Secure in 2010? Broken in 2011!

Matias Madou
Principal Security Researcher

Abstract

In 2010, a security research firm stumbled on a couple of vulnerabilities in Apache OFBiz, a widely used open source enterprise automation software project. As a proof of concept, it posted a video showing how easy it was to become an administrator exploiting one of the XSS issues in the application. To remain credible, the OFBiz team was forced to invest in security. In fact, as a result of digging into its bug database, the OFBiz team gathered security knowledge from different sources to make its product better, and made a big push to resolve the known issues in early 2010. Barely a year later, the exact same code base thought to be secure is again seriously broken. This scenario actually occurs quite frequently for several reasons.

In this Whitepaper, we discuss:

- The experiment
- The test application
- What's new in 2011
    - New vulnerabilities
    - New assessment techniques
- Continued testing
- Conclusion

The experiment

While investigating our gray-box analysis technique, we wanted to gather some empirical data to prove that we were taking a new approach to finding security issues. After choosing our test application and beginning our investigation, we actually found more interesting results than what we had expected. We also wound up conducting a white-box and black analysis to find out more about the security of the application, and how that compared to the state of the application only a year ago.

The test application

We looked for an application which was written in Java or .NET without taking the underlying packages into consideration. Most importantly, we had to choose an application that was widely used and one in which security improvements had been implemented.

After evaluating a dozen open source Enterprise Resource Planning (ERP) and/or Customer Relationship Management (CRM) packages, we chose the widely used Apache OFBiz for our experiment. Apache OFBiz is ranked the #1 Open Source ERP Software Application[1], and we saw additional value in Apache OFBiz for various reasons. First of all, a long list of well known end users like 1-800-flowers, Olympus.de, united.com and bt.com are using this software. But on top of that, the framework itself is used in various other products and projects. One of the most known ERP/CRM solutions based on Apache OFBiz is OpenTabs, which is used by enterprises like Toyota and HoneyWell. Needless to say our test application is well used and integrated in today's businesses.

[1] http://www.erpsoftware360.com/erp-open-source.htm

The security of the Apache OFBiz solution seemed to have undergone some scrutiny. By searching the web, it becomes clear that numerous vulnerabilities like XSS were reported and fixed from early on in the Apache OFBiz history. The most notable security incident was in 2010 when a security research firm discovered a couple of vulnerabilities in Apache OFBiz and posted a proof of concept video showing how easy it was to become an administrator exploiting one of the XSS issues in the application. The OFBiz team took action and fixed the security issues. In fact, as a result of digging into its bug database, the OFBiz team gathered security knowledge from different sources to make its product better, and made a push to resolve the known issues in early 2010. So from a security perspective, the application we've taken for our experiment has undergone security improvements.

What's new in 2011?

Both white hats and black hats seem to be coming up with new ways to break applications. In most cases, new rules can be added to current assessment techniques to find these new categories of vulnerabilities. Here, we'll discuss a problem found in the SUN JVM which can lead to a Denial-of-Service.

Second, new assessment techniques can find more of the same. Where the typical white-box and black-box techniques can find a big chunk of the vulnerabilities, a novel assessment technique called gray-box analysis takes the best of both worlds and finds additional exploits. We'll also discuss how the new gray-box analysis finds exploits of well known vulnerability categories.

a) New vulnerabilities

2011 had barely begun when a problem was discovered in the SUN JVM. CVE-2010-4476 points out that the Java Runtime Environment hangs when converting "2.2250738585072012e-308" to a binary floating-point number. More concrete, when the mentioned value is given as the first parameter in the API Double.parseDouble(param1), the JVM goes in to an infinite loop as it is oscillating between DBL_MIN (0x1p-1022), and the largest subnormal double-precision floating-point number (0x0.fffffffffffffp-1022) [2]. While chances are fairly slim that this value appears in regular java applications, it's obvious that an attacker can inject this special value in the right place in order to hang the application.

Interestingly enough, this problem was not reported for the first time in 2011. Dating back as early as 2001, there were bugs revealing this problem. At that time, the bug was not considered a 4-low priority, as its consequences were not well understood. When it became clear that application servers like Apache Tomcat were remotely vulnerable-- regardless of the web applications they were running, the bug got fixed in record time.

