Automated Vulnerability Testing and Detection Digital Twin Framework for 5G Systems

Danielle Dauphinais* Michael Zylka* Harris Spahic* Farhan Shaik*
Jingda Yang* Isabella Cruz* Jakob Gibson* Ying Wang*
*Stevens Institute of Technology, Hoboken, NJ Email: *{ddauphin, mzylka, hspahic,fshaik, jyang76, icruz1, jgibson1, ywang6}@stevens.edu

2023 IEEE 9th International Conference on Network Softwarization (NetSoft) | 979-8-3503-9980-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/NetSoft57336.2023.10175496

***Abstract*—Efficient and precise detection of vulnerabilities in 5G protocols and implementations is crucial for ensuring the security of its application in critical infrastructures. However, with the rapid evolution of 5G standards and the trend towards softwarization and virtualization, this remains a challenge. In this paper, we present an automated Fuzz Testing Digital Twin Framework that facilitates systematic vulnerability detection and assessment of unintended emergent behavior, while allowing for efficient fuzzing path navigation. Our framework utilizes assembly-level fuzzing as an acceleration engine and is demonstrated on the flagship 5G software stack: srsRAN. The introduced digital twin solution enables the simulation, verification, and connection to 5G testing and attack models in real-world scenarios. By identifying and analyzing vulnerabilities on the digital twin platform, we significantly improve the security and resilience of 5G systems, mitigate the risks of zero-day vulnerabilities, and provide comprehensive testing environments for current and newly released 5G systems.**

***Index Terms*—5G Security, Testing Framework, Digital Twin, Assembly-Level, Fuzzing**

I. Introduction

5G promises to provide multi-dimensional improvements that enable a plethora of communication and services across various verticals. However, the trend towards softwarization and virtualization, coupled with the high demand for open programmable 5G and Future G standards and stacks, necessitates rigorous testing for infrastructure and network security assurance against vulnerabilities.

Fuzz testing, a quantitative approach that randomly changes or generates all possible input cases, has been used to detect vulnerabilities in current and next-generation communication systems. This approach involves generating and injecting unexpected inputs, referred to as ”fuzzed” inputs, into a system by slightly modifying a base input and observing its effects. Existing works, such as protocol-based fuzzing [1] and rule-based fuzzing [2], have been proposed to detect vulnerabilities in protocols and implementations via fuzzing systems. Industry-level fuzzing platforms, such as Google's OSS-Fuzz [3] and Microsoft's OneFuzz [4], provide continuous fuzzing and fuzz-as-a-service for software testing. However, scalability and learning ability still pose challenges for current systems. Additionally, current fuzzing system designs rely solely on one or multiple defined fuzzing methodologies and eliminate interactions with other approaches that could compensate for the computational complexity of fuzz testing.

This paper aims to improve 5G security and resilience with sufficient automation and scalability. Building on our previous work [5] that demonstrated the feasibility of protocol-independent fuzz testing, the proposed digital twin further enhances the scalability of fuzz testing by innovatively introducing an assembly-level accelerator engine and allowing inter-system data analysis for more accurate risk assessment through standardizing interfaces and semantically searchable querying databases. Our digital twin prototype supports both centralized and distributed deployment, providing use case flexibility. Since the significant impacts of fuzz testing and potential intelligence to be developed greatly depend on the scale of data acquired, a distributed platform with seamless connectivity and integration are essential for 5G vulnerability detection, ultimately allowing zero-trust zero-touch 5G networks.

In summary, we propose a novel demonstration of a fuzz testing digital twin with sufficient automation and higher scalability. The major contributions of our system can be categorized as follows:

1) We have enabled an autonomous fuzzing digital twin that integrates command level and bit level fuzzing for 5G and future G, with an interface to other non-fuzzing systems. It can serve as a comprehensive testing system for large-scale software systems and a systematic vulnerability detection and defense system for real-world attack models.
2) By analyzing the significant-identifier based fuzzing data, our digital twin platform identified vulnerabilities in 5G RRC and demonstrated the feasibility of an efficient and effective vulnerability detection system for released or new release standards and specifications in cellular communication systems.
3) The assembly-level acceleration engine significantly improves the cycle of fuzz testing and enables large-scale fuzzing that fundamentally improves the scale of intelligence.
4) The flexibility of the centralized and distributed deployment structure and the supporting database allows collaboration across physical locations and time.

