



USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

CATANA: On the Dangers of SIM-Originating AT Commands

Tomasz Piotr Lisowski, *University of Birmingham*; Kristian Covic, *Fuzzware*;
Marius Muench, *University of Birmingham*

<https://www.usenix.org/conference/woot26/presentation/lisowski>

**This paper is included in the Proceedings of the
20th USENIX WOOT Conference on Offensive Technologies.**

August 10–11, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-57-1

Open access to the Proceedings of the
20th USENIX WOOT Conference on Offensive Technologies
is sponsored by USENIX.



CATANA: On the Dangers of SIM-Originating AT Commands

Tomasz Piotr Lisowski^{id}
University of Birmingham

Kristian Covic
Fuzzware

Marius Muench
University of Birmingham

Abstract

Hostile SIMs have been discussed as an attack vector against Mobile Equipment (ME) connected to cellular networks. One main attack path are *proactive commands* sent from the SIM to the victim device. In this work, we examine the threats posed by the *RUN AT* command which are SIM-originating requests for the ME to execute a specified AT command, effectively creating a SIM AT interface.

To explore this interface, we introduce the CATANA toolkit and use it to analyze real-world devices. Despite existing community knowledge on proactive commands and the dangers of AT commands, our investigation shows that SIM AT commands pose a significant security risk for MEs.

We survey 26 different MEs (8 IoT devices and 18 smartphones) and find that 9 expose the SIM AT interface, leading to the discovery of 4 vulnerabilities. We present case studies demonstrating the impact of discovered vulnerabilities, including command execution, arbitrary file read, downgrading connections to 2G, and Denial-of-Service (DoS) of the ME. Crucially, our work emphasizes the security benefit of hardening, deprecating, or disabling, the SIM AT interface.

1 Introduction

Cellular networks are ever advancing with increases in bandwidth, speed, feature-set, and coverage; This enables them to support a larger and more complex set of services and devices. A wide range of devices rely on these networks for connectivity, e.g., phones, cars, computers, and an ever-expanding assortment of Internet-of-Things (IoT) devices, including payment terminals, cameras, routers, wearables, *smart* devices, and more.

The SIM—be it a classic plastic card, an integrated SIM, or an Embedded SIM (eSIM)—remains attached to the modem of every connected device, providing the subscriber identity needed for authentication and other network services. However, instead of exclusively storing authentication secrets, SIMs are also small computers communicating with

the modem of the mobile device. In most cases, the modem's Communication Processor (CP)¹ initiates the communication with the Subscriber Identity Module (SIM) by reading files, involving it in the authentication flow, or interacting with programs stored on the SIM (i.e., applets). Moreover, SIMs can leverage *proactive commands* to invert this flow of communication and prompt the CP to execute a SIM-controlled commands [12].

Previous work frequently highlighted the security issues resulting from proactive command support [10, 21, 48, 49] and recent work argues for the inclusion of *hostile SIMs* in mobile security threat models [22]. While the capabilities of most proactive commands are restricted to the intended purpose outlined in the specification [12], the *RUN AT* command embeds the highly generic set of ATtention (AT) commands directly into the proactive request.

Hayes compatible interfaces—those supporting AT commands—have long been used to control and configure modems, ever since their initial introduction in the 1980s [19]. Both academic [18, 44] and industry-led research [6] demonstrated the security threats of AT commands for modems embedded in modern communication stacks. This resulted in a gradual tightening of AT interface security, with modern smartphones exposing it only to privileged users or after explicitly enabling it as a developer.

In this work, we investigate the SIM-based AT interface enabled by proactive commands², and develop CATANA, a toolkit for exploring this interface. CATANA consists of four tools enabling us to transceive SIM AT commands, automate testing of such commands, and inspect results from experiments. Equipped with CATANA, we set out to understand the prevalence of the SIM AT attack surface, and survey 8 IoT devices and 18 smartphones for their support of the SIM AT interface. Our survey shows that while smartphones rarely support SIM AT commands, IoT modems commonly expose this feature, with 2 and 7 of the tested devices supporting the SIM AT interface, respectively. Additionally, we identify 4

¹Also referred to as a *Communication Processor* or a *Baseband Processor*.

²We refer to this as the *SIM AT interface* for brevity.

vulnerabilities affecting multiple surveyed devices.

We highlight the dangers of the SIM AT interface and demonstrate the impact of identified vulnerabilities by presenting 4 case studies, ranging from Denial-of-Service (DoS) and a 2G downgrade, to arbitrary file read, and code execution.

In summary, we provide the following contributions:

- We introduce CATANA, a toolkit for interactively exploring the SIM AT interface, automating testing of AT commands and analyzing the results of experiments.
- We analyze the presence of the SIM AT interface for 26 devices, spanning smartphones, development boards, and IoT modems embedded in real-world systems.
- We implement 4 SIM-originating attacks against different modems by abusing AT commands: One code execution, one DoS, an arbitrary file read, and a 2G downgrade attack.
- We discuss mitigation of our attacks, and motivate the potential deprecation or removal of the SIM AT interface.

2 Background

2.1 ATtention Protocol

The AT protocol was first introduced in the Hayes Smartmodem in 1981. For the first time, control over features and configuration of the consumer-grade modem was performed in-band over the same serial interface alongside data transmissions [9, 35].

Adoption of AT commands continues to this day, with many modern devices still implementing a functionally identical design. Since the inception of AT commands, the International Telecommunication Union (ITU), the 3rd Generation Partnership Project (3GPP), and others have released standards describing universally understood commands and their function, although many vendors include additional proprietary commands that extend and alter the standardized set.

All AT commands begin with the letters *AT*³ followed by the command and, occasionally, numeric and string parameters. Commands can be categorized into one of 4 types: *read*, *write*, *execute*, *test*. The same command will often support more than one type of operation to implement, e.g., a get and set mechanism. Below is a generic functional description of each type of command:

Reading (*AT<cmd>?*) extracts information out of the modem, e.g., build version, or Access Point Names (APNs).

Writing (*AT<cmd>=[<p1>, <p2>, . . . , <pn>]*) reconfigures the modem, e.g., by connecting to a different network, or by sending key presses.

³Most *extended commands*, as defined by 3GPP, begin with *AT+*, with only individual exceptions [3].

Executing (*AT<cmd>*) is similar to writing in that it requests that the modem performs an operation, but does not provide parameters to the command.

Testing (*AT<cmd>=?*) variant will generally exist for every command and is used to detect whether a command is supported by the modem and what parameters are supported, without causing any side-effects.

2.2 SIM AT Interface

Cellular devices often follow a similar design, i.e., there is usually an Application Processor (AP) running the user-facing Operating System (OS) like Linux, iOS, Android, or a custom embedded firmware. Connected to the AP, through shared memory or a serial interface, is the CP. It is responsible for all radio communications with nearby cell towers, as well as a direct connection to the SIM, either a plastic card inserted into a connector, or an eSIM embedded into a Trusted-Platform Module (TPM) or other Integrated Circuit (IC).

The CP-SIM half-duplex interface uses Application Protocol Data Units (APDUs) to send commands and responses back and forth for the duration of the cellular connection. Each exchange is initiated with the CP sending a Command APDU (C-APDU), followed by the SIM sending back one Response APDU (R-APDU).

A proactive SIM mechanism exists to enable cards to also issue commands back to the CP using messages called *proactive commands*. The indication of an available command can be embedded inside regular R-APDUs flowing on the line, but the dispatch of the command itself requires an explicit *fetch* from the CP. A bit field indicating CP support for proactive SIM commands is transmitted as part of the *TERMINAL PROFILE* C-APDU during the SIM bootstrap process. One proactive command of particular interest to this work is *RUN AT*. It appears less frequently on phones, but standalone cellular modules sold to manufacturers of cars, car chargers, and other IoT devices will often offer *RUN AT* out-of-the-box.

Ultimately, the support for *RUN AT* exposes an interactive AT interface to the SIM, allowing the card to dispatch AT commands to the CP, that may bypass the expected flow, reducing the AP's visibility and allowing the SIM to perform covert malicious activity inside the device.

