SIMurai: Slicing Through the Complexity of SIM Card Security Research

Tomasz Piotr Lisowski, *University of Birmingham;* Merlin Chlosta, *CISPA Helmholtz Center for Information Security;* Jinjin Wang and Marius Muench, *University of Birmingham*

https://www.usenix.org/conference/usenixsecurity24/presentation/lisowski

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security '24 Artifact Appendix: SIMURAI: Slicing Through the Complexity of SIM Card Security Research

Tomasz Piotr Lisowski[†], Merlin Chlosta[‡], Jinjin Wang[†], Marius Muench[†]

[†]School of Computer Science, University of Birmingham
[‡]CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

SIM cards are commonly considered as trusted entities in mobile networks. But what if they were not? We design and implement SIMURAI, a software platform for security-focused SIM exploration and experimentation. At its core, SIMURAI features a flexible, software implementation of a SIM. In contrast to existing SIM research tooling, that typically involves a physical SIM *card*, a software implementation adds flexibility by allowing to deliberately violate application- and transmission-level behavior—a valuable asset for further exploration of SIM features and attack capabilities.

Along with our main artifact, the open-source SIMURAI implementation, we provide additional experiments to support the main claims of our paper. In particular, we demonstrate that: Smartphones can, in practice, use SIMurai to authenticate to cellular networks, that we enabled FirmWire-based fuzzing of the *proactive command* SIM functionality, and found crashes in baseband implementations.

A.2 Description & Requirements

Our main artifact is the SIMURAI platform. To replicate the experiments described in our paper, we further provide an experiment repository, that contains automated build scripts, descriptions, and files for three setups, two experiments, and two case studies. The three setups constitute common test beds in cellular security research. As such, the functionality of SIMURAI within these setups is a main feature.

Setups. These three setups showcase common ways of using SIMURAI, and are described in Section 6.1 of the paper.

(S1) Attaches SIMURAI to a COTS UE and registers the UE on our test network based on YateBTS and srsRAN. This setup requires physical hardware.

(S2) Attaches SIMURAI to a virtualized UE and network based on srsUE and srsRAN.

(S3) Connects SIMURAI to an emulated baseband firmware running inside the FirmWire emulation platform.

Experiments. Code for the main experiments conducted with SIMURAI, described in Section 6.2 of the paper.

(E1) Re-implements SIM Spyware using SIMURAI. Requires *(S1)* and physical hardware.

(E2) Launches a fuzzing campaign against emulated baseband firmware. Requires *(S3)*.

To ease the setup and replication, we provide automated Docker images for all setups and experiments.

Case Studies. The case studies are described in Section 6.3 of the paper. These are not part of the SIMurai platform and instead support the discussion of SIM-originating attacks (i.e., to assess their real-world feasibility). We do not provide automated experiment code due to *(i)* IP restrictions and *(ii)* complexity of required hardware setups. Instead, we provide packet captures as supplementary material.

(CS1) Demonstrates how a malicious SIM can send a crashing payload to a UE.

(CS2) Demonstrates how a hostile applet is installed over the air and sends the malicious payload to the UE.

A.2.1 Security, Privacy, and Ethical Concerns

We utilize Docker for sandboxing compilations and, in some cases, entire experiments. As the artifact relies on a number external dependencies, we suggest to exercise caution. *S1* requires a COTS UE, an SDR and a SIMtrace2. Please follow local regulations regarding the use of the local RF spectrum, change our configurations accordingly, and prevent leakage of electromagnetic fields from the setups. Lastly, for the fuzzing experiment, `afl-system-config` is run to optimize the experimentation host for fuzzing. *This lowers the security of the system*, so please ensure that this is acceptable for the system used for experimentation.

A.2.2 How to Access

- The SIMURAI implementation is available at: https://github.com/tomasz-lisowski/simurai/tree/usenixsec2024-ae

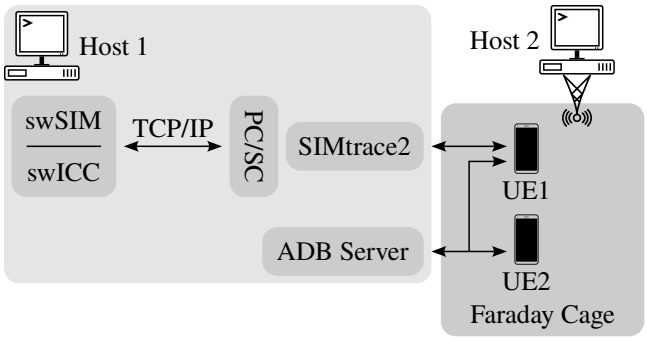

Figure 1: Overview of the physical setup (S1). SIMURAI runs on Host 1 and is connected to UE1. UE2 uses sysmoISIM-SJA2, and both UEs can be reached via ADB. The cellular test network (2G/4G/5G) is run by Host 2.

