I’m Alex The Entreprenerd, I’ve been developing on the EVM since 2019, having worked with BadgerDAO in shipping eBTC, Building multiple Yield Strategies, one of which was used in production by Yearn, as well as multiple small Open Source Contributions.

I’ve been Squashing bugs at Code4rena, Spearbit, and Sherlock, and have found bugs in Velodrome V2, Optimism, Tapioca, Ethos, Chainlink, Pool Together and many more.

I’m writing to share the work we’ve been doing at Recon, with the hope that this helps projects ship safer, more thoroughly reviewed code by making Invariant Testing accessible to all developers.

Introducing Recon

Recon helps you write and run invariant tests.

We offer a free Invariant Test Scaffolding Tool, to help developers build their invariant tests quickly.

Build your Handlers, free for all Open Source Foundry Repos

[

](https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Fda2a9b22-83ae-4968-9dc2-a23b9e010dd0_3020x1484.png)

We also offer a paid service to run invariant tests in the cloud which we call Recon Pro.

Which we’ve been offering to projects like eBTC and Centrifuge to help strengthen their security.

Recon Pro Shareable Job Page

[

](https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F7249bc25-9e68-44fd-a40d-b3ab85c921d9_3014x1542.png)

Why

Due to my work, I have reverse-engineered the vast majority of exploits that have happened over the past few years.

I’m going to tell two personal stories about a couple of projects that got exploited, and why this led me to build Recon.

ELASTIC SWAP

Back in 2022, donation attacks were somewhat of a novel vector.

The Elasticswap codebase was reviewed by Code4rena, an org I’ve been working with since 2021.

Vulnerable Lines- Credit: Decoding Elastic Swap’s $854K Exploit | by QuillAudits Team | Medium | Medium

[

](https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Fbfee4228-bc77-4532-9482-d5d94ab2133d_980x606.png)

An attacker found out that by donating some tokens, they were able to convince the AMM to give away more value than intended, leading to a loss of $850k.

A simple donation attack lead to this.

How is it possible that nobody tried this?

Was there a way to automate this?

EULER

Euler V1 went through multiple security reviews and brought forth a more mature way to discuss risk when it comes to TWAP oracles, as well as with isolated lending.

Many of my colleagues were using Euler, and many of my peers did Security Reviews of Euler either directly or as part of Due diligence for integrations.

Famously, Euler was exploited due to a small change, that was added as an afterthought.

New function that introduced the Euler Exploit - Credit:[Decoding Euler Finance's $197 Million Exploit | QuillAudits | Coinmonks]

[

](https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F13f5bbec-3473-43cc-8899-eeca84a8fc6c_1290x676.png)

This small change allowed for self-liquidation, which, when combined with a dynamic caller premium, allowed the exploiter to profit from their own liquidations.

Looking back at the details of the exploit, I believed, and still do, that the main issue was a lack of coverage.

The review for this change was a small, short, solo review, instead of what we have come to call a Security Contest, in which tens if not hundreds of professional whitehats all compete to find the most unique bugs.

Not enough eyes went through this change before it was enacted.

While the Exploit looked very complex from the outside, it could have been caught with a single global property: "No Direct Action should make an account Liquidatable".

The check would allow the fuzzer to do what's best, try random stuff, and show you what it was able to produce.

It was from these two incidents that I decided to do something about this

Initially, I wrote my own code generation tool for foundry, which would set some automated calls, and it would then check for an attacker having a gain in value at the end of that sequence.

Example output from codegen tool

[

](https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Fedfc51e5-0db7-45c7-80b7-129bc66b5fce_1082x1008.png)

This was able to find the Elasticswap exploit within a few hours with no adjustments (just brute force) and was able to find the bug within minutes with a small amount of tweaks.

Brute forcing my way into finding the Elasticswap exploit

[

1456×586 231 KB

](https://global.discourse-cdn.com/standard17/uploads/arbitrum1/original/2X/9/9334206974686970e09bd0838c9229208fb3d231.jpeg)

I was never able to reproduce the Euler exploit through this tool, the complexity of generating thousands of lines of poorly optimized boilerplate code was too much for me given that I had a day job.

This changed when I saw Antonio's research on Invariant Testing.

Antonio is a Security Researcher with a background in development and a[passion] [for] [Open] [Source], he has held multiple Invariant Testing Focused Workshops:

- [OpenSense]
- [Spearbit]
- [Defi Security Summit]

From looking at how Invariant Testing worked, it was clear that I was creating a poorly engineered version of Echidna, a [battle tested Open Source Invariant Testing tool from Trail of Bits].

At the time Antonio had created Fuzzy as a way to run fuzzers in the cloud, he also came up with a way to write Invariant Testing to be debuggable in Foundry, meaning you'd get the best of both worlds:

- Fast as possible broken properties
- Easy, comfy debugging with Foundry

From those two key ideas, we built Recon.

Recon has gone through multiple evolutions in the past five months and we are excited to finally share it with you.

All Open Source Foundry Projects can scaffold invariant tests for Free and can run invariant tests locally through our Open Source:

[Create-Chimera-App

](https://github.com/Recon-Fuzz/create-chimera-app/tree/main)

So far Recon has been tried by more than 150 Solidity developers and Security Researchers and we have run more than 800 fuzzing jobs on our cloud platform.

I share this with the hope that we can learn from the mistakes of the past, and with the belief, that by adding invariant testing, we can prevent a vast amount of exploits.

Recon Stats,from the homepage

[

1456×474 79 KB

](https://global.discourse-cdn.com/standard17/uploads/arbitrum1/original/2X/b/bed4214daaa57792b1545b78297d3c9bc5acc81a.jpeg)

If you wish to try Recon you can do so at:

getrecon.xyz

[Recon](getrecon.xyz)

Recon helps you build and run invariant tests

If you have specific questions or would like to work with us, you can reach out to me on twitter:

https://x.com/GalloDaSballo

Over the next months we will continue to work on Recon, with the goal of helping developers write safer code while preventing real world exploits.