INVESTIGATING THE PERFORMANCE OF FUZZERS ON WIREGUARD-GO

OVERVIEW

Businesses, universities, and governments rely on networks to transport critical confidential data. VPNs are widely used for network security as they provide confidentiality, authentication, and integrity.

WireGuard, a modern and light-weight alternative to older VPN protocols, has gained considerable popularity.

Fuzzers are a popular and effective tool for discovering vulnerabilities in software, however, there is a lack of systematic studies on fuzzing implementations of VPNs.

BACKGROUND

Fuzzers can be classified into 4 different categories. These classifications are not mutually exclusive.

Test Case Feedback	*Knowledge of Application Structure*	*Test Case Generation Method*	*Program Exploration Approach*
Dumb Does not use the information gathered from program execution to update test case generation	**White-box** Full awareness of code structure and execution state	**Random** Test cases are randomly generated	**Coverage-based** Attempts to test as much as the code as possible
Smart Uses the information gathered from program execution to update test case generation	**Grey-box** Partial awareness of code structure and execution state	**Mutation-based** Creates test cases by modifying a set of valid inputs	**Directed** Attempts to test certain parts of the code
	Black-box No awareness of application structure or execution state	**Generation-based** Creates test cases based on a model of what valid program input is	

OBJECTIVES

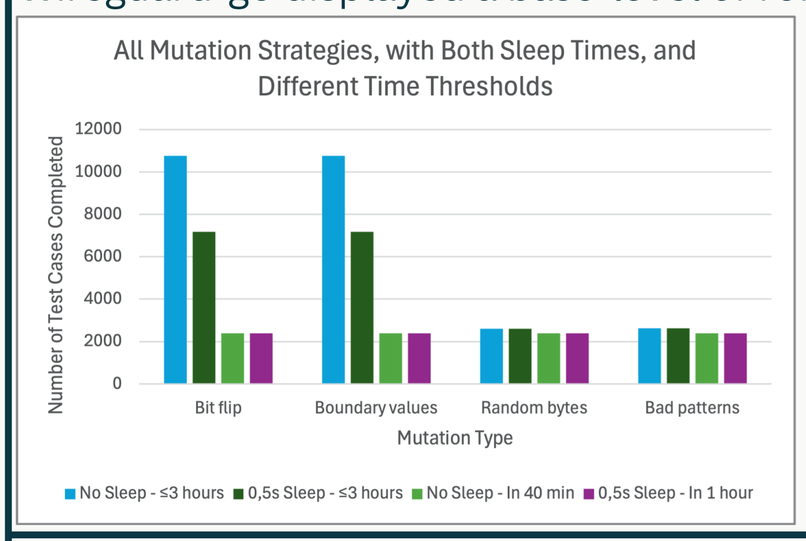
- Investigate the feasibility of using fuzzers to test Wireguard-go.
- Investigate how parameters, such as persistent connections, mutation strategies, and numbers of fields impact the efficiency and performance of Boofuzz and Peach.

METHODOLOGY

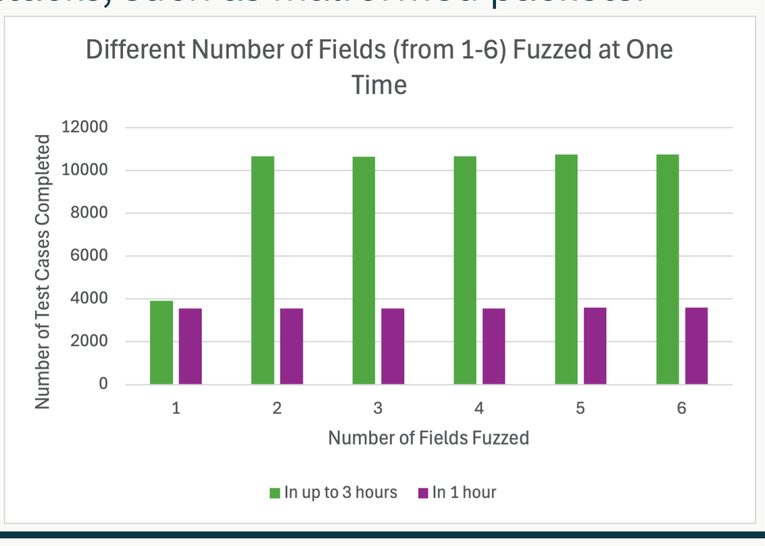
- Two different fuzzers – Boofuzz and Peach – were used to fuzz “Wireguard-go”.
- Different experiments were run to investigate the effects that predominant parameters for each fuzzer had on its performance by recording the number of iterations executed in a given time period.
- The parameters investigated were: Mutation strategy, maximum fields to mutate at a time, the number of fields to mutate at a time, switch count, and persistent vs non-persistent execution of the target program (Wireguard-go).

