Securing Wireless Coprocessors from Attacks in the Internet of Things

Jason Staggs, Sujeet Shenoi

Chapter 9

SECURING WIRELESS COPROCESSORS FROM ATTACKS IN THE INTERNET OF THINGS

Jason Staggs and Sujeet Shenoi

Abstract Wireless communications coprocessors are a vital component of numerous Internet of Things and mobile devices. These subsystems enable devices to communicate directly with peers and supporting network infrastructures. Previous research has shown that wireless communications coprocessors lack fundamental security mechanisms to combat attacks originating from the air-interface and application processor (main CPU). To mitigate the risk of exploitation, methods are needed to retroactively add security mechanisms to communications coprocessors.

This chapter focuses on securing a cellular baseband processor from attacks by hostile applications in the application processor. Such attacks often leverage attention (AT) commands to exploit vulnerabilities in baseband firmware. The attacks are mitigated by installing an AT command intrusion prevention system between the application processor and baseband processor interface.

Keywords: Wireless coprocessor, Internet of Things, intrusion prevention

1. Introduction

A Statista report [29] estimates that nearly 31 billion Internet of Things (IoT) devices will be in use by 2020. Consumer demand for smart devices has surged, launching a time-to-market race by manufacturers to release Internet of Things devices with rich features at affordable prices. Unfortunately, security has taken a back seat to features, significantly increasing the risks to devices, networks and users [16].

Further complicating matters are the diversity and complexity of wireless protocols and communications systems that interconnect Internet of Things devices [22]. Internet of Things devices use wireless coprocessors and protocols to support communications with smart devices and networks. These communica-

tions coprocessors are normally separate microcontrollers that are independent of the main CPUs. The heterogeneity of wireless communications coprocessors and protocol stacks has inadvertently increased the attack surfaces of Internet of Things and mobile devices. The security problems associated with wireless communications coprocessors are also inherited by devices such as remote terminal units and programmable logic controllers that help operate critical infrastructure assets.

Previous research has demonstrated that wireless communications coprocessors lack rudimentary security measures to combat attacks, especially data execution prevention (DEP), address space layout randomization (ASLR) and basic memory protections [2, 6, 14, 33, 34]. Although it is important to consider attacks that target the main CPUs of devices, it is equally imperative to consider attacks that focus on wireless network coprocessors. Internet of Things and mobile devices must be engineered to be more resilient to attacks that target communications coprocessing units, especially if the security threats posed by insecure devices that plague the Internet [16] and some operational technology environments are to be reduced.

This work focuses on mitigating attacks by applying security defenses to a specific type of wireless communications coprocessor – the cellular baseband processor. Baseband processors, also known as cellular modems, are present in all mobile phones and in many Internet of Things devices and industrial control systems that require cellular wide-area networking connections to the Internet [27]. These independent systems provide direct, unfiltered radio access to public cellular GSM, UMTS and LTE networks, and are attractive targets for attackers who seek to intercept, modify, fabricate or interrupt communications.

Despite concerns about attacks on communications coprocessors, relatively few attempts have been made to secure their external interfaces [20]. This work addresses the gap by mitigating exploitation attempts from the application processor (main CPU) that leverage malformed or unauthorized vendor-specific serial AT commands to target the baseband processor. Such attacks are commonly employed to unlock cell phones, but they can be repurposed to perpetrate nefarious baseband system compromises [13, 14, 30].

This chapter describes a proof-of-concept application-processor-interface-based AT command intrusion prevention system that combats exploitation attempts against the baseband processor. The intrusion prevention system relies on rules (signatures) based on AT command syntax and semantics. The signatures help detect and prevent malicious AT commands and payloads (parameters) from being sent to the baseband processor. The proof-of-concept system incorporates a Raspberry Pi 3 hardware platform for main CPU emulation and a SIM900 GSM module (baseband processor). Empirical testing reveals that the system combats baseband processor attacks from the application processor. Although the system employs hobbyist hardware, the underlying techniques can be applied to Internet of Things and mobile devices by making slight modifications to their operating systems or by adding a dedicated security coprocessor that inspects input/output messages on the data bus.

2. Security of Communications Stacks

The vast majority of Internet of Things and mobile devices incorporate wireless communications processors for wide-area networking and personal area networking radio needs. These independent coprocessors facilitate communications between other smart devices, sensors, motors, relays and supporting telecommunications infrastructure assets (e.g., cellular base stations). Communications technologies such as 802.15.4 (i.e., Zigbee and WirelessHART), 802.11 (i.e., Wi-Fi), Bluetooth and GSM/UMTS/LTE have emerged over the years and are now widely integrated in Internet of Things devices. In many cases, the wireless chipsets operate independently of the main CPUs and, therefore, have their own attack surfaces [17].

Previous research has demonstrated that wireless communications coprocessors lack basic security mechanisms. Although it is important to consider attacks that target the main CPUs of devices, it is imperative that future Internet of Things and mobile devices are engineered to be resilient to attacks that target communications coprocessors. Researchers have identified a number of security problems and vulnerabilities in wireless coprocessors and protocol stacks [2, 6, 33, 34]. Some attacks target vulnerabilities in coprocessor firmware while others target kernel modules or libraries used by the operating systems of the main CPUs to interact with wireless subsystems on the devices [26].

In particular, several vulnerabilities have been identified in Broadcom Wi-Fi chipsets. Beniamini [3–5] demonstrated heap and stack overflow vulnerabilities that lead to remote code execution (RCE) on a common Broadcom Wi-Fi chipset. These vulnerabilities stem from inadequate data field checking by the Wi-Fi coprocessor unit when processing certain 802.11r-2008 (fast BSS transition) authentication frames. Building on Beniamini's work, Artenstein [2] weaponized the exploits by incorporating them in a propagating worm that could target other Wi-Fi chipsets. Meanwhile, Seri and Livne [26] have developed a new attack called "BlueBorne" that targets Bluetooth implementations in operating systems used by billions of Internet of Things and mobile devices. Arguably, the most terrifying wireless chipset attacks involve the exploitation of cellular baseband processors.

