From Blue Boxes to Black Boxes: Adventures in Uncovering Mobile Device Functionality

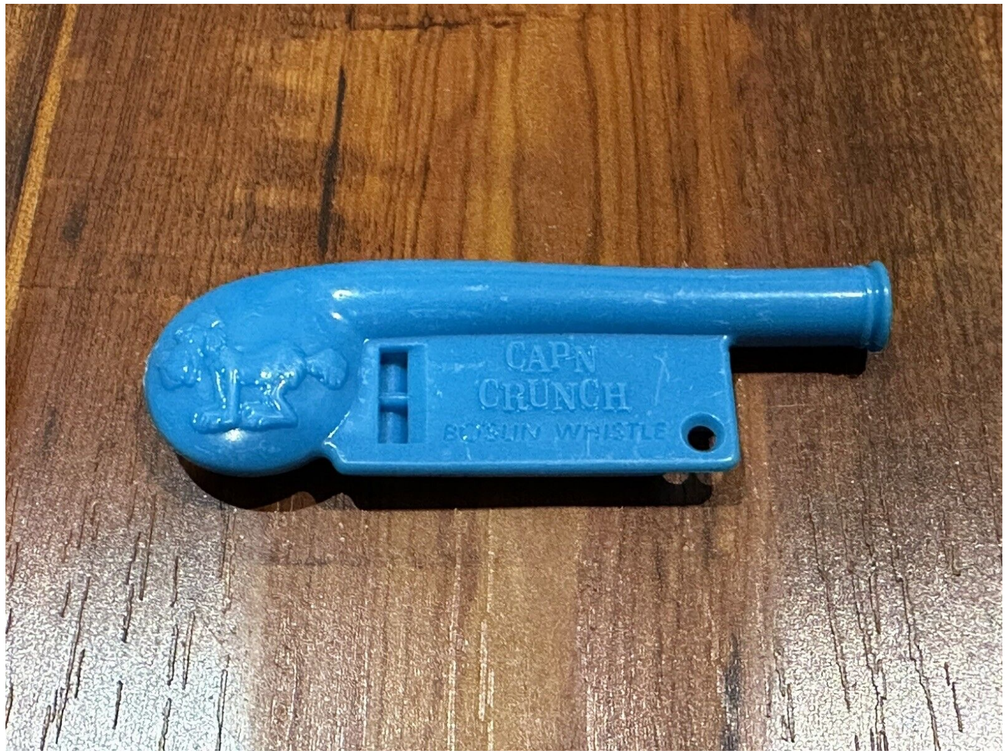
Kevin Butler

WOOT 2023

San Francisco, CA

May 25, 2023

Phone Phreaking



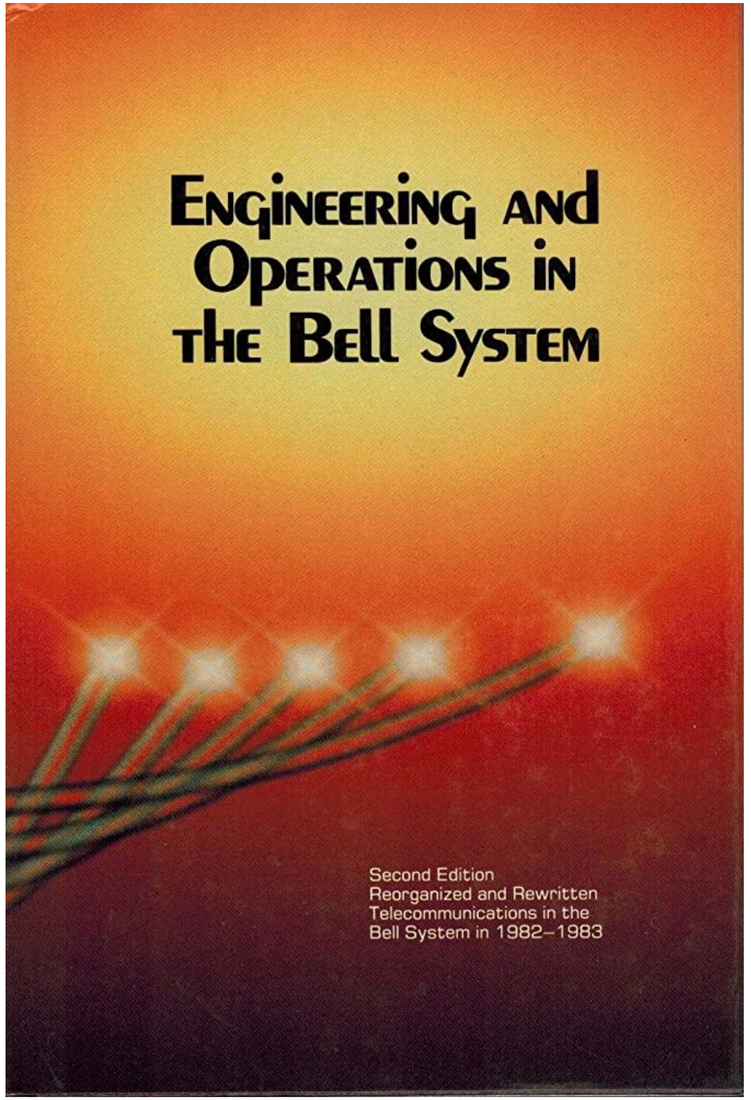
Blue Boxes

Credit: Maksym Kozlenko. Creative Commons

Credit: Joe Melena. ©Apple Computer

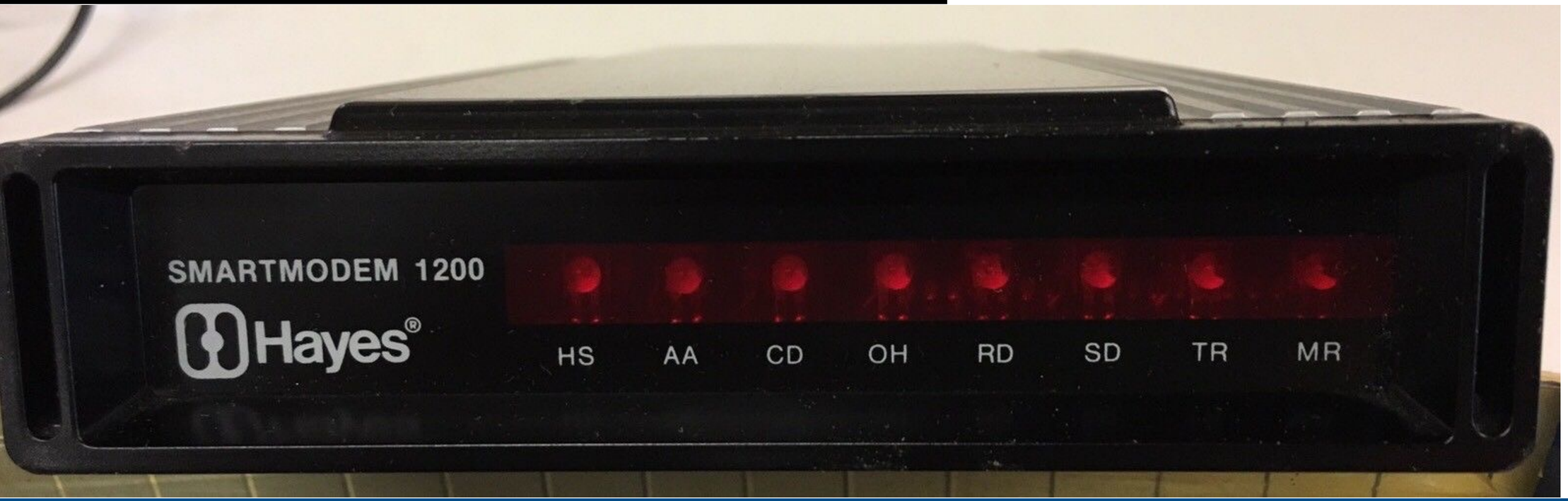
Why Telco?

- One of history's most important advances in human communication but very little information accessible about these networks
- In-band signal tones only became known through inadvertent disclosure in a Bell Systems journal
- Modern day telephony shares some of this obscurity, but complexity has also moved to the edge
- How do we uncover this functionality?

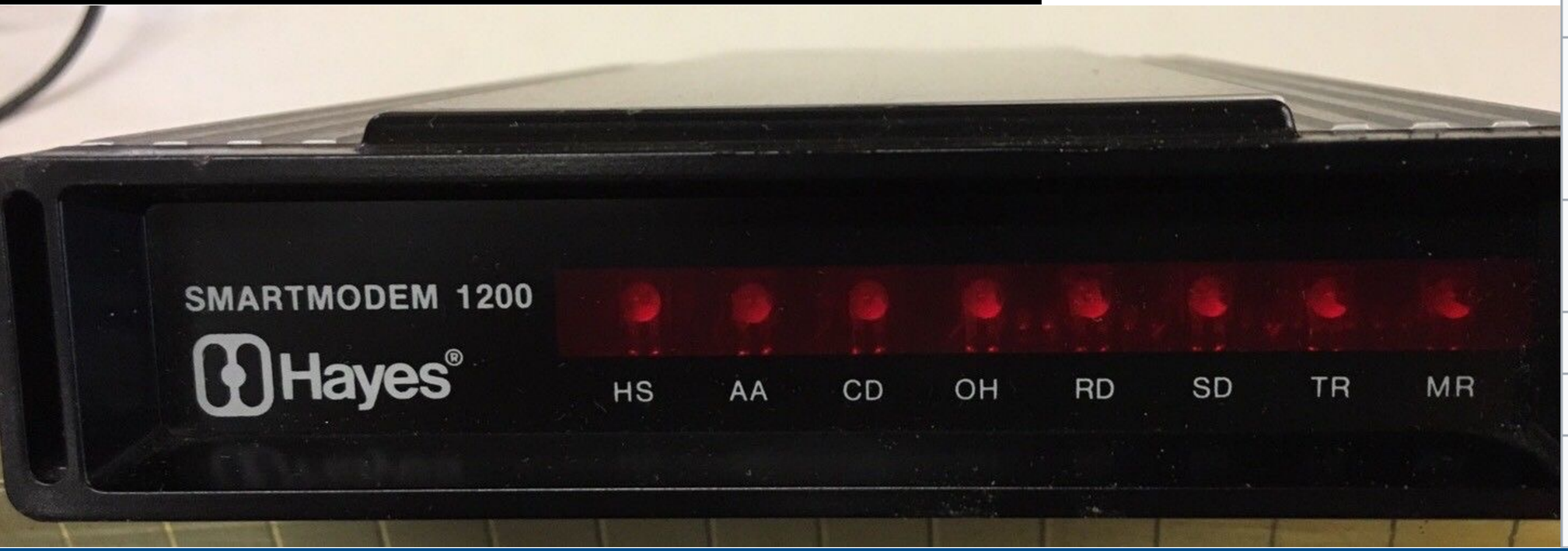


AT Commands


C-NET
BULLETIN BOARD SYSTEM
VERSION 11.1 NOVEMBER 24. 1986
(C) 1986 BY PERSPECTIVE SOFTWARE
THE ULTIMATE BULLETIN BOARD SYSTEM
AVAILABLE FOR THE C-64 MICROCOMPUTER


SMARTMODEM 1200
Hayes®
HS
AA
CD
OH
RD
SD
TR
MR

AT Commands



Modem A	Modem B
ATDT15551234	
	RING
	ATA
CONNECT	*CONNECT*
abcdef	*abcdef*
	+++
	OK
	ATH
NO CARRIER	*OK*

AT Commands

Modem A	Modem B
ATDT15551234	
	RING
	ATA
CONNECT	*CONNECT*
abcdef	*abcdef*
	+++
	OK
	ATH
NO CARRIER	*OK*

