Wisec '25

FirmState: Bringing Cellular Protocol States to Shannon Baseband Emulation

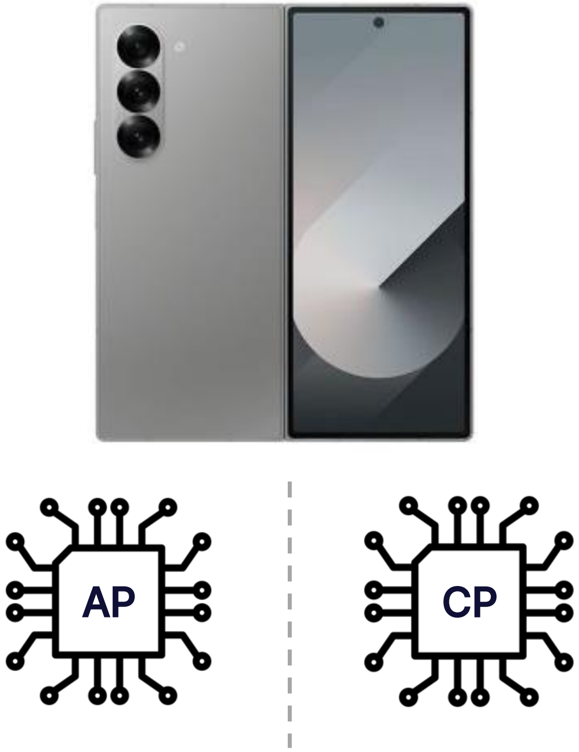
Suhwan Jeong, Beomseok Oh, Kwangmin Kim, Insu Yun, Yongdae Kim, CheolJun Park

ENKI WHITEHAT SysSecLab

Cellular Baseband

- Modern smartphones contain multiple specialized processors
    - Application Processor (AP) / Communication Processor (CP)
    - CP is commonly referred to as "**Baseband**"

- Baseband
    - Handles cellular communication
    - Exploded in our lives



Security of Baseband

- Large Attack Surfaces
  - Diverse cellular stacks

- Implemented in Memory Unsafe Languages
  - C / C++

- Limited Security Mitigations
  - No PIE, No ASLR

- Closed source

Previous Research

❖ Static Analysis [Recon '16 / BlackHat USA '21 / OffensiveCon '23 / Usenix '23 / ...]

- Complex and time-consuming reverse engineering
- No any pre-processing

❖ Dynamic Analysis (OTA) [Usenix '11 / WiMob '21 / GLOBECOM '22 /...]

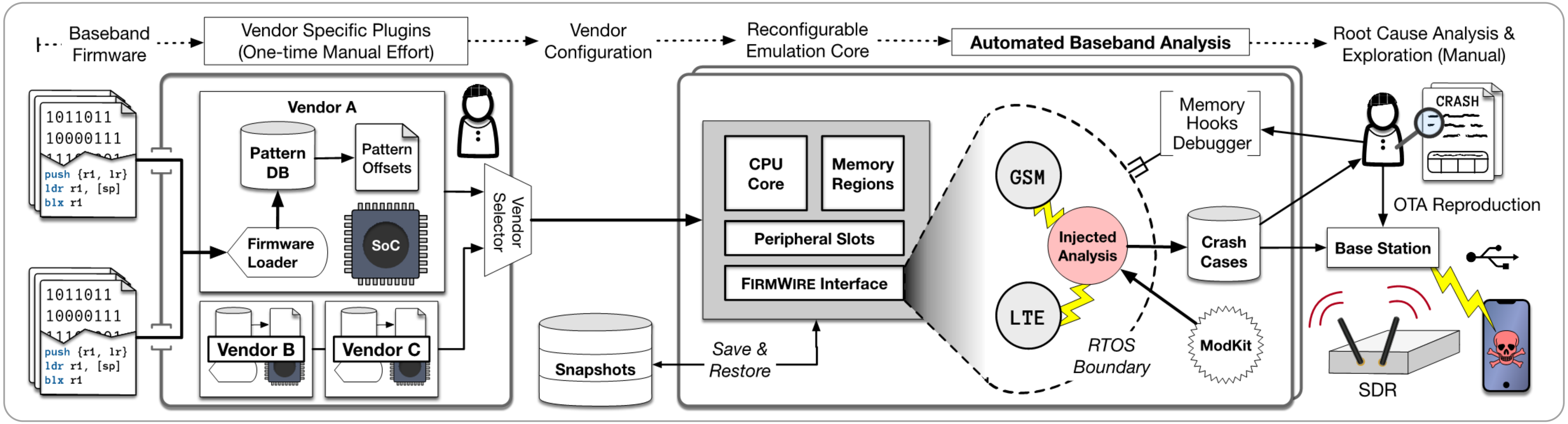
- No details about the crash
- Lightweight pre-processing, no false positive

❖ Dynamic Analysis (Emulation) [S&P 20 / OffensiveCon 20 / NDSS 22 / OffensiveCon 23 / S&P 24]

- Requires a diverse tasks for successful emulation
- Enables direct memory access

FirmWire

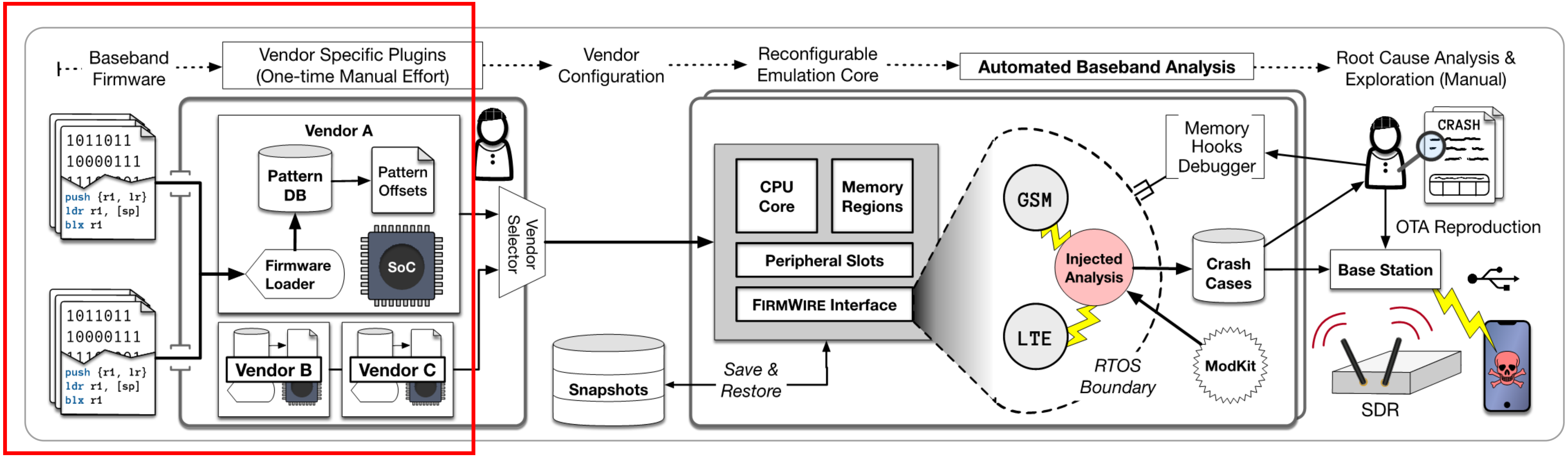
- ❖ State-of-the-art full-system baseband emulation platform