Cellular baseband stack exploits have been discussed for nearly a decade [10, 33, 34]. The exploitation of communications coprocessors presents an existential threat to the devices they support, and by extension, the physical environments in which they are used (e.g., industrial Internet of Things and industrial control systems). Novel ideas, techniques and mechanisms are needed to counter the threats to frail and insecure communications coprocessors.

This chapter presents an approach for retrofitting security in one of the external interfaces used by baseband processors in order to mitigate hostile activity originating from the main CPU. Admittedly, this is not a perfect solution to the overall problem, but the approach is useful when security controls have to be implemented in current mobile phones and legacy industrial control systems.

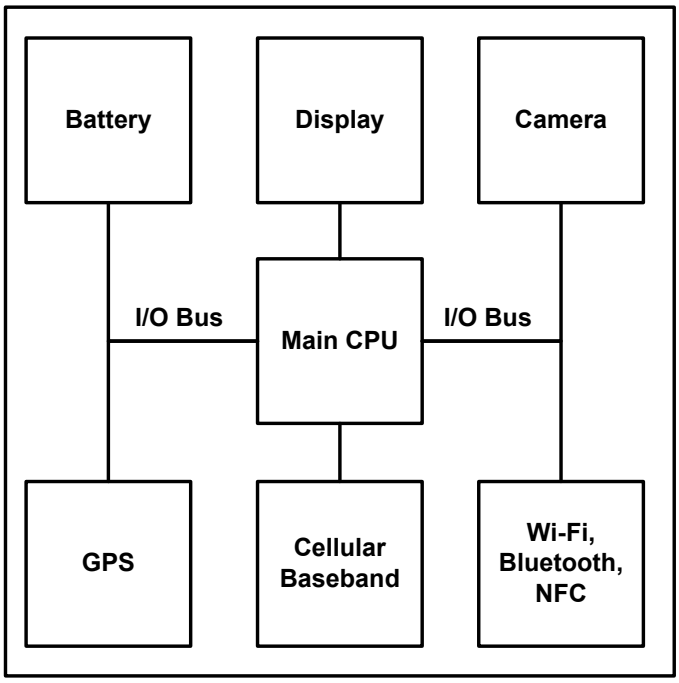

Figure 1. System of systems in a smartphone.

3. Cellular Baseband Processors

Baseband processors – also called cellular modems – are present in Internet of Things devices and mobile phones that require wide-area networking connections to the Internet and other cellular functions. The independent baseband systems, which provide direct, unfiltered access to public cellular networks, are prime targets for attackers interested in intercepting, modifying, fabricating or blocking voice, text, data and signaling traffic.

3.1 Symbiotic System of Systems

Internet of Things devices and modern mobile phones incorporate supporting microcontrollers and subsystems, each serving a dedicated and crucial role. These subsystems are interconnected directly or indirectly over a common data bus, enabling the smart device to provide rich interactions with the external environment via sensors (e.g., camera, microphone, gyroscope and accelerometer) and radios. Because the microcontrollers operate independently of the main CPUs, they have their own firmware that is usually stored in read-only memory (ROM).

Figure 1 presents a logical view of the peripheral systems in a smartphone. Examples include the GPS, camera, battery, Wi-Fi, near-field communications (NFC), Bluetooth and cellular baseband systems.

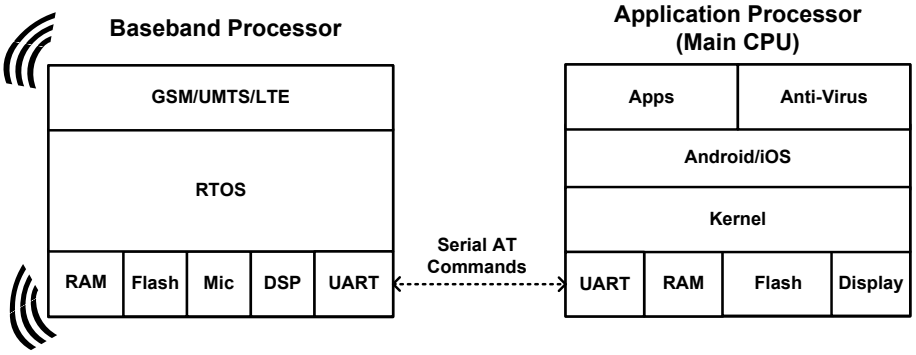

Figure 2. Relationship between the application processor and baseband processor.

Although the systems are viewed as being independent, it is important to note that some of these systems are often encapsulated in a single integrated circuit called a "system on a chip" (SoC). The systems may be housed in a single integrated circuit package, but a common input/output interface is still required for them to exchange commands and data (e.g., UART, CAN, SPI, I2C, USB or shared memory). Just like any other embedded system, each encapsulated system has its own volatile memory (RAM) and non-volatile memory (flash), along with various peripherals. This architecture enables the main CPU to focus on its operating system and user applications, while relying on the independent supporting systems to handle other tasks and provide external data upon request via hardware interrupts. It is common for the tasks performed by these systems to be subject to stringent synchronization and timing constraints, as in the case of a cellular protocol stack that runs on a baseband processor.

The two main components of cellular-enabled Internet of Things and mobile devices are the application processor and baseband processor. Figure 2 shows the relationship between the two processors [8].

The application processor system houses the main CPU and operating system (e.g., Linux, Android, Windows IoT or iOS). User applications and display interfaces execute on the application processor.