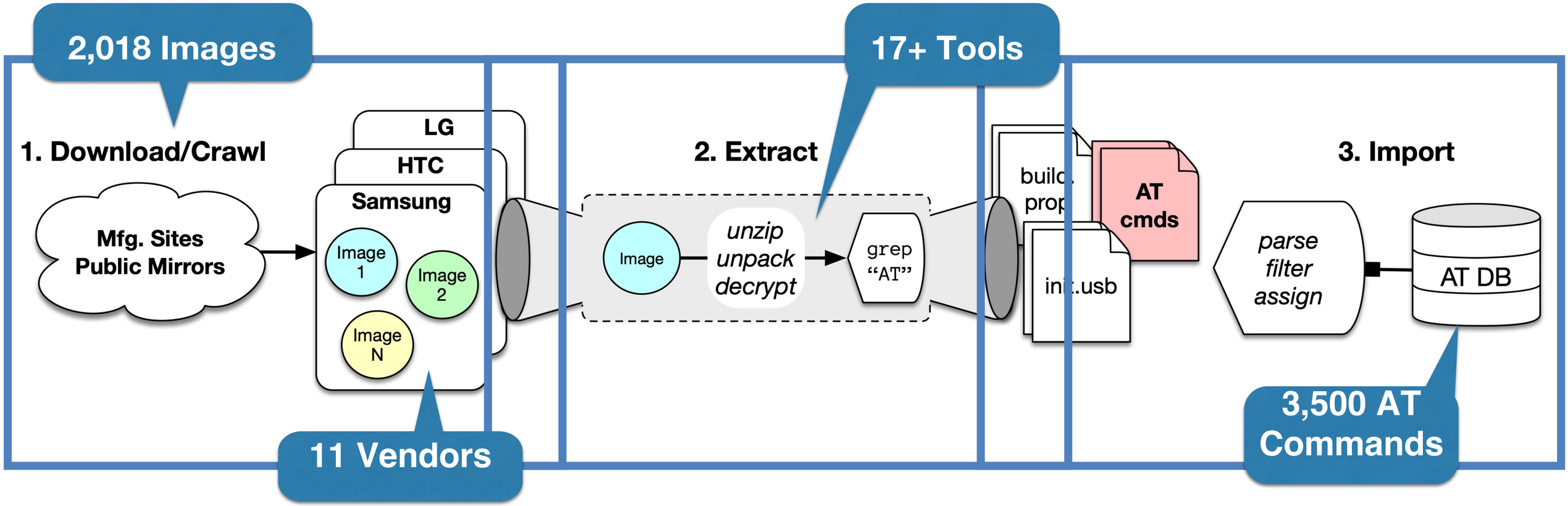
Prevalence of AT Commands

AT commands aren't new

Previous work on smartphones shows that a select few AT commands have an impact

- But we had little idea…
  - How many commands exist?
  - What their security impact is?
  - What the commands do?

Analysis Pipeline



Firmware Extraction

Image files are usually ZIPs, but some vendors have strange formats

No standard firmware distribution format and no tool that can extract all formats!

Time to write our own omnibus extractor from scratch

Algorithm

foreach (firmware)

Recursively extract() using vendor or standard tools until the raw files of the Android system image are exposed

foreach (file)

if (file) is an APK or ODEX, decompile() it to Java source or Smali bytecode

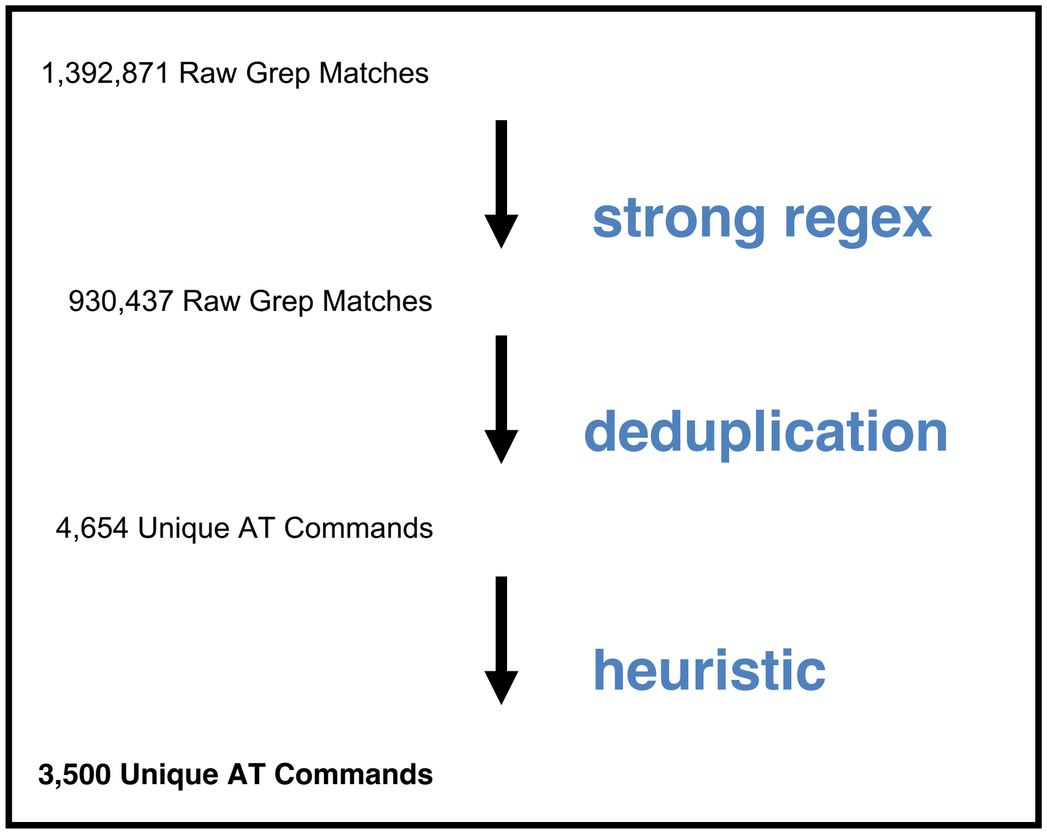
run(strings) on the binary file or source code

grep() strings for AT command regex

Filtering

Once we have a raw list of AT command from each image, time to filter

Apply a stronger AT regex and some heuristics




```
1  (?:[^a-zA-Z0-9]|^)  # Left of the AT must NOT
2                      # be a letter or number
3
4  (?P<cmd>            # Capture the match
5    AT[!@#$%^&*+]     # Match AT[symbol]
6    [_A-Za-z0-9]{3,}  # Match the name and
7  )
8
9  (?P<arg>            # Capture the match
10   \?  |             # Match AT+READ?
11                     # Match AT+CSET=0,1,"param"
12   =["'+=;%,?A-Za-z0-9]+ |
13   =\? |             # Match AT+TEST=?
14   =                 # Match a blank parameter
15 )?                  # Match AT+EXEC
```

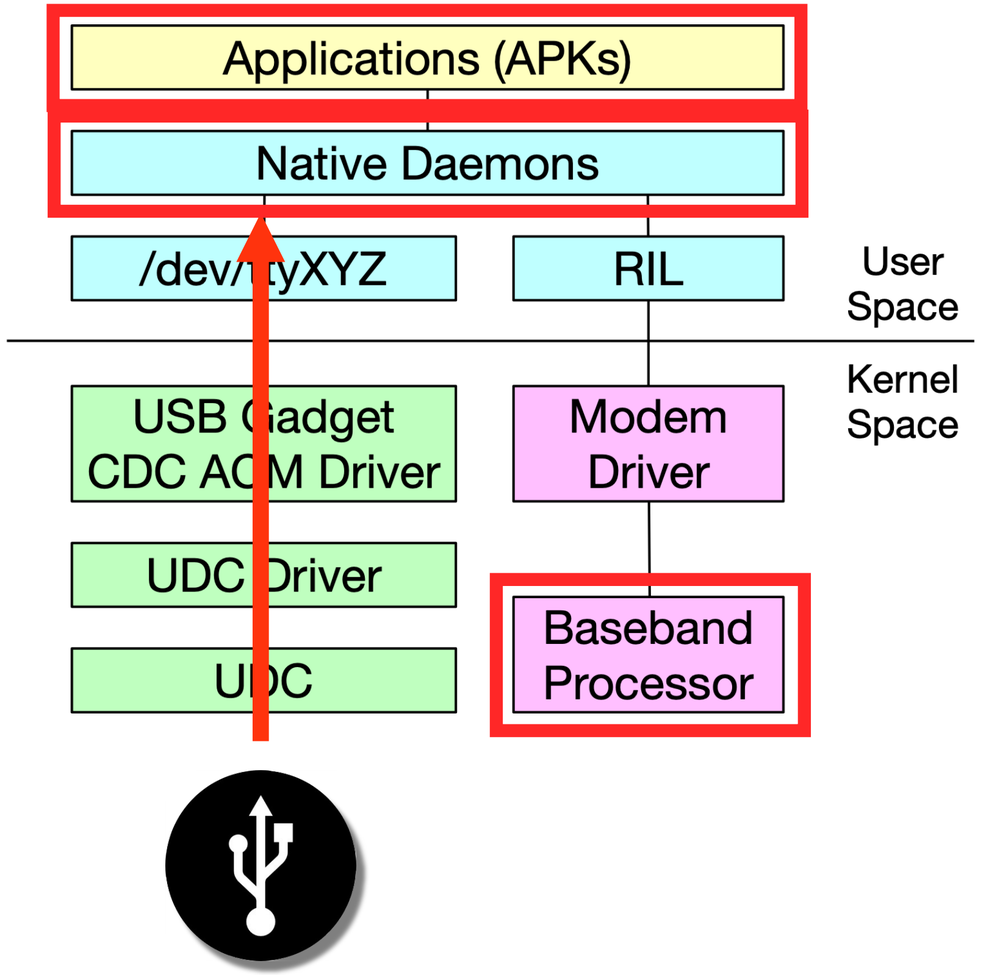
Attack Vector: Modem Interface

Your phone may expose a “modem interface” over USB, aka CDC ACM device

Commands flow from the USB port to a listening native daemon and either go to the modem or the Android system

They are multiplexed differently per-vendor

Some phones have a “hidden” modem configuration that can be activated externally with usbswitcher



Triaging the Results

Logs returned

- Lots of hints to functionality and possible security impact

Google	ATcmd#
/vendor/lib/libsec-ril_lte.so	183
/lib/libxgold-ril.so	73
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	23
/lib/bluez-plugin/audio.so	19
Samsung	
/bin/at_distributor	331
/md1rom.img	226
/app/FactoryTest_CAM.apk	145
/bin/sec_atd	142
/bin/engpc	140
LG	
/bin/atd	354
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	27
/app/LGATCMDService/arm/LGATCMDService.odex	19
/app/LGBluetooth4/arm/LGBluetooth4.odex	15

Triaging the Results

Logs returned

- Lots of hints to functionality and possible security impact

Phone Side-effect

- Menu pops up, WiFi disappears, etc.
- Phone reboots, factory resets itself

Google	ATcmd#
/vendor/lib/libsec-ril_lte.so	183
/lib/libxgold-ril.so	73
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	23
/lib/bluez-plugin/audio.so	19
Samsung	
/bin/at_distributor	331
/md1rom.img	226
/app/FactoryTest_CAM.apk	145
/bin/sec_atd	142
/bin/engpc	140
LG	
/bin/atd	354
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	27
/app/LGATCMDService/arm/LGATCMDService.odex	19
/app/LGBluetooth4/arm/LGBluetooth4.odex	15

Triaging the Results

Logs returned

- Lots of hints to functionality and possible security impact

Phone Side-effect

- Menu pops up, WiFi disappears, etc.
- Phone reboots, factory resets itself

No obvious effect

- Many commands return "OK" or "ERROR"
- Use IDA Pro to disassemble the AT command distributors
- read the assembly source

Google	ATcmd#
/vendor/lib/libsec-ril_lte.so	183
/lib/libxgold-ril.so	73
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	23
/lib/bluez-plugin/audio.so	19
Samsung	
/bin/at_distributor	331
/md1rom.img	226
/app/FactoryTest_CAM.apk	145
/bin/sec_atd	142
/bin/engpc	140
LG	
/bin/atd	354
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	27
/app/LGATCMDService/arm/LGATCMDService.odex	19
/app/LGBluetooth4/arm/LGBluetooth4.odex	15