2.3 SIM Tooling

There are three primary methods of exploring the SIM interface: Using programmable (research) cards, commercial cards with interposers, or using software-emulated cards.

Programmable Cards. Most smart cards issued to consumers for, e.g., telecommunications, banking, or access control, only include the Personal Identification Number (PIN) and PIN Unblocking Key (PUK), but the KIC, KID, and KIK

key material⁴ required for software installation onto the card, remains a secret. Cards intended for research and private use, such as the cards offered by Sysmocom [43], include KIC, KID, and KIK key material and support for *applet* installation. Most contemporary applets, including those compatible with Sysmocom SIM cards, are built on top of a Java-based framework and card API. These applets can be installed on compatible cards [32] and are executed by an embedded Java Card Virtual Machine (JCVM) while the card is powered and communicating with a mobile device.

Developers rely on the SIM Application Toolkit (SAT) to develop applets that integrate with the USIM application present on every modern SIM card [2, 11, 12]. This is necessary because cellular devices rely on the USIM application for authentication to the network, various network and SIM configurations, as well as non-volatile storage for contacts, Short Message Service (SMS) messages, and connection state.

Interposed Cards. A number of commercial SIM cards already contain proprietary applets from network operators or card manufacturers that offer their users a range of supplementary cellular services built on top of the SAT APIs. When this, or other, proprietary behavior of the card is desired, another approach can be employed to obtain programmability on the SIM-Mobile Equipment (ME) interface, namely, using interposers. These are thin sticker-like devices with electrical contacts on both sides. They attach on one side to the SIM card, and on the other, to the SIM connector. This enables a Machine-In-The-Middle (MITM) attack where the interposer may transparently bridge the two sets of contacts, but also drop messages, alter them, or inject new ones as a result of lacking authentication and encryption on the interface. This method offers researchers a similar capability as with the programmable cards, but at an increased level of complexity and reliance on hardware which was not designed for this purpose.

Beyond hardware interposers, SIM research tooling also enables an overt, but more flexible way of interposing by exposing the electrical interface from the SIM socket to an external device, optionally facilitating a connection to remote SIM cards over the Personal Computer/Smart Card (PC/SC) interface. Most prominently, SIMtrace2 forms a transparent bridge between the ME and a card inserted into the SIMtrace2 or a card connected to a host computer via an external card reader. This results in behavior that mimics the card, but with the ability to interpose the communications on a host instead of using interposing hardware.

Recent work by Gegenhuber et al., SIMULATOR, explores an alternate hardware approach to SIMtrace2 that lowers the cost of hardware interposing [14, 15].

⁴KIC and KID key and identifier material are used for ciphering, redundancy checks, cryptographic checksums, and digital signatures. KIK key material is used to protect KIC and KID [1].

Software Cards. Building on top of tools like SIMULATOR and SIMtrace2, recent work introduced fully software-emulated SIMs, for instance, Onomondo UICC [31], VSmartCard [26], and SIMURAI [22].

We build our work on top of SIMURAI [22] that consists predominantly of two components: SWICC, the framework core that forms a generic smart card emulator, and SWSIM that extends the core emulation to add SIM-specific features, e.g., proactive commands and Milenage authentication. A distinct feature of SIMURAI is the APDU pipeline flexibility that supports APDU handler overloading, and APDU response rewriting, that CATANA heavily relies on to achieve its functionality.

3 Related Work

AT Attacks. Miller et al. [23], Pereira et al. [36], Amaranter et al. [5], Hay et al. [16], and Tian et al. [44] explore the risks of AT over USB and AT over serial on commercial smartphones. Weinmann et al. [47] present multiple practical exploits using AT command handlers in the CP to trigger vulnerabilities. Furthermore, Herfurt et al. [17, 34] and Traynor et al. [45] showed that Bluetooth stacks of feature phones, in particular, the Dial Up Networking (DUN) of the—now defunct—Symbian mobile OS could result in AT dispatch.

Additionally, multiple works acknowledged the presence of the SIM AT attack surface [10, 21, 22] but did not further explore this attack surface. Our work builds on this line of work and shows the practical security implications of the SIM AT interface.

Fuzzing. Karim et al. [18] developed *ATFuzzer*, which uncovered multiple problems when injecting AT over Bluetooth and USB. Mulliner et al. [29] created a MITM setup to form a black-box fuzzer that injected SMS Protocol Data Units (PDUs) over Wi-Fi into victim devices running on multiple OSs, leading to discovery of various crashes and bugs.

Moreover, AT commands often support fuzzing of other components in the cellular stack by providing a dedicated control channel [13, 20].

Beyond cellular modems. Vaccari et al. [46] and Piracha et al. [37] exploited the MAC layer of the ZigBee network protocol to attack ZigBee end-devices. Despite offering interesting case studies for the dangers of AT commands, the underlying devices do not rely on SIMs.

Defenses. AT-based attacks are also discussed in the context of defenses. For instance, *RILDefender* [48] leverages the Android RIL to detect malicious activity, such as reception and transmission of binary SMS, or execution of AT commands. Furthermore, Mulliner et al. [28] developed another defense to explicitly protect against AT attacks using a *virtual modem*

that stands between the AP and CP to filter AT commands sent from the AP to the CP.

While both of the aforementioned defenses protect against AT commands sent through common interfaces, SIM AT commands that are executed directly on the CP, would remain undetected.

4 Motivation

Researchers frequently pointed out the dangers of unrestricted access to AT commands on modern modems, especially those embedded into smartphones [18, 44]. Similarly, the potential attack vector introduced by hostile SIMs has also been discussed in past work [10, 21, 22].

Although prior work regularly acknowledges the possibility of SIM AT attacks, there is a lack of studies examining the prevalence and dangers of the SIM AT interface in-depth, to the best of our knowledge. Given the severe consequences of AT attacks shown in the literature, e.g., lock-screen bypasses, memory corruptions, or event injection [44], we believe that the missing exploration of SIM-originating AT attacks forms a major blind spot in our understanding of the mobile security threat landscape.

Our work aims to fill this gap by building tools that aid in exploring the SIM AT interface and by understanding the attack surface introduced by the SIM AT interface to contemporary cellular devices.

Threat model. We adopt the threat model introduced by Lisowski et al. [22], where hostile SIMs originate from one of four sources:

- i Vulnerabilities in SIM software stacks allowing for remote exploitation by attackers.
- ii Physical access to the device or its SIM, allowing an attacker to exchange the SIM or install an interposer.
- iii Remote SIM administration available to malicious or compromised operators. This allows the remote extension of SIM functionality and carrying out attacks against deployed SIM cards.
- iv Supply-chain attacks, where backdoors are inserted into a SIM at any time between manufacturing and distribution to end users.

Although this appears to be a strong attacker model at first glance, we note that all four sources for hostile SIMs are relevant and realistic in the context of telecommunication security: SIMJacker attacks used vulnerabilities in the SIM software stack [42], NSA's ANT catalog documents physical SIMs as an attack vector [7], a recently observed operator data breach includes SIM authentication secrets [24], and

MITRE's 5G Hierarchy of Threats documents supply-chain attacks against SIM credentials as observed [25].

Regardless of the source of the hostile SIM, the attacker's objective remains the same: compromise the modem that the SIM is connected to. While this can enable eavesdropping and make the cellular connection unavailable in the case of smartphones, this objective has further importance in the context of IoT devices. Considering that IoT devices are often locked down and expose a limited variety of interfaces to attackers, hostile SIMs can serve as an entry vector for a full-system compromise beyond solely the device's modem.

5 Design

We identified five requirements that encompass various approaches to experimentation on the SIM AT interface; We aim to satisfy these requirements by building four tools that can streamline further research. We visualize the developed tools in Figure 1.

