Software Engineering Institute
Carnegie Mellon University

SEI Insights

CERT/CC Blog

Vulnerability Insights

CERT Basic Fuzzing Framework

POSTED ON MAY 26, 2010 BY WILL DORMANN [/AUTHOR/WILL-DORMANN] IN TOOLS [HTTPS://INSIGHTS.SEI.CMU.EDU/CERT/TOOLS/]

Hi folks. I've been involved in a fuzzing effort at CERT. One of the ways that I've been able to discover vulnerabilities is through "dumb" or mutational fuzzing [http://en.wikipedia.org/wiki/Fuzz_testing]. We have developed a framework for performing automated dumb fuzzing [http://msdn.microsoft.com/en-us/library/cc162782.aspx#Fuzzing_topic4]. Today we are releasing a simplified version of automated dumb fuzzing, called the Basic Fuzzing Framework (BFF).

Dranzer [http://www.cert.org/vuls/discovery/dranzer.html] was one of our first fuzz testing projects. By performing automated smart fuzz testing of ActiveX controls, I was able to discover thousands of vulnerabilities. Luckily, Microsoft has made some improvements [http://blogs.msdn.com/ie/archive/2008/05/07/ie8-security-part-ii-activex-improvements.aspx] to Internet Explorer to help minimize the impact of ActiveX vulnerabilities.

Another technique that I've used for discovering vulnerabilities is dumb fuzzing. Don't let the name fool you. Dumb fuzzing has the advantage of being more universal than smart fuzzing. Dranzer is limited in that it tests only ActiveX controls; with dumb fuzzing, you can switch targets easily after your dumb fuzzing environment is complete.

The Basic Fuzzing Framework (BFF) consists of two main parts:

1. a Linux virtual machine that has been optimized for fuzzing
2. a set of scripts and a configuration file that orchestrate the fuzzing run

The virtual machine is a stripped-down Debian installation with the following modifications:

- The Fluxbox window manager is used instead of the heavy Gnome or KDE desktop environments.
- Fluxbox is configured to not raise or focus new windows. This can help in situations where you may need to interact with the guest OS while a GUI application is being fuzzed.
- Memory randomization is disabled.
- VMware Tools is installed, which allows the guest OS to share a directory with the host.
- The OS is configured to automatically log in and start X.
- `sudo` is configured to not prompt for a password.
- `strip` is symlinked to `/bin/true`, which prevents symbols from being removed when an application is built.

The fuzzer used by the BFF is Sam Hocevar's excellent zzuf [http://caca.zoy.org/wiki/zzuf] application. zzuf was chosen for its deterministic behavior, number of features, and lightweight size. By invoking zzuf from a script (`zzuf.pl`), we are able to automate additional aspects of a fuzzing run:

- Collect program stderr output, valgrind memcheck, and gdb backtrace. This information can help a developer determine the cause of a crash.
- De-duplication of crashing testcases. Using gdb backtrace output, `zzuf.pl` will determine if a crash has been encountered before. By default, duplicate crashes are discarded.

The `zzuf.pl` reads the configuration options from the `zzuf.cfg` file. This file contains all of the parameters relevant to the current fuzz run, such as the target program and syntax, the seed file to be mutated, and how long the target application should be allowed to run per execution. The configuration file is copied to the guest OS when a fuzzing run has started. The zzuf script will periodically save its current progress within a fuzzing run as well. These two features work together to allow the fuzzing VM to be rebooted at any point, allowing the VM to resume fuzzing at the last stop point. The fuzzing script also periodically touches the `/tmp/fuzzing` file. A linux software watchdog will check for the age of this file; and, if it is older than the specified amount of time, the VM will automatically be rebooted. Because some strange things can happen during a fuzzing run, this robustness is necessary for full automation.

The BFF is preconfigured to automatically begin fuzzing a very old version of ImageMagick. A debug build of ImageMagic 5.2.0 is installed on the system, `zzuf.cfg` is set up with the fuzzing parameters, and a simple Netpbm seed file is included. When the machine is powered on, it will automatically log in and `zzuf.pl` will invoke the zzuf fuzzer. ImageMagick's `convert` program will repeatedly execute, attempting to convert the seed file into a bitmap file. The way that zzuf works is that each time the application is launched, the seed file will be mangled in a certain way. The goal of fuzzing is to determine malformed input that causes the target application to crash. The zzuf.pl script takes this one step further by collecting additional information about the crashes. Cases that are determined to be unique are saved.

To begin fuzzing on your own, simply follow these steps:

1. Unzip `scripts.zip` to c:\fuzz
2. Unzip `DebianFuzz.zip` to a directory of your choice.
3. Open `DebianFuzz.vmx` with VMware.
4. Create a snapshot in VMware

5. Power on the VM

You may need to verify that the shared folder is enabled in the VM preferences. Other virtualization products may work with some additional configuration. See the README.txt file in `scripts.zip` for more details.

Get your own BFF here [http://www.cert.org/download/bff] .

P.S. If you are interested in this sort of stuff, check out our job opportunities [http://www.cert.org/jobs/] .

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]