Triaging the Results

Logs returned

- Lots of hints to functionality and possible security impact

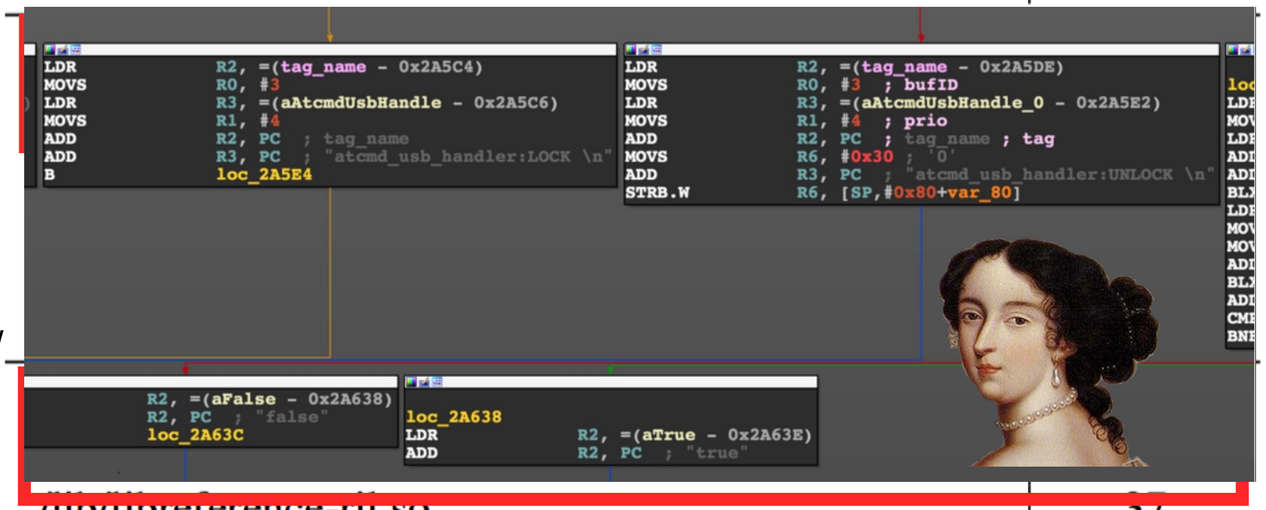
Phone Side-effect

- Menu pops up, WiFi disappears, etc.
- Phone reboots, factory resets itself

No obvious effect

- Many commands return “OK” or “ERROR”
- Use IDA Pro to disassemble the AT command distributors
- read the assembly source

Google	ATcmd#
/vendor/lib/libsec-ril_lte.so	183
/lib/libxgold-ril.so	73
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	23
/lib/bluez-plugin/audio.so	19
/lib/libreference-ril.so	37
/lib/hw/bluetooth.default.so	27
/app/LGATCMDService/arm/LGATCMDService.odex	19
/app/LGBluetooth4/arm/LGBluetooth4.odex	15



Sensitive Information Leaking

Path traversal vulnerability found in **AT%PROCCAT** and **AT%SYSCAT** commands

Allows reading of entire SDCard!

IMEI and plenty of other information can be leaked from your phone

Command	Action	Tested Phones
ATI	Manufacturer, model, revision, SVN, IMEI	G4/S8+/Nexus5/ ZenPhone2
AT%SYSCAT	Read and return data from `/sys/*`[9]	G3/G4
AT%PROCCAT	Read and return data from `/proc/*`	G3/G4
AT+DEVCONINFO	Phone model, serial number, IMEI, and etc.	Note2/S7Edge/S8+
AT+GMR	Phone model	G3/G4/Note2/S8+/ ZenPhone2
AT+IMEINUM	IMEI number	Note2/S7Edge/S8+
AT+SERIALNO	Serial number	Note2/S7Edge/S8+
AT+SIZECHECK	Filesystem partition information	Note2/S7Edge/S8+
AT+VERSNAME	Android version	S7Edge/S8+
AT+CLAC	List all supported AT commands	G3/G4/S7Edge/Nexus5/ ZenPad/ZenPhone2
AT+ICCID	Sim card ICCID	G3/G4/Nexus5

Android Security Bypassing

Make Calls

```
ATD3521174567
```

Enable USB debugging (LG)

```
AT%USB=adb
```

Bypass the lock screen (LG)

```
AT%KEYLOCK=0
```

Inject Touch Events

```
AT+CTSA=EVENT,X,Y
```

Command	Action	Tested Phones
ATD	Dial a number	G3/G4/S8+/Nexus5/ ZenPhone2
ATH	Hangup call	G3/G4/S8+/Nexus5/ ZenPhone2
ATA	Answer incoming call	G3/G4/Nexus5
AT%IMEI=[param]	Allows the IMEI to be changed	G3/G4
AT%USB=adb	Enables invisible ADB debugging	G3/G4
AT%KEYLOCK=0	Unlock the screen	G3/G4
AT+CKPD	Sends keypad keys ([0-9*#])	G3/G4/S8+
AT+CMGS	Sends a SMS message	ZenPhone2
AT+CGDATA	Connect to the Internet using data	G3/G4/Nexus5/ ZenPhone2
AT+CPIN	SIM PIN management	G3/G4/S8+/Nexus5/ ZenPhone2
AT$QCMGD	Delete messages (by index, all read/sent)	Nexus5

Android Security Bypassing

Make Call

ATD352

Enable US

AT%USE

Bypass the

AT%KEY

Inject Tou

AT+CTS

LILY HAY NEWMAN SECURITY AUG 29, 2018 7:00 AM

Exploiting Decades-Old Telephone Tech to Break Into Android Devices

So-called Attention commands date back to the 80s, but they can enable some very modern-day smartphone hacks.

HALIE CHAVEZ/GETTY IMAGES

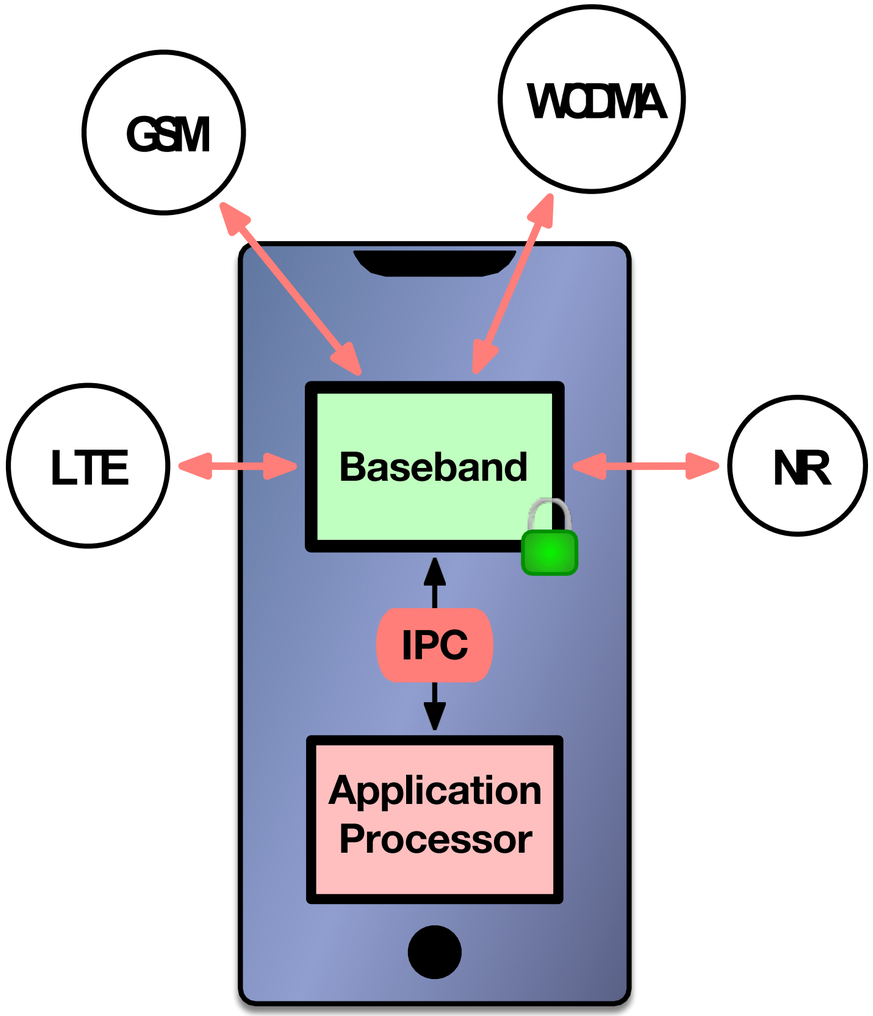
LG
LVE-SMP-18001
Severity: High

Samsung
Security Updates issued

Tested Phones
G3/G4/S8+/Nexus5/ ZenPhone2
G3/G4/S8+/Nexus5/ ZenPhone2
G3/G4/Nexus5
G3/G4
G3/G4
G3/G4
G3/G4/S8+
ZenPhone2
G3/G4/Nexus5/ ZenPhone2
G3/G4/S8+/Nexus5/ ZenPhone2
Nexus5

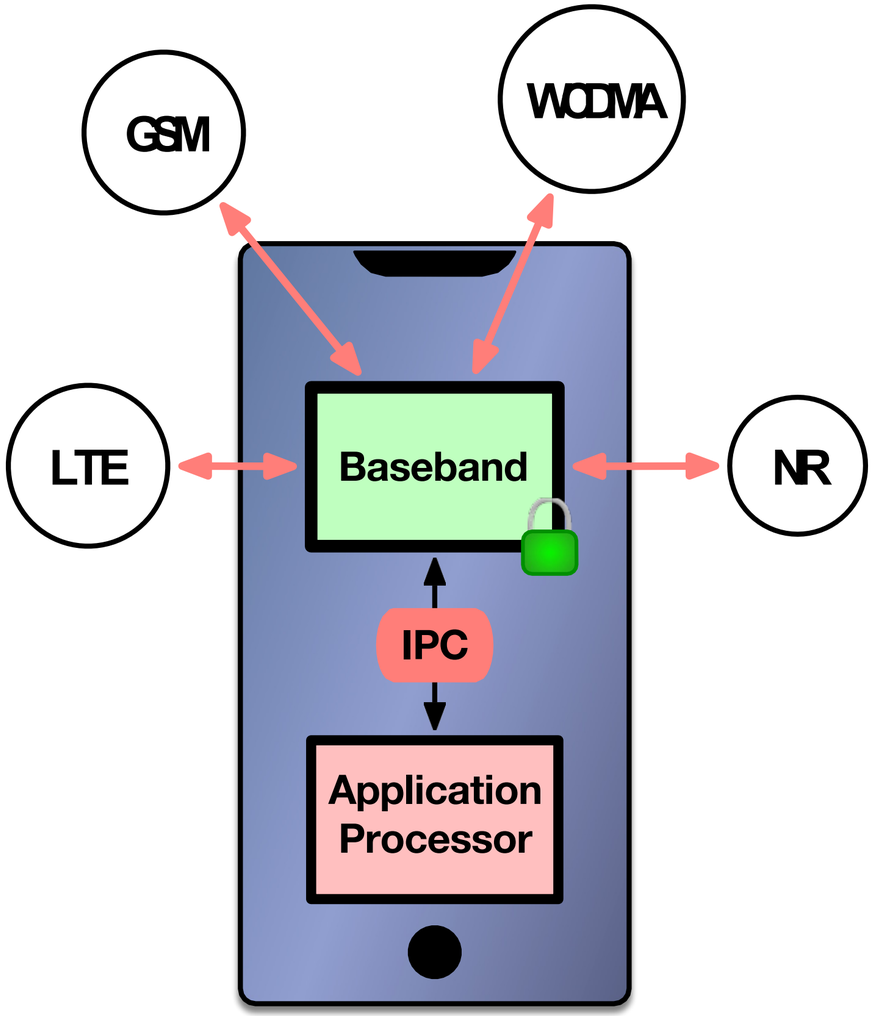
Baseband Processors

- Basebands implement multiple generations of 3GPP (and, for now, 3GPP2) cellular standards



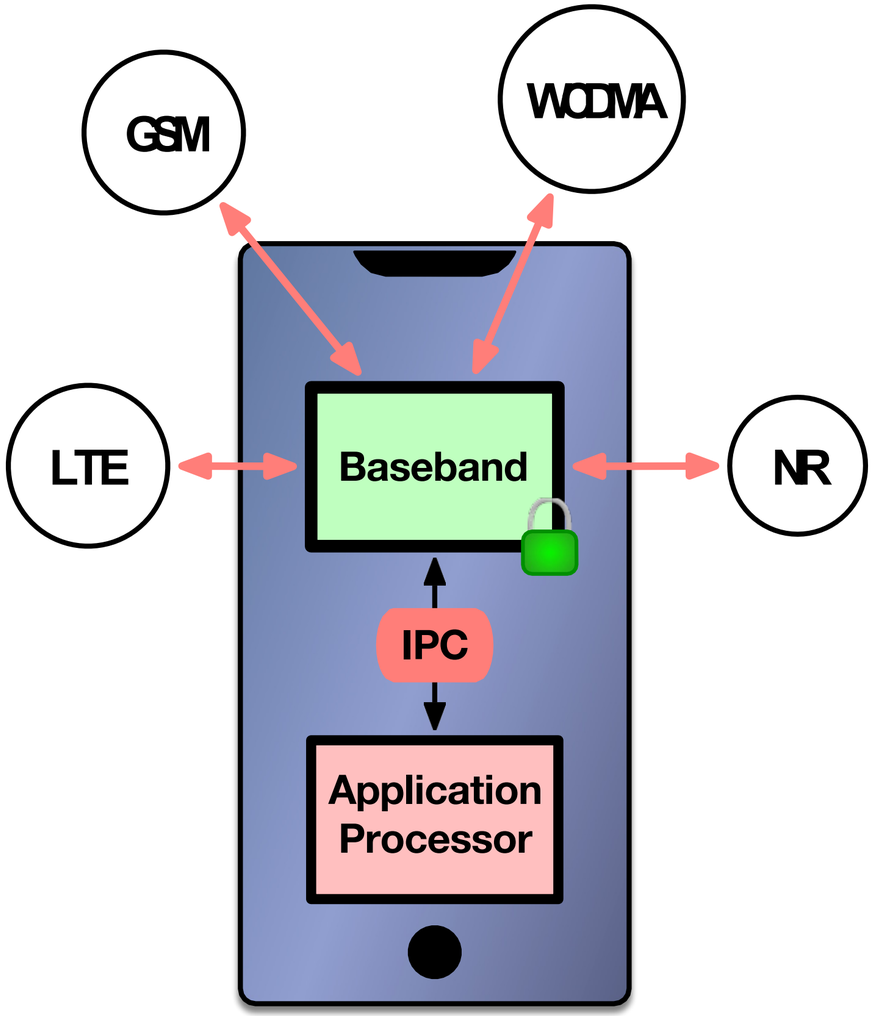
Why basebands?

