FirmState: Bringing Cellular Protocol States to Shannon Baseband Emulation

Suhwan Jeong
shjeong.b@enki.co.kr
ENKI WhiteHat
Seoul, Republic of Korea

Beomseok Oh
beomseoko@kaist.ac.kr
KAIST
Daejeon, Republic of Korea

Kwangmin Kim
kwangmin.kim@kaist.ac.kr
KAIST
Daejeon, Republic of Korea

Insu Yun
insuyun@kaist.ac.kr
KAIST
Daejeon, Republic of Korea

Yongdae Kim
yongdaek@kaist.ac.kr
KAIST
Daejeon, Republic of Korea

CheolJun Park
cheoljunp@khu.ac.kr
Kyung Hee University
Yongin, Republic of Korea

Abstract

Cellular baseband processors represent critical security components in modern mobile devices, yet they remain challenging to analyze due to their complexity and restricted access. While the FirmWire enables full-system baseband emulation, it lacks protocol state awareness, limiting its coverage and fidelity. While implementing such support demands substantial engineering effort, accurately modeling protocol states remains essential for comprehensive baseband security analysis. In this paper, we present FirmState, a state-aware methodology that augments baseband emulation, specifically targeting Samsung Shannon baseband. FirmState semi-automatically recovers and applies state information extracted from physical devices during actual network communication, enabling more complete code coverage and authentic behavior reproduction without extensive reverse engineering. Our evaluation demonstrates a significant improvement in code coverage, achieving 7.5% for RRC–2.7× higher than previous work. Additionally, our system newly supports NAS over FirmWire, with code coverage ranging from 4.5% to 9.2%, depending on the protocol state. Using our approach, we discovered and analyzed two 1-day vulnerabilities in Samsung's baseband implementation, demonstrating FirmState's effectiveness for baseband security. We make FirmState open-source to support further research in baseband security.

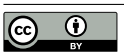
CCS Concepts

• **Security and privacy → Mobile and wireless security**.

Keywords

Baseband; Security; Cellular; Emulation

ACM Reference Format:
Suhwan Jeong, Beomseok Oh, Kwangmin Kim, Insu Yun, Yongdae Kim, and CheolJun Park. 2025. FirmState: Bringing Cellular Protocol States to Shannon Baseband Emulation. In *18th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec 2025), June 30-July 3, 2025, Arlington, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3734477.3734726



WiSec 2025, Arlington, VA, USA

ACM ISBN 979-8-4007-1530-3/2025/06
https://doi.org/10.1145/3734477.3734726

1 Introduction

A baseband processor presents one of the most critical attack surfaces in modern mobile devices. It manages radio communication between cellular devices and networks, handling a complex protocol stack that spans multiple generations of wireless standards. As a result, a baseband processor can contain various vulnerabilities. Given the widespread use of mobile devices and the wireless accessibility of this interface, such vulnerabilities can lead to significant security threats.

Consequently, researchers have employed various methods to test basebands. Primarily, they have relied on reverse engineering to assess baseband security [1, 4, 5, 13]. Alternative approaches, such as BASESPEC [12] and BASECOMP [11], leverage protocol specifications for comparative analysis with firmware, reducing reverse-engineering effort. Also, numerous black-box testing methods have successfully uncovered security flaws [3, 10, 15] However, these approaches face limitations: reverse engineering demands expert knowledge and manual effort, while black-box testing lacks internal visibility, making root cause analysis difficult and hindering fine-grained vulnerability assessment.

Recently, baseband emulation has emerged as a promising solution to overcome these obstacles. By replicating the real baseband operation in controlled environments, researchers can apply existing techniques for dynamic security testing, runtime introspection, and firmware debugging to identify vulnerabilities. Notable efforts in this direction include BaseSAFE [14], which implements protocol-specific emulation for MediaTek baseband with targeted coverage of RRC and NAS layers. More recently, FirmWire [9] advanced the state-of-the-art by enabling full-system emulation that executes unmodified baseband firmware. It supports comprehensive features for emulating the entire booting process of Samsung Shannon and MediaTek baseband. Additionally, its debugging capabilities enable function tracing, event logging, and memory analysis during execution, providing the foundation for automated fuzzing and vulnerability discovery in these complex systems.

