black hat
BRIEFINGS

AUGUST 6-7, 2025
MANDALAY BAY / LAS VEGAS

Uncovering 'NASty' 5G Baseband Vulnerabilities through Dependency-Aware Fuzzing

Ali Ranjbar & Tianchang Yang

Kai Tu, Saaman Khalilollahi, Kanika Gupta, Syed Rafiul Hussain

Introduction

Ali Ranjbar

- Research Assistant, The Pennsylvania State University
- Embedded systems, cellular security, reverse engineering, and fuzzing.
- `aranjbar.me`

PennState

Introduction

Tianchang Yang

- Research Assistant, The Pennsylvania State University
- Mobile network security, resiliency, and robustness: 5G, Open RAN, baseband (fuzzing, program analysis, ML)
- `tianchang-yang.github.io`

PennState

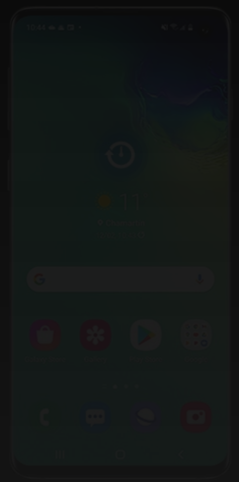
10:44
11°
Chamartín
12/02, 10:43
Galaxy Store
Gallery
Play Store
Google

09:30
5G
Coffee with Robin in 30 min
10.00 a.m. – 11.00 a.m. 20°C
Tue 19
20°

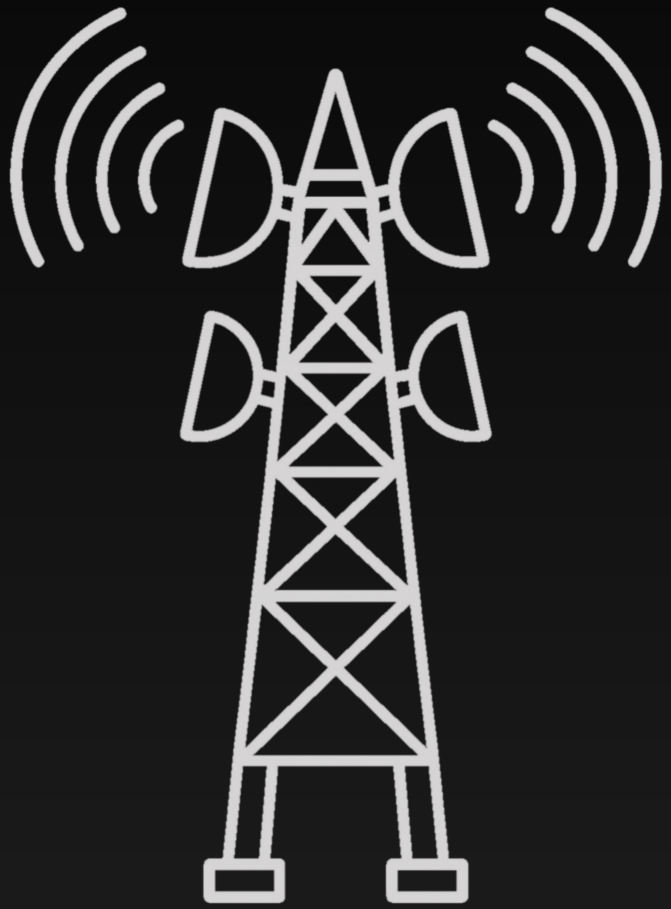
Cellular Network 101

Smartphone (UE)

Cellular Network 101

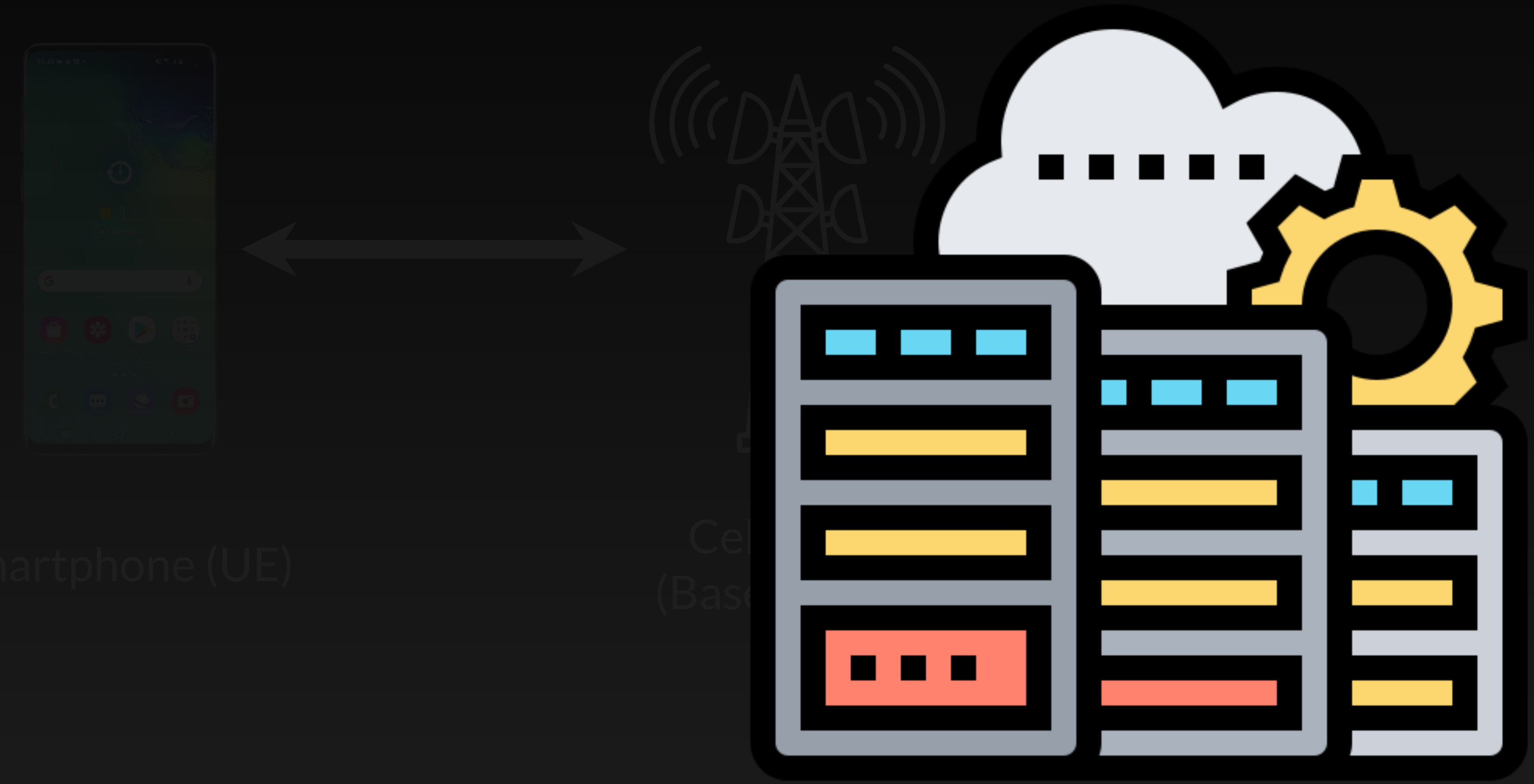


Smartphone (UE)



Cell tower
(Base station)

Cellular Network 101

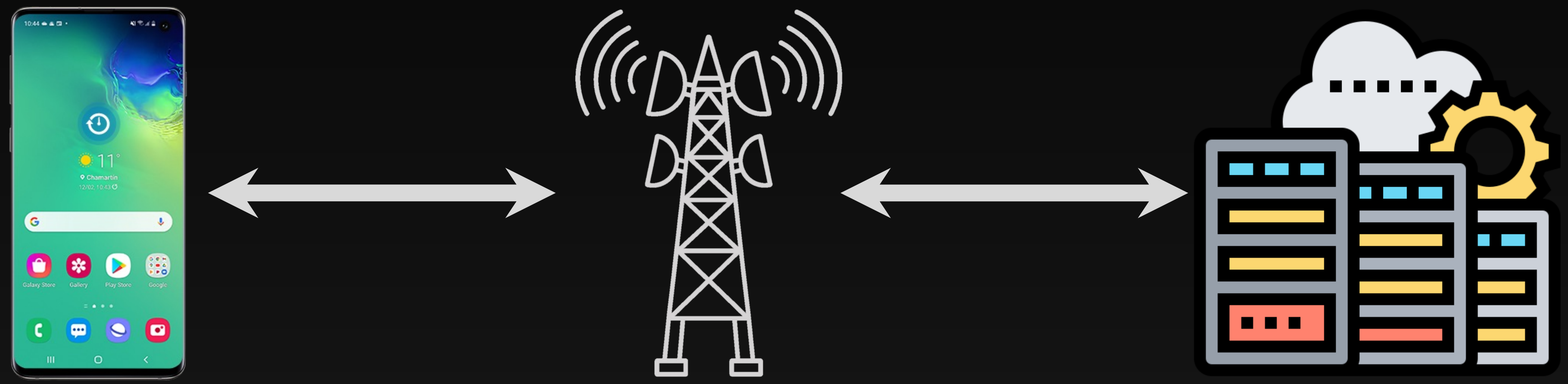


Smartphone (UE)

Ce
(Bas

Core network

Cellular Network 101

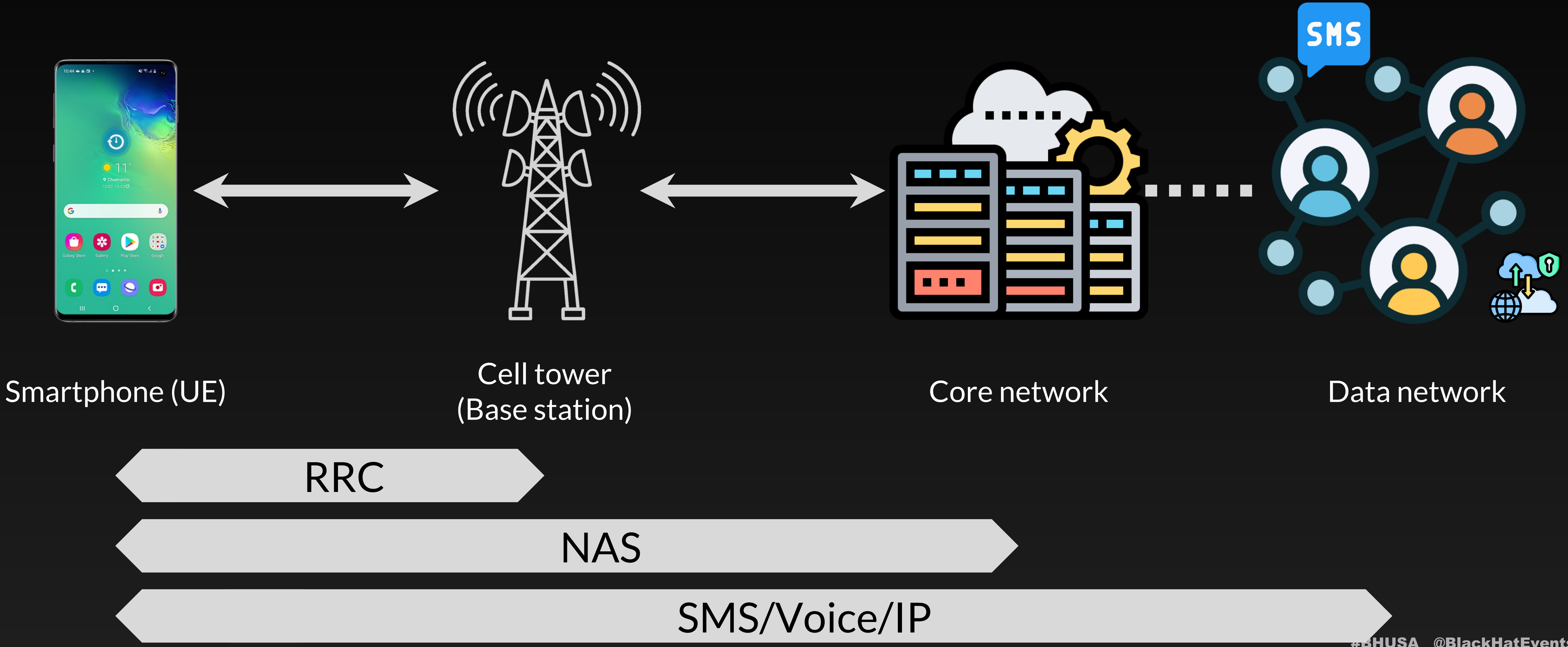


Smartphone (UE)

Cell tower
(Base station)

Core network

Cellular Network 101



Smartphone (UE)

Cell tower
(Base station)

Core network

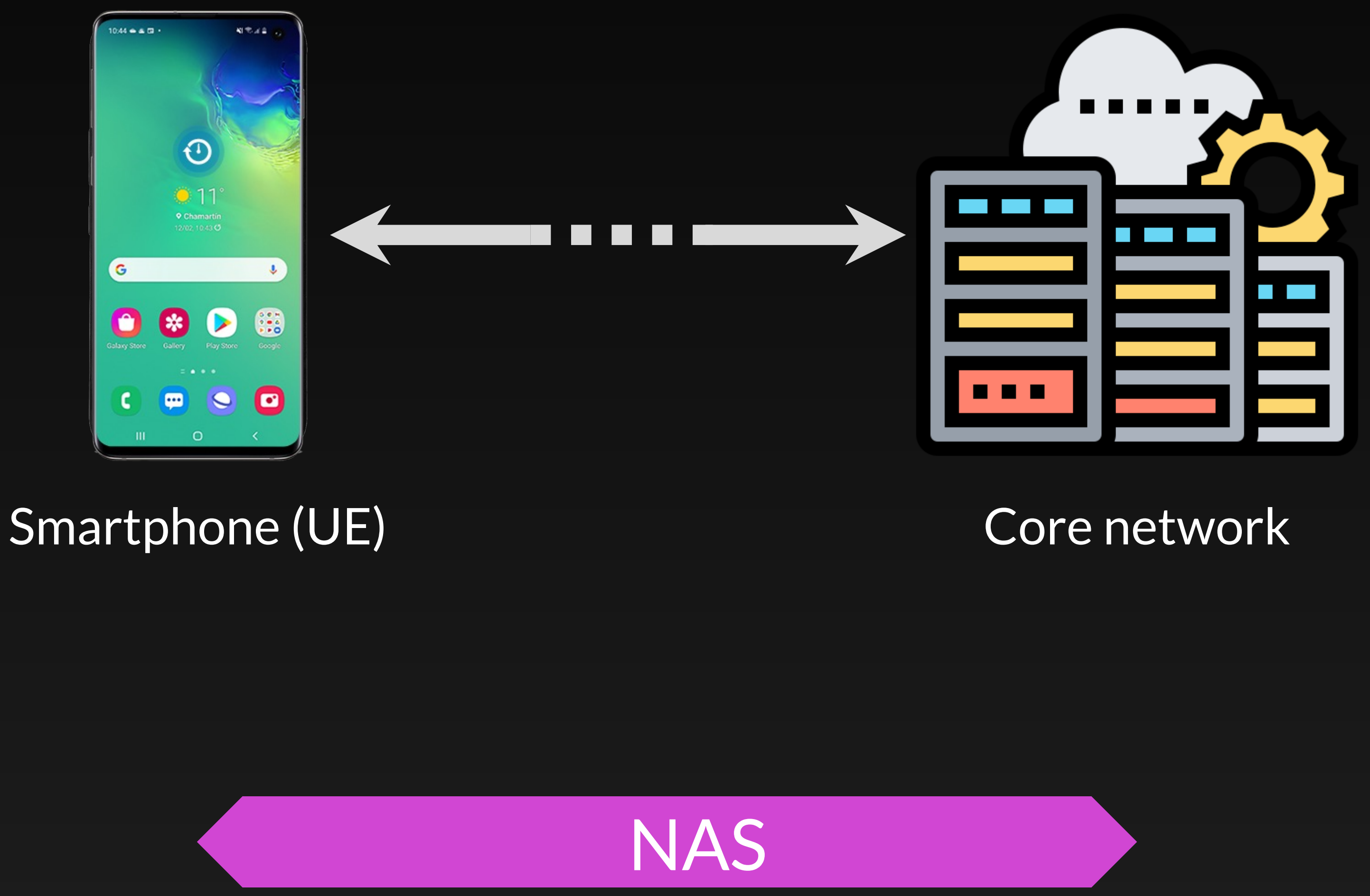
Data network

RRC

NAS

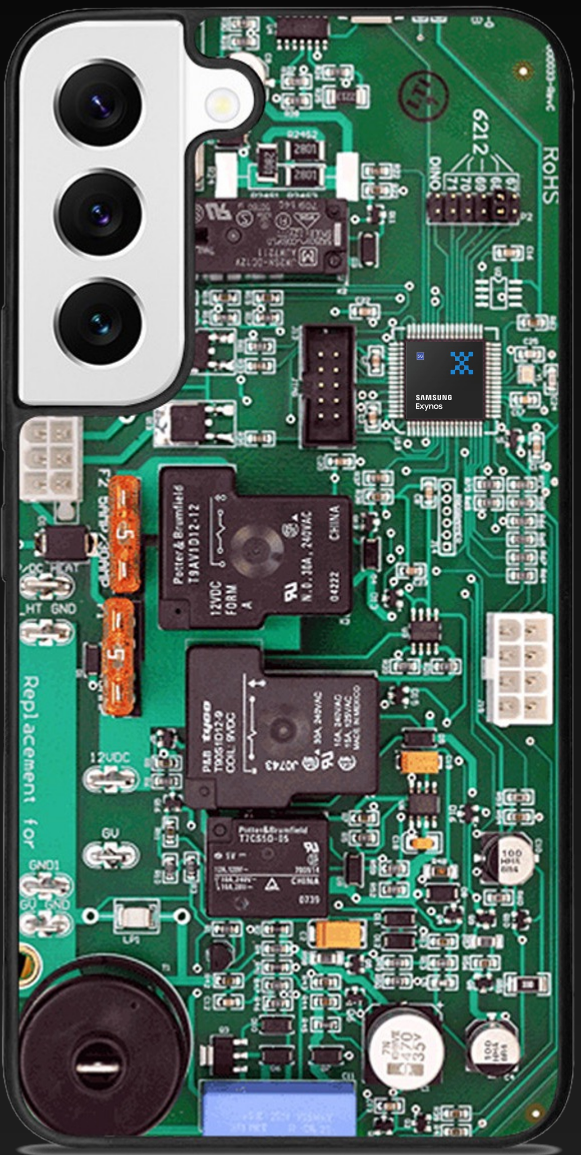
SMS/Voice/IP

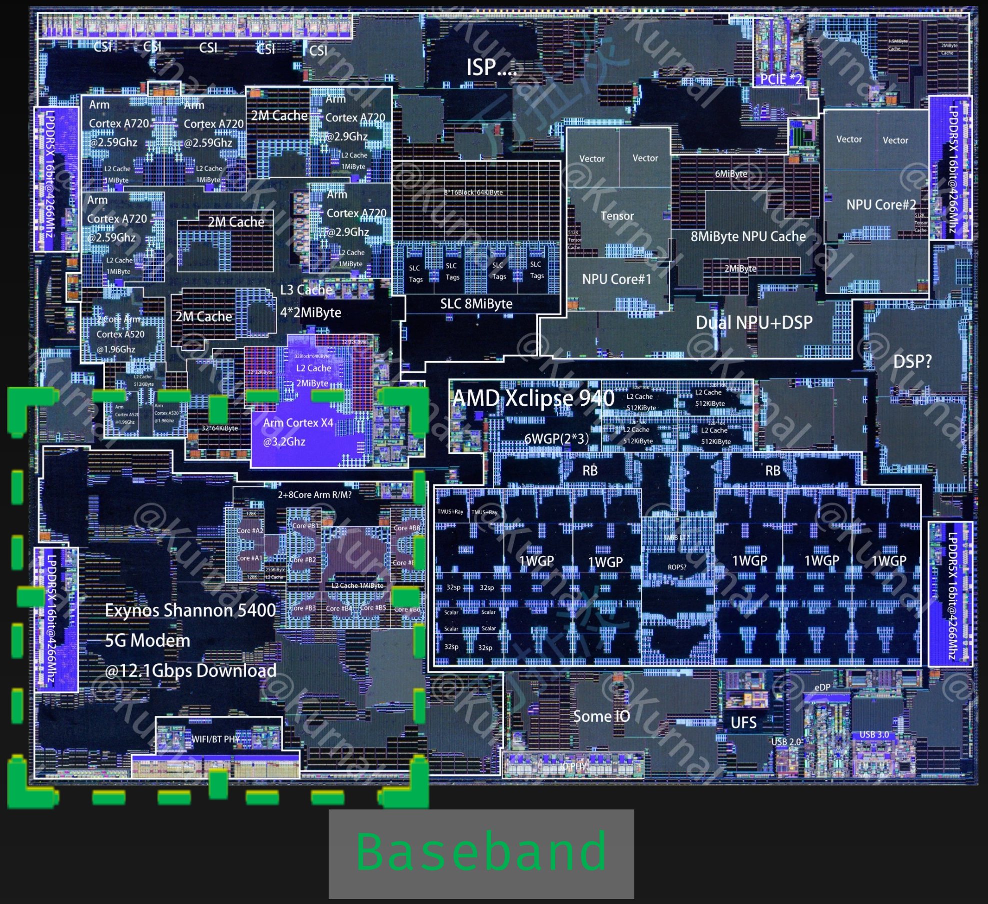
Non-Access Spectrum (NAS)



- NAS is mostly post-authentication
- NAS messages are encrypted and integrity protected – undertested
- Still results in issues not requiring operator keys to exploit

Baseband Overview





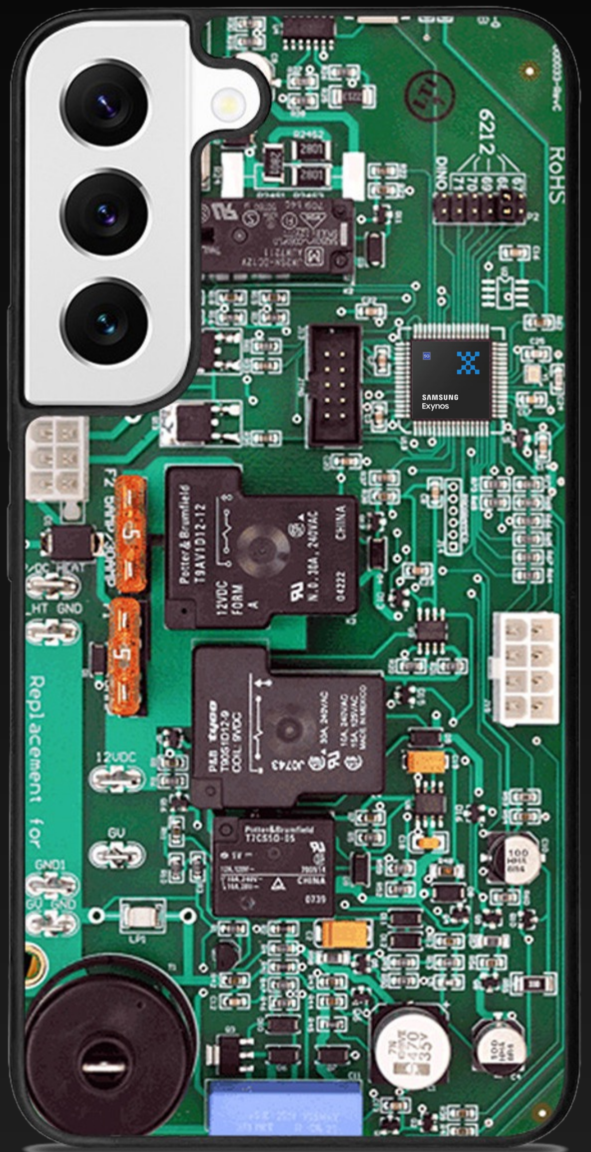
- Memory unsafe language
- Lack exploit protection

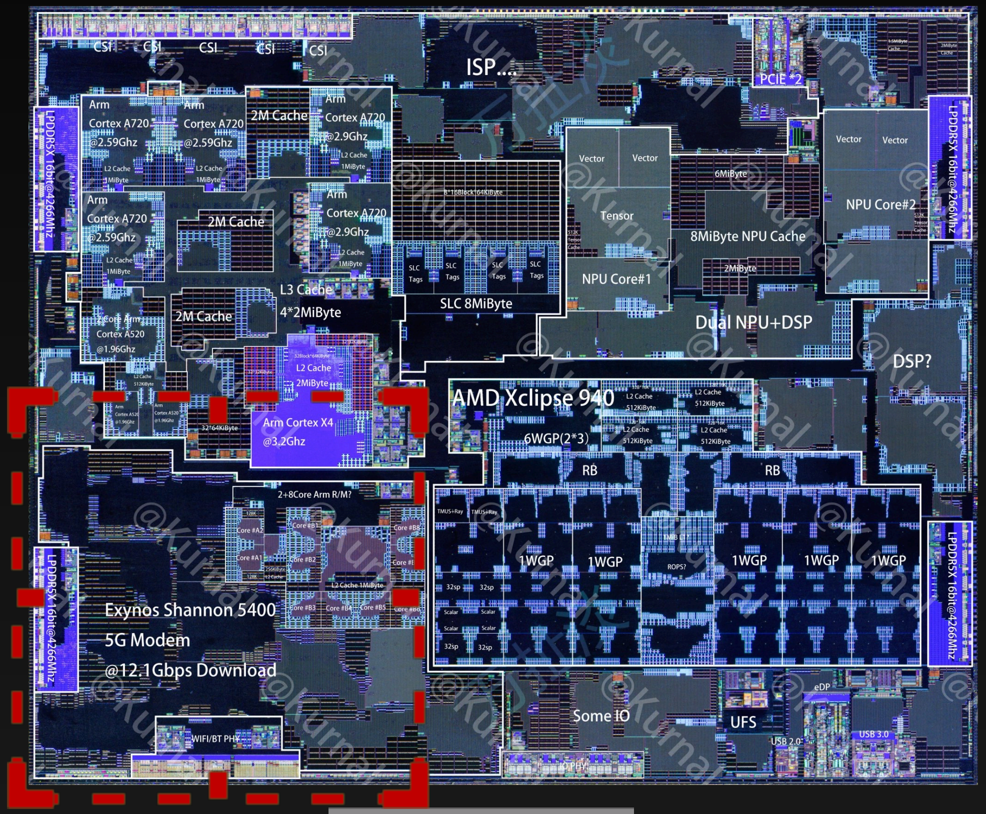
A	A	A	A	A	A	A	A	A	A

Buffer overflow

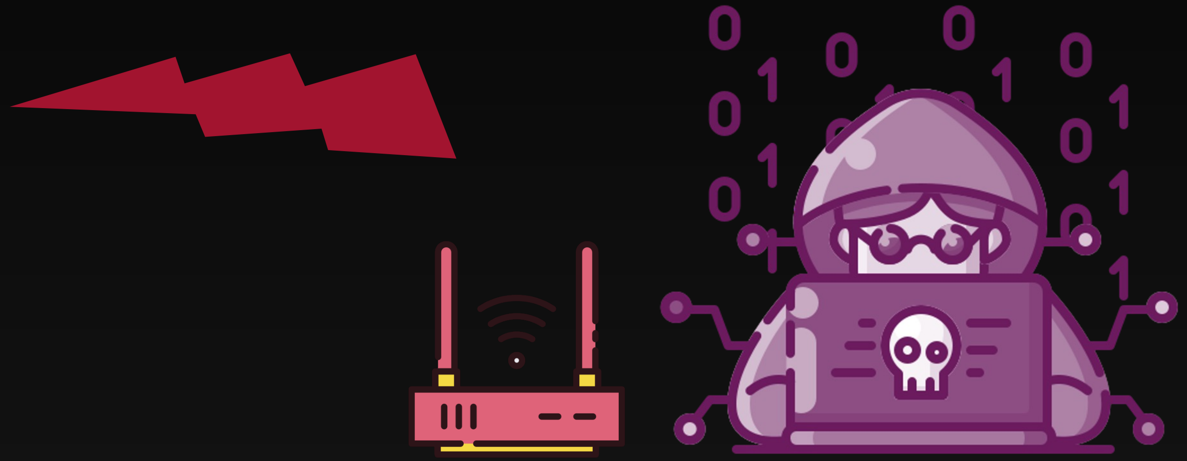
https://forums.anandtech.com/threads/samsung-exynos-soc-thread.2620862/

Baseband Overview





Baseband



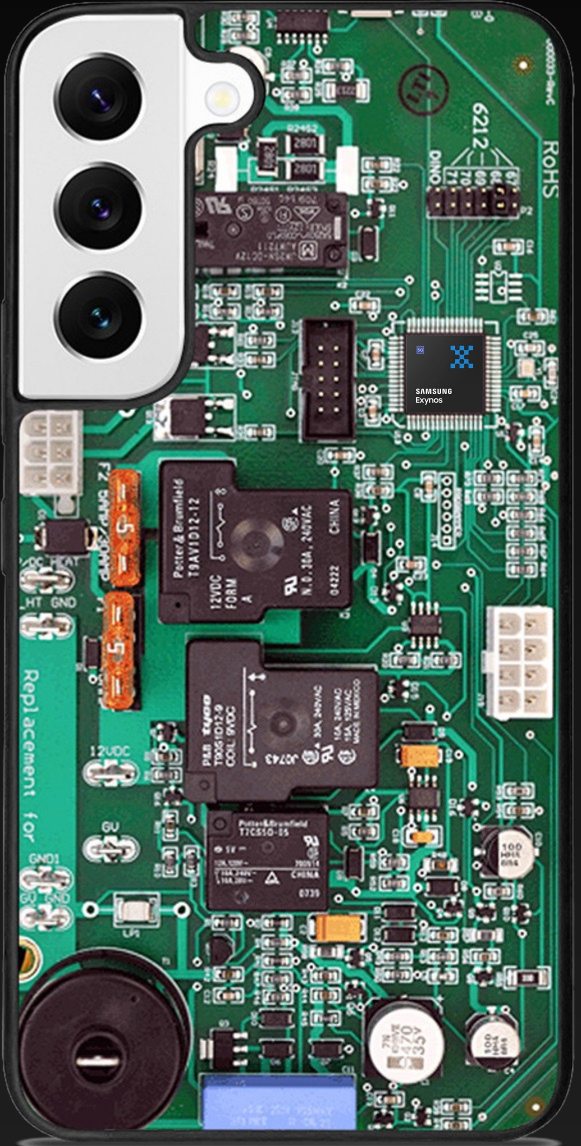
- Memory unsafe language
- Lack exploit protection