- Basebands implement multiple generations of 3GPP (and, for now, 3GPP2) cellular standards
  - More standards → more implementation bugs



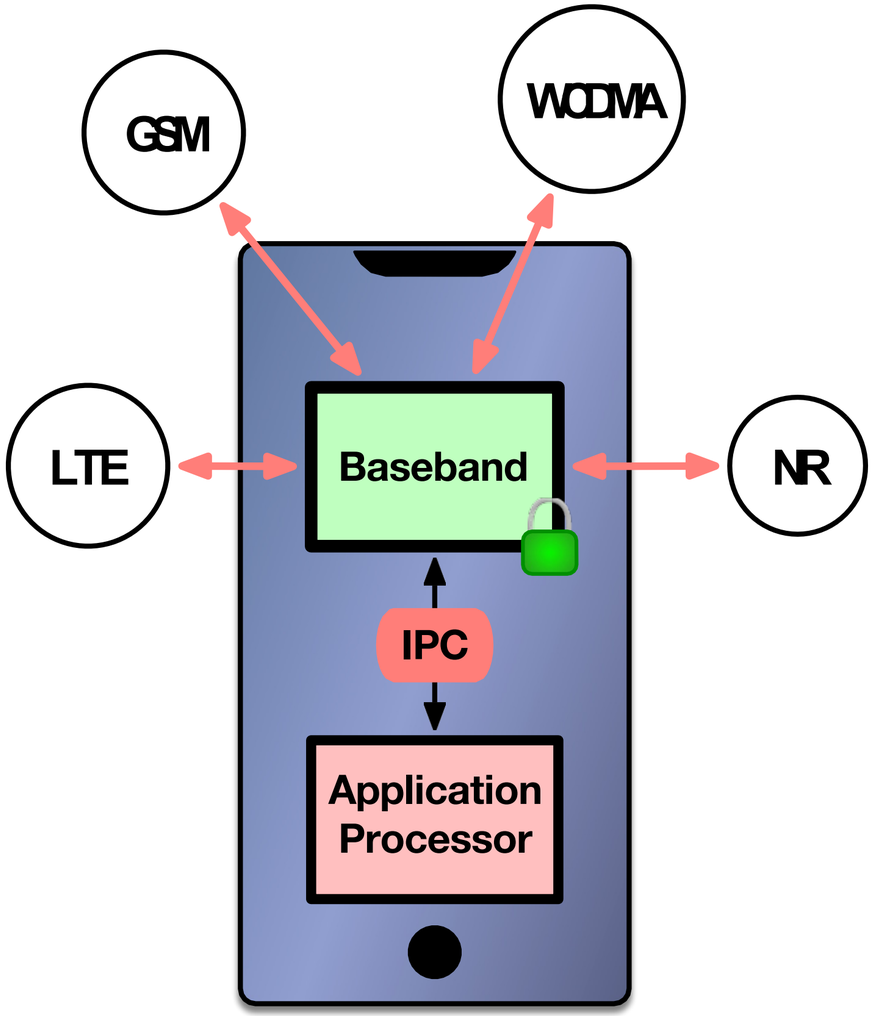
Why basebands?

- Basebands implement multiple generations of 3GPP (and, for now, 3GPP2) cellular standards
  - More standards → more implementation bugs
  - More bugs → more security vulnerabilities



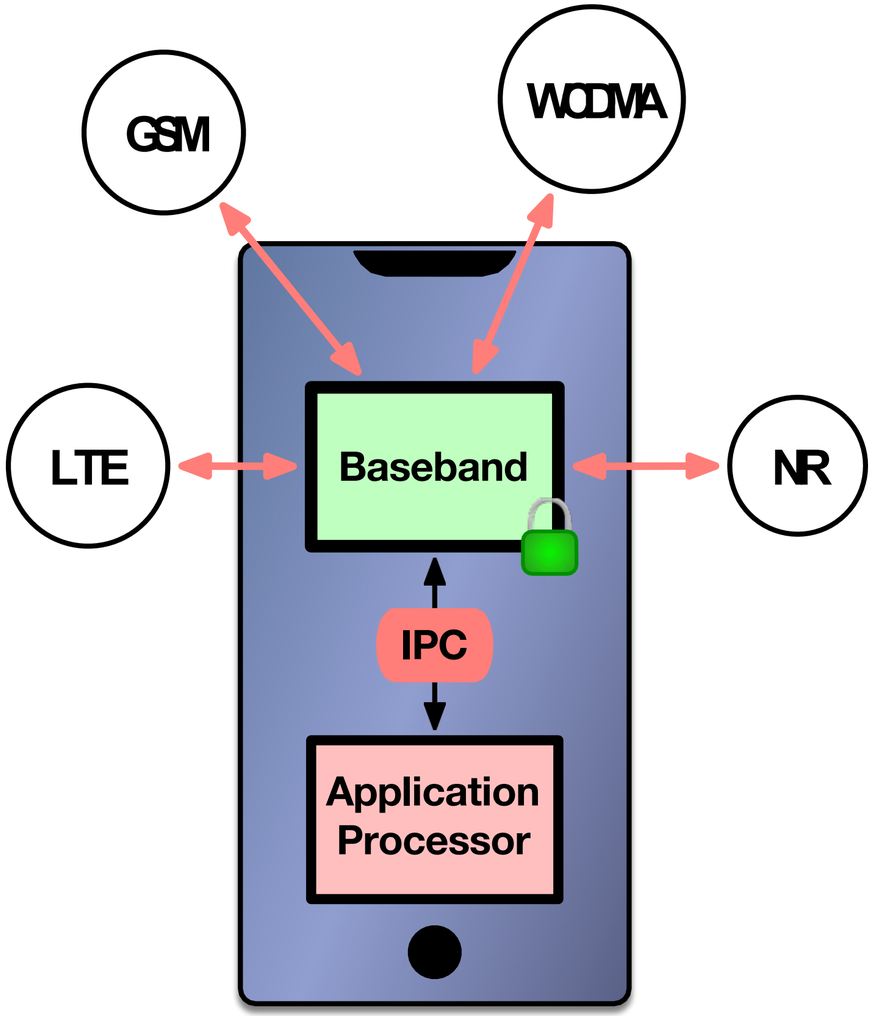
Why basebands?

- Basebands implement multiple generations of 3GPP (and, for now, 3GPP2) cellular standards
  - More standards → more implementation bugs
  - More bugs → more security vulnerabilities
  - More vulnerabilities means more exploitable bugs



Why basebands?

- Basebands implement multiple generations of 3GPP (and, for now, 3GPP2) cellular standards
  - More standards → more implementation bugs
  - More bugs → more security vulnerabilities
  - More vulnerabilities means more exploitable bugs
- Today, basebands are comparatively "easier" targets.
  Android/iOS userspace, kernel, and browsers are hard targets to exploit
  - But baseband functionality has been largely hidden



Auditing basebands without source code

- Firmware for basebands is shipped as a binary of a certain CPU architecture

```
if (boot_mode == DUMP_MODE) {
  boot_unk_common_setup();
  uart_putc('#');
  boot_crash_or_dump();
  boot_unk_crash();
  uart_puts(s_Done_40000550);
  FUN_40000d84(&DAT_00002e00);
}
else {
  if (boot_mode == BOOT_MODE) {
    boot_unk_common_setup();
    uart_putc('#');
    boot_prepare_mpu_next_ram();
    uart_puts(s_Boot_40000568);
    nextFnPointer = boot_comm_ap();
    if ((void *)0x10000000 < nextFnPointer) {
      boot_stage2(nextFnPointer);
      goto LAB_400004f0;
    }
    r0 = s_XxX!_40000570;
  }
  else {
    r0 = s_Unknown_4000055c;
  }
  uart_puts(r0);
}
```

Auditing basebands without source code

- Firmware for basebands is shipped as a binary of a certain CPU architecture
  - Samsung Exynos - ARM Cortex-R / A

```
if (boot_mode == DUMP_MODE) {
  boot_unk_common_setup();
  uart_putc('#');
  boot_crash_or_dump();
  boot_unk_crash();
  uart_puts(s_Done_40000550);
  FUN_40000d84(&DAT_00002e00);
}
else {
  if (boot_mode == BOOT_MODE) {
    boot_unk_common_setup();
    uart_putc('#');
    boot_prepare_mpu_next_ram();
    uart_puts(s_Boot_40000568);
    nextFnPointer = boot_comm_ap();
    if ((void *)0x10000000 < nextFnPointer) {
      boot_stage2(nextFnPointer);
      goto LAB_400004f0;
    }
    r0 = s_XxX!_40000570;
  }
  else {
    r0 = s_Unknown_4000055c;
  }
  uart_puts(r0);
}
```

Auditing basebands without source code

- Firmware for basebands is shipped as a binary of a certain CPU architecture
  - Samsung Exynos - ARM Cortex-R / A
  - Qualcomm - Hexagon DSP

```
if (boot_mode == DUMP_MODE) {
  boot_unk_common_setup();
  uart_putc('#');
  boot_crash_or_dump();
  boot_unk_crash();
  uart_puts(s_Done_40000550);
  FUN_40000d84(&DAT_00002e00);
}
else {
  if (boot_mode == BOOT_MODE) {
    boot_unk_common_setup();
    uart_putc('#');
    boot_prepare_mpu_next_ram();
    uart_puts(s_Boot_40000568);
    nextFnPointer = boot_comm_ap();
    if ((void *)0x10000000 < nextFnPointer) {
      boot_stage2(nextFnPointer);
      goto LAB_400004f0;
    }
    r0 = s_XxX!_40000570;
  }
  else {
    r0 = s_Unknown_4000055c;
  }
  uart_puts(r0);
}
```

Auditing basebands without source code

- Firmware for basebands is shipped as a binary of a certain CPU architecture
  - Samsung Exynos - ARM Cortex-R / A
  - Qualcomm - Hexagon DSP
  - MediaTek - MIPS16e2 / nanoMIPS

```
if (boot_mode == DUMP_MODE) {
  boot_unk_common_setup();
  uart_putc('#');
  boot_crash_or_dump();
  boot_unk_crash();
  uart_puts(s_Done_40000550);
  FUN_40000d84(&DAT_00002e00);
}
else {
  if (boot_mode == BOOT_MODE) {
    boot_unk_common_setup();
    uart_putc('#');
    boot_prepare_mpu_next_ram();
    uart_puts(s_Boot_40000568);
    nextFnPointer = boot_comm_ap();
    if ((void *)0x10000000 < nextFnPointer) {
      boot_stage2(nextFnPointer);
      goto LAB_400004f0;
    }
    r0 = s_XxX!_40000570;
  }
  else {
    r0 = s_Unknown_4000055c;
  }
  uart_puts(r0);
}
```