The baseband processor system serves as the cellular modem that independently handles cellular network communications between the mobile device and cell towers, including signaling for radio resource management, mobility management, connection management, voice calls, SMS text messages and cellular data. Like the application processor system, the baseband processor system contains dedicated hardware peripherals such as RAM, flash memory and digital signal processor, and provides direct access to the cell phone speaker and microphone [25]. The baseband processor system is analogous to a dial-up modem or Ethernet controller in a personal computer, providing layer 1 modulation and demodulation of carrier signals to encode and decode information over dedicated cellular links.

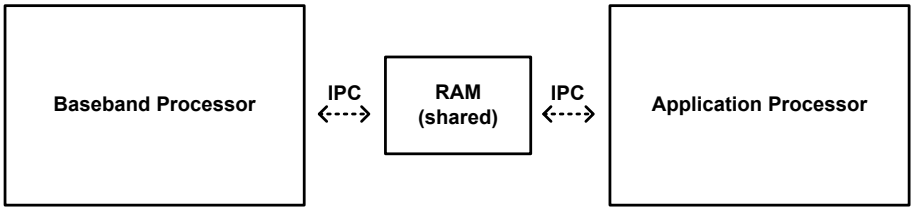

Figure 3. Shared memory architecture.

3.2 Baseband Firmware

In order to manage hardware resources and perform cellular modem tasks, the baseband processor typically runs a bootloader that initializes the hardware and loads a real-time operating system (RTOS) into memory [35]. The real-time operating system executes tasks for the entire cellular network stack (e.g., GSM, UMTS and LTE) [18, 34]. The tasks, which are engineered to provide reliable signal connectivity to carrier networks, generally have the requirement of minimal power consumption during operation.

Although baseband system codebases have been updated over the years to accommodate the latest cellular protocol specifications, significant portions of early GSM, UMTS and LTE codebases are still used in modern baseband stacks. Implementing the 3GPP cellular protocol specifications in software is a complex endeavor that can be ambiguous in some instances. Unfortunately, the messages used by cellular protocol stacks sometimes contain high concentrations of variable length fields that are not handled properly and, thus, can be targeted by fuzzing and vulnerability discovery activities. These factors create a perfect storm for exploiting baseband systems [12, 34].

It is important to note that cellular baseband processors are not limited to mobile phones. In fact, the processors are commonly found in Internet of Things devices such as vehicle telematics units, automated teller machines and smart meters used in electricity, gas and water distribution infrastructures. All these devices typically require low bandwidth connectivity to transmit data.

3.3 Baseband Architectures

Command and data exchange between the application and baseband processors depend on the device architecture. The two architectures are: (i) shared memory architecture; and (ii) independent memory architecture. Figures 3 and 4 illustrate the two architectures [8, 33].

In the shared memory architecture, the application processor and baseband processor address spaces are mapped to the same physical RAM [33]. In this case, a form of interprocess communications is employed to exchange information between baseband and application processes.

The independent memory architecture is more commonly used in Internet of Things devices and modern phones. The architecture requires a dedicated data

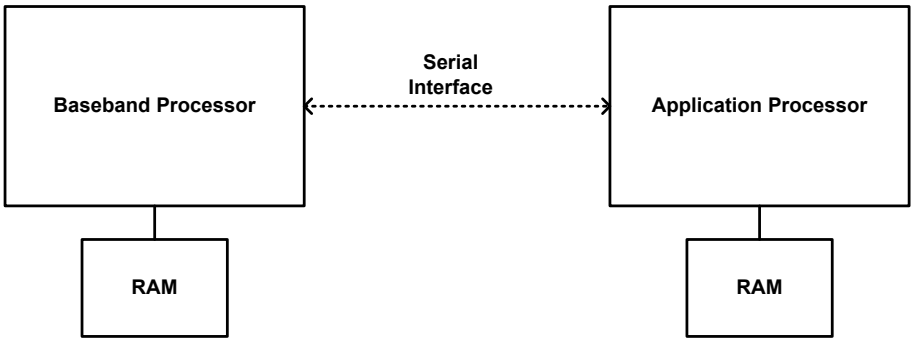

Figure 4. Independent memory architecture.

bus between the application and baseband processors to facilitate communications. Common data bus communications interfaces are UART, SPI, I2C and USB.

3.4 Serial Communications Protocols

Baseband processor command and control protocols vary from device to device. The protocols are used to instruct the baseband to execute cellular functions such as making a call and sending data. The protocols are also used to send information back to the application processor (e.g., notification of an incoming call, SMS or data). Common protocols include standard GSM AT commands [9, 23] and proprietary vendor-specific command protocols [7]. GSM AT commands are similar to the Hayes AT commands used by old dial-up modems [11]. Some baseband processor vendors incorporate additional proprietary commands for extended functionality and debugging; in some instances, they provide backdoors to the application processor [15, 30]. Additionally, the commands are usually sent in the clear and are not authenticated.

4. Securing the Baseband Processor

Despite the increased scrutiny leveled on communications coprocessors as potential attack targets, few attempts have been made to secure their external interfaces [13]. The application processor interface and the air-interface expose the baseband processor to untrusted data and, thus, a number of external threats (Figure 5).

Baseband firmware reverse engineering is required in order to fully appreciate the baseband system security issues. The mobile phone unlocking communities are the most advanced at understanding the complexities involved in baseband processor firmware reverse engineering and exploitation. The application processor interface is routinely leveraged by mobile phone unlocking enthusiasts to exploit vulnerabilities in baseband processors to unlock phones for use in other carrier networks. Although such attacks generally require root access

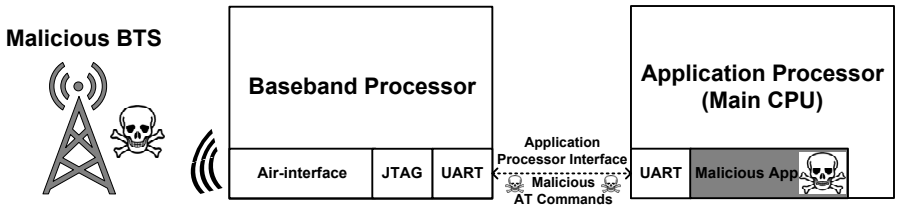

Figure 5. External input/output interfaces to the baseband processor.

to the application processor (e.g., by jailbreaking or rooting), the application processor has a rich attack surface of its own that could be used to indirectly target the baseband processor.