FirmWire

- ❖ State-of-the-art full-system baseband emulation platform

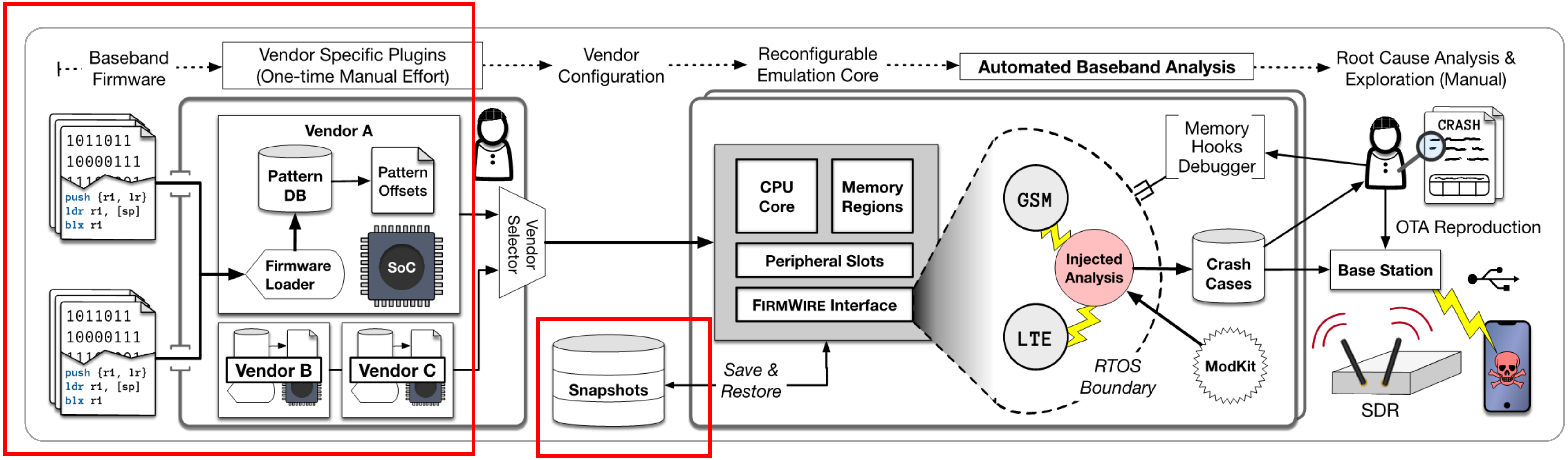
Samsung Shannon / MediaTek



FirmWire

- ❖ State-of-the-art full-system baseband emulation platform

Samsung Shannon / MediaTek



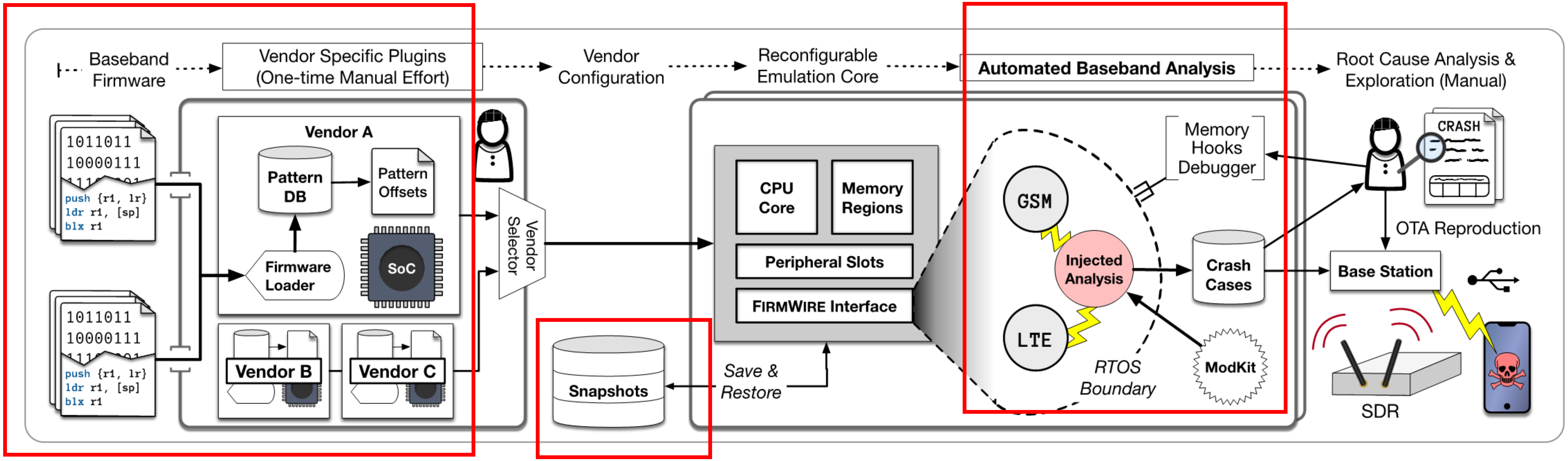
Memory snapshot

FirmWire

- State-of-the-art full-system baseband emulation platform

Samsung Shannon / MediaTek

Custom code injection



Memory snapshot

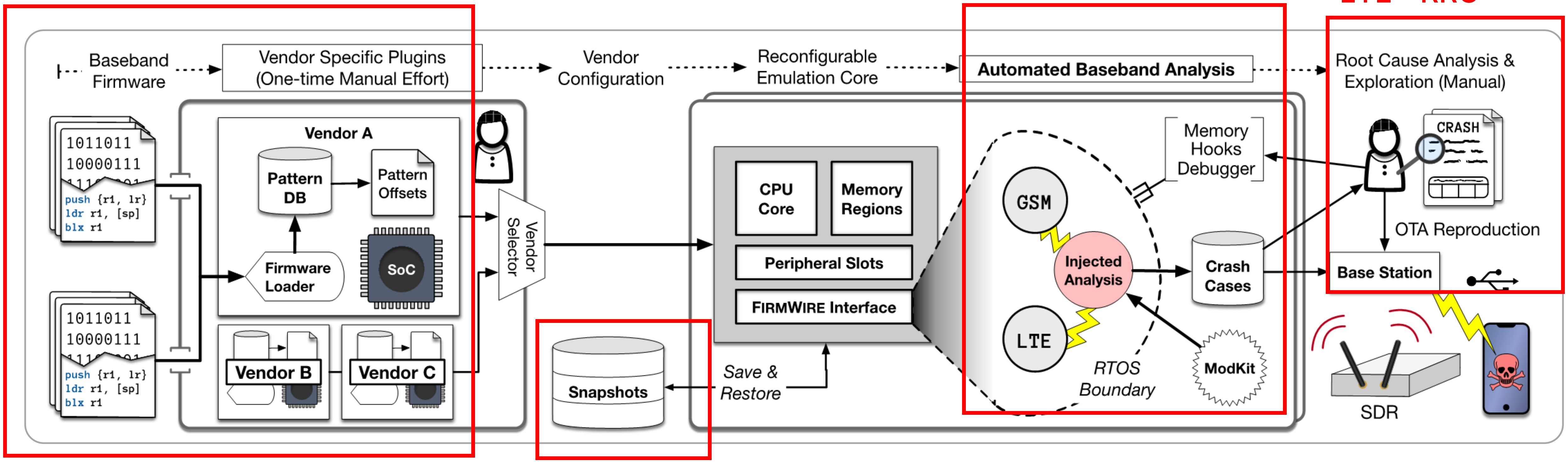
FirmWire

❖ State-of-the-art full-system baseband emulation platform

Samsung Shannon / MediaTek

Custom code injection

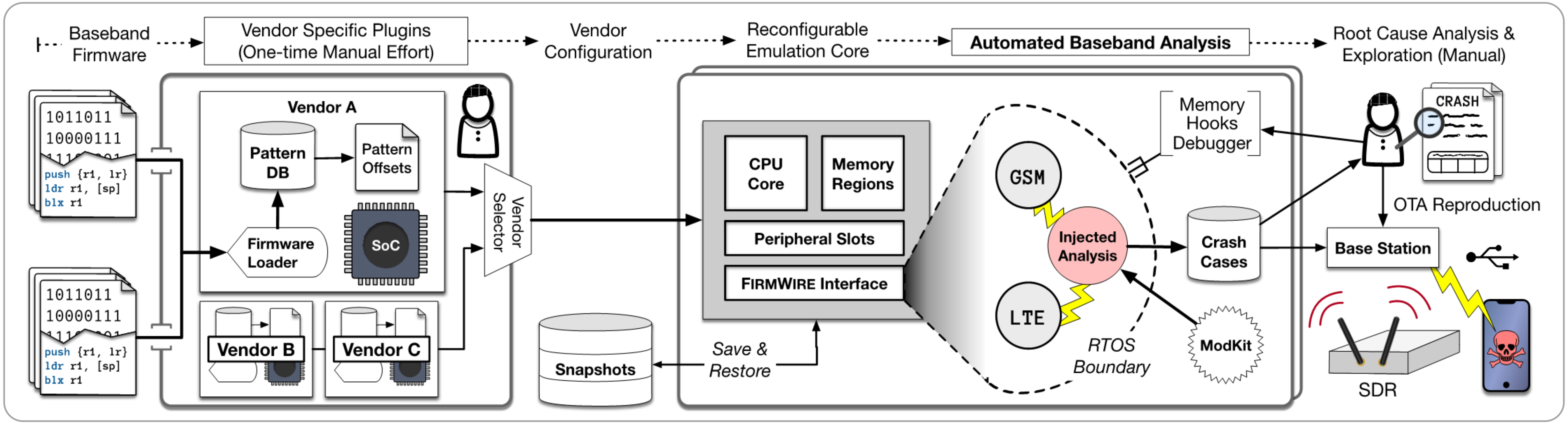
GSM – CC / SM
LTE - RRC



Memory snapshot

FirmWire

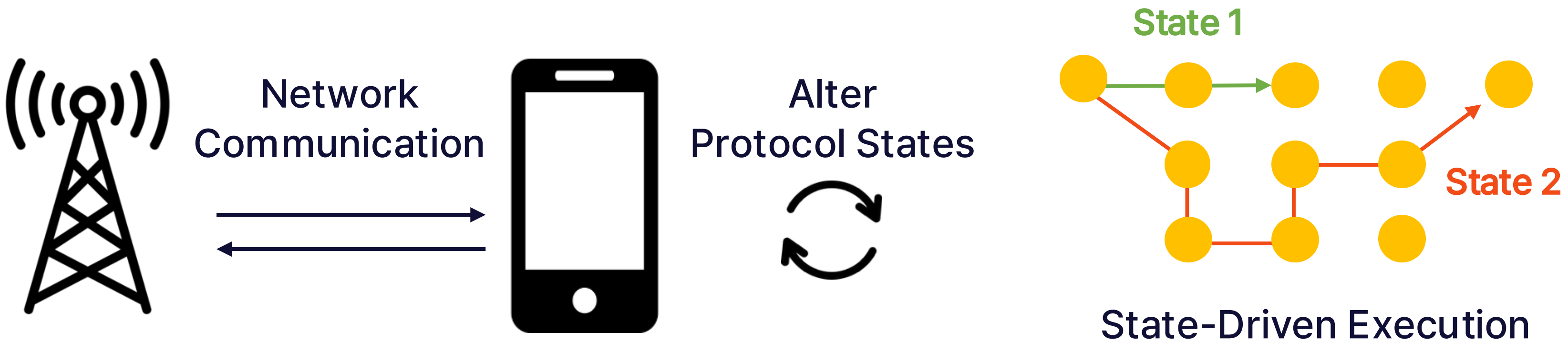
❖ Limitation: can not support the network communication



Challenge [C1]: Complex State Configuration

❖ Protocol states

- Fundamental to how baseband works (different states = different behaviors)
- Drastically change during cellular network communication



Network Communication

Alter Protocol States

State 1

State 2

State-Driven Execution

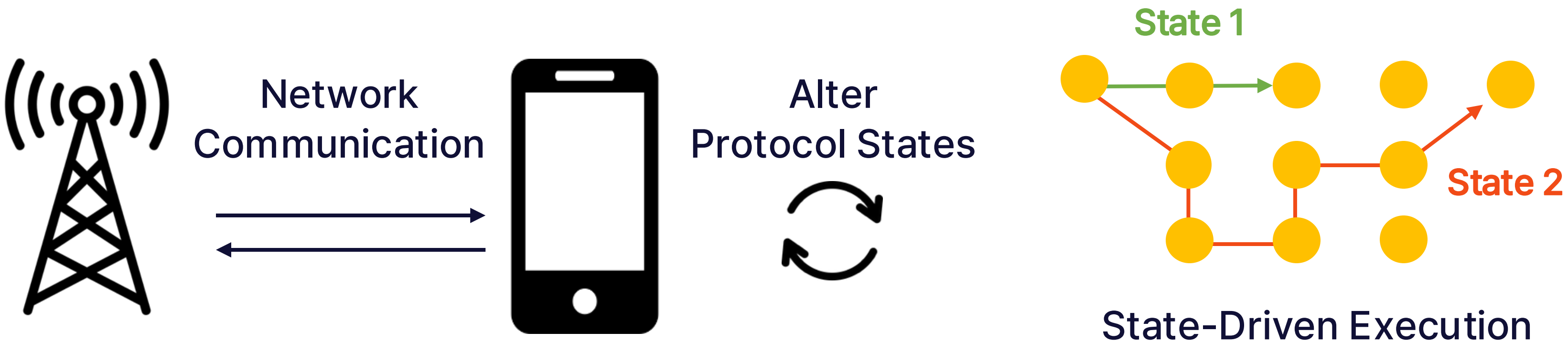
Challenge [C1]: Complex State Configuration

❖ Protocol states

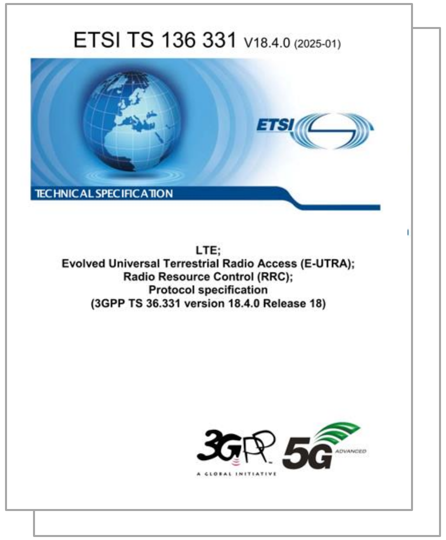
- Fundamental to how baseband works (different states = different behaviors)
- Drastically change during cellular network communication

❖ Main challenges of state configuration

1. Complex specifications (1000+ page documents)
2. Memory-level state representation



State-Driven Execution



ETSI TS 136 331 V18.4.0 (2025-01)

TECHNICAL SPECIFICATION

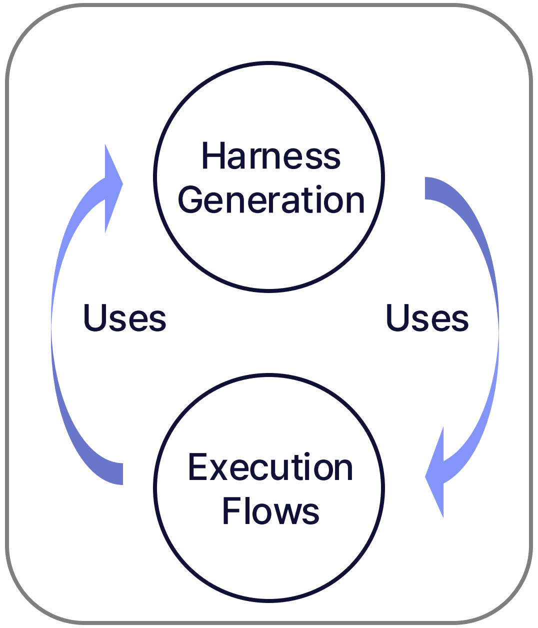
LTE;
Evolved Universal Terrestrial Radio Access (E-UTRA);
Radio Resource Control (RRC);
Protocol specification
(3GPP TS 36.331 version 18.4.0 Release 18)

LTE RRC Spec. (1165p)

Challenge [C2]: Control Flow Visibility

❖ Limited visibility into network-related execution flows

- FirmWire provides execution logs, only if the proper harness exists
- Circular dependency problem

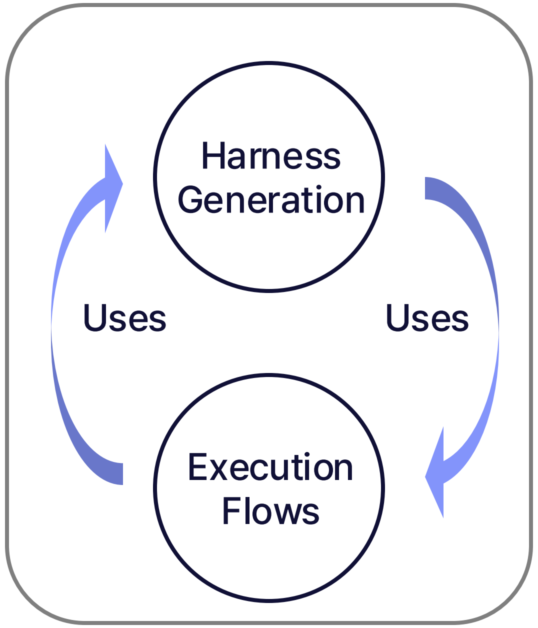


Circular Dependency

Challenge [C2]: Control Flow Visibility

- Limited visibility into network-related execution flows
  - FirmWire provides execution logs, only if the proper harness exists
  - Circular dependency problem

- Main challenges
  - Complex harness implementation
  - No reliable ground-truth



