CERT/CC Blog

Vulnerability Insights

One Weird Trick for Finding More Crashes

POSTED ON SEPTEMBER 23, 2013 BY WILL DORMANN [/AUTHOR/WILL-DORMANN] IN TOOLS [HTTPS://INSIGHTS.SEI.CMU.EDU/CERT/TOOLS/]

Hi folks. It's Will Dormann from the CERT Vulnerability Analysis team [http://www.cert.org/vuls/]. Today we're announcing the release of updates to both of our fuzzing tools, the CERT Basic Fuzzing Framework (BFF) version 2.7 and the CERT Failure Observation Engine (FOE) version 2.1. In this blog entry I will describe some of the major changes with these tools.

So what's the one weird trick for finding more crashes that caused software vendors to hate me? Okay, that may have been somewhat of a ruse. People apparently love to click links that promise weird old tricks. However, both BFF 2.7 and FOE 2.1 do have an optional feature that in our testing drastically increases the number of uniquely crashing test cases: **crasher recycling**. When enabled, the frameworks will add uniquely crashing test cases back into the pool of seed files for further fuzzing. When crashing test cases are fuzzed more, this can uncover both new vulnerabilities as well as new ways in which an already-known vulnerability can cause a crash.

Now on to the other changes...

Virtual Machine Changes

Up through BFF version 2.6, we have provided a Debian-based Linux virtual machine for fuzzing. In my testing, some applications were tricky to install, though. This was usually due to outdated or unavailable dependencies. Now we are providing an Ubuntu-based Linux virtual machine called UbuFuzz. We also are providing a 64-bit version called UbuFuzz64.

BFF 2.7 Changes

PIN - With some crashes, the stack is trashed completely. Because BFF uses the `gdb` backtrace to determine the uniqueness of a crash, this means that all crashes that completely trash the stack will look the same. Starting with BFF 2.7, crashes that completely trash the stack are further investigated using a customized PIN [http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool] tool that creates a call trace. Rather than trying after the crash to determine the code path that an application took, the PIN tool creates a call trace as the application executes. Using this technique, we can determine the execution path that an application takes, regardless of the state of the application's stack.

Minimization to Metasploit string - BFF 2.5 [http://www.cert.org/blogs/certcc/2012/04/cert_basic_fuzzing_framework_v.html] introduced a minimization-to-string feature. Given any particular crash, how do you determine if data or register values are potentially under your control, without needing to do taint analysis? With a crashing test case that has been minimized to a string of lowercase 'x' characters, it's quite easy to spot. However, if, for example, a particular register is `0x78787878`, how can you tell where from the file that value was obtained? This is why we've changed the default string minimization to use the Metasploit [http://www.metasploit.com/] pattern. With this change, the specific pattern observed can give you an indication of the offset within the file from which the bytes were obtained. If the fuzzed file is large enough, the Metasploit pattern may repeat. This is where the `tools\mtsp_enum.py` script comes in handy. When you give it a byte or string pattern that occurs multiple times within a file, it will modify those values so that each occurrence will be unique.

Test case reproduction - You can interactively debug a crashing test case with the configured target and commandline by using the `tools\repro.py` script. You can also choose a different debugger to run the target application.

FOE 2.1 Changes

`!exploitable` 1.6 - FOE 2.1 includes Microsoft !exploitable 1.6 [http://msecdbg.codeplex.com/], which includes a number of bug fixes and improvements.

Crash uniqueness by exception chains - Prior to FOE 2.1, crash uniqueness was determined by the `!exploitable` crash hash for the first exception encountered. Windows applications have the concept of exception handling [http://msdn.microsoft.com/en-us/library/windows/desktop/ms680657%28v=vs.85%29.aspx], however. When an exception is encountered, the application may choose to handle the exception and continue, potentially encountering further exceptions. FOE 2.1 uses configurable-depth exception chains to uniquely identify a crash, as well as taking the highest exploitability of **any** of the exceptions encountered when bucketing crashes by exploitability.

Dynamic timeouts for GUI applications - Prior to FOE 2.1, one of the most important configuration options for GUI applications is the timeout. CLI applications will execute as fast as your CPU will allow, but GUI applications are different. They usually do not terminate when they are "done." Therefore, it was critical to set a timeout that was long enough to allow the target application to process the fuzzed file, but not so long that the machine was sitting idle, wasting resources. FOE 2.1 determines if the target application will terminate on its own. If it does not, it will enable CPU-usage-based timeouts. This both improves fuzzing

throughput, and also simplifies configuration.

Zip-based seed file awareness - A number of modern file formats are zip based, such as DOCX or XPS. When fuzzing zip-based files, older FOE versions would simply fuzz the zip container. However, FOE 2.1 by default will automatically determine if each seed file is zip based, and for those seed files, FOE will mutate the contents of the zip file rather than the container itself.

Test case reproduction - See above.

Minimization to Metasploit string - See above.

Getting Started

Read the Quick Start Instructions

Quick start instructions for both tools can be found on their download pages: BFF 2.7 [http://www.cert.org/download/bff/], FOE 2.1 [http://www.cert.org/download/foe/].

Contact Us

If you have any questions or comments, please feel free to contact us [mailto:cert@cert.org?subject=CERT%2FCC%20BFF%202.7%20Feedback%20INFO%23817809].

About the Author

Will Dormann

Contact Will Dormann [https://www.sei.cmu.edu/contact.cfm]
Visit the SEI Digital Library for other publications by Will [https://resources.sei.cmu.edu/library/author.cfm?authorID=2547]
View other blog posts by Will Dormann [/author/will-dormann]