The Power of Fuzz Testing to Reduce Security Vulnerabilities
Transcript

Part 1: Why Fuzz Testing?

Julia Allen: Welcome to CERT's podcast series: Security for Business Leaders. The CERT program is part of the Software Engineering Institute, a federally-funded research and development center at Carnegie Mellon University in Pittsburgh, Pennsylvania. You can find out more about us at CERT.org.

Show notes for today's conversation are available at the podcast website.

My name is Julia Allen, I'm a senior researcher at CERT, working on operational resilience and software assurance. Today I'm very pleased to welcome Will Dormann, a member of CERT's Cyber Threat and Vulnerability Analysis Team. Today, Will and I will be discussing a software testing technique that's been gaining some traction. It helps identify programming errors and improve the quality of code. It's called fuzz testing. And it's particularly useful when a developer or a program, software development program, is wanting to identify software security defects and vulnerabilities. So welcome, Will, really glad to have you with us today.

Will Dormann: It's good to be here.

Julia Allen: Okay. So what exactly is fuzz testing? It's not a term that probably typically rolls of the tongue of most listeners. And how does it differ perhaps from more traditional testing approaches that they might be familiar with?

Will Dormann: Sure. Fuzz testing is a way of testing an application in a way that you want to actually break the program.

So traditional software testing is generally -- a software developer writes an application and it's supposed to perform some features. For example, I might write a calculator application. If I'm testing that application, I might want to make sure that it's actually doing the math that I want it to properly. And if I decide that that application is working properly, then that usually is sufficient for passing traditional testing.

The difference with fuzz testing is that the goal of a fuzz tester is you want to provide invalid input to an application. And you really want to cause that application to misbehave. So again, if I'm going back to the example application of a calculator app -- generally, people are going to be adding and doing various math operations on numbers. If I'm a fuzz tester, I might go ahead and try entering in some letters. Or I might just mash on the keyboard a little bit and just see does that application withstand whatever sort of malformed input I give to that application.

If you can do something that causes an application to crash, it can have a number of security impacts. At the very least, if you're running an application -- say I'm using a web browser and I'm looking at some web pages and then all of a sudden the web browser crashes. At the very least, I've got what we refer to as a denial of service condition, that's generally I'm using my application and then all of a sudden it disappears or crashes. I might lose any work that I've been doing with that app. But on the other end of the spectrum, in some cases when an application crashes, it can be something that an attacker might be able to exploit, which means they might be able to end up running code.

So to go the example of the web browser, if I'm visiting a website and the browser crashes, it could do that in an exploitable manner, and I could all of a sudden have malicious software on my machine. So it can be very interesting, it can be very useful seeing what sort of results come out of fuzz testing.

Julia Allen: Okay. So would it be fair to say that in the absence of fuzz testing for this class of software behavior that fuzz testing reveals -- so let's say I didn't do any fuzz testing and as you said, an attacker is able to take advantage of some weakness or some flaw that I didn't find. The fact that they can identify that can actually allow them to inject software, cause the software to misbehave, cause the software to do something that the original developer never intended, right?

Will Dormann: Absolutely. If I'm a popular software vendor and I'm writing some application and that application becomes really popular and a lot of people use it on the internet, if I haven't done any fuzz testing on that application before releasing it, that actually could put people at risk for using that software. Because after that software gets deployed to people on the internet, if some people start banging around on it, run some fuzz testers, they might uncover some flaws that I might not have discovered in my traditional testing as a software vendor.

Julia Allen: Okay. So there's all different kinds of testing techniques. You talked about functional testing, where you make sure a piece of software meets the functional requirements. Typically one of the traditional software testing techniques is errors. You want to check you error reporting routine, your error detection routine, and bounds conditions. But clearly fuzz testing goes to a different class of error. So what are some of the benefits of fuzz testing when you have all these different testing techniques that you need to consider? If you had to rank and stack them, where does fuzz testing fall on the list?

Will Dormann: Anybody that writes software for a computer or for a cell phone or really any software vendor, wants to be producing high quality code. If there's any situation that causes an application to crash, that's something that's not really well received by the end user. And as I mentioned earlier, there could be security impacts of having bugs in your application that might cause a crash.

