Java Security Traps Getting Worse

May 10 2007

A year ago at JavaOne , Fortify Software Founder and Chief Scientist Brian Chess gave a presentation titled " 12 Java Technology Security Traps and How to Avoid Them ."

A year later, how far have we come in addressing those inherent vulnerabilities, which include XSS (cross-site scripting), SQL injection and native methods that allow the import of C or C++ code - along with its bugs? Not a smidge - unless you count going backwards.

It's gotten worse, Chess said in an interview with eWEEK, "and I've got evidence to prove it."

Fortify, which markets source-code analysis technology, has access to a large database of common Java programming errors and vulnerabilities, gleaned not only from its customers but also from a year of running the Java Open Review project.

In that project, Fortify uses FindBugs , a static analysis tool that looks for bugs in Java code, to look over code in open-source projects such as Apache, Azureus and Tomcat. Fortify does an analysis on each inspected code set, publishes online how many issues it finds and then shares with project maintainers the vulnerability specifics.

What Fortify has found from running the project is that the defect density of open-source code is "astronomical," Chess said, pointing out one project in particular that Fortify has inspected over the past year: Net Trust , with an estimated 12.215 errors per 1,000 lines of code.

"That's huge for a project with 'trust' in its name," Chess said.

Ironically enough, Net Trust is a Google project to create a security mechanism for simple single sign-on and authentication. "But they were students doing not very good code," Chess said.

Net Trust is one of many examples that demonstrate that Java security traps, although known for some time, are snaring more programmers all the time as use of the language grows.

Java expert William Pugh agrees with Chess when it comes to Java security traps getting worse. "XSS is getting to be a very big issue," he said in an interview with eWEEK. "Tools like Fortify's tool set will look for problems with XSS, but it's not easy to cleanse your code of any XSS - vulnerabilities - . The statistics we've seen is that this is on its way to becoming the biggest vulnerability" in Java applications, if not in all Web attacks, he said.

Pugh is a professor in the department of Computer Science at the University of Maryland in College Park, as well as being the author of the FindBug tool that Fortify has used in the Java Open Review project.

Beyond XSS, Pugh said that the two issues people most talk about when they talk security in Java are typically untrusted malicious code and SQL injection. "In a case where you're running applets … - the question is, - what can those applets do? Can they change the behavior of the program you're running in any way?"

"People running stuff on servers, they don't run untrusted code," he said. " - But - another type of security vulnerability that's of much bigger concern is SQL injection. … Those are all continuing to be big issues."

Adding to the problem is that the teaching of secure coding remains

spotty at best.

To illustrate the lack of secure coding instruction, Chess points to a recent find he made, from none less than the Java giant, Sun. Specifically, the Sun introduction to servlet programming (Sun's simplest method for hooking Java up to the Web) contains cross-site scripting vulnerabilities.

One example of an XSS vulnerability are these lines from Sun's instructions:

```
try catch (Exception

e)

userName = firstname;

pw.print(" Thanks for your feedback, " + userName + "! ");
```

This code allows an attacker to inject code into the application that will be executed on a victim's browser, Chess said.

"The code expects that a user has entered a name like this: Bob," Chess wrote in an e-mail exchange. "But an attacker could set it up so that the data looks like this: sendDataToMotherShip() and then the victim's browser would execute a function named sendDataToMotherShip()."

A secure version of the server-side code, Chess said, would check input to make sure that it only contains an expected set of characters and no executable scripts.

"SQL injection problems still do sit at the top of the list" of Java security traps, he said. " - Developers are - trusting input they shouldn't trust."

If this is coming from Sun, who can we trust? "You'll see that the tutorial never mentions security," Chess said. "With that in mind, it's not surprising that it contains cross-site scripting vulnerabilities."

The problem is that Java - and other Web programming languages - make XSS a "really easy mistake to make," he said. "Think of writing the simplest Web page: The first thing you might do when you build a Java Web application. You tell it your name and it says Hello, Brian when I type in my name. The functions that Java provides just to do a 'Hello, world' function will allow XSS."

Web frameworks built into the Java language also make it easy to write an XSS vulnerability into Java code, he said.

"Worse, on top of that, is when we go and teach people how to write code in Java, we give examples of code that's vulnerable to XSS," he said.

As it now stands, the responsibility to watch for the most common Java security traps lies completely with the developer.

Can this situation ever be remedied? It could be, but it wouldn't be easy, Chess said. One step toward a solution would be to modify browsers to make XSS harder. What that would require is changing a Web standard and getting all the browser biggies - Microsoft, Mozilla and Apple - to sign on.

Even if somebody did talk them into going along with the plan, it would require a new browser to be pushed out to millions of users. "The fastest we could change that stuff, it would take years," Chess said. "[Changing] the fundamentals of frameworks or languages, that takes a really long time to do."

The reason why XSS deserves such a beating is because the situation more and more parallels where we were with buffer overflows 10 years ago. Both security vulnerabilities are very powerful for attackers, Chess said, as the flaws allow attackers to inject code into a system and provide complete takeover.

The reason buffer overflows have been such a problem is that frameworks in C and C++ made them very easy to create, he said. It happened with buffer overflows, and now it's happening with XSS.

Where it diverges, though, is that buffer overflows are somewhat hard to exploit, Chess said, requiring an attacker to be fairly knowledgable about a system's architecture and what's happening on that machine. "XSS vulnerabilities are much easier to exploit," he said. "Just go to your local bookstore, buy a book on Java and you can get started on XSS."

It's not a great state of affairs, but it's not time to throw in your hat, either. Fortify has seen open-source projects where developers have clearly taken an effort to prevent XSS. Such projects may have a few XSS vulnerabilities, compared with a project of similar scope from a clueless developer that has 50.

"We see when developers pay attention to it they can bring their numbers down," Chess said.

Still, Chess said, Fortify hasn't seen a magic turnover where developers are waking up to secure Java coding practices. Rather than relying on those developers, a more effective route may be to talk to framework owners and software makers to see what can be done to make the Web a safer place to program, Chess said. That, however, is no quick fix either.

"It's going to be a long road," he said.

Editor's Note: This story was updated to correct HTML codes around the characters in the code sample and to add input from William Pugh.

Copyright 2007 by Ziff Davis Media, Distributed by United Press International

Citation: Java Security Traps Getting Worse (2007, May 10) retrieved 26 April 2024 from https://phys.org/news/2007-05-java-worse.html

This document is subject to copyright. Apart from any fair dealing for the purpose of private study or research, no part may be reproduced without the written permission. The content is provided for information purposes only.