CERT/CC Blog

Vulnerability Insights

Visualizing CERT BFF String Minimization

POSTED ON JUNE 6, 2016 BY WILL DORMANN [/AUTHOR/WILL-DORMANN] IN TOOLS [HTTPS://INSIGHTS.SEI.CMU.EDU/CERT/TOOLS/]

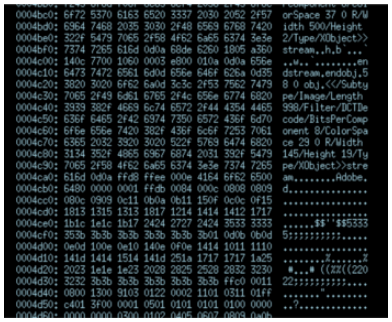
I've been working on a presentation called *CERT BFF - From Start to PoC* [https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=463982]. In the process of preparing my material, I realized that a visualization could help people understand what happens during the BFF string minimization process.

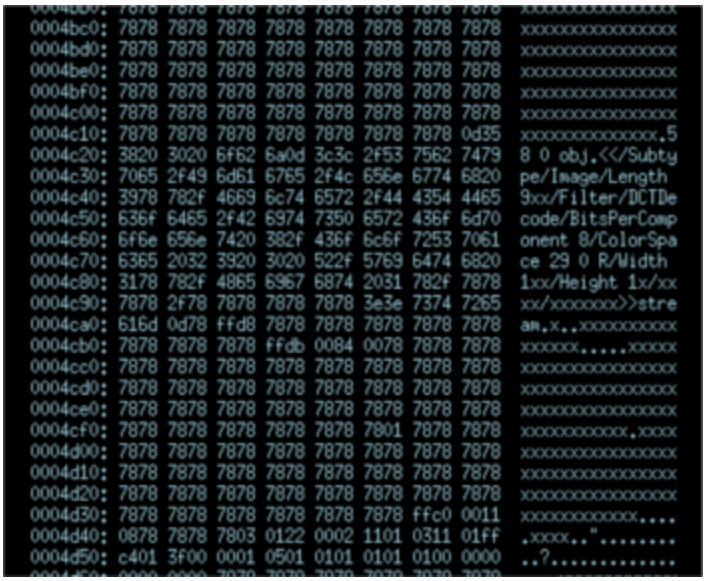
Starting with version 1.1 [https://insights.sei.cmu.edu/cert/2010/09/cert-basic-fuzzing-framework-update.html], which was released in 2010, CERT BFF produced a "minimized" version of a fuzzed file. This minimized file is a crashing test case that is minimally different from the seed file that it was derived from.

For example, consider the case where BFF produces a crashing test case that was created by modifying 100 bytes, but only one byte is required to reproduce the crash. The minimized test case is only one byte different from the seed file. This approach can aid in crash analysis by focusing where the analyst needs to look.

Starting with version 2.5 [https://insights.sei.cmu.edu/cert/2012/04/cert-basic-fuzzing-framework-25-released.html], which was released in 2012, BFF provides an improved minimization capability [https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=28043], which was designed by Allen Householder. This version of BFF has an optional "string" minimization capability. While conceptually similar to the standard minimization that occurs during a BFF run, a string-minimized crashing test case reflects a different use case. The output from a string minimization can help analysts see which bytes are used for the structure and crash reproduction as well as which bytes can likely be modified to arbitrary values (e.g., which bytes could hold the shellcode for a proof-of-concept [PoC] exploit).

Borrowing from the BFF 2.5 announcement blog post, here is a simple static visualization of the crashing test case on the left, and the string-minimized test case on the right:





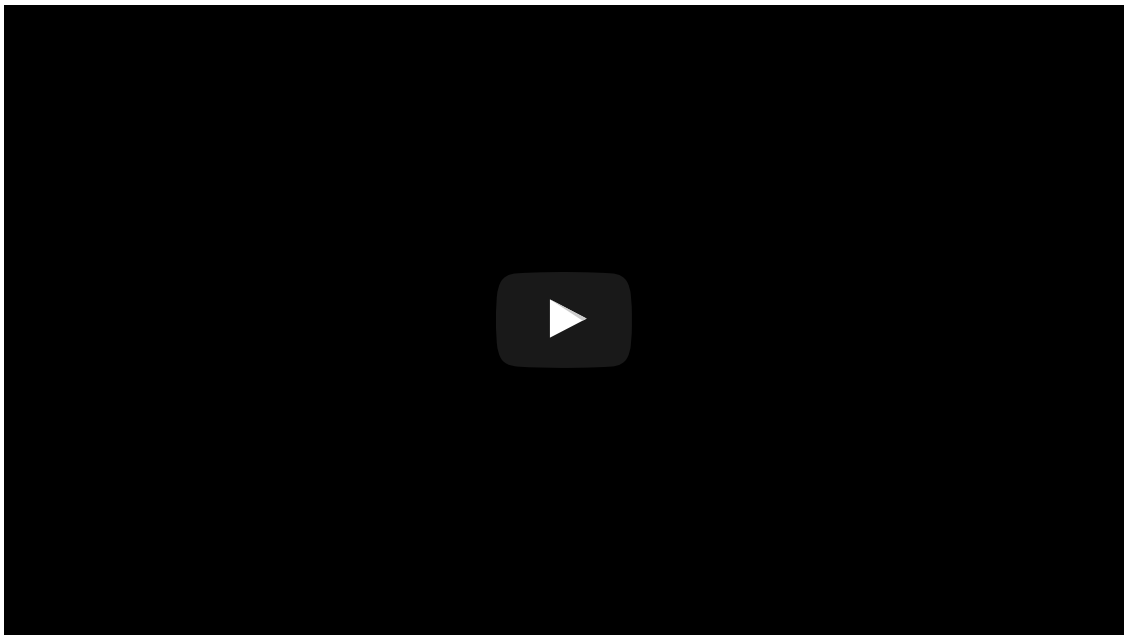
[https://insights.sei.cmu.edu/cert/2016/06/06/minimized-to-x.png]

While the difference between the start and end points are pretty clear, the actual steps taken to get there are not obvious. This is where my visualization comes in handy. I took a crashing MIFF test case from the default ImageMagick fuzz run that the CERT BFF performs, and I ran it through a string minimization. Only this time I made a small modification to the BFF code to save a copy of the trial test cases each step along the way.

BFF will randomly try to change a byte's value to 0x78, which is a lower-case 'x' character in ASCII. If it produces the same crash, then BFF keeps it and continues the minimization process with that file as the best case so far. It will continue the process until it's pretty sure that it's complete. This certainty level is configurable via the `tools/minimize.py` command-line. To see all of the minimization options available, run `tools/minimize.py -h`

What's interesting to see is that the result of string minimization is a file that can clearly show the structure of the file, but BFF is completely agnostic to the data that it is mutating. It is simply leveraging brute force to achieve its goal.

Without further ado, here is the animated visualization that I put together:



< Previous Article Next Article >

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]