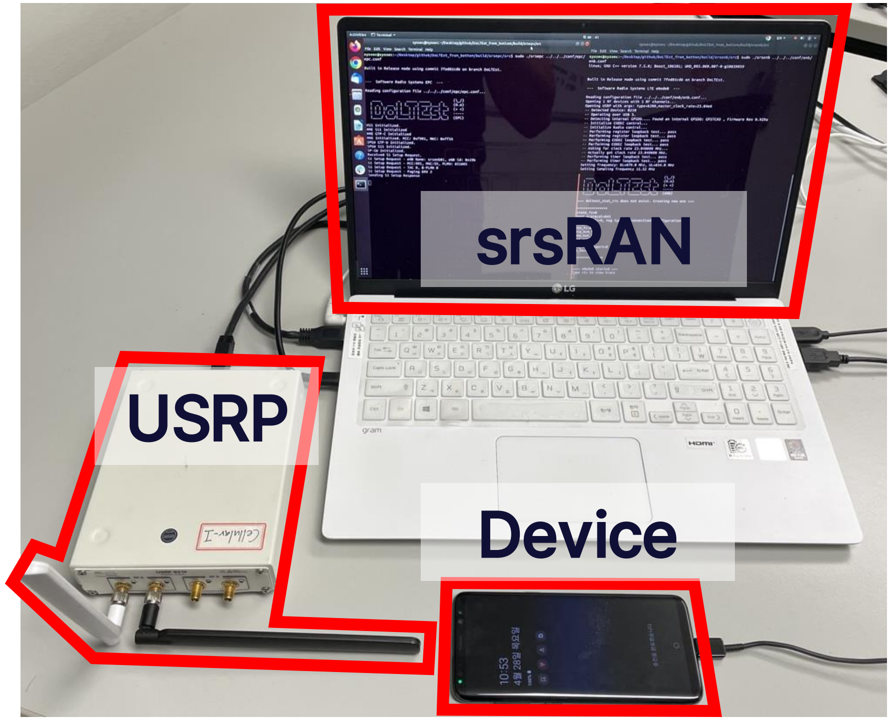
Circular Dependency

Our Approach [A1]: Runtime State Extraction

- ❖ Key Insight: Extract protocol states from real devices

Our Approach [A1]: Runtime State Extraction

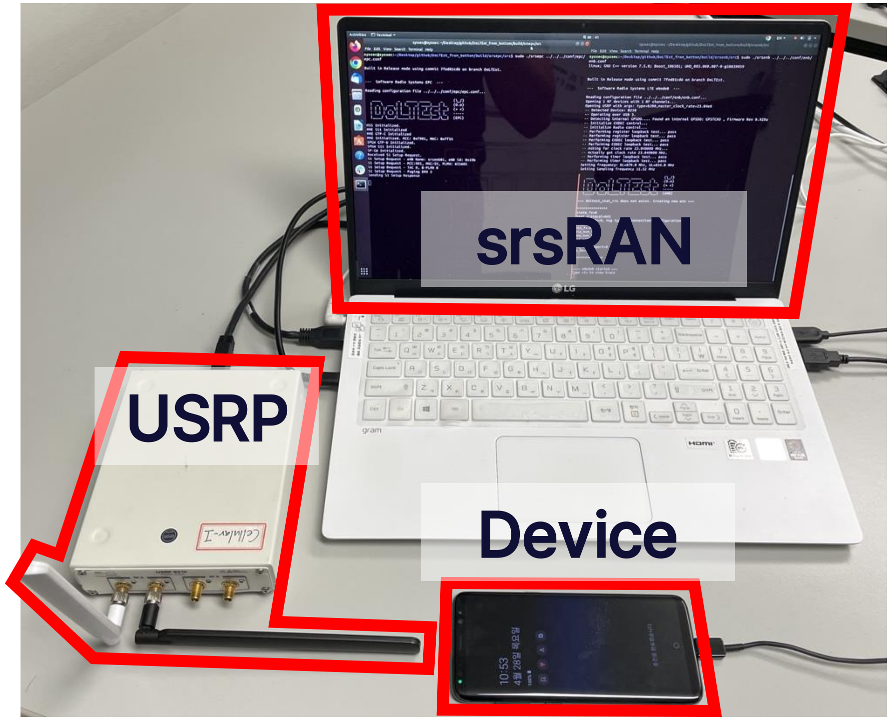
❖ Key Insight: Extract protocol states from real devices



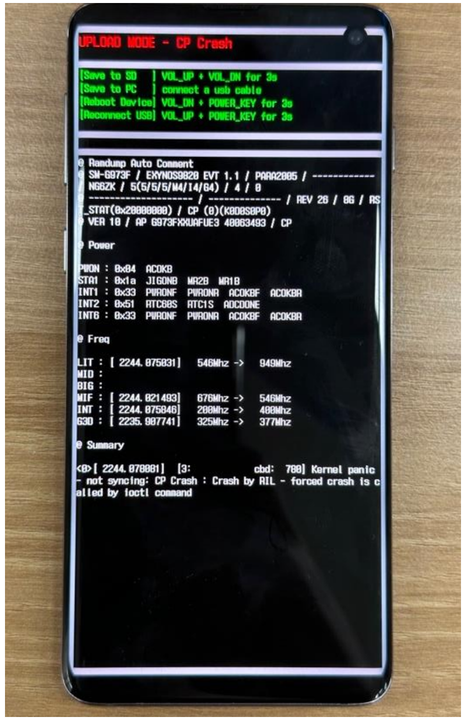
State Configuration

Our Approach [A1]: Runtime State Extraction

❖ Key Insight: Extract protocol states from real devices



State Configuration

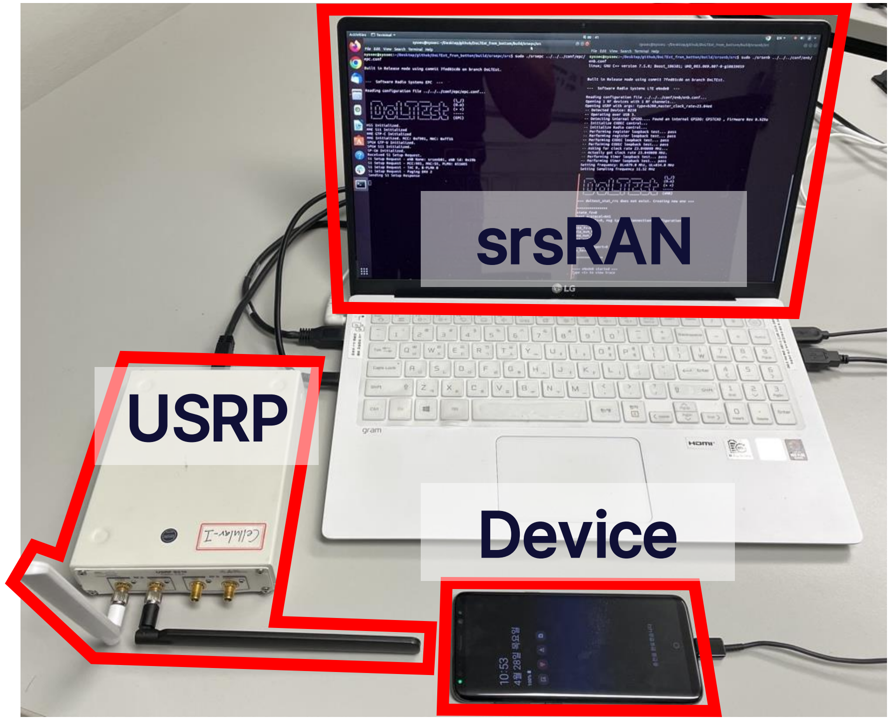


```
UPLOAD MODE - CP Crash

[Save to SD    ] VOL_UP + VOL_DN for 3s
[Save to PC    ] connect a usb cable
[Reboot Device] VOL_DN + POWER_KEY for 3s
[Reconnect USB] VOL_UP + POWER_KEY for 3s

@ Randump Auto Comment
@ SM-G973F / EXYNOS9820 EVT 1.1 / PARA2005 / ------------
/ NG6ZK / 5(5/5/5/M4/I4/G4) / 4 / 0
@ -------------------- / -------------- / REV 26 / 8G / RS
T_STAT(0x20000000) / CP (0)(K0D0S0P0)
@ VER 10 / AP G973FXXUAFUE3 40063493 / CP

@ Power

PWON : 0x04  ACOKB
STA1 : 0x1a  JIGONB  MR2B  MR1B
INT1 : 0x33  PWRONF  PWRONR  ACOKBF  ACOKBR
INT2 : 0x51  RTC60S  RTC1S  ADCDONE
INT6 : 0x33  PWRONF  PWRONR  ACOKBF  ACOKBR

@ Freq

LIT : [ 2244.075031]   546Mhz ->   949Mhz
MID :
BIG :
MIF : [ 2244.021493]   676Mhz ->   546Mhz
INT : [ 2244.075046]   200Mhz ->   400Mhz
G3D : [ 2235.907741]   325Mhz ->   377Mhz

@ Summary

<0>[ 2244.078081]  [3:            cbd:  700] Kernel panic
- not syncing: CP Crash : Crash by RIL - forced crash is c
alled by ioctl command
```

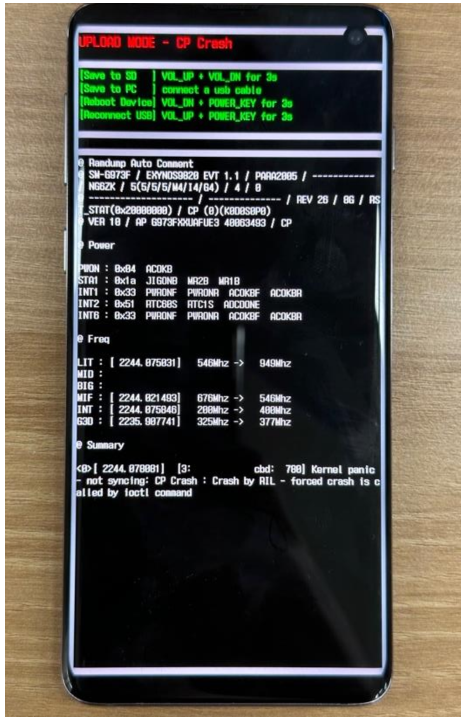
Force Crash

Our Approach [A1]: Runtime State Extraction

❖ Key Insight: Extract protocol states from real devices



State Configuration



```
UPLOAD MODE - CP Crash

[Save to SD    ] VOL_UP + VOL_DN for 3s
[Save to PC    ] connect a usb cable
[Reboot Device] VOL_DN + POWER_KEY for 3s
[Reconnect USB] VOL_UP + POWER_KEY for 3s

@ Randump Auto Comment
@ SM-G973F / EXYNOS9820 EVT 1.1 / PARA2005 / ------------
/ NG6ZK / 5(5/5/5/M4/I4/G4) / 4 / 0
@ -------------------- / -------------- / REV 26 / 8G / RS
T_STAT(0x20000000) / CP (0)(K0D0S0P0)
@ VER 10 / AP G973FXXUAFUE3 40063493 / CP

@ Power

PWON : 0x04  ACOKB
STA1 : 0x1a  JIGONB  MR2B  MR1B
INT1 : 0x33  PWRONF  PWRONR  ACOKBF  ACOKBR
INT2 : 0x51  RTC60S  RTC1S  ADCDONE
INT6 : 0x33  PWRONF  PWRONR  ACOKBF  ACOKBR

@ Freq

LIT : [ 2244.075031]   546Mhz ->   949Mhz
MID :
BIG :
MIF : [ 2244.021493]   676Mhz ->   546Mhz
INT : [ 2244.075846]   200Mhz ->   400Mhz
G3D : [ 2235.907741]   325Mhz ->   377Mhz

@ Summary

<0>[ 2244.078081]  [3:            cbd:  780] Kernel panic
- not syncing: CP Crash : Crash by RIL - forced crash is c
alled by ioctl command
```