Tian et al. [30] describe a methodology and analysis framework for identifying and assessing vendor-specific AT commands that are injected via the USB modem interfaces of Android devices. They discovered hundreds of commands that can be used to bypass screen locks, enable developer debugging tools and perform firmware updates. Additionally, they discovered other vendor-specific AT commands that could be used to probe and potentially manipulate baseband processors. Over-the-air remote code execution attacks against the air-interfaces of baseband processors are also a concern because they could be leveraged to compromise the baseband processors remotely.

Some baseband processors incorporate a dedicated JTAG interface that can be used to help reverse engineer and/or read/write to the flash memory of the processor. JTAG is a hardware debugging interface that is commonly used by manufacturers for device testing and verification. Although this interface could provide substantial low-level access to the code, data and operational context of a baseband processor, it is outside the scope of this work. Instead, this section investigates techniques that could be used to retrofit security around the baseband processor to mitigate external attacks from the application processor interface.

4.1 Retrofitting Security

Embedded systems typically have design requirements that stress reliability and efficiency. This is especially true for embedded systems with strict power constraints and those that support human or physical processes (e.g., cell phones, programmable logic controllers and pacemakers) [19]. Unfortunately, security usually comes in second after performance requirements and is often considered only after a serious incident impacts consumers. Reactive approaches to addressing security problems are rarely robust and often have serious ramifications [16]. As a result, proactive measures that incorporate security engineering best practices must be considered early in system design and definitely before system integration.

The longevity of embedded system deployments (e.g., industrial control systems) and the lack of regular firmware updates make them ideal targets for

adversaries. The ubiquity of insecure communications interfaces and protocols contributes to inherent vulnerabilities in embedded systems that persist over their lifespans – these are collectively referred to as "forever days."

Retrofitting security in an insecure external communications interface of an embedded system is an active research challenge. Bump-in-the-wire solutions that secure unencrypted IP-based industrial control system protocols such as Modbus/TCP and DNP3 have been developed [32]. Additionally, specialized firewalls have been deployed to perform rigorous message filtering across trust domains in industrial control system environments [21, 31].

In the case of Internet of Things and mobile devices, the interface between the application and baseband processors is fundamentally insecure. The interface transports serial character streams of commands and data that initiate cellular processes in the devices [23, 24]. Unfortunately, this interface enables malware executing on the application processor to target the baseband processor with malicious commands and data.

4.2 AT Command Filtering

Mulliner et al. [20] have developed a "virtual modem" protection mechanism that mitigates malicious injections of cellular signaling traffic from mobile phones. The virtual modem mediates signaling traffic between the application processor and baseband processor. This independent system intercepts and inspects AT commands before forwarding them to the baseband processor.

The virtual modem solution incorporates an AT command filter that enforces a security policy on messages destined for the baseband processor. The policy specifies temporal thresholds on the frequencies of transmitted AT commands. The enforcement of the strict policy on AT commands that initiate critical cellular operations mitigates denial-of-service attacks. However, this solution does not address attacks on the baseband processor itself.

5. Baseband Processor Exploitation

The mobile phone unlocking communities have for years focused on understanding how proprietary baseband processors work in order to bypass network carrier locking restrictions [13]. The Apple iOS and Android communities have acquired substantial expertise in reverse engineering, vulnerability analysis and exploit development for targeting baseband processor firmware. Their focus is on baseband processor exploitation from the perspective of the application processor interface, not the air-interface.

Baseband processor exploitation over the air-interface generally occurs by leveraging remote code execution or denial-of-service vulnerabilities in frail cellular protocol stack implementations. The vulnerabilities are triggered by sending specially-crafted cellular messages (e.g., signaling, voice or data) to the baseband processor. These messages are usually malformed and are constructed to take advantage of inadequate checking of GSM/UMTS message field lengths (e.g., resulting in buffer/heap overflows) or they employ improper

variable data types (e.g., resulting in integer overflows) [33, 34]. In contrast, baseband exploitation via the application processor interface typically involves sending crafted AT commands that trigger remote code execution vulnerabilities in the AT command handler code of the baseband processor.

Modern phones support hundreds of AT commands for initiating cellular activities and providing vendor-specific features. This has created large attack surfaces for the baseband and application processors that can be leveraged by attackers to rewrite firmware, bypass security mechanisms, exfiltrate sensitive device data, unlock screens and inject touch events [30].

The approach adopted in this research is to identify suspicious AT commands and create the associated signatures. These AT command signatures are employed in real time to thwart exploitation attempts against the baseband processor.

5.1 AT Command Exploitation Methodology

The iOS and Android unlocking/jailbreaking communities are great sources for information about baseband processor exploitation. This information can also be used to assist in developing reactive and proactive security mechanisms that detect and mitigate attacks against the baseband processor. This work has leveraged the `theiphonewiki.com` community resources to understand baseband processor exploitation from the application processor. It has also drawn on vulnerability and exploit information from `theiphonewiki.com` to develop mitigation techniques.

The first step in unlocking a phone is to understand how and where network carrier locking is implemented. In most cases, this is handled by the baseband processor. Traditionally, a security researcher who intends to unlock a phone unpacks and reverse engineers the baseband firmware of the target phone. Next, the researcher identifies sections of code and data in the firmware where the carrier lock logic is implemented.