5.1 Requirements

We identified the following as the primary design requirements for tools aiding research of this interface:

- R1** Using a fully software-emulated SIM card when prioritizing flexibility, performance, and iteration time, especially when a cellular network connection is not required for those experiments.
- R2** Real SIM cards may exhibit behavior that is proprietary and secret, be it with a custom applet, or keys needed to authenticate to real networks. In this case, there must be a way to interpose a real card to allow for interaction with the AT interface exposed to the SIM, without requiring applet installation onto commercial SIMs.
- R3** Devices do not always have an easily accessible SIM slot, therefore using tools such as the SIMtrace2 [33] is not possible. In this case, reprogrammable cards where the administrative keys of the card are provided to the researcher, may be required. These types of cards often contain a JCVM and support Java Card (JC) applet installation. The applet can receive APDUs that contain an embedded AT command which can then be replayed to the phone using proactive commands. The APDU can be sent to the SIM over another interface, e.g., *AT+CSIM* using another AT interface exposed by the device, or using binary SMS.
- R4** Tian et al. found 3500 unique AT commands when standards and Android-facing CP firmware are considered [44]. There are a number of other mobile OSs like KaiOS and iOS, as well as other CPs intended for embedded use, potentially including additional unique AT

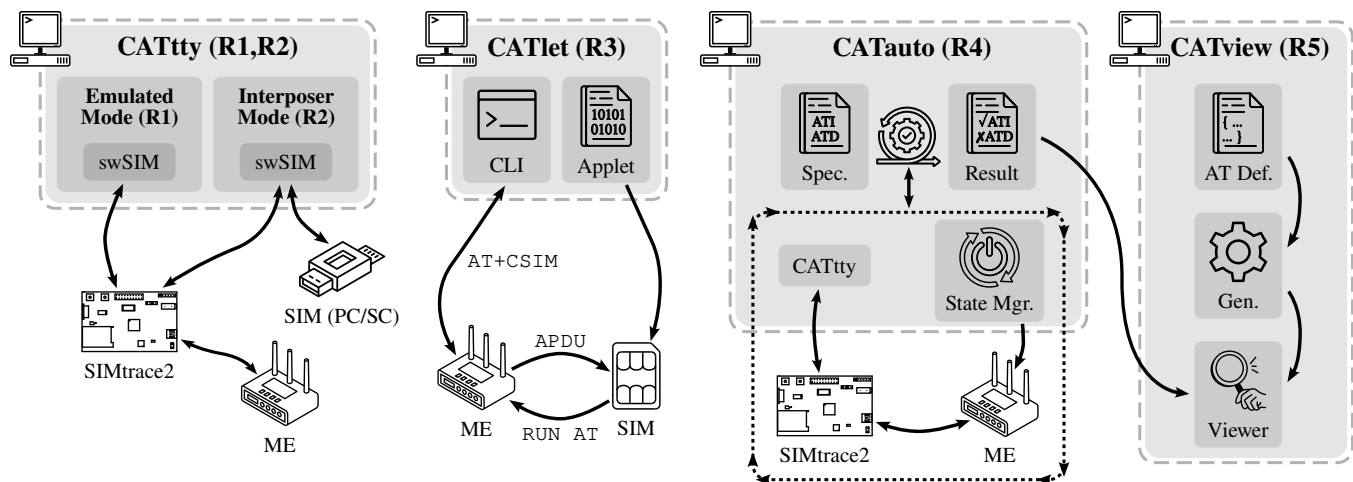


Figure 1: Overview of the CATANA toolkit, consisting of four independent tools. CATVIEW enables the execution of SIM AT commands by leveraging an emulated or interposed SIM and CATLET interactively executes SIM AT commands from a specialized SIM applet. CATAUTO enables iterative execution of SIM AT commands against a target and produces a result file, which can then be analyzed using CATVIEW.

commands. Consequently, automated AT command testing using the SIM as originator is essential to understand the landscape of the explored attack surface.

R5 A human-readable and portable file should be provided to the researcher as a result of the SIM AT interface analysis. This will aid in further targeting of efforts and enable passing on to supplementary analysis tooling down-stream.

5.2 Tooling

CATTTY (R1 & R2) can operate in two modes: the first offers complete card emulation, and the second interposes another PC/SC-attached card. Both modes utilize the tooling provided by SIMURAI, and both modes rely on the SIMtrace2-ME connection as the communication medium. The user may also decide to use the tool *interactively*, which offers them an AT shell for rapid prototyping, or as *one-shot* for injecting a single AT command for non-interactive automation.

When using the emulated card, the default proactive application of swSIM is overridden by a proprietary one that injects *RUN AT* proactive commands during runtime. When an experiment requires certain features of another existing card, be it an emulated card or a physical one, the user can choose to interpose that card using a built-in interposer layer based on swSIM that offers a predominantly transparent pass-through for incoming APDUs. This way a swSIM instance can behave like the card it is interposing, while redirecting messages related to the proactive capability to a proprietary proactive application included in CATTTY to maintain the ability to inject *RUN AT* proactive commands without interference from the interposed card.

CATLET (R3) comprises two parts: the first is a JC applet installable on compatible SIM cards, and the second is a host program that communicates with the applet over a secondary AT command interface exposed by the target device.

The applet receives specially crafted *ENVELOPE* APDUs that encode an AT command from the user. This AT command will be dispatched to the CP using the *RUN AT* proactive command and the card will listen for a response. Once a response is received inside an *ENVELOPE* coming from the CP, the AT response is extracted and copied into an internal buffer.

A host program waits for the user to input an AT command, then packs it into a packet that is placed inside an *ENVELOPE* APDU, and finally packs the APDU into a *AT+CSIM* command that can be sent over the exposed AT interface to eventually land in the SIM applet. The host can request the AT response as well as applet status for debugging purposes by sending packets over the *AT+CSIM*-based path. In essence, this forms a bidirectional channel with the host and SIM applet assuming the roles of a controller and a peripheral, respectively.

CATAUTO (R4) builds on top of CATTTY to automate the dispatch of AT commands to devices supporting the SIMtrace2. It collects the AT commands and AT responses together into an output file intended for further offline analysis.

CATVIEW (R5) analyzes the output from CATAUTO to produce a human-readable and portable JSON file listing all AT commands that produce a non-error response. Each command in the list includes a description and a list of variants, where each variant is also accompanied by a list of responses observed for that variant.

6 Implementation

CATANA contains 3331 lines of Zig and 338 lines of Java, distributed as follows, Zig: ~43% for CATTTY, ~22% for CATVIEW, ~22% for CATLET, and ~13% for CATAUTO. Java: 100% for the JC applet of CATLET.

6.1 CATTTY

CATTTY takes advantage of the SIMURAI framework, in particular the SWSIM component, to implement all features described in the design.

It runs an emulated SIM powered by SWSIM with the ability to submit *RUN AT* proactive commands to the CP. This is achieved using a proactive application compatible with SWSIM that allows us to issue *RUN AT* proactive commands asynchronously from other threads such as the one listening for user input when running CATTTY interactively. The state of the application is tracked carefully to prevent deadlocks, e.g., when an AT command is submitted to the CP but does not lead to a response, or when a user wants to replace a previously entered AT command with a new one before the prior is submitted to the CP. The *one-shot* feature has its own simpler proactive application to isolate complexity and serve as a minimal example for *RUN AT* dispatch using SWSIM.

We observed that the dispatch of *RUN AT* proactive commands should not occur before the first *TERMINAL PROFILE* or *ENVELOPE* is received, to avoid invalid AT responses. We refer to this as the *trigger* point, since CATTTY will always wait until a trigger event has occurred before sending any AT commands.

SWSIM does not include an interposing functionality by default; as such, we pair up the SWSIM framework with the *scraw* library that provides thin cross-platform wrappers to interact with PC/SC-connected cards. We realize interposing at the Transmission Protocol Data Unit (TPDU) layer used by SIMURAI (unlike PC/SC which operates at a higher APDU layer) using a number of distinct steps. Handling incoming APDUs starts in our proprietary APDU handler registered using *swicc_apduh_pro_register*. Here, we need to drop APDUs that could put the interposer or the interposed card in an undefined state, namely:

- *UPDATE BINARY* and *UPDATE RECORD* to avoid persistent changes to the card's EEPROM and to avoid handling differences in write behavior of different cards as indicated by 'Data Coding' in the historical bytes of the Answer-To-Reset (ATR).
- *MANAGE CHANNEL* to enforce use of the default logical channel and prohibit the spawning of secondary ones.
- *SEARCH RECORD* due to a lack of support in SWSIM. Commands that are not processed by any APDU handler do not get parsed when the C-APDU header is received and

as a result, we have no way of knowing what procedure byte to send back which requires us to return an error indicating an unsupported class or instruction.