The thing about fuzz testing is it generally is a low effort sort of testing in that when an application is fuzz tested, depending on how the testing actually occurs, a lot of that can really be automated so that the application is just when it's being fuzz tested, it will receive malformed input from this tester and it'll go through a bunch of different cases. And eventually it might come across a crash. And as the crashes are encountered, the software vendor can go back and see, “okay, here's the programming flaw that caused this crash to occur.” They can fix that. And then just as they're developing the software, they can just perform this testing. And in the end, it'll end up improving the quality of the code. The fewer bugs that you have in an application, the fewer potential vulnerabilities that you have.

Julia Allen: Right. So what you're saying is, you can automate it to a fairly high degree, which means it's repeatable. You can include it as part of your regression testing suite. When you're updating the software you can rerun it through the fuzz test cases because you've got that fairly automated and you can compare the results, right?

Will Dormann: Absolutely. If I am a vendor and I have a fuzz testing framework set up -- let's say I have version 1 off my product and I go through a fuzz testing routine. And I come to a certain level of comfortableness with, or “comftorbility,” with the quality of the code, if I go and I

add more features to that software later on, I will just need to go back or at least continue to do fuzzing so that any new code that gets added to the software, that also gets exercised through fuzz testing as well.

Part 2: Fuzz Testing Techniques; CERT's Dranzer Tool for ActiveX Controls

Julia Allen: Excellent. So how about a few examples of some fuzz testing techniques? You talked about malformed input and various ways that you could break the software. But are there some other techniques that you could educate us about?

Will Dormann: Yeah, sure. There are actually two main categories of fuzz testing. And really, the two ways that a fuzz testing process will get broken down to is either smart testing or dumb fuzz testing. And the terms may sound a little bit odd in that one might sound like a much better method than the other. But it really just describes how the fuzz testing tool is performing the fuzzing on the application.

So with a smart fuzzer, which is also known as a generational fuzzer, what that is doing is you have a tool which is generating input to an application from scratch. So you're really starting from a base level where you have no data but your fuzz tester is smart enough to generate data that's formatted in a way that kind of makes sense for the application that's receiving it.

So just as an example, if you have a PDF viewing application that you want to fuzz test, Adobe Reader or Foxit, or there's a number of applications that can view PDF documents. If I have a smart fuzzer for a PDF document, what that means is that the fuzzing tool that I have actually understands the specification for the PDF file format and it has the ability to create a PDF file from scratch. And when it creates that file, it might mangle some of the values in it. It might, if you have a property of that document that specifies a height or a width, it might use like a really large number for that. Or if I have a string value, it might use a really large string value. Or it would just try to basically take the specification for a particular type of file, for example, and just change that file around in ways that might cause the application to misbehave.

We actually have an example of a smart fuzz testing tool that CERT has released, I believe it was about a year ago, called Dranzer. And it is an ActiveX fuzz testing tool. And it falls in the category of a smart fuzzer in that it fully understands the ActiveX specification and it generates data that an ActiveX control can handle. And that's actually something that is publically available. People can download the Dranzer tool and they can run it on ActiveX controls that they might have on a Windows system.

Julia Allen: You've piqued my curiosity. So what about a dumb fuzzer?

Will Dormann: Sure. The other way that you can do fuzzing is called dumb fuzzing or mutational fuzzing. And the term dumb makes people think that it's not really effective or it's maybe not a good way of testing an application. But it really just has to do with how the fuzzer is generating malformed data. So rather than -- in the case of a smart fuzzer you're starting from scratch and you're generating data based on a specification or some other properties. With a dumb fuzzer, what you do is you actually start out with a validly formed file, for example, if you're doing file fuzzing. And you'll take that file and you'll change bits in that file to corrupt it.