A method for remote code execution is then needed to patch the memory of the targeted baseband processor. This can be accomplished by analyzing the AT command interface handler for software vulnerabilities that provide control of the system (e.g., memory corruption). The vulnerabilities are normally triggered (exploited) using anomalous AT commands sent from the application processor. The anomalous AT commands, which are usually valid per specification, are constructed to exploit vulnerabilities in the AT command handler to gain control over the program counter of the baseband processor. In some cases, the AT commands may contain shellcode that is eventually executed in order to unlock the phone. Common memory corruption vulnerabilities that are routinely identified and exploited include stack- and heap-based buffer overflows [1, 13].

Figure 6 shows a proof-of-concept stack overflow exploit employed by the `Purplesn0w` unlock to trigger an AT command vulnerability in the iPhone 3GS X-Gold 608 baseband processor [14]. In the example, the second parameter of

```
at+xlog=1,"jjjjjjjjjjjjjjjjjjjjjjjj44445555PPPP"
j's = junk padding
R4 = 4
R5 = 5
PC = P
```

Figure 6. iPhone 3GS `Purplesn0w` AT command exploit.

the `+xlog` AT extended command is crafted to gain control of the program counter of the baseband processor.

6. AT Command Intrusion Prevention System

Data entering the baseband processor from an external source should be considered to be untrusted until it is vetted for signs of malicious behavior. A proof-of-concept AT command intrusion prevention system was developed to mitigate exploitation or dynamic vulnerability discovery attempts against the baseband processor that originate from the application processor interface. The intrusion prevention system enables users to define rules (signatures) based on the AT command syntax and semantics. The signatures are used to detect and prevent malicious AT commands and payloads (parameters) from being sent to the baseband processor. Signatures are also specified to detect and prevent fuzzing attempts and unauthorized uses of vendor-specific AT commands.

6.1 AT Command Syntax

The Hayes AT and GSM AT command sets are character-based messaging protocols (strings) that are commonly used by application processors to instruct baseband processors to perform cellular operations. Cellular routines for voice, SMS and data are invoked by AT commands to instruct baseband processors to perform the relevant functions. The AT command specifications cover hundreds of commands. This work focuses on application processor interface protocols that use standard AT and GSM AT commands, as well as proprietary vendor-specific AT commands [30]. Other types of proprietary messaging protocols exist [7]; however, the rigorous treatment of these protocols is the subject of future research.

Several categories of AT command messages, each with its own syntax, have been specified [9, 28]. The three AT command categories considered in this research are basic, extended and s-parameter commands.

- **Basic Commands:** Basic commands have the structure `AT<x><n>` or `AT&<x><n>` where `<x>` is a command and `<n>` denotes the command parameters.
- **Extended Commands:** Extended commands have the structure `AT+<x><n>` or `AT%<x><n>` where `<x>` is a command and `<n>` denotes the command parameters.

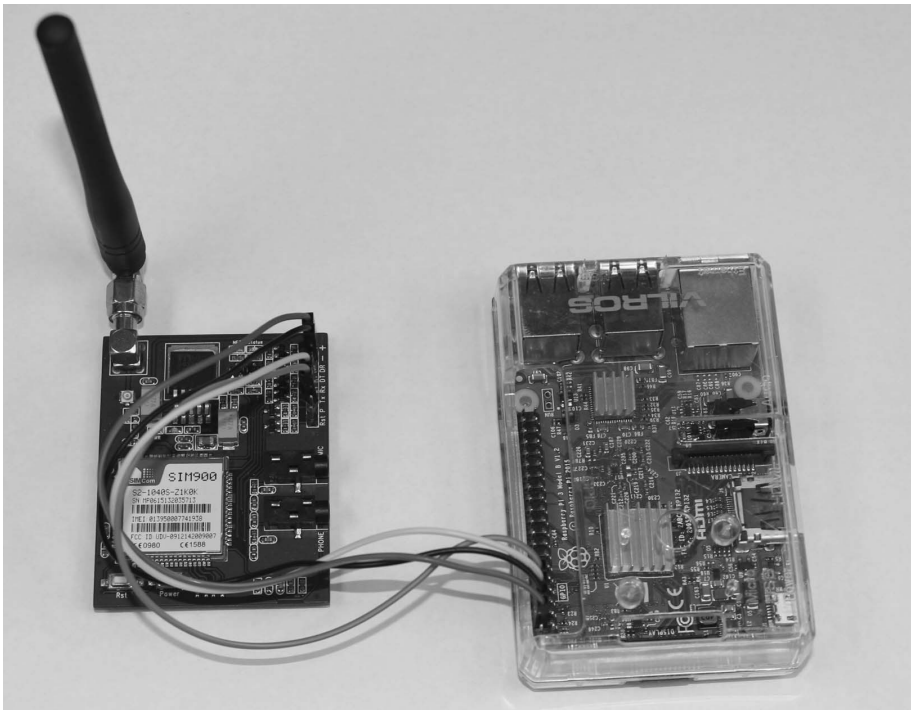

Figure 7. SIM900 GSM module and Raspberry Pi 3.

- **S-Parameter Commands:** S-parameter commands have the structure `ATS<n>=<m>` where `<n>` is the index of the s-register to be set and `<m>` is the value to be assigned.

6.2 Design and Implementation

A SIM900 GSM module and a Raspberry Pi 3 hardware platform running a Linux Debian operating system (Raspbian) were selected to develop the proof-of-concept AT command intrusion prevention system (Figure 7). The SIM900 GSM module was selected as the baseband processor because of its wide support and availability of documentation. The flexibility provided by the Raspberry Pi was leveraged to program it to emulate application processor functionality. The GSM module was connected to the GPIO pins of the Raspberry Pi to facilitate serial UART communications. OpenBTS was used as the test GSM network.