While these approaches have significantly advanced baseband emulation research, both have limitations, as shown in Table 1. BaseSAFE is limited to a specific baseband model (MT6795) and does not support full-system emulation. Instead, it focuses on protocol-specific fuzzing, limiting its execution context. As a result, it hinders root cause analysis, harnesses development, and deeper insights into internal operations. FirmWire takes this further with

full-system emulation, allowing researchers to inspect execution flow, memory behavior, and internal operations. However, it lacks supporting network interaction; it cannot establish or simulate communication with cellular networks. Essentially, it emulates a phone that has just booted up but has not yet connected to a cellular network. To mitigate this, it provides ModKit, which enables injecting harnesses for testing specific protocol layers. However, FirmWire still struggles with state-aware emulation, as its harnesses– including the LTE RRC layer harness–cannot consider protocol state transitions, limiting the fidelity of complex protocol interactions.

In this paper, we present FirmState, a state-aware methodology that augments baseband emulation by integrating state information of cellular protocols, which specially targets Samsung Shannon baseband. It leverages runtime information from physical devices to (semi-)automatically recover and apply these states, enabling more complete code coverage and authentic behavior reproduction without extensive reverse engineering. We implement our methodology on the state-of-the-art baseband emulator, FirmWire, focusing on two critical protocol layers (*i.e.* RRC and NAS). Our evaluation demonstrates significant improvements in code coverage, achieving 7.5% coverage of the RRC, which is 2.7x higher than FirmWire. Additionally, our system introduces the first to support NAS layer emulation in this environment, achieving between 4.5% to 9.2% coverage across different states. Additionally, state information obtained from real phones through a one-time effort can be adapted to different firmware versions or device generations with minimal reverse engineering using FirmState.

Scope. FirmState applies broadly to Shannon baseband, but we scope our implementation to LTE RRC and NAS layers where state handling is critical. These can provide an effective demonstration of our design by addressing the most security-relevant protocols.

In summary, our contributions are as follows:

- We present a novel methodology for baseband state-aware emulation, which enhances state-of-the-art emulation techniques by integrating protocol state information. We release FirmState as open-source for future research as https://github.com/1nteger-c/FirmState.
- Our approach achieves 7.5% code coverage for RRC layer emulation– 2.7x higher than FirmWire–and expands emulation capabilities to include NAS, enabling the discovery of two 1-day NAS vulnerabilities as well as deeper security analysis.
- We demonstrate the adaptability of one state information on different firmware version or device generation.

2 Background

2.1 Cellular Baseband

Every modern smartphone is equipped with multiple processors, including an application processor (AP) and a baseband processor (BP). The baseband processor is responsible for cellular communication and typically runs as a real-time operating system (RTOS). The main operations of the baseband processor is to scan for available cellular networks, decode network information messages, and authenticate with the appropriate network. To achieve this, the baseband continuously exchanges messages with the cellular network, operating across multiple protocol layers to handle signaling and data transmission.

Approach	Full-Emulation	Target BP	Target Layer	State-Aware
BaseSAFE [14]	✗	M	RRC, NAS	✗
FirmWire [9]	✓	S, M	RRC	✗
Our work	✓	S	RRC, NAS	✓

M: MediaTek, S: Samsung | ✓: Supported, ✗: Not Supported

Table 1: Comparison with existing LTE baseband emulation approaches

The cellular network follows a layered architecture, which is divided into two primary domains: user-plane and control-plane. The user-plane is responsible for transmitting user data, while the control-plane manages network signaling and session control. One of the key protocols in the control-plane is Radio Resource Control (RRC), which manages radio resources and connection establishment between the User Equipment (UE) and the base station. Another critical protocol is Non-Access Stratum (NAS), which functions between the UE and the Evolved Packet Core (EPC), handling authentication, security, and session management. Notably, the NAS protocol incorporates the Authentication and Key Agreement (AKA) procedure, which is fundamental to cellular security.

