Understanding and Detecting Bugs in Network Operating Systems

Penghui Li 1155137827
phli@cse.cuhk.edu.hk

ABSTRACT

Network systems connect the Internet world. Bugs in network systems can lead to denial-of-service, performance issues, *etc.* Understanding network system bugs can benefit both the system developers and security analysts.

However, to the best of our knowledge, the bugs in network systems have not been well understood. The potential security impact has not been well investigated as well. In this project, we aim to understand the bugs in network systems. We hope to learn lessons to further guide future bug detection in network systems.

1 INTRODUCTION

Network Operating Systems (NOSs) connect multiple computers and devices and manage network resources in the settings of Software Defined Networking (SDN). NOSs manage multiple requests concurrently and provide the security in multiuser environments. To date, there are many NOSs being proposed and matained by network service vendors. For example, Cisco Internetwork Operating System is a family of network operating systems used on many Cisco Systems routers and current Cisco network switches [4]; DD-WRT [1] is a Linux based open-sourced NOS suitable for various WLAN routers and embedded systems.

Due to their popularity and critical uses, the bugs in these network operating systems can lead to severe security consequences. In particular, network operating systems could have performance bugs, which cause excessive resource consumption and could negatively affect user experiences. They can be further leveraged by attackers for launching application-layer denial-of-service (DoS) attacks [3]. By specially crafting inputs to trigger a performance bug in the network operating systems, the attacker can exhaust the server's resources (*e.g.*, memory and CPU) and make the services unavailable to normal users.

To the best of our knowledge, there currently has little knowledge about the prevalence of performance bugs in the wild. Prior studies on compilers generally focus on memory corruptions, whereas performance bugs, especially in close-sourced network operating systems, are not well investigated and understood yet. Some works studied performance issues in the regular expression engines [11, 13], desktop software [5], and Android applications [7]. Network operating systems, however, have not been covered. To this end, we conduct a comprehensive study of the performance bugs in network operating systems to answer the following research questions:

- What are the main categories of the bugs?
- What are the main factors that cause bugs?
- How widespread are the bugs in network operating systems?
- How severe are the bugs in network operating systems?

We empirically analyze XXXknown performance bugs in mainstream network operating systems and thoroughly summarize their characteristics. We observe that there has been a continuous growth in the number of reported performance bugs in the past 3 years. We further identify that the ways that network operating systems handle the language's context-sensitive features are the dominant root cause of the performance bugs. We reveal that the developers of network operating systems usually mitigate such exploitation by enforcing a hard limit for the maximum number of backtracking for certain tasks, which, however, limits the intended functionality of the network operating systems.

In this work, we also explore detecting performance bugs in network operating systems. To the best of our knowledge, there is not any tailored tools specially designed for detecting performance bugs in network operating systems. Fuzzing [2, 6, 10] eliminates the limitations of static analysis techniques (*e.g.*, high false positives) [5, 8, 9]. It has become the go-to approach with thousands of vulnerabilities in real-world software. Without a doubt, performance bugs in network operating systems can be fuzzed as well. However, existing fuzzers [6, 10] for performance bugs are not aware of special *contex* of network operating systems and cannot effectively generate inputs to thoroughly exercise them. Therefore, besides the bit/byte level mutation [14], we summarize the rules of the inputs. We develop a syntax-tree based mutation strategy [12] to preserve and extend useful rules during the input mutation. This helps to generate inputs more effectively.

We mainly focus on CPU exhaustion performance bugs. To detect them, we monitor the program execution under the

CSCI '5570, Spring 2021, CUHK
2021.

generated inputs. We employ a statistical model using Chebyshev inequality to label abnormal cases as performance bugs. This can potentially cause duplicate bug reports as multiple bug reports with only slight difference in the inputs actually trigger the same bug. It is time-consuming and impractical to leverage human efforts to manually de-duplicate them. Existing bug de-duplicating methods using coverage profile and call stacks are inaccurate and not applicable to performance bugs. For example, it is hard to obtain an accurate and deterministic call stack that reveals the situation when a performance bug is triggered. Therefore, we propose a execution trace similarity comparison algorithm to de-duplicate the reports. Specifically, we represent the execution trace of each report into a vector; we compute the cosine similarity between vector pairs and classify vectors (bug reports) with high similarity as the same bug.

