A Combined Approach to Vulnerable Code Clone Discovery

Harrison Fernandez
Computer Science and Information Security Major
Professor Sven Dietrich, Mathematics and Computer Science Department

Abstract

This poster focuses on code cloning, the reuse of code in same or different programs. Given code cloning tends to create bugs and vulnerabilities, we perform a longitudinal study on code similarity and vulnerabilities in the open source firmware project, Tomato, encompassing more than twelve years of development. This poster explores cloned code, how it exists in Tomato, and, in a combined approach, examines the vulnerable binary and source code of the Busybox tool.

Introduction

Code can be referred to as source or binary.

```
include<iostream>
Using namespace std;
Int main()
{ cout<<''Hello World!''<<endl
   return 0; }
```

```
00000942: 00000010 00000000 01101000 01100101 01101100 01101100 ..hell
00000948: 01101111 00100000 01110111 01101111 01110010 01101100 o worl
0000094e: 01100100 00000000 00000001 00011011 00000011 00111011 d....;
00000954: 01000100 00000000 00000000 00000000 00000111 00000000 D.....
0000095a: 00000000 00000000 01110000 11111101 11111111 11111111 ..p...
```

Figure 1: Source (left) and binary (right) to display "Hello World".

With the availability of code on the internet (i.e. Github), code cloning is common practice.

- Code cloning affects maintenance and introduces vulnerabilities in large programs[1].
- Vendors of embedded devices are notorious for poor security practices[2], yet routers handle internet traffic, passwords, and more.
- Vulnerabilities may live in software for more than three months without updates or disclosure[3].
- With no way to hold developers accountable, code cloning may affect computer systems and real-time devices.

Research Question

How have vulnerable code clones lived and persisted through software over time?

- Even if the source code is patched, is the binary code patched?

Methodology

We analyze binary and source code of the Tomato firmware project.

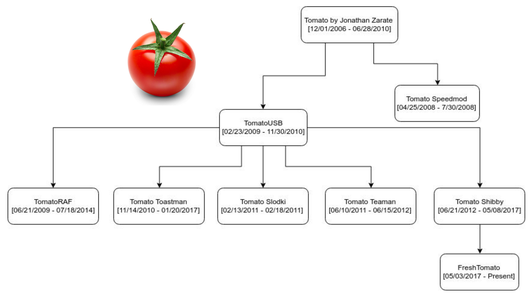

Figure 2: Flow chart of Tomato firmware lineage

The reverse enginneering tool, Radare2, is used to analyze and compare binary files. Note binary similarity between forks may differ due to compiler options and computer architecture, affecting similarity scores.

Binary Analysis

Analysis is done on the BusyBox binary across the Tomato forks. We search for two vulnerabilities:

- CVE 2011-2716: BusyBox version allows remote servers to execute commands on client side.
- CVE 2016-2148: BusyBox version allows remote attackers to have unknown impact.

Figure 3: BusyBox logo

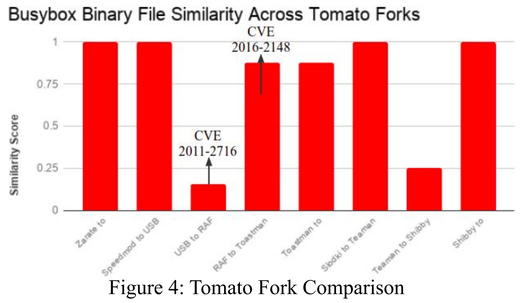

Figure 4: Tomato Fork Comparison
USB to RAF scored of 0.156, RAF to Toastman scored 0.877, and Teaman to Shibby scored 0.249.

We analyze USB to RAF as well as RAF to Toastman, but disregard Teaman to Shibby as there is no known vulnerability.

A highly similar function denoted as 'not a match' is what we hope to be a patch. In Figure 5, note the sym.ext2fs_write_bitmaps function: it is 98% similar yet it is not a match.

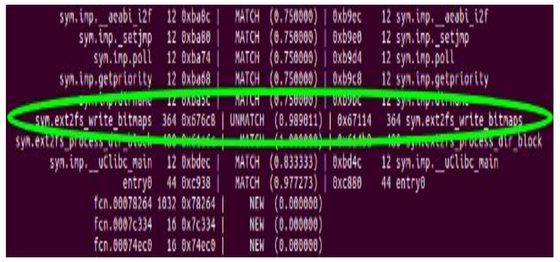

Figure 5: Toastman (left) and RAF (right) BusyBox Function Comparison

With a deeper look in Figure 6, there is no difference between them.