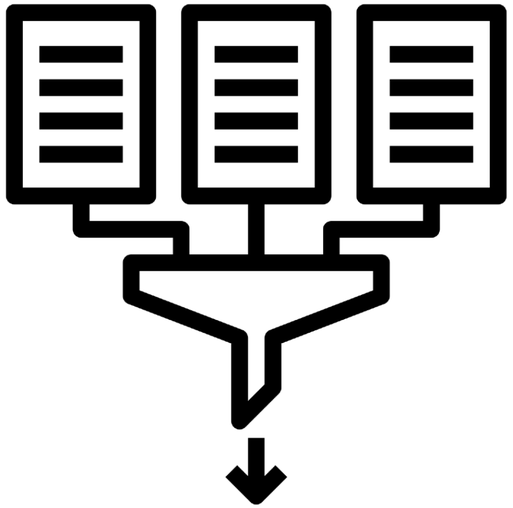
Auditing basebands without source code

- Firmware for basebands is shipped as a binary of a certain CPU architecture
  - Samsung Exynos - ARM Cortex-R / A
  - Qualcomm - Hexagon DSP
  - MediaTek - MIPS16e2 / nanoMIPS
- Using a disassembler allows us to recover program structures from machine code

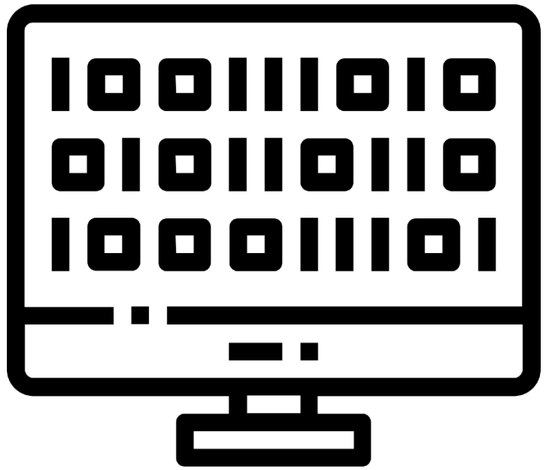
```
if (boot_mode == DUMP_MODE) {
  boot_unk_common_setup();
  uart_putc('#');
  boot_crash_or_dump();
  boot_unk_crash();
  uart_puts(s_Done_40000550);
  FUN_40000d84(&DAT_00002e00);
}
else {
  if (boot_mode == BOOT_MODE) {
    boot_unk_common_setup();
    uart_putc('#');
    boot_prepare_mpu_next_ram();
    uart_puts(s_Boot_40000568);
    nextFnPointer = boot_comm_ap();
    if ((void *)0x10000000 < nextFnPointer) {
      boot_stage2(nextFnPointer);
      goto LAB_400004f0;
    }
    r0 = s_XxX!_40000570;
  }
  else {
    r0 = s_Unknown_4000055c;
  }
  uart_puts(r0);
}
```

Baseband Testing Strategies

Over-the-air testing



Binary Static Analysis

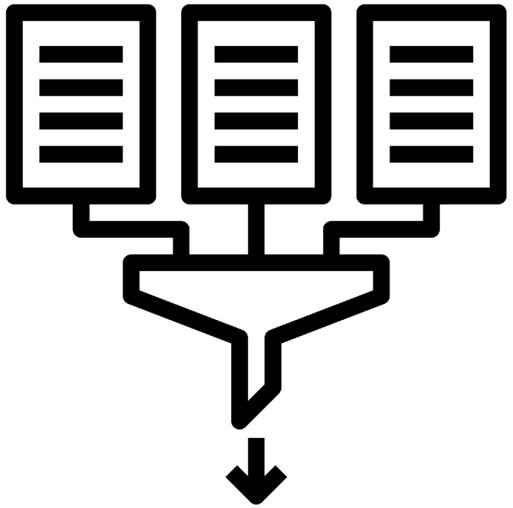


Emulation

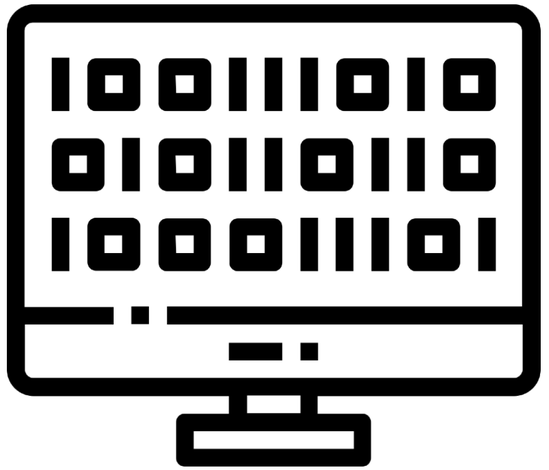
Baseband Testing Strategies

Over-the-air testing

Manual & non–deterministic
Lack of crash details



Binary Static Analysis

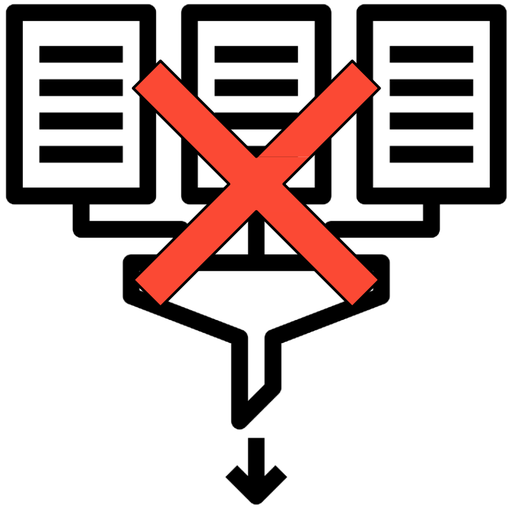


Emulation

Baseband Testing Strategies

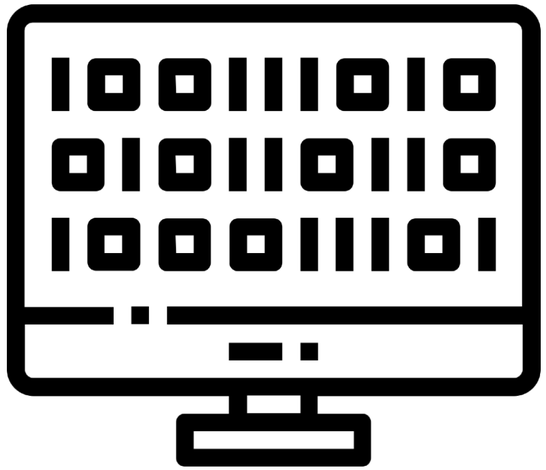
Over-the-air testing

Manual & non–deterministic
Lack of crash details



Binary Static Analysis

Many complex protocols and firmware versions to analyze

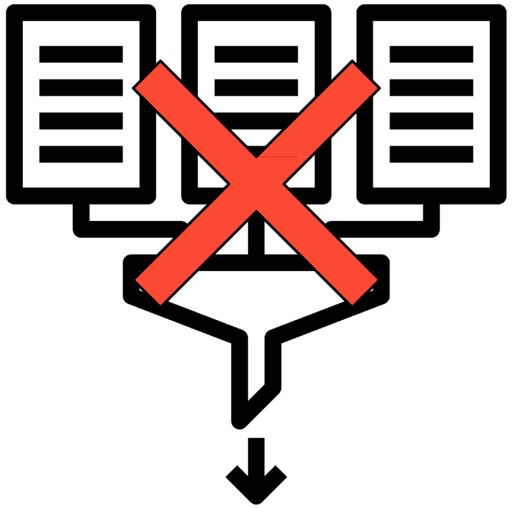


Emulation

Baseband Testing Strategies

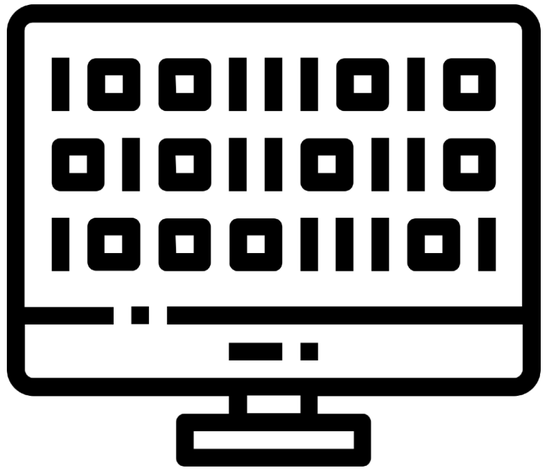
Over-the-air testing

Manual & non–deterministic
Lack of crash details

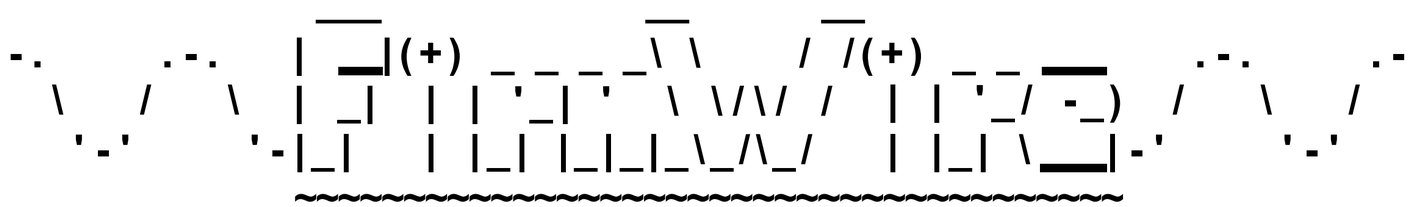


Binary Static Analysis

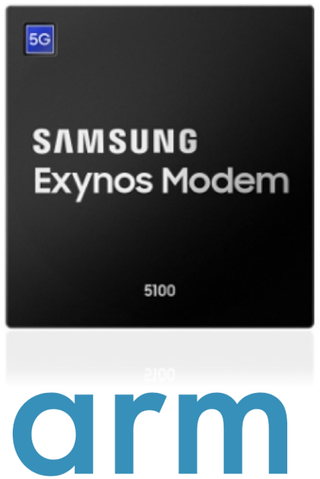
Many complex protocols and firmware versions to analyze



Emulation

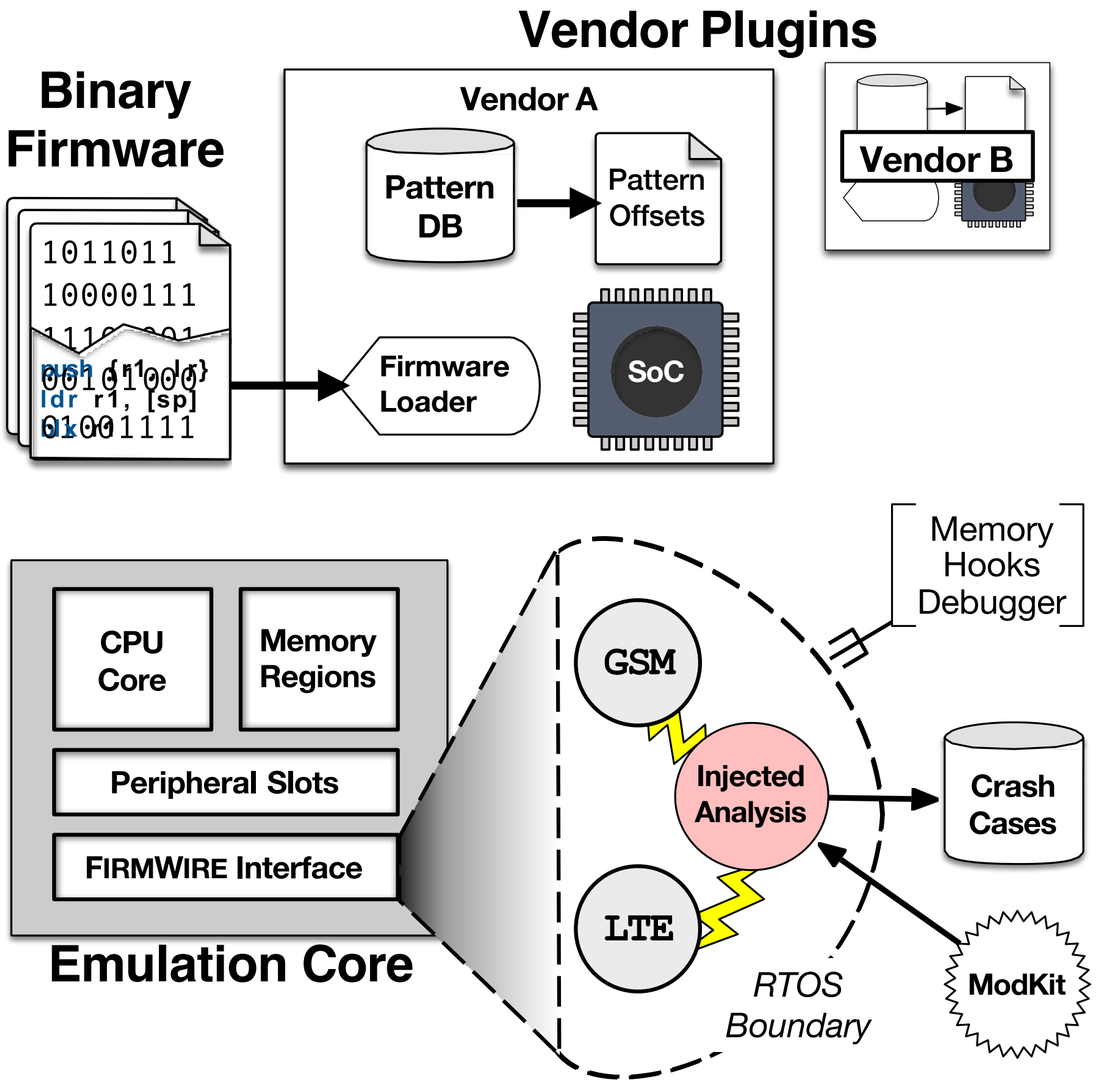


- FirmWire is the first dynamic analysis platform to support emulating Samsung and MediaTek baseband firmware from boot
- Built on PANDA (QEMU derivative) and allows for binary-only, coverage-guided fuzzing and memory inspection
- Mostly written in Python with Avatar2 as an underlying framework



