Understanding and Detecting Concurrency Attacks

Paper #82

The University of Hong Kong
@cs.hku.hk

Abstract

1. Introduction

Multi-threaded program is hard to be correct. Concurrency bugs are common in modern multi-threaded programs[3, 4, 8, 9] including atomic violation, ..., and especially data race. Extant work well explore interleaving that cause concurrency bugs, and efficiently detect explicit concurrency bugs that direct to severe consequences such as execution order violation, wrong output and program crash.

Recent studies[5, 7] show rise of concern about *concurrency attacks*. Just like sequential bugs leading to attacks, concurrency bugs also lead to concurrency attacks. By triggering concurrency bugs and employing subtle inputs, hackers may leverage the corrupted memory to conduct attacks including privilege escalations[], hijacking code execution[], bypassing security checks[], and breaking database integrity[5]. These vulnerabilities often hide in large amount of concurrency bug reports. For example, bug information leveraged by a xxx attack is hidden in 1000 race reports produced by TSAN[6], a famous and widely used data race detector. Also, despite threads conducting concurrency bugs, behaviors of another thread may also be affected when data race bugs infecting shared memory (e.g.heap overflow)[1?].

Unfortunately, although great progress has been made to detect and replay severe bugs(e.g.ConMem[9]), extant work still lacks exploration of concurrency attacks from enormous concurrency bugs. Our study over several known concurrency bugs[1, 2] shows that even concurrency bugs have been successfully detected and reported, professionals may still lack knowledge about how severe consequences these bugs may cause. For instance, *apache-25520*[1] has been reported over years and well studied by researchers[4]. We are the first to exploit a new heap overflow attack leveraging on this bug and break the HTML integrity.

We studied 26 attacks and find two major challenges for exploring concurrency attacks from concurrency bugs. First, concurrency attacks may be implicit and hidden in common concurrency bugs. Existing concurrent bug detectors focus on bug happening itself. ConMem[9] first propose to consider concurrency bugs that may cause severe consequences (e.g.program crash) and ConSeq[8] uses severe consequence report to help diagnose concurrency bugs. However, our observation shows that explicit error (e.g.program crash) is not necessary for concurrency attacks. Some concurrency bugs (e.g.*xxx-xxx*) may not cause program crash or interrupted, but attackers may still be able to conduct attacks with crafted input. Worse still, crash bugs may even lead to more server vulnerabilities. In *CVE-2017-7533* conducted by our team, although the data race primarily causes kernel crash, we crafted the input and successfully conduct a privilege escalation attack without crashing the kernel. Current concurrency bug detecting tools are not designed to analyze this kind of latent vulnerabilities, and hence may direct wrong level of warnings towards the bugs.

Second, extant work ignores indicating the *victim thread* of concurrency attacks. A concurrency bug may become much more vulnerable when attackers employee another thread to construct their attacks. In CVE-2017-7533, we do not only leverage two threads to trigger a data race and construct kernel heap overflow, but also require another *victim thread* to lay the target structure on the same heap. By crafting inputs and corrupt the target structure, we finally achieve arbitrary code execution and get a root shell. Automatically indicating the victim thread would be of vital helpful for developers to better understand the latent vulnerabilities. For example, in the Apache-25520 case, after knowing about information of victim thread, we successfully increased the severity of the bug and conducted an integrity violation.

To address the two challenges, we introduce a new model covering all concurrency attacks we studied.

2. Background

3. Overview

4. Reducing Benign Schedules

5. Discussions

6. Evaluation

7. Related Work

[]

8. Conclusion

References

[1] Apache bug 25520. `https://bz.apache.org/bugzilla/show_bug.cgi?id=25520`.

[2] Apache bug 46215. `https://bz.apache.org/bugzilla/show_bug.cgi?id=46215`.

[3] S. Lu, S. Park, C. Hu, X. Ma, W. Jiang, Z. Li, R. A. Popa, and Y. Zhou. Muvi: automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pages 103–116, 2007.

[4] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug character-istics. In *Thirteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '08)*, pages 329–339, Mar. 2008.

[5] P. B. Todd Warszawski. Acidrain: Concurrency-related attacks on database-backed web applications. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, pages 5–20. ACM, 2017.

[6] Threadsanitizer. `https://code.google.com/p/data-race-test/wiki/ThreadSanitizer`, 2015.

[7] J. Yang, A. Cui, S. Stolfo, and S. Sethumadhavan. Concurrency attacks. In *the Fourth USENIX Workshop on Hot Topics in Parallelism (HOTPAR '12)*, June 2012.

[8] W. Zhang, J. Lim, R. Olichandran, J. Scherpelz, G. Jin, S. Lu, and T. Reps. ConSeq: detecting concurrency bugs through se-quential errors. In *Sixteenth International Conference on Ar-chitecture Support for Programming Languages and Operating Systems (ASPLOS '11)*, pages 251–264, Mar. 2011.

[9] W. Zhang, C. Sun, and S. Lu. ConMem: detecting severe con-currency bugs through an effect-oriented approach. In *Fifteenth International Conference on Architecture Support for Program-ming Languages and Operating Systems (ASPLOS '10)*, pages 179–192, Mar. 2010.