability class rather than rushing through checklists, because shallow knowledge will only get you so far before you hit walls that require genuine technical depth to overcome.

Free resources

The following resources are available for free.

PortSwigger Academy

PortSwigger Academy is a free online learning platform offering interactive, hands-on labs and comprehensive content to teach web security and Burp Suite usage. Developed by the creators of Burp Suite, it features learning paths structured from beginner to expert levels,

covering topics like SQL injection, Cross-Site Scripting (XSS), and Server-Side Request Forgery (SSRF).

Link: <https://portswigger.net/web-security/all-labs>

Flask tutorials by Miguel Grinberg

If you want to learn how to exploit web applications, you must learn how to create them. Learning web development usually takes a few weeks to a few months, it's not mandatory to learn but people with a better understanding of programming usually understand how to install and setup web apps. Because most of the times, you won't have the luxury of running apps with Docker.

Link: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>

Hamy's CVE Hunting video demonstration

0xHamy has been credited with over 5 CVEs on Vvweb. Vvweb is a CMS that you can setup to test your skills in a controlled environment, this does require basic knowledge of web applications though.

Link: <https://www.youtube.com/watch?v=QYEcVZ4uNRk>

OverTheWire – Natas

Over the Wire – Wargames is one of the oldest hacking platforms that I know of, it's where I first started learning about Linux.

They have various tracks for learning different skills, Bandit is focused on Linux and common networking tools, Natas is focused on server-side web hacking. There are a lot of challenges in Natas that focus on PHP source code review.

Link: <https://overthewire.org/wargames/natas/>

Paid resources

The following resources are available for purchase; they are very affordable by North American standards.

HackTheBox subscription

A HackTheBox subscription allows you to enroll into tracks that are designed for beginners or advanced users who want to learn certain skills. A subscription usually costs \$10-\$20 and lasts for a month.

Link: <https://www.hackthebox.com/>

Offensive Bug bounty hunter 2.0

I learned bug hunting from the first version of this course. The course is taught by a real bug hunter so what you are going to learn is based on practical techniques.

Link: <https://www.udemy.com/course/offensive-bug-bounty-hunter-20/?couponCode=MT180825A>

Where should you start?

You have got the resources but where do you start?

I am sure you have opened at least one link from the resources I mentioned but you have probably come across terms and concepts that you have never heard of in your entire life. I understand what that feels like because I have been there.

Back in my day, you'd have to spend hours upon hours finding the right blogs that would fill missing pieces of the puzzle, today however, it's much easier but costs a little money.

One of our members, Xoriath, would recommend the following resource for total beginners:

TryHackMe Roadmap: <https://tryhackme.com/hacktivities?tab=roadmap>

You just need a course to hold your hand at first and once you establish a strong foundation, you can move into learning about specific vulnerability classes.

Note

If you want to learn this the way I did, you'll need an exceptional amount of patience. My skills didn't develop overnight. I've been relentlessly committed to the craft, almost to the point of obsession and I doubt most people today would willingly take the same path.

In an era obsessed with instant results, few are willing to put in years of hard, unglamorous work. That's why I now offer condensed courses and curated resources: not because the field is easier, but because most people simply don't have the discipline to grind through it the long way. This work is serious, and there are no shortcuts that matter.

Tools

The following are some of the free tools you can use. I have been credited with 18 CVEs using just Burp Suite Community Edition, GDB, and Firefox. These free tools are more than sufficient for identifying many vulnerabilities and getting hands-on experience with web applications and binaries.

- **Burp Suite Community Edition** – A web vulnerability scanner and proxy tool widely used for testing web applications.
- **Caido Basic** – A network and vulnerability scanning tool useful for reconnaissance and automated checks.
- **GDB** – The GNU Debugger, essential for analyzing binaries, understanding crashes, and performing low-level exploitation.
- **Firefox Browser** – A reliable browser for testing web applications, extensions, and observing client-side behavior.
- **Python** – A programming language for writing scripts, exploits and tools.