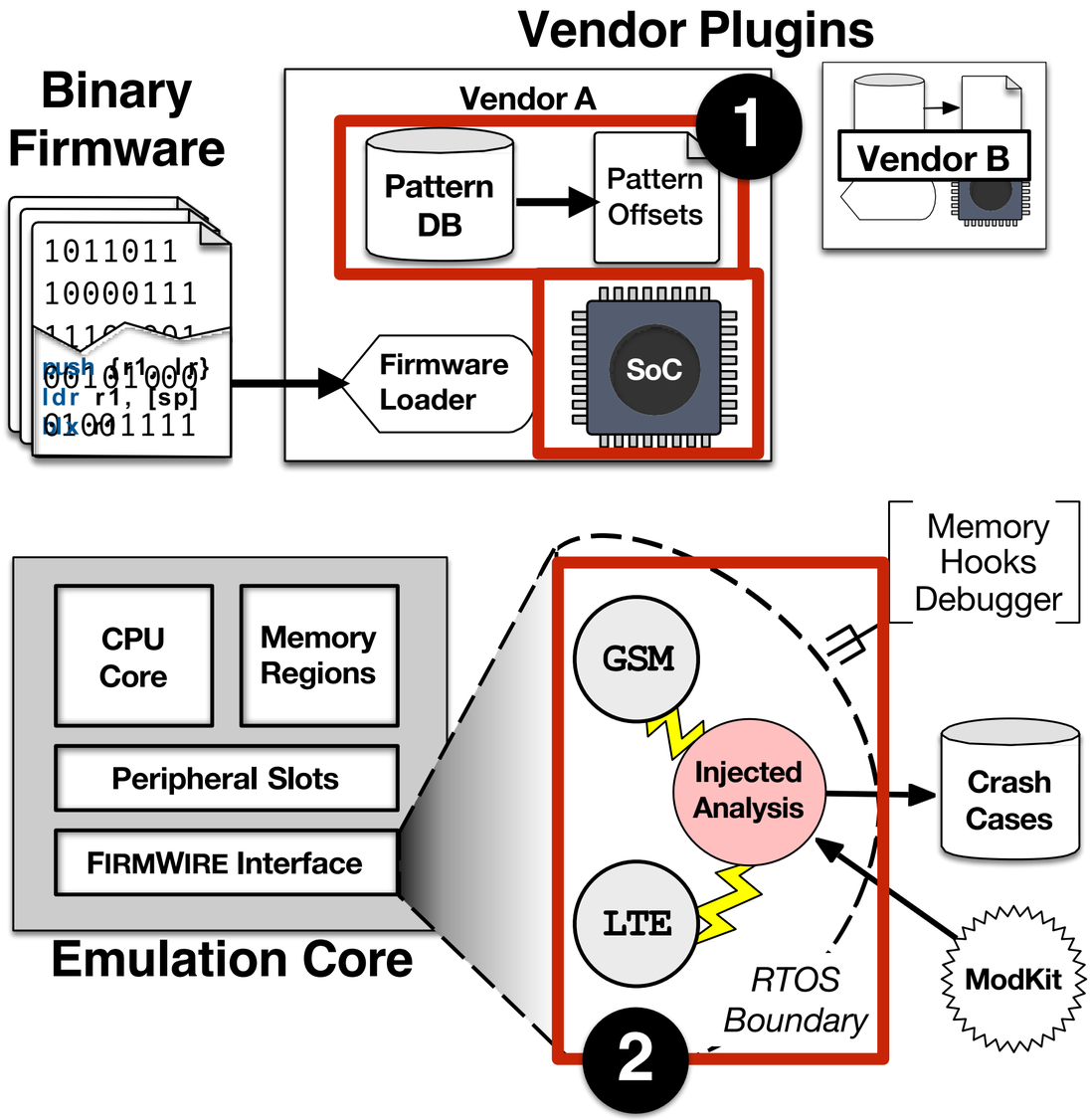
FirmWire Features

- It supports multiple platforms, chipsets, and phone models through *vendor plugins*
  - MTK: support for MIPS16e2
  - Shannon: support for ARM Cortex-R
- It offers cross-platform RTOS introspection and task injection
- Used to find multiple over-the-air triggerable bugs in GSM and LTE implementations

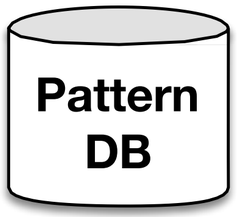


FirmWire Features

- It supports multiple platforms, chipsets, and phone models through *vendor plugins*
  - MTK: support for MIPS16e2
  - Shannon: support for ARM Cortex-R
- It offers cross-platform RTOS introspection and task injection
- Used to find multiple over-the-air triggerable bugs in GSM and LTE implementations



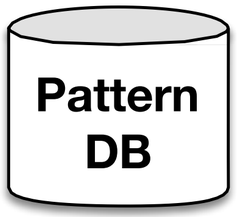
Vendor Plugin: PatternDB



FindPattern(“BootTable”, [FirmwareBytes]) → *Offset*

```
Pattern := {
    name := "BootTable"
    pattern := [
      # Search for two stable 4-byte
      values "00008004 200c0000”,
      # Fuzzy pattern for other
      images "00000004 ????0100"
    ]
    required := true # Fail boot if not
    found # Process the found table and
    extract post_lookup :=
    parse_memory_table
    # Adjust found address
    offset := -0x14
    # Make sure it’s 4 byte aligned
    align := 4
}
```

Vendor Plugin: PatternDB



FindPattern(“BootTable”, [FirmwareBytes]) → *Offset*

```
Pattern := {
    name := "BootTable"
    pattern := [
      # Search for two stable 4-byte
      values "00008004 200c0000”,
      # Fuzzy pattern for other
      images "00000004 ????0100"
    ]
    required := true # Fail boot if not
    found # Process the found table and
    extract post_lookup :=
    parse_memory_table
    # Adjust found address
    offset := -0x14
    # Make sure it’s 4 byte aligned
    align := 4
}
```

Vendor	# Patterns
Samsung	18
MediaTek	9

Vendor Plugin: Multiple SoCs

```
from firmwire.peripherals import *

class VendorBaseSOC:
    common_peripherals = [
    SOCPeripheral(UARTPeripheral,
          base=0x84000000,
          size=0x1000)
    ]

class SOC123(VendorBaseSOC):
    name = "SOC123"
    # SoC specific peripherals
    peripherals = [
      SOCPeripheral(PMICPeripheral,
        base=0x80000000,
        size=0x100)
    ]
    # SoC specific attributes
    CHIP_ID = 0x01230000
    SOC_BASE = 0x82000000
    TIMER_BASE = SOC_BASE + 0x8000
```

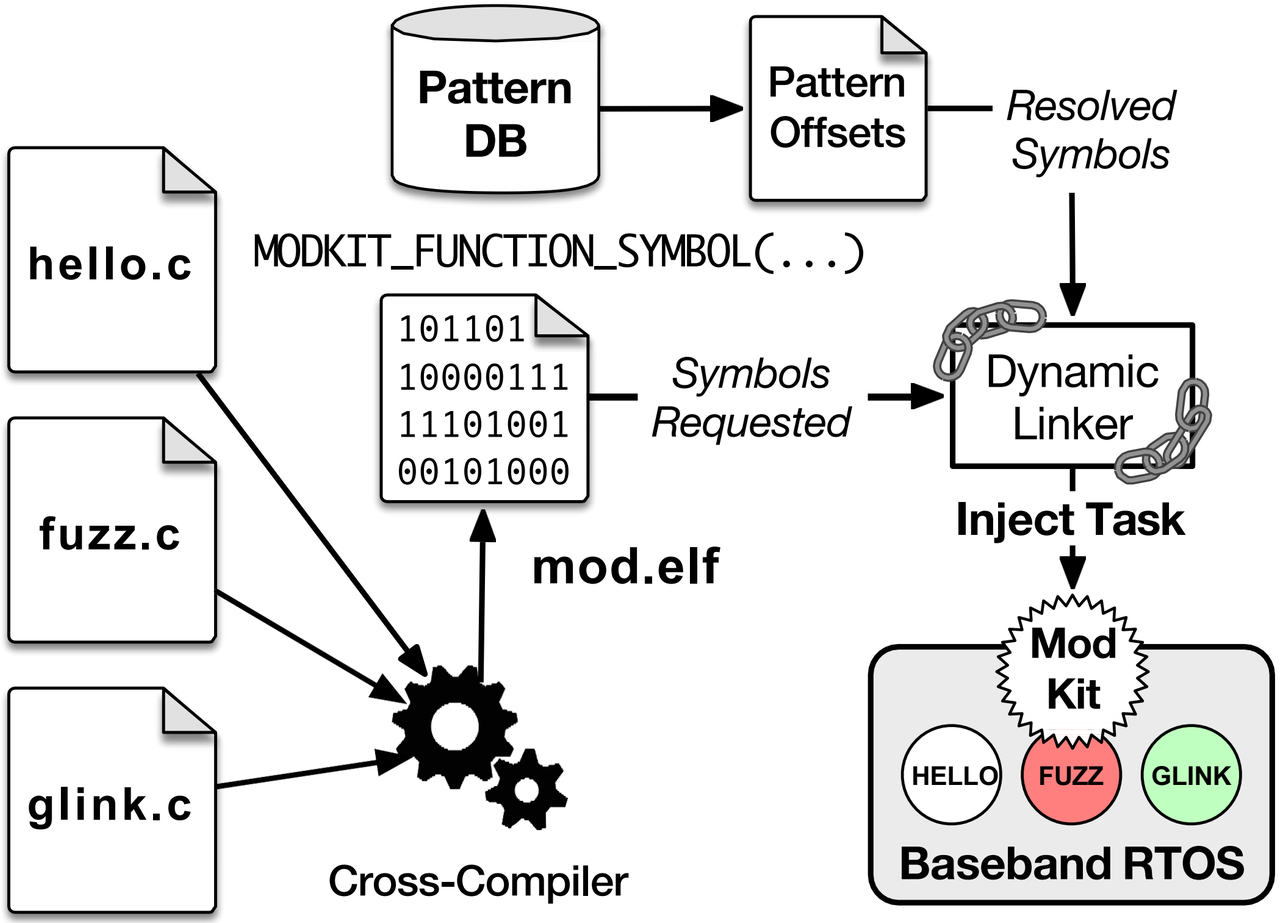
Vendor Plugin: Multiple SoCs

```
from firmwire.peripherals import *

class VendorBaseSOC:
    common_peripherals = [
    SOCPeripheral(UARTPeripheral,
            base=0x84000000,
            size=0x1000)
    ]

class SOC123(VendorBaseSOC):
    name = "SOC123"
    # SoC specific peripherals
    peripherals = [
      SOCPeripheral(PMICPeripheral,
        base=0x80000000,
        size=0x100)
    ]
    # SoC specific attributes
    CHIP_ID = 0x01230000
    SOC_BASE = 0x82000000
    TIMER_BASE = SOC_BASE + 0x8000
```

