CTF Writeup Guide Structure, Template and Examples

A CTF writeup is a written record of how you solved a capture the flag challenge. A strong one states the goal, shows your reconnaissance, explains each exploitation step with evidence, identifies the root cause and ends with the flag and a lesson learned. Write it while you hack, not afterward.

Every experienced competitor has the same story. They solved a tricky challenge at midnight, felt great and three weeks later could not remember a single step. A wellbuilt CTF writeup fixes that problem permanently. It turns a onetime win into a reusable reference, a portfolio piece and a teaching tool for other people.This guide shows you how to write a CTF writeup from scratch. You will get a repeatable structure, a notetaking workflow, advice on documenting specific vulnerability classes and a list of mistakes that make writeups unreadable. Whether you are posting your first report or polishing your fiftieth, the goal is the same: make the reasoning visible, not just the result.



What a CTF Writeup Is and Is Not

A CTF writeup is an explanation of a solution. It is not a list of commands and it is not a screenshot dump. The difference matters. A list of commands tells the reader what you typed. A

writeup tells the reader why you typed it, what you expected to happen, what actually happened and what you did next.

Think of a writeup as a short investigation report. In a real penetration test, a client does not want a transcript of your terminal. They want findings, evidence, impact and remediation. A good CTF challenge writeup trains that same habit. It also helps you separate lucky guesses from genuine understanding, because you cannot explain a step you did not understand.

Writeups come in several forms. A quick note might be five lines in a private file. A public blog post might run two thousand words with diagrams. A team writeup might be a shared document that captures how three people solved different parts of one challenge. The structure below works for all of them. Only the level of detail changes.

Why Documenting Every Challenge Pays Off

The first benefit is retention. Research on learning consistently shows that explaining something in your own words strengthens memory far more than rereading it. When you write down why a filter bypass worked, you encode the idea in a way that screenshots never will.

The second benefit is pattern recognition. After ten writeups, you notice that many web challenges share the same shapes: a hidden parameter, a trusted header, a predictable identifier, a file path that is not validated. Your own archive becomes a searchable database of techniques. When a new challenge looks unfamiliar, you can scan your notes for a similar symptom.

The third benefit is career value. Hiring managers and security leads cannot easily judge raw scoreboard positions, but they can read a clear report in five minutes. A cybersecurity CTF writeup shows that you can reason about systems, communicate technical findings and work methodically. Those skills transfer directly to consulting, red teaming and bug bounty work.The fourth benefit is community. Public writeups, published after an event ends, help beginners who are stuck on the same problem you once faced. Sharing your reasoning is one of the most respected contributions you can make.

The SevenPart Writeup Structure

Consistency makes writeups faster to produce and easier to read. Use the same skeleton every time and you will never stare at a blank page.

Section	Purpose	Typical length
Overview	Name, category, difficulty, points, goal	3–5 lines
Reconnaissance	What you learned before touching anything	100–200 words

Enumeration	Hidden routes, parameters, technologies	150–300 words
Vulnerability discovery	The weakness and how you spotted it	150–300 words
Exploitation steps	Numbered actions with evidence	200–500 words
Root cause and fix	Why it happened, how to prevent it	100–250 words
Flag and lessons	Final result and takeaways	50–100 words

Overview

Start with the facts a reader needs to orient themselves: the challenge name, the event or platform, the category and the objective. If the challenge description contained hints, quote the relevant line. Keep this section short. Its job is context, not storytelling.

Reconnaissance and Enumeration

Many writeups skip straight to the exploit and that is the biggest reason they teach so little. Reconnaissance is where real skill lives. Record what you saw in the page source, the response headers, the technology fingerprints and the application behavior. Then separate passive observations from active probing, because that distinction shows disciplined methodology.

Vulnerability Discovery

Explain the moment of suspicion. What made you believe the application was weak? Maybe an error message leaked a query fragment. Maybe an identifier in the URL incremented by one. Maybe a parameter reflects your input without encoding. This section is the intellectual core of the report.