II. System Design

Our goal is to design and implement a 5G fuzz testing platform as a use-case strategy to complete the proving circle, by revealing the consequences of unintended behaviors in 5G systems. We integrate fuzz testing into a hybrid formal methods design, guided by its results to describe unintended emergent behaviors and evaluate their consequences under various attack environments. Figure 1 shows an overview of the framework. Based on the assumptions detected by the hybrid formal methods, test cases are created to violate the assumptions fed into the system for data analysis. The data will also be used to build intelligent machine learning-based models that can detect patterns for various software stack implementations. The formal methods specification created in a flagship stack can be replicated to multiple implementations and stacks.

To start the tester, we first provide a list of elements to be fuzzed, which are then initialized in the command queue. The program will then sequentially run each fuzz test, and the results of each test will be logged in a database. The results can be displayed in a graphical user interface. Using a method of informed prediction, such as our example formal methods, a script will self-generate additional tests on high vulnerability areas of the code.

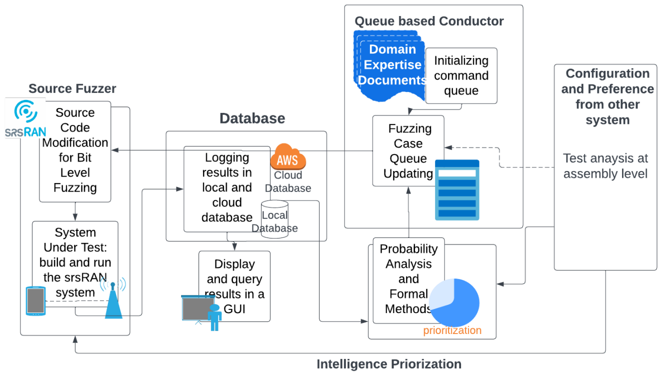

Queue based Conductor
Domain Expertise Documents
Initializing command queue
Source Fuzzer
Source Code Modification for Bit Level Fuzzing
srsRAN
Database
Logging results in local and cloud database
AWS Cloud Database
Local Database
Fuzzing Case Queue Updating
Configuration and Preference from other system
Test anaysis at assembly level
System Under Test: build and run the srsRAN system
Display and query results in a GUI
Probability Analysis and Formal Methods
prioritization
Intelligence Priorization

Fig. 1. The framework of Automomous Fuzzing Digital Twin

A. Source Fuzzer

The source fuzzer is a platform designed to automatically perform bitfuzzing tests on any 5G codebase, modularly integrating a host of search methods (statistical or formal) to continuously generate targeted commands for the code base, free of user input. The source fuzzer is split into three pieces: the "Fuzz Element" class, the "Source Element" class, and global accessible methods. The "Fuzz Element" class acts as a container for each fuzz command. It contains the location of the value to be changed (file name in the srsRAN repository, line number with character offset), the fuzz values to be injected, a copy of the original file for recovery purposes, and the methods necessary to inject the fuzz values into the code.

The "Source Element" class acts as a container for the main execution of our program. It contains a queue of "Fuzz Element" commands to be run, the methods necessary to initialize and run the queue, as well as the control structure of how each executed command will be run. An executed command will inject the current value to be fuzzed, build the 5G protocol, establish a 5G connection, and log the results to the database. We achieve this task through a combination of internal method calls and external global function calls.

B. Database

Each test that the source fuzzer executes results in a set of output files. We perform some preliminary parsing on these files to extract useful high-level information, and then store the extracted information and raw files in a database. An overview of this process is shown in Fig. 2. For our use case, we implemented a private 'local' database as well as a cloud 'central' database. The 'local' database instance runs on the same server that is hosting a source-fuzzer instance. This database stores the parsed information in a local PostgreSQL server and stores the raw files as a file store. The cloud database runs on Amazon Web Services (AWS) and uses AWS RDS to store Structured Query Language (SQL) data, and stores raw files in an AWS S3 bucket. The database implementation is shown in Fig. 3. We chose to use this multiple database system for a couple of reasons: first, the local database has a low *time to deploy* and is easiest to reproduce; second, the cloud database solves the scalability issue of a private database; third, we can deploy multiple servers, each executing a source-fuzzer instance as well as a local database, that all pipe into a single central database.