We integrate abovementioned techniques into NetPerfFuzz. We demonstrated NetPerfFuzz outperformed the state-of-the-art works [6, 10] *better* performance slowdown, and *higher* code coverage. NetPerfFuzz is highly effective in generating test cases to trigger new code in the testing network operating systems. Though we have not identified any new bugs yet after 6-hour testing, we plan to further keep NetPerfFuzz running for more time. Hopenfully, our techniques can help find new bugs in the future.

In summary, we make the following contributions in this work.

- We conduct a systematic study on performance bugs in network operating systems. We present an empirical understanding of the performance bugs and the patches.
- We develop a new system, NetPerfFuzz, to effectively generate useful inputs to detect performance bugs in network operating systems.
- We demonstrate NetPerfFuzz can significantly outperform the state-of-the-art works.

2 CONCLUSION

In this work, we conduct a comprehensive literature review of TCP incast control. We present an in-depth exploration of existing solutions to tackle TCP incast congestion problem. We categorize the solutions by their implemented network architecture layers and outline the general challenges of the problem. We show that the feedback mechanism is a practical approach to monitor the network traffic and prevent TCP incast congestion. We probe several limitations in prior works and introduce potential future work on this topic. Though well studied, there are still potential directions that can be further explored to improve the state-of-the-art works. We do hope this study can shed some light on future research of TCP incast control.

REFERENCES

[1] . 2021. DD-WRT. https://dd-wrt.com/.

[2] William Blair, Andrea Mambretti, Sajjad Arshad, Michael Weissbacher, William Robertson, Engin Kirda, and Manuel Egele. 2020. HotFuzz: Discovering Algorithmic Denial-of-Service Vulnerabilities Through Guided Micro-Fuzzing. In *Proceedings of the 2020 Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[3] Scott A. Crosby and Dan S. Wallach. 2003. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of the 12th USENIX Security Symposium (Security)*. Washington, DC.

[4] Bradley Edgeworth, Aaron Foss, and Ramiro Garza Rios. 2014. *IP routing on Cisco IOS, IOS XE, and IOS XR: an essential guide to understanding and implementing IP routing protocols*. Cisco Press.

[5] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. 2012. Understanding and detecting real-world performance bugs. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. Beijing, China.

[6] Caroline Lemieux, Rohan Padhye, Koushik Sen, and Dawn Song. 2018. PerfFuzz: automatically generating pathological inputs. In *Proceedings of the 27th International Symposium on Software Testing and Analysis (ISSTA)*. Amsterdam, Netherlands.

[7] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2014. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. Hyderabad, India.

[8] Adrian Nistor, Po-Chun Chang, Cosmin Radoi, and Shan Lu. 2015. Caramel: Detecting and fixing performance problems that have non-intrusive fixes. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*. Florence, Italy.

[9] Adrian Nistor, Linhai Song, Darko Marinov, and Shan Lu. 2013. Toddler: Detecting performance problems via similar memory-access patterns. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. San Francisco, CA.

[10] Theofilos Petsios, Jason Zhao, Angelos D. Keromytis, and Suman Jana. 2017. SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. Dallas, TX.

[11] Yuju Shen, Yanyan Jiang, Chang Xu, Ping Yu, Xiaoxing Ma, and Jian Lu. 2018. Rescue: Crafting regular expression dos attacks. In *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Montpellier, France.

[12] Junjie Wang, Bihuan Chen, Lei Wei, and Yang Liu. 2019. Superion: Grammar-aware greybox fuzzing. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. Montréal, Canada.

[13] Valentin Wüstholz, Oswaldo Olivo, Marijn JH Heule, and Isil Dillig. 2017. Static detection of DoS vulnerabilities in programs that use regular expressions. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Uppsala, Sweden.

[14] Michal Zalewski. 2021. american fuzzy lop. https://github.com/google/AFL.