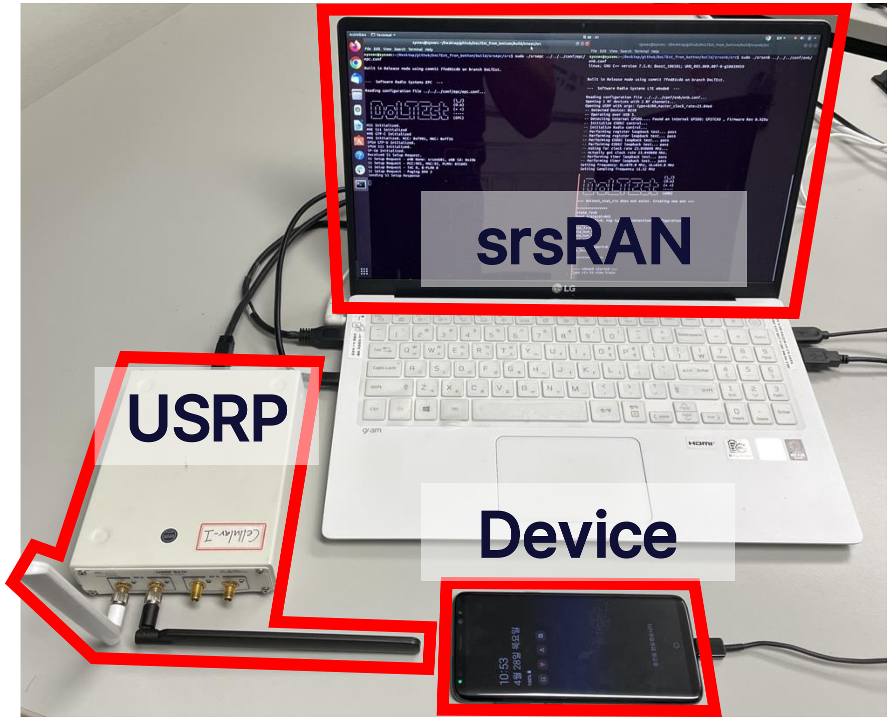
Force Crash

0100
1001
1101

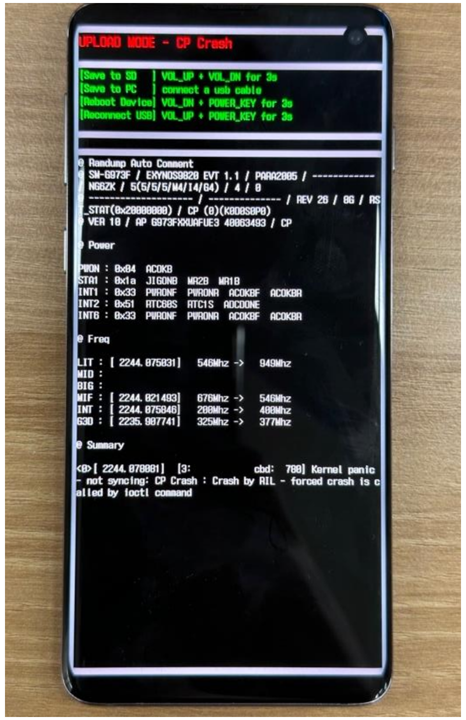
Memory dump

Our Approach [A1]: Runtime State Extraction

❖ Key Insight: Extract protocol states from real devices



State Configuration



```
UPLOAD MODE - CP Crash

[Save to SD    ] VOL_UP + VOL_DN for 3s
[Save to PC    ] connect a usb cable
[Reboot Device] VOL_DN + POWER_KEY for 3s
[Reconnect USB] VOL_UP + POWER_KEY for 3s

@ Ramdump Auto Comment
@ SM-G973F / EXYNOS9820 EVT 1.1 / PARA2005 / ------------
/ NG6ZK / 5(5/5/5/M4/I4/G4) / 4 / 0
@ --------------------- / -------------- / REV 26 / 8G / RS
T_STAT(0x20000000) / CP (0)(K0D0S0P0)
@ VER 10 / AP G973FXXUAFUE3 40063493 / CP

@ Power

PWON : 0x04  ACOKB
STA1 : 0x1a  JIGONB  MR2B  MR1B
INT1 : 0x33  PWRONF  PWRONR  ACOKBF  ACOKBR
INT2 : 0x51  RTC60S  RTC1S  ADCDONE
INT6 : 0x33  PWRONF  PWRONR  ACOKBF  ACOKBR

@ Freq

LIT : [ 2244.075031]   546Mhz ->   949Mhz
MID :
BIG :
MIF : [ 2244.021493]   676Mhz ->   546Mhz
INT : [ 2244.075046]   200Mhz ->   400Mhz
G3D : [ 2235.907741]   325Mhz ->   377Mhz

@ Summary

<0>[ 2244.078081]  [3:           cbd:  780] Kernel panic
- not syncing: CP Crash : Crash by RIL - forced crash is c
alled by ioctl command
```

Force Crash

0100
1001
1101

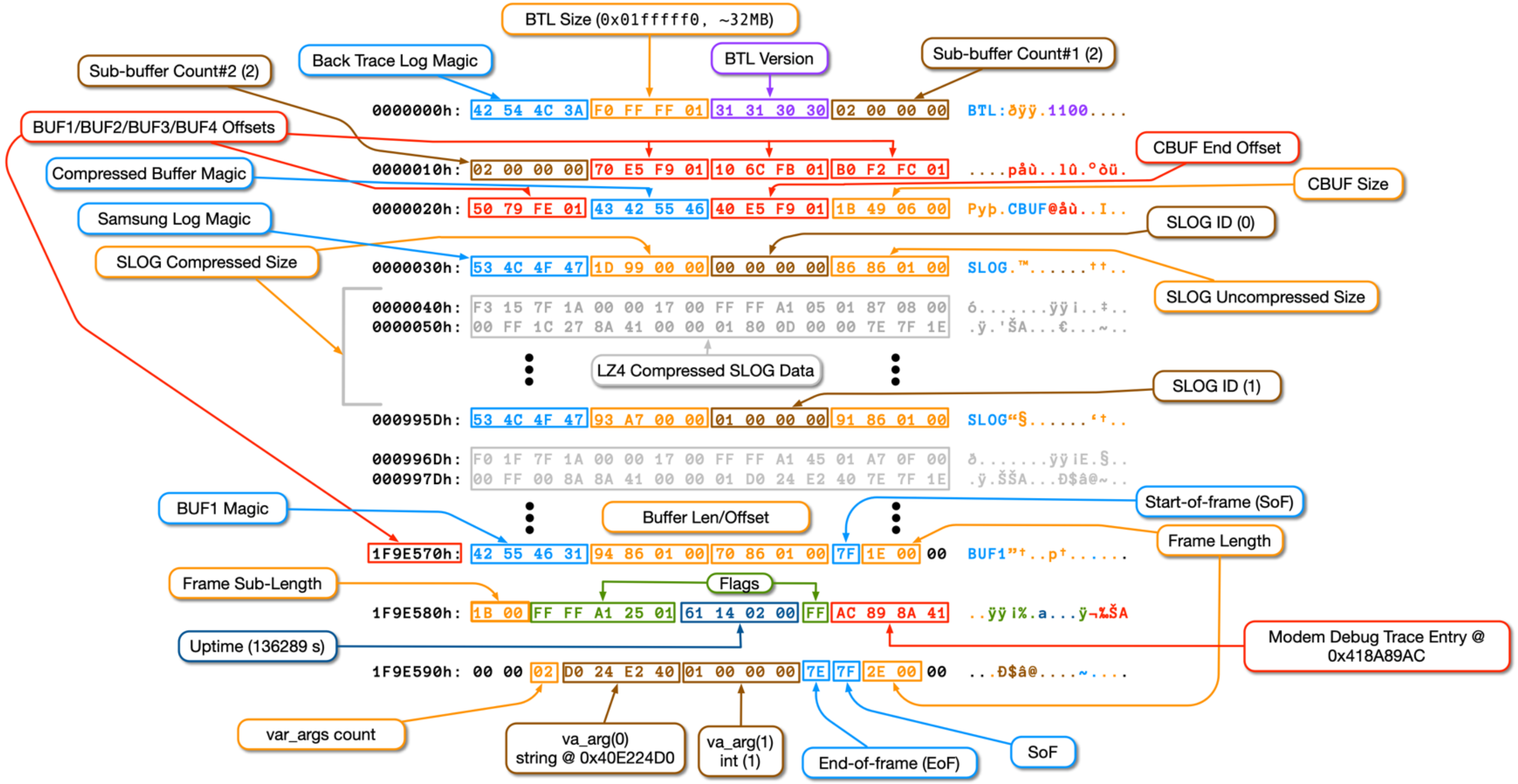
Memory dump

State Information

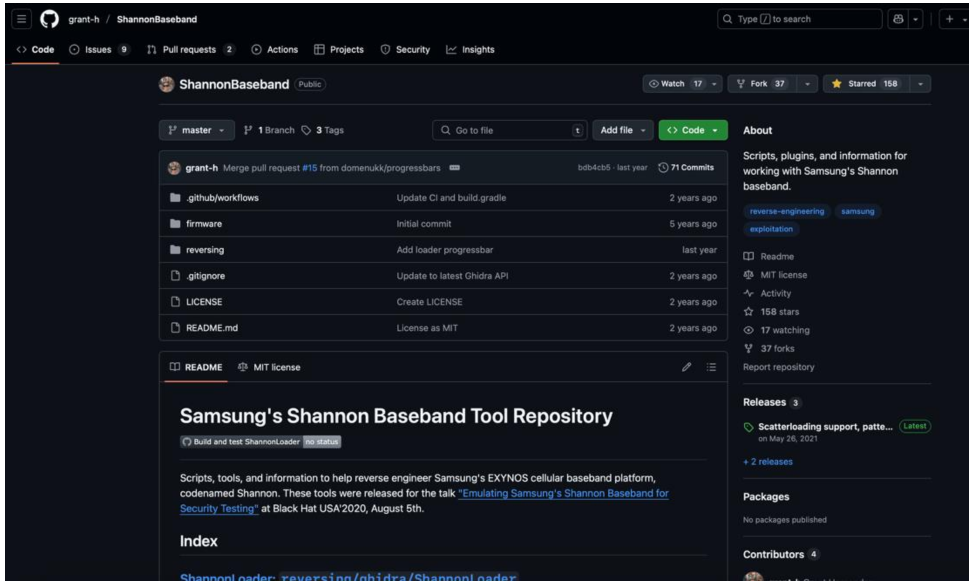
Our Approach [A2]: Control Flow Recovery

❖ Back Trace Log (BTL)

- Diverse information of real execution flow is encoded



```
0000000h: 42 54 4C 3A F0 FF FF 01 31 31 30 30 02 00 00 00  BTL:ðÿÿ.1100....
0000010h: 02 00 00 00 70 E5 F9 01 10 6C FB 01 B0 F2 FC 01  ....påù..lû.°òü.
0000020h: 50 79 FE 01 43 42 55 46 40 E5 F9 01 1B 49 06 00  Pyþ.CBUF@åù..I..
0000030h: 53 4C 4F 47 1D 99 00 00 00 00 00 00 86 86 01 00  SLOG.™......††..
0000040h: F3 15 7F 1A 00 00 17 00 FF FF A1 05 01 87 08 00  ó.......ÿÿ¡..‡..
0000050h: 00 FF 1C 27 8A 41 00 00 01 80 0D 00 00 7E 7F 1E  .ÿ.'ŠA...€...~..
    ⋮                                           ⋮
000995Dh: 53 4C 4F 47 93 A7 00 00 01 00 00 00 91 86 01 00  SLOG“§......‘†..
000996Dh: F0 1F 7F 1A 00 00 17 00 FF FF A1 45 01 A7 0F 00  ð.......ÿÿ¡E.§..
000997Dh: 00 FF 00 8A 8A 41 00 00 01 D0 24 E2 40 7E 7F 1E  .ÿ.ŠŠA...Ð$â@~..
    ⋮                                           ⋮
1F9E570h: 42 55 46 31 94 86 01 00 70 86 01 00 7F 1E 00 00  BUF1”†..p†......
1F9E580h: 1B 00 FF FF A1 25 01 61 14 02 00 FF AC 89 8A 41  ..ÿÿ¡%.a...ÿ¬‰ŠA
1F9E590h: 00 00 02 D0 24 E2 40 01 00 00 00 7E 7F 2E 00 00  ...Ð$â@....~....
```



grant-h Merge pull request #15 from domenukk/progressbars — bdb4cb5 · last year — 71 Commits

.github/workflows	Update CI and build.gradle	2 years ago
firmware	Initial commit	5 years ago
reversing	Add loader progressbar	last year
.gitignore	Update to latest Ghidra API	2 years ago
LICENSE	Create LICENSE	2 years ago
README.md	License as MIT	2 years ago

About

Scripts, plugins, and information for working with Samsung's Shannon baseband.

reverse-engineering samsung exploitation

Readme
MIT license
Activity
158 stars
17 watching
37 forks
Report repository

Releases 3

Scatterloading support, patte... Latest
on May 26, 2021

+ 2 releases

Packages

No packages published

Contributors 4

README MIT license

Samsung's Shannon Baseband Tool Repository

Build and test ShannonLoader no status

Scripts, tools, and information to help reverse engineer Samsung's EXYNOS cellular baseband platform, codenamed Shannon. These tools were released for the talk "Emulating Samsung's Shannon Baseband for Security Testing" at Black Hat USA'2020, August 5th.