- Reproducible setups, experiments, and auxiliary PCAPs: https://github.com/tomasz-lisowski/simurai-usenixsec2024-ae/tree/usenixsec2024-ae.

A.2.3 Hardware Dependencies

Only *S1* and *E1* have physical hardware dependencies:

- For *(S1)*, one physical UE, a SIMtrace 2, and external hardware for a cellular research network is required. In the provided experiment setup, we use a Motorola One Vision (XT1970-3) phone and a 2G network hosted via YateBTS using a bladeRF software-defined radio.
- For *E1*, one additional UE is required, with a SIM allowing it to connect to the research network. As above, we use a Motorola One Vision (XT1970-3) and a sysmoISIM-SJA2 programmable SIM card.

All other setups and experiments do not require special hardware and can be run on Linux-based x86 workstations. We recommend at least 4 CPU cores with 8 GB of memory and 50 GB of disk space. During the artifact evaluation, a remote machine was made available to the reviewers that had all hardware dependencies connected.

A.2.4 Software Dependencies

Our experiments are known to work on an Ubuntu 22.04 host. Additionally, the following software is required:

- Docker, for building and running the artifact code. We provide installation instructions in the artifact repository.
- Wireshark, for inspecting packet captures of *CS1 & CS2*, with a custom profile.

All other software dependencies (e.g., pcsc, srsRAN, FirmWire, ADB, scrcpy, etc.) are automatically fetched as part of the Docker-based build setup for the experiments.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Obtain the SIMURAI experiment repository:

```
$ git clone --branch usenixsec2024-ae \
    https://github.com/tomasz-lisowski/simurai-
     usenixsec2024-ae.git
```

Then, run the install script[1] from the experiment repository:

```
$ ./1__install.sh
```

Note that this will ask for the identifier of the setup, experiment, or case study that should be installed.

A.3.2 Basic Test

We prepared a script to perform basic tests step-by-step:

```
$ ./2__basic_test.sh
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): SIMURAI is a flexible tool for SIM security research and exploration. It can be integrated into representative cellular research lab setups and connected to an emulation platform. This is described in Section 6.1 of the paper and is demonstrated by *E-S1, E-S2*, and *E-S3*.

(C2): SIMURAI can simulate SIM spyware. This is described in Section 6.2.1 of the paper and is demonstrated by *E-E1*.

(C3): SIMURAI enables virtual fuzzing campaigns against emulated baseband firmware, allowing the discovery of CVE-2023-50806 and CVE-2024-27209 on unpatched modems. This is described in Section 6.2.2 of the paper and demonstrated by *E-E2*.

(C4): A SIM interposer can send malicious inputs to a UE, without interaction of the SIM card. This is described in Section 6.3.1 of the paper and demonstrated by the packet captures provided for *E-CS1*.

(C5): Malicious SIM applets can be provisioned remotely over the air without interaction by the victim. This is described in Section 6.3.2 of the paper and demonstrated by the packet captures provided for *E-CS2*.

[1] requires Docker, see installation instructions in the artifact repository documentation.

A.4.2 Experiments

Each experiment contains a *README.md* file in their respective folder. This file contains more detailed instructions and guidance for the interpretation of results. We also supply a global `1__install.sh` and `2__run.sh` scripts that allow installation and execution of specific setups and experiments. Alternatively, it is possible to run the scripts from within the respective setup or experiment folders.

(E-S1): Setup 1 - Physical UE.
[20 human-minutes, 10 compute-minutes, 10GB disk]
Shows how SIMURAI integrates with physical UEs *(C1)*.

Preparation. Create a physical setup similar to the setup shown in Figure 1. Ensure that UEs (without SIMURAI) can attach to the research network. On the host connected to the UEs, run:

```
$ ./1__setup/1__physical_ue/1__install.sh
```

Execution. To execute the experiment, run:

```
$ ./1__setup/1__physical_ue/2__run.sh 2
```

The command-line argument (in this case `2`), alters how the setup will be run. Please refer to the repository for more information.
Results. Both UEs successfully registered on the network. The network status of UE can be obtained by entering the code `*#*#4636#*#*` in the dialer.

(E-S2): Setup 2 - srsUE.
[10 human-minutes, 20 compute-minutes, 10GB disk]
Integrates SIMURAI with srsUE and connects to srsRAN via zeroMQ *(C1)*.

Preparation. Build the docker environment:

```
$ ./1__setup/2__srsue/1__install.sh
```

Execution. Run the experiment script:

```
$ ./1__setup/2__srsue/2__run.sh
```

Results. A 20 second initialization period for srsRAN, and a 10 second-long exchange between the srsRAN network, srsUE, and SWSIM, that involves around 16 APDU messages and successful Milenage authentication. The detailed logs for srsEPC, srsENB, srsUE, and SWSIM, are available in `log/*.log` files.

(E-S3): Setup 3 - Emulation Platform.
[10 human-minutes, 30 compute-minutes, 10GB disk]
Showcases SIMURAI's integration with FirmWire *(C1)*.

Preparation. Build the docker environment:

```
$ ./1__setup/3__emulated_ue/1__install.sh
```

Execution. Run the experiment script:

```
$ ./1__setup/3__emulated_ue/2__run.sh
```

Results. A 5 minute long run of FirmWire showcasing interaction between the emulator and FirmWire, as well as around 160 exchanged command-response pairs containing SWSIM ICCID. The detailed FirmWire log is stored in `result/firmwire.log`.

(E-E1): Experiment 1 - Simulating Spyware.
[30 human-minutes, 5 compute-minutes, 10GB disk]
Demonstrates SIMURAI simulating SIM spyware *(C2)*.

Preparation. Build the docker environment:

```
$ ./2__experiment/1__spyware/1__install.sh
```

Execution. Run the experiment script:

```
$ ./2__experiment/1__spyware/2__run.sh 2
```

The command-line argument (in this case `2`), alters how the setup will be run. Please refer to the repository for more information.
Results. Once the "SIM toolkit" application shows up in the application list on the phone, you can click the item named "Steal location via SMS", then "Run". This will trigger the spyware on SIMurai. Once the spyware is activated, SIMurai will proceed by sending a request to obtain location information of UE1. Then the location will be sent to UE2 using an SMS.

(E-E2): Experiment 2 - Fuzzing emulated UEs.
[20 human-minutes, 12 compute-hours, 20GB disk]
Runs a fuzzing campaign with FirmWire, leveraging the initial state provided by SIMURAI *(C3)*.

Preparation. Prepare the experiment environment:

```
$ ./2__experiment/2__fuzzing/1__install.sh
```

Execution. Run the experiment script:

```
$ ./2__experiment/2__fuzzing/2__run.sh
```

Results. A fuzzing campaign with found crashes. The AFL output directory is located in `result/out/`. Test cases can be replayed by running:

```
$ ./2__experiment/2__fuzzing/3__replay.sh
    replay_target
```

(E-CS1): Case Study 1 - Interposer.
[10 human-minutes, 10MB disk]
Showcases how a modified interposer sends a malicious payload to a UE *(C4)*. Detailed description, diagrams, and artifact files are available under `3__case_study/1__interposer` and the associated `README.md` file.

Preparation. Install the Wireshark profile.
Execution. Inspect the provided packet captures with Wireshark.
Results. The two SIM sniffers show that the interposer manipulates the SIM communication, and injects the crashing payload.

(E-CS2): Case Study 2 - OTA update.
[10 human-minutes, 50MB disk]
Showcases a hostile applet being provisioned to the SIM via an OTA update in a 4G network *(C5)*. Detailed description, diagrams, and artifact files are available under `3__case_study/2__remote_ota_install` and the associated `README.md` file.

Preparation. Install the provided Wireshark profile. It can be found inside `util/wireshark_profile`.
Execution. Inspect the provided packet captures with Wireshark, and the screen capture. The PCAP shows both cellular traffic and SIM communication to observe the installation.
Results. Firstly, no applet is present (shown in screen capture). Then, the malicious applet is installed remotely over the cellular connection. Subsequently, the crashing payload is sent to the UE.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2024/.