White hatting a protocol means exploiting a vulnerability in a smart contract to save funds, rather than steal them. Engineers do this to catch vulnerabilities earlier and stop bad guys from attacking a given protocol.

Here are some best practices on how to do this.

What not to do

If you ever see a vulnerability in live code, you should not exploit it.

Even if you have the best intentions, we've seen even the smartest people mess up an attack, losing even more money than if a black hat exploited it. Even if you exploit the contract perfectly, there could be legal ramifications.

Often, people try to "gray hat" by doing the exploit and promising to give the money back in exchange for a reward. This is called "blackmail" or "holding funds ransom" and is not the actions of a white hat. Just by holding funds and starting to negate a ransom (even if you call it a "reward"), you are now a black hat.

There are two exceptions:

- A malicious exploit that is already in the mempool.
- There is a safe harbor agreement.

We will talk about those at the end of the article.

To summarize, if you see a tasty bug in production and you feel a strong sense of urgency, refrain from attacking it.

Why?

1. We've seen many white hat attempts that have failed because:
2. Get front run
3. Fail a transaction that exposes the bug to black hats
4. There can be legal repercussions.

— The OWASP group has a vulnerability disclosure cheat sheet that all security researchers should also be familiar with.

What to do if you do find a live vulnerability

1. Get in contact with the protocol, bug bounty platform, or whoever is responsible for owning the codebase.
2. Make sure you're communicating on a secure line.
3. Prove the bug exists.
4. Come up with a fix.
5. Roll out a post-mortem.

The steps can get quite subjective and hairy depending on the circumstances, as each bug is often unique with its own set of issues.

To help with this, we spoke withMichael Lewellen from Openzeppelin, who has been through several live vulnerability mitigation crisis including:

- Compound
- Thirdweb
- and many more!

Leaning on his experience, this has helped give us a list of actions one needs to take and considerations depending on the circumstances.

The Steps

You found a bug in a live smart contract. The clock is now ticking.

These are the steps and considerations to take.

Step 1: Find those responsible for the codebase

You can often find those responsible in one of the following places:

- Project documentation
- Bug bounty platforms
- security

tab on their code repository (like GitHub or GitLab)

- In the smart contract itself

Adding an SOS email to a smart contract is a simple but often overlooked step. Once you've found the correct individuals, you can move on to step 2.

You can also optionally reach out to "smart contract firefighters" likeSEAL 911, who often have the context and contacts to connect you with the right people.

Step 2: Make sure you're on a secure communication channel

You're about to share an issue that could potentially be exploited for millions of dollars. You need to make sure you're on a secure communication channel.

Some good options are:

- A secret end-to-end encryption channel on Telegram or Signal
- A bug bounty platform's secret messenger channel

Once you're in this channel, ensure that only the people who need to be there are there. The more people learn about the bug, the more chance that the bug gets leaked.

Step 3: Verify the bug

Once you're as sure as you can be that you're speaking on a secure channel, this is when you tell them about the bug.

Disclosing the bug is going to be the same process as doing an audit report writeup, with extra care on having a working proof of code.

Nothing can be done if the protocol can't verify the bug exists. It's incredibly important for the reporter to ensure the exploit works, so this isn't a huge waste of everyone's time. Using a fork test (on foundry, hardhat, titanoboa, etc.) will be one of the best ways to prove the bug exists.

You can see a great example of a working fork test from the famous Euler hack inCiara's codebase. The key point is that, directly from the test, she picks a live RPC URL to connect to and forks the chain with this line:

vm.createSelectFork("eth", 16817995);

You can also optionally just run forge test --fork-url RPC_URL_HERE

if you're using Foundry.

The key part here, is to make sure the exploit and the proof of code/proof of exploit works, so everyone can be on the same page with what the issue is.

Making sure everyone understands the issue well will help move to the next and most important phase: the fix.

Step 4: Come up with a fix

This is the hardest part of the entire process. Good news is that you're communicating with the protocol to devise a solution together.

There are going to be a lot of considerations to take into account:

1. Is there a governance window to do an upgrade?
2. Should we just redeploy?
3. When do we alert people of the incident?

This is where your creativity is needed. How do you patch the protocol so that as few users are affected? Time is ticking, the pressure is mounting, but you must stay stay calm and collected. A botched fix can result in even more damage being done.

There are several considerations when it comes to implementing the fix:

1. Use an MEV-resistant RPC_URL
2. Batch all your transactions together, using something like awhitehat kit, so others don't see the exploit ahead of time
3. Work through a plan on what to do if this goes poorly

Step 5: Do a post-mortem

A post-mortem is when you recount the event in a public letter so the entire community can learn from the event that took place

.

Not doing a post-mortem is a bad reflection on you, and it will make it so web3 doesn't improve.

The industry needs to learn:

1. What the issue was
2. How it was introduced
3. How to fix it

Often, protocols will attempt to shift blame in their post-mortems. Don't do that. Just give the facts so we can all learn.

Hopefully, when all this is done, the protocol will reward you for helping them find the bug. Usually, a 10% TVL reward with a $5M cap is the "rule of thumb", however, there is no hard and fast rule.

But… What if?

The above steps are the ideal process of dealing with a live vulnerability, but considerations often make following this path difficult.

They don't have a security contact or bug bounty

If there is really no one responsible for the code (this is web3 after all), you may have to announce and give people a long window of opportunity for them to leave the protocol before announcing the issue. This should be a last last last resort.

The protocol ignores the bug

Give them a window to fix or acknowledge it, otherwise tell them you'll need to go public with the information. Give people the chance to leave the protocol before you publicly disclose the issue. Use this as a last resort!

They don't pay you for your work?

This is a difficult one. You can always blast them on Twitter, but that can backfire. Ideally, everyone works together. A protocol that doesn't pay their white hats will mean fewer white hats working on their protocol.

And that can be a dangerous precedent to set.

The transaction is already in the mempool

This is one of the few times when doing an exploit might

be acceptable. If you see a malicious transaction in the mempool, you know that the exploit is imminent. In this case, it might

be okay to front-run the attack and exploit the protocol first. This can still be legal ramifications unless the protocol has a safe harbor agreement.

A Safe Harbor is a framework in which protocols can offer legal protection to whitehats who aid in the recovery of assets during an active exploit. The Security Alliance (SEAL) has a minimal example for protocols that want to make sure white hats can front-run malicious transactions without fear of legal repercussions.

These will often include an address/location for white hats to send funds to immediately after an exploit has occurred.

The important consideration for this agreement, though, is that they only apply to active

attacks. For example, a malicious transaction is only in the mempool. If a security researcher goes out and attempts to initiate an attack, the safe harbor is void.

Summary

If you take anything away from this article, take these away:

1. Exploiting a protocol, regardless of intentions, is the wrong initial step
2. Reach out to protocols if you find an active bug
3. The only exception to rule #1 might be if:
4. Their is a malicious transaction already in the mempool
5. Their is a safe harbor agreement in place

Happy ethical hacking!

-- If you'd like to learn smart contract auditing, make sure to check outCyfrin Updraft - the ultimate learning tool for smart contract development.

Otherwise, if you're ready to start white hacking, check outCodeHawks, top competitive auditing platform granting auditors rewards when findings vulnerabilities in open contests.