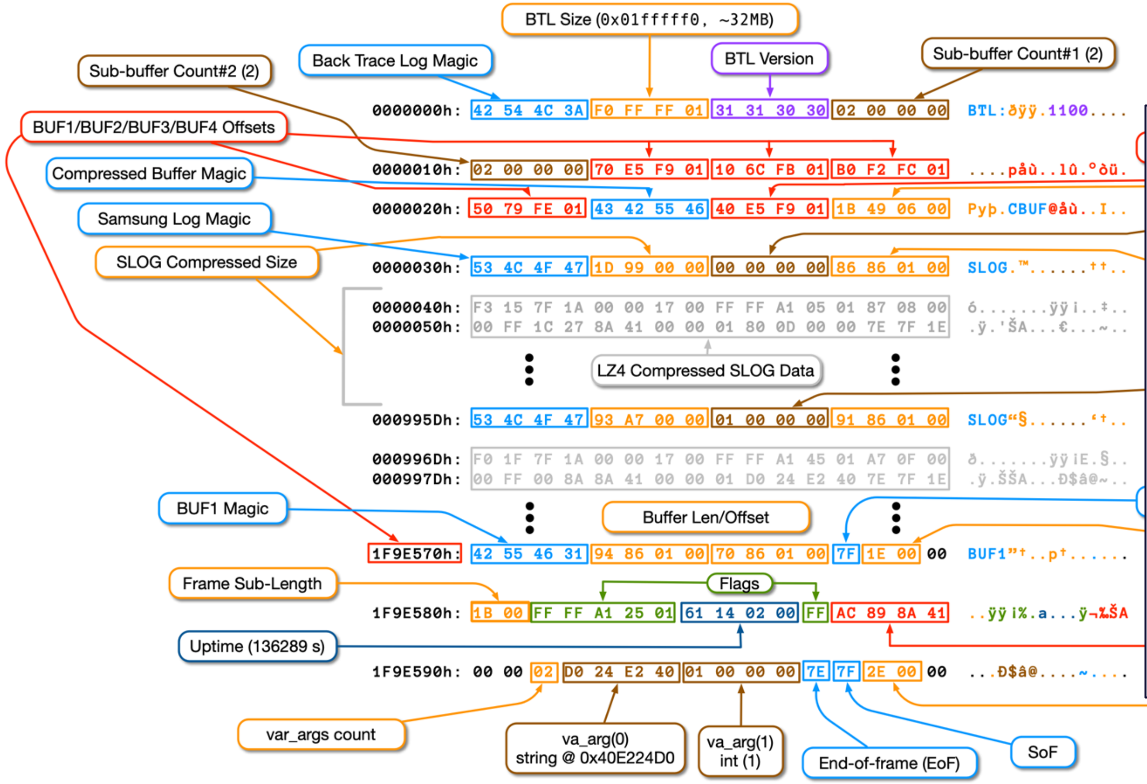
Index

ShannonLoader: `reversing/ghidra/ShannonLoader`

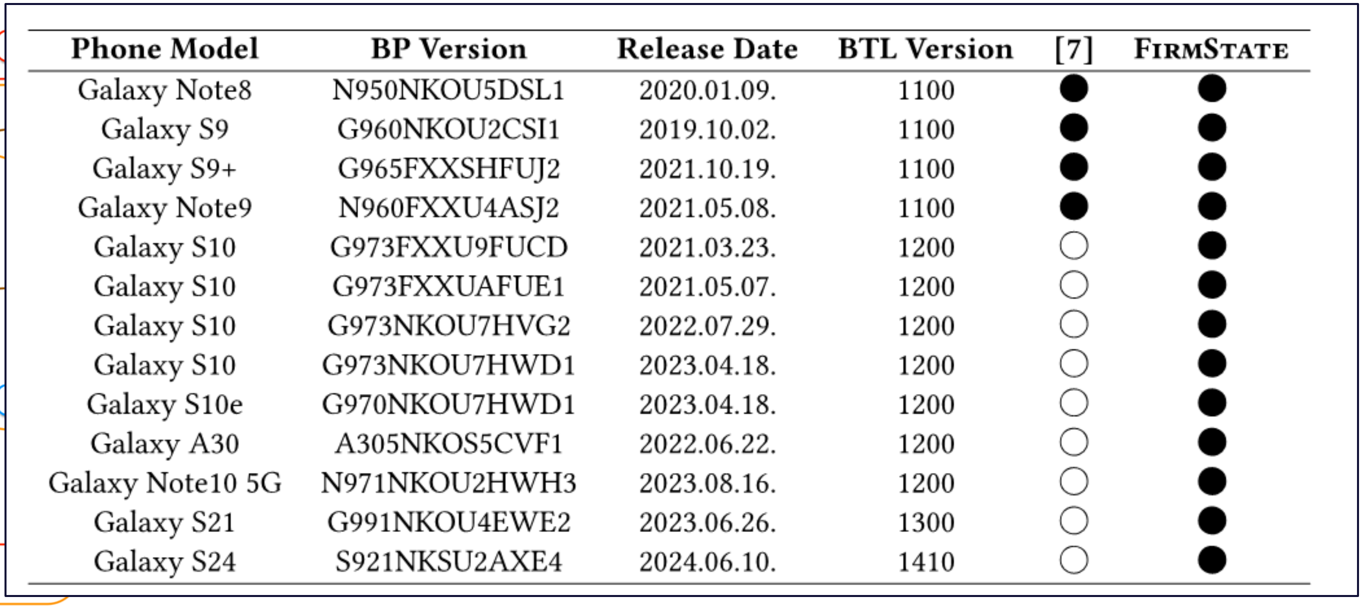
Our Approach [A2]: Control Flow Recovery

❖ Back Trace Log (BTL)

- Diverse information of real execution flow is encoded



```
0000000h: 42 54 4C 3A F0 FF FF 01 31 31 30 30 02 00 00 00  BTL:ðÿÿ.1100....
0000010h: 02 00 00 00 70 E5 F9 01 10 6C FB 01 B0 F2 FC 01  ....påù..lû.°òü.
0000020h: 50 79 FE 01 43 42 55 46 40 E5 F9 01 1B 49 06 00  Pyþ.CBUF@åù..I..
0000030h: 53 4C 4F 47 1D 99 00 00 00 00 00 00 86 86 01 00  SLOG.™......††..
0000040h: F3 15 7F 1A 00 00 17 00 FF FF A1 05 01 87 08 00  ó.......ÿÿ¡..‡..
0000050h: 00 FF 1C 27 8A 41 00 00 01 80 0D 00 00 7E 7F 1E  .ÿ.'ŠA...€...~..
    ⋮                                               ⋮
000995Dh: 53 4C 4F 47 93 A7 00 00 01 00 00 00 91 86 01 00  SLOG“§......‘†..
000996Dh: F0 1F 7F 1A 00 00 17 00 FF FF A1 45 01 A7 0F 00  ð.......ÿÿ¡E.§..
000997Dh: 00 FF 00 8A 8A 41 00 00 01 D0 24 E2 40 7E 7F 1E  .ÿ.ŠŠA...Ð$â@~..
    ⋮                                               ⋮
1F9E570h: 42 55 46 31 94 86 01 00 70 86 01 00 7F 1E 00 00  BUF1”†..p†......
1F9E580h: 1B 00 FF FF A1 25 01 61 14 02 00 FF AC 89 8A 41  ..ÿÿ¡%.a...ÿ¬‰ŠA
1F9E590h: 00 00 02 D0 24 E2 40 01 00 00 00 7E 7F 2E 00 00  ...Ð$â@....~....
```



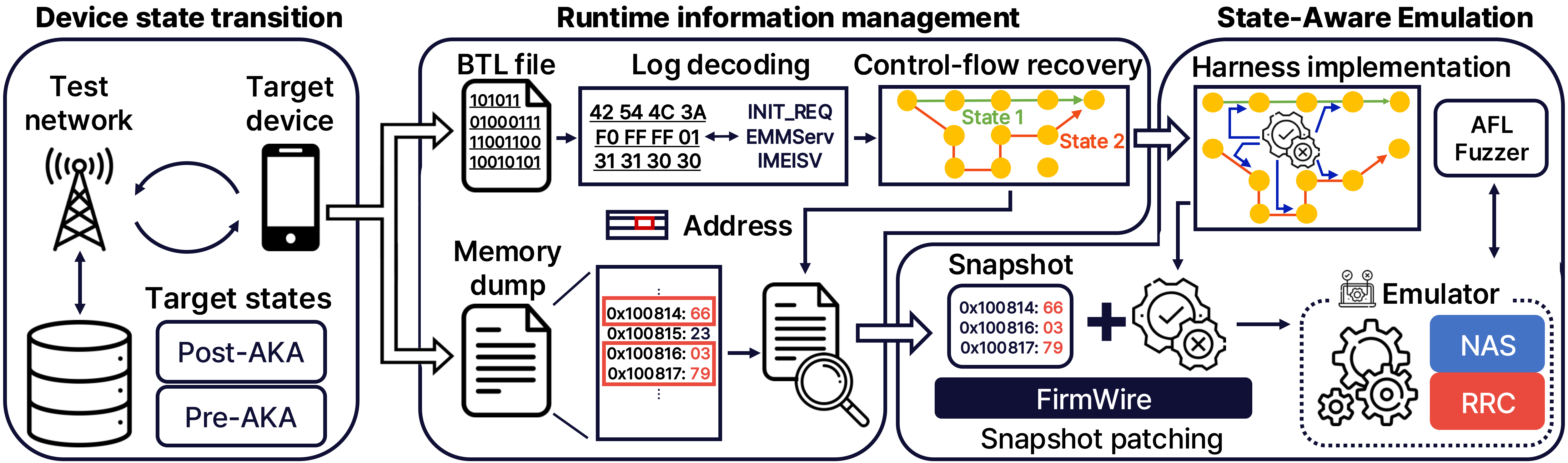
Phone Model	BP Version	Release Date	BTL Version	[7]	FIRMSTATE
Galaxy Note8	N950NKOU5DSL1	2020.01.09.	1100	●	●
Galaxy S9	G960NKOU2CSI1	2019.10.02.	1100	●	●
Galaxy S9+	G965FXXSHFUJ2	2021.10.19.	1100	●	●
Galaxy Note9	N960FXXU4ASJ2	2021.05.08.	1100	●	●
Galaxy S10	G973FXXU9FUCD	2021.03.23.	1200	○	●
Galaxy 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	○	●

Support for 4 distinct BTL format versions (~S24)