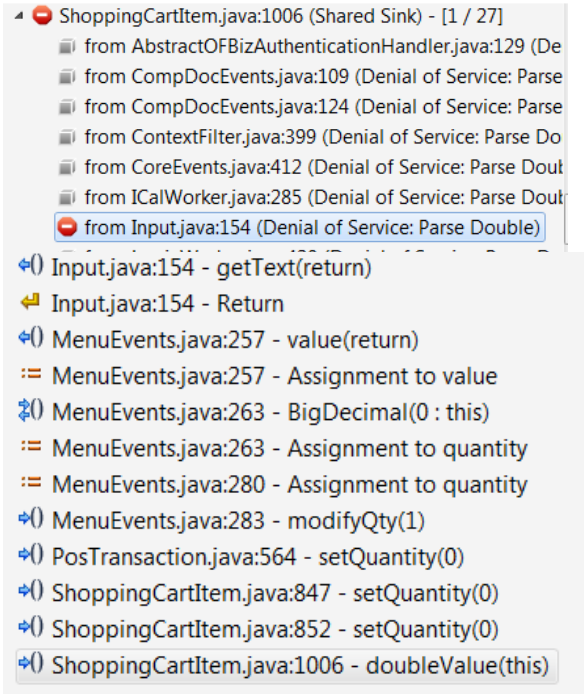
While the root cause of this problem lies in the SUN JVM, some enterprise environments cannot quickly upgrade to the latest JVM. These enterprises must also rely on other mechanisms to reduce their exposure. As most enterprises already use WAF technology, new rules can be added to protect against this type of attack. However, as always, it's important not to forget any pattern. As the CVE only mentions "2.2250738585072012e-308", it's incorrect to only protect against this pattern. The rule should also protect against "0.22250738585072012e-307" up till "0.00…0022250738585072012". Some comments on the web also suggested that there are other tiny values that can trigger this behavior too, so how to protect against these?

[2] http://www.exploringbinary.com/java-hangs-when-converting-2-2250738585072012e-308/

A second way to remediate is patching the exposed applications. The Apache Tomcat problem was quickly remediated by pushing out a fixed 5.5, 6.0 and 7.0 version. The solution is a white-listing approach, where they only accept strings with a certain length, making it impossible for the magic value to go in the infinite loop.

Now, when looking in Apache OFBiz, we found numerous of these problems using different analysis techniques. Doing a simple grep of "Double.parseDouble" on the Apache OFBiz application reveals the API is used some 25 times. However, that does not say anything about the actual paths from a source which can be used by an attacker to a vulnerable sink. By means of static analysis, (white-box analysis), such smarter analysis can be done. We scanned the code and found many of these paths. For example, check out Figure 1: there is an analysis trace from the Input class to the ShoppingCartItem. As static analysis points out, paths that are possible in theory, doesn't mean they can be taken in practice. In this case, however, the analysis trace can be executed in practice as well.

Figure 1: Example of a ShoppingCartItem




```
UtilMisc.toMap("requestedQuantity", UtilFormatOut.formatQuantity(quantity.doubleValue()),
               "productName",       this.getName(),
               "productId",         productId);
```

While static analysis does not have the "show-me" factor, penetration testing tools (black-box analysis) do. When attacking Apache OFBiz with a penetration testing tool configured to find the Denial-of-Service problems, the penetration testing tool takes the application down pretty quickly. As such, there is plenty of opportunity to launch a Denial-of-Service attack by exploiting this problem. A list of example URLs:

http://yourofbiz.com/ecommerce/control/modifycart (update_0, update_1, …)
http://yourofbiz.com/ecommerce/control/additem/showcart (quantity, add_product_id)
http://yourofbiz.com/ecommerce/control/additem/quickadd (quantity)
http://yourofbiz.com/ecommerce/control/additem/keywordsearch (quantity)
http://yourofbiz.com/ecommerce/control/additem/advancedsearch (quantity)
http://yourofbiz.com/ecommerce/control/additem/showPromotionDetails (quantity)
http://yourofbiz.com/ecommerce/control/additem/product (quantity,add_amount)
http://yourofbiz.com/ecommerce/control/additem/lastViewedProduct (update_0)
http://yourofbiz.com/ecommerce/control/additem/showForum (quantity)
http://yourofbiz.com/ecommerce/control/additem/category (quantity)
http://yourofbiz.com/ecommerce/control/additem/main (quantity)
http://yourofbiz.com/ecommerce/control/additem (quantity)
http://yourofbiz.com/ecommerce/control/additem/setDesiredAlternateGwpProductID (…)

b) New assessment techniques

Black-box analysis such as penetration testing and white-box analysis such as static analysis are common these days as enterprises take security seriously. The strength of Black-box analysis is the evidence such tool can provide to show that an issue is exploitable and the application is really broken. The weakness of such tool is it fails to find inputs to test all the executable paths through the application which results in testing only a portion of the application instead of the entire application. White-box analysis' strength and weakness are just the other way around: the analysis is thorough while lacking the convincing aspect of why an issue should really be fixed.