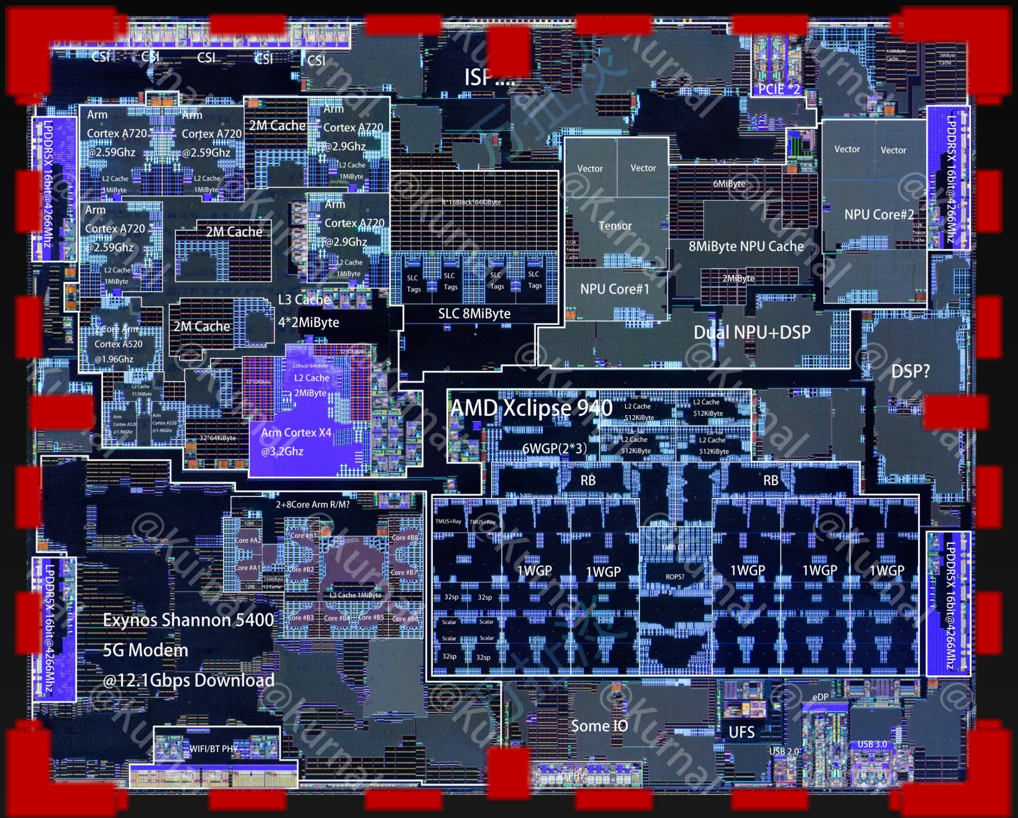
A	A	A	A	A	A	A	A	A	A

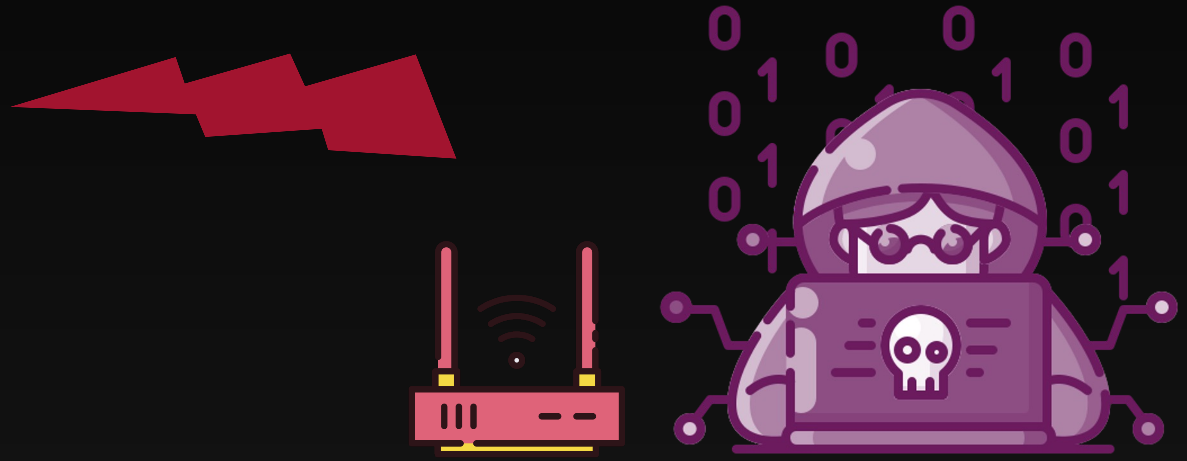
Buffer overflow

https://forums.anandtech.com/threads/samsung-exynos-soc-thread.2620862/

Baseband Overview







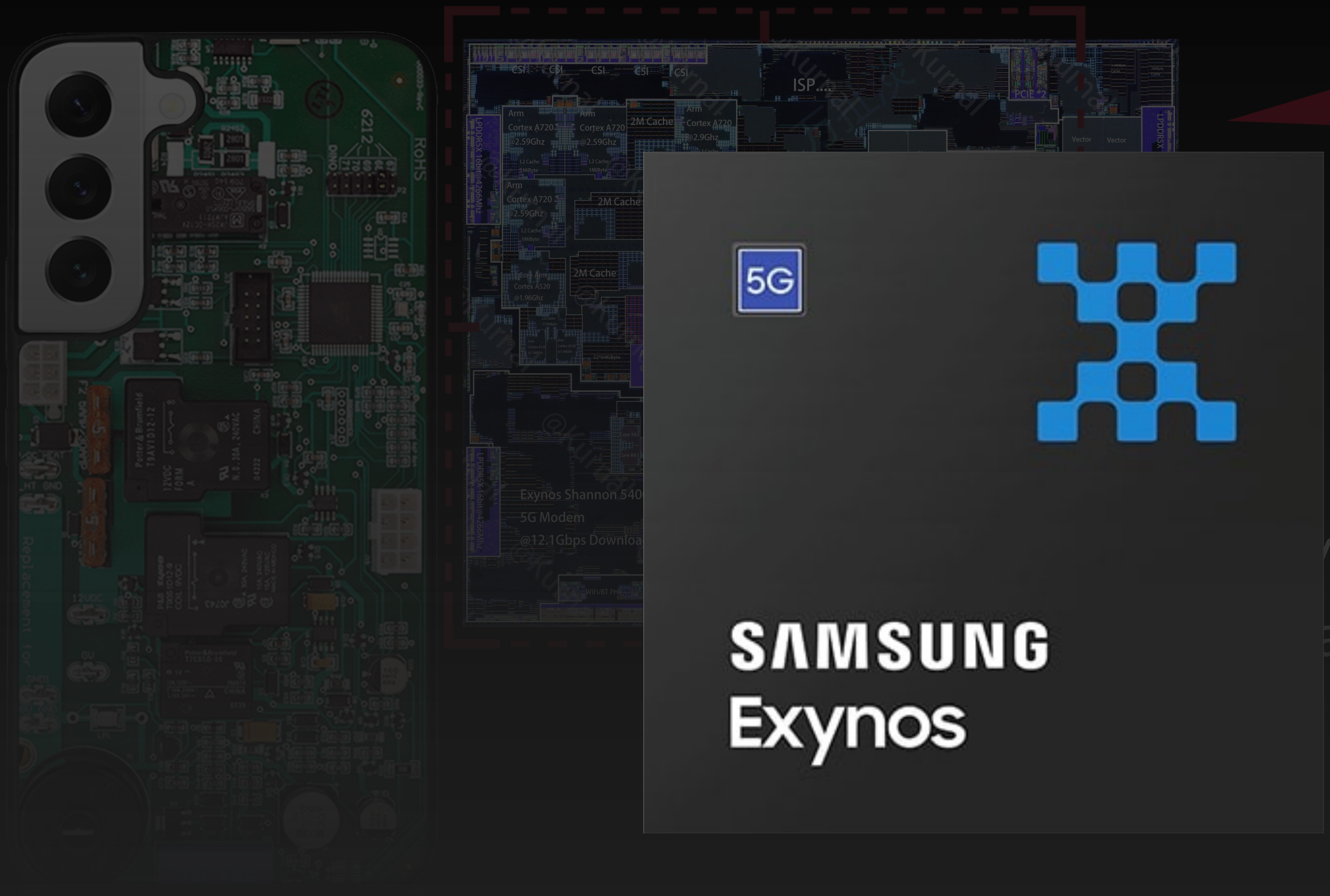
- Memory unsafe language
- Lack exploit protection

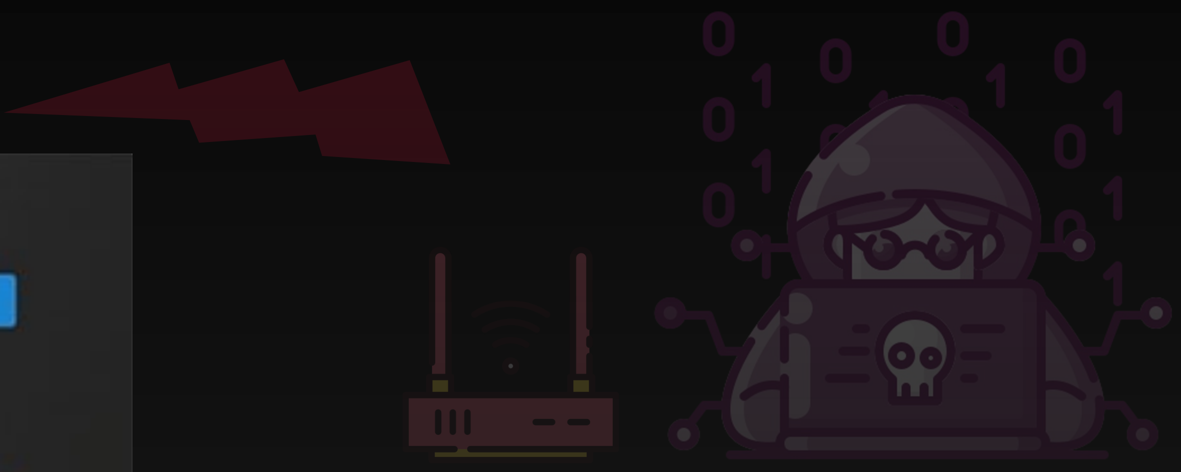
A	A	A	A	A	A	A	A	A	A

Buffer overflow

https://forums.anandtech.com/threads/samsung-exynos-soc-thread.2620862/

Baseband Overview





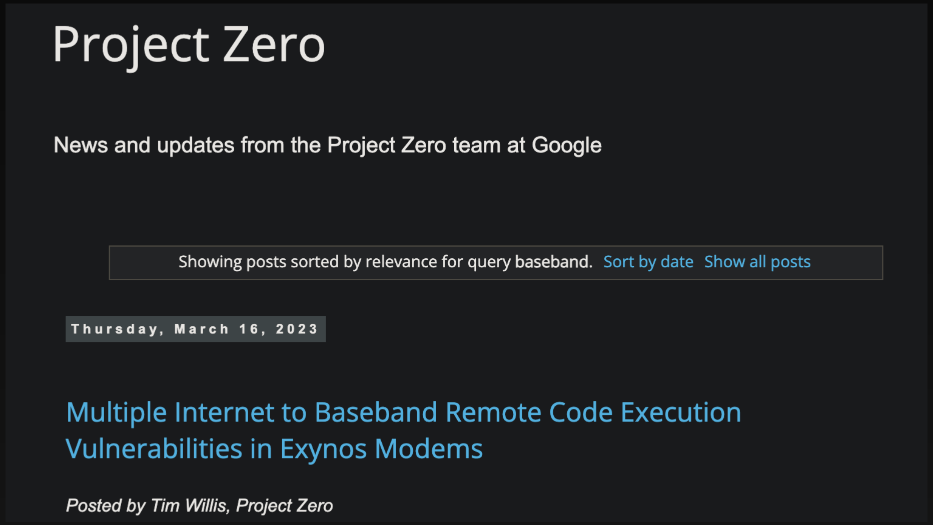
Memory unsafe language

Lack exploit protection

A A A A A A A A A

Buffer overflow

Baseband exploits in-the-wild



Project Zero

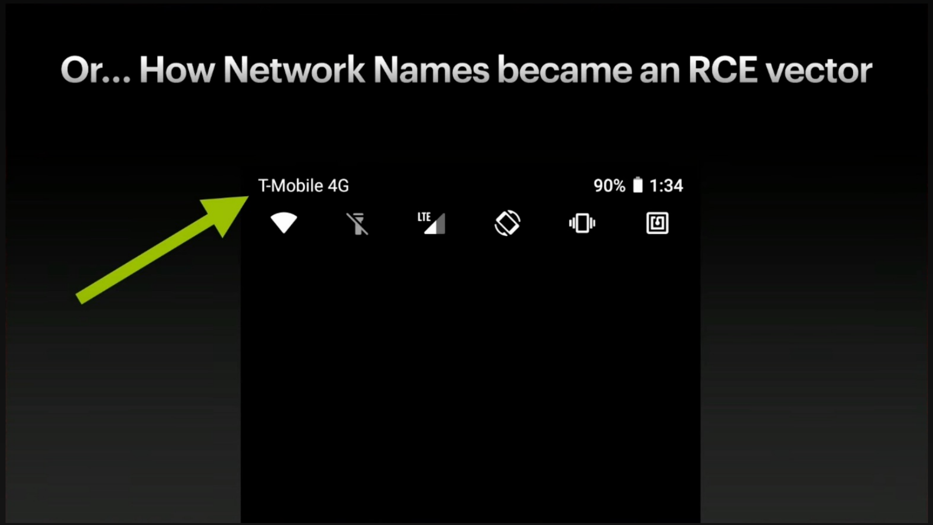
News and updates from the Project Zero team at Google

Showing posts sorted by relevance for query baseband. Sort by date Show all posts

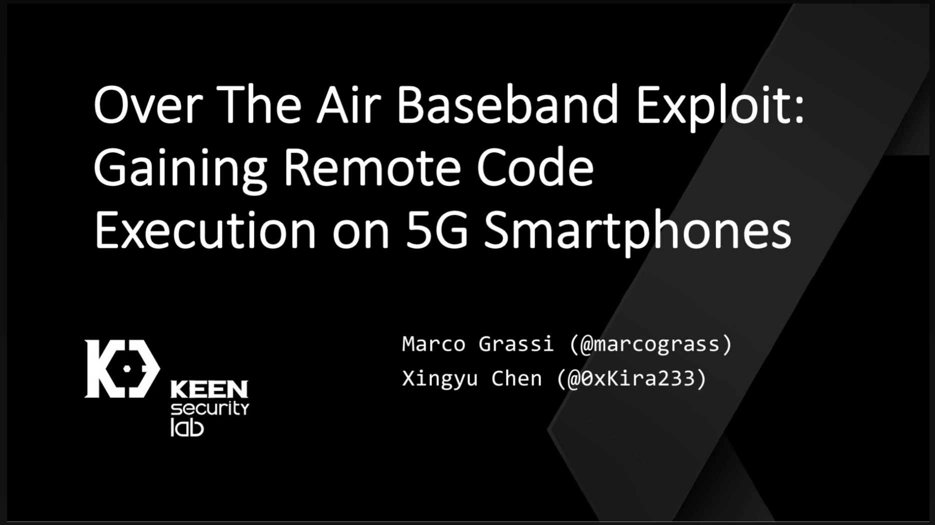
Thursday, March 16, 2023

Multiple Internet to Baseband Remote Code Execution Vulnerabilities in Exynos Modems

Posted by Tim Willis, Project Zero



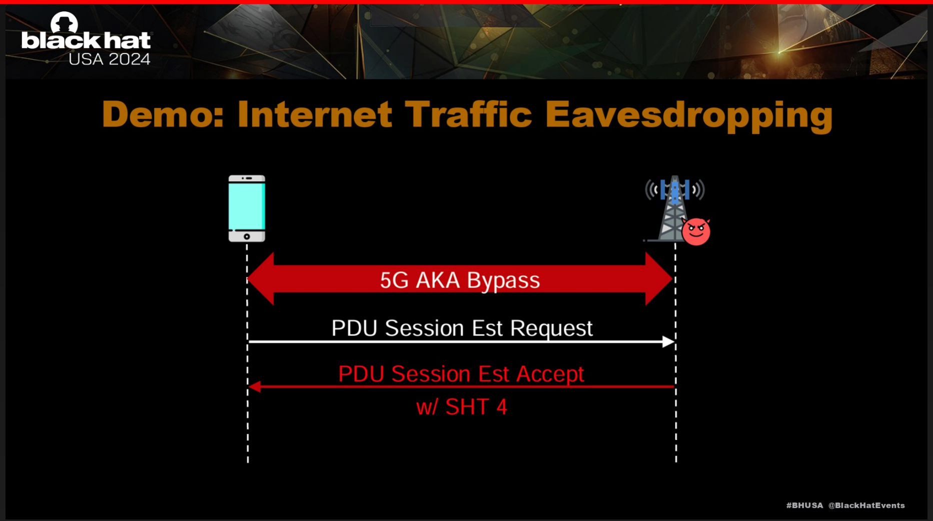
Or… How Network Names became an RCE vector



Over The Air Baseband Exploit: Gaining Remote Code Execution on 5G Smartphones

KEEN security lab

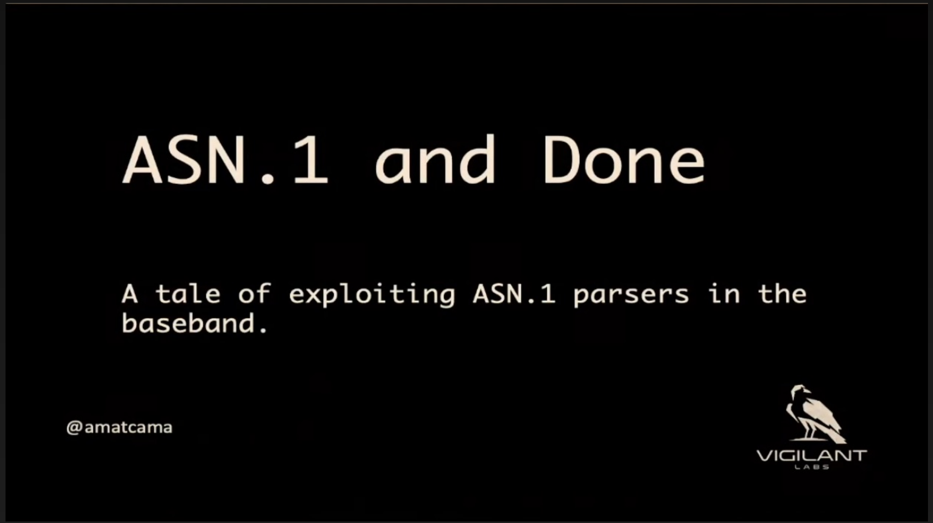
Marco Grassi (@marcograss)
Xingyu Chen (@0xKira233)



black hat USA 2024

Demo: Internet Traffic Eavesdropping

#BHUSA @BlackHatEvents

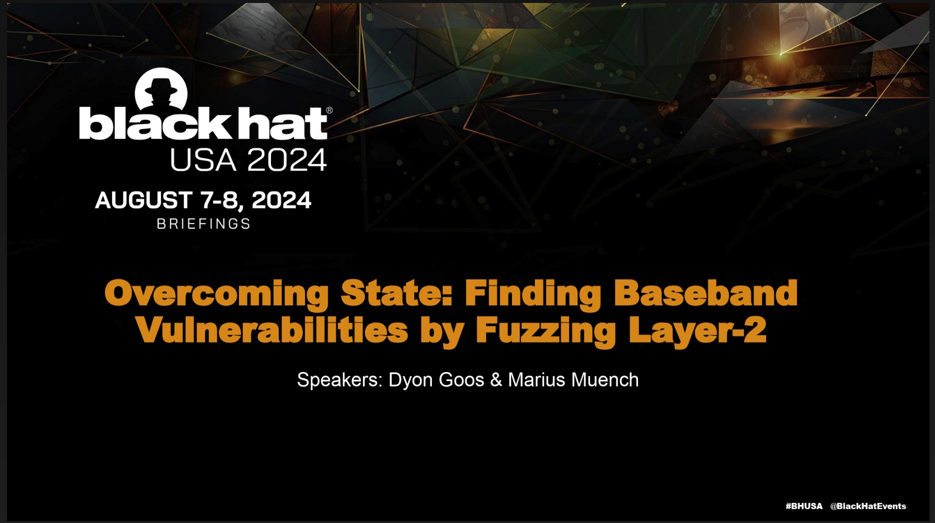


ASN.1 and Done

A tale of exploiting ASN.1 parsers in the baseband.

@amatcama

VIGILANT LABS



black hat USA 2024

AUGUST 7-8, 2024

BRIEFINGS

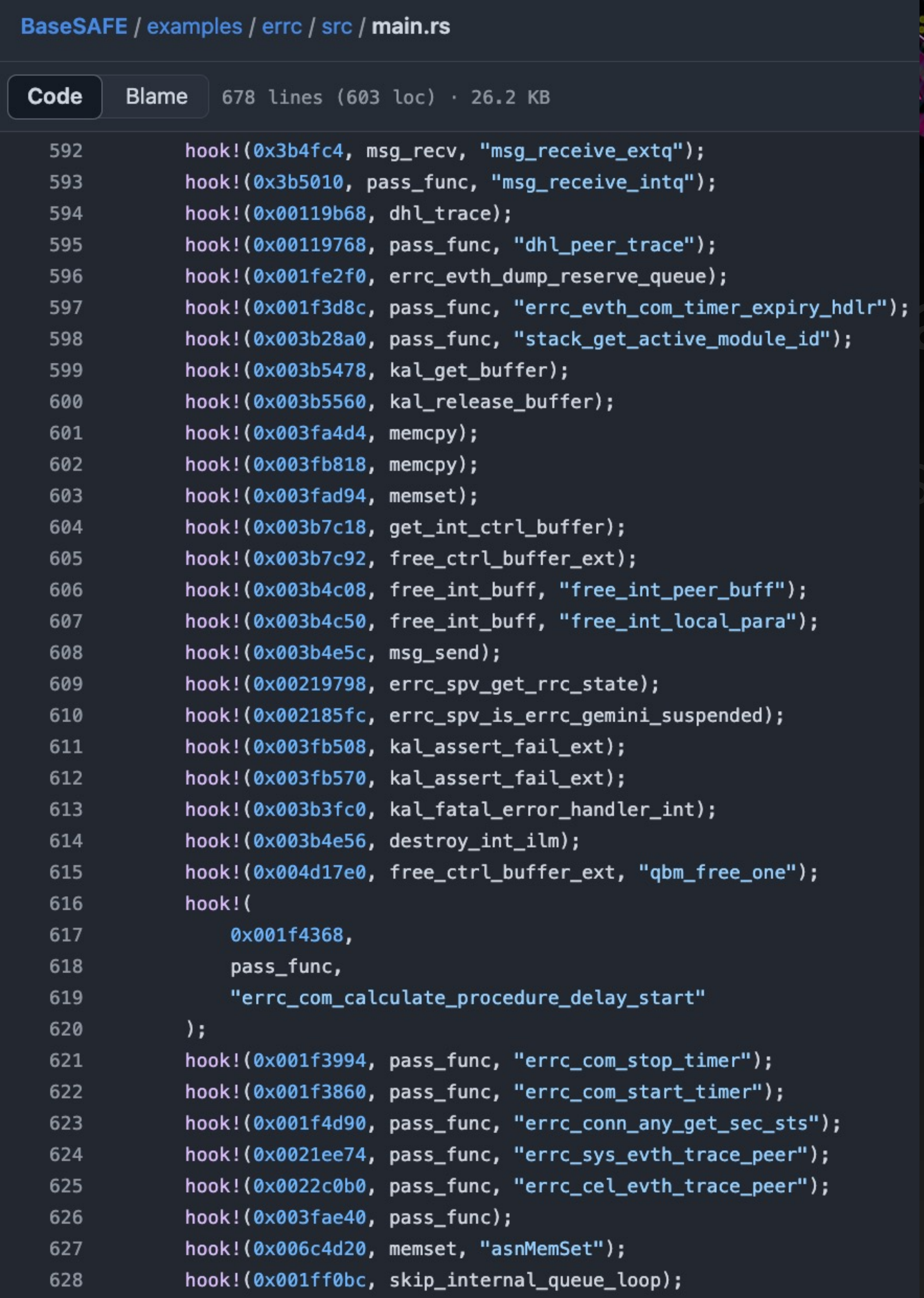
Overcoming State: Finding Baseband Vulnerabilities by Fuzzing Layer-2

Speakers: Dyon Goos & Marius Muench

#BHUSA @BlackHatEvents

From exploits to frameworks: Baseband research

- 2020: BaseSAFE: Baseband SAnitized Fuzzing through Emulation.



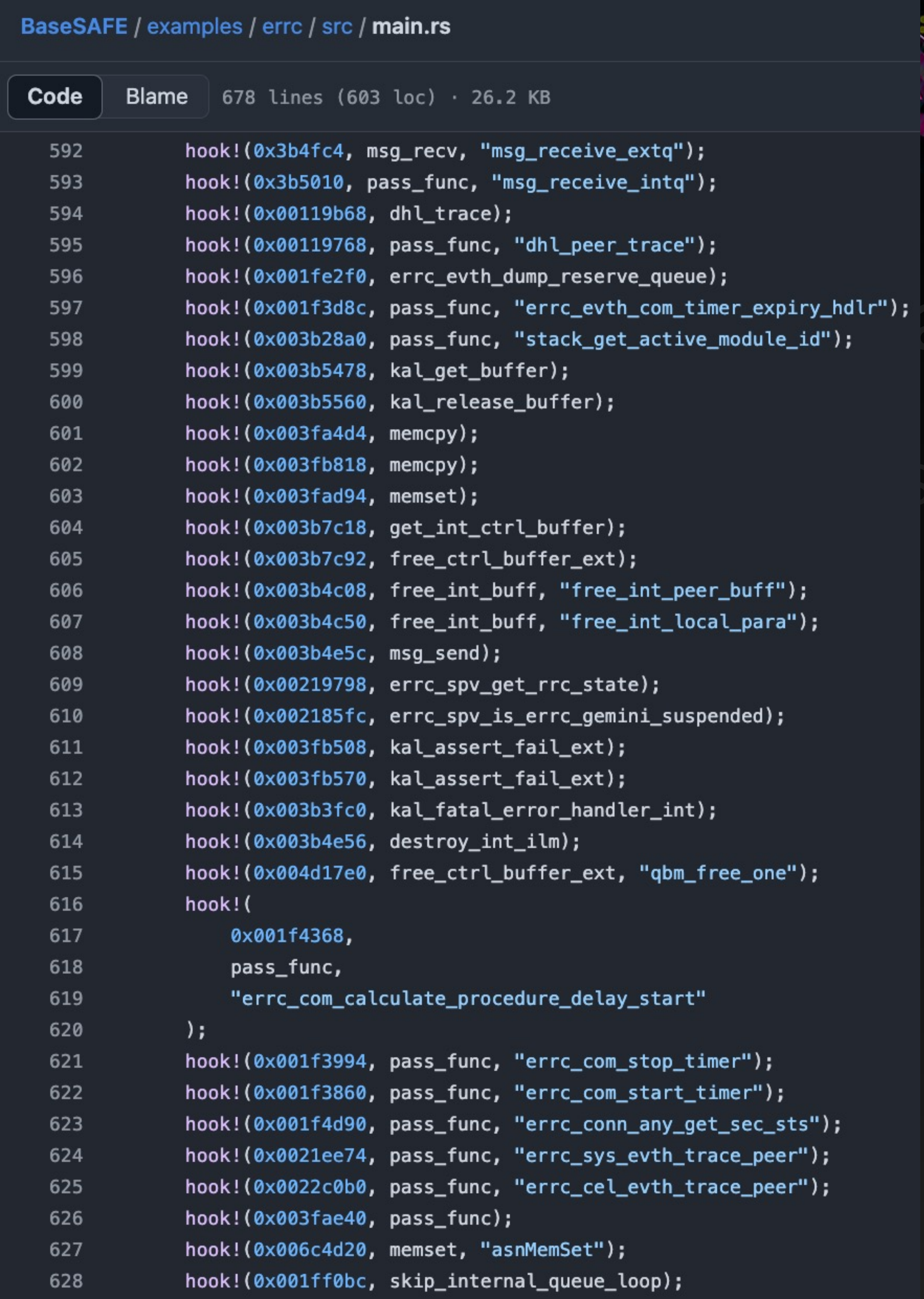
BaseSAFE / examples / errc / src / main.rs

678 lines (603 loc) · 26.2 KB

```
        hook!(0x3b4fc4, msg_recv, "msg_receive_extq");
        hook!(0x3b5010, pass_func, "msg_receive_intq");
        hook!(0x00119b68, dhl_trace);
        hook!(0x00119768, pass_func, "dhl_peer_trace");
        hook!(0x001fe2f0, errc_evth_dump_reserve_queue);
        hook!(0x001f3d8c, pass_func, "errc_evth_com_timer_expiry_hdlr");
        hook!(0x003b28a0, pass_func, "stack_get_active_module_id");
        hook!(0x003b5478, kal_get_buffer);
        hook!(0x003b5560, kal_release_buffer);
        hook!(0x003fa4d4, memcpy);
        hook!(0x003fb818, memcpy);
        hook!(0x003fad94, memset);
        hook!(0x003b7c18, get_int_ctrl_buffer);
        hook!(0x003b7c92, free_ctrl_buffer_ext);
        hook!(0x003b4c08, free_int_buff, "free_int_peer_buff");
        hook!(0x003b4c50, free_int_buff, "free_int_local_para");
        hook!(0x003b4e5c, msg_send);
        hook!(0x00219798, errc_spv_get_rrc_state);
        hook!(0x002185fc, errc_spv_is_errc_gemini_suspended);
        hook!(0x003fb508, kal_assert_fail_ext);
        hook!(0x003fb570, kal_assert_fail_ext);
        hook!(0x003b3fc0, kal_fatal_error_handler_int);
        hook!(0x003b4e56, destroy_int_ilm);
        hook!(0x004d17e0, free_ctrl_buffer_ext, "qbm_free_one");
        hook!(
            0x001f4368,
            pass_func,
            "errc_com_calculate_procedure_delay_start"
        );
        hook!(0x001f3994, pass_func, "errc_com_stop_timer");
        hook!(0x001f3860, pass_func, "errc_com_start_timer");
        hook!(0x001f4d90, pass_func, "errc_conn_any_get_sec_sts");
        hook!(0x0021ee74, pass_func, "errc_sys_evth_trace_peer");
        hook!(0x0022c0b0, pass_func, "errc_cel_evth_trace_peer");
        hook!(0x003fae40, pass_func);
        hook!(0x006c4d20, memset, "asnMemSet");
        hook!(0x001ff0bc, skip_internal_queue_loop);
```

ameworks: Baseband research

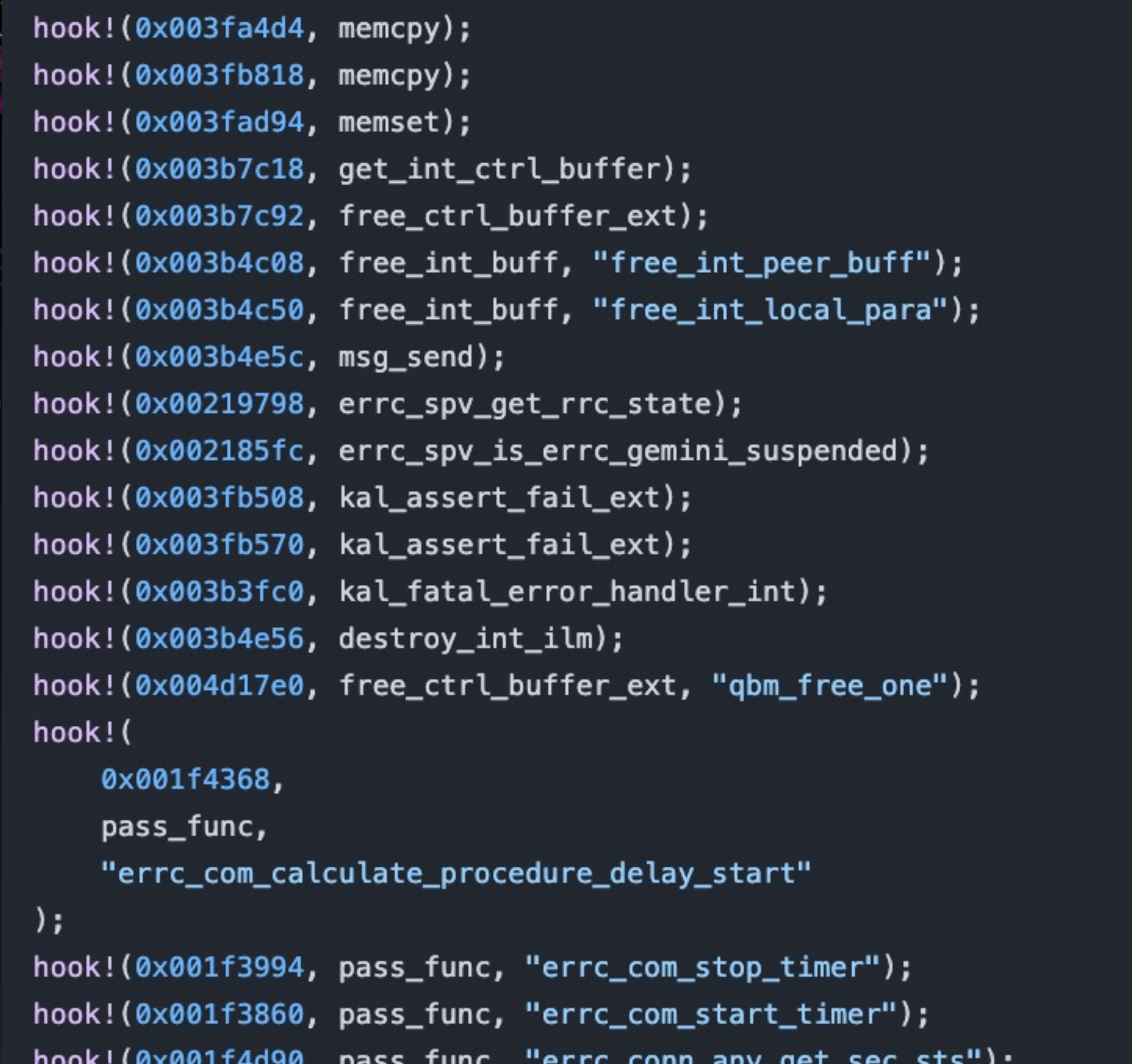
SAnitized Fuzzing through Emulation.