2.2 FirmWire

FirmWire [9] is the state-of-the-art baseband emulator that enables full-system emulation of unmodified firmware binaries. Built upon QEMU, it supports the complete booting process and offers several features for security research, such as a snapshot functionality for improving the efficiency of fuzzing tasks and memory debugging tool integration. Additionally, FirmWire provides function hooking capabilities, allowing researchers to intercept and analyze system functions. For instance, it hooks the logging function to redirect debug messages to standard output, offering deeper visibility into the system's internal operations.

While FirmWire emulates essential hardware peripherals to mimic real device behavior, it does not support direct interaction with a cellular network. To address this limitation, it provides ModKit, which enables researchers to inject protocol-specific harnesses for handling messages at various protocol layers. Due to this reason, although FirmWire has successfully demonstrated emulation of the RRC layer, it lacks state-aware capabilities—its harnesses operate without considering protocol state transitions. This limitation reduces the fidelity of complex protocol interactions, making it less suitable for analyzing state-dependent security vulnerabilities.

3 FirmState

To overcome the limitations of existing approaches discussed in subsection 2.2, we introduce FirmState, a state-aware methodology that addresses the critical challenges in baseband emulation. Our approach significantly enhances Samsung Shannon baseband emulation by (semi-)automatically recovering and applying state information extracted from physical devices during actual network communication. This methodology eliminates complex reverse engineering by directly capturing execution traces and memory dumps, enabling more complete code coverage and authentic behavior reproduction. In this chapter, we identify the specific challenges that arise when implementing state-aware baseband emulation, present our technical approaches to overcome these obstacles, and detail the integrated architecture of FirmState.

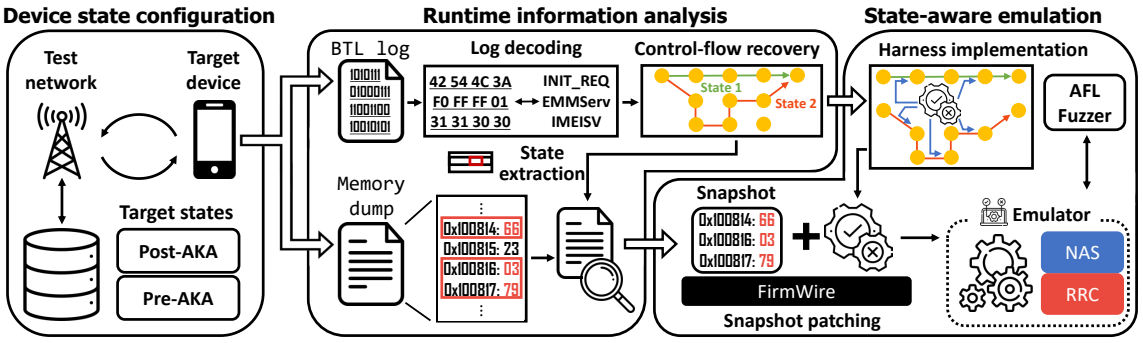


Figure 1: Overview of our design

3.1 Challenges

[C1] Complex State Management. Baseband manages numerous protocol states that drastically alter its behavior during cellular network communication. During communication, baseband manages protocol states that control different behaviors depending on the current state. As these states highly affect the behavior of baseband, effective emulation requires accurate understanding and handling of these states. Without them, emulation attempts like previous works [9, 14] will have significantly lower coverage compared to those that incorporate state awareness. However, achieving this state awareness remains a significant challenge due to two reasons. First, state variables from already complex specifications are embedded within intricate firmware at the memory level, making it difficult for researchers to identify and analyze in actual implementations. Second, setting state variables with values valid for actual network interactions is even harder. These characteristics create significant obstacles for emulation efforts, highlighting why addressing state complexity is a cornerstone challenge in this domain.

[C2] Control Flow Visibility. Even with proper state management, emulation efforts are hampered by limited visibility into the actual execution paths during network communication. While FirmWire [9] provides execution logs, these primarily reveal baseband behavior in isolation, without the context of actual network interaction. This creates a circular dependency: to understand network-related execution flows, we need working emulation, yet building accurate emulation requires understanding those flows. This visibility problem introduces several critical challenges. First, it complicates harness implementation, as researchers cannot easily determine correct data structures or necessary constants. Second, without runtime information during network communication, no reliable ground truth exists for validating emulation correctness, leaving researchers uncertain whether their implementations authentically replicate baseband behavior. This limitation forces researchers to make educated guesses about operational behaviors, significantly increasing the complexity of reverse engineering.