- *GET RESPONSE* is used for command chaining, i.e., allowing a command to queue up a response for later retrieval using this APDU. When the interposed card indicates that a response is available, we must manually generate a procedure byte for the *GET RESPONSE* command and allow it to be sent to the interposed card once the header is acknowledged.

The second step to interposing occurs in the APDU handler override function registered with *swicc_apduh_override_register*. We short circuit this function when any of the blocked commands are received, otherwise we dispatch the current APDU to the interposed card over PC/SC. When a response is received, we scan the status word for a normal success or a proactive indication and overwrite it with our own if a proactive command created for the user of CATTTY is ready to be injected. We gracefully handle any errors coming out of the interposed card, but a breakdown of the PC/SC communication can occur if a state mismatch arises as a consequence of an unforeseen interaction between SWSIM and the interposed card.

6.2 CATLET

This tool establishes a bidirectional connection between the CATLET Command Line Interface (CLI) and a CATLET applet running on the SIM. The CP communicates with the SIM using APDUs⁵, and the host connects to an exposed AT interface of the target device to obtain a regular (non-SIM) AT interface, allowing the CLI to dispatch proprietary packets to the applet. The CLI forms packets based on user input, packs them into *ENVELOPE* C-APDUs and sends them to the active SIM using the *AT+CSIM* command that is expected to be supported by the exposed AT interface. This communication channel allows the host to dispatch user-specified AT commands as well as other commands that extract state and AT responses from the applet.

Although the SAT offers a large number of features, it does not provide an insight into all C-APDUs received by the card unless they are proprietary *ENVELOPE* commands. As a consequence of this abstracted design, CATLET relies on *ENVELOPE* C-APDUs and packs a proprietary packet format into a proprietary SAT template object. The applet must register to receive the proprietary messages using *toolkitRegistry.setEvent(EVENT_UNRECOGNIZED_ENVELOPE)*. Whenever this type of proprietary C-APDU is received by the USIM application, the *EVENT_UNRECOGNIZED_ENVELOPE* event is dispatched to every SAT applet. This event allows us to

⁵Non-extended C-APDUs can carry at most 256 bytes in their data field when using protocol T=0. This sets an upper length limit on AT commands dispatched from the CATLET CLI to the applet.

```
[ {
  "title": "Control Call Event",
  "description": "AT^DSCI is used to
    enable/disable ^DSCI URC (call event).",
  "base": "AT^DSCI",
  "variant": [
    { "suffix": "?", "response": [
      "\r\n^DSCI: 0\r\n\r\nOK\r\n" ] },
    { "suffix": "=?", "response": [
      "\r\n^DSCI: (0-1)\r\n\r\nOK\r\n" ] },
    { "suffix": "", "response": [
      "\r\nOK\r\n", "\r\nOK\r\n" ] } ]
} ]
```

Listing 1: A minimal example of the JSON output generated by CATVIEW.

trigger our applet and extract the value from the proprietary SAT template object immediately at the time of reception. In case the received packet instructs the applet to send an AT command, we extract the embedded command and repack it into a *RUN AT* proactive command that gets queued for transmission to the CP. In effect, this mechanism allows the SIM card to execute AT commands on behalf of the user of the CATLET CLI.

6.3 CATAUTO

CATAUTO reads a list of AT commands from an input text file, each delimited by a newline. Then, the main orchestration loop sequentially repeats the following steps for each AT command included in the input:

- i Start up a *one-shot* CATTTY instance and the *simtrace2-cardem-pcsc* driver of SIMtrace2 that bridges the SWSIM-based SIM hosted by CATTTY with the ME. Lastly, we trigger an ME reboot using *adb reboot* and begin waiting for a response.
- ii When an AT response arrives, CATAUTO writes the command and response to an output file.
- iii Finally, *SIGTERM* and *SIGINT* are sent to *simtrace2-cardem-pcsc* and CATTTY, respectively, and CATAUTO closes the control loop using *pcsc-scan* to wait until the PC/SC middleware no longer detects the SWSIM card.

6.4 CATVIEW

Analysis begins with the output file generated by CATAUTO and ends with a JSON file formatted according to Listing 1. The resulting JSON file consists of an array of commands, each including a description of the command's function, and a list of supported variants. Inside each variant is another array

containing a list of all responses obtained with that variant of the command.

At first, CATVIEW reads in all AT commands and responses, then unpacks the data into an internal representation optimized for further processing. Next, we apply a filter that removes any responses of the form *\r\nERROR\r\n* along with any variants that become empty as a result of the response removal. The remaining data gets converted into an internal representation modeled after the JSON schema, and written to a file as a string.

7 Evaluation

We evaluate CATANA by gauging the utility it offers to the research community. Consequently, our evaluation aims to answer two key questions: (1) *How prevalent is the support for AT commands via SIM* and (2) *What is the security impact of accepting AT commands from the SIM interface?*

To answer these questions, we first perform a survey of representative devices, and then present a series of case studies demonstrating vulnerabilities and architectural weaknesses exploitable using SIM AT commands.

7.1 Device Survey

Our device survey is based on a composition of data sources. While SIM AT command support is ultimately tested on-device, information about the device's architecture is obtained from documentation (if available) or by reverse-engineering the corresponding CP firmware.

7.1.1 Smartphones

CPs, including the ones deployed on smartphones, communicate their proactive capabilities to a SIM upon insertion. We selected a range of smartphones, covering 10 vendors, 18 phone models, and 6 chipset manufacturers, and retrieved their terminal profiles to assess their support for the *RUN AT* proactive command.

Table 1 shows our results. Previous studies [18,44] and publicly available information suggests that, for most phone models, the CP supports execution of AT commands dispatched from the device's Radio Interface Layer (RIL), with the exception of Apple iPhone models. However, among all tested phones, only three report support for *RUN AT*. Two of the affected models come from the same vendor, and all affected models leverage a Qualcomm CP chipset. However, models from other vendors, even when using a Qualcomm CP, do not offer support for *RUN AT*, suggesting that the feature is explicitly added through vendor personalization.

Beyond inspecting the terminal profile, we also investigated how Android and CP firmware handle SIM AT commands. Generally, proactive commands, including *RUN AT*, are first

Vendor	Model	CP Chipset	SIM AT
Apple	iPhone 15	Qualcomm (X70 5G)	○
	iPhone 16e	Apple (C1)	○
Google	Pixel 7	Samsung (Exynos 5300)	○
	Pixel 9	Samsung (Exynos 5400)	○
Motorola	One Vision	Samsung (Exynos 9609)	○
	G32	Qualcomm (SM6225)	○
OnePlus	5T	Qualcomm (MSM8998)	○
	Nord CE 2	MediaTek (Dimensity 900)	○
OPPO	Find X5	Qualcomm (SM8350)	●
	Reno14 F 5G	Qualcomm (SM6450)	●
Samsung	Galaxy A14	MediaTek (MT6769)	○
	Galaxy A72	Qualcomm (SM7125)	○
	Galaxy S22	Samsung (Exynos 2200)	○
	Galaxy Z Flip6	Qualcomm (SM8650)	○
ASUS	Zenfone 9	Qualcomm (SM8475)	●
Huawei	P40 Lite	Huawei Kirin 810	○
Oukitel	C19	Mediatek (MT6735)	○
ZTE	Blade A54	Unisoc (SC9863A)	○

● : Supported ○ : Not supported

Table 1: Smartphones tested for SIM AT support. This information is based on the terminal profile sent by the phone to the SIM.

handled by the CP, and forwarded to the AP if it is necessary for handling.