A method is needed to conduct real-time passive analysis of AT commands in transit to the baseband processor. Monitoring the transmission of AT commands requires an understanding of the dataflow from the source (application processor operating system) to the destination (baseband processor real-time

operating system). In general, AT command inspection may be performed in three ways, each requiring different levels of device access and intrusiveness: (i) adding an additional security coprocessor tasked with moderating AT commands; (ii) kernel-level modifications; and (iii) user-level library preloading [23, 24]. This research leveraged the library pre-loading technique for AT command inspection because of its ease of implementation.

In order to simulate the cellular functionality of a malicious application or rooted phone, a test application was written for the Raspberry Pi 3 to send arbitrary and malicious AT commands to the SIM900 GSM module. The application sends AT commands to the SIM900 GSM module by creating a file descriptor to the serial device `/dev/uart` and calling the `libc write()` function to write a stream of characters to the serial device.

The `write()` function is hooked to inspect AT commands before being written to the serial device. An easy way to hook a library function in Linux is to use the `LD_PRELOAD` environment variable [23, 24], which enables a designated shared library to be loaded before any other shared libraries. This precedence technique is leveraged to overwrite stock shared library symbols in order to define an alternative version of `write()`. This method gains control of the data content passed to the baseband processor.

After hooking the `write()` function with `LD_PRELOAD`, control is passed to the intrusion prevention code that checks the AT commands and data. Only verified `write()` function calls passed with a file descriptor to the baseband processor (UART device) are processed further. If a `write()` call does not contain a file descriptor that points to a valid serial device, then the function calls the regular `libc` version of `write()`.

If the altered version of `write()` is invoked, then the buffer containing the message is parsed by the intrusion prevention system and matched against the predefined rules (signatures). If a rule is triggered by a particular AT command, then the command is dropped and the function simply returns to the calling function the number of bytes that were supposed to be written to the device. This ensures that the intrusion prevention logic is transparent to the underlying application and that the application is unaware that the AT command has been dropped.

Alternatively, if the AT command does not trigger on a rule, it is passed on and written to the UART interface for serial transmission to the baseband processor. The inspection system thus moderates potentially malevolent commands and data sent over an insecure, albeit trusted, serial interface.

Figure 8 illustrates the application-processor-based AT command intrusion prevention process used in the proof-of-concept implementation.

6.3 Intrusion Prevention System

The application-processor-based AT command intrusion prevention system has three components: (i) AT command parser; (ii) rule parser; and (iii) intrusion detection/prevention engine.

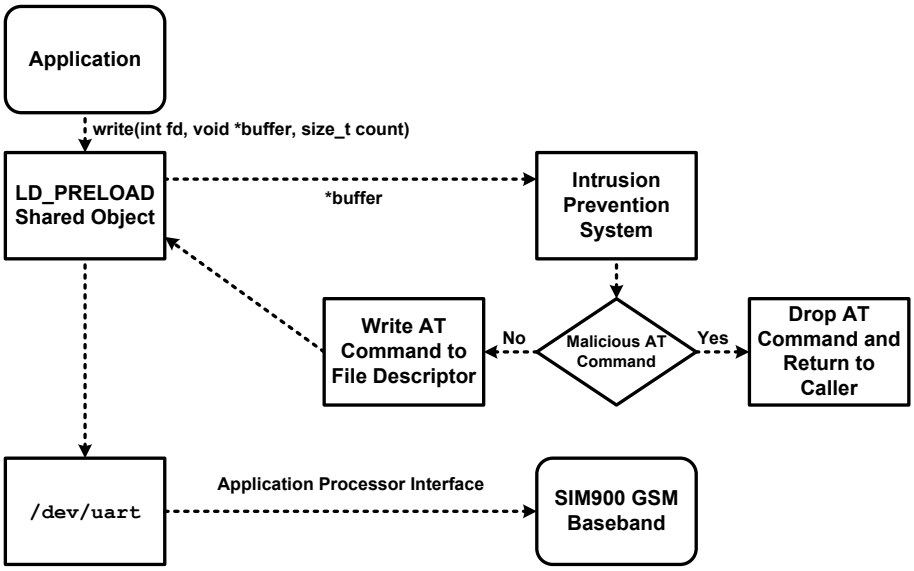

Figure 8. Application-processor-based AT command intrusion prevention system.

The AT command parser parses the command contained in the buffer that is passed to the `write()` call. In addition to performing syntactic checks, an appropriate data structure is created for the AT command (e.g., basic, extended or s-parameter). The data structure, which contains granular attributes that apply to the AT command, is employed when applying the rule-based logic.

The rule parser reads in the rules that are defined in an external configuration file. A rule has the syntax: `<operator> <command><parameter(s)>`. Note that the parameters are operator specific.

Four basic operators are employed by the intrusion prevention system:

- **Length:** This operator provides the length of an AT command parameter. It is useful for flagging AT commands with abnormal parameter lengths (e.g., used to induce stack-based buffer overflows, where the padding size of the exploit is known to be at least a certain length).
  - *Example 1:* `length +xlog 25` blocks the AT `+xlog` command when the parameter length is exactly 25 characters.
  - *Example 2:* `length +xlog >= 25` blocks the AT `+xlog` command when the parameter length is greater than or equal to 25 characters.
- **Match:** This operator matches a substring against the parameters of a specific AT command or all AT commands. If a match occurs, the AT command is blocked. The operator is useful for catching script kiddies or commonly-used shellcode snippets (e.g., used in phone unlocking attempts).

  - *Example 1:* `match "baddata"` blocks all commands containing the string `"baddata"` in their parameters.
  - *Example 2:* `match +xapp "\0xDE\0xAD\0xBE\0xEF"` blocks `+xapp` commands containing the shellcode `"\0xDE\0xAD\0xBE\0xEF"` as a parameter.

- **Msg_Sequence:** This operator triggers an alert when a contiguous sequence of AT commands is encountered; the commands are also blocked. It is useful for establishing context-based rules where a single AT command may not be malicious, but a series of commands in the specified order could be malicious.
  - *Example 1:* `msg_sequence +xlog +xlog` triggers an alert on back-to-back occurrences of the `+xlog` command; the commands are also blocked.

