Poster: Automatic Network Protocol Fingerprint Discovery with Difference-Guided Fuzzing

Yuxi Zhu*, Hanyi Peng*, Jiahao Cao†, Renjie Xie†, Xinda Wang‡, Mingwei Xu*†, Jianping Wu*†
*Department of Computer Science and Technology, Tsinghua University, Beijing, China
†Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
‡Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA
{zyx23@mails, peng-hy21@mails, caojh2021@, xrj21@mails, xumw@}.tsinghua.edu.cn, xinda.wang@utdallas.edu, jianping@cernet.edu.cn

***Abstract*—Network protocol fingerprinting is a critical technique for identifying various implementations of network protocols, which is essential for vulnerability assessment and security management. However, current fingerprinting methods such as Nmap still heavily rely on manual probe crafting, requiring experts with domain knowledge and leading to inefficiencies and potential oversights. This paper introduces pFuzz, an automatic network protocol fingerprint discovery system utilizing difference-guided fuzzing, to address the challenge of the vast search space inherent in fingerprinting. We propose a difference tree to model the nested recursive condition structure of network protocols and a packet oracle map to capture and utilize multi-field relationships revealed by value co-occurrence. Our evaluation of pFuzz on the widely used TCP/IP protocol demonstrates its effectiveness and efficiency on discovering fingerprints.**

***Index Terms*—Fingerprint Discovery, Network Protocol, Fuzzing, Difference Tree**

I. Introduction

Fingerprinting is a technique that can identify different implementations of the same network protocol specification. Different implementations may exhibit varying behaviors under specific conditions, even when following the same specification. This variance often arises from ambiguities in the specification or due to engineering compromises for implementation ease [1]. As a result, malicious attackers can use fingerprinting to select suitable targets or tailor their attacks appropriately [2]. Popular vulnerability exploitation frameworks, such as Metasploit [3], incorporate fingerprinting tools to detect specific implementation vulnerabilities. Furthermore, fingerprinting assists administrators in profiling networks and identifying risky protocol implementations that are vulnerable.

As a fundamental working paradigm, a network protocol fingerprinting tool dispatches one or more probes to a target implementation, analyzes the responses, and matches the target to one of the candidate profiles. The probes, as the initial and most crucial aspect of fingerprinting, significantly affect the quantity of detectable fingerprints and the efficiency of identifying these fingerprints across various implementations of the same protocol. Current tools like Nmap [4] for TCP/IP fingerprinting, and fpdns [5] for DNS fingerprinting, along with existing studies [6], [7], utilize manually-crafted probes. These require experts with domain knowledge in protocols for their design, making them labor-intensive and difficult to adapt to new protocols or updates in existing protocols. Furthermore, human designers might overlook some complex or corner cases. Although some research efforts [1], [8] have attempted to automate some aspects of the fingerprinting process, they continue to depend significantly on fields selected by humans, and overlook the the relationship between fields, thus suffering low degree of automation and low quality of fingerprints.

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

Automating the probe design process can free up manpower and potentially enhance the quality of the fingerprints while identifying more distinct fingerprints. A simple approach is to send all possible probes to various implementations of the same protocol and select those that effectively differentiate them. However, the immense volume of potential probes makes this approach impractical; for instance, a simple 20-byte TCP header probe alone offers over 1.46×10^{48} possibilities.

In this paper, we propose pFuzz, an automatic network protocol fingerprint discovery system with difference-guided fuzzing to overcome the challenge posed by the large search space. It leverages heuristic information derived from responses of the target implementations to guide probe generation, thereby enhancing the ability to efficiently generate probes with strong discrimination capabilities. Specifically, we develop a difference tree that reflects the nested recursive condition structure of network protocol implementations, enabling a more detailed exploration of variations at different levels. Given that some fingerprints involve complex conditions related to multiple fields, we introduce a packet oracle map to capture and leverage the multi-field relationships revealed by value co-occurrence. We evaluate pFuzz's effectiveness in identifying TCP/IP fingerprints across various protocol stack implementations in different operating systems. Preliminary results show that pFuzz identified around 52 discriminating fingerprints, exceeding the 34 human-crafted fingerprints found by Nmap [4] and the 24 identified by SinFP [7]. We also demonstrate the efficiency of pFuzz in discovering fingerprints.