3.2 Our Approach

[A1] Runtime State Extraction. We leverage runtime state extraction from running devices, eliminating the need for time-consuming static analysis for each firmware variant. For the state extraction, we utilize memory dumps of Shannon baseband generated through a controlled crash mechanism accessible via hidden menus. By first placing the device in a specific desired state (e.g., pre-AKA and post-AKA) using a controlled test network environment and then performing extraction, we can directly obtain memory-level representations of those target states. As these memory dumps contain the specific state information, we can use them in state recovery with appropriate post-processing procedures. In particular, our analysis reveals that Shannon baseband manages protocol states using specific C structure types, allowing us to target corresponding memory regions for the extraction. However, identifying the exact memory regions that store essential state information for emulation presents a new challenge. We resolve it with our control flow recovery approach described in [A2].

[A2] Control Flow Recovery. We leverage and extend the Back Trace Log (BTL) mechanism in Shannon baseband to recover the control flow of runtime baseband during network communication. BTL is a proprietary debugging feature of Shannon baseband that records execution traces with timestamps and parameters, providing visibility into firmware runtime behavior. While previously used mainly for crash analysis [7], our investigation reveals BTL contains comprehensive information about runtime execution paths during network interaction, enabling recovery of previously unavailable baseband control flow. For example, by analyzing BTL captured after a device receives an RRC Reconfiguration Message, we can trace the exact control flow triggered by this message, identify accessed data structures, and understand detailed data processing. These retrieved control flows help generate appropriate harnesses, validate emulation results, and identify state variables' memory locations, supporting our state extraction approach described in [A1]. This detailed understanding of runtime behavior overcomes the circular dependency described in [C2] and simplifies the reverse engineering that would otherwise require extensive static analysis.

3.3 Overview

As illustrated in Figure 1, FirmState consists of three integrated phases: device state configuration, runtime information analysis, and state-aware emulation.

Device State Configuration. Our design fundamentally relies on extracting meaningful information from the device to facilitate baseband emulation with minimal effort. This approach begins with a properly configured testbed which can manipulate the target device into desired states. The testbed allows us to create controlled network conditions and trigger specific state transitions within the baseband (particularly focusing on pre-AKA and post-AKA

states) that enables systematic observation of the modem's behavior. By precisely configuring the device to reach these target states that mirror real conditions, we establish reference points for our design. This state-aware approach prioritizes capturing essential information that directly influences baseband functionality, reducing complexity without requiring comprehensive modeling of the entire system.

Runtime Information Analysis. Our design extracts and correlates two complementary data sources once the device reaches the target state. First, we leverage BTL file to understand control flow, as these execution logs contain crucial path data revealing how baseband operates in specific states. This approach eliminates the need to analyze the entire firmware with its numerous functions and complex paths. We developed an adaptive framework that automatically identifies BTL format versions, ensuring compatibility across different Shannon baseband generations. This auto-detection ensures our methodology remains effective across various Shannon baseband without requiring manual configuration.

The second component involves extracting targeted memory segments containing state information from the overall memory dump. Our design uses decoded BTL to semi-automatically locate state-related structures in memory dumps, rather than analyzing the complete memory. This targeted extraction significantly reduces the complexity of state recovery while ensuring only relevant information is captured for emulation.

State-Aware Emulation. To achieve state-aware baseband emulation, we must apply retrieved state memory within a properly designed harness capable of handling target protocol messages. For harness generation, we leverage analyzed control flow information. Shannon baseband is structured according to protocol layers (Tasks), with functions managed through a msgGroup variable. Effective harness generation requires identifying the exact Task and msgGroup to inject messages at the correct position and understanding the C structure of these messages. The BTL logs and reconstructed control flow provide this information, enabling us to implement a basic harness that can process target messages.