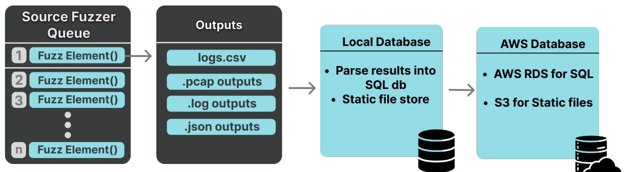


Fig. 2. The process from fuzz element to database

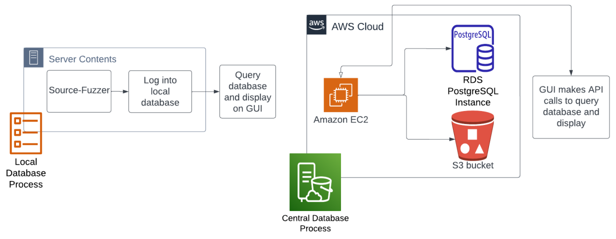


Fig. 3. The database implementation

C. Test Analysis at Assembly Level

We have developed a path analyzer that generates semantic dynamic call graphs while analyzing the execution of a process. It is capable of reading memory at any point during execution time and comparing dynamic traces between different fuzzing cases. The path analyzer leverages radare2 to gather information about the executable and debug the program while monitoring function calls and returns. Figure 4 illustrates the relationship among Path Analyzer, radare2, and the executable file. Path Analyzer interacts with radare2 through its API r2pipe, and radare2 forks and attaches itself to the srsRAN executable.

During the process execution, Path Analyzer outputs the function calls and their returns. Figure 5 shows an example of the output generated by Path Analyzer on srsepc, depicting the control flow of the first two functions that were executed from main, as well as the functions called inside the signal handler function. This allows for a dynamic analysis of the program and its control flow.

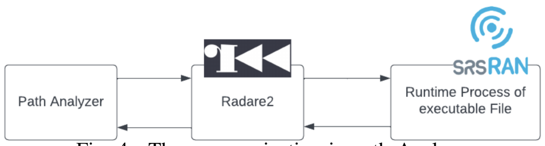
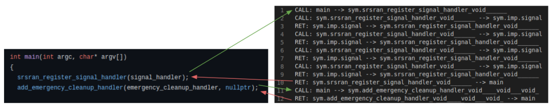


Fig. 4. The communication in path Analyzer



```
int main(int argc, char* argv[])
{
  srsran_register_signal_handler(signal_handler);
  add_emergency_cleanup_handler(emergency_cleanup_handler, nullptr);
```

```
CALL: main --> sym.srsran_register_signal_handler_void______
CALL: sym.srsran_register_signal_handler_void______ --> sym.imp.signal
RET: sym.imp.signal --> sym.srsran_register_signal_handler_void______
CALL: sym.srsran_register_signal_handler_void______ --> sym.imp.signal
RET: sym.imp.signal --> sym.srsran_register_signal_handler_void______
CALL: sym.srsran_register_signal_handler_void______ --> sym.imp.signal
RET: sym.imp.signal --> sym.srsran_register_signal_handler_void______
CALL: sym.srsran_register_signal_handler_void______ --> sym.imp.signal
RET: sym.imp.signal --> sym.srsran_register_signal_handler_void______
RET: sym.srsran_register_signal_handler_void______ --> main
CALL: main --> sym.add_emergency_cleanup_handler_void___void__void_
RET: sym.add_emergency_cleanup_handler_void___void__void_ --> main
```

Fig. 5. Comparison of srsepc main source code and dynamic call graph

III. Implementations and Results

Our pilot 5G software stack srsRAN, works as our system prototype used emulate a 5G network, consisting of a virtualized simulation of network, ZeroMQ and a live implementation. Further information on ZeroMQ can be found in its documentation [6]. Within the srsRAN codebase, three items must establish a connection: srsUE connects with srsENB, which then connects with srsEPC. For detailed documentation on the srsRAN architecture and setup, please refer to its documentation [7].