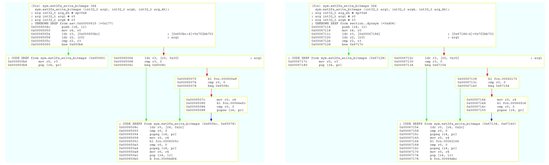

Figure 6: Toastman (left) and RAF (right) BusyBox sym.ext2fs_write_bitmaps function CFG representation.

Source Code Analysis

To further examine binary analysis results, we analyze the source code of write_bitmaps function.

First, we must conduct a survey of three state-of-the-art source clone detectors: Deckard, NiCad, and Simian, utilizing Github diffs as ground truth.

Tool	Project	Lines of Code (LoC)	Timing
Deckard	LibreOffice	15.55 million	8 min 30 sec.
	MongoDB	4.85 million	1 min 15 sec.
	Radare2	1.57 million	4 min 57 sec.
Nicad	LibreOffice	15.55 million	1 min.
	MongoDB	4.85 million	8 min 56 sec.
	Radare2	1.57 million	7 min 20 sec.
Simian	LibreOffice	15.55 million	8 min.
	MongoDB	4.85 million	3 hours 50 min
	Radare2	1.57 million	24 min.

Table 1: Timing and scalability results from SCC survey

Tool	Settings	F-score
Deckard	MIN-TOKENS = 30, STRIDE = 2, SIMILARITY 0.95	0.9831
Nicad	difference threshold of 0.30, granularity: functions-blind	0.9091
Simian	minline threshold = 6, ignore character case, curly braces, identifier case, modifiers, string case = true	0.9655

Table 2: Accuracy test results from SCC survey

RAF	Toastman
000000113 dist:0.0 FILE ./busybox-raf/e2fsprogs/ext2fs/rw_bitmaps.c LINE:45:8 NODE_KIND:131 nVARs:6 NUM_NODE:116 TBID:0 TEID:49	000000041 dist:0.0 FILE ./busybox-toastman/e2fsprogs/ext2fs/rw_bitmaps.c LINE:45:8 NODE_KIND:131 nVARs:6 NUM_NODE:116 TBID:0 TEID:49
000000068 dist:0.0 FILE ./busybox-raf/e2fsprogs/ext2fs/rw_bitmaps.c LINE:277:18 NODE_KIND:131 nVARs:11 NUM_NODE:166 TBID:1216 TEID:1288	000000026 dist:0.0 FILE ./busybox-toastman/e2fsprogs/ext2fs/rw_bitmaps.c LINE:277:18 NODE_KIND:131 nVARs:11 NUM_NODE:166 TBID:1216 TEID:1288

Figure 6: Source code comparison of RAF and Toastman's write_bitmaps function using Deckard.

- Source files are code clones, yet the binary functions are 'not a match'.
- We find source code is patched in common.c, but in different place than binary code.

Conclusion

- Deckard performs the best in our survey.
- The USB to RAF fork is a timely and effective patch of CVE 2011-2716, as functions are new or not a match.
- Source code analysis helps search for binary fix, yet unknown in RAF to Toastman fork.
- Continue to find vulnerable code and hold programmers accountable.

References

1. Chen, D., Egele, M., Woo, M., & Brumley, D. (2016). Towards automated dynamic analysis for linux-based embedded firmware. In Proceedings of 2016 network and distributed system security symposium.
2. Costin, A., Zaddach, J., Francillon, A., & Balzarotti, D. (2014). A large-scale analysis of the security of embedded firmwares. In Proceedings of 23rd usenix security symposium (p. 95-110).
3. Li, F., & Paxson, V. (2017). A large-scale empirical study of security patches. In Proceedings of 2017 acm sigsac conference on computing and communications security (p. 2201-2215).

Acknowledgements

Support for student stipends, supplies, and/or equipment used in this research was supplied by the Program for Research Initiatives in Science and Math (PRISM) at John Jay College. PRISM is funded by the Title V and the HSI-STEM programs within the U.S. Department of Education; the PAESMEM program through the National Science Foundation; and New York State's Graduate Research and Technology Initiative and NYS Education Department CSTEP program.
The authors would also like to thank the Honors Program staff and faculty for their mentorship and advisement.

Contact Information

Contact me at Harrison.fernandez@jjay.cuny.edu.