Vendor	Galaxy Model	Chipset	#SLoC
Samsung	S7/S7 Edge	S335AP	25
	S8/S8+	S355AP	29
	S9	S360AP	33
	S10/S10e	S5000AP	25
MediaTek	A10s	MT6762	14
	A41	MT6768	12

Modifying basebands with Task Injection

- To test protocols, fuzzing harnesses are compiled and injected into the modem's memory
- The harnesses **reuse the existing modem APIs** found using patterns to send create tasks and send fuzzing inputs towards targeted tasks



Fuzzing Campaigns

- We built 4 fuzzing harnesses and used coverage-guided fuzz testing (AFL++)

Fuzzing Campaigns

- We built 4 fuzzing harnesses and used coverage-guided fuzz testing (AFL++)
  - Samsung: LTE RRC, GSM SM, GSM CC

Fuzzing Campaigns

- We built 4 fuzzing harnesses and used coverage-guided fuzz testing (AFL++)
  - Samsung: LTE RRC, GSM SM, GSM CC
  - MediaTek: LTE RRC

Fuzzing Campaigns

- We built 4 fuzzing harnesses and used coverage-guided fuzz testing (AFL++)
  - Samsung: LTE RRC, GSM SM, GSM CC
  - MediaTek: LTE RRC
- NAS - we targeted the decoders for SM and CC

Fuzzing Campaigns

- We built 4 fuzzing harnesses and used coverage-guided fuzz testing (AFL++)
  - Samsung: LTE RRC, GSM SM, GSM CC
  - MediaTek: LTE RRC
- NAS - we targeted the decoders for SM and CC
- RRC - we targeted the ASN.1 decoders for BCCH/DCCH messages

2G Call Control (CC)

- 2G & 3G circuit switched (CS) calling uses "Call Control" (CC)
- messages CC SETUP is sent from the network ⇒ mobile device
- The packet is made up of Information Elements (IEs)

Table 9.70/3GPP TS 24.008: SETUP message content (network to mobile station direction)

IEI	Information element	Type/Reference	Presence	Format	Length
	Call control Protocol discriminator	Protocol discriminator 10.2	M	V	1/2
	Transaction identifier	Transaction identifier 10.3.2	M	V	1/2
	Setup Message type	Message type 10.4	M	V	1
D-	BC repeat indicator	Repeat indicator 10.5.4.22	C	TV	1
04	Bearer capability 1	Bearer capability 10.5.4.5	O	TLV	3-16
04	Bearer capability 2	Bearer capability 10.5.4.5	O	TLV	3-16
1C	Facility	Facility 10.5.4.15	O	TLV	2-?

Our fuzzed CC message

53 - Protocol discriminator (Samsung specific)
Should be PD = 3 for CC
05 - **CC SETUP IEI**

04 - **Bearer Capabilities IEI**
30 - **Bearer Capabilities length (ignored)**
30 30 30 30 30 30 30 30 30 30 30 30 30 30 - Bearer IE

1c - Facility IEI
30 - Facility length
a1 - Facility component type=INVOKE

1C	Facility	Facility 10.5.4.15	O	TLV	2-?

Our fuzzed CC message

53 - **Protocol discriminator (Samsung specific)**
Should be PD = 3 for CC
05 – **CC SETUP IEI**

04 - **Bearer Capabilities IEI**
30 - **Bearer Capabilities length (ignored)**
30 30 30 30 30 30 30 30 30 30 30 30 30 30 - Bearer IE

1c - Facility IEI
30 - Facility length
a1 - Facility component type=INVOKE

Facility ASN.1 decoder goes out of control!

1C	Facility	Facility 10.5.4.15	O	TLV	2-?

Fuzzing results

- **Discovered 7 crashes, 4 of which were previously unknown**

Fuzzing results

- **Discovered 7 crashes, 4 of which were previously unknown**
  - LTE RRC - 2 **critical**, and 1 **high**

Fuzzing results

- **Discovered 7 crashes, 4 of which were previously unknown**
  - LTE RRC - 2 **critical**, and 1 **high**
  - GSM CC - 1 **critical**

Fuzzing results

- **Discovered 7 crashes, 4 of which were previously unknown**
  - LTE RRC - 2 **critical**, and 1 **high**
  - GSM CC - 1 **critical**
  - GSM SM (ground-truth)

Fuzzing results

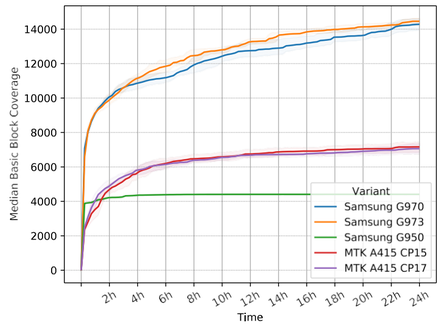
- **Discovered 7 crashes, 4 of which were previously unknown**
  - LTE RRC - 2 **critical**, and 1 **high**
  - GSM CC - 1 **critical**
  - GSM SM (ground-truth)
- Ratings given by Samsung

Fuzzing results

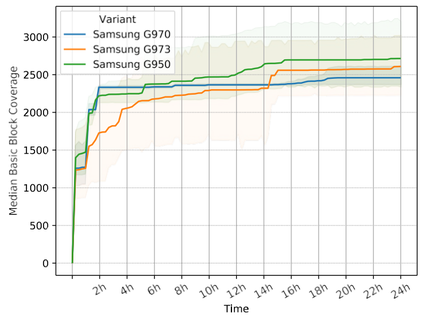
- **Discovered 7 crashes, 4 of which were previously unknown**
  - LTE RRC - 2 **critical**, and 1 **high**
  - GSM CC - 1 **critical**
  - GSM SM (ground-truth)
- Ratings given by Samsung
- Highest CVE - CVE-2020-25279 (9.8 critical, CC SETUP)

Fuzzing results

- **Discovered 7 crashes, 4 of which were previously unknown**
  - LTE RRC - 2 **critical**, and 1 **high**
  - GSM CC - 1 **critical**
  - GSM SM (ground-truth)
- Ratings given by Samsung
- Highest CVE - CVE-2020-25279 (9.8 critical, CC SETUP)



See paper for more details



OTA Crash Reproduction

OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations

OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations
- No SIM credentials were required, making all attacks pre-authentication

OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations
- No SIM credentials were required, making all attacks pre-authentication
-

LTE RRC (OpenLTE)

OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations
- No SIM credentials were required, making all attacks pre-authentication
-

LTE RRC (OpenLTE)

  - Modified the RRCConnectionReconfiguration encoder to instead throw the fuzzed RRC packets

OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations
- No SIM credentials were required, making all attacks pre-authentication
- **LTE RRC (OpenLTE)**
  - Modified the RRCConnectionReconfiguration encoder to instead throw the fuzzed RRC packets
- **GSM (YateBTS)**

OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations
- No SIM credentials were required, making all attacks pre-authentication
- **LTE RRC (OpenLTE)**
  - Modified the RRCConnectionReconfiguration encoder to instead throw the fuzzed RRC packets
- **GSM (YateBTS)**
  - **SM** - Changed the Protocol Configuration Options (PCO) encoder

OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations
- No SIM credentials were required, making all attacks pre-authentication
- **LTE RRC (OpenLTE)**
  - Modified the RRCConnectionReconfiguration encoder to instead throw the fuzzed RRC packets
- **GSM (YateBTS)**
  - **SM** - Changed the Protocol Configuration Options (PCO) encoder
  - **CC** - Changed the Call Setup encoder and initiated a call

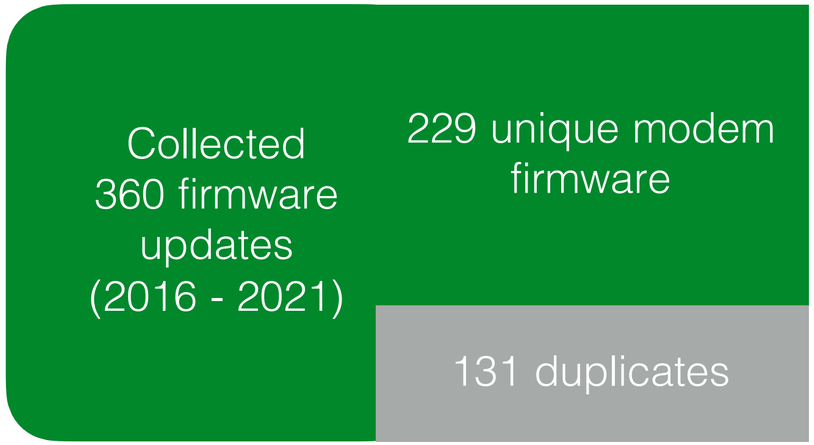
OTA Crash Reproduction

- We replayed crashing fuzz inputs over-the-air by modifying open source base stations
- No SIM credentials were required, making all attacks pre-authentication
- **LTE RRC (OpenLTE)**
  - Modified the RRCConnectionReconfiguration encoder to instead throw the fuzzed RRC packets
- **GSM (YateBTS)**
  - **SM** - Changed the Protocol Configuration Options (PCO) encoder
  - **CC** - Changed the Call Setup encoder and initiated a call
- **The basebands crashed with each message**

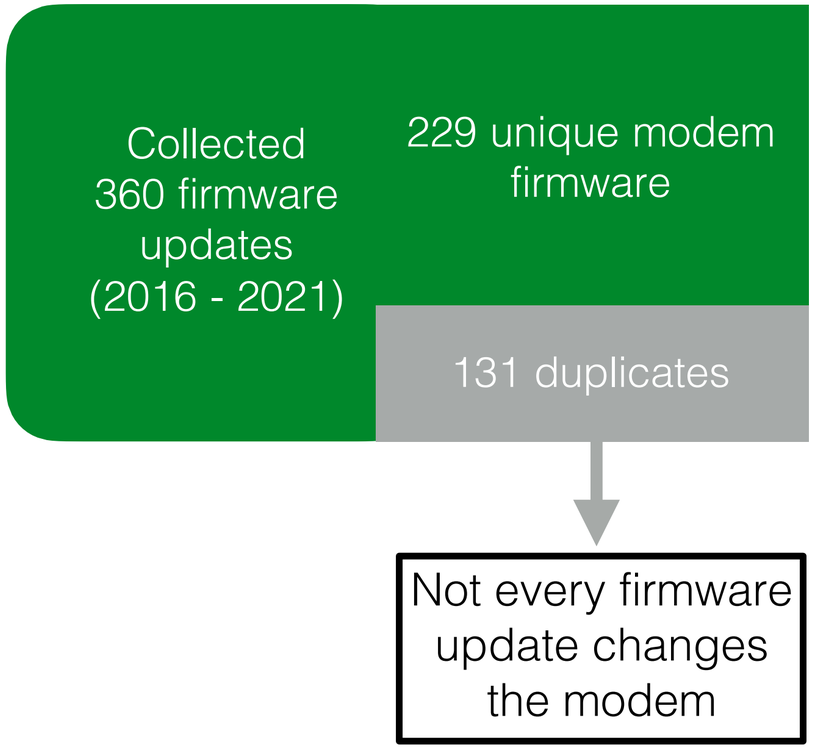
Scaling FirmWire

Collected
360 firmware
updates
(2016 - 2021)