We analyzed the proactive command handling inside the Shannon-based firmware images running on the Samsung Exynos CPs, in particular: Google Pixel 9 and Motorola One Vision. We found that the main proactive command parser explicitly adds the *RUN AT* proactive command to a block list. Nevertheless, the baseband *still* contains the code for parsing and handling these commands, including code paths that forward them to the AP. This code is unreachable under normal execution, but it could still be triggered under control-flow hijacking exploits.

On APs running Android, an incoming *RUN AT* proactive command is first parsed inside the Telephony Card Application Toolkit (CAT) service and then forwarded to the OEM-customizable SIM Toolkit (STK) app that implements the command handlers. In AOSP’s reference STK app, *RUN AT* is flagged as a non-interactive command, i.e., a command that requires no further user confirmation before execution.

Most phones report no support for execution of SIM AT commands. Nevertheless, Android and CP firmware often retain the infrastructure needed to execute them, often guarded solely by a single block list check.

7.1.2 IoT Modems

To assess the support for SIM AT commands in IoT modems, we examined 8 modems from 4 vendors. We inspected modems integrated into larger embedded systems, as well as standalone devices, such as development boards and USB modems. We picked prominent internet-connected embedded systems, e.g., Electric Vehicle (EV) chargers, an industrial router, and an automotive telematics unit from a car.

During our survey, we found that nearly all IoT modems follow an internal AP-CP architecture, where a real-time CP implements the cellular stack and a more powerful modem-internal AP controls the CP’s operation. The internal AP also provides connectivity to the larger system, for instance, through USB, or a separate serial (AT) interface. To better understand the wider ecosystem, we also surveyed the operating system used by the internal APs.

Figure 2 shows the inspected IoT modems and Table 2 summarizes the results of our survey. All explored Quectel modems use a Qualcomm-based CP and an internal AP running Android on an ARM Cortex-A7 core, even if the main CPU of the device interfacing with the modem is less powerful. Likewise, the examined Sierra Wireless modem also leverages a Qualcomm CP, but bases the internal AP firmware on an adaptation of ThreadX. Nordic Semiconductor nrf9160 uses an in-house designed modem, and the software running on the internal AP can be freely chosen by the developer, with common options including ZephyrOS and bare-metal operation built with Nordic’s nrfx library [30]. Lastly, the Simcom SIM800C does not include a dedicated AP, but relies solely on an ARM-based MediaTek MT6261 CP.

All surveyed devices provide an AT interface for operating and configuring the modem. Except for the nRF9160 and the SIM800C, all other modems also support the *RUN AT* proactive command. In these cases, the modem’s CP commonly does not implement exhaustive handling of the SIM AT commands. Instead, incoming SIM AT commands will generally be forwarded to the internal AP where they get processed.

Most IoT modems support SIM-originating AT commands. In regards to most commands, handling is implemented on the internal AP rather than the CP, offering a rich attack surface to hostile SIMs.

7.2 Case Studies

For our case studies, we selected three representative devices from each of the prevalent cellular-connected device classes: AUTEL Maxi US AC W12-L-4G (IoT), a Quectel EG25-G modem (standalone), and an OPPO Reno 14 F 5G (phone).

7.2.1 Code Execution

Target and Context. The AUTEL Maxi US AC W12-L-4G car charger consists of a GigaDevice GD32F407 ARM Cortex-

Vendor	Model	CP Chipset	AP OS	Device(s)	AT	SIM AT
Quectel	EC25-AFX	Qualcomm MDM9607	Linux	Standalone, EV Charger	●	●
	EC25-EUX	Qualcomm MDM9207	Linux	Industrial Router	●	●
	EG25-G	Qualcomm MDM9607	Linux	Standalone	●	●
	RM520N-GL	Qualcomm SDX65	Linux	Standalone	●	●
	AG525R-GL	Qualcomm SA415M	Linux	Automotive Telematics Control Unit	●	●
Nordic	nRF9160-DK	Nordic nRF9160	Custom	Development Board	●	○
Sierra Wireless	RC7620	Qualcomm MDM9207	ThreadX	Development Board, AC Charging Controller	●	●
Simcom	SIM800C	MediaTek MT6261	N/A	Standalone	●	○

● : Supported ○ : Not supported

Table 2: IoT Modems tested for SIM AT support. The AP OS refers to the internal AP embedded within the modem.

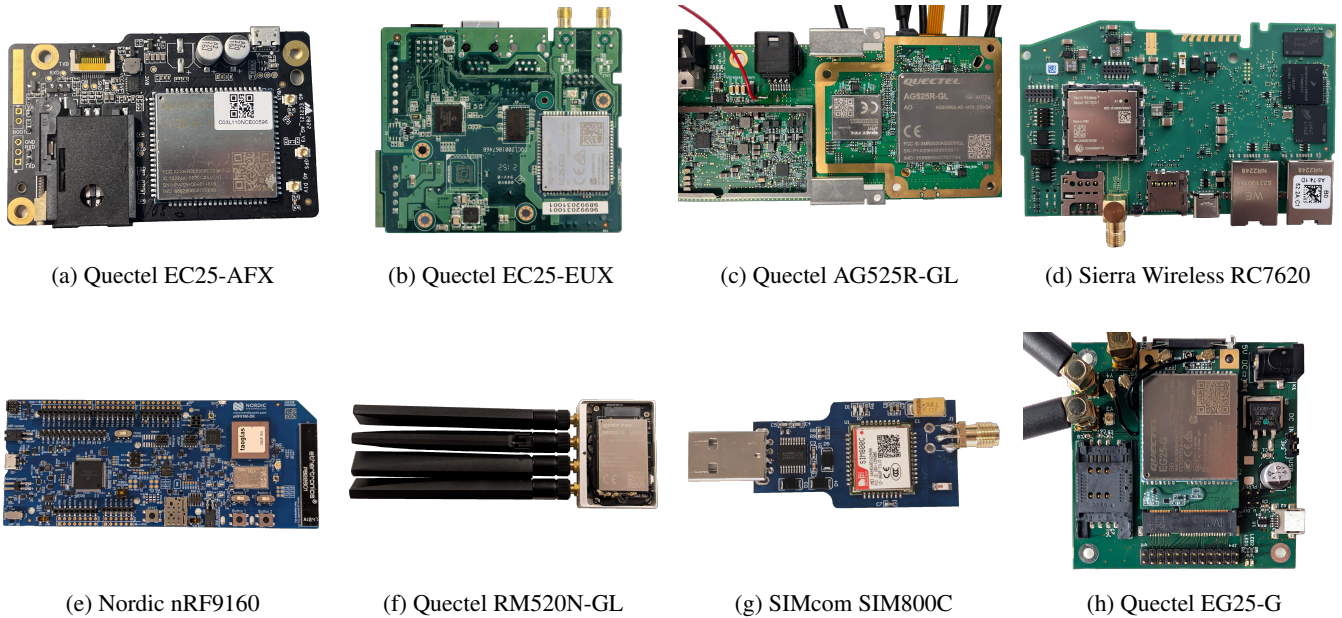


Figure 2: Tested IoT modems. (a), (b), (c), and (d) are modems embedded in real-world IoT devices, while (e), (f), (g), and (h) are standalone modems. We also tested standalone version of modems (a), (b), and (d).

M4-based microcontroller and a Quectel EC25AFXDGA cellular modem and navigation module (with an ARM Cortex-A7 as the AP and a Qualcomm MDM9607 as the CP). The Cortex-A7 runs a Linux-based firmware, while the Cortex-M4 is running a ThreadX Real-Time Operating System (RTOS).

The SIM card is connected to the Quectel module, and for an initial investigation, we extracted the AP root File System (FS) from a Quectel EC25-AFX development board. Reverse engineering showed that the *atfwd_daemon* binary receives AT commands from various sources, including the SIM AT interface. While this binary generally forwards incoming AT commands to the respective sink, it also directly implements a small number of AT handlers.

Vulnerability. Further reverse engineering showed that the AT handlers implemented within *atfwd_daemon* would often

use libc’s *system()* function to invoke further functionality. While called with static parameters for most instances, we found two identical cases of an unsafe format string, allowing attackers to inject a string into a shell command:

```
echo txvoiceiirs %d %d %d %s > /run/alsaucm_test
and
echo rxvoiceiirs %d %d %d %s > /run/alsaucm_test
```

Although the corresponding AT handlers apply a character block list preventing usual shell escapes, we found a bypass by injecting a newline character.