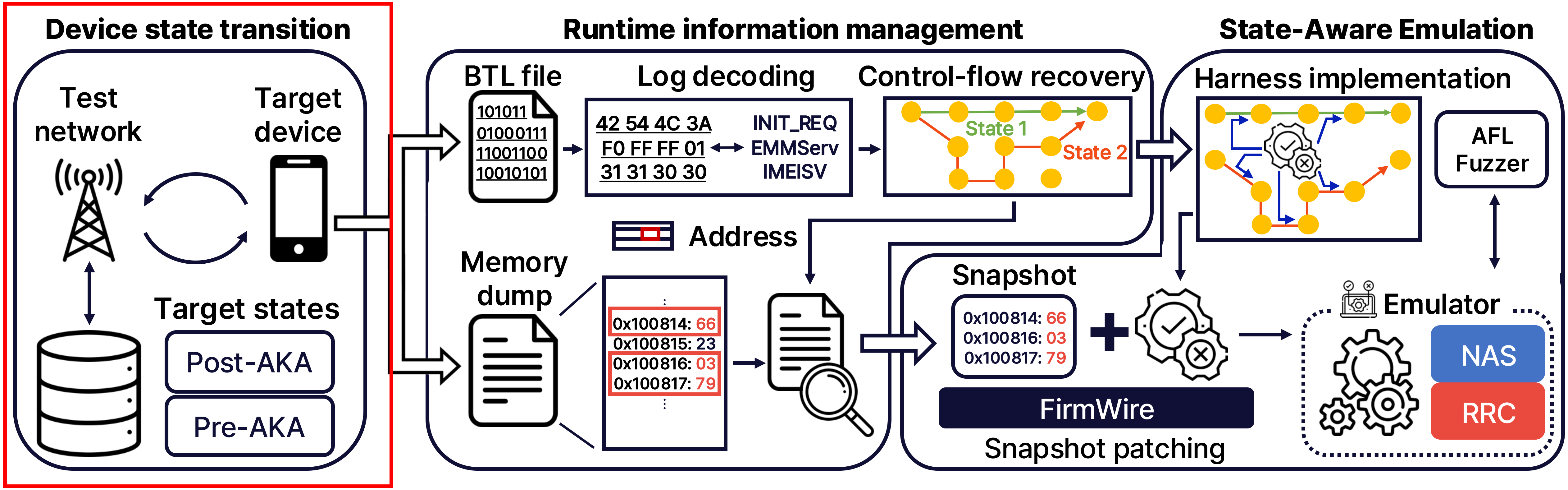
Overview – FirmState

❖ State-aware methodology enhancing Shannon baseband emulation

- https://github.com/1nteger-c/FirmState

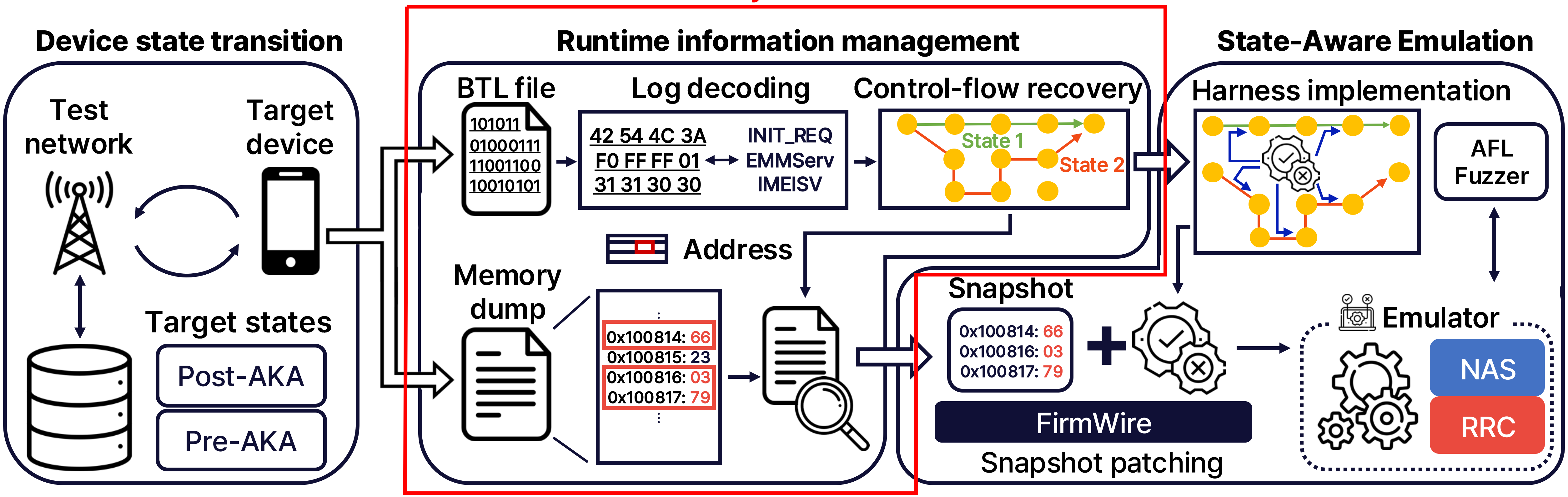


Overview – FirmState



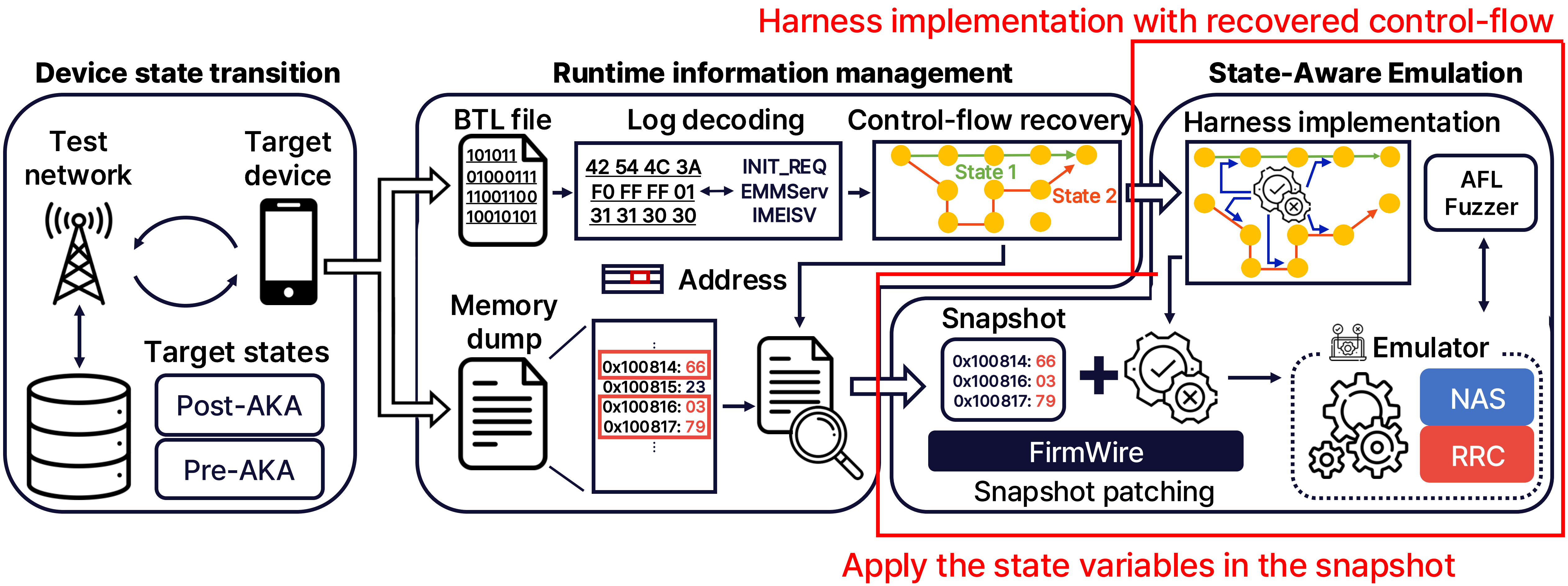
Overview – FirmState

Control flow recovery from BTL



State extraction from memory dump

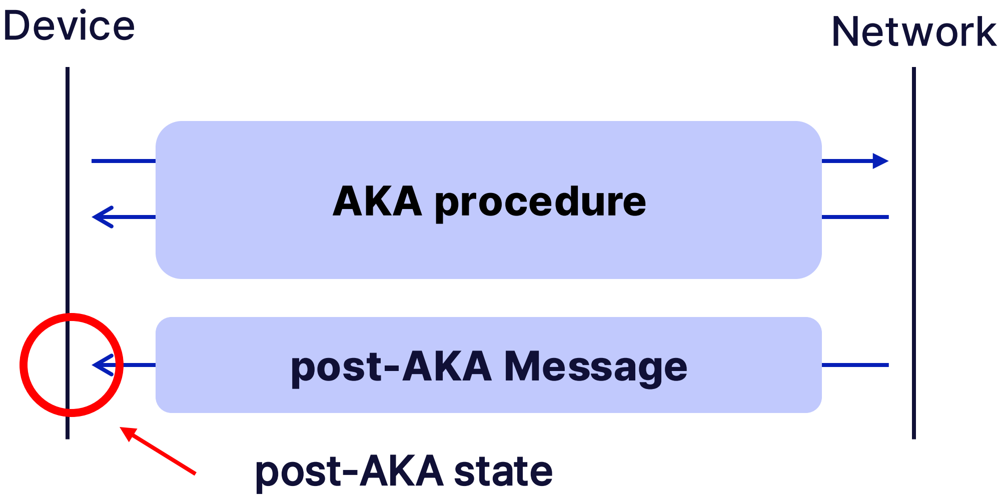
Overview – FirmState



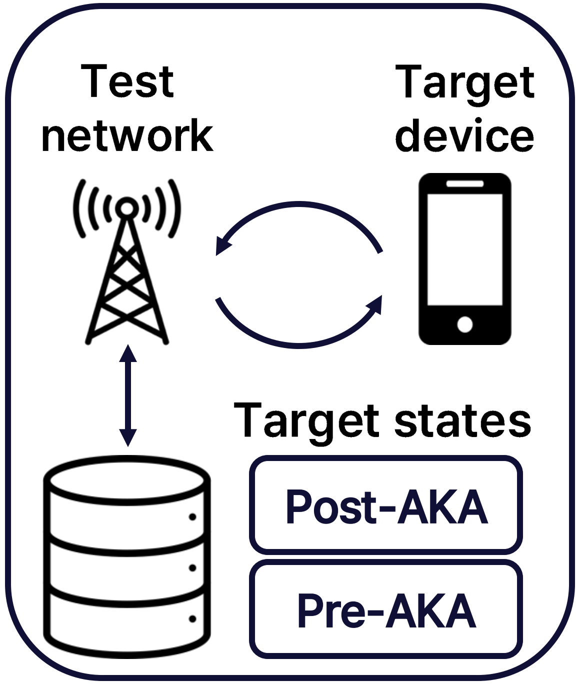
Phase 1: Device State Transition

- ❖ Controlled testbed enables precise baseband state manipulation
  - Controls network conditions and protocol message sequences
  - Can reach target protocol states

- ❖ Implementation based on open-source infrastructure
  - srsRAN 4G, USRP B200



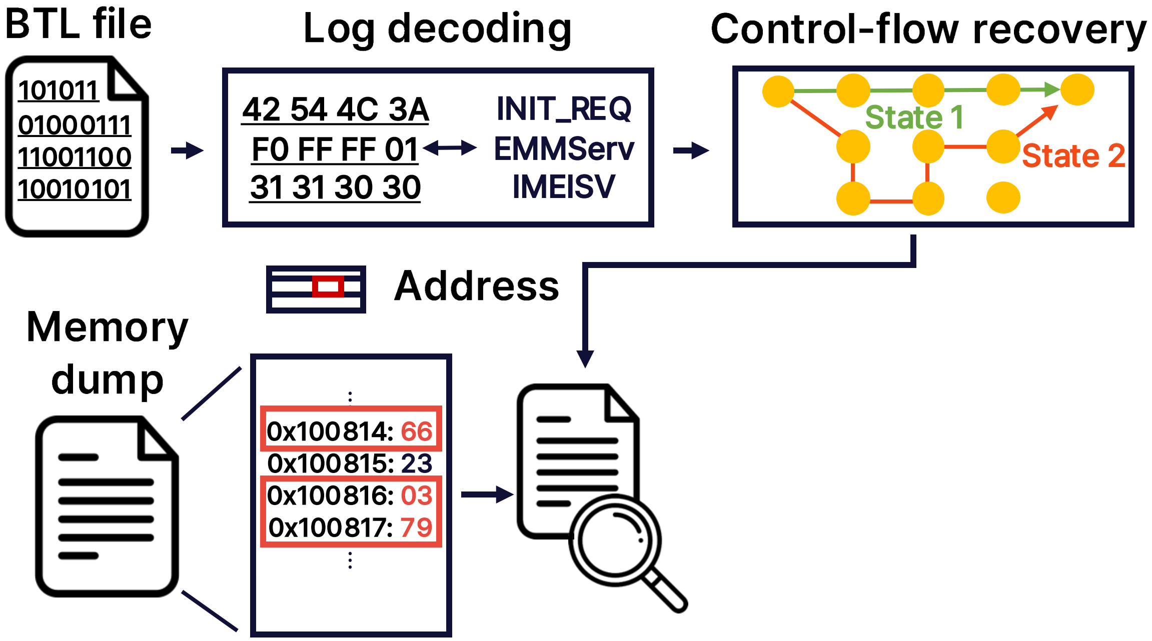
Device state transition

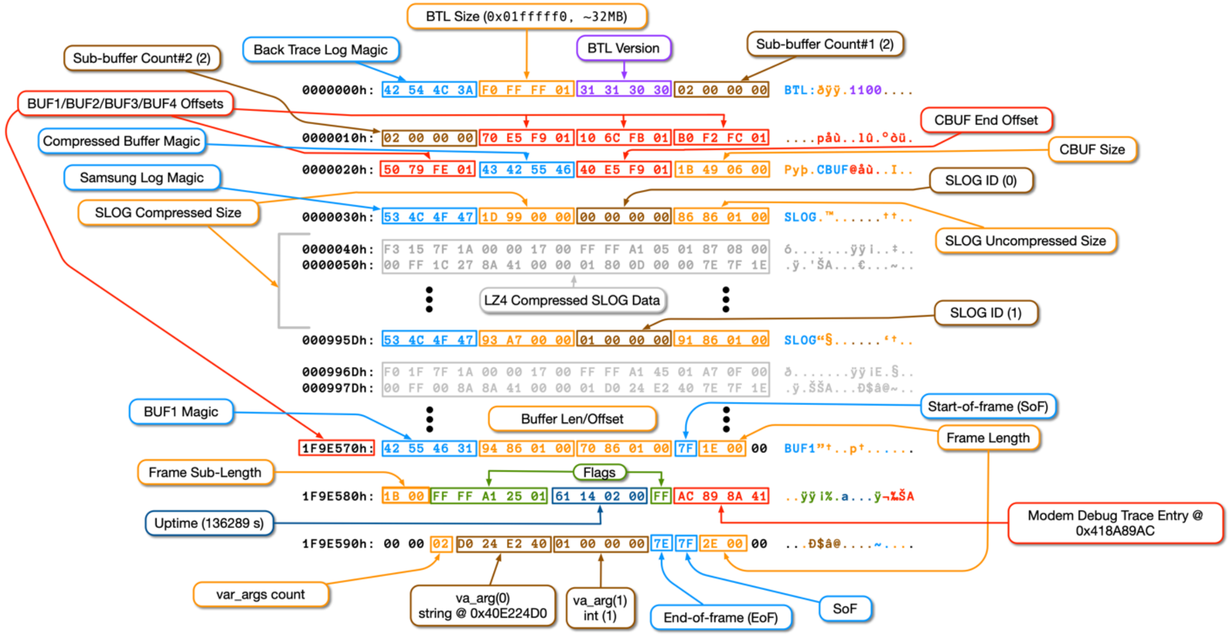


Phase 2: Runtime Information Management

❖ FirmState correlates two critical data sources

- BTL file analysis: Understanding actual control flow execution
- Memory dump processing: Extract state information

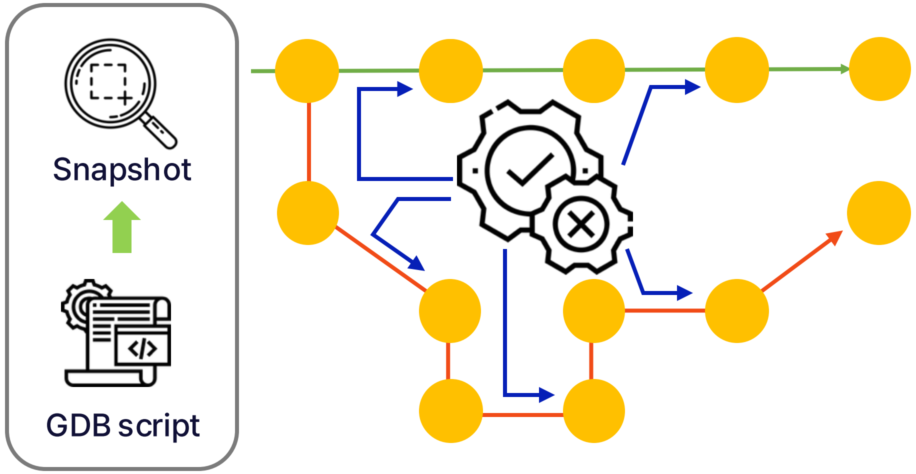




```
0000000h: 42 54 4C 3A F0 FF FF 01 31 31 30 30 02 00 00 00  BTL:ðÿÿ.1100....
0000010h: 02 00 00 00 70 E5 F9 01 10 6C FB 01 B0 F2 FC 01  ....påù..lû.°òü.
0000020h: 50 79 FE 01 43 42 55 46 40 E5 F9 01 1B 49 06 00  Pyþ.CBUF@åù..I..
0000030h: 53 4C 4F 47 1D 99 00 00 00 00 00 00 86 86 01 00  SLOG.™......††..
0000040h: F3 15 7F 1A 00 00 17 00 FF FF A1 05 01 87 08 00  ó.......ÿÿ¡..‡..
0000050h: 00 FF 1C 27 8A 41 00 00 01 80 0D 00 00 7E 7F 1E  .ÿ.'ŠA...€...~..
    ⋮                                      ⋮
000995Dh: 53 4C 4F 47 93 A7 00 00 01 00 00 00 91 86 01 00  SLOG“§......‘†..

000996Dh: F0 1F 7F 1A 00 00 17 00 FF FF A1 45 01 A7 0F 00  ð.......ÿÿ¡E.§..
000997Dh: 00 FF 00 8A 8A 41 00 00 01 D0 24 E2 40 7E 7F 1E  .ÿ.ŠŠA...Ð$â@~..
    ⋮                                      ⋮
1F9E570h: 42 55 46 31 94 86 01 00 70 86 01 00 7F 1E 00 00  BUF1”†..p†......
1F9E580h: 1B 00 FF FF A1 25 01 61 14 02 00 FF AC 89 8A 41  ..ÿÿ¡%.a...ÿ¬‰ŠA
1F9E590h: 00 00 02 D0 24 E2 40 01 00 00 00 7E 7F 2E 00 00  ...Ð$â@....~....
```

