Bug Business #1: Inside Logic Flaws with EdOverflow

BY INTIGRITI · FEBRUARY 3, 2020 · LAST UPDATED ON MARCH 6, 2025

Want to read the latest version? Visit the link below:
https://www.intigriti.com/researchers/blog/hacker-spotlight/bug-business-1-inside-logic-flaws-with-edoverflow

Bug Business is a series of interviews in which experts from the bug bounty industry shine their light on bug types and trends. Edwin Foudil — aka EdOverflow — is not only known for his legendary frog profile picture, but also for his active contributions to the bug bounty community and his work on security.txt — a proposed standard which allows websites to define security policies.

In the bug bounty space, Ed is generally known for his unusual and creative bug discoveries, more often than not in the underlying logic of applications and their components. We are more than honoured to have him with us for this interview to learn more about his workflow and thought process.

Welcome, Ed! Should we refer to you as an academic, bug bounty hunter, or a security researcher?

When it comes to the IT field, I would probably refer to myself as a web designer and developer, with a passion for security research and bug bounty hunting.

A lot of people know you for your work in the security field, so it's interesting you describe yourself as a web developer in the first place.

I have a pretty broad, all-round range of interests, cybersecurity being one of them. I mainly got into security and bug bounties through open-source software. My contributions to open-source projects gave me a boost to work on the security aspects of web development and writing secure code.

Could you describe the journey that brought you here?

Oddly enough, everything started with fishing — not to be confused with phishing. The beautiful views at the lake sparked my interest in photography and Photoshop, which got me into web design and web development. I think the actual bridge from web development to security came from my profound curiosity: whenever I build a project I am continuously asking myself whether I am doing the right thing or if someone could break my code. You start reading stories about data breaches in the media and wonder how that could possibly happen. Reading about these attacks got me interested in trying ethical hacking.

So I started testing local Swiss companies that had security contacts. I participated in their vulnerability disclosure programs. After some time, I found other local bug bounty programs and got hooked.

You went from finding bugs in local Swiss companies to co-creating security.txt, a proposed standard which allows websites to define security policies. How did that come along?

Being a hacker, I like identifying and solving problems. In fact, I used to maintain a to-do list with project ideas during high school. Eventually, summer break came along, and I had the chance to go to DEF CON in Las Vegas. There, I had a fantastic time meeting like-minded people, and on my way back, I made a stop in New York, where I met Joel Margolis (@0xteknogeek). We had grilled cheese sandwiches and spent a lot of time chatting about the challenges the security industry was facing. Inspired and back at my hotel room, I grabbed my to-do list and started working on one of the bullet points: "security.txt". I spent the night writing my first specification, published it on GitHub, and then sent it to some friends before going to bed.

> **Introducing security.txt: A standard that allows websites to define security policies.**
> https://t.co/M890w5liOM
>
> **— Ed (@EdOverflow)** August 12, 2017

When I woke up, my phone was going off the rails. Multiple people from the industry had shown their support and wanted security.txt to become a thing. This was when I decided to dedicate the next few months into fleshing out the concept into a formal Internet Draft.

Those 'next few months' turned into three years. How does an idea become an Internet standard?

You can post it to the IETF mailing list which will enable other people to comment on your idea and give constructive criticism. This is a very important part of the standardisation process; you need to be open to all sorts of feedback to make the specification as good as possible. Half-baked ideas cannot become Internet standards.

When we hit the fourth or fifth draft, I went to an IETF meeting in London where I got to present security.txt in a five-minute talk for the IETF SecDispatch working group. At these meetings, you get to explain your idea, why you think it is important, how you would like to see it implemented, and why you think the IETF should adopt your work. Afterwards, they reach a consensus on what should be done with your idea by humming to the most desirable option. For security.txt, the consensus was to assign a document shepherd within the IETF to follow-up on the project. Once the document shepherd thinks the proposal is ready, the "Last Call" stage starts, which is the last stage before RFC approval. Last Call finished on January 7th, so now it gets sent to the RFC editors who will double-check the vocabulary and grammar after which they will send it to the Internet Engineering Steering Group (IESG) who will decide whether it should become an RFC or not.

What a journey! You can't really publish an Internet standard overnight, can you?

Since these things will be set in stone for years, there are a lot of steps involved in the process. I was lucky to have the support of my co-author Yakov Shafranovich (@nightwatchcyber), who was more familiar with the IETF process.

With all that work going into security.txt for the past few years — did you have any time left to do bug bounties?

I do not rely on bug bounties financially, so I only do it in my spare time. Sometimes I spend an hour per week, sometimes a couple more, but only when I feel like it.