Example Attack. We implemented a two-stage attack, with the first stage creating a custom executable wrapper to bypass the block list. The second stage then uses this wrapper, leading to an end-to-end code execution capability obtained by dispatching *RUN AT* proactive commands from the SIM.

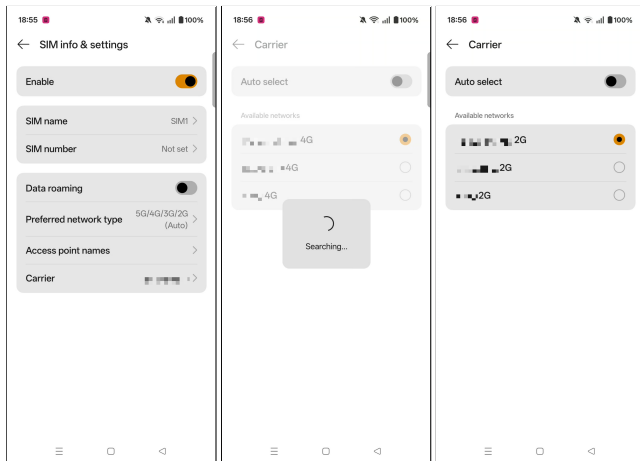


Figure 3: 2G Downgrade on OPPO Reno 14 F 5G. The left shows that all network types are selected, the center shows that the phone discovers multiple 4G networks, and the right shows that, after the downgrade, the phone only sees 2G networks. The phone also switches from a 4G network to a 2G network after the attack.

7.2.2 Denial-of-Service & 2G Downgrade

Target and Context. Contemporary smartphones often aggregate processing into one System-on-Chip (SoC), featuring an AP, frequently running Android or iOS, and a CP running a proprietary firmware from the vendor, e.g., Qualcomm, MediaTek, Samsung, or Huawei. The OPPO Reno 14 F 5G, chosen as our target, is based on the Qualcomm SM6450 SoC and supports the *RUN AT* proactive command.

Attack Surface Identification. For initial analysis, we obtained a copy of the Over-the-Air (OTA) firmware package for our target. To identify supported AT commands from the modem binary (*NON-HLOS.bin*), we used a regex-based search approach similar to Tian et al. to extract AT commands from the baseband firmware [44]. At the time of writing, the command list collected by Tian et al. was not available; therefore, we enhanced the 211 commands collected from the OTA firmware with another ≈ 800 commands that served as input to CATAUTO, our automated AT tester. We fed the obtained AT responses into CATVIEW to filter out unsupported commands. This led to 198 AT commands and their variants supported by the AT interface offered to the SIM card.

To characterize the potential security impact and to understand the parameter format of each command, we examined publicly available documentation and information. Vendors tend to re-use proprietary AT commands across many series of modems, including those intended for public use by developers, resulting in a rich attack surface.

Example Attacks. To run our attacks, we leverage an interactive AT shell using CATTTY. We leverage the following three AT commands for proof-of-concept attacks:⁶

1. *AT\$QCPWRDN* for powering down the phone (temporary device DoS until reboot),
2. *AT+CFUN=0* for powering down the CP (temporary modem DoS until reboot),
3. *AT+COPS=0,,0* for restricting access to 2nd generation cellular networks (2G downgrade).

We show the 2G downgrade in Figure 3. The downgrade cannot be mitigated by the user performing any of the following actions: toggling airplane mode, switching to manual network selection (only shows 2G networks), disabling the SIM card in cellular settings, toggling mobile data, or changing network generation preferences in the cellular tab.

7.2.3 Arbitrary File Exfiltration

Target and Context. Most Quectel modules support AT commands for file system interactions. Depending on the module's series, the user can choose between different storage locations for their files, including a User File Storage (UFS) residing on flash memory, a Security File System (SFS), RAM storage, and an external SD card [39–41]. However, regardless of the location, reverse engineering showed that the underlying file systems are not implemented by the cellular processor itself, but handed off to the accompanying Android AP by leveraging the Trivial File Transfer Protocol (TFTP). The AP stores the files in dedicated directories (e.g., */data/ufs* for the UFS on the EG-25G module).

Vulnerability. We found outdated firmware versions for Quectel modems that exhibit a direct path traversal vulnerability for the file system interactions. An attacker could use relative paths to break out of the dedicated storage area, freely accessing, creating, and modifying files in other locations on the file system.

While this vulnerability is fixed on newer firmware versions, we noticed that the tftp daemon on the AP does not check whether files are ordinary files or symbolic links. While the AT interface to the file system cannot introduce symbolic links by itself, we hypothesize that there are multiple ways for an unprivileged, physical attacker to contrive such a file system. For instance, for modules with SD card support, this becomes trivial, as files on the SD card can be modified offline before re-inserting the SD card to the target device. While most modules exhibit such support, the platforms on which they are integrated most commonly are not. In these cases, an

⁶We also provide proof-of-concept videos to illustrate attacks against phones, available from <https://doi.org/10.5281/zenodo.21236017>.

attacker has another way to manipulate the file system: Flashing a custom `usrdata` partition to manipulate the contents of the UFS directory.⁷

As Quectel APs are Android-based, they also exhibit a fastboot interface, which can be entered by driving special pins high during boot, using Quectel-specific tooling, or using the `AT+QFASTBOOT` command if supported by the module. Thus, an attacker could prepare a malicious partition, including a symbolic link, and flash it to the victim module via fastboot.

Example Attack. While we found the symbolic-link-based path traversal vulnerability present on all tested modules, we focus this part of the case study on the EG25-G modem. After inserting a malicious symbolic link on the `ufs` partition, we use `CATTTY` in interposer mode to exfiltrate the contents of the linked file.

We interpose a commercial SIM, allowing us to first establish a legitimate data connection to the internet via SIM AT commands. Then, we use the Quectel proprietary `AT+QSMTP` commands [38] to prepare an email including the victim file as an attachment. We then send-out the file to an attacker-controlled server, disclosing its contents.

Beyond File Exfiltration. Quectel also provides AT commands for directly operating on files, such as `AT+QFOPEN`, `AT+QFCLOSE`, `AT+QFWRITE`, and `AT+QFREAD`. While we could successfully open and close files via the SIM AT interface, direct read and write operations did not succeed. The underlying reason is that read and write commands put the AT interface into a *data mode* [39] that does not seem to be supported by the SIM AT interface.

However, we note that this vulnerability could still be exploited via the regular USB AT interface if present. For most consumer modems, the symbolic-link-based file access may even enable code execution, as the TFTP daemon runs with root privileges. By writing to files such as `/sys/kernel/uevent_helper` or `/proc/sys/kernel/hotplug` [4], scripts at an attacker-controlled location can be executed—including scripts located within the previously prepared file storage locations.

8 Mitigation

We propose multiple mitigations to reduce the dangers of SIM AT-based attacks. The simplest approach is to deprecate support for the SIM AT interface, alongside any other proactive commands introducing security risks. These interfaces had legitimate use cases in historical applications when modems and connected devices were resource-constrained and required out-of-band management. However, we believe that

⁷We note that most other partitions of the Quectel firmware are cryptographically signed and validated during boot to make them resistant to attacker modification.

the security risks of proactive commands, including `RUN AT`, outweigh their benefits in modern communication stacks.

This intuition is already reflected in the observed support in recent smartphones, where supporting the SIM AT interface is the exception. However, instead of disabling `RUN AT` support on the CP based on a block list, we argue that a comprehensive mitigation should also remove associated code paths from the CP firmware and the AP's software stack to avoid reintroduction of this attack surface through regressions.

As deprecating dangerous proactive commands requires cooperation from vendors and manufacturers, we also believe that there is value in consumer-facing mitigations. For instance, detection and prevention frameworks leveraging low-level modem logs, such as RILDefender [48] or BaseTrace [8], could be extended to detect artifacts of AT commands originating from hostile SIMs. Additionally, hardened builds of Android could consider disabling the processing of SIM AT commands. While this does not protect against AT commands directly handled on the CP, it reduces the attack surface, limiting the potential impact of SIM AT commands.