BaseSAFE / examples / errc / src / main.rs

678 lines (603 loc) · 26.2 KB

```
        hook!(0x3b4fc4, msg_recv, "msg_receive_extq");
        hook!(0x3b5010, pass_func, "msg_receive_intq");
        hook!(0x00119b68, dhl_trace);
        hook!(0x00119768, pass_func, "dhl_peer_trace");
        hook!(0x001fe2f0, errc_evth_dump_reserve_queue);
        hook!(0x001f3d8c, pass_func, "errc_evth_com_timer_expiry_hdlr");
        hook!(0x003b28a0, pass_func, "stack_get_active_module_id");
        hook!(0x003b5478, kal_get_buffer);
        hook!(0x003b5560, kal_release_buffer);
        hook!(0x003fa4d4, memcpy);
        hook!(0x003fb818, memcpy);
        hook!(0x003fad94, memset);
        hook!(0x003b7c18, get_int_ctrl_buffer);
        hook!(0x003b7c92, free_ctrl_buffer_ext);
        hook!(0x003b4c08, free_int_buff, "free_int_peer_buff");
        hook!(0x003b4c50, free_int_buff, "free_int_local_para");
        hook!(0x003b4e5c, msg_send);
        hook!(0x00219798, errc_spv_get_rrc_state);
        hook!(0x002185fc, errc_spv_is_errc_gemini_suspended);
        hook!(0x003fb508, kal_assert_fail_ext);
        hook!(0x003fb570, kal_assert_fail_ext);
        hook!(0x003b3fc0, kal_fatal_error_handler_int);
        hook!(0x003b4e56, destroy_int_ilm);
        hook!(0x004d17e0, free_ctrl_buffer_ext, "qbm_free_one");
        hook!(
            0x001f4368,
            pass_func,
            "errc_com_calculate_procedure_delay_start"
        );
        hook!(0x001f3994, pass_func, "errc_com_stop_timer");
        hook!(0x001f3860, pass_func, "errc_com_start_timer");
        hook!(0x001f4d90, pass_func, "errc_conn_any_get_sec_sts");
        hook!(0x0021ee74, pass_func, "errc_sys_evth_trace_peer");
        hook!(0x0022c0b0, pass_func, "errc_cel_evth_trace_peer");
        hook!(0x003fae40, pass_func);
        hook!(0x006c4d20, memset, "asnMemSet");
        hook!(0x001ff0bc, skip_internal_queue_loop);
```



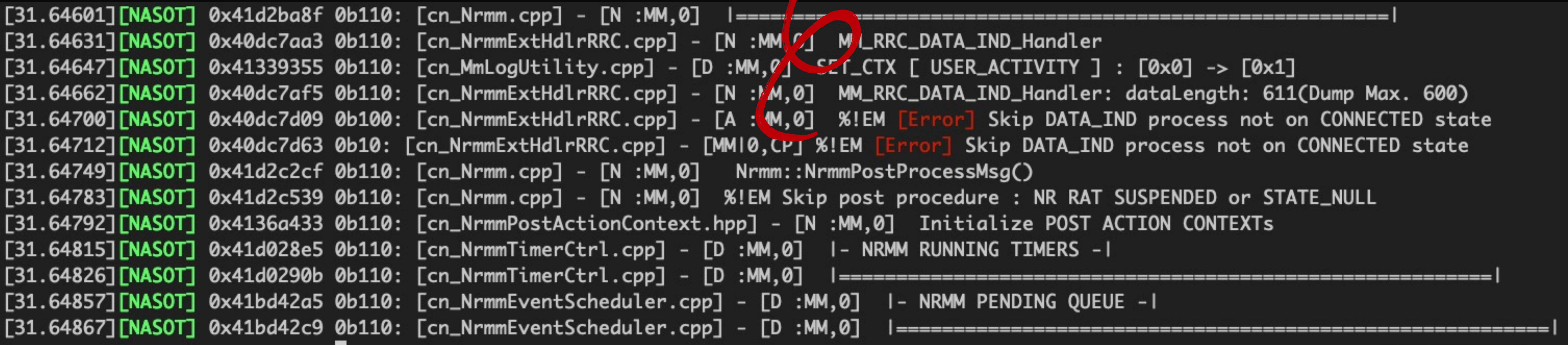
```
hook!(0x003fa4d4, memcpy);
hook!(0x003fb818, memcpy);
hook!(0x003fad94, memset);
hook!(0x003b7c18, get_int_ctrl_buffer);
hook!(0x003b7c92, free_ctrl_buffer_ext);
hook!(0x003b4c08, free_int_buff, "free_int_peer_buff");
hook!(0x003b4c50, free_int_buff, "free_int_local_para");
hook!(0x003b4e5c, msg_send);
hook!(0x00219798, errc_spv_get_rrc_state);
hook!(0x002185fc, errc_spv_is_errc_gemini_suspended);
hook!(0x003fb508, kal_assert_fail_ext);
hook!(0x003fb570, kal_assert_fail_ext);
hook!(0x003b3fc0, kal_fatal_error_handler_int);
hook!(0x003b4e56, destroy_int_ilm);
hook!(0x004d17e0, free_ctrl_buffer_ext, "qbm_free_one");
hook!(
    0x001f4368,
    pass_func,
    "errc_com_calculate_procedure_delay_start"
);
hook!(0x001f3994, pass_func, "errc_com_stop_timer");
hook!(0x001f3860, pass_func, "errc_com_start_timer");
hook!(0x001f4d90, pass_func, "errc_conn_any_get_sec_sts");
```

From exploits to frameworks: Baseband research

- 2020: BaseSAFE: Baseband SAnitized Fuzzing through Emulation.
- 2022: FirmWire: Transparent Dynamic Analysis for Cellular Baseband Firmware.
  - Supports Samsung Galaxy S7 – S10 (4G only!)
  - Requires manual harnessing to overcome complex baseband state.

Input rejected immediately!

From exploits to frameworks: Baseband research

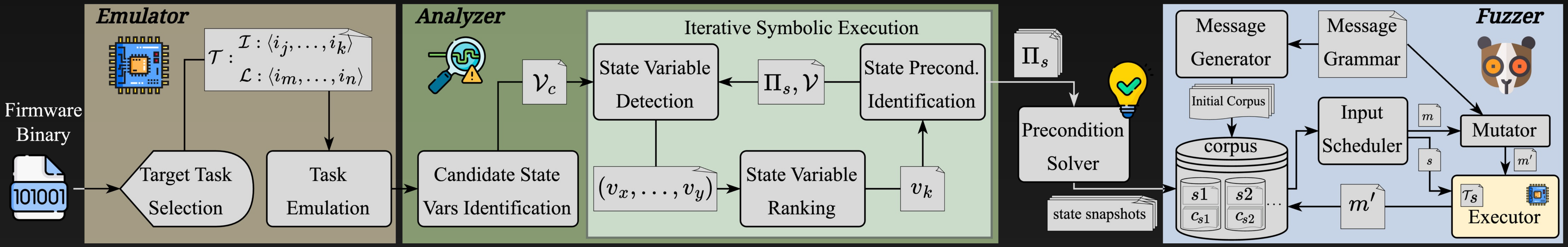


```
[31.64601][NASOT] 0x41d2ba8f 0b110: [cn_Nrmm.cpp] - [N :MM,0]  |==========================================================|
[31.64631][NASOT] 0x40dc7aa3 0b110: [cn_NrmmExtHdlrRRC.cpp] - [N :MM,0]  MM_RRC_DATA_IND_Handler
[31.64647][NASOT] 0x41339355 0b110: [cn_MmLogUtility.cpp] - [D :MM,0]  SET_CTX [ USER_ACTIVITY ] : [0x0] -> [0x1]
[31.64662][NASOT] 0x40dc7af5 0b110: [cn_NrmmExtHdlrRRC.cpp] - [N :MM,0]  MM_RRC_DATA_IND_Handler: dataLength: 611(Dump Max. 600)
[31.64700][NASOT] 0x40dc7d09 0b100: [cn_NrmmExtHdlrRRC.cpp] - [A :MM,0]  %!EM [Error] Skip DATA_IND process not on CONNECTED state
[31.64712][NASOT] 0x40dc7d63 0b10: [cn_NrmmExtHdlrRRC.cpp] - [MM|0,CP] %!EM [Error] Skip DATA_IND process not on CONNECTED state
[31.64749][NASOT] 0x41d2c2cf 0b110: [cn_Nrmm.cpp] - [N :MM,0]   Nrmm::NrmmPostProcessMsg()
[31.64783][NASOT] 0x41d2c539 0b110: [cn_Nrmm.cpp] - [N :MM,0]  %!EM Skip post procedure : NR RAT SUSPENDED or STATE_NULL
[31.64792][NASOT] 0x4136a433 0b110: [cn_NrmmPostActionContext.hpp] - [N :MM,0]  Initialize POST ACTION CONTEXTs
[31.64815][NASOT] 0x41d028e5 0b110: [cn_NrmmTimerCtrl.cpp] - [D :MM,0]  |- NRMM RUNNING TIMERS -|
[31.64826][NASOT] 0x41d0290b 0b110: [cn_NrmmTimerCtrl.cpp] - [D :MM,0]  |==========================================================|
[31.64857][NASOT] 0x41bd42a5 0b110: [cn_NrmmEventScheduler.cpp] - [D :MM,0]  |- NRMM PENDING QUEUE -|
[31.64867][NASOT] 0x41bd42c9 0b110: [cn_NrmmEventScheduler.cpp] - [D :MM,0]  |==========================================================|
```

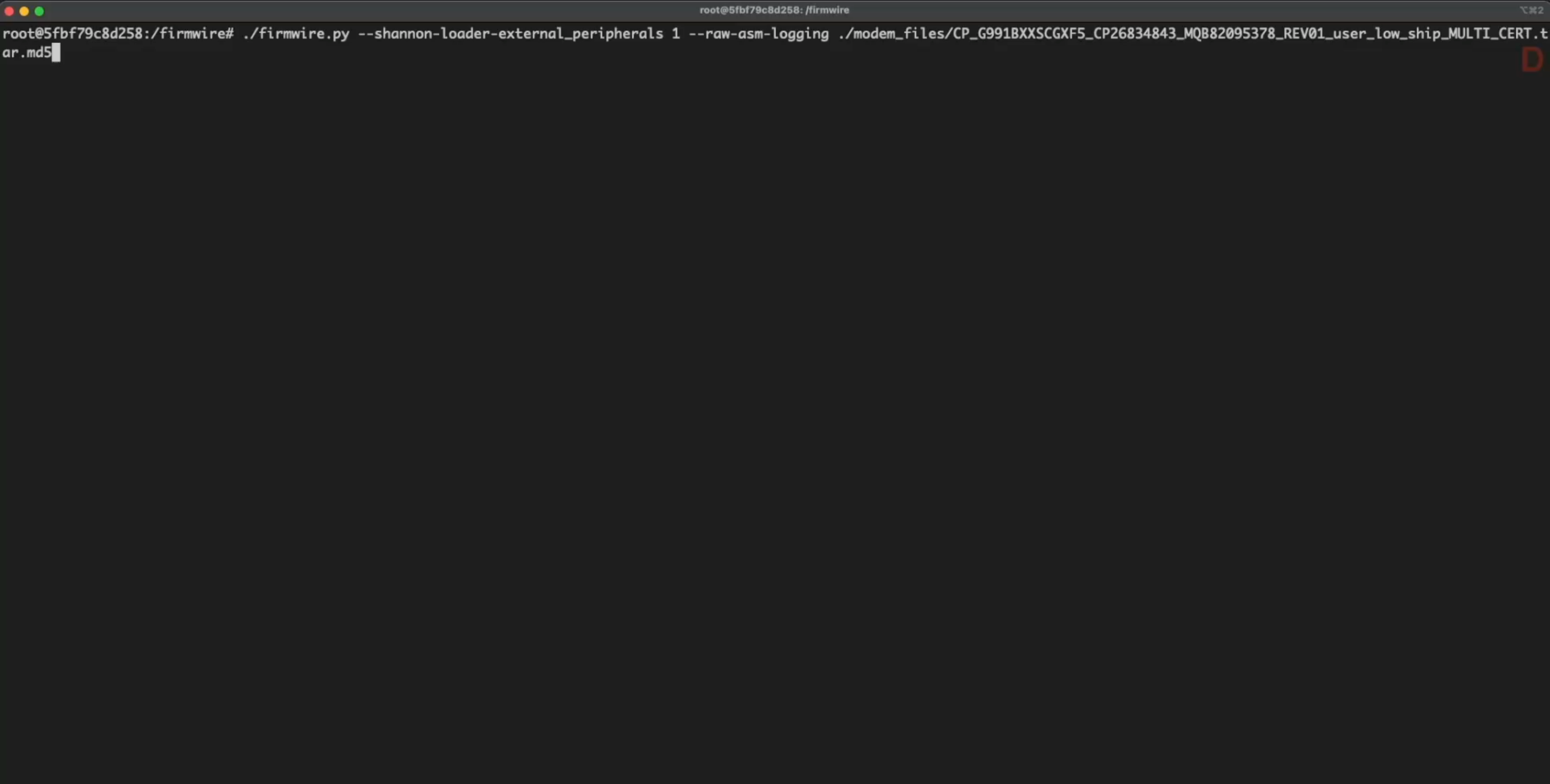
- Requires manual harnessing to overcome complex baseband state.

Introducing Loris

- The first framework to emulate Samsung’s 5G Shannon Basebands.
- Allows symbolic analysis of basebands using angr.
- Enables automated, state-aware fuzzing of modern 4G and 5G basebands.



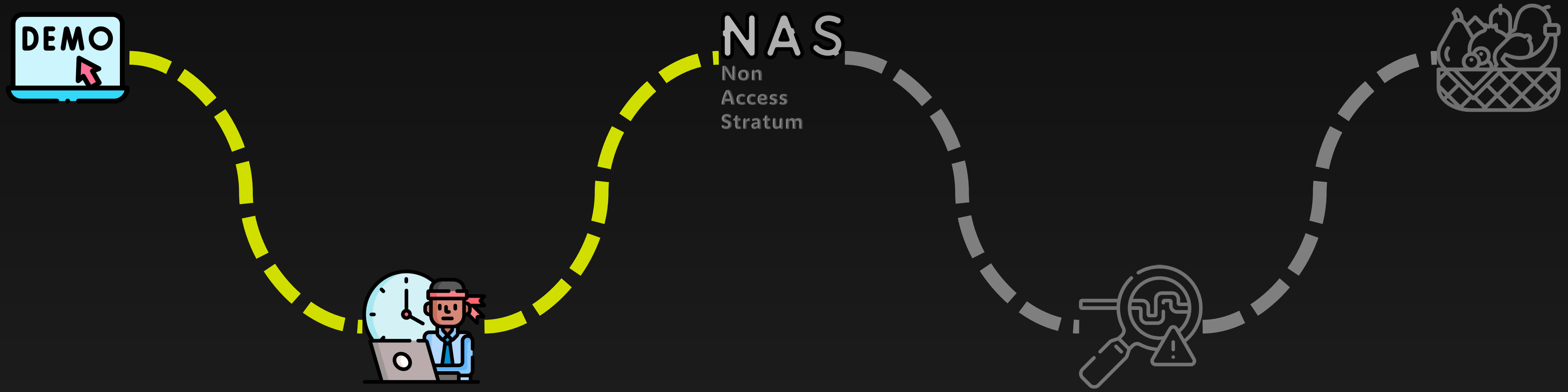
Quick Demo



```
root@5fbf79c8d258:/firmwire# ./firmwire.py --shannon-loader-external_peripherals 1 --raw-asm-logging ./modem_files/CP_G991BXXSCGXF5_CP26834843_MQB82095378_REV01_user_low_ship_MULTI_CERT.t
ar.md5
```

https://drive.google.com/file/d/1oGHDfGwSLMAEBtcRmbA9bRzWGDfFbK8j/view?usp=sharing

How did we get here?



In search of 5G NAS task: Task Metadata

- Samsung ShannonOS runs over 100 tasks:
  - Samsung Galaxy S21 contains 120+ tasks.
  - Google Pixel 6 contains 140+ tasks.
- The metadata can be found from function that creates ‘`mainTask`’.
- A global array stores the TaskStructs for all tasks.

```
0x00: TaskStruct
   ...
  0x10: Stackbase
   ...
  0x24: Name Pointer
   ...
  0x2c: Stacksize
  0x30: Main Function
  0x34: Pre-main Function
   ...
  0x140: Subtask
   ...
0x240: End of structure
```

MSD_OT	CDMOT	PBM	LteRrm	SNDCP	IMS_CC
L1HOT	L2HPDCPRXDELIV_OT	DS_PBM	LTE_L1LC	REG_SAP	LBS
L2LTXOT	L2LMACTXPROXY_OT	ATI	LteRrc	AS_SAP	SHM
L1OT	L2LMACTXENC_OT	MTI	LteRrc_DS	SMS_SAP	UL2CC
L2LRXOT	AsyncJob	SMS	LTEL2LRx	CC_SS_SAP	UL2DL
L2HTXOT	CLM	CC	LTEL2LTx	SIM_SAP	UL2UL
L2HRXOT	Acpm	MM	LTEL2HTx	DBG_SAP	UDATA
L3OT	Default	SM	LTEL2HRx	DS_REG_SAP	UBMCTask
NASOT	DM	SS	LTEL2MON	DS_AS_SAP	ephyFramework
QMOT	DM_TX	L1C	LTE_TLP	DS_SMS_SAP	syncTask
PSSOT	BDA	PPP	LTE_MTM	DS_CC_SS_SAP	recMailTask
PPPCOT	CIQD	GDA	NR_MTM	DS_SIM_SAP	sendMailTask
PPPTOT	CIQD_FE	CDH	LTE_DM	DS_DBG_SAP	BTL
PPPROT	Background	VSUP	EDFS	MMC	SecuCh
L2HPDCPTX_OT	TpTest	VCG	URRC	MMC_IF	Background1
L2LMACTX_OT	TaskReg	VCE	HSPA_CALIBRATION	SR_IF	Background2
L2HRLCRX_OT	DBGUNS	SAEL3	LLC	LTE_MMC_GL1	Background3
L2HRLCRETX_OT	DBGCMD	DS_SAEL3	GRR	USAT	UDC
L2LMACRX_OT	DBGCMD2	PDNMGR	RLC	DS_USAT	SHUB_MSG
L2HPDCPRX_OT	InitPacketHandler	SIM	GMAC	LTE_TCPIP	SSH
L2HRLCTX_OT	PacketHandler	DS_SIM	GLAPD	LTE_SISO_ASYNC	CPCOP

MSD_OT	CDMOT	PBM	LteRrm	SNDCP	IMS_CC
L1HOT	L2HPDCPRXDELIV_OT	DS_PBM	LTE_L1LC	REG_SAP	LBS
L2LTXOT	L2LMACTXPROXY_OT	ATI	LteRrc	AS_SAP	SHM
L1OT	L2LMACTXENC_OT	MTI	LteRrc_DS	SMS_SAP	UL2CC
L2LRXOT	AsyncJob	SMS	LTEL2LRx	CC_SS_SAP	UL2DL
L2HTXOT	CLM	CC	LTEL2LTx	SIM_SAP	UL2UL
L2HRXOT	Acpm	MM	LTEL2HTx	DBG_SAP	UDATA
L3OT	Default	SM	LTEL2HRx	DS_REG_SAP	UBMCTask
NASOT	DM	SS	LTEL2MON	DS_AS_SAP	ephyFramework
QMOT	DM_TX	L1C	LTE_TLP	DS_SMS_SAP	syncTask
PSSOT	BDA	PPP	LTE_MTM	DS_CC_SS_SAP	recMailTask
PPPCOT	CIQD	GDA	NR_MTM	DS_SIM_SAP	sendMailTask
PPPTOT	CIQD_FE	CDH	LTE_DM	DS_DBG_SAP	BTL
PPPROT	Background	VSUP	EDFS	MMC	SecuCh
L2HPDCPTX_OT	TpTest	VCG	URRC	MMC_IF	Background1
L2LMACTX_OT	TaskReg	VCE	HSPA_CALIBRATION	SR_IF	Background2
L2HRLCRX_OT	DBGUNS	SAEL3	LLC	LTE_MMC_GL1	Background3
L2HRLCRETX_OT	DBGCMD	DS_SAEL3	GRR	USAT	UDC
L2LMACRX_OT	DBGCMD2	PDNMGR	RLC	DS_USAT	SHUB_MSG
L2HPDCPRX_OT	InitPacketHandler	SIM	GMAC	LTE_TCPIP	SSH
L2HRLCTX_OT	PacketHandler	DS_SIM	GLAPD	LTE_SISO_ASYNC	CPCOP

MSD_OT	CDMOT	PBM	LteRrm	SNDCP	IMS_CC
L1HOT	L2HPDCPRXDELIV_OT	DS_PBM	LTE_L1LC	REG_SAP	LBS
L2LTXOT	L2LMACTXPROXY_OT	ATI	LteRrc	AS_SAP	SHM
L1OT	L2LMACTXENC_OT	MTI	LteRrc_DS	SMS_SAP	UL2CC
L2LRXOT	AsyncJob	SMS	LTEL2LRx	CC_SS_SAP	UL2DL
L2HTXOT	CLM	CC	LTEL2LTx	SIM_SAP	UL2UL
L2HRXOT	Acpm	MM	LTEL2HTx	DBG_SAP	UDATA
L3OT	Default	SM	LTEL2HRx	DS_REG_SAP	UBMCTask
NASOT	DM	SS	LTEL2MON	DS_AS_SAP	ephyFramework
QMOT	DM_TX	L1C	LTE_TLP	DS_SMS_SAP	syncTask
PSSOT	BDA	PPP	LTE_MTM	DS_CC_SS_SAP	recMailTask
PPPCOT	CIQD	GDA	NR_MTM	DS_SIM_SAP	sendMailTask
PPPTOT	CIQD_FE	CDH	LTE_DM	DS_DBG_SAP	BTL
PPPROT	Background	VSUP	EDFS	MMC	SecuCh
L2HPDCPTX_OT	TpTest	VCG	URRC	MMC_IF	Background1
L2LMACTX_OT	TaskReg	VCE	HSPA_CALIBRATION	SR_IF	Background2
L2HRLCRX_OT	DBGUNS	SAEL3	LLC	LTE_MMC_GL1	Background3
L2HRLCRETX_OT	DBGCMD	DS_SAEL3	GRR	USAT	UDC
L2LMACRX_OT	DBGCMD2	PDNMGR	RLC	DS_USAT	SHUB_MSG
L2HPDCPRX_OT	InitPacketHandler	SIM	GMAC	LTE_TCPIP	SSH
L2HRLCTX_OT	PacketHandler	DS_SIM	GLAPD	LTE_SISO_ASYNC	CPCOP

MSD_OT	CDMOT	PBM	LteRrm	SNDCP	IMS_CC
L1HOT	L2HPDCPRXDELIV_OT	DS_PBM	LTE_L1LC	REG_SAP	LBS
L2LTXOT	L2LMACTXPROXY_OT	ATI	LteRrc	AS_SAP	SHM
L1OT	L2LMACTXENC_OT	MTI	LteRrc_DS	SMS_SAP	UL2CC
L2LRXOT	AsyncJob	SMS	LTEL2LRx	CC_SS_SAP	UL2DL
L2HTXOT	CLM	CC	LTEL2LTx	SIM_SAP	UL2UL
L2HRXOT	Acpm	MM	LTEL2HTx	DBG_SAP	UDATA
L3OT	Default	SM	LTEL2HRx	DS_REG_SAP	UBMCTask
NASOT	DM	SS	LTEL2MON	DS_AS_SAP	ephyFramework
QMOT	DM_TX	L1C	LTE_TLP	DS_SMS_SAP	syncTask
PSSOT	BDA	PPP	LTE_MTM	DS_CC_SS_SAP	recMailTask
PPPCOT	CIQD	GDA	NR_MTM	DS_SIM_SAP	sendMailTask
PPPTOT	CIQD_FE	CDH	LTE_DM	DS_DBG_SAP	BTL
PPPROT	Background	VSUP	EDFS	MMC	SecuCh
L2HPDCPTX_OT	TpTest	VCG	URRC	MMC_IF	Background1
L2LMACTX_OT	TaskReg	VCE	HSPA_CALIBRATION	SR_IF	Background2
L2HRLCRX_OT	DBGUNS	SAEL3	LLC	LTE_MMC_GL1	Background3
L2HRLCRETX_OT	DBGCMD	DS_SAEL3	GRR	USAT	UDC
L2LMACRX_OT	DBGCMD2	PDNMGR	RLC	DS_USAT	SHUB_MSG
L2HPDCPRX_OT	InitPacketHandler	SIM	GMAC	LTE_TCPIP	SSH
L2HRLCTX_OT	PacketHandler	DS_SIM	GLAPD	LTE_SISO_ASYNC	CPCOP

NASOT

black hat
BRIEFINGS

Building an emulator: From Cortex-R to Cortex-A

Memory layout

- Cortex-A lacks an MPU; requires extracting MMU tables for memory mappings.

Set TTBR0 co-processor register

```
ldr   r0, =page_table_address
mcr   p15, 0x0, r0, cr2, cr0, 0x0
```

Building an emulator: From Cortex-R to Cortex-A

Memory layout

- Cortex-A lacks an MPU; requires extracting MMU tables for memory mappings.

```
r0 = virtual address
r1 = physical address | perm | attr
str  r1, [page_table_address, r0, lsr #18]
```

Upper bits of virtual address as offset

Building an emulator: From Cortex-R to Cortex-A

Memory layout

- Cortex-A lacks an MPU; requires extracting MMU tables for memory mappings.

```
r0 = virtual address
r1 = physical address | perm | attr
str  r1, [page_table_address, r0, lsr #18]
```

```
ldr  r0, =page_table_address
mcr  p15, 0x0, r0, cr2, cr0, 0x0
```

Boot Stage Translation

```
00000000 - 00100000 rwx
40000000 - 58800000 rwx
80000000 - 86000000 rw-
87000000 - 87100000 rw-
87200000 - 87300000 rw-
88100000 - 88200000 rw-
8f000000 - 9f000000 rw-
```

Building an emulator: From Cortex-R to Cortex-A

Memory layout

- Cortex-A lacks an MPU; requires extracting MMU tables for memory mappings.

```
r0 = virtual address
r1 = physical address | perm | attr
str  r1, [page_table_address, r0, lsr #18]
```

```
ldr  r0, =page_table_address
mcr  p15, 0×0, r0, cr2, cr0, 0×0
```

```
00000000 - 00100000 r-x
40000000 - 40100000 rw-
40100000 – 42b00000 r-x
42b00000 - 49d00000 rw-
49d00000 - 4a700000 r--
4a700000 - 4d800000 rw-
50000000 - 57e00000 rw-
80000000 - 86000000 rw-
87000000 - 87100000 rw-
87200000 - 87300000 rw-
88000000 - 88300000 rw-
8a000000 - 8b000000 rw-
8f000000 - 9f000000 rw-
c0000000 - e0000000 rw-
e0000000 - e8000000 r--
e8000000 - f0000000 rw-
```

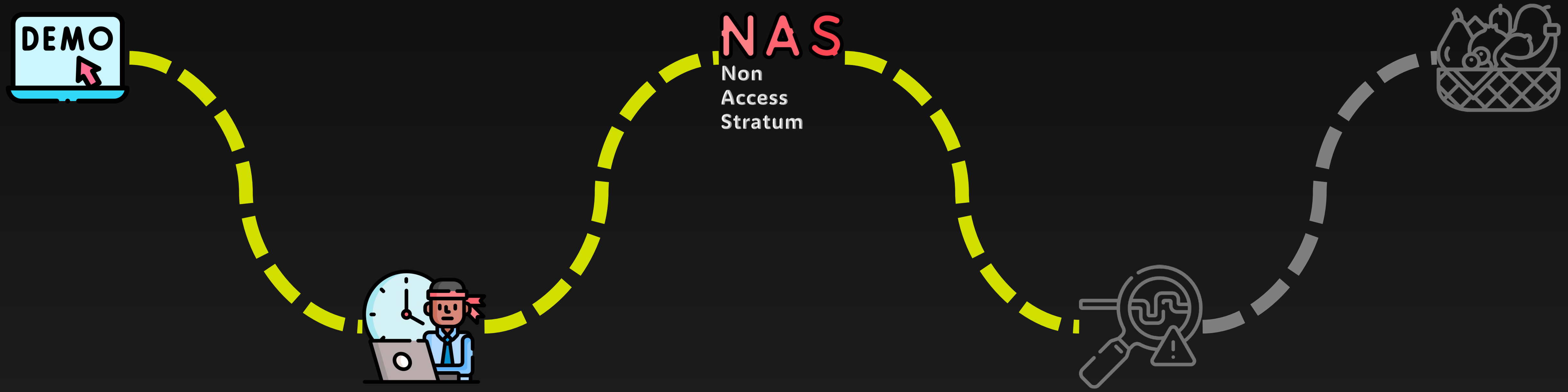
Building an emulator: From Cortex-R to Cortex-A

Timers

- Shannon Timer: Well reverse engineered already
  - ShannonEE (G. Hernandez – hardwear.io 22)
- But new devices use 8 timers instead of 6
- And new interrupt handler is required: Cortex-A15MPCore

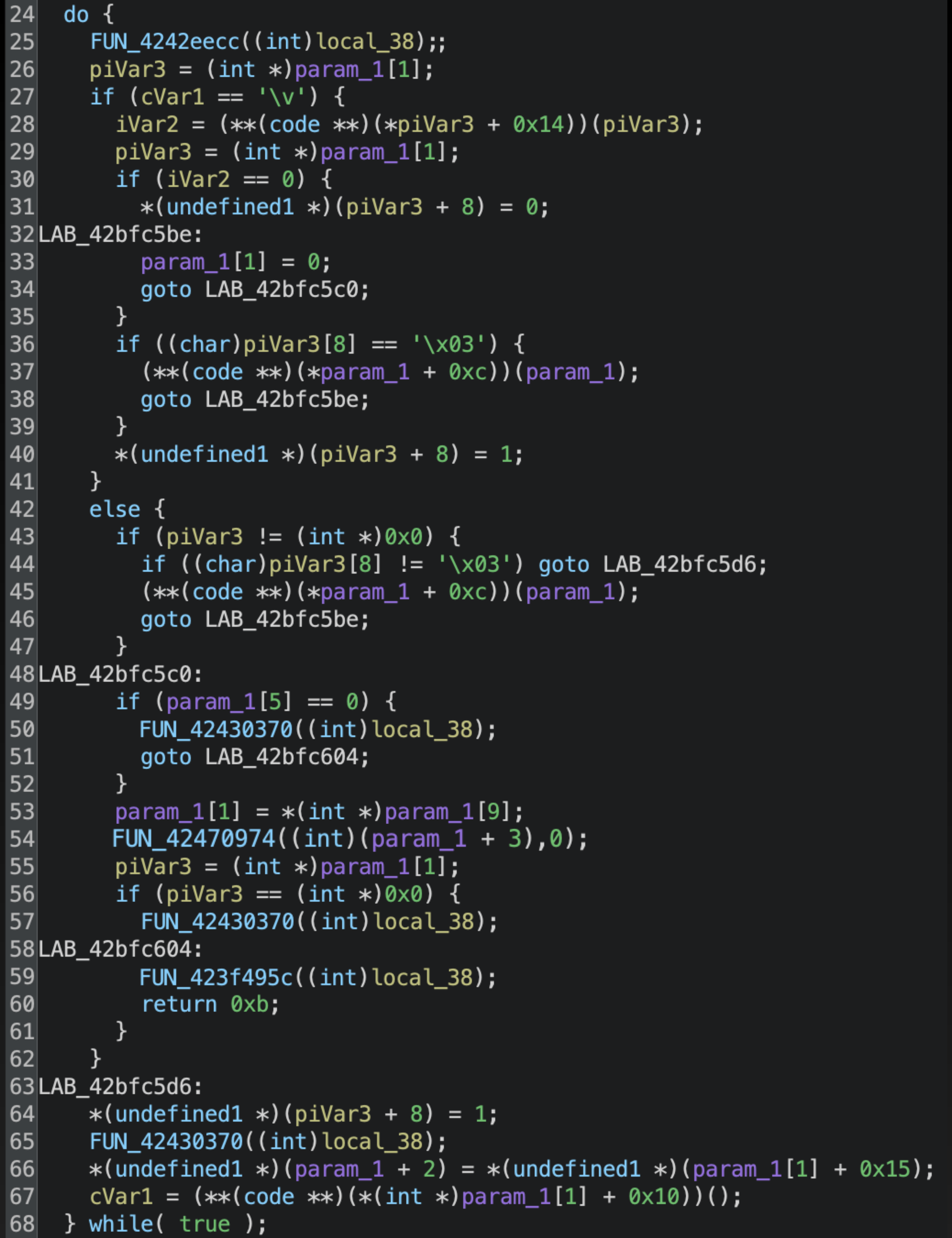
- Exynos Multi Core timer (MC timer) is utilized for first time.