However, this basic harness may not execute properly without proper state configuration. To address this challenge, our design incorporates a snapshot-patching procedure. While the original FirmWire [9] includes functionality to capture snapshots after booting to reduce execution time, our design extends this capability to load specific memory segments (such as state information) and patch specific functions to manage or force particular control flows. For example, we can bypass MAC validation functions that would otherwise unnecessarily complicate emulation. This patching procedure offers versatility beyond state configuration and can be applied in various ways to streamline emulation.

By combining the harness with properly configured states, we establish the foundation for target emulation. Nevertheless, discrepancies may still exist between device logs and emulator output. To address these differences, our design incorporates a progressive refinement process through an error correction feedback loop. By identifying differences between device and emulator logs, we can pinpoint inaccuracies in our emulation. This iterative process allows us to refine our approach by revisiting previous stages as needed, ultimately achieving successful baseband emulation that accurately reflects real device behavior.

Phone Model	BP Version	Release Date	BTL Version	[7]	FirmState
Galaxy Note8	N950NKOU5DSL1	2020.01.09.	1100	●	●
Galaxy S9	G960NKOU2CSI1	2019.10.02.	1100	●	●
Galaxy S9+	G965FXXSHFUJ2	2021.10.19.	1100	●	●
Galaxy Note9	N960FXXU4ASJ2	2021.05.08.	1100	●	●
Galaxy S10	G973FXXU9FUCD	2021.03.23.	1200	○	●
Galaxy S10	G973FXXUAFUE1	2021.05.07.	1200	○	●
Galaxy S10	G973NKOU7HVG2	2022.07.29.	1200	○	●
Galaxy S10	G973NKOU7HWD1	2023.04.18.	1200	○	●
Galaxy S10e	G970NKOU7HWD1	2023.04.18.	1200	○	●
Galaxy A30	A305NKOS5CVF1	2022.06.22.	1200	○	●
Galaxy Note10 5G	N971NKOU2HWH3	2023.08.16.	1200	○	●
Galaxy S21	G991NKOU4EWE2	2023.06.26.	1300	○	●
Galaxy S24	S921NKSU2AXE4	2024.06.10.	1410	○	●

Table 2: BTL Versions of commercial Samsung phones

4 Implementation

Our design builds upon FirmWire [9], providing protocol-level emulation for Shannon baseband focusing on LTE RRC and NAS layers. We selected these protocols for their critical role in device control and security, and their complex state-dependent nature demonstrates our methodology's effectiveness. Our implementation required two foundational components: a snapshot-patching procedure and BTL decoding extensions.

Snapshot-Patching Procedure. We developed a GDB-based approach that extends FirmWire's snapshot capabilities with custom patching scripts. By setting strategic breakpoints before snapshot capture and executing specific GDB commands, we precisely modify state-related memory structures and function behaviors. This produces patched snapshots that accurately configure states, eliminating the need for manual intervention and enabling precise control over the baseband's internal state.

BTL Decoder Extension. We extended BTL decoding capabilities to extract runtime information from baseband. After collecting BTL files from 13 different devices with varying versions (shown in Table 2), we identified four distinct BTL versions. We analyzed their internal structure, identifying key differences in header format, field alignment, and element offsets while preserving the fundamental structure. We refactored the available scripts [7], which supported a single version, and implemented an adaptive parsing system that automatically detects the BTL version and adjusts accordingly. Our decoder successfully interprets all four BTL versions, providing unified access to runtime information across device generations.

RRC Layer Emulation. We generate the harness by referencing control flow from BTL file. During state extraction, we observe the memory range of the RRC state from memory dumps, comparing the states between the basic harness and the BTL. After incorporating state information with our snapshot-patching procedure, we achieve successful RRC emulation with proper state handling, validated by comparing device logs with emulator logs under identical conditions. Our implementation's effectiveness is demonstrated through code coverage comparisons in subsection 5.1.

NAS Layer Emulation. For NAS layer emulation, we analyze decoded BTL to identify key C structures containing NAS states from text identifiers like RegStatus or SAEMM_REGI_INIT. We extract relevant memory regions and apply snapshot-patching for pre-AKA state emulation. Post-AKA state emulation presented additional challenges with Message Authentication Code (MAC) checking functions impeding control flow, which we resolved by bypassing the MAC validation function identified through emulator logs. Using debugging tools, we confirmed that various NAS messages are properly processed by appropriate handler functions.