Lastly, following the suggestions offered in recent work [27], another route for mitigation are defenses based on SIM interposers, i.e., small physical machine-in-the-middle implants between SIM and CP. Newly developed or repurposed interposers could drop proactive `RUN AT` commands before they even arrive at the CP, allowing for a hardware-based defense against attacks from hostile SIMs. We leave the exploration of such mitigations to future work.

9 Discussion

Impact of SIM AT attacks. As shown by previous works [18, 44] and our case studies, SIM AT attacks enable a diverse set of attacker capabilities. Beyond Denial-of-Service rendering the cellular connection unusable, most of these capabilities can serve as an entry vector for further exploitation.

All surveyed MEs report basic information about the device type, firmware version, and even the corresponding IMEI via SIM AT commands, allowing attackers to fingerprint a device/user before launching targeted attacks. Downgrading connections to 2G without user consent eases fake base station attacks due to the lack of mutual authentication in 2G. Transceivance of SMS using AT commands further facilitates hostile SIMs to establish a Command and Control (C&C) channel to attacker controlled infrastructure such that initiating calls can turn a victim ME into an eavesdropping device.

Moreover, code execution on the internal AP of an IoT modem may be used as a stepping-stone for compromising the main system it is integrated with. In the simplest case, a compromised internal AP allows for machine-in-the-middle attacks. However, we also noticed that firmware running on the IoT's main CPU may expect only benign inputs from the connected modem. For instance, the main firmware for one of the surveyed devices contains a heap-based buffer

overflow when receiving a malformed IP address from the modem's internal AP during connection setup, demonstrating the additional dangers of a compromised internal AP.

Limitations. While the CATANA suite provides comprehensive access for exploring SIM AT commands and attacks, we acknowledge that our tools are not without limitations.

A main roadblock for CATTTY is synchronization of file system state between the emulated and interposed SIM. To avoid inconsistencies, we drop all requests from the ME that seek to update the file system's contents. While this may cause malfunctions of the ME, we did not encounter such an issue in our testing.

The automated testing loop provided by CATAUTO is affected by a fundamental implementation of the proactive SIM protocol: the processing of commands such as *RUN AT* is processed asynchronously. As a result, when CATAUTO does not receive a response to a sent SIM AT command in a given time window, it cannot distinguish between a still running AT command, a timeout in processing, and a crash of the ME. Thus, we recommend analysts to manually inspect the ME's behavior whenever CATAUTO encounters such a situation.

Lastly, when designing the AT specification for CATVIEW, we prioritized simplicity over full expressiveness. While the specification allows to encode the structure for the majority of AT commands, it does not include semantics for optional and context-dependent AT parameters as for instance used by vendor-specific configuration commands. Nonetheless, we encountered only a minor amount of commands affected by this limitation which was not hindering our testing. We leave deriving a grammar for expressing such semantics and integrating it with CATANA to future work.

Beyond SIM AT attacks. The attacks described in this paper build on the proactive *RUN AT* command. While we deliberately focus on this specific command in our paper, previous work has shown the general dangers of proactive commands, including memory corruption in parsers [22], real-world spyware [42], and launching a browser [10], using the *SEND SS & SEND SMS*, *PROVIDE LOCAL INFORMATION*, and *LAUNCH BROWSER* commands, respectively.

While researching SIM AT commands, we also discovered a new variation of these previously-known dangerous proactive commands. On recent versions of Android, a hostile SIM can force a phone to visit an attacker controlled website without any user interaction using the *LAUNCH BROWSER* command. Surprisingly, this worked even if the phone's lockscreen is active, providing a potential entry-point for a more complex exploit chain.⁸

These examples show that proactive SIMs and the general attack surface available to hostile SIMs requires additional

scrutiny, as most of the according interfaces were built with benign SIMs in mind. With CATANA, we provide a set of tools to extensively explore a subset of this attack surface and we hope that our findings will inspire additional hardening against hostile SIMs.

10 Conclusion

In this paper, we examined the dangers of SIM-originating AT commands. We presented CATANA, a series of tools for exploring the SIM AT interface and assessing the attack surface available to hostile SIMs. We surveyed a range of smartphones and IoT devices to understand the prevalence of the SIM AT interface in representative devices. We also conducted 4 case studies demonstrating 4 practical attacks against real-world devices and discussed potential defenses.

In conclusion, our study showed that hostile SIMs remain a security threat for the mobile communication landscape, especially in the context of IoT devices. We hope that our findings are insightful to the community and contribute towards a shift to a more secure ecosystem by deprecating legacy interfaces available to SIMs.

Open Science

CATANA is available from: <https://open.liskra.org/git/repo/catana/catana.git>.

We also archive a stable version of CATANA at: <https://doi.org/10.5281/zenodo.21236119>.

Ethical Considerations

Harms vs. Benefits. This paper discusses tooling for identifying vulnerabilities in mobile communication devices and presents 4 vulnerabilities embedded in real-world devices.

The tools we created and the information about the presented vulnerabilities could be abused for malicious purposes, e.g., for surveilling users, gaining illegitimate access to devices, or disrupting cellular communication. To reduce the potential for abuse, we follow the coordinated vulnerability disclosure practices as outlined below. We further refrain from publishing “*weaponized*” tooling, such as non-interactive applets directly compromising a victim device.

Our decision to publish this research is based on our belief that the creation of secure systems requires in-depth knowledge about potential hazards. While publishing the discovered vulnerabilities may put users at risk in the short-term, the findings of our work will be instrumental in deriving long-lasting mitigations and is likely to lead to more secure systems in the long-term. To minimize the risks posed to users, we coordinated with affected vendors to ensure that patches for the discussed vulnerabilities are available before publication of this work.

⁸We reported our finding to Google, who acknowledged this vulnerability under CVE-2025-48618 and released a patch.

Coordinated Disclosure

We disclosed the findings discussed in this paper to: Google, Oppo, Quectel, Semtech (formerly Sierra Wireless), and Qualcomm, in March 2026. After preliminary responses from vendors, we decided to further inform GSMA in May 2026.

Oppo and Google triaged issues outlined in our report and considered them informative, but outside the scope of their respective vulnerability reward programs. Semtech confirmed our findings and plans to integrate patches created by the baseband vendor, Qualcomm. Quectel confirmed our findings, but noted that the shell command injection was already known and fixed in newer firmware versions. At the time of writing, Quectel implemented a mitigation for the symbolic link-based attack and is developing a mitigation for the exposed SIM AT interface. Qualcomm confirmed our findings and created a hardened configuration that disables the SIM AT interface by default. GSMA disseminated our report to the CVD panel for triaging and tracks the issue under CVD-2026-0122.

Acknowledgments

We would like to express our sincere gratitude to Mihai Ordean and Fernanda Marinho Garcia for their valuable feedback on this manuscript.

Parts of this work were funded through the NGI Mobifree fund project “SIMcurity”. The NGI Mobifree fund is established by NLnet with financial support from the European Commission’s Next Generation Internet programme under the aegis of DG Communications Networks, Content and Technology. The NGI Mobifree R&D programme is part of the Horizon Europe research and innovation programme under grant agreement No. 101135795.