How do you still find bugs when so many people are doing bug bounties full time? You only have a few hours to spend and are missing out on hours of recon and information gathering.

I do have an automated setup that gathers information and metadata about bug bounty assets. When I do find time to hack, I perform manual testing as much as possible. This does, of course, include testing

for the more generic issues such as IDORs and XSS, but I primarily enjoy learning about new concepts while hunting. The best thing is that I do not need my phone or laptop to think about new attack vectors. I can ponder about potential logic flaws while on the train, bus, or while laying in bed.

How do you discover logic flaws in applications?

Before you can discover logic flaws, you need to dig deep. You have to understand the application. By that, I do not only mean having a visual overview or understanding of how the components interact but also the code and coding style of the application — especially when there is a lot of client-side code involved. Get a feel for what the developers were doing and what was going through their minds. Why did they design it this way? Why does it do that? Why did they develop the application in the first place? I know from personal experience as a developer that you might be focused on writing something and have not noticed the functionality and interactions that take place elsewhere in the application with the component you are developing. That is where logic flaws usually arise.

How do you get into a developer's brain? How do you get to know a company in a way so that you can see logic flaws they don't?

I usually start off by reading the target's documentation and using their app as a user. I will Google the target and read up about them, refining searches to discover what other users and journalists have to say about the product. You will learn so much more that way than just restricting yourself to the application. I do not have a clear structure to my approach, but I will always end up graphing out the functionalities and the way components interact. I do that manually with a whiteboard or a piece of paper: I need a clear overview in order to be able to understand the application. Then it is like solving a maths problem where some things will stand out. Individual components relying on each other's data are always interesting: follow the data flow and see if there is any way to make components behave in unexpected ways. It takes some practice and experience to spot these things, but eventually, you will start developing a gut instinct for it. For example, the more you learn about how companies process financial transactions, the more implementations and potential mistakes you will notice. It is important to keep track of your thought process and ideas, as it is an ever-evolving learning process, and you might realise the existence of a logic flaw weeks after you tested the application.

If I had to pinpoint one area that is particularly interesting to look for logic flaws in, I would go for integrations. If you see an app incorporating a third party component, go use that third party and learn about how it is integrated. Most integrations are set up with a specific use case in mind, but the developers adding them may not know about the entire scope of functionality the integration enables. It is also very difficult, if not potentially impossible, to write unit tests for these cases: you cannot really test the logic behind it the way I am in an automated fashion, whereas you can easily test sanitisation of the data coming from third parties.

Talking about integrations, your blog article "CI knew there would be bugs here" got nominated for @Portswigger's "best write-up of the year". How did you come up with that?

There is a thing I try to look for which I refer to as a "goldmine": new and lucrative areas of research previously unexplored by bug bounty hunters. I got the idea of looking for bugs in Continuous Integration services from my work on open-source projects. I saw Travis CI everywhere whenever I submitted a pull request, and I knew the logs were public, so I just wondered what kind of problems could arise. Then I discovered some previous research in that area, but I had never seen anyone apply it in the bug bounty context.

How do you keep a goldmine secret? When do you decide to publish?

My intentions are not always necessarily money-driven. In reality, I need to find real-world cases to support my research and inevitably, the more cases I report, the more people will know about it. The only thing you can really do against that is being quick, so if possible, I try to automate it. With the Continuous Integration work, we were a group of five people that reported as much as we could find in a short space of time.

Having your techniques leak is unavoidable. At the end of the day, publishing your work is an opportunity for other people to get involved and for you to improve your tooling. You will always have the advantage of being the first — nobody can take that from you.

What is the best way for bug bounty hunters to learn how to do research and discover these 'goldmines'?

The best tip I could give is to write about your bug bounty hunting experiences. You do not have to be the best writer in the world to spin up a bug bounty blog. Just do it and you will naturally develop a feel for interesting areas to explore. Read other people's research and compare them with yours: could you have found this? What would you have done differently? Follow the industry and see where new technologies are arising. Read up about them and check how they integrate. Find problems, tackle them, and most importantly, write about them. Even if you do not have a solution in mind, you might discover it by structuring your thoughts while writing.

Do you think we could ever automate the detection of logic flaws?

Detecting logic flaws is a hard, if not impossible, process to automate. If a developer cannot write unit tests for these, it seems unlikely that we will see them showing up in tools or scanner results any time soon.

What is the best way for companies to prevent logic flaws?

Since you cannot rely on tooling, you need humans to detect or prevent them early on in the development life cycle. I hope that by publishing more work in this area, people will become more aware of their existence. Not just the developers, but also the decision-makers or business people that unwillingly introduce logic flaws in the early design stages of a product.