Gray-box analysis is about marrying the two analyses techniques and overcoming each other's weaknesses. The goal is to find more issues and fix the issues faster. To find more issues, white-box techniques are used to make sure the coverage is satisfying. To fix issues faster, information from inside and outside the application is given to prove that the issue is real and information is provided as to where the issue can be fixed.

Gray-box analysis starts off as black-box analysis where attacks are sent out to a running application. In addition, a monitor component is installed in the running application which observes the incoming attacks and the execution of the application. We call the attacking component the Dynamic Analysis and the runtime component the Real-Time Analysis. By observing the executed code, the monitor can give real time feedback to the Dynamic Analysis that is attacking the application.

In essence, the Real-Time Analysis component can give multiple hints to the Dynamic Analyzer. First, the Real-Time Analyzer can tell the Dynamic Analyzer the attack surface. It may be the case that not all pages in the application have a link from the main application. Normal black-box analysis has a minimal chance of penetrating these hidden pages. Now, with gray-box analysis, it's trivial for the monitor to tell about these pages which we call the attack surface. Second, the Real-Time Analyzer can describe the exact consequence of an attack performed by the Dynamic Analyzer, while in a black-box analysis scenario, the success of an attack was determined based on what ,(if anything at all), was coming back to the black-box analyzer. Now, the Real-Time Analyzer knows if an attack was successful or not, and can transfer that piece of information back to the Dynamic Analyzer.

In the case of our test application, Apache OFBiz, we analyzed the increase in attack surface, the effect of this increase on the findings, and the number of findings in general compared to pure black analysis. First, it was obvious that the number of scanned directories improved dramatically. The number of scanned directories was 3.5Xpure black box analysis. These new directories were the direct reason of a dozen new Cross-Site Scripting (XSS) vulnerabilities in Apache OFBiz. Where the pure black-box analysis didn't find the "webslinger" directory, the gray-box analysis did scan this directory and found five new XSS issues.

From an attacker perspective, at least five new URLs could be created to get the XSS in the "webslinger" directory. However, with the information we saw in the Real-Time Analyzer component, the analyzer determined that all five issues had one root cause in the code. Now, really detailed information can be given back to the developer to fix this problem. In gray-box analysis, the five URLs together with line of code details where the problem occurs in the application can be transferred to the developer. In this case, the developer fixing this problem in the code has to fix one root issue in code and will as such reduce the exposure of the application to XSS by five issues.

Continues testing

Let's come back to the root question, how could the team working on this application prevent the application being broken in 2011? The only solution is continuous, automated testing of the application, even if that application's code is frozen and in production. Companies with a good security initiative do continuous testing of their applications while they are under development, however, once someone signs off for the security in the applications and they are in production, the applications may go out of that cycle of continuous testing. From that moment on, companies rely on bandages in the form of a WAF to "fix" problems when they appear. In most cases, the code is fixed when breached, so the code is always patched too late.

Conclusion

Applications in production need continuous, automated testing with the latest security knowledge. When a new vulnerability comes to light, it's important to scan the applications in production for these problems. Similarly, when analysis techniques are available which take a different approach, it may be a good idea to scan the applications in production to see if they have kept up with the latest vulnerability finding techniques. The attackers can use these techniques too, which means they know more about the applications you're running!

About the author

Matias Madou is Principal Security Researcher for HP Enterprise Security Products (former Fortify). He works on technical projects, ranging from kicking off an insider threat project, to spearheading new protection mechanisms in the runtime tools, to leading the correlation and integration of current HP Fortify security solutions and has been instrumental in defining and refining the HP Fortify solutions offerings.

When he's away from his desk, he's serving as an instructor to advanced training courses, helping the field on short notice or presenting at anchor industry conferences including: DefCon, RSA, BruCon, Owasp, and more.

He holds a Ph.D. in computer engineering from Ghent University, where he studied application security through program obfuscation to hide the inner workings of an application. During his Ph.D., he collaborated with top research and industry players in the field of program obfuscation, which has helped mold the research he continues to work on today.

References

- Real-Time Hybrid Analysis: Find More, Fix Faster by Brian Chess