Implementation

- ❖ Snapshot-Patching Procedure: state application
  - Seamless integration with FirmWire's snapshot system
- ❖ Support pre/post-AKA states
  - Higher protocol coverage at RRC & fidelity
- ❖ Newly support LTE NAS

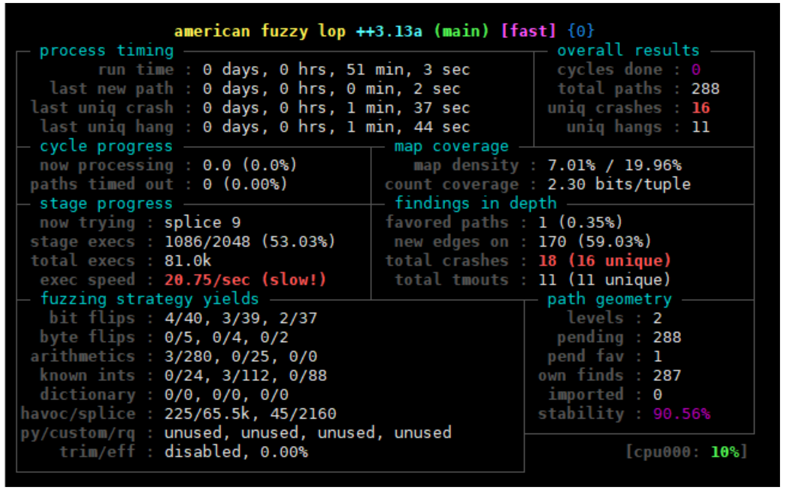


Snapshot-Patching Procedure

Evaluation [1] - Fuzzer Performance

❖ Comparison with FirmWire baseline

- 24-hour evaluation periods with 3 independent runs



```
                 american fuzzy lop ++3.13a (main) [fast] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 51 min, 3 sec      │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 2 sec       │  total paths : 288   │
│ last uniq crash : 0 days, 0 hrs, 1 min, 37 sec      │ uniq crashes : 16    │
│  last uniq hang : 0 days, 0 hrs, 1 min, 44 sec      │   uniq hangs : 11    │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 0.0 (0.0%)        │    map density : 7.01% / 19.96%       │
│ paths timed out : 0 (0.00%)         │ count coverage : 2.30 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : splice 9              │ favored paths : 1 (0.35%)             │
│ stage execs : 1086/2048 (53.03%)    │  new edges on : 170 (59.03%)          │
│ total execs : 81.0k                 │ total crashes : 18 (16 unique)        │
│  exec speed : 20.75/sec (slow!)     │  total tmouts : 11 (11 unique)        │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ───────┤
│   bit flips : 4/40, 3/39, 2/37                      │    levels : 2         │
│  byte flips : 0/5, 0/4, 0/2                         │   pending : 288       │
│ arithmetics : 3/280, 0/25, 0/0                      │  pend fav : 1         │
│  known ints : 0/24, 3/112, 0/88                     │ own finds : 287       │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : 0         │
│havoc/splice : 225/65.5k, 45/2160                    │ stability : 90.56%    │
│py/custom/rq : unused, unused, unused, unused        ├───────────────────────┘
│    trim/eff : disabled, 0.00%                       │          [cpu000: 10%]
└─────────────────────────────────────────────────────┘
```

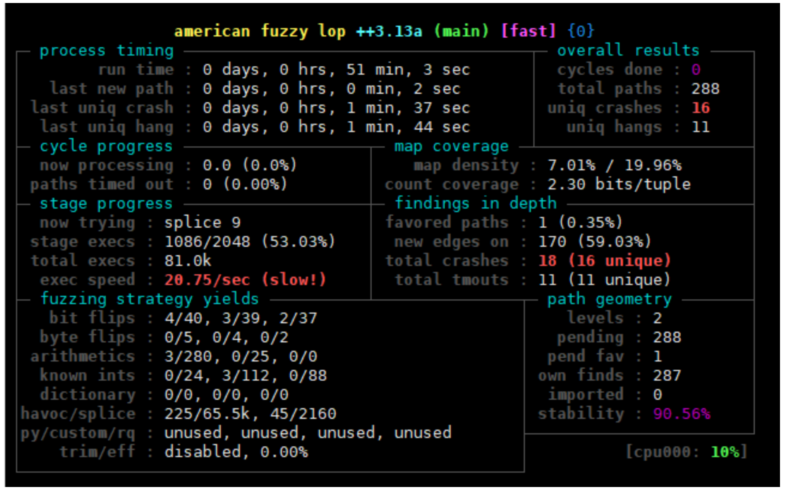
Evaluation [1] - Fuzzer Performance

❖ Comparison with FirmWire baseline

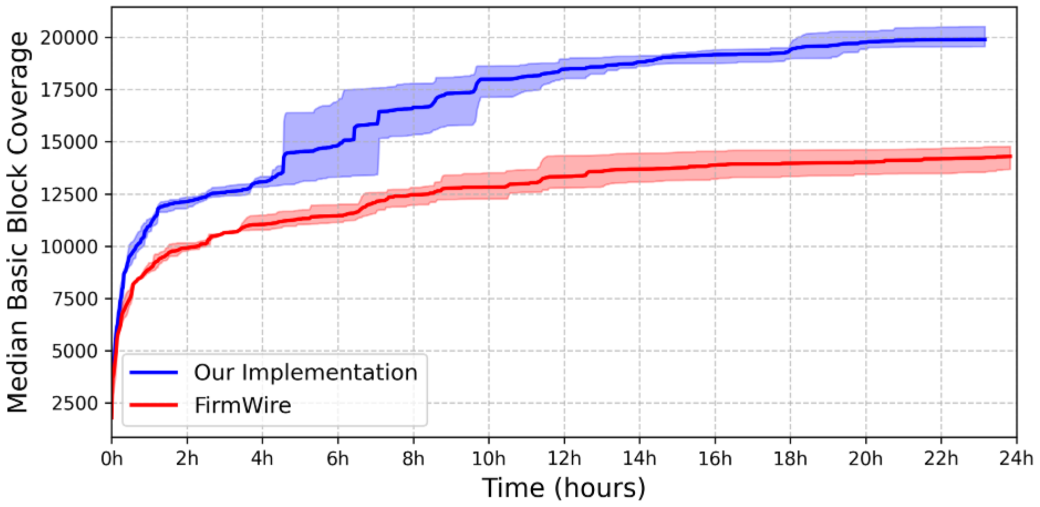
- 24-hour evaluation periods with 3 independent runs

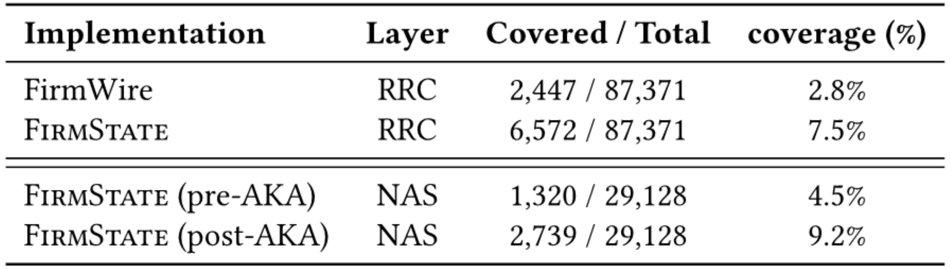
❖ Significant Coverage Improvements

- RRC: 7.5% coverage (2.7× improvement over FirmWire's 2.8%)
- NAS: 4.5%-9.2% coverage (previously unsupported)
- Two 1-day vulnerabilities discovered in different protocol states



```
             american fuzzy lop ++3.13a (main) [fast] {0}
 process timing                                      overall results
        run time : 0 days, 0 hrs, 51 min, 3 sec       cycles done : 0
   last new path : 0 days, 0 hrs, 0 min, 2 sec        total paths : 288
 last uniq crash : 0 days, 0 hrs, 1 min, 37 sec      uniq crashes : 16
  last uniq hang : 0 days, 0 hrs, 1 min, 44 sec        uniq hangs : 11
 cycle progress                       map coverage
  now processing : 0.0 (0.0%)           map density : 7.01% / 19.96%
 paths timed out : 0 (0.00%)         count coverage : 2.30 bits/tuple
 stage progress                       findings in depth
  now trying : splice 9              favored paths : 1 (0.35%)
 stage execs : 1086/2048 (53.03%)     new edges on : 170 (59.03%)
 total execs : 81.0k                 total crashes : 18 (16 unique)
  exec speed : 20.75/sec (slow!)      total tmouts : 11 (11 unique)
 fuzzing strategy yields                            path geometry
   bit flips : 4/40, 3/39, 2/37                       levels : 2
  byte flips : 0/5, 0/4, 0/2                         pending : 288
 arithmetics : 3/280, 0/25, 0/0                     pend fav : 1
  known ints : 0/24, 3/112, 0/88                   own finds : 287
  dictionary : 0/0, 0/0, 0/0                        imported : 0
havoc/splice : 225/65.5k, 45/2160                  stability : 90.56%
py/custom/rq : unused, unused, unused, unused
    trim/eff : disabled, 0.00%                          [cpu000: 10%]
```



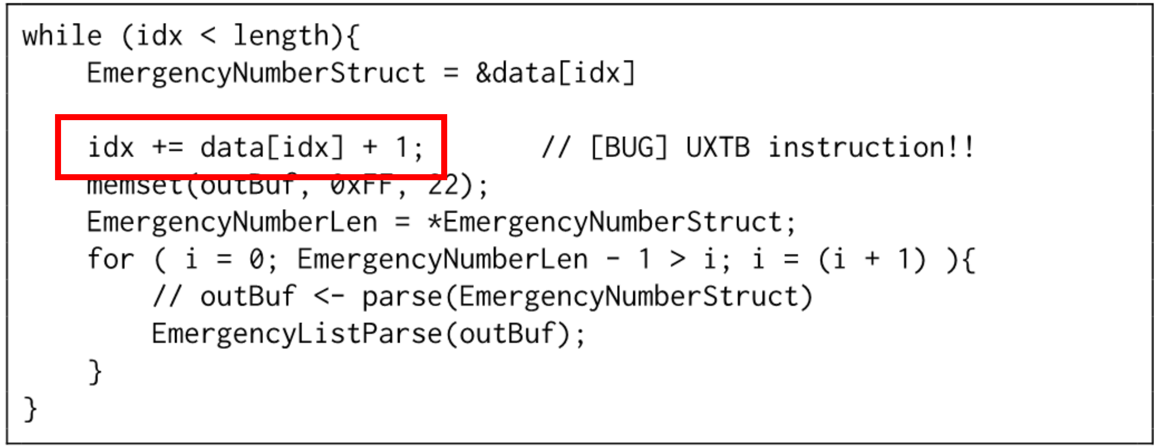


Implementation	Layer	Covered / Total	coverage (%)
FirmWire	RRC	2,447 / 87,371	2.8%
FIRMSTATE	RRC	6,572 / 87,371	7.5%
FIRMSTATE (pre-AKA)	NAS	1,320 / 29,128	4.5%
FIRMSTATE (post-AKA)	NAS	2,739 / 29,128	9.2%

Evaluation [2] - Root Cause Analysis

- ❖ Proper emulation directly results to root cause analysis
  - Instruction Trace Analysis (QEMU)
  - Debugging (GDB)

- ❖ Vulnerability Details
  - Pre-AKA: Integer underflow in buffer copying mechanism
  - Post-AKA: Infinite loop in Emergency Number List parsing



```
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);
    }
}
```