Back at it: The 5G NAS Task



NAS
Non
Access
Stratum

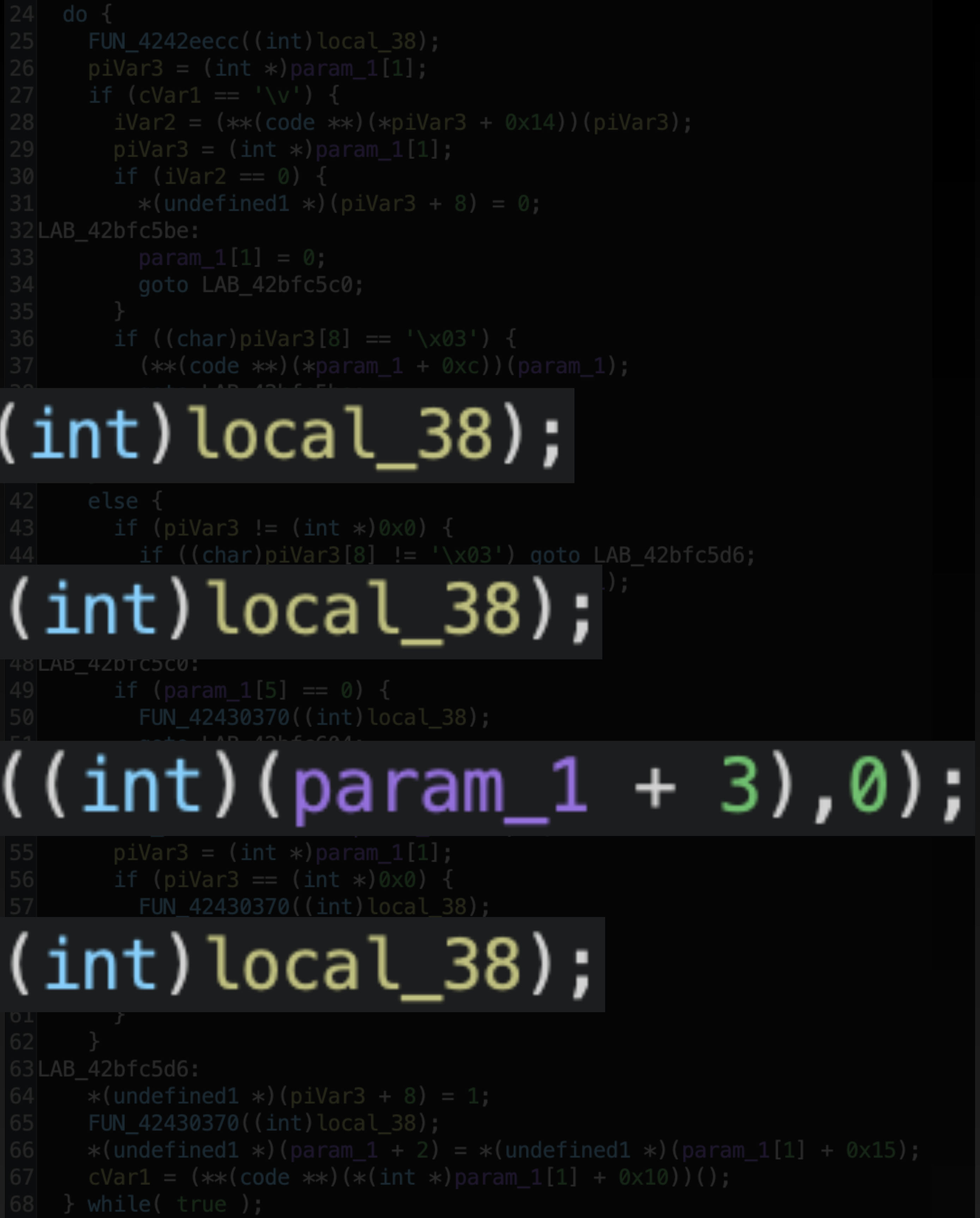
Starting at the main function



```
 do {
  FUN_4242eecc((int)local_38);;
  piVar3 = (int *)param_1[1];
  if (cVar1 == '\v') {
   iVar2 = (**(code **)(*piVar3 + 0x14))(piVar3);
   piVar3 = (int *)param_1[1];
   if (iVar2 == 0) {
    *(undefined1 *)(piVar3 + 8) = 0;
LAB_42bfc5be:
    param_1[1] = 0;
    goto LAB_42bfc5c0;
   }
   if ((char)piVar3[8] == '\x03') {
    (**(code **)(*param_1 + 0xc))(param_1);
    goto LAB_42bfc5be;
   }
   *(undefined1 *)(piVar3 + 8) = 1;
  }
  else {
   if (piVar3 != (int *)0x0) {
    if ((char)piVar3[8] != '\x03') goto LAB_42bfc5d6;
    (**(code **)(*param_1 + 0xc))(param_1);
    goto LAB_42bfc5be;
   }
LAB_42bfc5c0:
   if (param_1[5] == 0) {
    FUN_42430370((int)local_38);
    goto LAB_42bfc604;
   }
   param_1[1] = *(int *)param_1[9];
   FUN_42470974((int)(param_1 + 3),0);
   piVar3 = (int *)param_1[1];
   if (piVar3 == (int *)0x0) {
    FUN_42430370((int)local_38);
LAB_42bfc604:
    FUN_423f495c((int)local_38);
    return 0xb;
   }
  }
LAB_42bfc5d6:
  *(undefined1 *)(piVar3 + 8) = 1;
  FUN_42430370((int)local_38);
  *(undefined1 *)(param_1 + 2) = *(undefined1 *)(param_1[1] + 0x15);
  cVar1 = (**(code **)(*(int *)param_1[1] + 0x10))();
 } while( true );
```

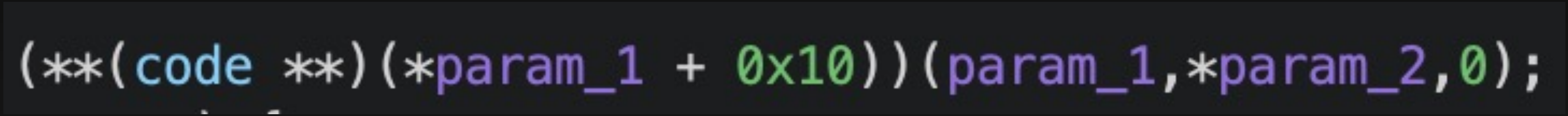
- You see these a lot of times:
- It’s easy. They’re function calls at some addresses.

```
FUN_4242eecc((int)local_38);

FUN_42430370((int)local_38);

FUN_42470974((int)(param_1 + 3),0);

FUN_423f495c((int)local_38);
```



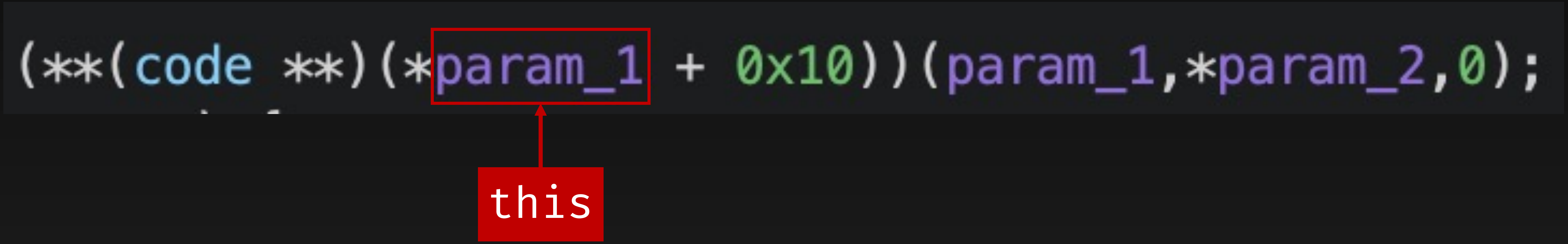
```
  do {
    FUN_4242eecc((int)local_38);
    piVar3 = (int *)param_1[1];
    if (cVar1 == '\v') {
      iVar2 = (**(code **)(*piVar3 + 0x14))(piVar3);
      piVar3 = (int *)param_1[1];
      if (iVar2 == 0) {
        *(undefined1 *)(piVar3 + 8) = 0;
LAB_42bfc5be:
        param_1[1] = 0;
        goto LAB_42bfc5c0;
      }
      if ((char)piVar3[8] == '\x03') {
        (**(code **)(*param_1 + 0xc))(param_1);

    else {
      if (piVar3 != (int *)0x0) {
        if ((char)piVar3[8] != '\x03') goto LAB_42bfc5d6;

LAB_42bfc5c0:
      if (param_1[5] == 0) {
        FUN_42430370((int)local_38);

      piVar3 = (int *)param_1[1];
      if (piVar3 == (int *)0x0) {
        FUN_42430370((int)local_38);

      }
    }
LAB_42bfc5d6:
    *(undefined1 *)(piVar3 + 8) = 1;
    FUN_42430370((int)local_38);
    *(undefined1 *)(param_1 + 2) = *(undefined1 *)(param_1[1] + 0x15);
    cVar1 = (**(code **)(*(int *)param_1[1] + 0x10))();
  } while( true );
```

What About This?



```
(**(code **)(*param_1 + 0x10))(param_1,*param_2,0);
```

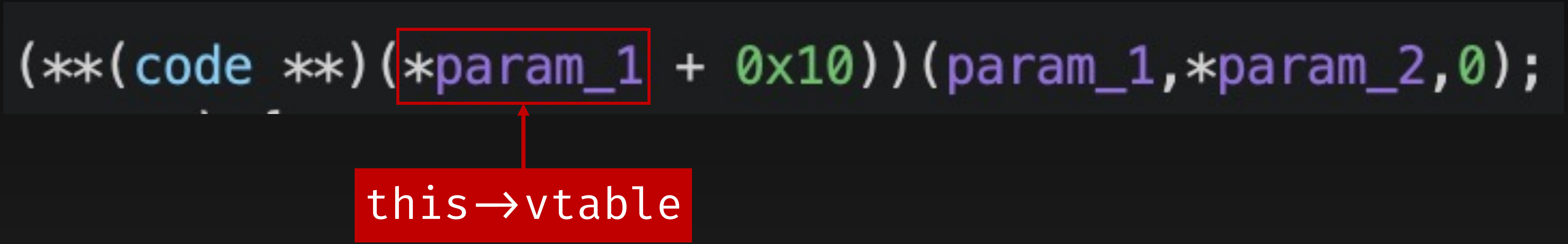
What About This?



```
(**(code **)(*param_1 + 0x10))(param_1,*param_2,0);
```

this

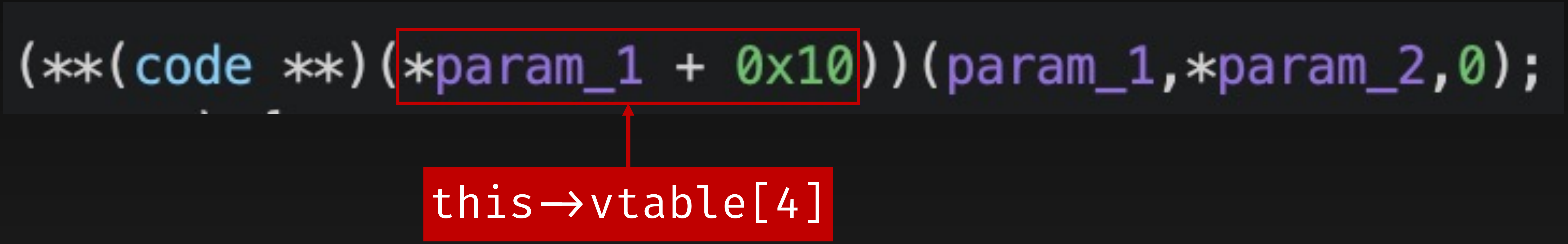
What About This?



```
(**(code **)(*param_1 + 0x10))(param_1,*param_2,0);
```

this→vtable

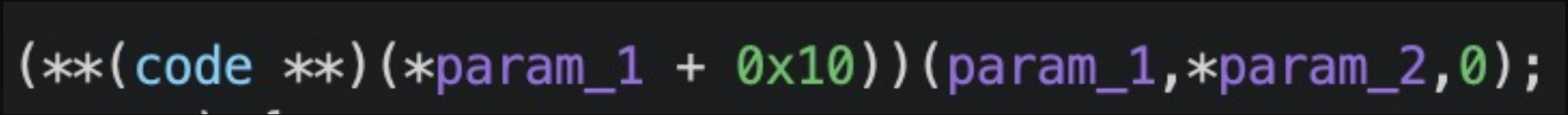
What About This?



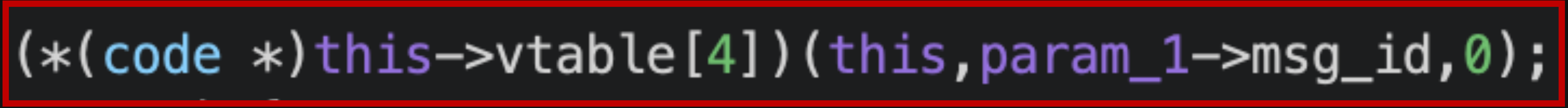
```
(**(code **)(*param_1 + 0x10))(param_1,*param_2,0);
```

this→vtable[4]

We Can Improve It:

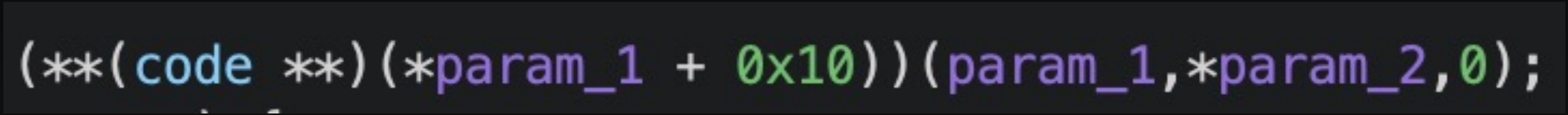


```
(**(code **)(*param_1 + 0x10))(param_1,*param_2,0);
```

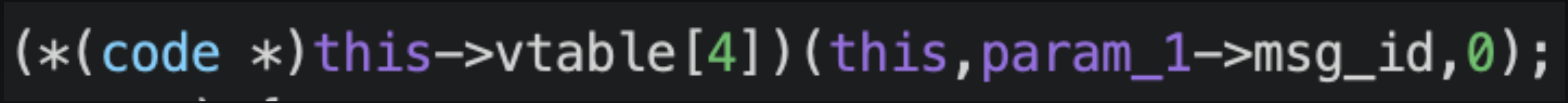


```
(*(code *)this->vtable[4])(this,param_1->msg_id,0);
```

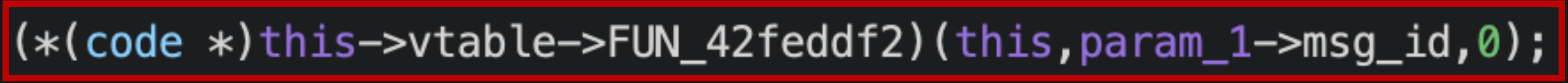
And Even Something Better



```
(**(code **)(*param_1 + 0x10))(param_1,*param_2,0);
```



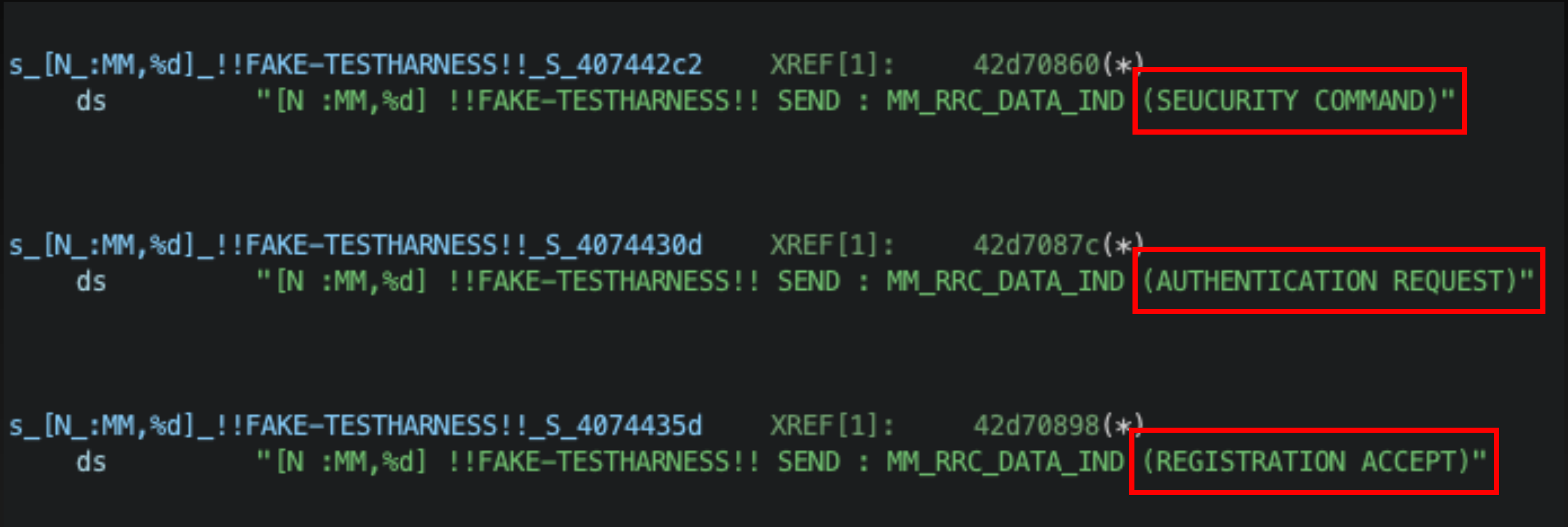
```
(*(code *)this->vtable[4])(this,param_1->msg_id,0);
```



```
(*(code *)this->vtable->FUN_42feddf2)(this,param_1->msg_id,0);
```

Harnessing The NAS task

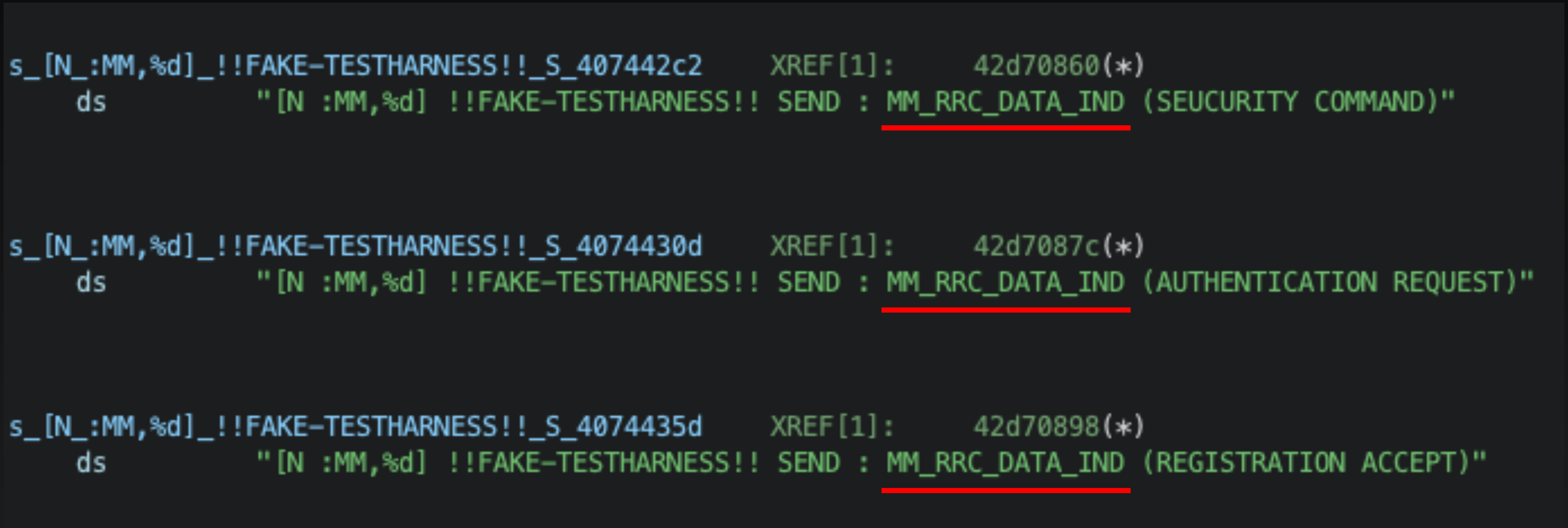
- Searching for message names revealed some



```
s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_407442c2     XREF[1]:     42d70860(*)
     ds            "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (SEUCURITY COMMAND)"


s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_4074430d     XREF[1]:     42d7087c(*)
     ds            "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (AUTHENTICATION REQUEST)"


s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_4074435d     XREF[1]:     42d70898(*)
     ds            "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (REGISTRATION ACCEPT)"
```

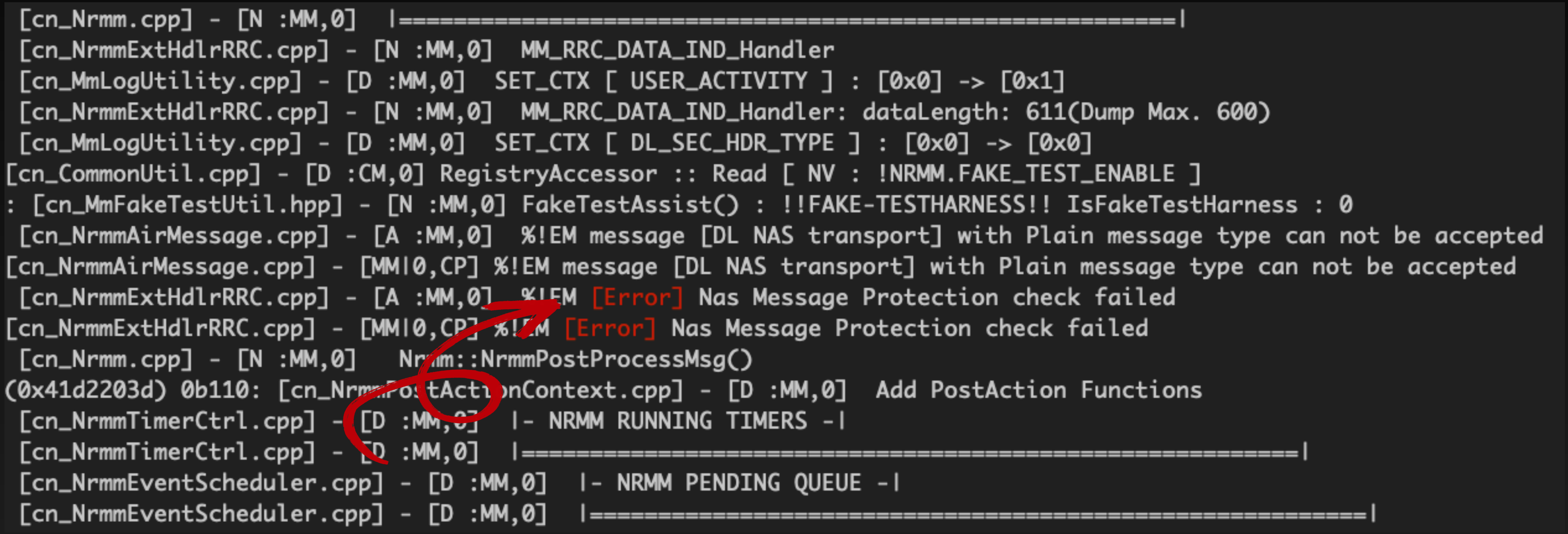
Harnessing The NAS task

- Searching for message names revealed some



```
s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_407442c2     XREF[1]:     42d70860(*)
    ds          "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (SEUCURITY COMMAND)"


s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_4074430d     XREF[1]:     42d7087c(*)
    ds          "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (AUTHENTICATION REQUEST)"


s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_4074435d     XREF[1]:     42d70898(*)
    ds          "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (REGISTRATION ACCEPT)"
```

Bypassing Security Checks in NAS

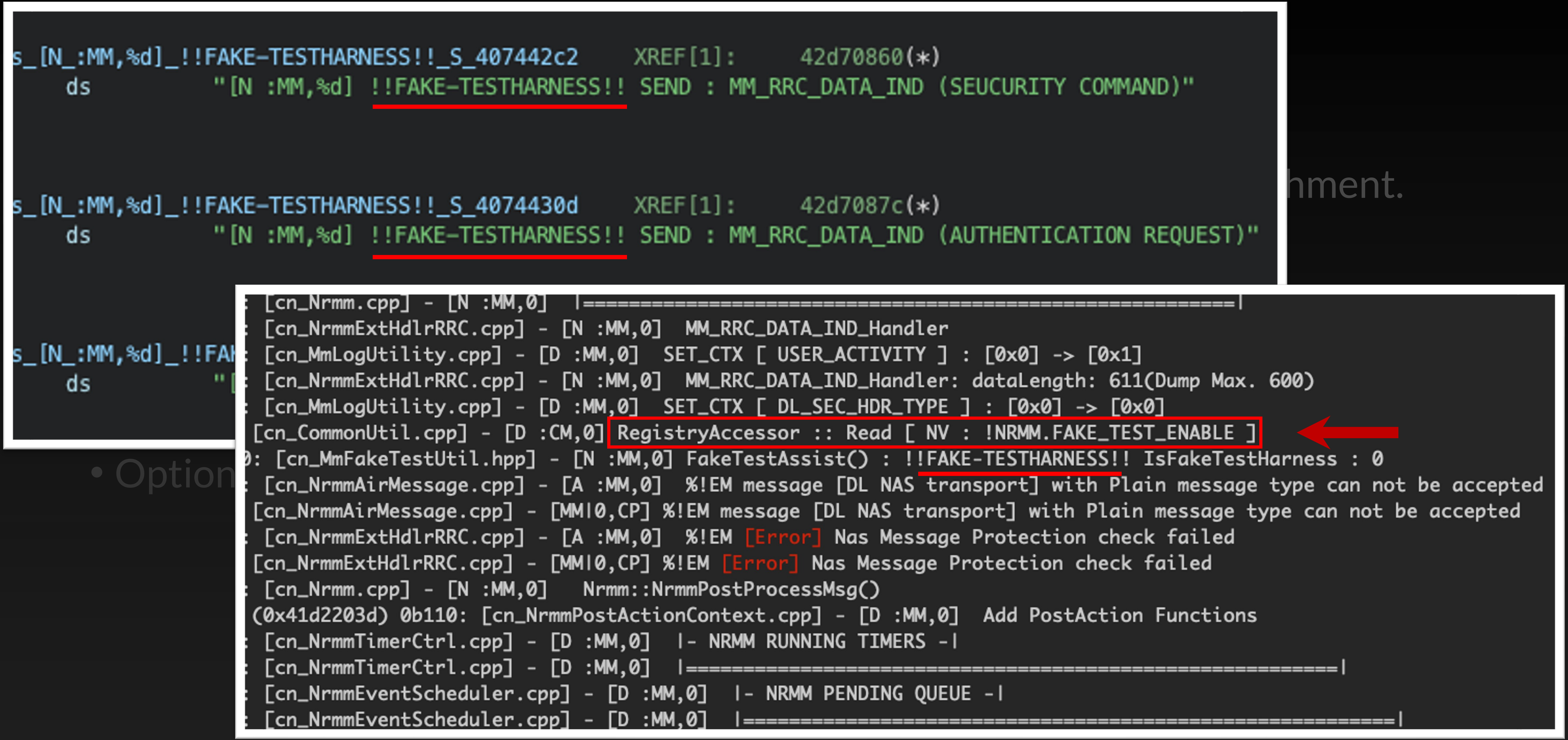


```
 [cn_Nrmm.cpp] - [N :MM,0]  |=========================================================|
 [cn_NrmmExtHdlrRRC.cpp] - [N :MM,0]  MM_RRC_DATA_IND_Handler
 [cn_MmLogUtility.cpp] - [D :MM,0]  SET_CTX [ USER_ACTIVITY ] : [0x0] -> [0x1]
 [cn_NrmmExtHdlrRRC.cpp] - [N :MM,0]  MM_RRC_DATA_IND_Handler: dataLength: 611(Dump Max. 600)
 [cn_MmLogUtility.cpp] - [D :MM,0]  SET_CTX [ DL_SEC_HDR_TYPE ] : [0x0] -> [0x0]
[cn_CommonUtil.cpp] - [D :CM,0] RegistryAccessor :: Read [ NV : !NRMM.FAKE_TEST_ENABLE ]
: [cn_MmFakeTestUtil.hpp] - [N :MM,0] FakeTestAssist() : !!FAKE-TESTHARNESS!! IsFakeTestHarness : 0
 [cn_NrmmAirMessage.cpp] - [A :MM,0]  %!EM message [DL NAS transport] with Plain message type can not be accepted
[cn_NrmmAirMessage.cpp] - [MM|0,CP] %!EM message [DL NAS transport] with Plain message type can not be accepted
 [cn_NrmmExtHdlrRRC.cpp] - [A :MM,0]  %!EM [Error] Nas Message Protection check failed
[cn_NrmmExtHdlrRRC.cpp] - [MM|0,CP] %!EM [Error] Nas Message Protection check failed
 [cn_Nrmm.cpp] - [N :MM,0]   Nrmm::NrmmPostProcessMsg()
(0x41d2203d) 0b110: [cn_NrmmPostActionContext.cpp] - [D :MM,0]  Add PostAction Functions
 [cn_NrmmTimerCtrl.cpp] - [D :MM,0]  |- NRMM RUNNING TIMERS -|
 [cn_NrmmTimerCtrl.cpp] - [D :MM,0]  |=========================================|
 [cn_NrmmEventScheduler.cpp] - [D :MM,0]  |- NRMM PENDING QUEUE -|
 [cn_NrmmEventScheduler.cpp] - [D :MM,0]  |===========================================|
```

Bypassing Security Checks in NAS

- Most of NAS messages are exchanged after security context establishment.
  - So, they're encrypted and integrity protected.

- Option 1: Handling encryption and integrity during fuzz testing and program ⟶ hard, not scalable
- Option 2: Leveraging other vulnerabilities: CVE-2023-50804 ⟶ patched
- Option 3: !!FAKE-TESTHARNESS!!



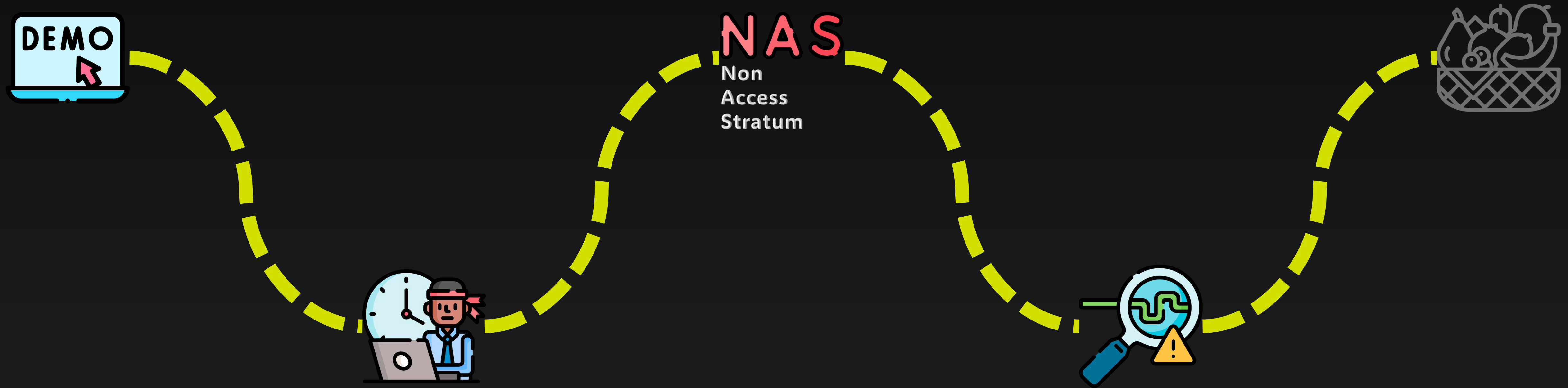
```
s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_407442c2     XREF[1]:     42d70860(*)
    ds          "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (SEUCURITY COMMAND)"


s_[N_:MM,%d]_!!FAKE-TESTHARNESS!!_S_4074430d     XREF[1]:     42d7087c(*)
    ds          "[N :MM,%d] !!FAKE-TESTHARNESS!! SEND : MM_RRC_DATA_IND (AUTHENTICATION REQUEST)"


s_[N_:MM,%d]_!!FAK
    ds          "[
```

```
  [cn_Nrmm.cpp] - [N :MM,0]   |==================================================|
: [cn_NrmmExtHdlrRRC.cpp] - [N :MM,0]  MM_RRC_DATA_IND_Handler
: [cn_MmLogUtility.cpp] - [D :MM,0]  SET_CTX [ USER_ACTIVITY ] : [0x0] -> [0x1]
: [cn_NrmmExtHdlrRRC.cpp] - [N :MM,0]  MM_RRC_DATA_IND_Handler: dataLength: 611(Dump Max. 600)
: [cn_MmLogUtility.cpp] - [D :MM,0]  SET_CTX [ DL_SEC_HDR_TYPE ] : [0x0] -> [0x0]
 [cn_CommonUtil.cpp] - [D :CM,0] RegistryAccessor :: Read [ NV : !NRMM.FAKE_TEST_ENABLE ]
0: [cn_MmFakeTestUtil.hpp] - [N :MM,0] FakeTestAssist() : !!FAKE-TESTHARNESS!! IsFakeTestHarness : 0
: [cn_NrmmAirMessage.cpp] - [A :MM,0]  %!EM message [DL NAS transport] with Plain message type can not be accepted
 [cn_NrmmAirMessage.cpp] - [MM|0,CP] %!EM message [DL NAS transport] with Plain message type can not be accepted
: [cn_NrmmExtHdlrRRC.cpp] - [A :MM,0]  %!EM [Error] Nas Message Protection check failed
 [cn_NrmmExtHdlrRRC.cpp] - [MM|0,CP] %!EM [Error] Nas Message Protection check failed
: [cn_Nrmm.cpp] - [N :MM,0]   Nrmm::NrmmPostProcessMsg()
 (0x41d2203d) 0b110: [cn_NrmmPostActionContext.cpp] - [D :MM,0]  Add PostAction Functions
: [cn_NrmmTimerCtrl.cpp] - [D :MM,0]   |- NRMM RUNNING TIMERS -|
: [cn_NrmmTimerCtrl.cpp] - [D :MM,0]   |==============================================|
: [cn_NrmmEventScheduler.cpp] - [D :MM,0]   |- NRMM PENDING QUEUE -|
: [cn_NrmmEventScheduler.cpp] - [D :MM,0]   |==============================================|
```

hment.

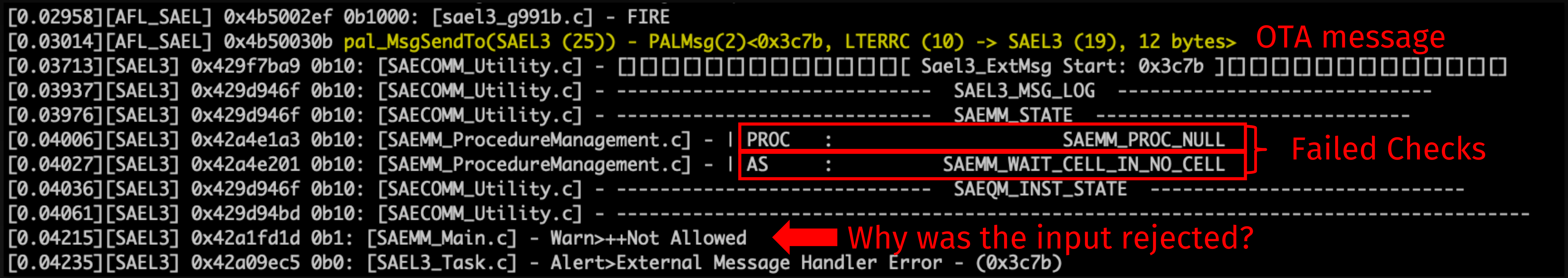
• Option

How did we really test it?



NAS
Non
Access
Stratum

Why Is Testing NAS Task Difficult?