These tools provide a solid starting point for security research and CVE discovery without requiring paid software.

Bug hunting tips

Below are bug hunting tips from Delta Obscura members in their own words.

Hamed Kohi (0xHamy)

Tip #1

If you are looking to find vulnerabilities, don't test a target by looking for XXE everywhere, instead of looking for specific vulnerabilities, look for specific features.

Let me explain, in my experience websites that allow file upload by URL are usually vulnerable to SSRF because you can use internal URLs. Now if you want to find the same bug everywhere else, investigate what other websites have upload by URL functionality.

Similarly, when reading bug hunting reports, look for the functionalities that were targeted and compare them to your target to find similarities to exploit. Alternatively, you can use GitHub to look for software that may have the same functionalities as the ones exploited by others. Github allows searching for code patterns so use that to your advantage.

Report writing

To write your reports, we recommend following a consistent template that's compatible with MITRE and VulDB. Both of those site forms don't allow you to submit images or videos as part of your reports.

Historically, most proof of concepts have been written in text not word or Markdown document or videos. Nowadays, different platforms have different report writing procedures.

GitHub's vulnerability reporting system for example can sometimes have placeholder for writing a report but we recommend the following format:

- **Description**
Describe the vulnerability and the software and version it affects. Use only one paragraph.
- **Reproduction steps**
Use numbered steps for reproducing the vulnerability; if you have to use images, upload them to imgur and link them.
- **Exploit (usually optional)**
Always upload exploits to GitHub as private gists.
- **Impact**
Describe impact with a paragraph or numbered list.
- **Mitigation**
Describe Mitigation with a paragraph or numbered list.

Those are all headings with text under them. I personally use the following pattern:

<https://hkohi.ca/vulnerability/8>

CVE Numbering Authorities (CNAs)

After you discover a vulnerability and report it to the vendor, you can typically approach a CNA (CVE Numbering Authority) after **7–14 days** if you haven't received a response. CNAs usually contact the vendor directly, either to coordinate a fix or to confirm whether the issue qualifies as a vulnerability.

In most cases, **reporting to a CNA is the most reliable way to obtain a CVE**. Vendors may or may not assist you in getting one, and while some non-CNA vendors are willing to help with CVE assignments, there's no guarantee.

MITRE

MITRE is the organization that manages the CVE program itself. Their process is a bit old-school, largely email-based and less automated but they are the authoritative source for CVE assignments. If no other CNA fits, reporting directly to MITRE ensures your vulnerability gets recorded, although the process may take longer.

Link: <https://cveform.mitre.org/>

VulDB

VulDB provides a modern web-based interface for vulnerability reporting and tracking. In addition to assigning CVEs, it rewards researchers with points that increase their ranking in the community, providing recognition for your work. The process is generally more transparent and easier to monitor than traditional email-only systems.

Link: <https://vuldb.com/?id.add>

Trend Micro's ZDI

ZDI is a well-known vulnerability program that not only assigns CVEs but also offers monetary bounties for high-impact zero-day vulnerabilities. If your bug is significant, ZDI may purchase it from you, handle vendor coordination, and ensure it is disclosed responsibly while you receive credit, and possibly a payout.

Link: <https://www.zerodayinitiative.com/>

GHSA

GitHub has its own vulnerability reporting and tracking system known as GitHub Security Advisory or GHSA. To see how it works, open this [repository](#) and click on “Report a vulnerability” button, this will open up a page where you can report vulnerabilities and keep track of them. Maintainers can use this page to assign you a CVE with click of a button.

This is often the fastest way to get CVEs through GitHub but not all repositories have a GHSA so you’d have to talk to the vendor about creating one. It usually takes a few minutes.

Other CNAs

Some companies like the Apache Software Foundation have their own CVE assignment program, Apache is CNA and has the authority to assign CVEs on their own.