Infinity loop in decoding EmergencyNumberList

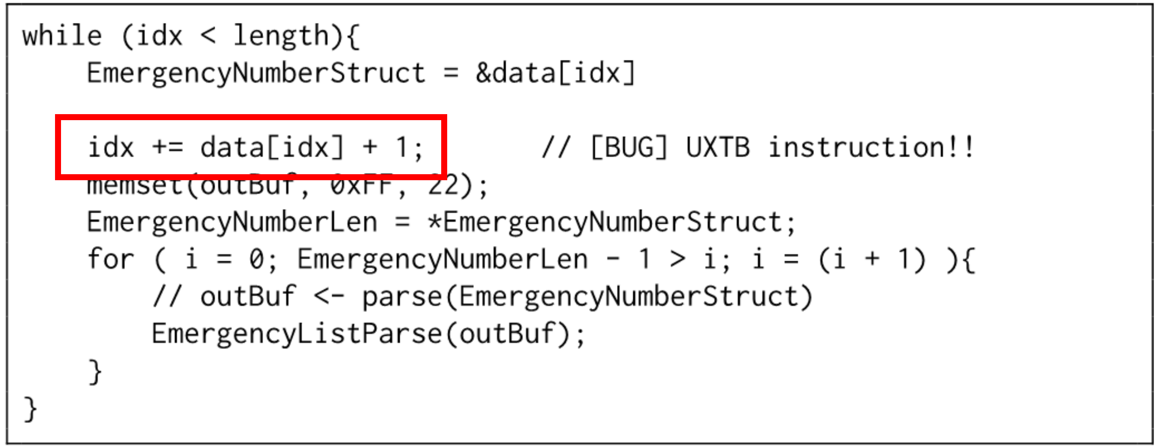
Evaluation [2] - Root Cause Analysis

❖ Proper emulation directly results to root cause analysis

- Instruction Trace Analysis (QEMU)
- Debugging (GDB)

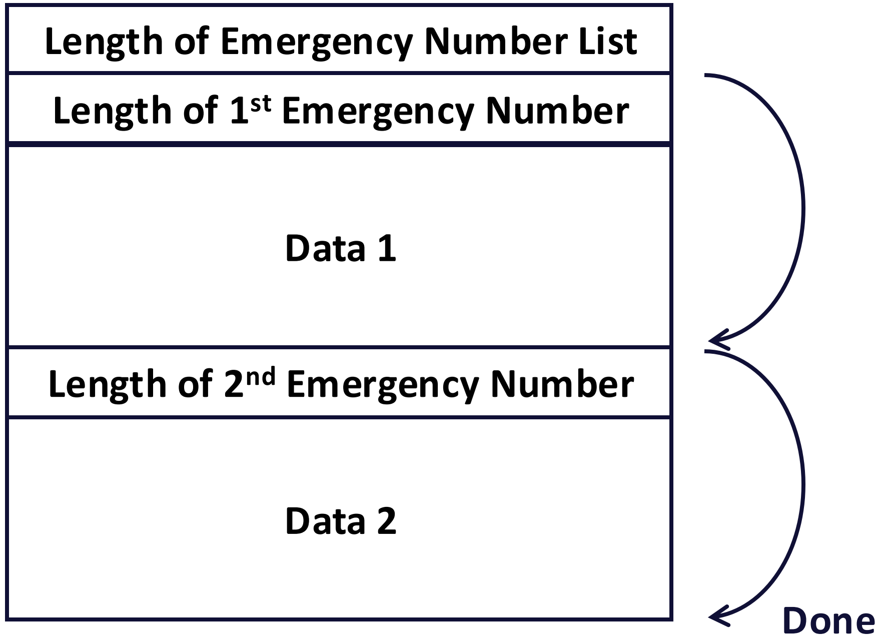
❖ Vulnerability Details

- Pre-AKA: Integer underflow in buffer copying mechanism
- Post-AKA: Infinite loop in Emergency Number List parsing



```
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);
    }
}
```

Infinity loop in decoding EmergencyNumberList



Length of Emergency Number List
Length of 1st Emergency Number
Data 1
Length of 2nd Emergency Number
Data 2
Done

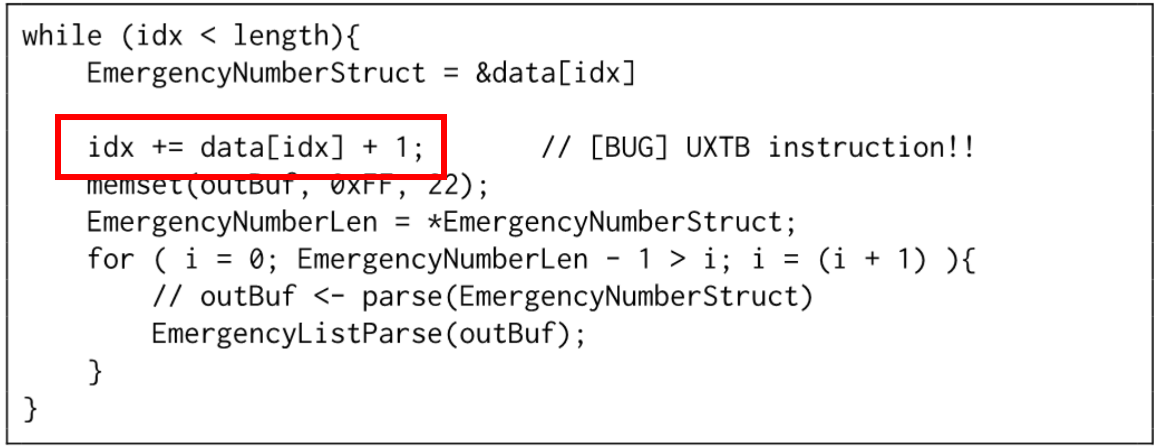
Evaluation [2] - Root Cause Analysis

❖ Proper emulation directly results to root cause analysis

- Instruction Trace Analysis (QEMU)
- Debugging (GDB)

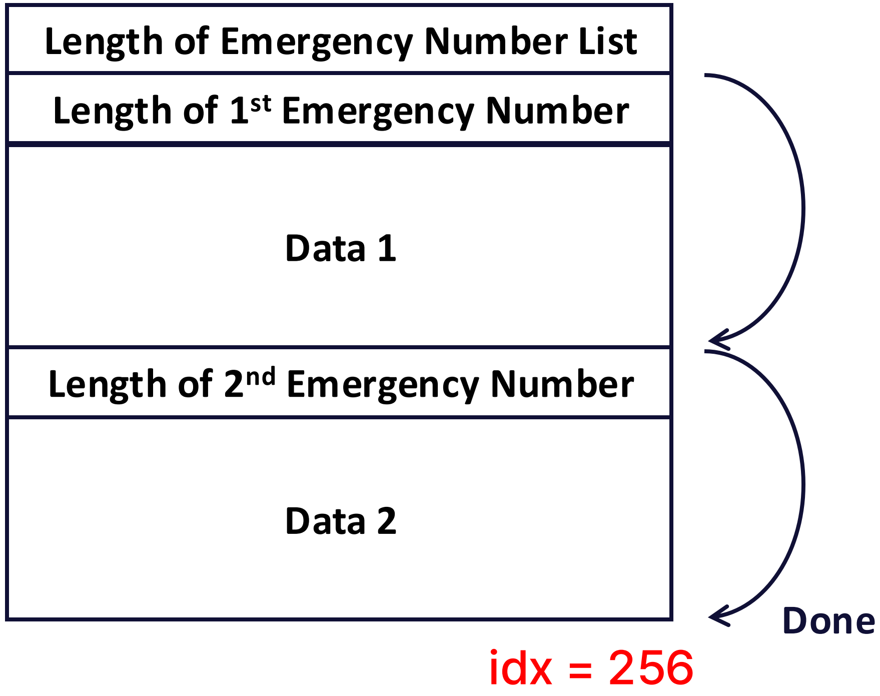
❖ Vulnerability Details

- Pre-AKA: Integer underflow in buffer copying mechanism
- Post-AKA: Infinite loop in Emergency Number List parsing



```
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);
    }
}
```

Infinity loop in decoding EmergencyNumberList



Length of Emergency Number List

Length of 1st Emergency Number

Data 1

Length of 2nd Emergency Number

Data 2

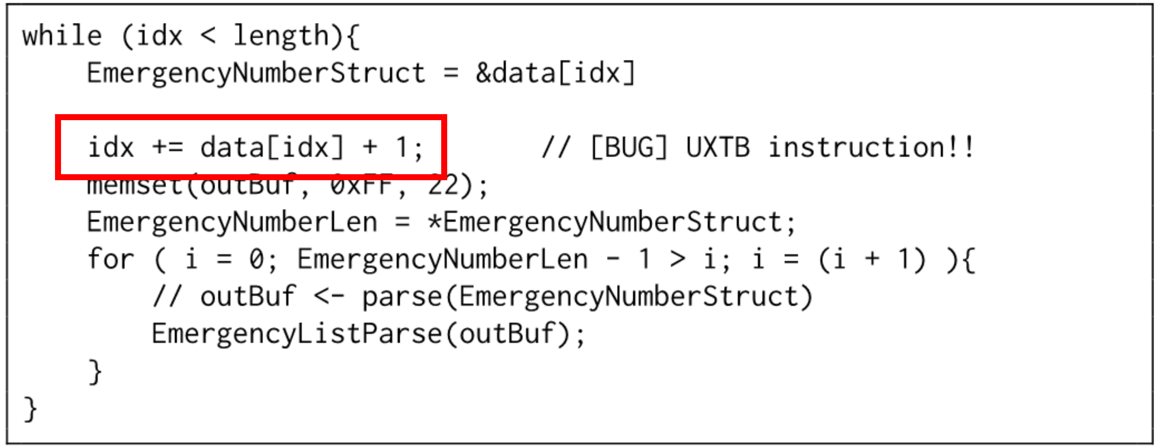
Done

idx = 256

Evaluation [2] - Root Cause Analysis

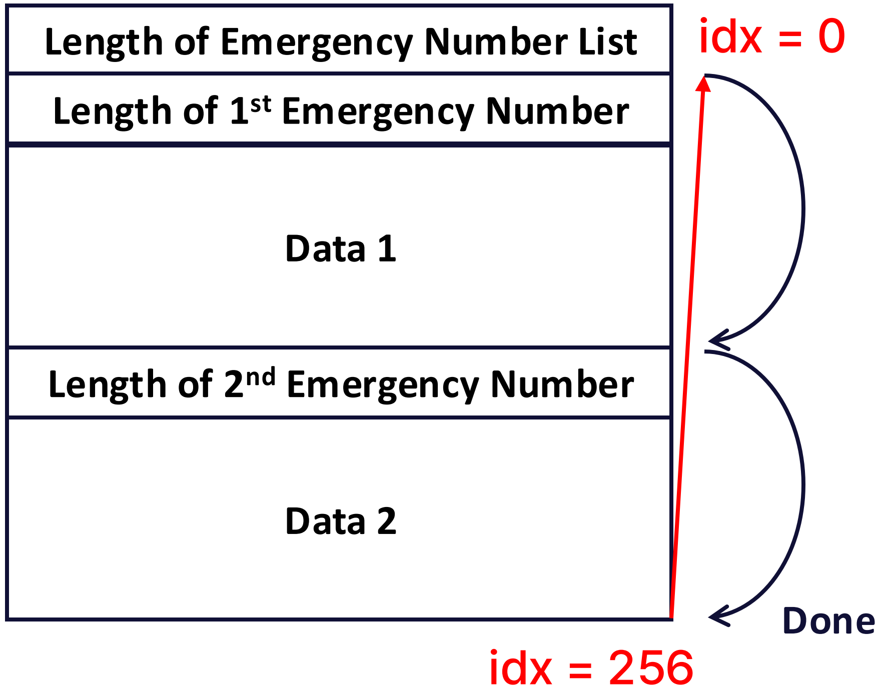
- ❖ Proper emulation directly results to root cause analysis
    - Instruction Trace Analysis (QEMU)
    - Debugging (GDB)

- ❖ Vulnerability Details
    - Pre-AKA: Integer underflow in buffer copying mechanism
    - Post-AKA: Infinite loop in Emergency Number List parsing



```
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);
    }
}
```

Infinity loop in decoding EmergencyNumberList



Length of Emergency Number List
idx = 0
Length of 1st Emergency Number
Data 1
Length of 2nd Emergency Number
Data 2
Done
idx = 256

Related Works

- Bridging the Gap between Emulation and Over-The-Air Testing for Cellular Baseband Firmware
  - Uses memory dumps for state restoration

IEEE S&P 2025

- Stateful Analysis and Fuzzing of Commercial Baseband Firmware
  - Uses symbolic analysis for state restoration
  - Extends FirmWire for newer Shannon baseband

Conclusion

- FirmState enables state-aware Shannon baseband emulation
  - Improves code coverage (x2.7) & fidelity
  - Enables previously unsupported NAS layer emulation
  - Discovered two 1day vulnerabilities

- Contact Information:
  - Suhwan Jeong (shjeong.b@enki.co.kr)
  - GitHub Repository: https://github.com/1nteger-c/FirmState

- ENKI WhiteHat (Offensive Security Research)
- KAIST SysSec Lab (Prof. Yongdae Kim)

GitHub Repo.