Exploitation Steps

Number each action. For every step, include the input, the observed result and a onesentence explanation. Evidence beats claims, so add a trimmed request, a response snippet, or a screenshot where it helps.

Root Cause and Remediation

Close the loop by explaining why the flaw existed. This is what separates a CTF exploit writeup from a simple answer key. If the challenge concatenated user input into a query, say so and describe the fix in plain language.

Flag and Lessons

State the flag only if the event rules allow it, then write two or three sentences about what you would do differently next time.

Capturing Notes While You Hack

The best writeups are assembled from notes taken in real time. Trying to reconstruct a solution from memory produces vague, tidy narratives that skip every dead end. Dead ends are valuable. They show how you think.

Use a simple timestamped log. A plain text file works, as does any notetaking tool you like. Record each command or request, the result and a short reaction. Something like tried a single quote in the id parameter, got a database error, suspect string concatenation is perfect.For web challenges, an intercepting proxy does much of the recording for you. Learning what Burp Suite is and how its history, repeater and comparer tools work will save you hours, because every request you send is already logged and can be exported later as evidence.

Take screenshots as you go, but crop them. A fullscreen capture with browser tabs, notifications and unrelated windows distracts the reader. Capture only the request, the response, or the page state that proves your point. Name your files with step numbers so you never have to guess their order.Finally, save your working payloads. If an exploit took twelve attempts to refine, store the final version in a snippets file. In the future you will be grateful.

Writing the Reconnaissance and Enumeration Sections

A recon section should read like a briefing. Start with what the application appears to be a login portal, a notetaking tool, a file viewer, an admin dashboard. Mention the technology clues, such as framework error pages, cookie names, or static file paths.

Then describe enumeration. Which directories did you find? Which parameters did the application accept? Did you review robots.txt, a sitemap, JavaScript files, or API documentation? Listing these items shows the reader that your next move was based on evidence.A useful trick is to write one sentence per finding in this form: observation, interpretation, next action. For example The session cookie is a base64 string. It decodes to a JSON object containing a role field. Next I will test whether the server trusts it. That rhythm keeps the reader inside your reasoning and turns a dry list into a story.Avoid dumping full

scanner output. Summarize the interesting lines and link or attach the rest if the reader needs it. A hundred lines of port scan results hide the one line that mattered.

Documenting CrossSite Scripting Challenges

Reflected and stored XSS challenges are popular because they teach how browsers parse untrusted data. When you write one up, explain the context first. Is your input placed in HTML body text, inside an attribute, inside a script block, or in a URL? The context decides which payload works.



Show the failed attempts briefly. If a basic script tag was blocked and an event handler worked, say so and explain the filter you believe was in place. Readers learn more from the bypass logic than from the final payload.Then describe the delivery. In a stored challenge, where did the payload persist and who triggered it? In a challenge with a simulated administrator, how did you prove execution and how did you capture the flag? Handson practice is the fastest way to build this intuition and the crosssite scripting labs give you safe targets where you can write your own reports.End the section with the defensive lesson context aware output encoding, a strict content security policy and input validation as a secondary layer.

Documenting Authentication Bypass and IDOR Challenges

Access control challenges are easy to solve and hard to explain well. The exploit is often a single changed number, so the writeup must carry its weight in reasoning.

Describe the legitimate flow first. Show what a normal user sees, which identifiers appear in requests and how the server responds. Then show the modification of the altered identifier, the missing authorization check, or the forged token. Include both the original and modified requests side by side so the difference is obvious.

Explain impact in business terms. Reading another user's record is not just a flag; it is a privacy breach. If you were able to modify data or escalate to an administrator role, say that clearly. For deeper practice with these patterns, work through the IDOR and broken access control labs and document each one as a mini report.The remediation note should mention server side authorization on every object access, indirect references where sensible and automated tests that verify one user cannot reach another user's data.

Going Beyond the Exploit Showing the Root Cause

Most public writeups stop when the flag appears. The best ones continue, because the root cause is where the real learning lives.