References

- [1] 3GPP. Security mechanisms for the (U)SIM application toolkit. TS 23.048 V5.9.0, June 2005.
- [2] 3GPP. Universal Subscriber Identity Module (USIM) Application Toolkit (USAT). TS 31.111 V19.3.0, September 2025.
- [3] 3GPP. AT command set for User Equipment (UE). TS 27.007 V19.6.0, March 2026.
- [4] Alex Plaskett and McCaulay Hudson. Technical Advisory: Tesla Telematics Control Unit - ADB Auth Bypass. <https://www.nccgroup.com/research/technical-advisory-tesla-telematics-control-unit-adb-auth-bypass/>, 2025.
- [5] João Amarante and João Barros. Exploring USB connection vulnerabilities on android devices breaches using the android debug bridge. 2017.
- [6] Manuel E. Correia André Pereira and Pedro Brandão. Charge Your Device with the Latest Malware. Black Hat EU, 2014.
- [7] Jacob Applebaum, Judith Horchert, Ole Reissmann, Marcel Rosenbach, Jörg Schindler, and Christian Stöcker. NSA’s Secret Toolbox: Unit Offers Spy Gadgets for Every Need. Spiegel International, [spiegel.de](https://www.spiegel.de), 2013.
- [8] Lukas Arnold, Matthias Hollick, and Jiska Classen. Catch You Cause I Can: Busting Rogue Base Stations Using Cellguard and the Apple Cell Location Database. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2024.
- [9] David Briddock. Tech Origins Modem. *Micro Mart*, (1313):28, May 2014.
- [10] David Burgess. Proactive SIMs. DeepSec 2021, deepsec.net, 2021.
- [11] ETSI. Card Application Toolkit (CAT). TS 102 223 V18.3.0, June 2026.
- [12] ETSI. Universal Subscriber Identity Module (USIM) Application Toolkit (USAT). TS 131 111 V19.3.0, January 2026.
- [13] Matheus E Garbelini, Zewen Shang, Shijie Luo, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. 5ghoul: Unleashing chaos on 5 g edge devices via stateful multi-layer fuzzing. *IEEE Transactions on Dependable and Secure Computing*, 2025.
- [14] Gabriel K Gegenhuber, Philipp É Frenzel, and Adrian Dabrowski. Poster: SIMulator: SIM Tracing on a (Pico-) Budget. In *18th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 288–289, 2025.
- [15] Gabriel K Gegenhuber, Philipp É Frenzel, and Adrian Dabrowski. SIMulator: SIM Tracing on a (Pico-) Budget. *arXiv preprint arXiv:2506.20800*, 2025.
- [16] Roe Hay. fastboot oem vuln: Android Bootloader Vulnerabilities in Vendor Customizations. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2017.
- [17] Martin Herfurt and Collin Mulliner. Blueprinting: Remote device identification based on Bluetooth fingerprinting techniques. 2004.
- [18] Imtiaz Karim, Fabrizio Cicala, Syed Rafiul Hussain, Omar Chowdhury, and Elisa Bertino. Opening Pandora’s box through ATFuzzer: dynamic analysis of AT interface for Android smartphones. In *Annual Computer Security Applications Conference (ACSAC)*, 2019.

- [19] David U. Kim. Cds 224 autodial modem and pc-talk iii. volume 3, pages 28–29, 02 1987.
- [20] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. Touching the untouchables: Dynamic security analysis of the lte control plane. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.
- [21] Karl Koscher and Eric Butler. The Secret Life of SIM Cards. DEF CON, 2013.
- [22] Tomasz Piotr Lisowski, Merlin Chlosta, Jinjin Wang, and Marius Muench. SIMurai: Slicing Through the Complexity of SIM Card Security Research. In *USENIX Security Symposium*, 2024.
- [23] Charlie Miller, Dion Blazakis, Dino DaiZovi, Stefan Esser, Vincenzo Iozzo, and Ralf-Philip Weinmann. Base-band Attacks. In *iOS hacker’s handbook*, pages 327–363. John Wiley & Sons, 2012.
- [24] Ministry of Science and ICT (South Korea). MSIT Releases Final Investigation Results on SK Telecom Data Breach. korea.net, 2025. [Online, accessed July 7, 2026].
- [25] MITRE FiGHT. FGT1195.501. Supply Chain Compromise: SIM Credentials Theft. fight.mitre.org, 2025. [Online, accessed July 7, 2026].
- [26] Frank Morgner and Dominik Oepen. VSmartCard. github.com. [Online, accessed July 7, 2026].
- [27] Marius Muench and Tomasz Piotr Lisowski. SIMsalabim: Blitz and Tricks with SIM cards. RE//verse.io, youtube.com, 2025.
- [28] Collin Mulliner, Steffen Liebergeld, Matthias Lange, and Jean-Pierre Seifert. Taming Mr Hayes: Mitigating signaling based attacks on smartphones. In *Conference on Dependable Systems and Networks (DSN)*, 2012.
- [29] Collin Mulliner and Charlie Miller. Fuzzing the Phone in your Phone. Black Hat USA 2009, blackhat.com, 2009.
- [30] Nordic Semiconductors. nRF Connect SDK - 2.5.3. <https://docs.nordicsemi.com/bundle/ncs-2.5.3/page/nrf/index.html>, 2024.
- [31] Onomondo. Onomondo software SIM/USIM implementation. github.com. [Online, accessed July 7, 2026].
- [32] Osmocom Contributors. pySIM. Osmocom Wiki, osmocom.org. [Online, accessed July 7, 2026].
- [33] Osmocom Contributors. SIMtrace2. Osmocom Wiki, osmocom.org. [Online, accessed July 7, 2026].
- [34] Mano Paul. Bluesnarfing. In *Information Security Management Handbook, Volume 3*, pages 355–364. Auerbach Publications, 2009.
- [35] Michele Pepe. Dennis Hayes. *CRN: The Newsweekly for Builders of Technology Solutions*, (1075), 2003.
- [36] André Pereira, Manuel Correia, and Pedro Brandão. USB Connection Vulnerabilities on Android Smartphones: Default and Vendors’ Customizations. In Bart De Decker and André Zúquete, editors, *Communications and Multimedia Security*, pages 19–32, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [37] Waqas Ahmad Piracha, Morshed Chowdhury, Biplob Ray, Sutharshan Rajasegarar, and Robin Doss. Insider Attacks on Zigbee Based IoT Networks by Exploiting AT Commands. In V. S. Shankar Sriram, V. Subramaniaswamy, N. Sasikaladevi, Leo Zhang, Lynn Batten, and Gang Li, editors, *Applications and Techniques in Information Security*, pages 77–91, Singapore, 2019. Springer Singapore.
- [38] Quectel. EC2x&EG9x&EM05 SMTP AT Commands Manual. Technical report, 2017.
- [39] Quectel. LTE Standard FILE Application Note. Technical report, 2020.
- [40] Quectel. EC200U & EG915U Series FILE Application Note. Technical report, 2021.
- [41] Quectel. RG50xQ & RM5xxQ Series FILE Application Note. Technical report, 2022.
- [42] Adaptive Mobile Security. Simjacker Technical Paper. enea.com, Version 10OCT19-v1.01, 2019.
- [43] sysmocom. sysmoISIM-SJA5 programmable SIM/USIM/ISIM/HPSIM cards. sysmocom.de. [Online, accessed July 7, 2026].
- [44] Dave (Jing) Tian, Grant Hernandez, Joseph I. Choi, Vanessa Frost, Christie Raules, Patrick Traynor, Haywardh Vijayakumar, Lee Harrison, Amir Rahmati, Michael Grace, and Kevin R. B. Butler. ATtention Spanned: Comprehensive Vulnerability Analysis of AT Commands Within the Android Ecosystem. In *USENIX Security Symposium*, 2018.
- [45] Patrick Traynor, Chaitrali Amrutkar, Vikhyath Rao, Trent Jaeger, Patrick McDaniel, and Thomas La Porta. From mobile phones to responsible devices. *Security and Communication Networks*, 4(6):719–726, 2011.
- [46] Ivan Vaccari, Enrico Cambiaso, and Maurizio Aiello. Remotely Exploiting AT Command Attacks on Zig-Bee Networks. *Security and Communication Networks*, 2017(1):1723658, 2017.

- [47] Ralf-Philipp Weinmann. Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2012.
- [48] Haohuang Wen, Phillip A Porras, Vinod Yegneswaran, and Zhiqiang Lin. Thwarting Smartphone SMS Attacks at the Radio Interface Layer. In *Symposium on Network and Distributed System Security (NDSS)*, 2023.
- [49] Jinghao Zhao, Boyan Ding, Yunqi Guo, Zhaowei Tan, and Songwu Lu. SecureSIM: Rethinking Authentication and Access Control for SIM/eSIM. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2021.