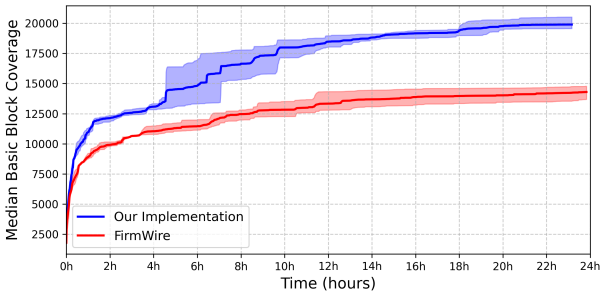


Figure 2: Comparison of RRC fuzzing code coverage between our implementation and original Firmwire

5 Evaluation

We evaluate our approach from three perspectives: (1) fuzzing performance, comparing code coverage with FirmWire and presenting two 1-day vulnerabilities verified through over-the-air testing, (2) root cause analysis of these vulnerabilities, and (3) adaptability across different firmware versions and device models.

5.1 Fuzzer Performance

Setup. We conducted our evaluations using a server equipped with an Intel i7-14700K processor (28 cores) and 32GB of memory running Ubuntu 22.04. To ensure fair comparison, we tested both our implementation and FirmWire using identical firmware (Galaxy S10: G973FXXUAFUE3) and the same fuzzing engine, AFL++[2]. All coverage measurements were performed over a consistent 24-hour fuzzing period and repeated across *three* independent runs to ensure statistical validity. As FirmWire used Ghidra-based scripts for tracking basic block coverage, we adapted them to equivalent IDA scripts to ensure consistency in our measurement methodology.[1]

Measurement Results. Our measurement results demonstrate the effectiveness of our state-aware emulation approach. As shown in Figure 2, our implementation consistently achieves higher code coverage than FirmWire across the 24-hour evaluation period, with the shaded areas representing the range from three independent runs and solid lines indicating median coverage. We also measure basic blocks specific to the target protocol layers. Our approach achieves 7.5% code coverage on the RRC layer (2.7× higher than FirmWire's 2.8%) and between 4.5% to 9.2% on the NAS layer for pre-AKA and post-AKA states respectively. The increase in NAS coverage from pre-AKA to post-AKA states clearly demonstrates that proper state handling significantly impacts coverage, confirming both the importance of our state-aware approach and its effectiveness. These results clearly show that FirmWire's lack of state consideration limits thorough code coverage in baseband testing.

Fuzzing Results. The fuzzing campaigns yield several candidate findings, which we then verify by replaying each candidate payload on physical devices using our state-setting environment. This verification confirms two crashes: one in the pre-AKA state and another in the post-AKA state. We also analysis the root cause of each vulnerability in subsection 5.2.

[1]For details on the measurement methodology, please refer to the Evaluation section of FirmWire.

Implementation	Layer	Covered / Total	coverage (%)
FirmWire	RRC	2,447 / 87,371	2.8%
FirmState	RRC	6,572 / 87,371	7.5%
FirmState (pre-AKA)	NAS	1,320 / 29,128	4.5%
FirmState (post-AKA)	NAS	2,739 / 29,128	9.2%

Table 3: Code Coverage Results

5.2 Root Cause Analysis

One key advantage of baseband emulation is the ability to perform detailed root-cause analysis. We employed QEMU's executed block retrieval functionality and GDB for dynamic analysis to identify precise vulnerability mechanisms.

pre-AKA vulnerability. The pre-AKA vulnerability manifests in a buffer copying mechanism during message parsing. The implementation uses an unsigned integer to track remaining data length while copying fixed 0x20-byte chunks. When processing data whose length is not a multiple of 0x20, the final subtraction operation causes integer underflow, which results in an infinite copying loop that ultimately crashes the modem.

post-AKA vulnerability. The post-AKA vulnerability exists in the Emergency Number List parsing routine, as shown in Listing 1. This parser implements a hierarchical structure with a top-level length field followed by multiple entries. The vulnerability stems from using a BYTE instruction for incrementing the index pointer during parsing. With a crafted length value of 0xFF, the increment operation causes integer overflow to 0x00, forcing the parser into an infinite loop as it repeatedly processes the same memory region.