- **Block:** This operator adds an AT command to the blacklist, which causes the command to be dropped as soon as it is encountered. It is useful for blocking vendor-specific AT commands that could be abused by attackers [30].
  - *Example 1:* `block at%imei=` prevents the IMEI of a phone from being changed.
  - *Example 2:* `block at+fus?` prevents a phone from going into the firmware download mode.
  - *Example 3:* `block at+xabbtrace` prevents the baseband trace configuration from being returned.

7. Experimental Analysis and Testing

The application-processor-based AT command intrusion prevention system was subjected to several tests to verify that it could detect and prevent malicious AT command exploitation attempts on the baseband processor without degrading device performance. Specifically, the subscriber should not experience noticeable delays and should be able to use the device as intended (e.g., to make/receive phone calls and send/receive SMS texts and data). Several malicious AT commands were used in the tests, including some that target the X-Gold 608 and 618 baseband processors in older iPhones. These malicious AT commands were selected because of their documentation and use in iPhone baseband unlocks.

Another consideration was to appropriately tune the AT command signatures to minimize false positives and false negatives. Tuning requires the profiling of messages to establish baselines. Signatures that are too general increase the false positive rate and block valid AT commands. Signatures that are too specific increase the false negative rate and fail to block malicious AT commands. Tuning the signatures was determined to be as much art as a science;

a thorough treatment of AT command signature tuning is a topic for future work.

Malicious AT commands used in the tests were obtained from Tian et al. [30] and `theiphonewiki.com`. During the tests, regular cellular communications procedures were performed, including sending and receiving voice calls, SMS messages and streaming data while periodically injecting malicious AT commands.

The following tests were conducted:

- **Test 1:** Injection of a single malicious AT command during a two-minute voice call.
- **Test 2:** Injection of a malicious AT command every 100 ms during a two-minute voice call.
- **Test 3:** Injection of a single malicious AT command while the baseband was not being used (i.e., in the standby mode).
- **Test 4:** Injection of a malicious AT command every 100 ms while the baseband was not being used (i.e., in the standby mode).
- **Test 5:** Injection of a malicious AT command during a streaming data session.
- **Test 6:** Injection of a malicious AT command every 100 ms during a streaming data session.

Table 1 shows the test results along with the rules used to detect and block malicious AT commands. Five malicious or risky AT commands were used in the tests, all of which were successfully detected by the intrusion prevention system. The intrusion prevention functionality did not noticeably impact the normal use of the cellular modem (e.g., making and receiving calls, and sending and receiving SMS messages and packetized data). Additionally, no false positives were observed during tests. Future work will pursue a rigorous testing regimen that considers all the cellular functionality under real-world conditions.

8. Conclusions

Wireless communications coprocessors provide wide-area and personal-area networking capabilities to Internet of Things and mobile devices. These coprocessors have been the targets of exploitation research in recent years. In particular, the baseband processors, which are responsible for cellular communications, are attractive targets for adversaries interested in intercepting, modifying, interrupting or fabricating voice, text, data and signaling traffic.

This research has made key contributions to securing baseband processors from exploitation attempts by hostile applications that execute on the application processor. Retrofitting an AT command intrusion prevention system between the application processor and baseband processor mitigates the negative effects of malicious AT commands. Because the intrusion prevention

Table 1. Malicious AT command test cases.

AT Command	**IPS Rule**	**Detected**
at+xlog=1, ”jjjjjjjjjjjjjjjjjjjjjjjjjjjj44445555PPPP”	length +xlog >= 30	Yes
at+xapp=”aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaa4444555566667777PPPP”	length +xapp >= 50	Yes
at%IMEI=”490154203237518”	block at%imei=	Yes
at+fns=”00000000000000000000000000000000000000 0000000000000000000000111111222233334444555566 66677”	length +fns >=1024	Yes
at+stkprof=1,”064a541c044b1878222803d010700132 0133f8e720470000bf9f154000170100546e56402000000 05c130100266e5640ddddddddeeeeeeeeb890512000000 00010101010202020206113010000c000000”;”\x10\x32 \x0F\x27\xBA\x43 \x17\x1C\x0E\xA4\x0B\xA5 \x01\x35\x21\x78\x78\x29\x0C\xD0\xA8\x47 \x0B\x01\x61\x78\xA8\x47\xC0\x46 \xC0\x46 \xC0\x46\xC0\x46\xC9\x18\x11\x70 \x02\x34\x01 \x32\xEF\xE7\xC0\x46\xC0\x46 \x01\x37\x38 \x47\x30\x30\x41\x29\x01\xDA 09pG79pG02480 3A1013101601FBD00004C711140F0B51C4B8026 8BB03601188008911A4C301CA04700250990 ....CONTINUED................... 20xx”	match ”\x32\x0F\x27”	Yes

system is vendor agnostic and focuses on analyzing text-based AT commands, the approach is easily implemented to secure baseband processors produced by diverse manufacturers.

Future research will focus on integrating the AT command intrusion prevention technique in Internet of Things and mobile device platforms and operating systems. Additionally, techniques for retrofitting security mechanisms in other

vulnerable wireless chipsets will be explored. Finally, efforts will focus on employing cryptographically-sound techniques for firmware attestation to combat threats ranging from unauthorized surveillance to insidious system compromises.

References

[1] Aleph One, Smashing the stack for fun and profit, *Phrack*, vol. 7(49), 1996.

[2] N. Artenstein, Broadpwn: Remotely compromising Android and iOS via a bug in Broadcom's Wi-Fi chipsets, presented at *Black Hat USA*, 2017.

[3] G. Beniamini, Over the Air: Exploiting Broadcom's Wi-Fi Stack (Part 1), Project Zero Team, Google, Mountain View, California (`googleprojectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html`), April 4, 2017.