This digital twin serves as a means to emulate a software replica of a mobile network on a single computer. These software libraries and scripts are used to test the security of the 5G network within the framework.

The current framework provides the necessary information to execute a fuzz test and create a single fuzz element. It takes approximately one minute and fifteen seconds to complete a single test. We estimate that up to 50 tests can be performed in one hour, averaging approximately 1200 tests per day.

Graphical User Interface: A user interface (GUI) has been developed to allow ease of access to testers to view the test results easily without manually searching for and examining files on the server or writing SQL queries to the private database. The GUI enables users to query the results of tests stored in the private database using various parameters, including filename (required), line number (optional), line content (optional), test value or values (optional), and the test result (optional).

Once a user submits their query, the GUI will send a request to the private database via a function call and receive a JSON containing records of all tests that meet the given parameters. The filename, function name, line number, test value, and result of these tests will then be displayed by the GUI for the user to parse through, as shown in Fig. 6(a). The users can click on the test row to reach a page with all the information stored in the private database for that test shown in Fig. 6(b).

Demonstration and Setup: We have two main modes of running our Bit Fuzzer implementation. The first is virtual, we can run the 5G architecture using ZeroMQ, which simulates the UE, gNB, Core Network communication virtually on our server. The second is a live test, using SRSRAN's over the air 5G NSA architecture. We use a software defined radio, USRPB210, to act as our gNB which with communicate with a compatible COTS UE receiver. Once either 5G system is initialized, we run our Bit Fuzzer implementation. It will take

(a) Search results in the GUI (b) Test results in the GUI

Fig. 6. GUI Display

in a predefined set of vulnerable functions to alter, sequentially change those functions, recompile the newly edited 5G codebase and attempt to establish communication between the ENB and the UE. The ongoing communication is monitored, and the results of each test are sent to the database for future analysis. This process is repeated automatically until all given commands are completed, or the system encounters an emergency failure; at which point the user is pinged with additional details.

Our ZeroMQ setup is capable of running on any system granted the system requirements necessary to run the 5G process of interest are present. The over the air setup requires a software defined radio transmitter to act as the gNB, and a user device to act as the UE.

IV. Conclusion and Future Work

In this study, we have demonstrated a prototype of a novel fuzz-testing digital twin with sufficient automation and higher scalability. The designed and implemented digital twin platform can serve as a comprehensive testing system for large-scale 5G and future G systems and a systematic vulnerability detection and defense system for real-world attack models. In the future, the system prototype will be verified and demonstrated by collaborating and interfacing with external systems.

Acknowledgment

This effort was sponsored by the Defense Advanced Research Project Agency (DARPA) under grant no. D22AP00144. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

[1] S. Potnuru and P. K. Nakarmi, "Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G," in *International Conference on Wireless and Mobile Computing, Networking and Communications*, vol. 2021-October. IEEE Computer Society, 2021, pp. 295–300.

[2] Z. Salazar, H. N. Nguyen, W. Mallouli, A. R. Cavalli, and E. M. Montes De Oca, "5Greplay: A 5G Network Traffic Fuzzer - Application to Attack Injection," in *ACM International Conference Proceeding Series*. Association for Computing Machinery, 8 2021.

[3] Google, "Google/oss-fuzz: Oss-fuzz - continuous fuzzing for open source software." [Online]. Available: https://github.com/google/oss-fuzz

[4] Microsoft, "Microsoft/onefuzz: A self-hosted fuzzing-as-a-service platform." [Online]. Available: https://github.com/microsoft/onefuzz

[5] Jingda Yang, Ying Wang, Tuyen X. Tran, and Yanjun Pan, "5G RRC Protocol and Stack Vulnerabilities Detection via Listen-and-Learn," in *IEEE Consumer Communications & Networking Conference*, 2023.

[6] P. Hintjen. (2022) Ømq - the guide. [Online]. Available: https://zguide.zeromq.org/docs/preface/

[7] S. R. Systems. (2022) srsran 22.10 documentation. [Online]. Available: https://docs.srsran.com/en/latest/