So again, in the example of a PDF document, if I want to fuzz test a PDF viewer -- if I'm going to use the dumb fuzzing technique, what I will do is I will start out with a valid PDF and then my dumb fuzzing tool will modify that file in various ways and look to see does the application handle these modifications okay. Some dumb fuzzers might just randomly change certain bits

in a file. Some might use different patterns that have been shown to uncover bugs more quickly than others. But generally, what you're doing is taking a valid piece of data -- it could be a file, it could be network traffic, it could be anything, any way that you can get data to an application. But you are taking data and mangling it and just seeing how well does the application handle that mangling?

Julia Allen: Okay. Because I mean, essentially what you're trying to is to see how it's going to behave no matter what is thrown at it, including things that are totally unexpected. I remember in my days as a software developer, we used to have the statement “but the software was never designed to do that.”

Will Dormann: Sure.

Julia Allen: The software’s not designed to respond to that condition.

Will Dormann: Yeah.

Julia Allen: And obviously as you've found, we find in the security arena that all kinds of stuff gets thrown at software, right?

Will Dormann: Absolutely. Files can get corrupted inadvertently. Sometimes different applications might produce files that are not quite compatible with each other. So you might not actually be intending to do anything bad or you might not be intending to test the application, but sometimes fuzz testing can happen inadvertently. In the days -- it doesn't really happen too often with modern systems. But in the days of dial-up modems, sometimes you've got line noise on your telephone line. And can both of those systems handle some amount of corruption between the two systems? And ideally, they should be able to handle all sorts of those situations. But that's also a sort of way that fuzz testing occurs -- maybe not intentionally but still, you want an application to be able to handle malformed data.

Part 3: Use Automated Fuzz Testing when Building and Buying Software

Julia Allen: We've talked about a variety ways that fuzz testing can be used. And I know that it's showing up more and more as a solid best practice for developers who are building software to be more secure. In your work, how have you seen fuzz testing reveal or be used to identify some of the most common software vulnerabilities that attackers take advantage of?

Will Dormann: Okay. We actually have seen fuzz testing pretty closely tied to finding security issues. As I had mentioned earlier, when you are fuzz testing an application, generally you're looking for a software crash. And I had mentioned some sort of crashes can be exploitable. Fuzz testing is a way that a lot of vulnerabilities -- it's an easy way for vulnerabilities to be uncovered.

So to go back to the original, or the example that I mentioned earlier with the Dranzer ActiveX fuzz testing tool. This tool is specifically designed to find a couple of very basic vulnerabilities but it still finds them pretty frequently. And two of the vulnerabilities that it finds are what's referred to as a buffer overflow. And that is, if I have an ActiveX control that has some sort of method that takes a string parameter -- so essentially I've got an application that's accepting a string as data as input -- what Dranzer will do is it will send a very large string to that method. And that is the exact sort of thing that can uncover a buffer overflow. And in the software security world, a buffer overflow -- it's one of the most basic, quite often exploitable mistakes that people can make when they write an application.

Without getting into the real nitty gritty details, when you have a buffer overflow, that's allowing an attacker to change control over what that application is doing. And as soon as an attacker has control over what an application is doing, they might be able to cause that program to run code.

The other sort of thing that Dranzer looks for are integer overflows, which is another type of programming flaw. And those can be a little bit trickier to exploit. But one of the very first vulnerabilities that I found with the Dranzer tool actually happened to be an integer overflow vulnerability in an ActiveX control that came with Windows. Basically you've got a component that comes with pretty much every Windows system. And people that are using the Internet Explorer web browser, if they visit a website that tries to attack this ActiveX control or exploit that ActiveX control, they could end up with malicious code running on their system. So this was very early on in the development of the Dranzer tool. This was before it was made publically available. We were reviewing the tool internally and testing out a couple things. And one of the very first things that we ran into was an exploitable vulnerability in an ActiveX control. So there's definitely a very strong tie between fuzz testing and vulnerabilities or software security.

Julia Allen: Excellent. Well, thank you for those examples.

Will Dormann: Sure.

Julia Allen: As we come to our close, in your own experiences and perhaps what you see other organizations doing, are there any rules of thumb for planning for the fuzz testing process and framework, like the number of people that are typically involved, how much it costs, how long it takes. I know that likely depends on the size and complexity of the software, but do you have any rules of thumb?