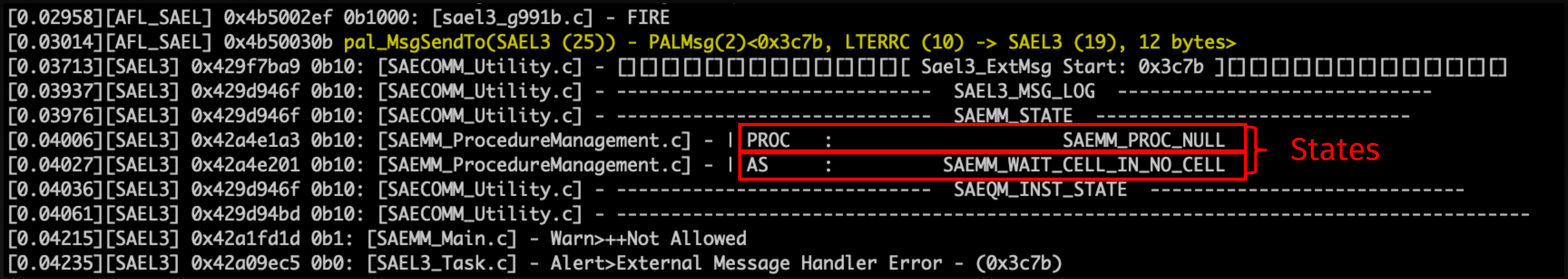
```
[0.02958][AFL_SAEL] 0x4b5002ef 0b1000: [sael3_g991b.c] - FIRE
[0.03014][AFL_SAEL] 0x4b50030b pal_MsgSendTo(SAEL3 (25)) - PALMsg(2)<0x3c7b, LTERRC (10) -> SAEL3 (19), 12 bytes>
[0.03713][SAEL3] 0x429f7ba9 0b10: [SAECOMM_Utility.c] - □□□□□□□□□□□□□[ Sael3_ExtMsg Start: 0x3c7b ]□□□□□□□□□□□□□
[0.03937][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - -----------------------------  SAEL3_MSG_LOG  -----------------------------
[0.03976][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - -----------------------------  SAEMM_STATE  -----------------------------
[0.04006][SAEL3] 0x42a4e1a3 0b10: [SAEMM_ProcedureManagement.c] - | PROC    :                             SAEMM_PROC_NULL
[0.04027][SAEL3] 0x42a4e201 0b10: [SAEMM_ProcedureManagement.c] - | AS      :                  SAEMM_WAIT_CELL_IN_NO_CELL
[0.04036][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - -----------------------------  SAEQM_INST_STATE  -----------------------------
[0.04061][SAEL3] 0x429d94bd 0b10: [SAECOMM_Utility.c] - -------------------------------------------------------------------------------------
[0.04215][SAEL3] 0x42a1fd1d 0b1: [SAEMM_Main.c] - Warn>++Not Allowed
[0.04235][SAEL3] 0x42a09ec5 0b0: [SAEL3_Task.c] - Alert>External Message Handler Error - (0x3c7b)
```

OTA message

Failed Checks

Why was the input rejected?

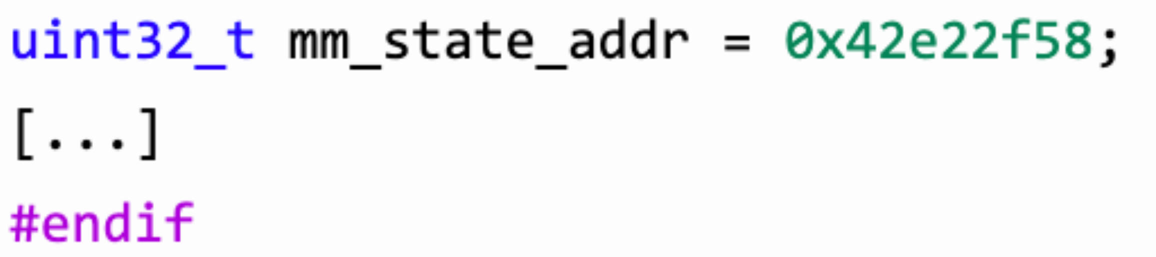
Why Testing NAS Task is Difficult?



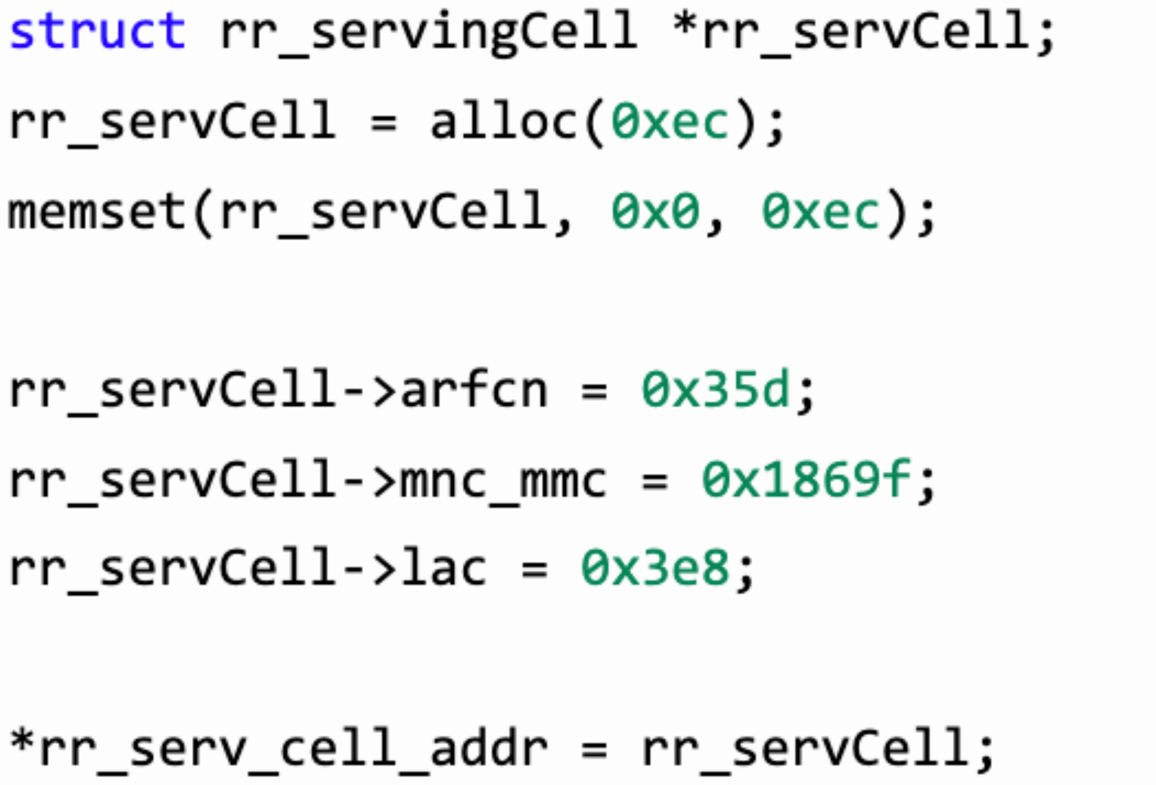
```
[0.02958][AFL_SAEL] 0x4b5002ef 0b1000: [sael3_g991b.c] - FIRE
[0.03014][AFL_SAEL] 0x4b50030b pal_MsgSendTo(SAEL3 (25)) - PALMsg(2)<0x3c7b, LTERRC (10) -> SAEL3 (19), 12 bytes>
[0.03713][SAEL3] 0x429f7ba9 0b10: [SAECOMM_Utility.c] - □□□□□□□□□□□□□□□□[ Sael3_ExtMsg Start: 0x3c7b ]□□□□□□□□□□□□□□□
[0.03937][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - ----------------------------  SAEL3_MSG_LOG  ----------------------------
[0.03976][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - ----------------------------  SAEMM_STATE  ----------------------------
[0.04006][SAEL3] 0x42a4e1a3 0b10: [SAEMM_ProcedureManagement.c] - | PROC    :                        SAEMM_PROC_NULL
[0.04027][SAEL3] 0x42a4e201 0b10: [SAEMM_ProcedureManagement.c] - | AS      :             SAEMM_WAIT_CELL_IN_NO_CELL
[0.04036][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - ----------------------------  SAEQM_INST_STATE  ----------------------------
[0.04061][SAEL3] 0x429d94bd 0b10: [SAECOMM_Utility.c] - ----------------------------------------------------------------------------------
[0.04215][SAEL3] 0x42a1fd1d 0b1: [SAEMM_Main.c] - Warn>++Not Allowed
[0.04235][SAEL3] 0x42a09ec5 0b0: [SAEL3_Task.c] - Alert>External Message Handler Error - (0x3c7b)
```

States

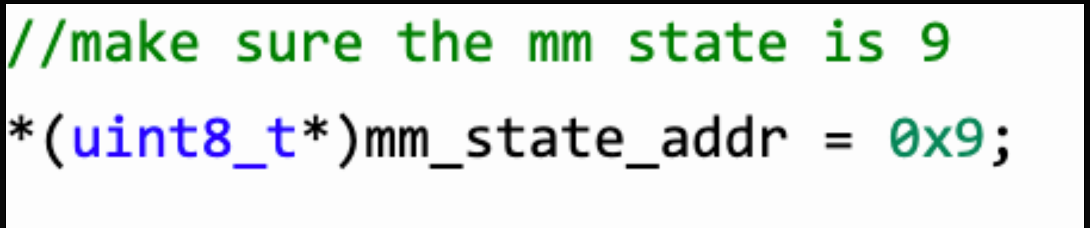
How states were handled so far



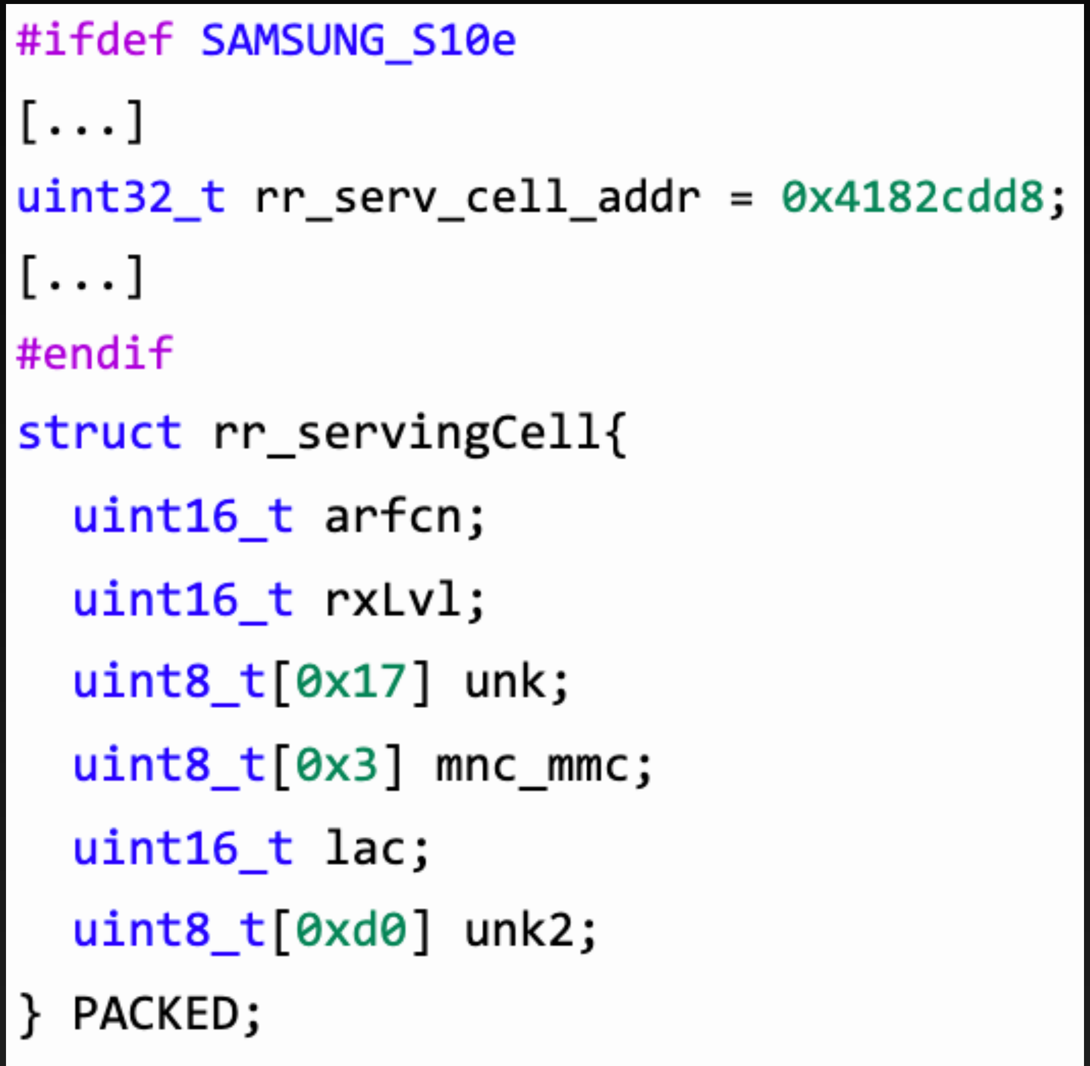
```
uint32_t mm_state_addr = 0x42e22f58;
[...]
#endif
```



```
struct rr_servingCell *rr_servCell;
rr_servCell = alloc(0xec);
memset(rr_servCell, 0x0, 0xec);

rr_servCell->arfcn = 0x35d;
rr_servCell->mnc_mmc = 0x1869f;
rr_servCell->lac = 0x3e8;

*rr_serv_cell_addr = rr_servCell;
```



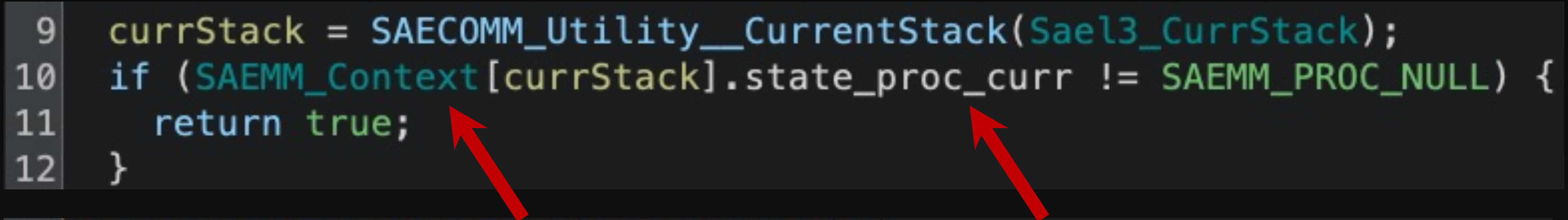
```
//make sure the mm state is 9
*(uint8_t*)mm_state_addr = 0x9;
```



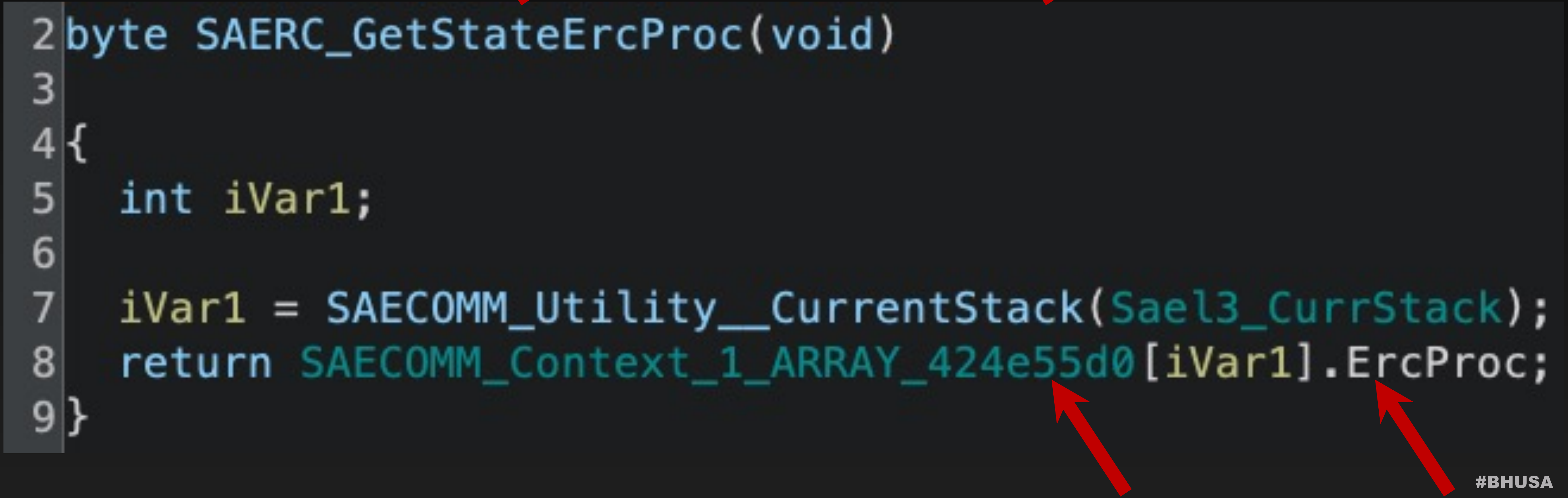
```
#ifdef SAMSUNG_S10e
[...]
uint32_t rr_serv_cell_addr = 0x4182cdd8;
[...]
#endif
struct rr_servingCell{
  uint16_t arfcn;
  uint16_t rxLvl;
  uint8_t[0x17] unk;
  uint8_t[0x3] mnc_mmc;
  uint16_t lac;
  uint8_t[0xd0] unk2;
} PACKED;
```

States in old-G vs 5G

4G States



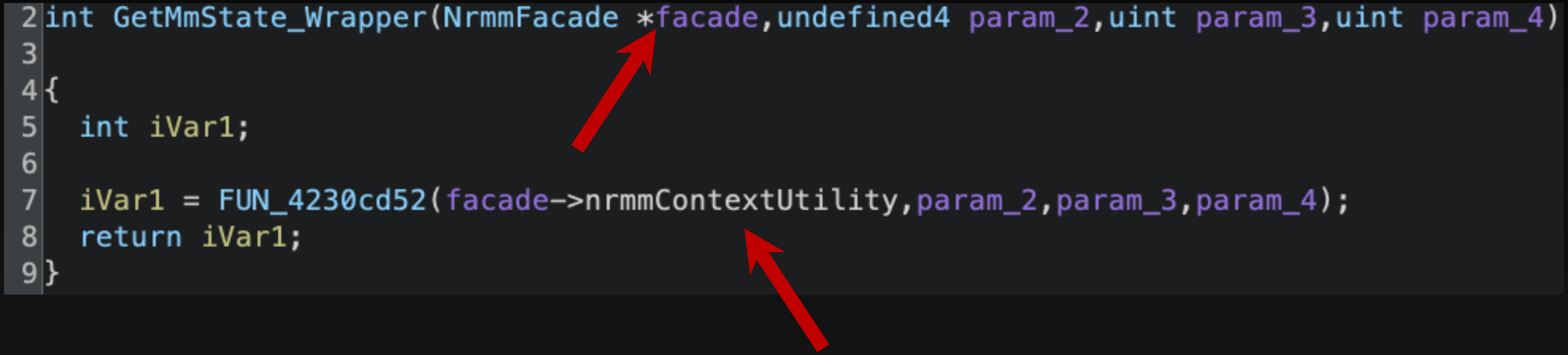
```
  currStack = SAECOMM_Utility__CurrentStack(Sael3_CurrStack);
  if (SAEMM_Context[currStack].state_proc_curr != SAEMM_PROC_NULL) {
    return true;
  }
```



```
byte SAERC_GetStateErcProc(void)

{
  int iVar1;

  iVar1 = SAECOMM_Utility__CurrentStack(Sael3_CurrStack);
  return SAECOMM_Context_1_ARRAY_424e55d0[iVar1].ErcProc;
}
```

States in old-G vs 5G

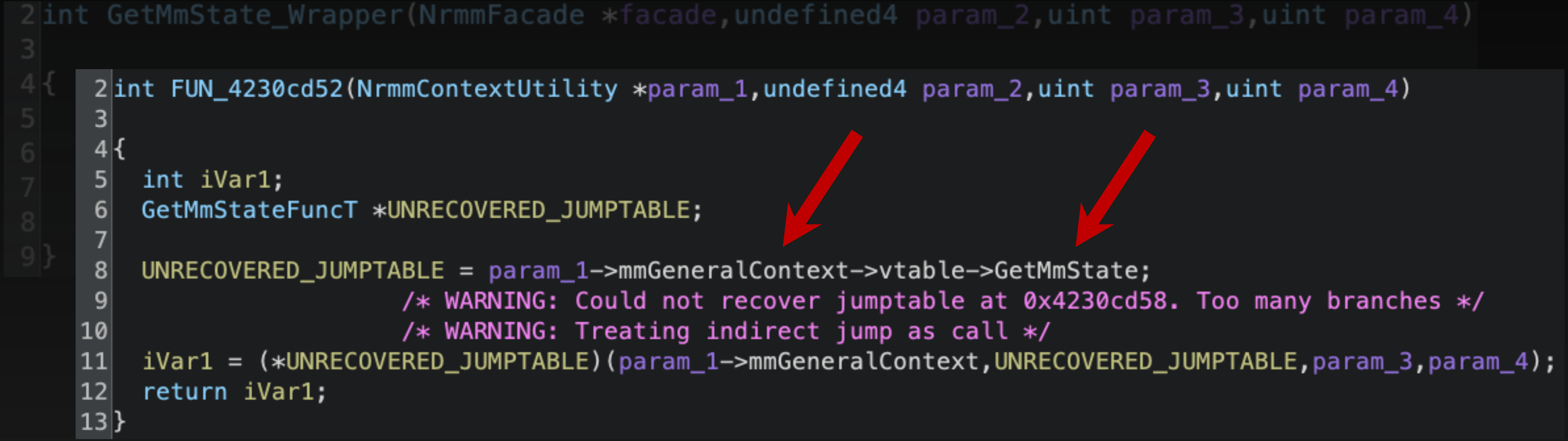
5G States



```
int GetMmState_Wrapper(NrmmFacade *facade,undefined4 param_2,uint param_3,uint param_4)

{
  int iVar1;

  iVar1 = FUN_4230cd52(facade->nrmmContextUtility,param_2,param_3,param_4);
  return iVar1;
}
```

States in old-G vs 5G

5G States

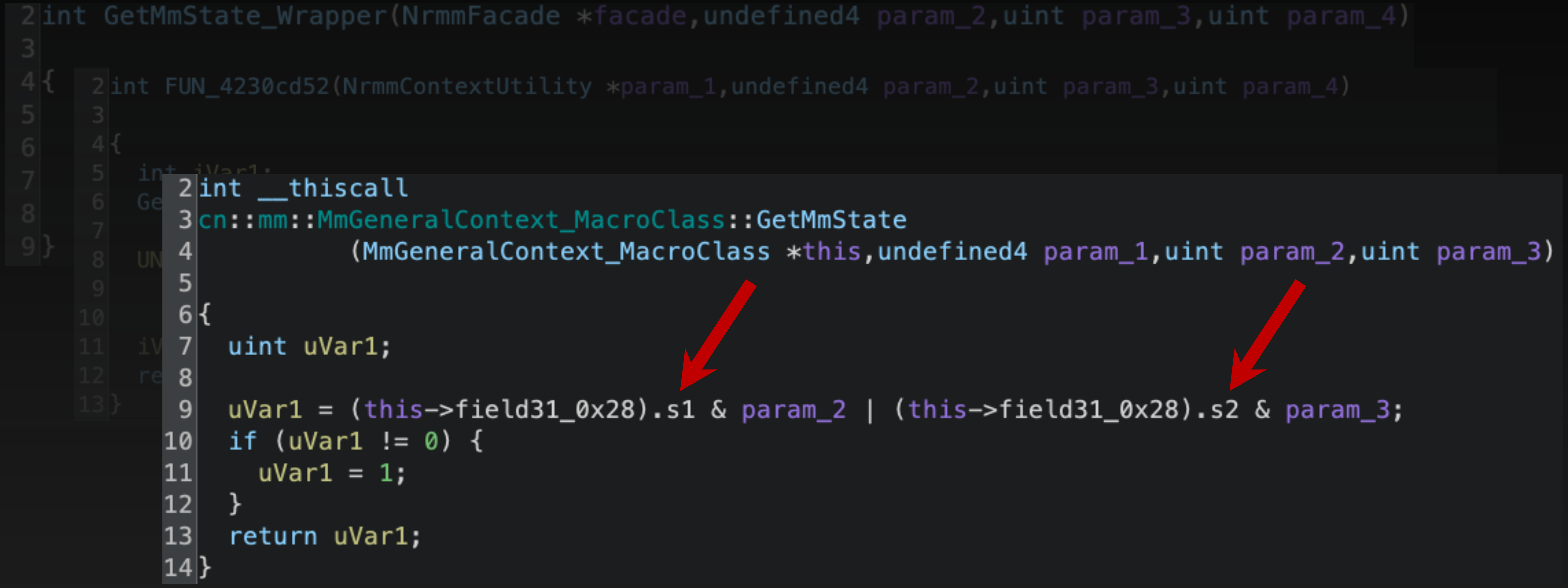


```
int GetMmState_Wrapper(NrmmFacade *facade,undefined4 param_2,uint param_3,uint param_4)

{
}
```

```
int FUN_4230cd52(NrmmContextUtility *param_1,undefined4 param_2,uint param_3,uint param_4)

{
  int iVar1;
  GetMmStateFuncT *UNRECOVERED_JUMPTABLE;

  UNRECOVERED_JUMPTABLE = param_1->mmGeneralContext->vtable->GetMmState;
                    /* WARNING: Could not recover jumptable at 0x4230cd58. Too many branches */
                    /* WARNING: Treating indirect jump as call */
  iVar1 = (*UNRECOVERED_JUMPTABLE)(param_1->mmGeneralContext,UNRECOVERED_JUMPTABLE,param_3,param_4);
  return iVar1;
}
```

States in old-G vs 5G

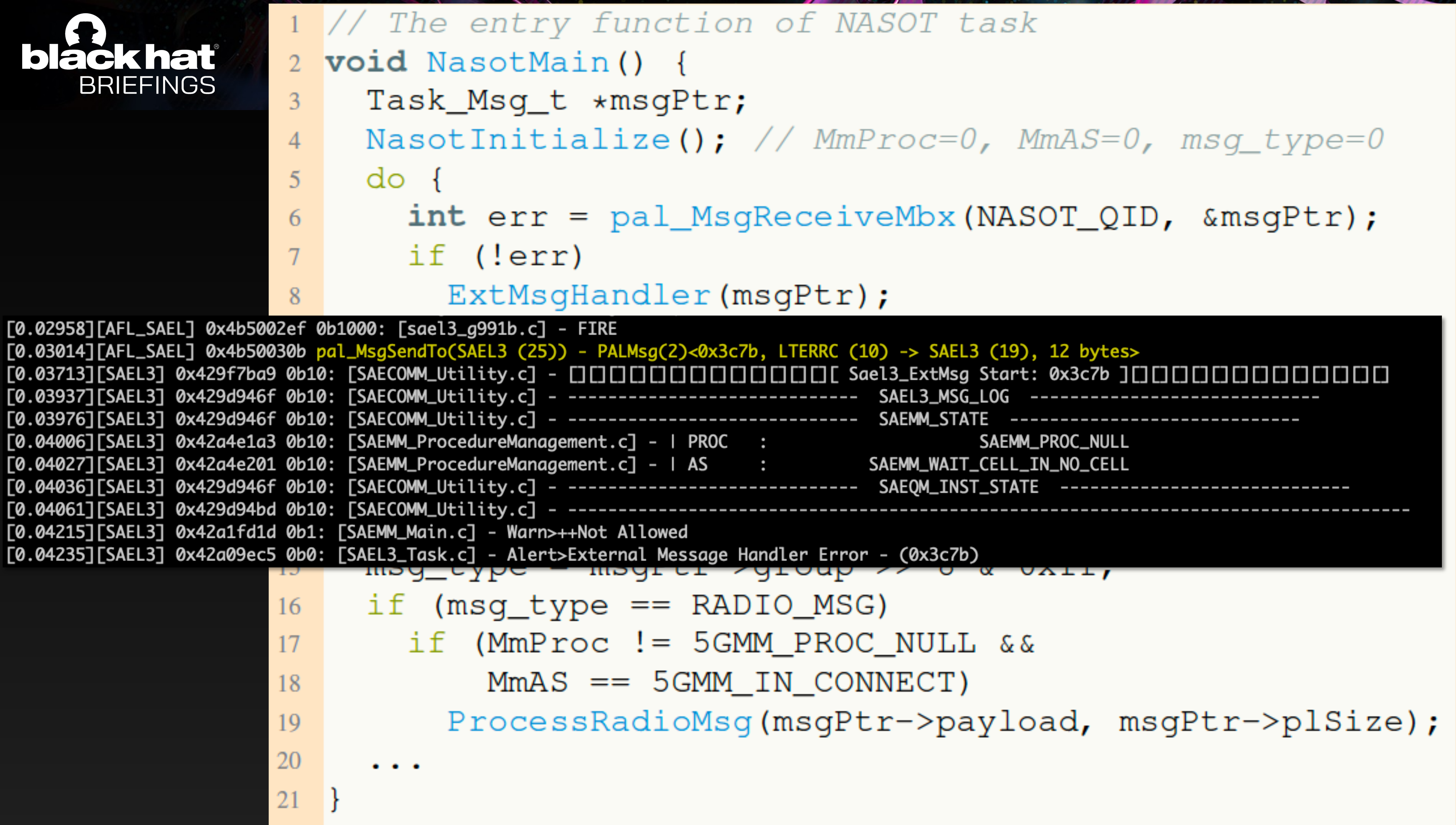
5G States



```
int GetMmState_Wrapper(NrmmFacade *facade,undefined4 param_2,uint param_3,uint param_4)

{





}
```

```
int FUN_4230cd52(NrmmContextUtility *param_1,undefined4 param_2,uint param_3,uint param_4)

{
  int iVar1;
  Ge

  UN


  iV
  re
}
```

```
int __thiscall
cn::mm::MmGeneralContext_MacroClass::GetMmState
          (MmGeneralContext_MacroClass *this,undefined4 param_1,uint param_2,uint param_3)

{
  uint uVar1;

  uVar1 = (this->field31_0x28).s1 & param_2 | (this->field31_0x28).s2 & param_3;
  if (uVar1 != 0) {
    uVar1 = 1;
  }
  return uVar1;
}
```



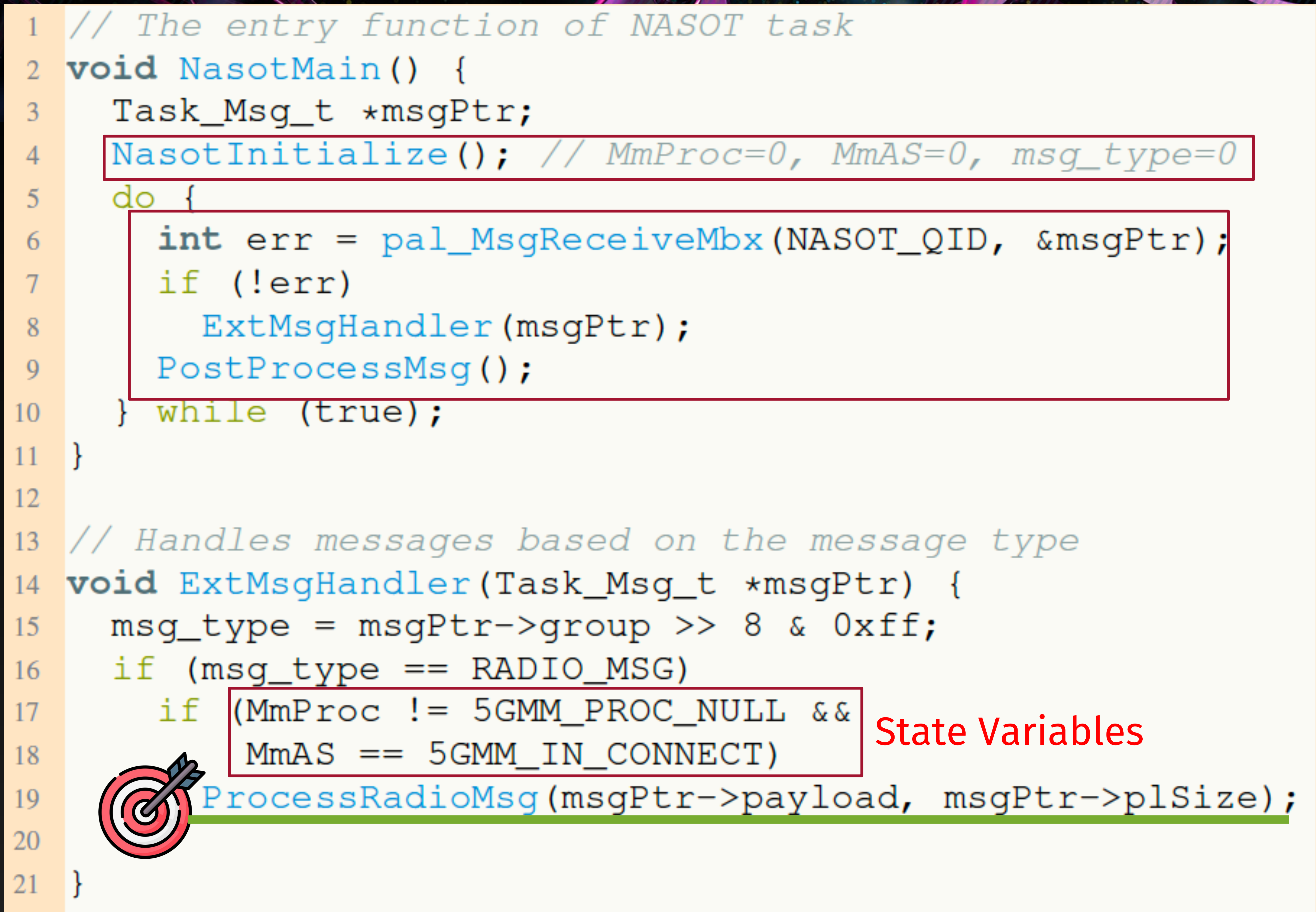
```c
// The entry function of NASOT task
void NasotMain() {
  Task_Msg_t *msgPtr;
  NasotInitialize(); // MmProc=0, MmAS=0, msg_type=0
  do {
    int err = pal_MsgReceiveMbx(NASOT_QID, &msgPtr);
    if (!err)
      ExtMsgHandler(msgPtr);
```

```
[0.02958][AFL_SAEL] 0x4b5002ef 0b1000: [sael3_g991b.c] - FIRE
[0.03014][AFL_SAEL] 0x4b50030b pal_MsgSendTo(SAEL3 (25)) - PALMsg(2)<0x3c7b, LTERRC (10) -> SAEL3 (19), 12 bytes>
[0.03713][SAEL3] 0x429f7ba9 0b10: [SAECOMM_Utility.c] - □□□□□□□□□□□□□□□[ Sael3_ExtMsg Start: 0x3c7b ]□□□□□□□□□□□□□□□
[0.03937][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - ----------------------------  SAEL3_MSG_LOG  ----------------------------
[0.03976][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - ----------------------------  SAEMM_STATE  ----------------------------
[0.04006][SAEL3] 0x42a4e1a3 0b10: [SAEMM_ProcedureManagement.c] - | PROC    :                SAEMM_PROC_NULL
[0.04027][SAEL3] 0x42a4e201 0b10: [SAEMM_ProcedureManagement.c] - | AS      :           SAEMM_WAIT_CELL_IN_NO_CELL
[0.04036][SAEL3] 0x429d946f 0b10: [SAECOMM_Utility.c] - ----------------------------  SAEQM_INST_STATE  ----------------------------
[0.04061][SAEL3] 0x429d94bd 0b10: [SAECOMM_Utility.c] - ----------------------------------------------------------------------------------
[0.04215][SAEL3] 0x42a1fd1d 0b1: [SAEMM_Main.c] - Warn>++Not Allowed
[0.04235][SAEL3] 0x42a09ec5 0b0: [SAEL3_Task.c] - Alert>External Message Handler Error - (0x3c7b)
```

```c
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
          MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Initialization

Message processing Loop



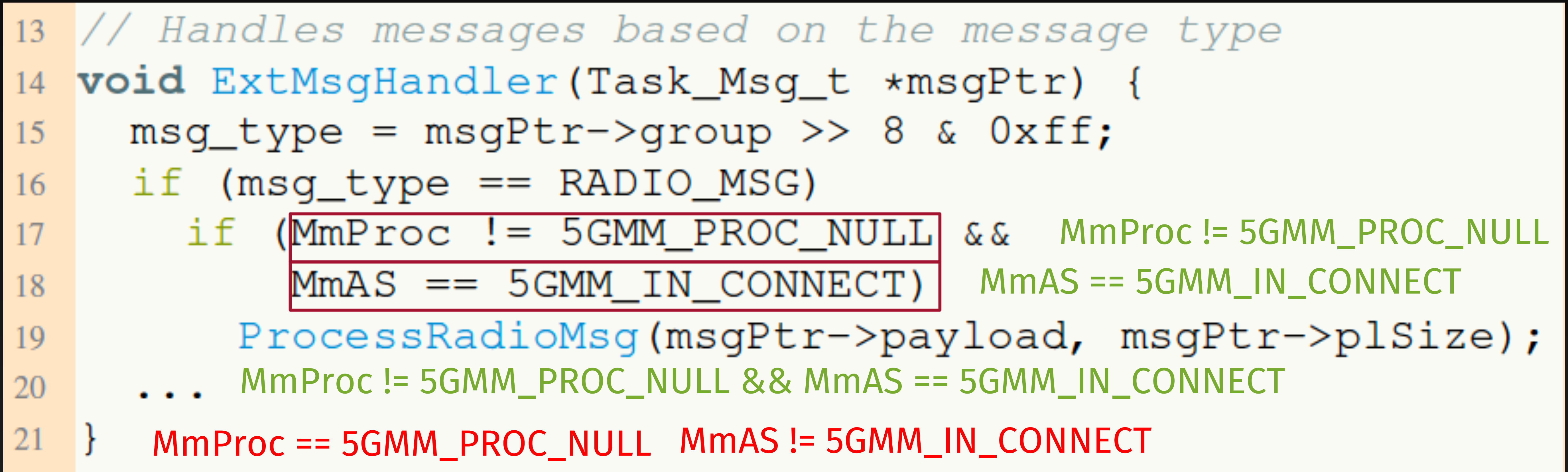
```
// The entry function of NASOT task
void NasotMain() {
  Task_Msg_t *msgPtr;
  NasotInitialize(); // MmProc=0, MmAS=0, msg_type=0
  do {
    int err = pal_MsgReceiveMbx(NASOT_QID, &msgPtr);
    if (!err)
      ExtMsgHandler(msgPtr);
    PostProcessMsg();
  } while (true);
}

// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);

}
```

State Variables

Symbolic Execution Preliminaries

MmProc, MmAS is symbolic (can represent any value)

msgPtr, msg_type, group, ... are all symbolic



```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

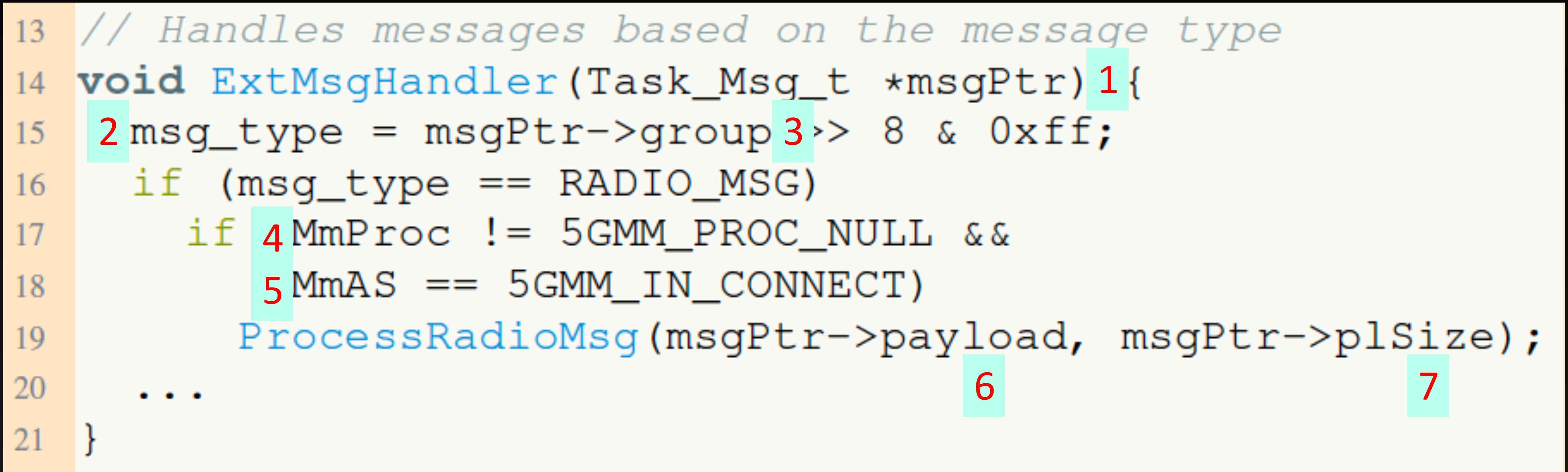
MmProc != 5GMM_PROC_NULL

MmAS == 5GMM_IN_CONNECT

MmProc != 5GMM_PROC_NULL && MmAS == 5GMM_IN_CONNECT

MmProc == 5GMM_PROC_NULL MmAS != 5GMM_IN_CONNECT

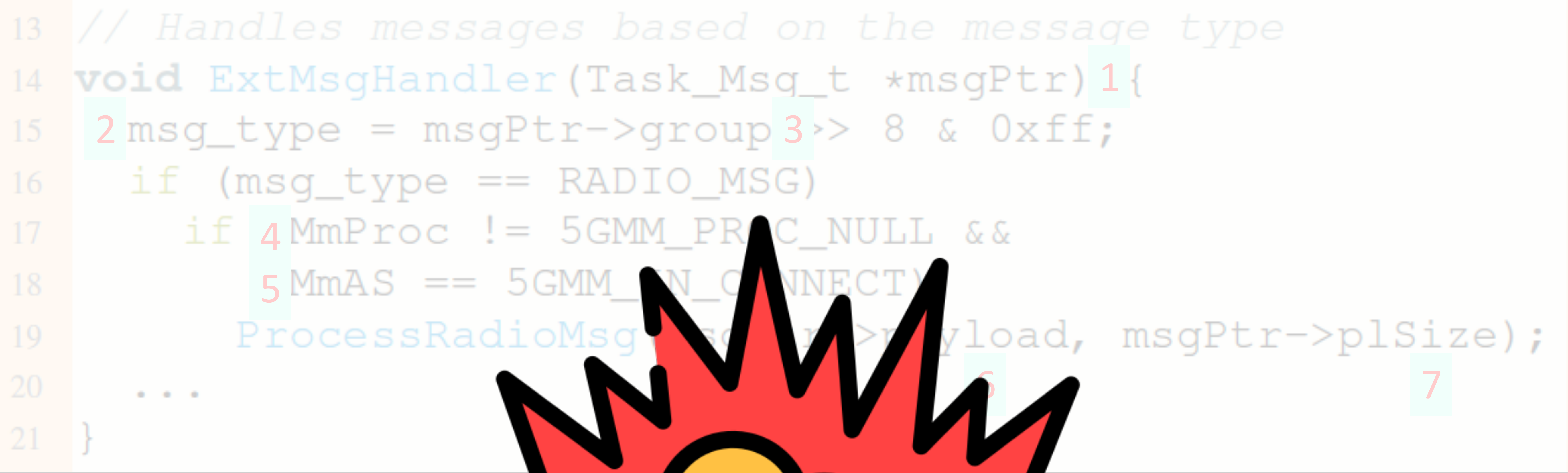
The State Explosion Problem



```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr){
 msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if MmProc != 5GMM_PROC_NULL &&
       MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

- 5 lines of code
- 7 symbolic variables (2 states)
- 4 paths

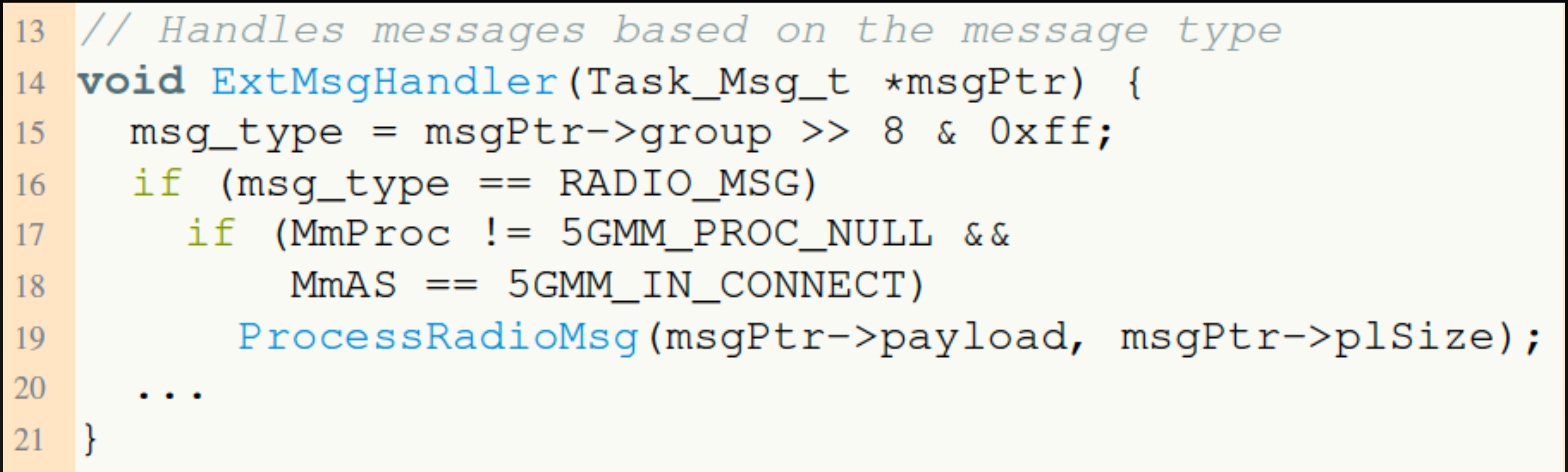
The State Explosion Problem



```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr)1{
 2msg_type = msgPtr->group3>> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if 4MmProc != 5GMM_PROC_NULL &&
       5MmAS == 5GMM_IN_CONNECTED)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
                                   6                  7
   ...
}
```

- 5 lines of code
- 7 symbolic variables (2
- 4 paths

How about we only analyze state variables?

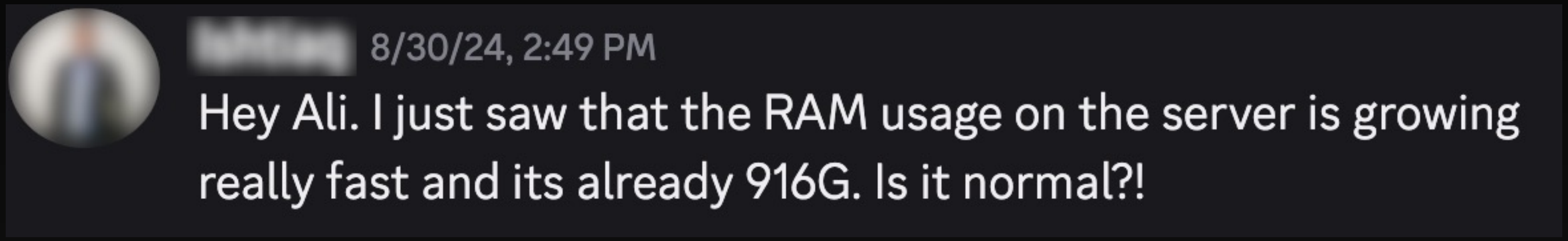


```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
          MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
   ...
}
```

- How do we identify state variable?

Is it enough?

No



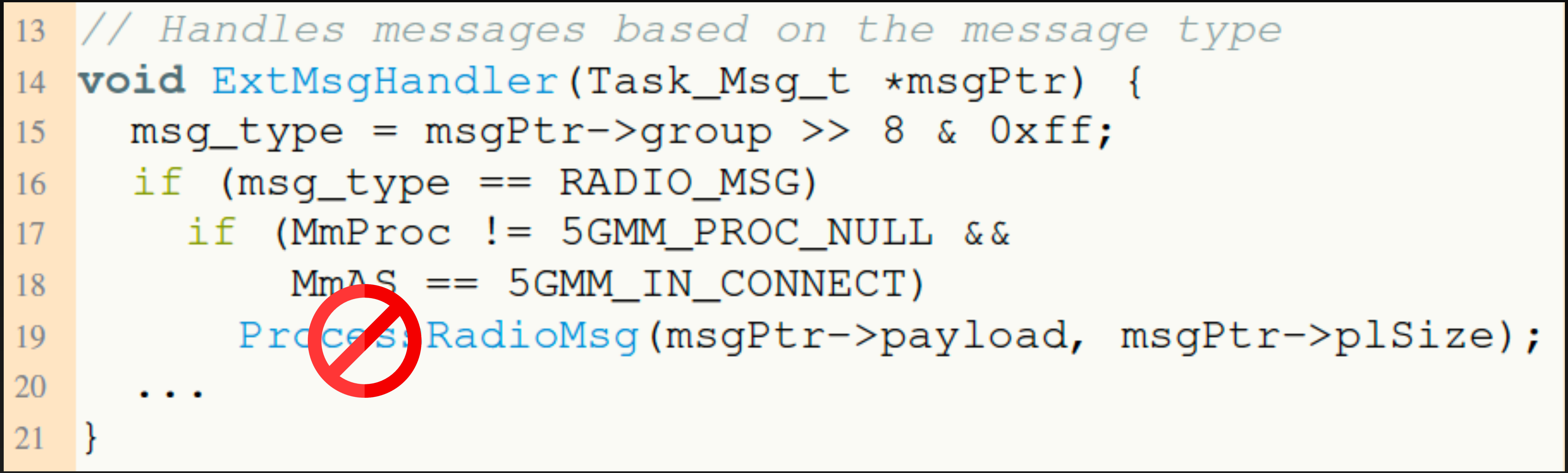
[redacted] 8/30/24, 2:49 PM

Hey Ali. I just saw that the RAM usage on the server is growing really fast and its already 916G. Is it normal?!

- ~100 state variables
- ~ 4 hours
- ~9k paths
- *> 1 TB Memory consumed*

Is It Enough?

No



```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

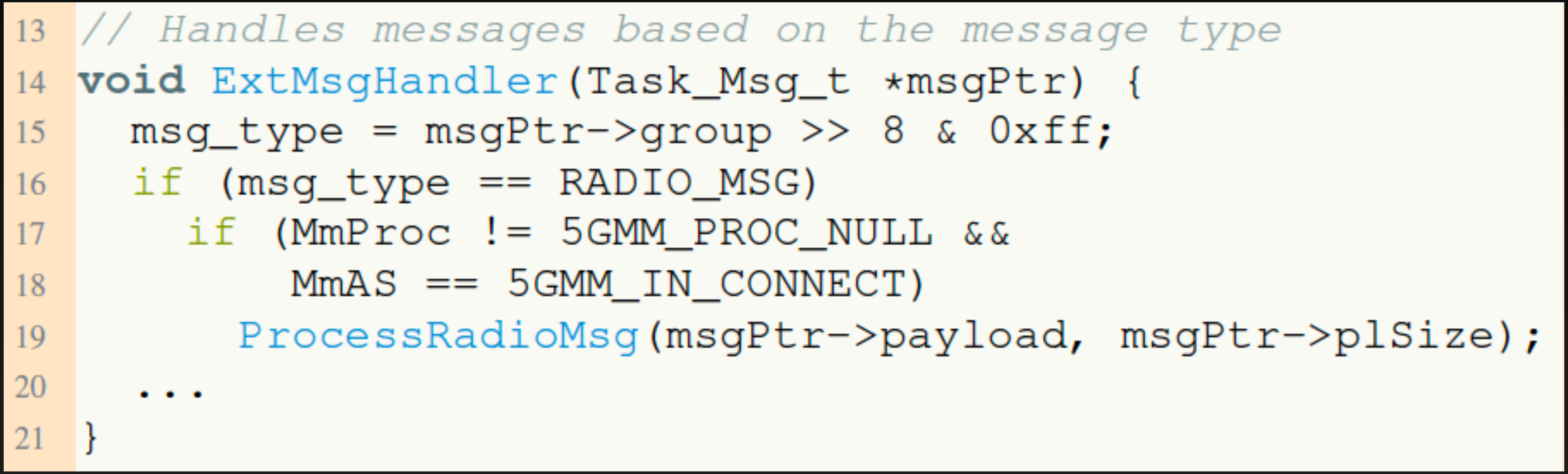
"Stateful Analysis and Fuzzing of Commercial Baseband Firmware" (IEEE S&P 2025).

- State variable identification
- Function pointer
- State variable analysis prioritization
- Use identified state variable conditions
- Grammar-aware test generation
- ...

Iterative Symbolic Analysis

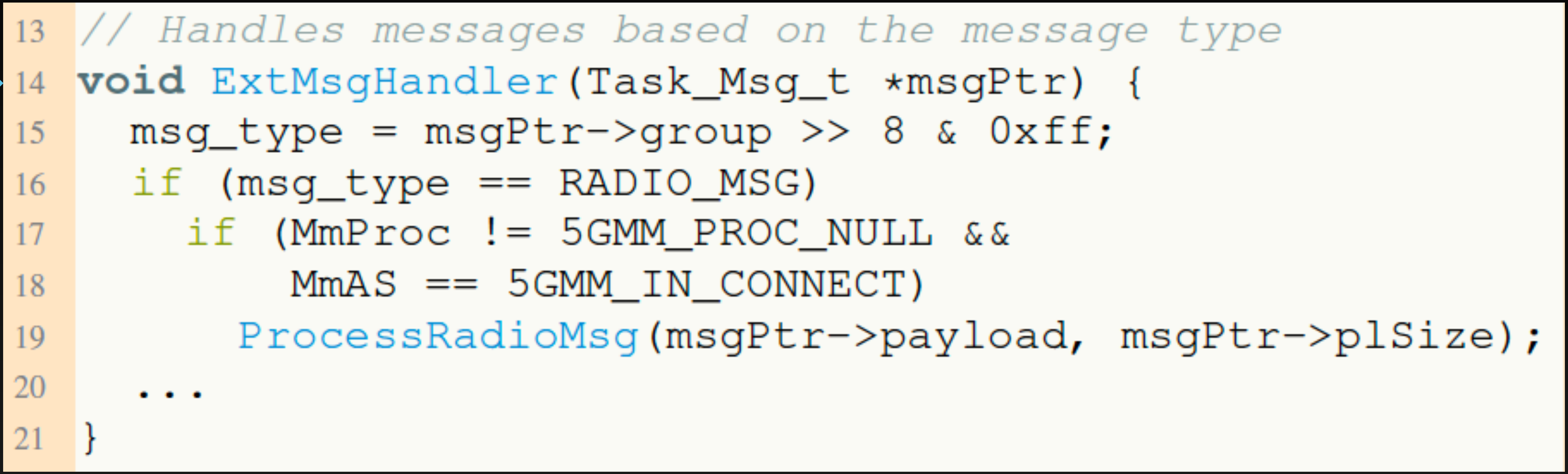
- Gradually increase symbolic variables
- Built upon previous results
- Ensures completed symbolic execution in each iteration

Demonstration of Iterative Symbolic Analysis



```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Iteration 1

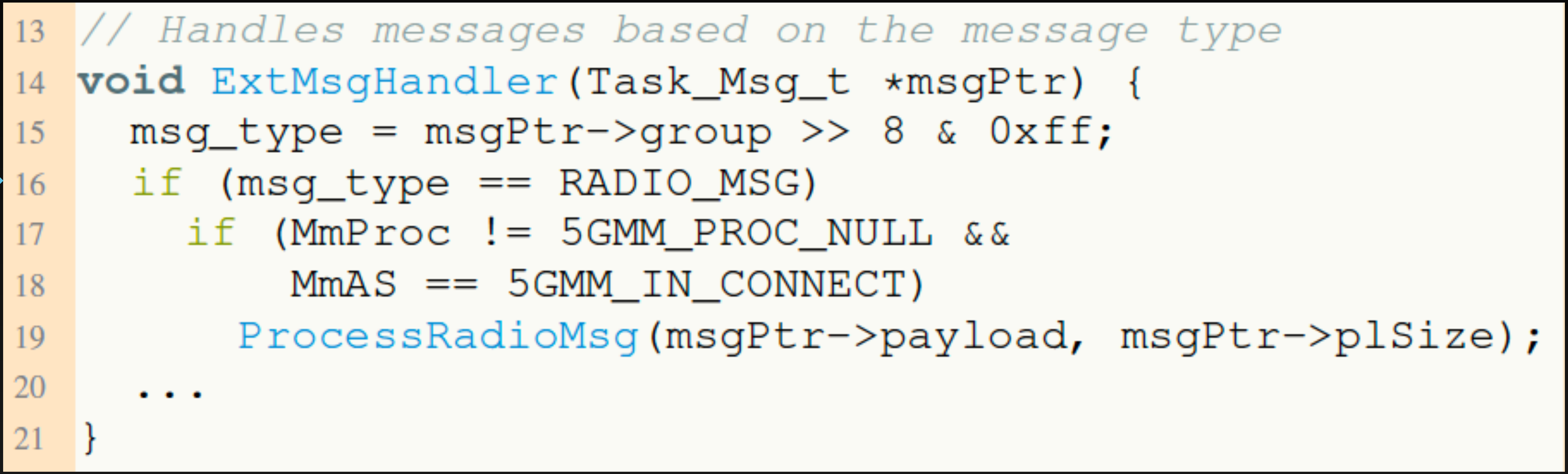


```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr}

State variables: {MmProc, MmAS}

Iteration 1

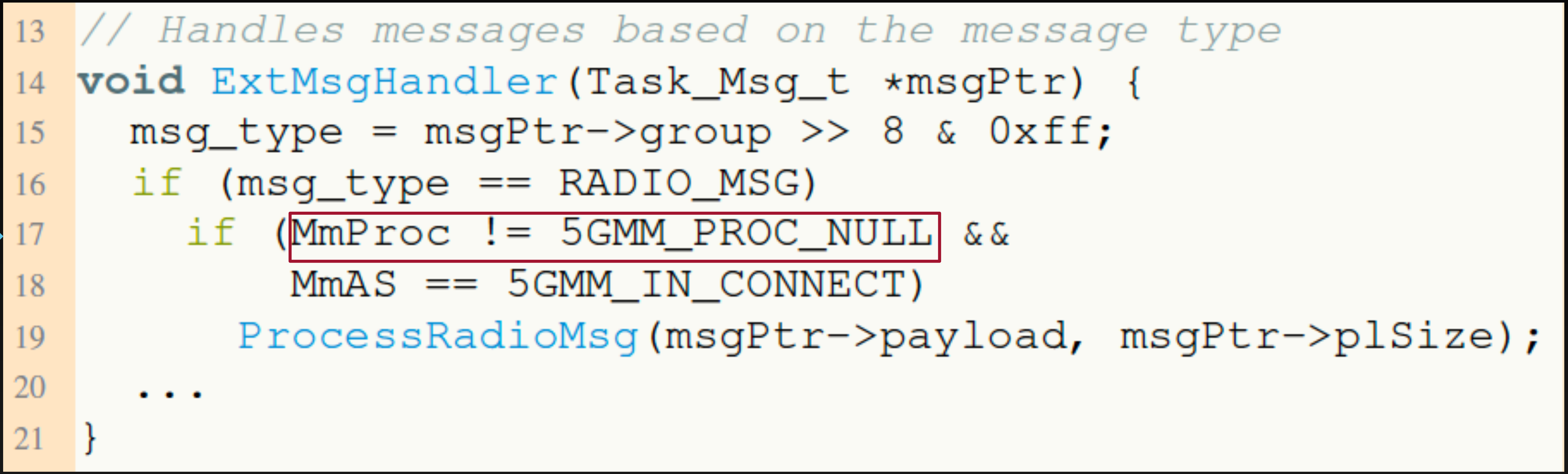


```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr}

State variables: {MmProc, MmAS}

Iteration 1

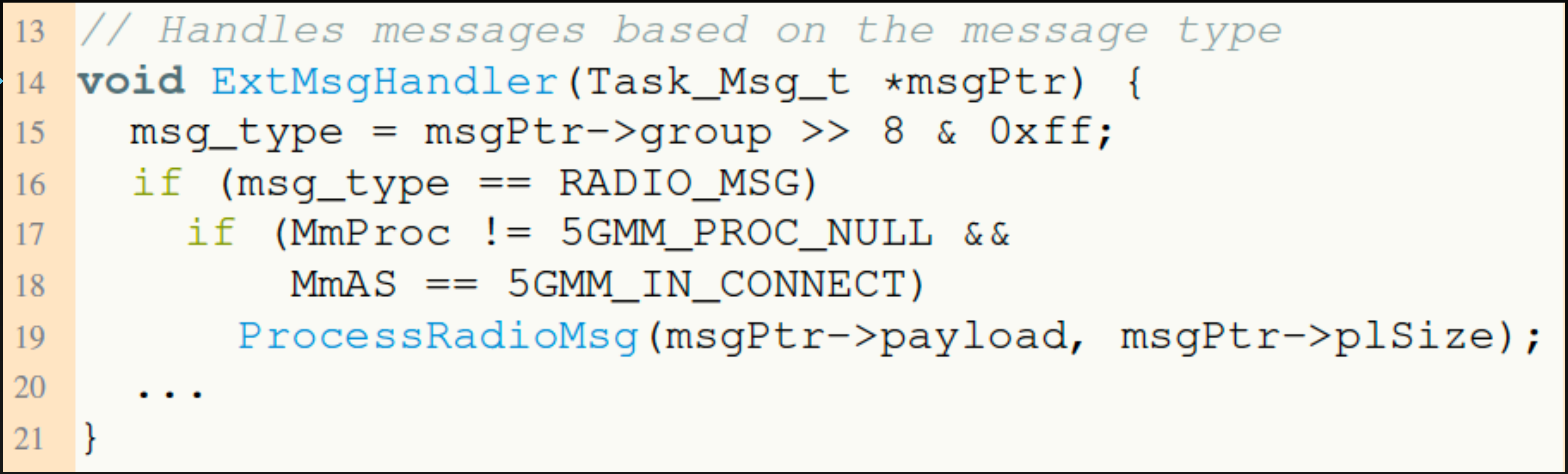


```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr}

State variables: {MmProc, MmAS}

Iteration 2

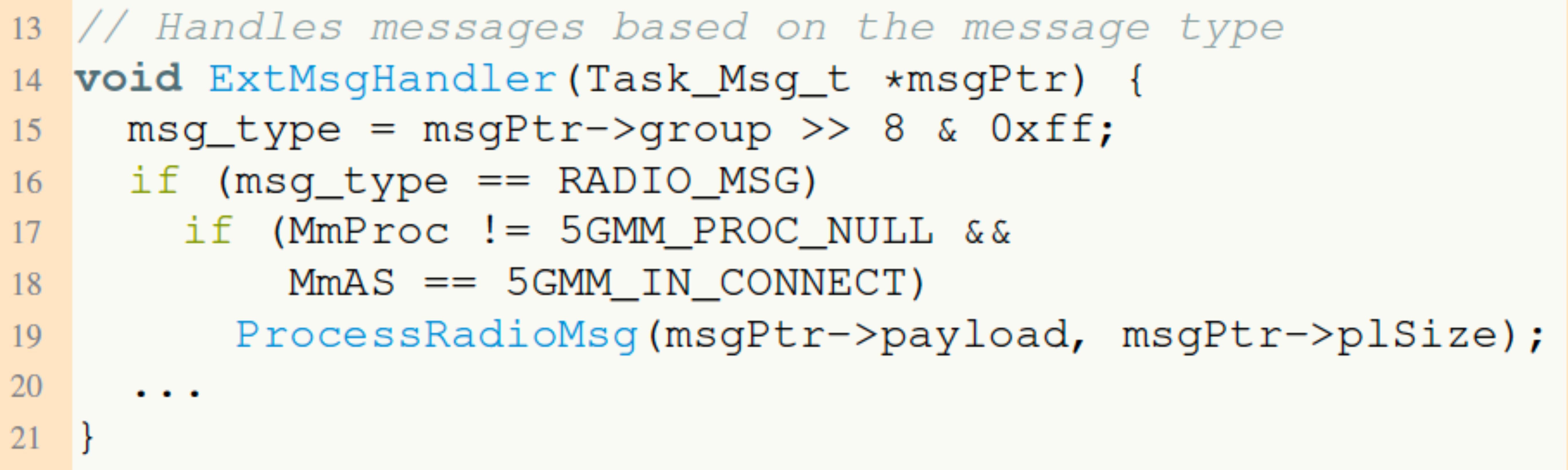


```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr}

State variables: {MmProc, MmAS}

Iteration 2

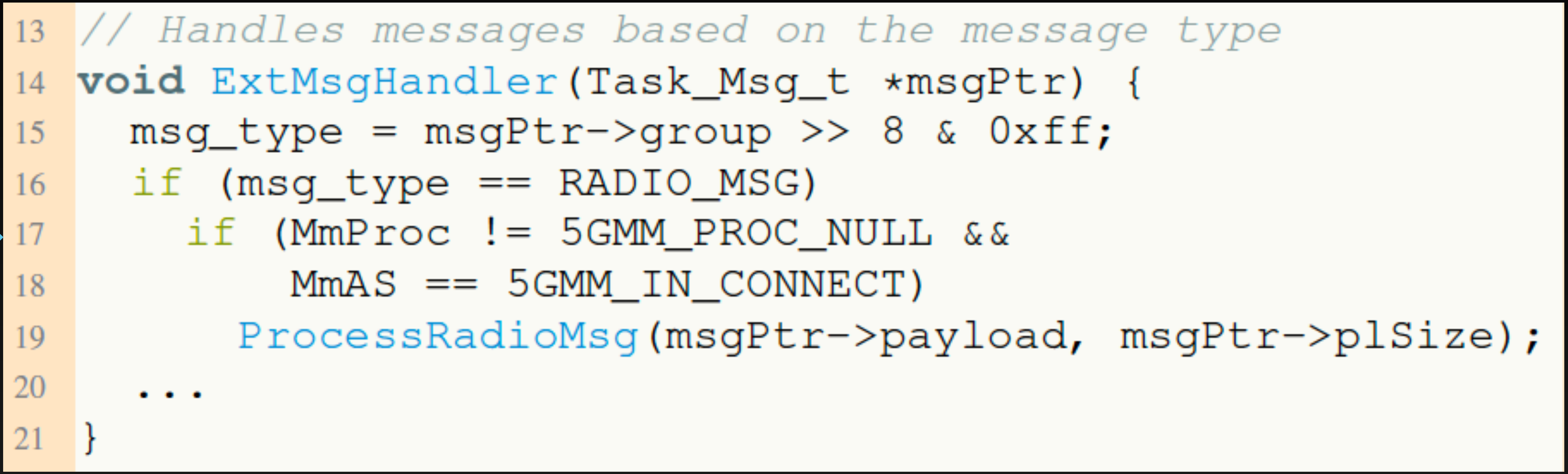