If the challenge provides source code, walk through the vulnerable lines. Point to the exact place where untrusted data crosses a trust boundary without validation. Then show a corrected version. Practicing this habit sharpens your ability to review code and the secure code review guide offers a practical process you can apply when a challenge ships its source.Even when no source is available, reason about the likely implementation. Say "the server probably builds the query by string concatenation" and explain the evidence. A reasoned hypothesis is far better than silence.You can also compare your manual findings with tooling. Reading about automated code review helps you decide which flaws a scanner would catch and which ones, like business logic errors, need a human eye. Adding a short would a tool find this? note makes your writeup stand out.

Screenshots, Payloads and Spoiler Etiquette

Presentation matters. Use code blocks for requests, responses and payloads so formatting is preserved. Highlight the important line rather than pasting an entire response. Keep paragraphs short, because technical readers scan before they commit to reading.

Spoilers deserve thought. Most events ask participants not to publish solutions until the competition ends. Respect those rules. If you are writing about an active challenge, keep the report private or use a spoiler warning and withhold the final flag. For retired challenges and practice platforms, public sharing is usually welcome.Never include real credentials, tokens, or personal data in a report, even from a lab. Replace them with placeholders. This is a good habit that prepares you for client work, where redaction is mandatory.

Where to Practice and Build Your Archive

You cannot write good reports without good challenges. Look for a platform that offers realistic, legal targets, graduated difficulty and clear objectives. A dedicated web security CTF lab is ideal, because each solved challenge becomes a readymade writeup subject.Broaden your range with end to end application targets. The web application hacking labs cover reconnaissance, injection, access control and logic flaws, which gives you a diverse set of reports to publish and compare over time.Aim for one solved challenge and one finished writeup per week. After three months you will have a dozen reports and your technique will have improved visibly.

Common Writeup Mistakes to Avoid

Skipping the failures. Readers need to see dead ends because they reveal your thinking.Pasting raw output. Always trim to the relevant lines and add commentary.Using vague verbs. I exploited the login and said nothing. I replaced the username field with a boolean condition that made the query always true says everything.Forgetting remediation. A report without a fix teaches only attack, which is half the discipline.Overloading with tools. Mentioning ten tools when two were used pads the report and confuses beginners.Writing for yourself only. Imagine a reader who knows the basics but has never seen this challenge. Define terms on first use.

Conclusion

Writing a CTF writeup is one of the highest return habits in cybersecurity learning. It forces you to understand each step, builds a personal reference library and creates proof of skill that employers can read in minutes. Use the sevenpart structure, take notes while you work, explain root causes and always respect event rules about spoilers. Start small. Pick one solved challenge today, open a blank file and fill in the skeleton from this guide. Repeat that process weekly and your reports will become sharper, faster and more useful to everyone who reads them.If you are building a long term learning path, start from the AppSecMaster home page and choose a track that matches your level.

Frequently Asked Questions (FAQs)

What is a CTF writeup?

A CTF writeup is a written explanation of how a capture the flag challenge was solved. It describes the reconnaissance, the vulnerability, the exploitation steps and the final flag, usually with screenshots or code snippets as evidence. It serves as a learning record and a portfolio piece.

How long should a CTF writeup be?

Length depends on difficulty. An easy challenge might need 300 to 500 words, while a hard multistage problem can justify 2,000 or more. Focus on clarity and completeness rather than a target word count and keep every section purposeful.

Should I publish my CTF writeups publicly?

Publishing is encouraged once the event has ended and the rules allow it. Public reports help others learn and build your reputation. For active competitions, keep your work private or wait until organizers permit sharing.

What should a good CTF writeup include?

Include the challenge overview, reconnaissance findings, enumeration results, how you discovered the vulnerability, numbered exploitation steps with evidence, the root cause, a remediation suggestion and the flag with key lessons learned.

What is the difference between a CTF writeup and a CTF walkthrough?

The terms overlap heavily. A walkthrough usually emphasizes step by step instructions for reproducing a solution, while a writeup also stresses reasoning, analysis and lessons. Many authors use them interchangeably, but the best documents combine both.