Listing 1: Infinity Loop in decoding EmergencyNumberList

```
while (idx < length){
    EmergencyNumberStruct = &data[idx]
    ...
    idx += data[idx] + 1;       // [BUG] UXTB instruction!!
    memset(outBuf, 0xFF, 22);
    EmergencyNumberLen = *EmergencyNumberStruct;
    for ( i = 0; EmergencyNumberLen - 1 > i; i = (i + 1) ){
        // outBuf <- parse(EmergencyNumberStruct)
        EmergencyListParse(outBuf);
    }
}
```

5.3 Adaptability

One question regarding our methodology is whether FirmState need the exact device with the target firmware for supporting state-aware emulation. To investigate this concern, we evaluated our approach's adaptability across different devices and firmware versions. Our tests successfully demonstrated NAS emulation on recent Galaxy S10 and Galaxy S9 versions using state information extracted from an older (2021) Galaxy S10 firmware.

This cross-device implementation required only two adjustments. First, updating firmware-specific constants in the harness (e.g., the identifier for "LTE_PDCP_DATA_IND" differs between S9 and S10 implementations). FirmState automatically identifies these differences and converts them to appropriate values. Second, mapping memory layouts across versions, which proved straightforward since state-related structures remain largely consistent due to standardized protocol specifications. For NAS layer specifically, registration and authentication structures maintained uniform definitions across implementations, with FirmState automatically detect and

adjust the required base address offsets. These findings demonstrate that FirmState transfers effectively across different device types and firmware versions with minimal effort, making baseband security analysis accessible without deep baseband expertise.

6 Discussion

Self-evolving State Exploration. While effective for predetermined states, FirmState struggles with short-duration state. As further work, creating a feedback mechanism within FirmState could be a promising approach. During fuzzing, when the emulator transitions to a new state, it could capture memory dump and BTL from this state. This data might then serve as input for FirmState's analysis pipeline, enabling emulation of newly discovered states without additional device state configuration. This approach would extend FirmState's capabilities, potentially revealing previously unidentified execution paths and security vulnerabilities.

Broader Applications of Methodology. While our work focused on Shannon baseband emulation, the core methodology we propose has broader applications beyond this specific context. The techniques can enhance static analysis of baseband, which traditionally suffers from limited visibility into state-dependent behaviors and complex execution paths. Our BTL decoding and state recovery techniques provide context that improves the precision of static analysis by helping them better understand path constraints and reduce false positives when identifying vulnerabilities. Furthermore, our approach can be adapted to other platforms such as MediaTek baseband with minimal modification, provided that memory dumps and execution logs can be acquired. The key requirement across any implementation is simply access to runtime state information.

7 Related Work

Reverse Engineering. Previous research [1, 5, 13] have utilized reverse engineering to analyze baseband, requiring manual effort. Recent research [11, 12] has aimed to this effort by leveraging protocol specifications. BASESPEC [12] introduces a comparative analysis that examines message structures implementation by leveraging cellular specifications, successfully uncovering multiple functional errors and memory-related vulnerabilities. BASECOMP [11] extends this effort by introducing a probabilistic inference model to identify integrity protection mechanisms, significantly reducing manual analysis efforts while discovering severe security flaws. Unlike these works, FirmState reduces reverse engineering effort by recovering control flow with the extracted runtime information.

Emulation-based Analysis. Grassi *et al.* [6] showed how MediaTek's baseband could be reverse engineered and emulated to discover memory vulnerabilities. Natalie Silvanovich [16] demonstrated an emulation-based approach to analyzing and exploiting vulnerabilities in Samsung baseband. However, both efforts were industry-driven, and their implementations were not released as open-source. In contrast, two major works have been open-sourced in academia, significantly advancing baseband emulation research. Maier *et al.* [14] emulate the RRC and NAS layers of a specific MediaTek firmware and find memory corruptions through fuzzing. Hernandez *et al.* [8] took this further by developing a QEMU-based emulation framework for Samsung and MediaTek baseband, achieving full-system emulation that supports broader firmware analysis. FirmState enhances state-of-the-art baseband emulator to enable state-aware protocol emulation, improving fidelity and bridging the gap between emulation and real-world baseband behavior.