```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc}

State variables: {MmProc, MmAS}

Iteration 2

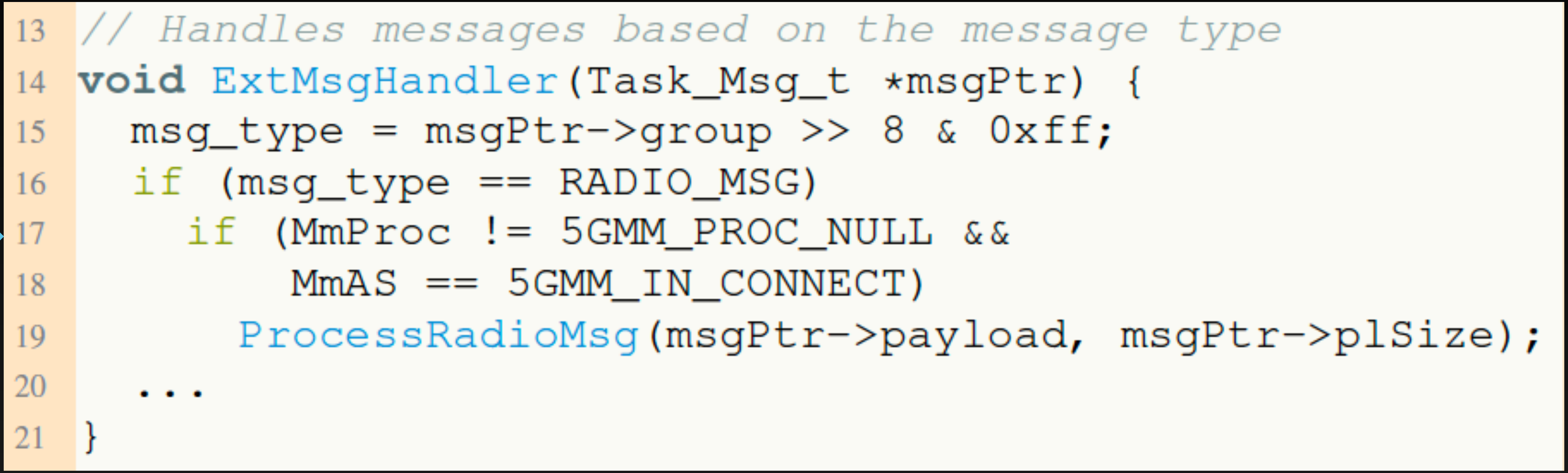


```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc}

State variables: {MmProc, MmAS}

Iteration 2



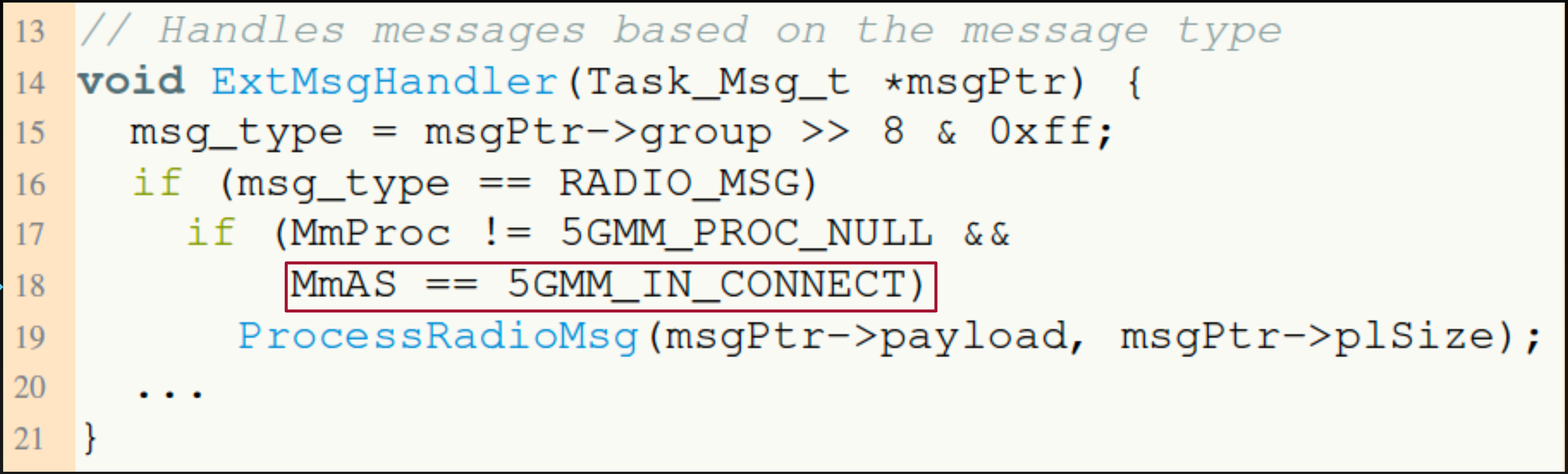
```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc}

State variables: {MmProc, MmAS}

Condition: MmProc != 5GMM_PROC_NULL

Iteration 2



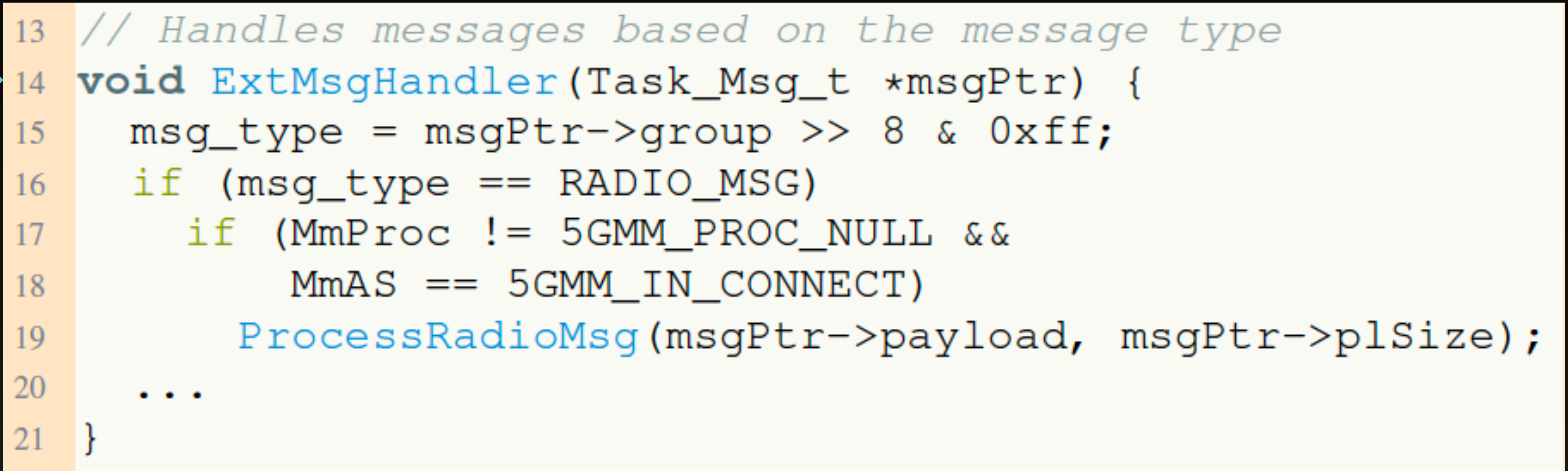
```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc}

State variables: {MmProc, MmAS}

Condition: MmProc != 5GMM_PROC_NULL

Iteration 3



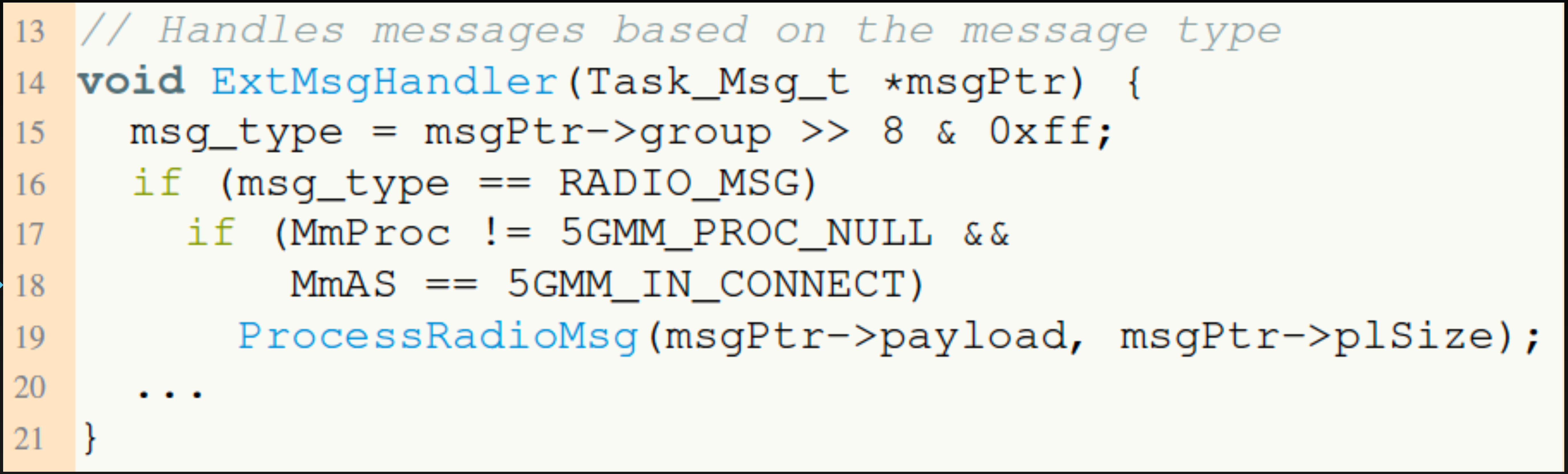
```c
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc, MmAS}

State variables: {MmProc, MmAS}

Condition: MmProc != 5GMM_PROC_NULL

Iteration 3



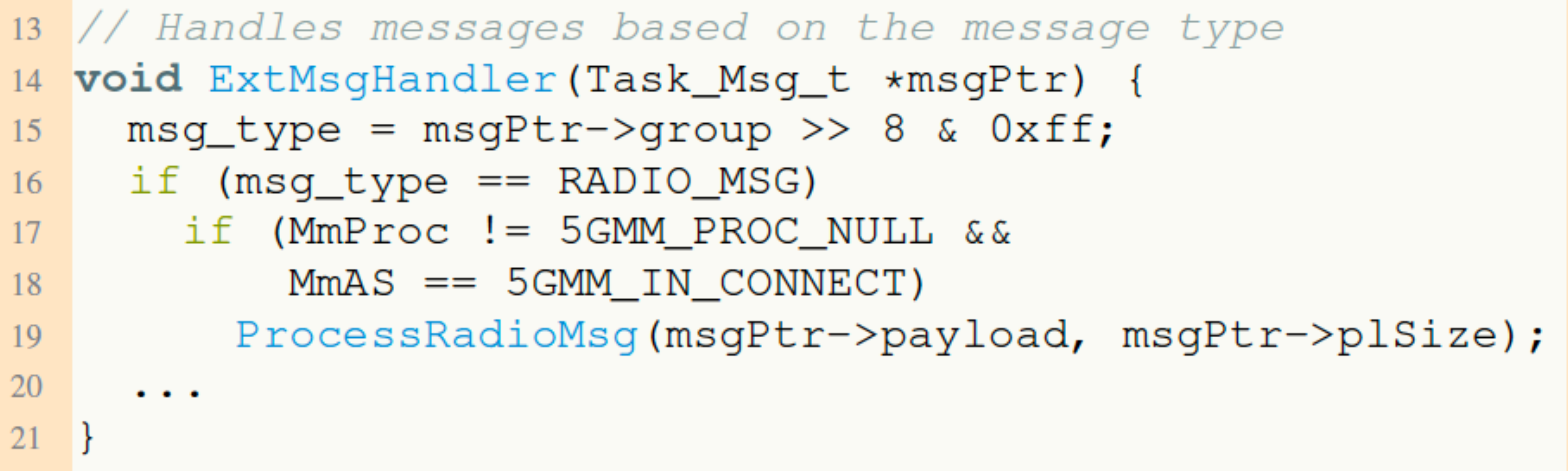
```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc, MmAS}

State variables: {MmProc, MmAS}

Condition: MmProc != 5GMM_PROC_NULL

Iteration 3



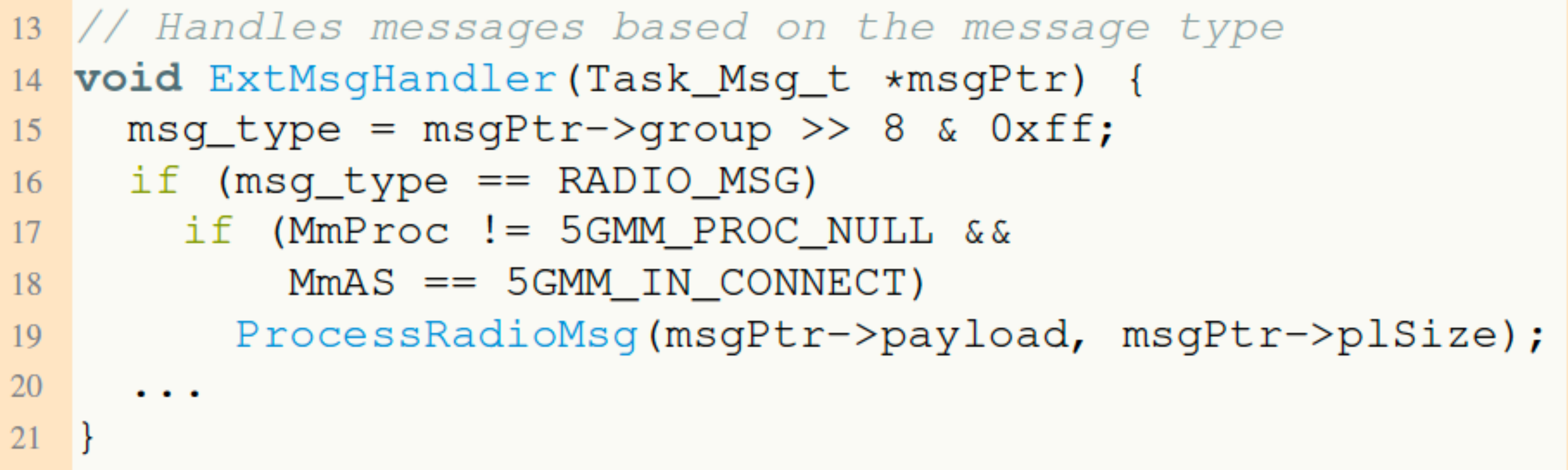
```
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc, MmAS}

State variables: {MmProc, MmAS}

Condition: MmProc != 5GMM_PROC_NULL,
MmAS == 5GMM_IN_CONNECT

Iteration 3



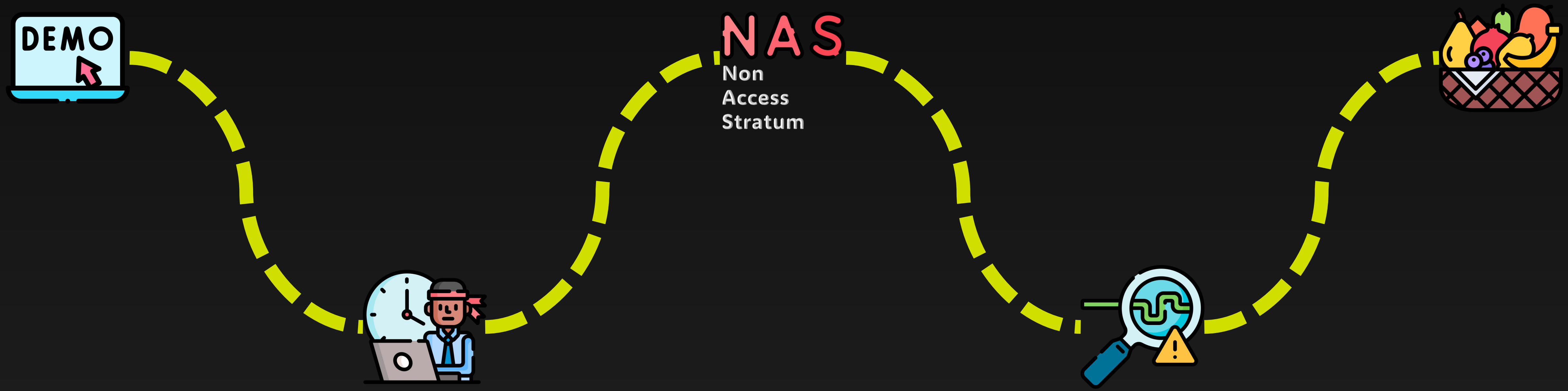
```c
// Handles messages based on the message type
void ExtMsgHandler(Task_Msg_t *msgPtr) {
  msg_type = msgPtr->group >> 8 & 0xff;
  if (msg_type == RADIO_MSG)
    if (MmProc != 5GMM_PROC_NULL &&
        MmAS == 5GMM_IN_CONNECT)
      ProcessRadioMsg(msgPtr->payload, msgPtr->plSize);
  ...
}
```

Symbolic variables: {msgPtr, MmProc, MmAS}

State variables: {MmProc, MmAS}

Condition: MmProc != 5GMM_PROC_NULL,
MmAS == 5GMM_IN_CONNECT

Let’s wrap it up!

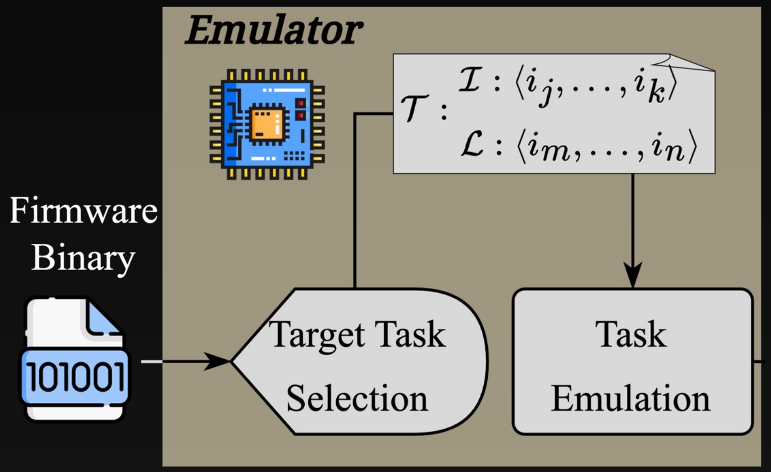


NAS
Non
Access
Stratum

Loris Architecture

Firmware Binary

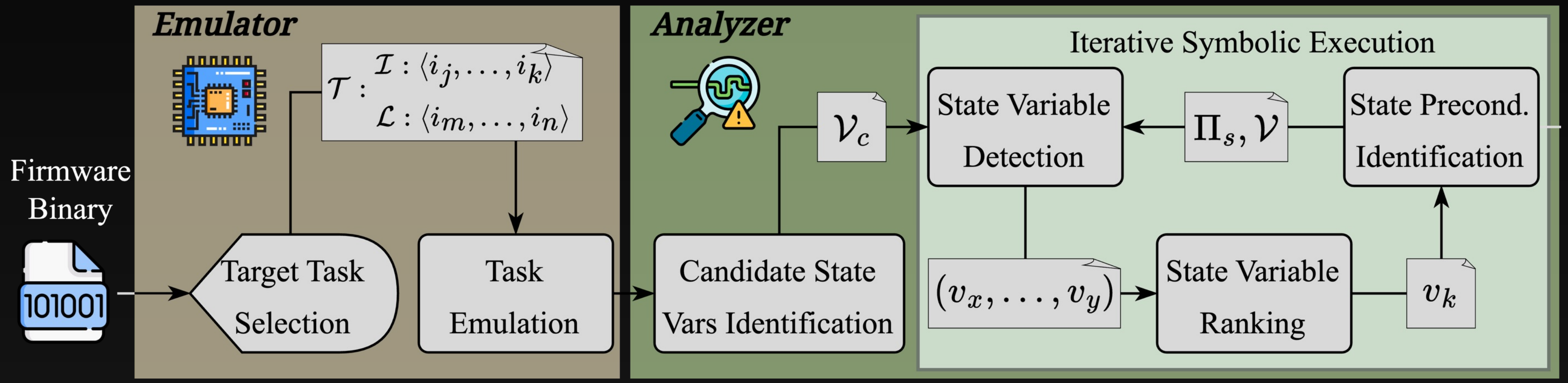
Loris Architecture


Emulator
$\mathcal{T}: \mathcal{I}: \langle i_j, \ldots, i_k \rangle$
$\mathcal{L}: \langle i_m, \ldots, i_n \rangle$
Firmware Binary
101001
Target Task Selection
Task Emulation

Emulator

- Based on FirmWire (NDSS’22)
- Added support for new 5G Exynos baseband.

Loris Architecture


Emulator
$\mathcal{T}:$ $\mathcal{I}:\langle i_j,\ldots,i_k\rangle$
$\mathcal{L}:\langle i_m,\ldots,i_n\rangle$
Firmware Binary
101001
Target Task Selection
Task Emulation
Analyzer
$\mathcal{V}_c$
Candidate State Vars Identification
Iterative Symbolic Execution
State Variable Detection
$\Pi_s,\mathcal{V}$
State Precond. Identification
$(v_x,\ldots,v_y)$
State Variable Ranking
v_k

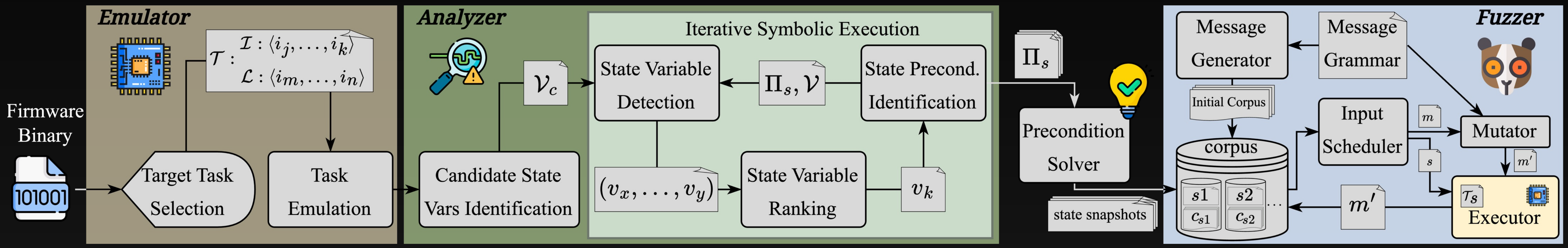
Emulator

- Based on FirmWire (NDSS’22)
- Added support for new 5G Exynos baseband.

Iterative symbolic analysis

- Sate variables detection
- State variable analysis
- Checkpoint-based path pruning

Loris Architecture



Emulator

- Based on FirmWire (NDSS'22)
- Added support for new 5G Exynos baseband.

Iterative symbolic analysis

- Sate variables detection
- State variable analysis
- Checkpoint-based path pruning

Grammar-aware fuzzing

- No seeds are required
- Grammar-aware mutations
- Target task state initialization

Vulnerability Discovery

- Developed a unified harness that accepts any message type with target state from our LibAFL-based fuzzer.

- The harness automatically initializes the target task and delivers the message via baseband APIs.

- We fuzzed 4G NAS (SAEL3) and 5G NAS (NASOT)
  - Samsung Galaxy S21, S20, S10, A41
  - Google Pixel 6

Discovered Seven 0-Days

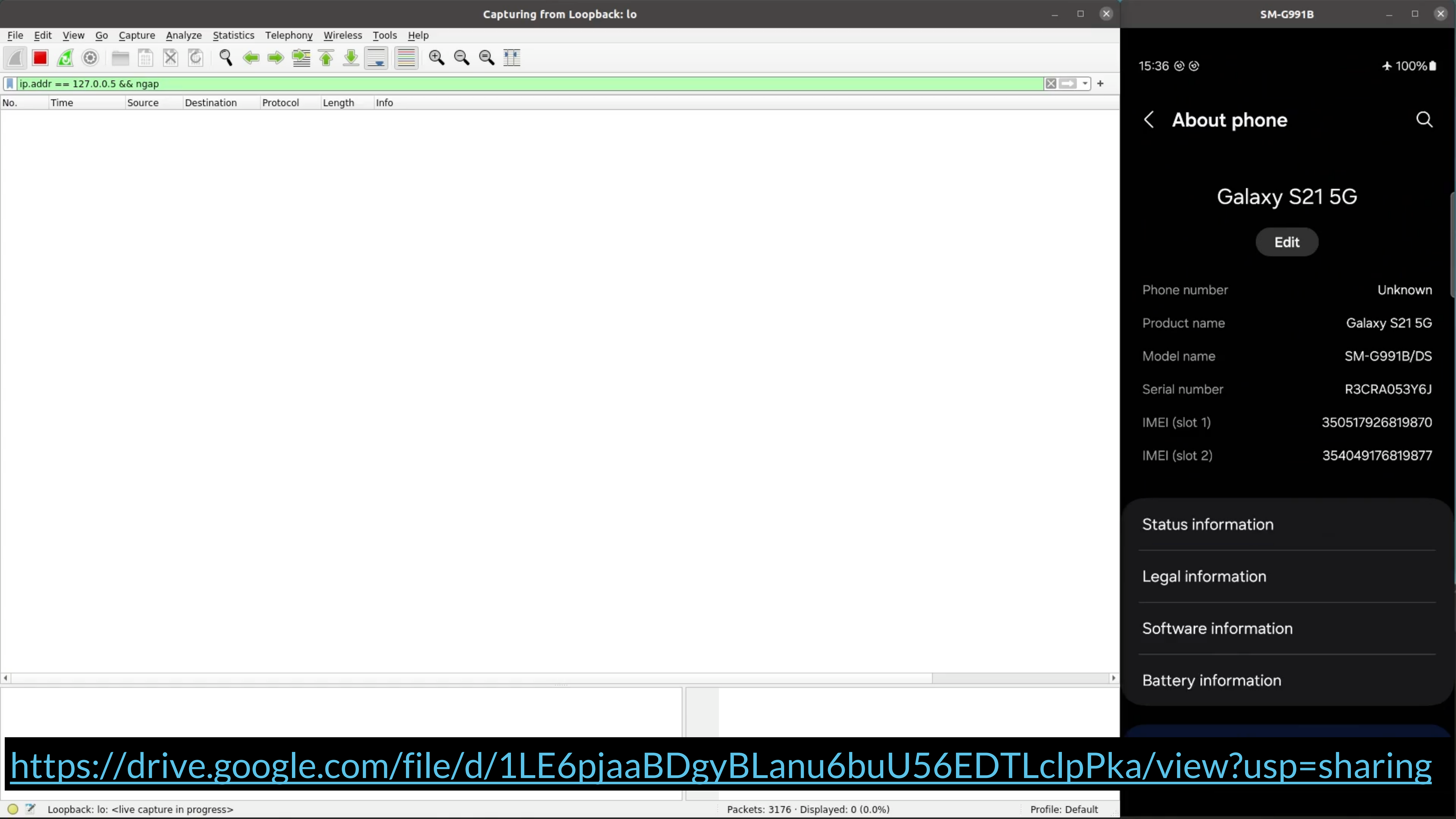
- We fuzzed 4G NAS (SAEL3) and 5G NAS (NASOT)
  - Samsung Galaxy S21, S20, S10, A41
  - Google Pixel 6

- **Discovered 7 crashes, all of which were previously unknown!**
  - 5G NAS: 1 critical, 2 high, 3 moderate, 1 low
  - 4G NAS: 1 additional heap overflow but unexploitable!

- **5 CVEs:** CVE-2024-52923, CVE-2024-52924, CVE-2025-26784, CVE-2025-26785, and CVE-2025-27891.

OTA Crash Reproduction

- Used a USRP B210 with OpenAirInterface.
- Modified Open5GS as the malicious core network.
- The basebands crashed with each message.



ip.addr == 127.0.0.5 && ngap

No. Time Source Destination Protocol Length Info

Loopback: lo: <live capture in progress>

Packets: 3176 · Displayed: 0 (0.0%)

Profile: Default

About phone

Galaxy S21 5G

Phone number	Unknown
Product name	Galaxy S21 5G
Model name	SM-G991B/DS
Serial number	R3CRA053Y6J
IMEI (slot 1)	350517926819870
IMEI (slot 2)	354049176819877

Status information

Legal information

Software information

Battery information

https://drive.google.com/file/d/1LE6pjaaBDgyBLanu6buU56EDTLclpPka/view?usp=sharing

Real World Impact

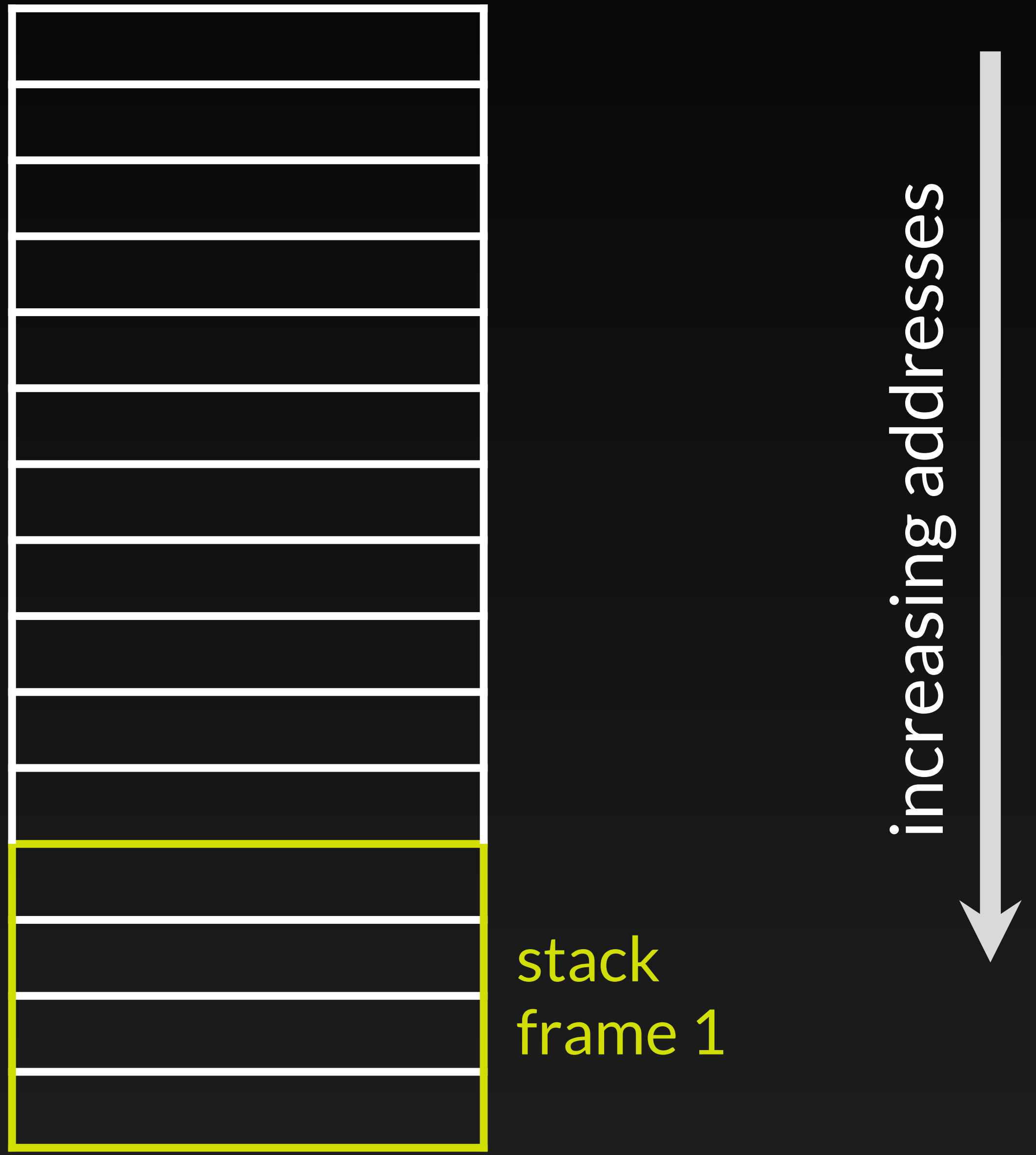
- Discovered 0-days: stack overflow and heap overflow.