II. Design

As shown in Figure 1, the core functionality of pFuzz operates within a loop structure. In each cycle, a carefully

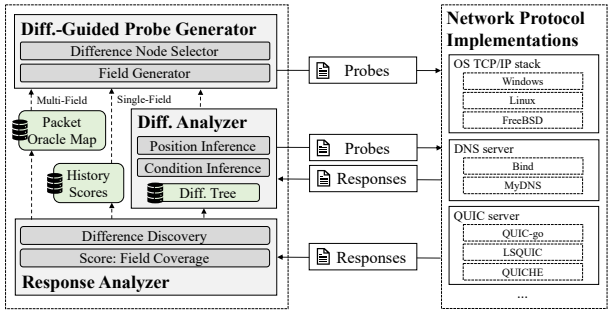

Fig. 1. The architecture of pFuzz.

crafted probe is automatically generated and dispatched to the target network protocol implementations, with responses subsequently collected. The differences in the responses are analyzed and scored to direct the generation of probes in subsequent cycles, facilitating the efficient discovery of fingerprints. Throughout the entire process, the only human intervention required is the initial input of the protocol packet format. Specifically, pFuzz consists of the following modules:

Difference-Guided Probe Generator. The probe generator fills every field to create a probe, guided by previously identified differences in responses. In other words, it tries to find new fingerprints from identified differences, which are often related. As network protocols are typically implemented as nested conditional statements, relationships among differences have similar patterns. To map and record these relationships, we thus introduce a difference tree. The difference tree comprises all known differences, where a difference node is a child of another if its condition is stricter. When a probe triggers a difference, all its ancestor nodes must have been triggered as well. Thus, a difference node can potentially represent a specific code block in the implementations. Using the difference tree, probe generation is as follows: first, a difference node is selected, setting certain fields based on predefined conditions. Remaining flexible fields are determined using heuristics. For enumerable fields, historical scores guide the selection of values by calculating the upper confidence limit. For fields with numerous options, mutation of previously selected values is employed. This method facilitates thorough exploration of each code block represented by a difference node, with historical data recorded and analyzed per node.

Response Analyzer. The response analyzer identifies new differences and scores each round of exploration. When responses are received, the analyzer compares the differences found with known ones. If an unknown difference is detected, the difference analyzer is activated. Otherwise, the response analyzer assigns a score reflecting the coverage of the target implementations. Since the implementations are treated as black boxes, response field values serve as a reference. Probes that trigger rare response values receive higher scores. These scores are stored as historical data to guide the difference-based probe generator. And if a score is the highest achieved by a specific field value, the corresponding probe is stored in the packet oracle map to capture multi-field relationships.

Difference Analyzer. The difference analyzer determines the position and condition of a new difference. To identify the position, it sends probes that either satisfy or don't satisfy the conditions of known differences, observing whether the difference persists. Next, the condition is established by sequentially altering each field to a random value to see if it affects the newly found difference. The broadest set of fields that can trigger the difference is then defined as the condition.

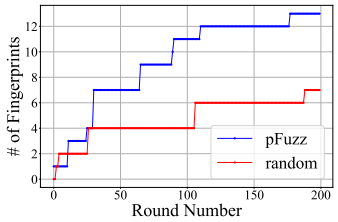

Fig. 2. The number of fingerprints identified over running rounds.

III. Preliminary Evaluation

We evaluate pFuzz's ability to identify TCP/IP fingerprints across different protocol stack implementations in Windows 11, Ubuntu 22.04, and FreeBSD 14. Table I shows pFuzz identifying 52 fingerprints, significantly exceeding the 34 human-crafted fingerprints in Nmap [4] and the 24 in SinFP [7]. By replacing the difference-guided probe generator with a random one, Figure 2 shows that pFuzz continues to discover new fingerprints more rapidly than the random method.

TABLE I
The Number of Fingerprints

Tool	`pFuzz`	`Nmap` [4]	`SinFP` [7]
# of Fingerprints	52	34	$\approx$ 24

Acknowledgment

The research is supported by the National Natural Science Foundation of China (NSFC) under Grant 62221003, 62202260, and 623B2062 (except Xinda Wang). Jiahao Cao and Mingwei Xu are the corresponding authors of the paper.

References

[1] J. Caballero, M. G. Kang, S. Venkataraman, D. Song, P. Poosankam, and A. Blum, "FiG: Automatic Fingerprint Generation," 2007.

[2] "Ms08_067 python exploit script - updated 2018," accessed on June 30, 2024. [Online]. Available: https://github.com/andyacer/ms08_067

[3] "Metasploit," accessed on June 30, 2024. [Online]. Available: https://www.metasploit.com/

[4] "Nmap," accessed on June 30, 2024. [Online]. Available: https://nmap.org/

[5] "fpdns," accessed on June 30, 2024. [Online]. Available: https://github.com/kirei/fpdns

[6] E. Kollmann, "Chatter on the Wire: A look at DHCP traffic," Aug. 2014. [Online]. Available: http://chatteronthewire.org/download/chatter-dhcp.pdf

[7] P. Auffret, "SinFP, unification of active and passive operating system fingerprinting," *Journal in Computer Virology*, vol. 6, no. 3, pp. 197–205, Aug. 2010. [Online]. Available: http://link.springer.com/10.1007/s11416-008-0107-z

[8] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New Directions in Automated Traffic Analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. Virtual Event Republic of Korea: ACM, Nov. 2021, pp. 3366–3383. [Online]. Available: https://dl.acm.org/doi/10.1145/3460120.3484758