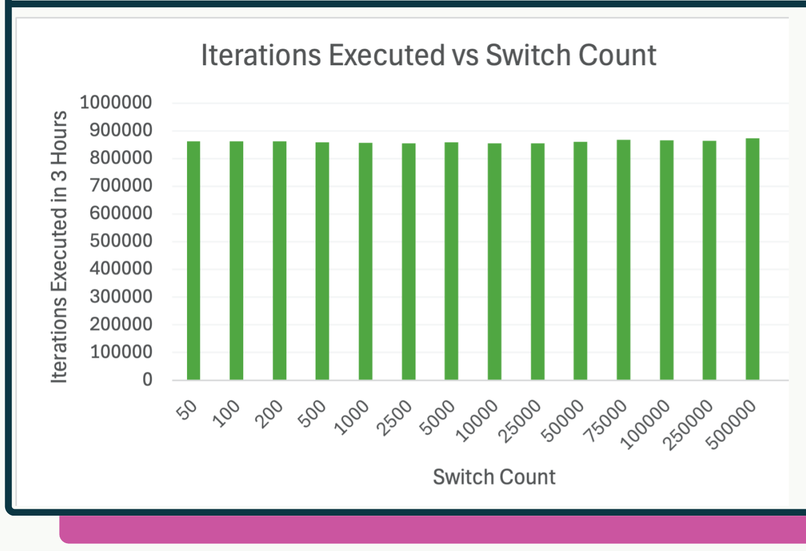
KEY FINDINGS

Wireguard-go displayed a base-level of robustness against common mistakes and attacks, such as malformed packets.

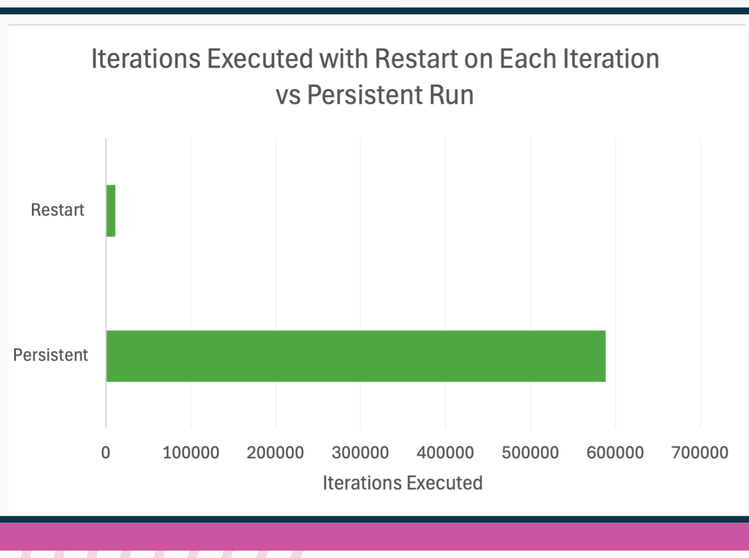


While the number of fields fuzzed, the maximum fields mutated per iteration, and the mutation strategies used did influence performance, the performance impact was minimal. Therefore, the impact of these parameters on performance is not sufficient to dictate decision-making when designing tests.





Switch count had no effect on performance. The variation between test cases was within the range of naturally occuring variation.



The most notable finding was that restarting the process on each iteration incurs significant overhead.

CONCLUSIONS

- Increasing the number of fields simultaneously mutated reduces performance, introducing a trade-off between test case coverage and overall performance.
- Restarting the process on each iteration should be avoided wherever possible, as it adds significant overhead.
- While Peach is an effective tool, it has not been maintained. Therefore, newer fuzzers that are compatible with updated software tools should be considered first, as they are better documented and would be less challenging to set up.
- Boofuzz is a generation-based fuzzer, which can be difficult to use. However, it provides better coverage, is well-documented, and is well suited for network protocol fuzzing.

Michal Sacks – sckmic006@myuct.ac.za
Thalia Hawthorn – hwttha001@myuct.ac.za

Supervised by: Prof. Josiah Chavula (UCT)
Marco Slaviero (Thinkst Canary)

SCHOOL OF IT