- Requirements of turning stack overflow to RCE vector:

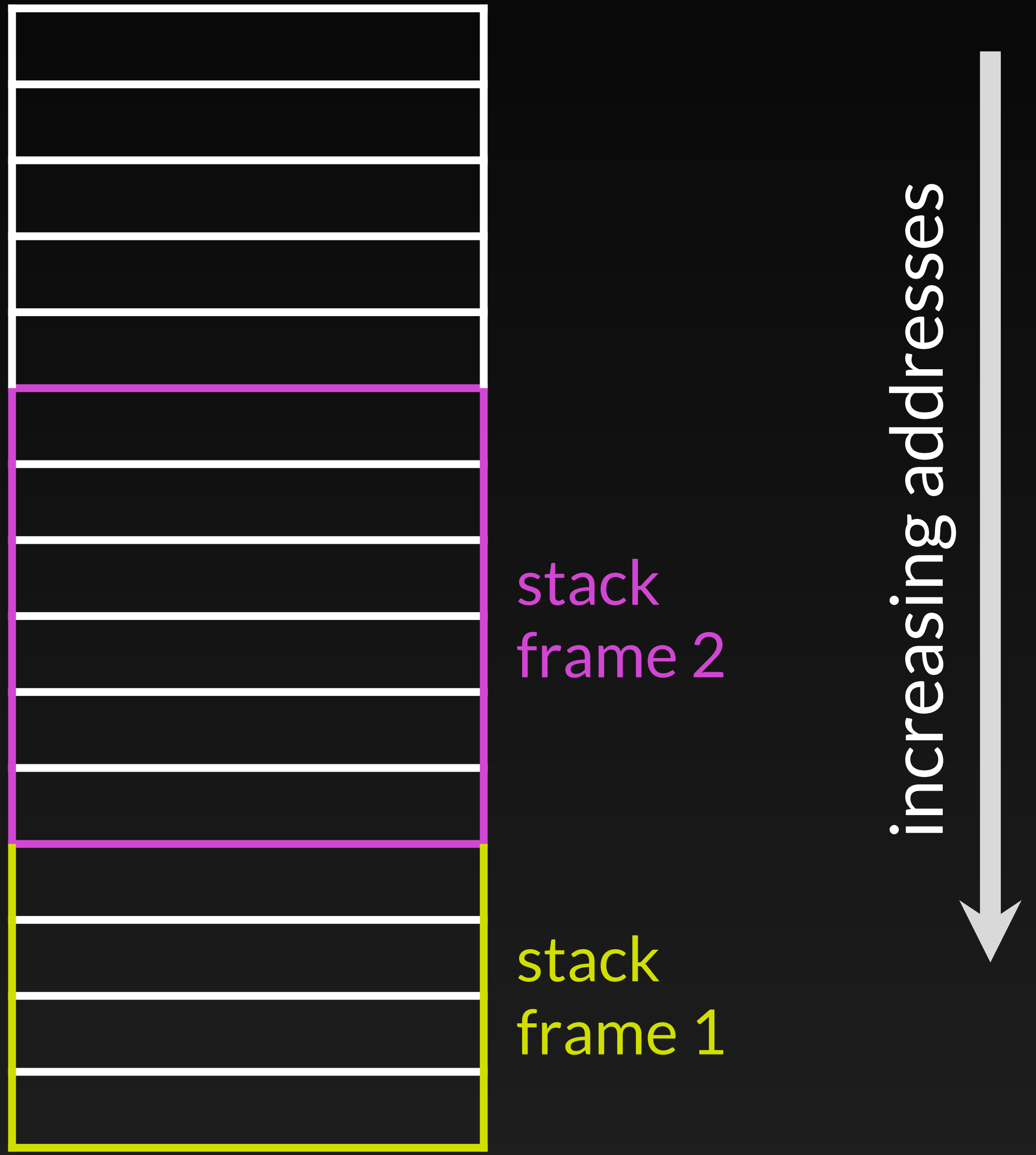
  1. RWX stack – eXecute Never bit must be 0
  2. No stack protection – sleepy canaries

- Heap overflow can still lead to RCE; might be limited to small payloads.
  - A better gain: write-what-where primitive

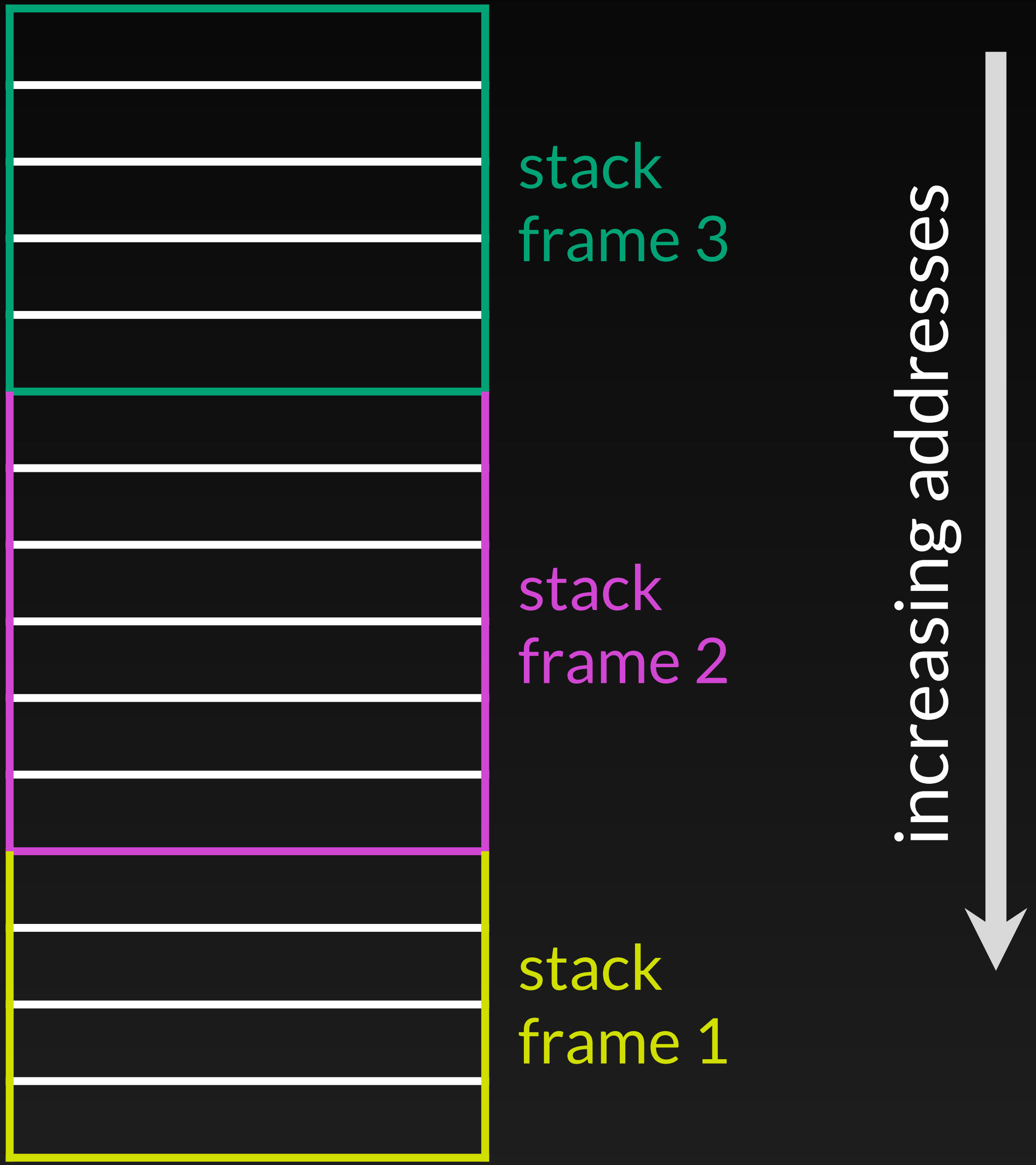
Stack Canaries 🐣



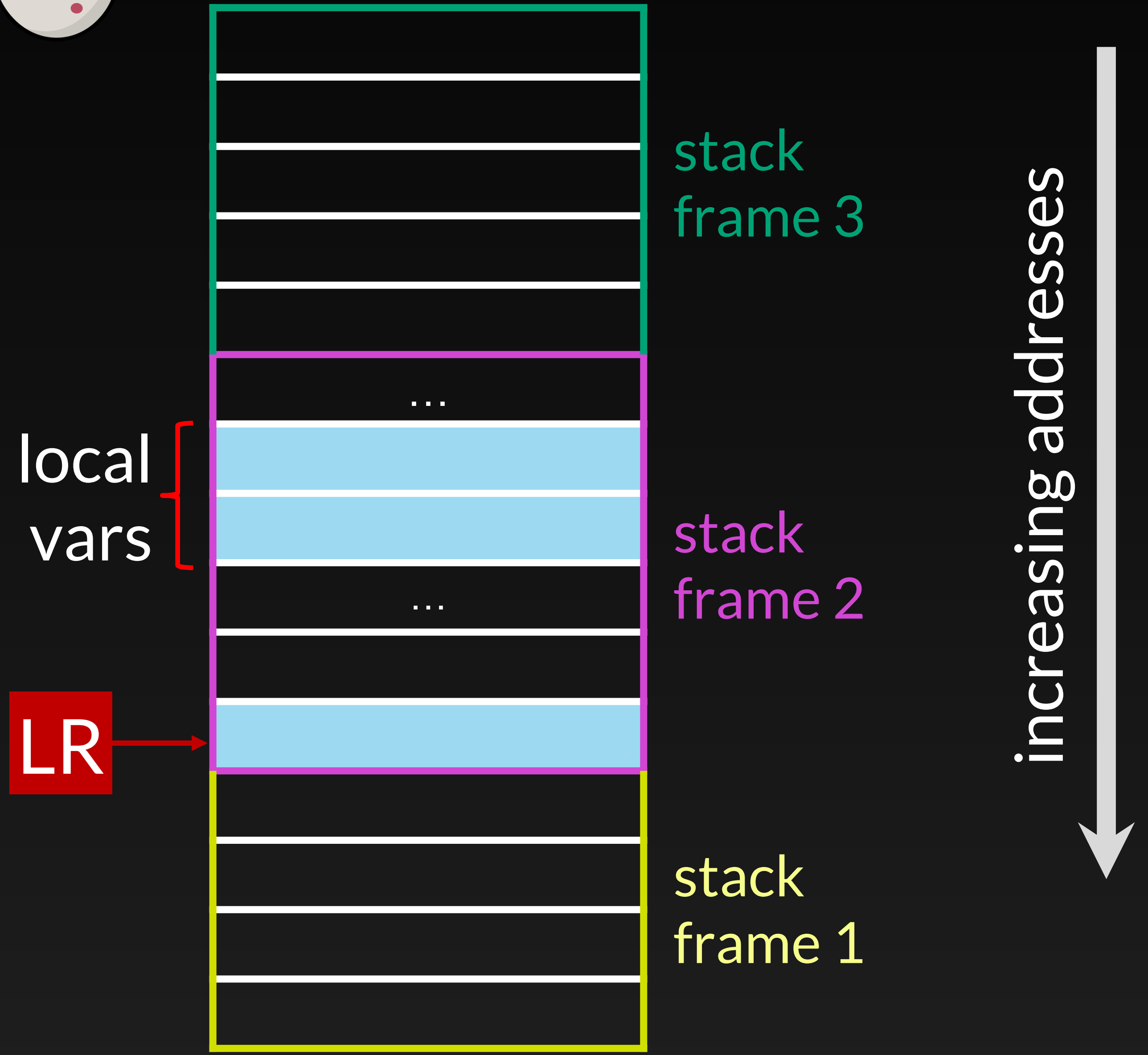
Stack Canaries 🐣



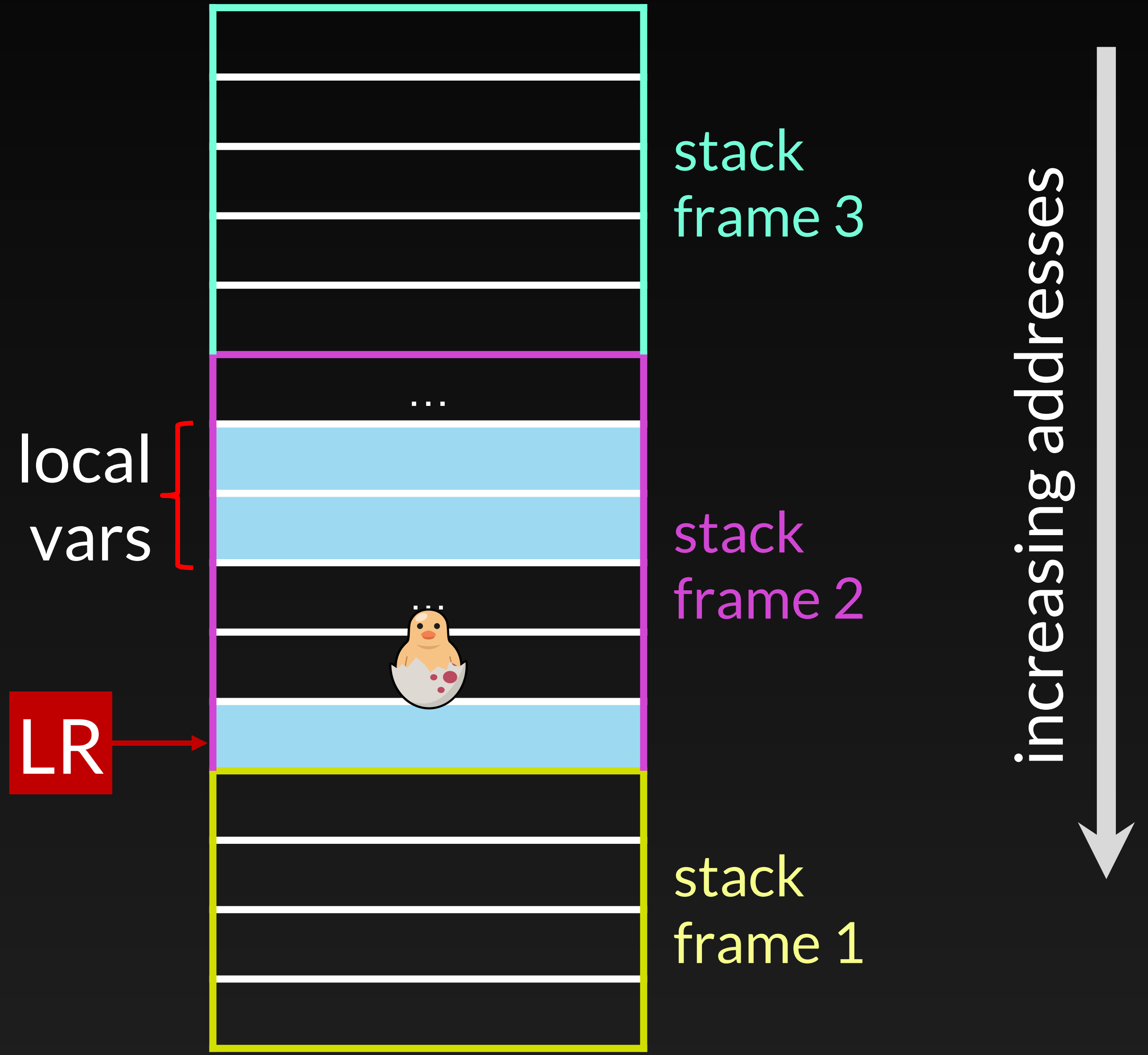
Stack Canaries 🐣



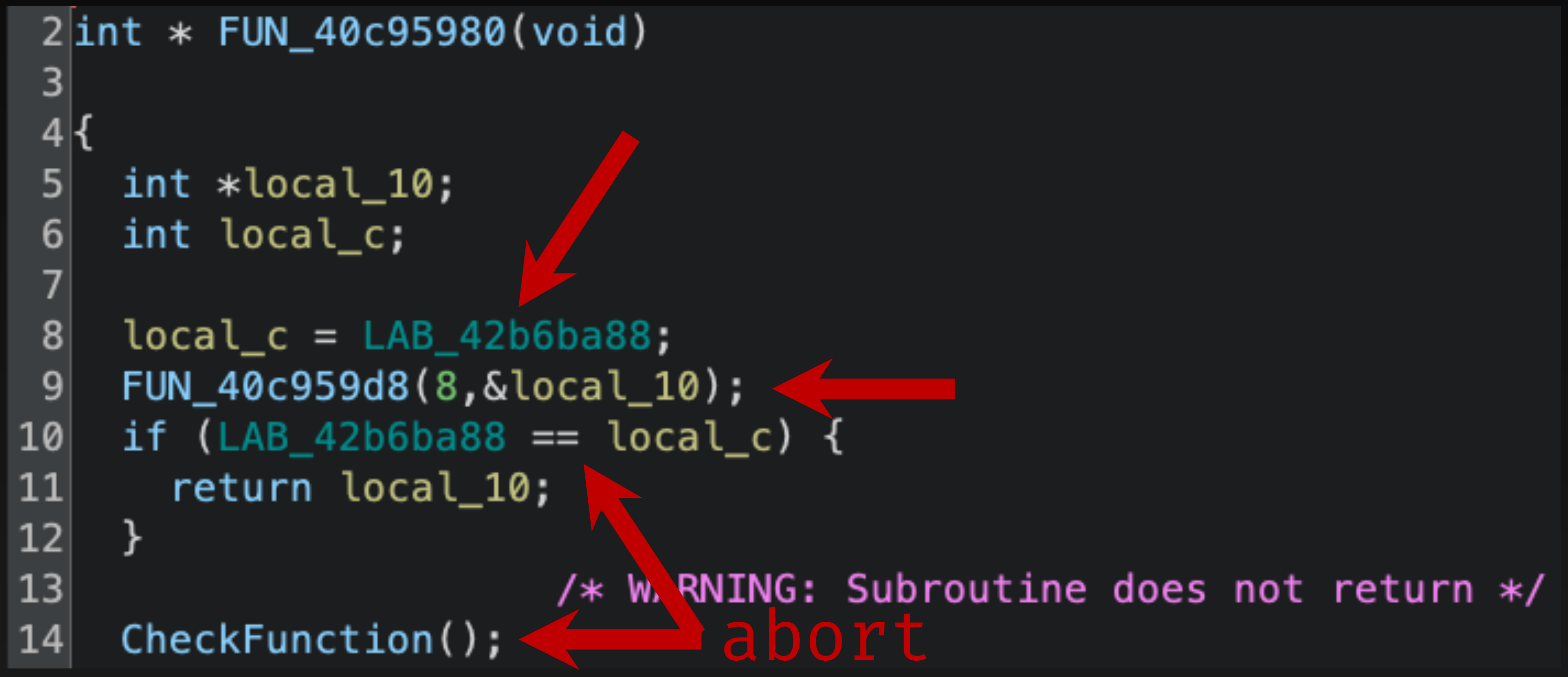
Stack Canaries 🐣



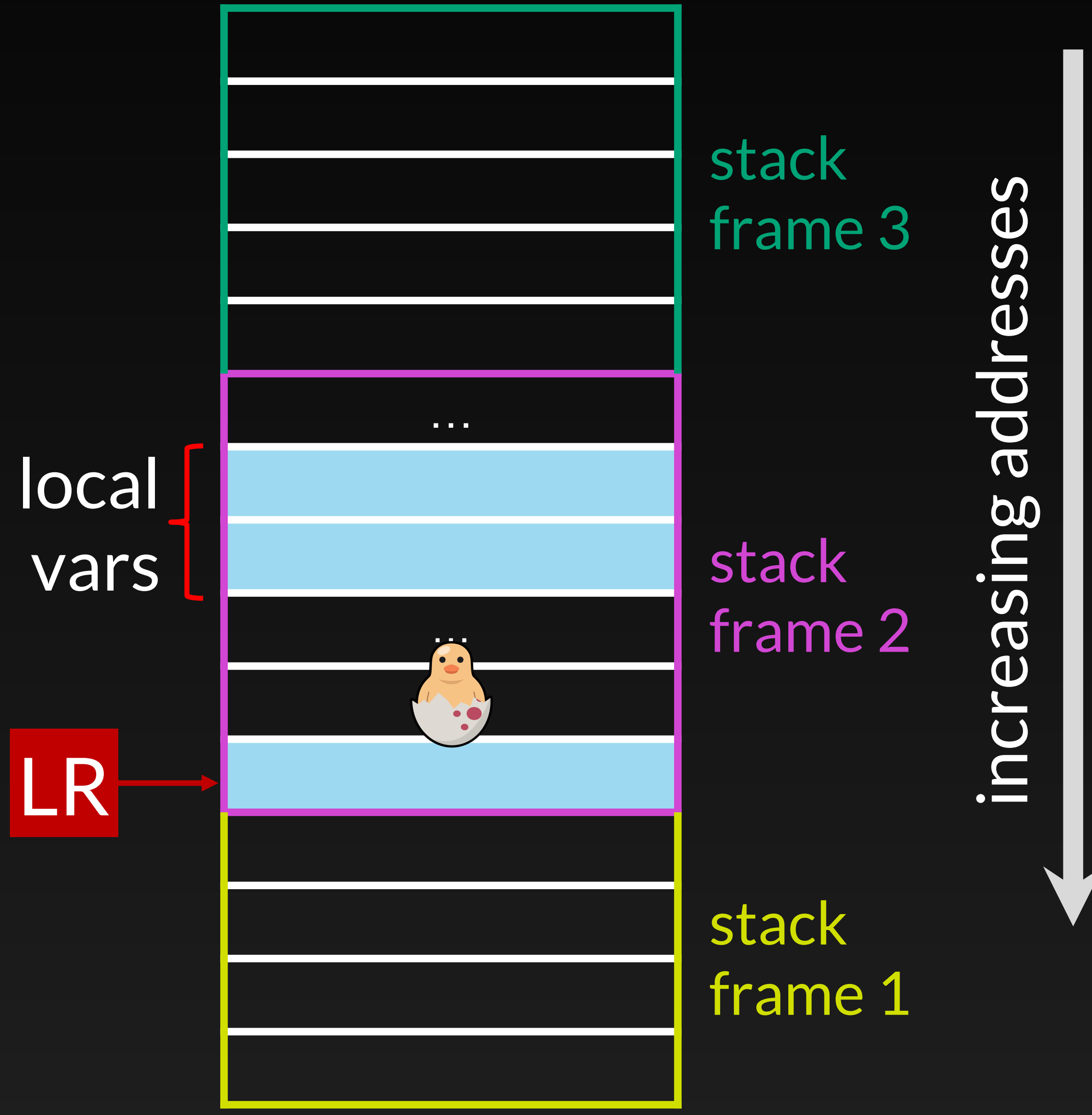
Stack Canaries



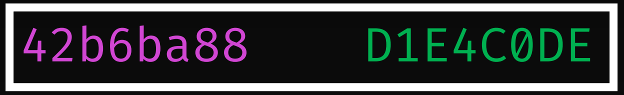
Stack Canaries



```
int * FUN_40c95980(void)

{
  int *local_10;
  int local_c;

  local_c = LAB_42b6ba88;
  FUN_40c959d8(8,&local_10);
  if (LAB_42b6ba88 == local_c) {
    return local_10;
  }
                    /* WARNING: Subroutine does not return */
  CheckFunction();
```

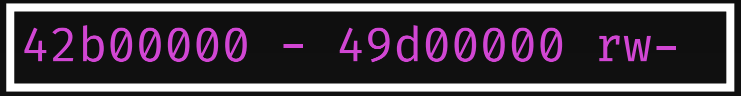


Stack Canaries

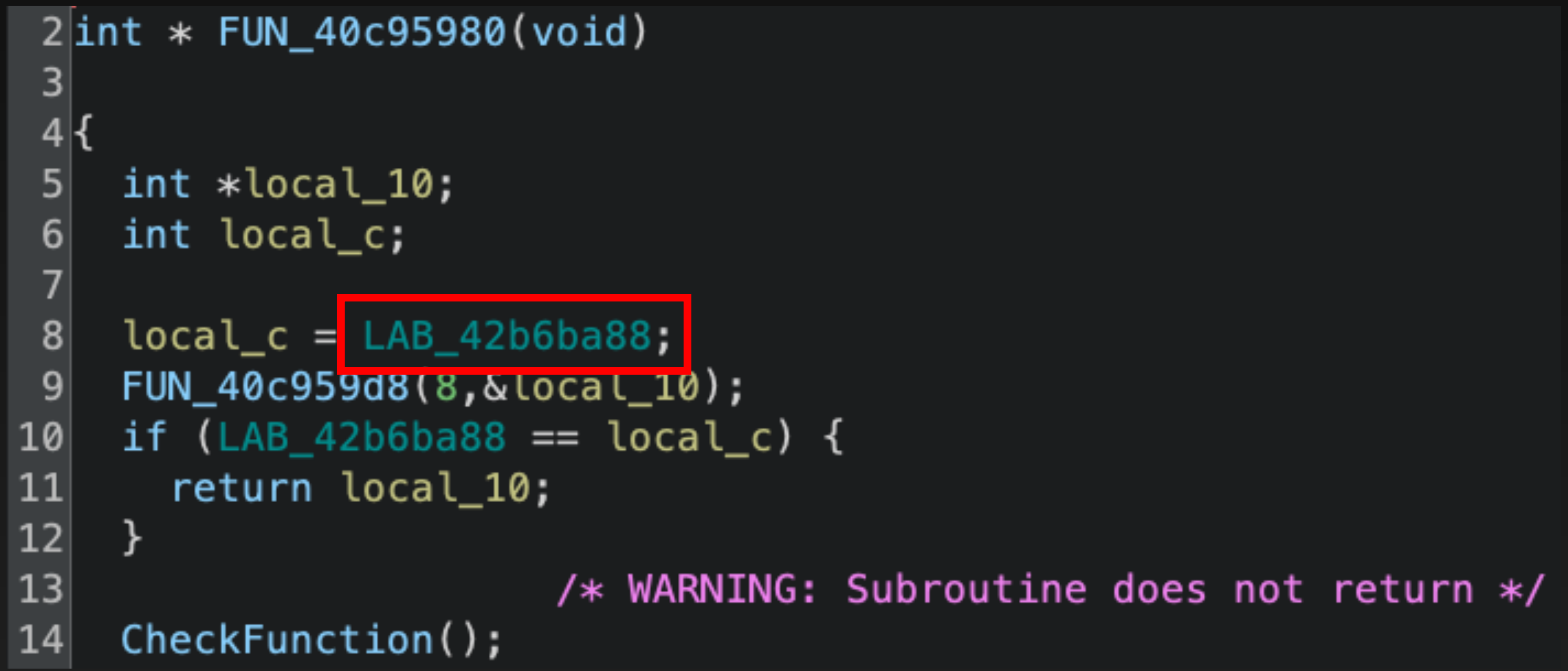


```
42b6ba88      D1E4C0DE
```

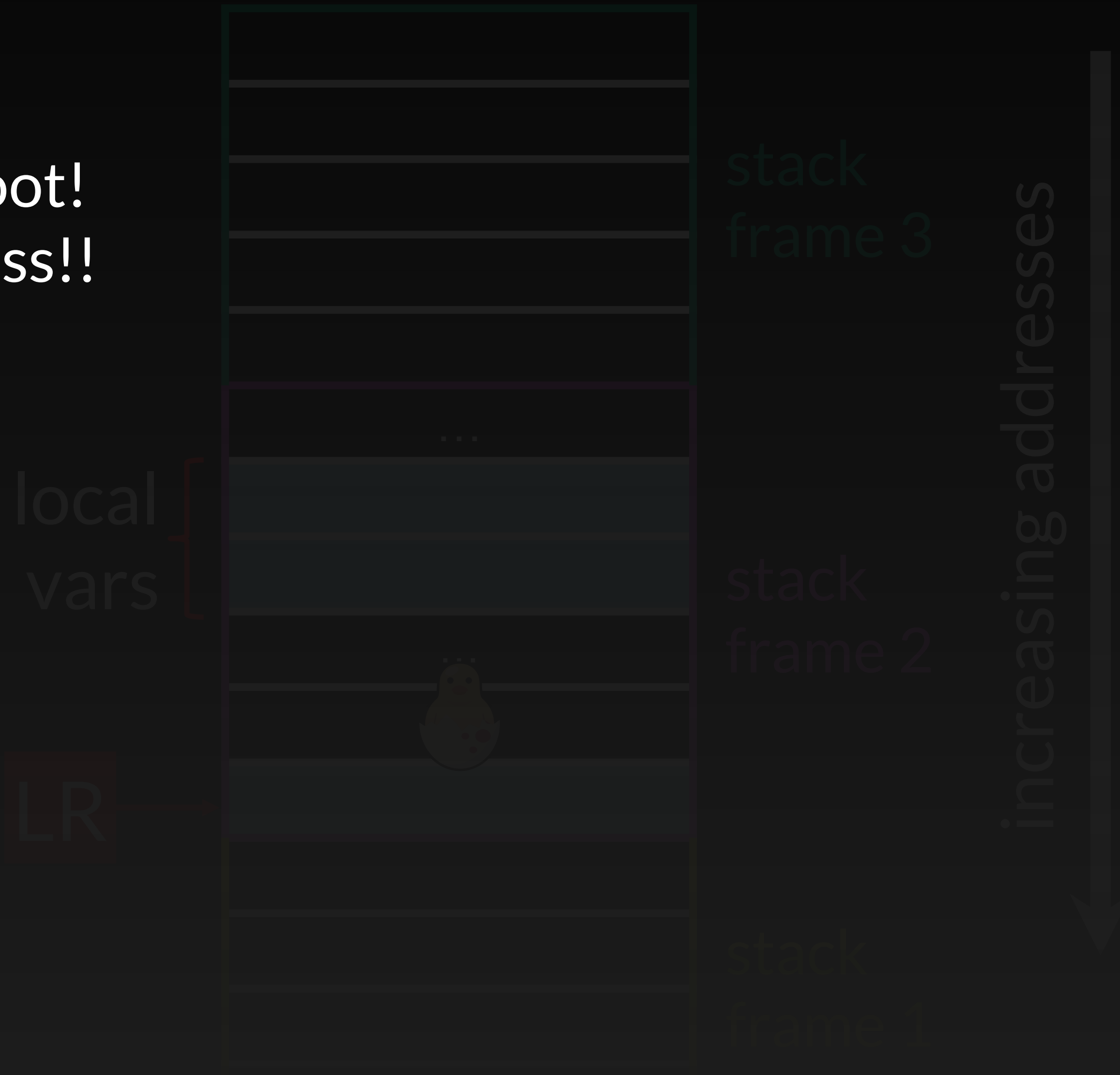
- Hexspeak for “Die for Code”
- Changed to a random integer during boot!
- Lives in a memory page with write access!!



```
42b00000 - 49d00000 rw-
```



```
int * FUN_40c95980(void)

{
  int *local_10;
  int local_c;

  local_c = LAB_42b6ba88;
  FUN_40c959d8(8,&local_10);
  if (LAB_42b6ba88 == local_c) {
    return local_10;
  }
                    /* WARNING: Subroutine does not return */
  CheckFunction();
```



From buffer overflow to RCE

- Heap overflow can yield a clean write-what-where primitive.
  - Black Hat USA ’23

black hat
USA 2023

AUGUST 9-10, 2023
BRIEFINGS

Over the Air, Under the Radar

Attacking and Securing the Pixel Modem

Xuan Xing   Eugene Rodionov   Xiling Gong   Farzan Karimi

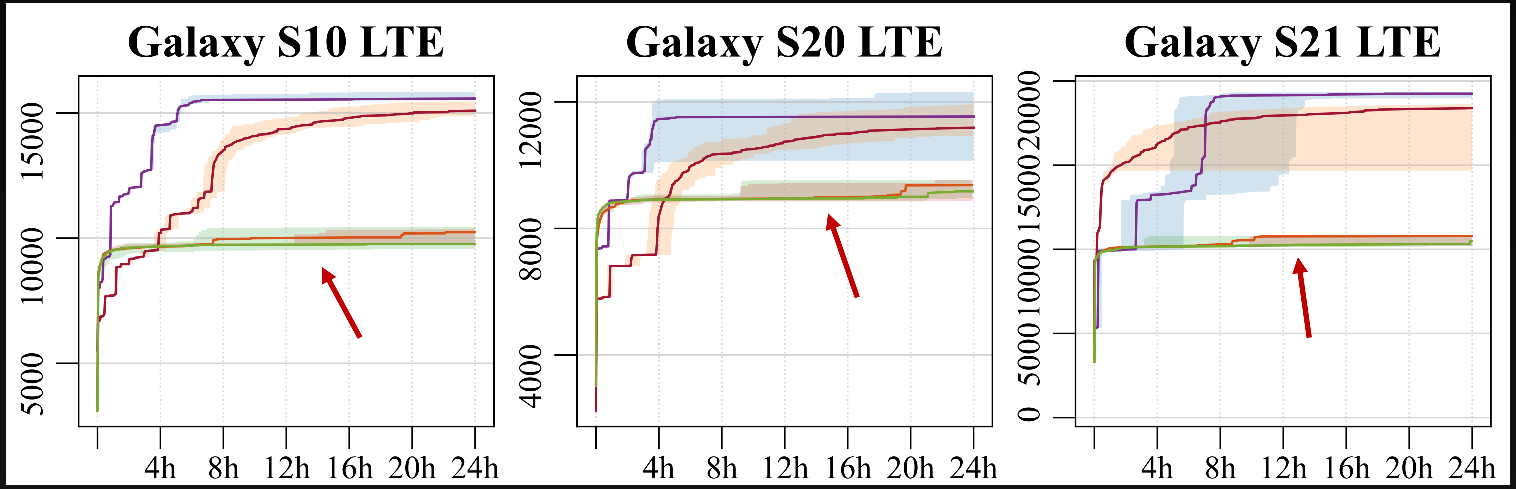
#BHUSA @BlackHatEvents

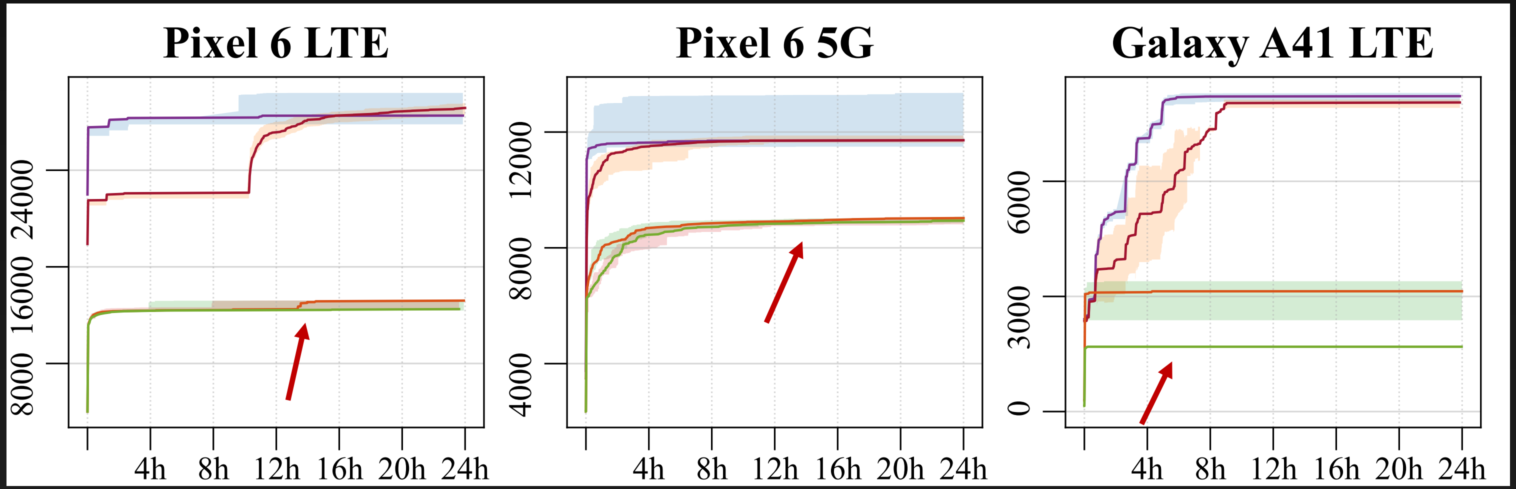
- But requires an RWX to obtain RCE.

- Return Oriented Programming (ROP) is the solution!

- Example exploit can set the `NRMM.FAKE_TEST_ENABLE` flag in the NV RAM.

Loris Covers ~200% Code





Parting Thoughts

- The complexity of baseband are increasing due to generation shifts, added functionalities, and new peripherals.

- However, automated systematic analysis using insights gained from understanding these firmware leads to efficient analysis and better results.

- *Complexity ≠ Better Security*

- More research is needed for baseband security (i.e., more protocols).

Thank You & Questions

Code

Paper

- Kai Tu, Saaman Khalilollahi, Kanika Gupta, Syed Rafiul Hussain
- Samsung Mobile Security
- Google Android Security

Ali Ranjbar

aranjbar@psu.edu

aranjbar.me

Tianchang Yang

tzy5088@psu.edu

tianchang-yang.github.io