If you find a vulnerability on Apache, you report through their channels, as of right now, that channel is their main security email address.

CVE Alternatives

Ever since the news of MITRE’s budget cuts appeared on April 2025, a lot of companies and countries have been trying to create an alternative vulnerability tracking system that can replace MITRE or become independent of MITRE.

These approaches have resulted in the creation of GCVE which is a European vulnerability tracking system to function as an alternative vulnerability database.

Which one should you use?

It all depends on circumstances, if the target doesn’t have GHSA or its own CVE numbering system, use VulDB. If they have GHSA, use GitHub.

We highly recommend using VulDB over others.

Responsible disclosure

Responsible disclosure is the practice of reporting a security vulnerability to the organization that owns or maintains the affected system, software, or service in a way that gives them enough time to fix the issue before it becomes public.

The goal is to protect users while still ensuring that the vulnerability is eventually documented for transparency and for the security community to learn from it. Responsible disclosure contrasts with *full disclosure* (where details are published immediately) and *coordinated disclosure* (where a third party like CERT helps manage communication).

How the 90-day rule works

The “**90-day rule**” is a widely accepted guideline, popularized by Google Project Zero, for balancing user protection with public accountability:

1. **Day 0 — Vulnerability reported:** A researcher notifies the vendor (usually via email, bug bounty platform, or CERT coordination).
2. **Vendor acknowledgement:** The vendor confirms receipt of the report and begins working on a fix.
3. **90-day countdown starts:** The researcher agrees to keep details private during this period so the vendor can patch.
4. **Public disclosure deadline:** After **90 days**, the vulnerability details are published whether or not a fix is ready.
 - *If the vendor patches earlier*, disclosure may happen sooner.
 - *If the vendor requests more time for good reason* (e.g., complex fixes affecting many users), researchers sometimes grant a short extension, but this is the exception, not the rule.
5. **Why 90 days?** It’s long enough for most organizations to develop and test a patch, but short enough to prevent indefinite delays or cover-ups.

When a vendor doesn’t respond to a vulnerability report after 90 days or responds but doesn’t intend to fix it, you can publicly disclose it.

How to put your skills to the test?

If you have completed this guide and have been able to identify **at least one** vulnerability on your own. You may be eligible for mentorship from Delta Obscura. We will help you find your first CVE.

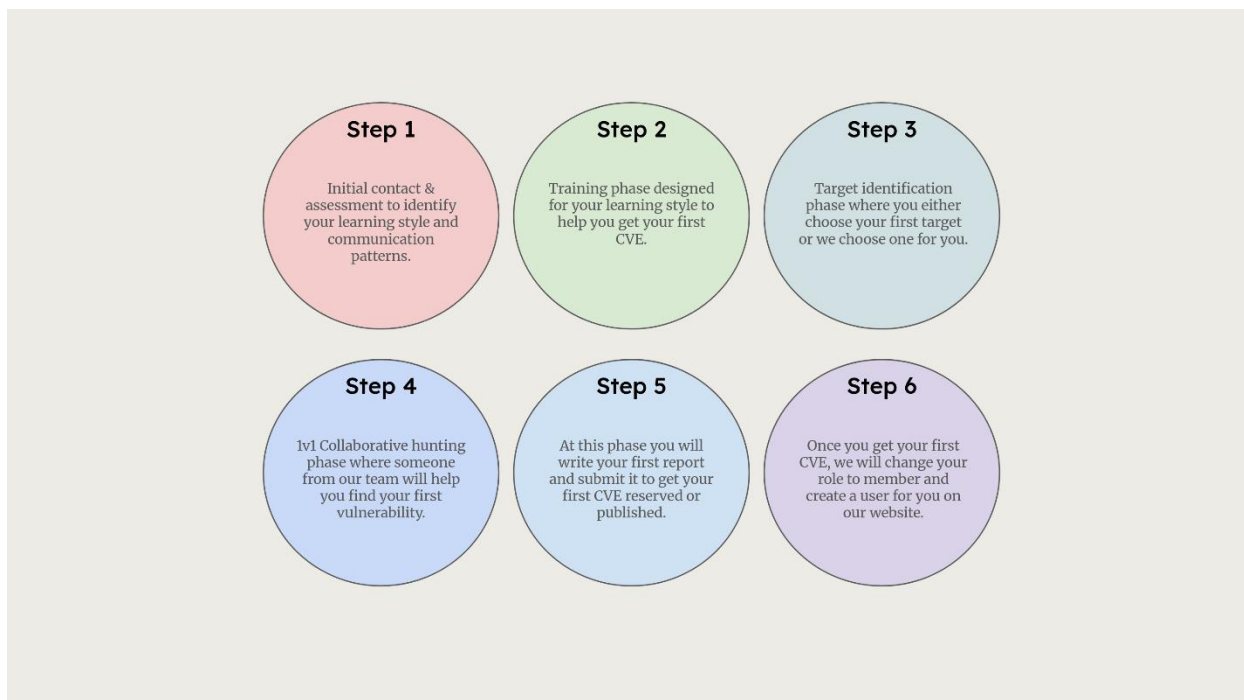
Finding vulnerabilities is just half the battle, your first 0day is the beginning of your career, if you want to take it further and work with us on cybersecurity missions and make an actual impact, consider joining Delta Obscura:

<https://delta.cyberm.ca/join-the-team/>

Our first team mission consisted of 4 contributors from 3 countries, in just a few months, we were able to secure up to 446,000+ digital assets (mostly websites):

<https://delta.cyberm.ca/mission/mission-cyber-sentinel/>

This is how our mentorship program works:



Conclusion

Your Mindset Trumps Everything Else

The biggest takeaway from all of this? How you think matters way more than what tools you own. You don't need fancy scanners or industry buddies to hunt CVEs, just genuine curiosity and the drive to dig into things when they don't make sense. The researchers who actually make a difference aren't walking encyclopedias of exploits. They're the ones who put themselves in an attacker's shoes and refuse to give up when the trail goes cold.

Learn the System Before You Break It

Here's what separates the real finds from the noise: actually understanding what you're testing. Memorizing attack payloads is useless compared to knowing how something should work normally. When you really get how web apps, APIs, IoT gadgets, or binaries are supposed to behave, the weird stuff jumps out at you. That's where the good bugs hide.

Targets Are Literally Everywhere

New researchers always ask "but what do I actually hack?" and honestly, that question answers itself once you start looking. GitHub repos, DockerHub containers, even basic Google searches will give you more targets than you know what to do with. Focus on open-source stuff that lots of people actually use. What seemed like an impossible mountain of random software suddenly becomes a treasure map with clear starting points.

Quality Beats Quantity Every Single Time

Chasing every tiny bug is a waste of time. Learn to spot what actually matters, check if real companies use the software with tools like BuiltWith, figure out if your bug would actually hurt users in the wild. One solid, high-impact CVE is worth infinitely more than a pile of theoretical edge cases nobody cares about.

You Don't Need to Break the Bank

PortSwigger Academy, OverTheWire, Hamy's walkthroughs, tons of excellent training costs absolutely nothing. Sure, HackTheBox subscriptions and paid courses can speed things up, but showing up consistently beats throwing money at the problem every time.

Finding Bugs Is Only Half the Battle

Once you've got something, you need to report it properly. Whether that's through MITRE, VulDB, GitHub Security Advisory, or whoever's handling disclosures for that project, sloppy reporting kills your credibility fast. Do it right, follow responsible disclosure, and people will take your future findings seriously.

No Magic Shortcuts, But the Path Is Clear

CVE hunting isn't a get-rich-quick scheme. It's about building real skills through lots of practice, actually reading documentation instead of skipping it, understanding how things work, and testing your theories. You don't need to be brilliant, just stubborn enough to keep trying until something clicks.