8 Conclusion

Baseband security analysis is challenging due to closed-source firmware and state-dependent behavior. While the state-of-the-art baseband emulator, FirmWire, supports executing unmodified baseband firmware and provides debugging capabilities, its lack of network interaction prevents protocol state awareness, limiting coverage and fidelity. In this paper, we present FirmState, a state-aware methodology that enhances baseband emulation by recovering and applying state information from a real device, allowing emulator to more accurately reproduce stateful protocol behavior. We show that FirmState improves code coverage and enables NAS layer emulation, which was previously unsupported. We discovered two one-day vulnerabilities on NAS and demonstrated that FirmState enables detailed root cause analysis. Furthermore, we showed that state information extracted from a real device can be adapted to different firmware versions and device models, demonstrating its applicability beyond a single target.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2024-00437252, Development of anti-sniffing technology in mobile communication and AirGap environments).

References

[1] Amat Cama. 2018. A walk with Shannon. *OPCDE* (2018).
[2] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. 2020. AFL++: Combining Incremental Steps of Fuzzing Research. In *USENIX WOOT*.
[3] Matheus E Garbelini, Zewen Shang, Shijie Luo, and Sudipta Chattopadhyay. 2023. *5GHOUL: Unleashing Chaos on 5G Edge Devices.* Technical Report. SUTD.
[4] Nico Golde. 2018. There's Life in the Old Dog Yet: Tearing New Holes into Intel/iPhone Cellular Modems. *Comsecuris* (2018).
[5] Nico Golde and Daniel Komaromy. 2016. Breaking Band: reverse engineering and exploiting the shannon baseband. *REcon* (2016).
[6] Marco Grassi and Xingyu Chen. 2020. Exploring the MediaTek Baseband. *OffensiveCon* (2020).
[7] Grant Hernandez. 2021. ShannonBaseband: Samsung Shannon Baseband Research. GitHub repository. https://github.com/grant-h/ShannonBaseband
[8] Grant Hernandez and Marius Muench. 2020. Emulating Samsung's Baseband for Security Testing. *BlackHat USA* (2020).
[9] Grant Hernandez, Marius Muench, Dominik Maier, Alyssa Milburn, Shinjo Park, Tobias Scharnowski, Tyler Tucker, Patrick Traynor, and Kevin Butler. 2022. FIRMWIRE: Transparent dynamic analysis for cellular baseband firmware. In *NDSS*.
[10] Syed Rafiul Hussain, Imtiaz Karim, Abdullah Al Ishtiaq, Omar Chowdhury, and Elisa Bertino. 2021. Noncompliance as deviant behavior: An automated black-box noncompliance checker for 4g lte cellular devices. In *CCS*.
[11] Eunsoo Kim, Min Woo Baek, CheolJun Park, Dongkwan Kim, Yongdae Kim, and Insu Yun. 2023. BASECOMP: A Comparative Analysis for Integrity Protection in Cellular Baseband Software. In *USENIX Security*.
[12] Eunsoo Kim, Dongkwan Kim, CheolJun Park, Insu Yun, and Yongdae Kim. 2021. BaseSpec: Comparative Analysis of Baseband Software and Cellular Specifications for L3 Protocols.. In *NDSS*.
[13] Daniel Komaromy. 2023. Basebanheimer: Now I Am Become Death, The Destroyer Of Chains. *OffensiveCon* (2023).
[14] Dominik Maier, Lukas Seidel, and Shinjo Park. 2020. BaseSAFE: Baseband sanitized fuzzing through emulation. In *ACM WiSec*.
[15] C Park, Sangwook Bae, B Oh, Jiho Lee, Eunkyu Lee, Insu Yun, and Yongdae Kim. 2022. DoLTEst: In-depth Downlink Negative Testing Framework for LTE Devices. In *USENIX Security*.
[16] Natalie Silvanovich. 2023. How to Hack Shannon Baseband (from a Phone). *OffensiveCon* (2023).