Will Dormann: Yeah, it really varies on the software vendor size, what sort of application they're writing. As to actually how much effort to put into that, that's where I was going with the automation discussion here.

Julia Allen: It seems to me if you have an automated environment as we talked about before, an ability to regression test as you change, that that really drives your cost and your schedule down, right?

Will Dormann: Absolutely. And it's something -- I'm just one person on the vulnerability analysis team here but I'm able to pretty effectively do fuzzing. And that's just because I'm not actually sitting down and manually spending effort doing the fuzzing. But rather I've spent a bit of effort in the automation aspect of things.

So for ActiveX, I had a mechanism that retrieved ActiveX controls and ran the Dranzer test on those controls and cataloged the reports or the results into a database. For the file fuzzing, I've got a similar automated setup. But regardless of how you end up doing it, the automation is really a key aspect of fuzzing because once you've got that environment set up, you can actually just let it go on its own. So rather than having a dedicated person that's sitting and doing fuzzing and spending effort doing fuzzing, if you can have some sort of environment set up where the fuzzing can be something that is a little bit autonomous -- it can just go on its own -- that's something that can happen concurrently with the actual software development. So if I've got a fuzzing environment set up, I can have that environment fuzzing the application that I'm developing. And then at the same time, I don't need to sit there and babysit it. I can just go

ahead and actually do more software development or do other tasks and then just check back every now and then to see what are the results of the fuzzing process.

Julia Allen: Excellent. Well let me ask you one last question before we close, and that is -- so let's say I'm not a developer of software but I'm an acquirer or a buyer of software. You've mentioned certainly doing fuzz testing on vendor products. But let's say I have a contract or a service-level agreement with an outside party and I've asked them to develop a piece of software for me. And it would be, from what you've said, it would be very prudent to require them to do this type of testing in their development. But for you as the receiver or as the customer for the software, might it make sense to do some fuzz testing of your own as part of accepting the software for delivery?

Will Dormann: Absolutely. It's a very good idea. And actually I'm not a software developer myself. I don't really write software. But I use software. And that's exactly the same kind of situation that somebody procuring software is -- they're in the same sort of situation. Even though they don't have access to the source code, they're not doing developing, the nice thing about fuzzing is that you don't need that access to actually test the software.

So if I'm receiving an application from somebody, if I have the ability to do some fuzz testing on that application, that testing can actually give me an indication of the code quality that I'm receiving. If I have two different applications -- let's say I have a particular image format that I'm interested in. If I have two different applications that can process that type of image, one way that I could compare those two applications is if I do a fuzz test run on both of those apps. And if one application crashes left and right and the other application is pretty solid and maybe will go for a couple days without crashing, that's probably a good indication of the general code quality and the amount of effort that the vendor has put into the product with respect to secure coding.

Julia Allen: Great. Well listen Will, this has been a fabulous introduction, and think with lots of good detail and guidance to encourage people to consider this testing approach, both for developing and acquiring software. Do you have some sources that you like, in addition to CERT website, where our listeners can learn more on the subject?

Will Dormann: Sure. There is, on the CERT website which you had mentioned, we actually have a paper that describes the Dranzer fuzz testing tool, and also the automation that we had used to do that fuzz testing. In addition to what's on the CERT website, there is a website called OWASP, O-W-A-S-P, that stands for the--

Julia Allen: Right, the Open Web Application Security Project, right.

Will Dormann: Exactly. They have a good reference for what is fuzz testing about; what are the sort of things that can cause an application to crash. That seems to be a pretty good resource for fuzzing. There actually is a website called fuzzing.org and I believe it's tied to a book that has been published on fuzzing. So aside from referencing that book, they also list several different fuzz testing tools. Almost all of them are freely Fwavailable open source sorts of applications. So just pick up a tool and have at it and see what you can come up with. Poke around in some applications and you might actually find something interesting.

Julia Allen: Well, this has been great, Will. Again, thank you so much for your time, expertise, the excellent guidance and pointers that you provided for our listeners. I really appreciate it.

Will Dormann: No problem; happy to help.