[4] G. Beniamini, Over the Air: Exploiting Broadcom's Wi-Fi Stack (Part 2), Project Zero Team, Google, Mountain View, California (`googleprojectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_11.html`), April 11, 2017.

[5] G. Beniamini, Over the Air – Vol.2, Pt. 3: Exploiting the Wi-Fi Stack on Apple Devices, Project Zero Team, Google, Mountain View, California (`googleprojectzero.blogspot.com/2017/10/over-air-vol-2-pt-3-exploiting-wi-fi.html`), October 11, 2017.

[6] A. Blanco and M. Eissler, One firmware to monitor 'em all, presented at the *Ekoparty Security Conference*, 2012.

[7] G. Delugre, Reverse engineering a Qualcomm baseband, presented at the *Twenty-Eighth Chaos Communication Congress*, 2011.

[8] J. Drake, P. Fora, Z. Lanier, C. Mulliner, S. Ridley and G. Wicherski, *Android Hacker's Handbook*, John Wiley and Sons, Indianapolis, Indiana, 2014.

[9] European Telecommunications Standards Institute, Digital Cellular Telecommunications System (Phase 2+), AT Command Set for GSM Mobile Equipment (ME), GSM 07.07, Version 5.5.5, TS/SMG-040707Q, Sophia Antipolis, France, 1996.

[10] N. Golde and D. Komaromy, Breaking band: Reverse engineering and exploiting the Shannon baseband, presented at *REcon*, 2016.

[11] History of Computers, The modem of Dennis Hayes and Dale Heatherington (`history-computer.com/ModernComputer/Basis/modem.html`), 2016.

[12] B. Hond, Fuzzing the GSM Protocol, Master's Thesis, Computing Science Program, Radboud University, Nijmegen, The Netherlands, 2011.

[13] iPhone Dev Team, `ultrasn0w`, *The iPhone Wiki* (`www.theiphonewiki.com/wiki/Ultrasn0w`), 2009.

[14] iPhone Dev Team, `Purplesn0w`, *The iPhone Wiki* (`www.theiphonewiki.com/wiki/Purplesn0w`), 2015.

[15] P. Kocialkowski, Samsung Galaxy Back-Door (`redmine.replicant.us/projects/replicant/wiki/SamsungGalaxyBackdoor`), February 4, 2014.

[16] B. Krebs, Mirai botnet authors avoid jail time, *Krebs on Security* (`krebsonsecurity.com/tag/mirai-botnet`), September 19, 2018.

[17] A. Lonzetta, P. Cope, J. Campbell, B. Mohd and T. Hayajneh, Security vulnerabilities in Bluetooth technology as used in IoT, *Journal of Sensor and Actuator Networks*, vol. 7(3), article no. 28, 2018.

[18] L. Miras, The baseband playground, presented at the *Ekoparty Security Conference*, 2011.

[19] M. Moe, Go ahead, hackers. Break my heart, *Wired*, March 14, 2016.

[20] C. Mulliner, S. Liebergeld, M. Lange and J. Seifert, Taming Mr. Hayes: Mitigating signaling based attacks on smartphones, *Proceedings of the Forty-Second Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2012.

[21] J. Nivethan and M. Papa, A Linux-based firewall for the DNP3 protocol, *Proceedings of the IEEE Symposium on Technologies for Homeland Security*, 2016.

[22] M. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. Grieco, G. Boggia and M. Dohler, Standardized protocol stack for the Internet of (important) Things, *IEEE Communications Surveys and Tutorials*, vol. 15(3), pp. 1389–1406, 2013.

[23] F. Sanglard, Tracing the Baseband: Part 1 (`fabiensanglard.net/cellphoneModem/index.php`), May 11, 2010.

[24] F. Sanglard, Tracing the Baseband: Part 2 (`fabiensanglard.net/cellphoneModem/index2.php`), May 11, 2010.

[25] M. Sauter, *From GSM to LTE: An Introduction to Mobile Networks and Mobile Broadband*, John Wiley and Sons, Chichester, United Kingdom, 2014.

[26] B. Seri and A. Livne, Exploiting BlueBorne in Linux-based IoT devices, Armis, Palo Alto, California, 2019.

[27] W. Shaw, *Cybersecuriy for SCADA Systems*, PennWell, Tulsa, Oklahoma, 2006.

[28] SIMCom Wireless Solutions, AT Commands Set, SIM900_ATC_V1.00, Shanghai, China, 2010.

[29] Statista, Internet of Things (IoT) connected devices installed based worldwide from 2015 to 2025 (in billions), Frankfurt, Germany (`www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide`), 2018.

[30] D. Tian, G. Hernandez, J. Choi, V. Frost, C. Ruales, P. Traynor, H. Vijayakumar, L. Harrison, M. Grace and K. Butler, ATtention spanned: Comprehensive vulnerability analysis of AT commands within the Android ecosystem, *Proceedings of the Twenty-Seventh USENIX Security Symposium*, pp. 273–290, 2018.

[31] Tofino Security, Tofino Firewall LSM, Lantzville, Canada (`www.tofinosecurity.com/products/Tofino-Firewall-LSM`), 2017.

[32] P. Tsang and S. Smith, YASIR: A low-latency, high-integrity security retrofit for legacy SCADA systems, *Proceedings of the Twenty-Third IFIP TC 11 International Information Security Conference*, pp. 445–459, 2008.

[33] R. Weinmann, All your baseband are belong to us, presented at the *Hack.lu Conference*, 2010.

[34] R. Weinmann, Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks, *Proceedings of the Sixth USENIX Conference on Offensive Technologies*, 2012.

[35] H. Welte, Anatomy of Contemporary GSM Cellphone Hardware (`ondoc.logand.com/d/373/pdf`), 2010.