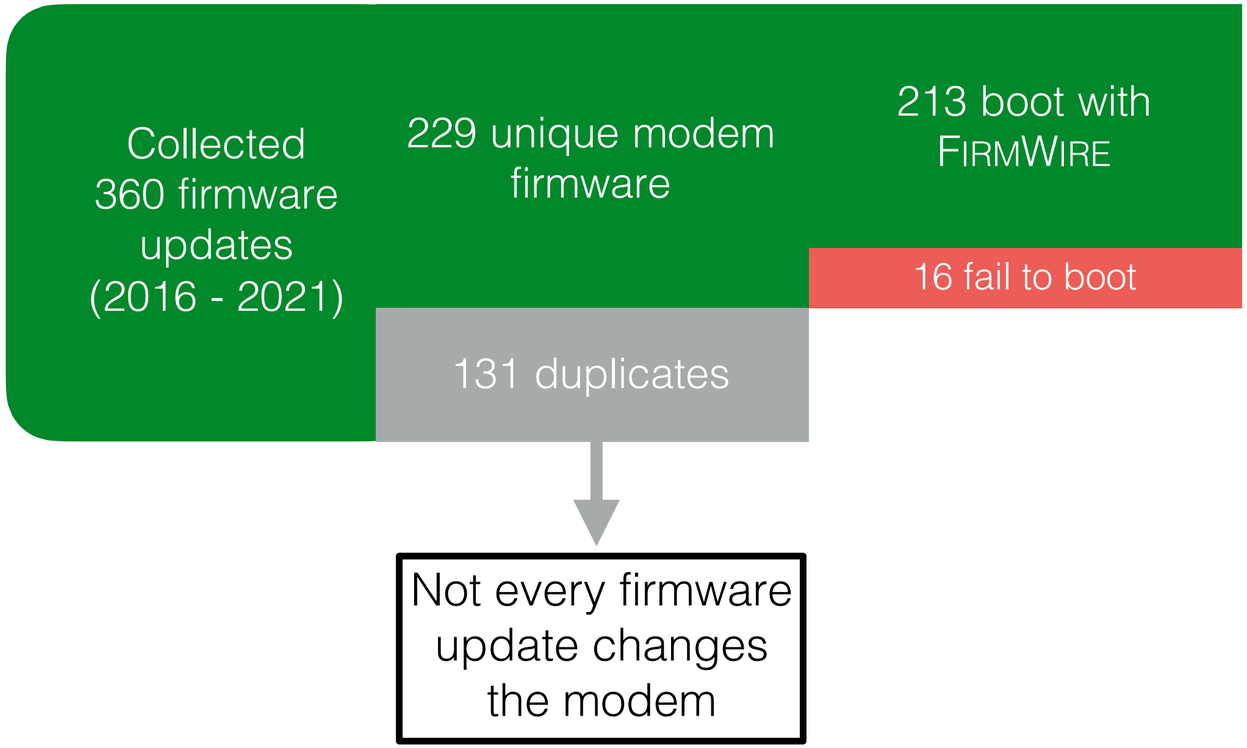
Scalability Testing



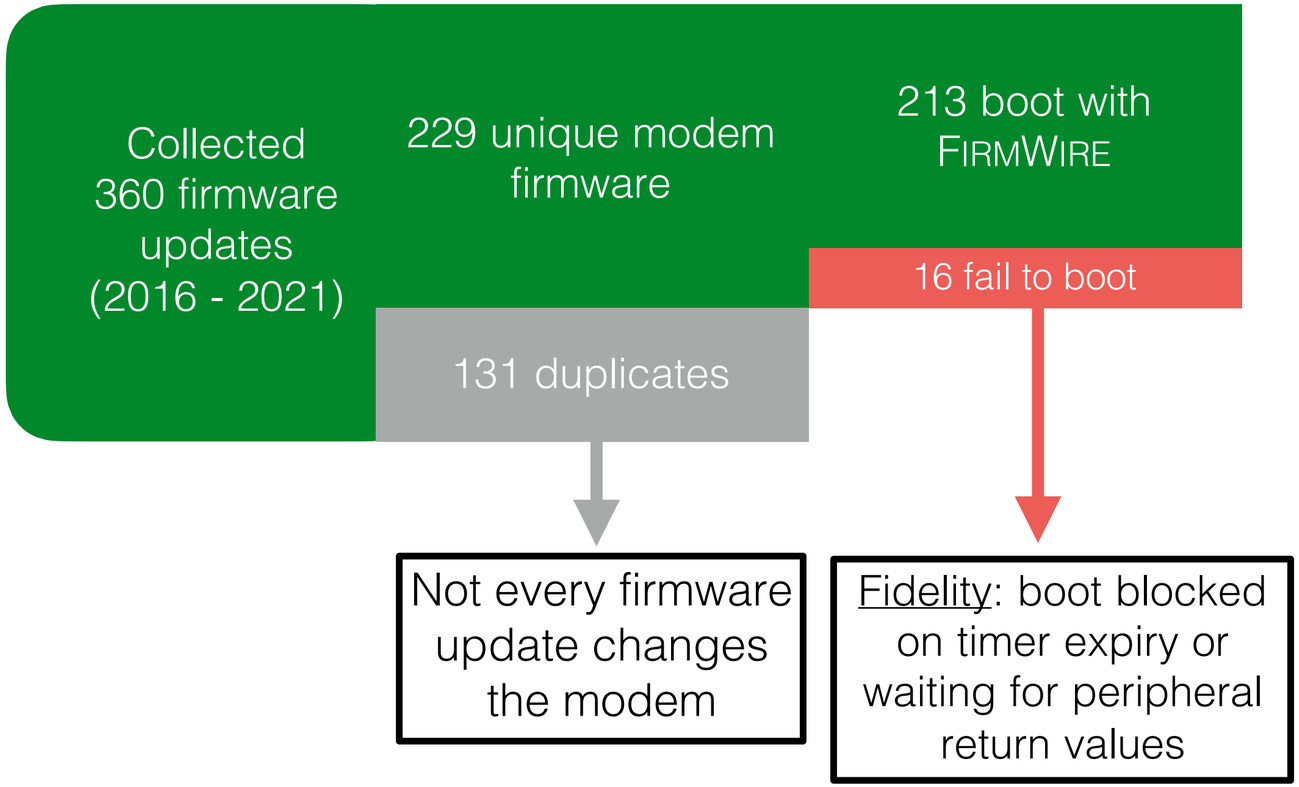
Scalability Testing



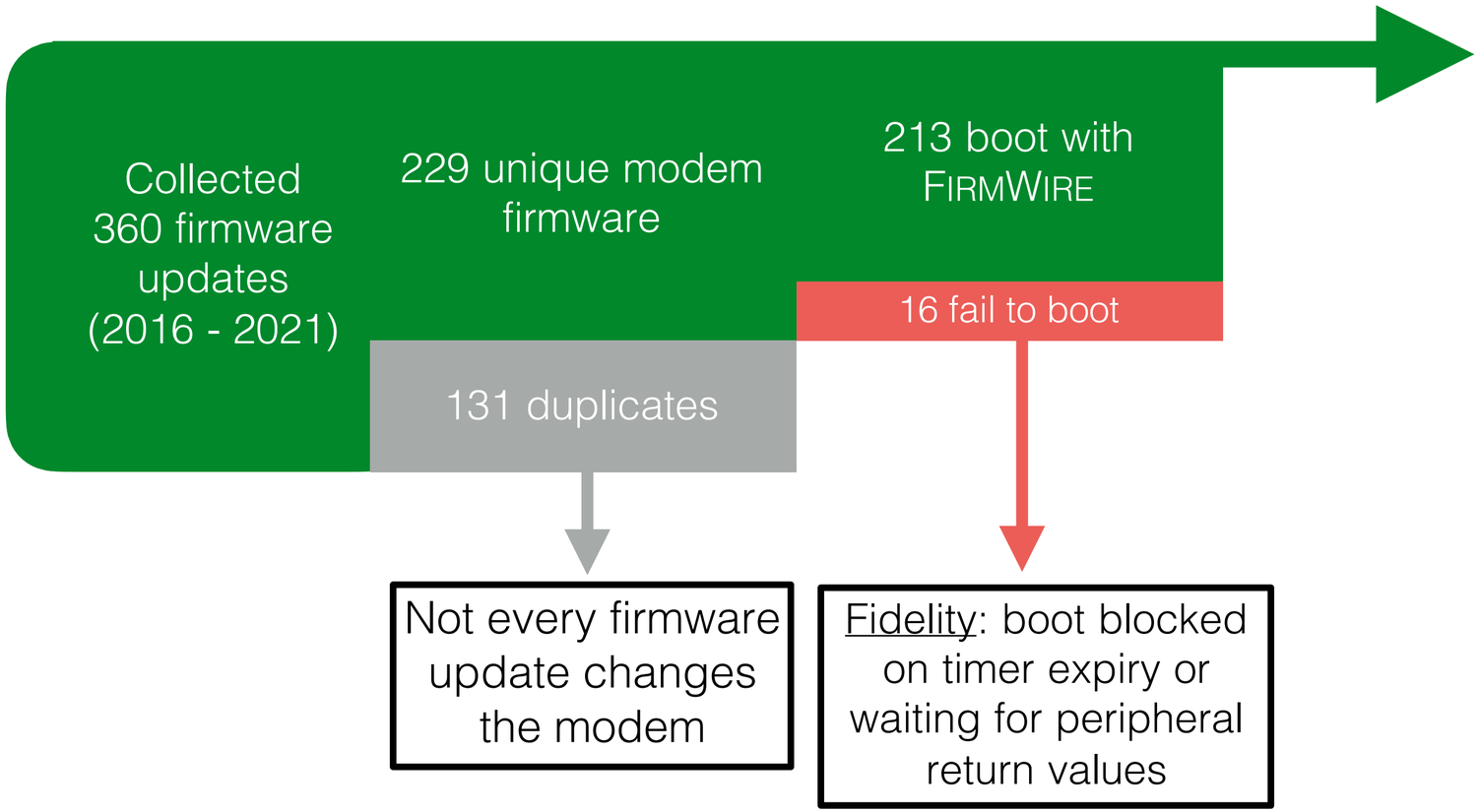
Scalability Testing



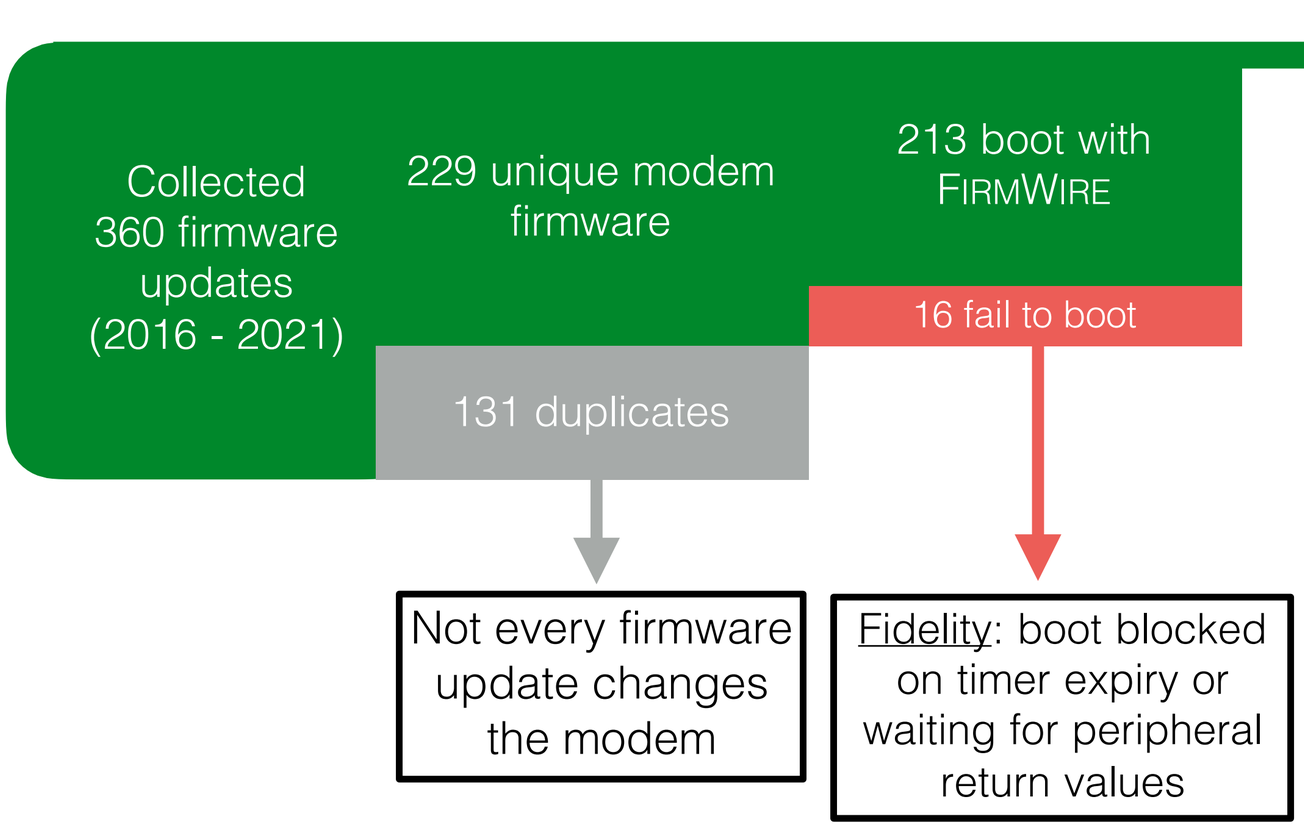
Scalability Testing



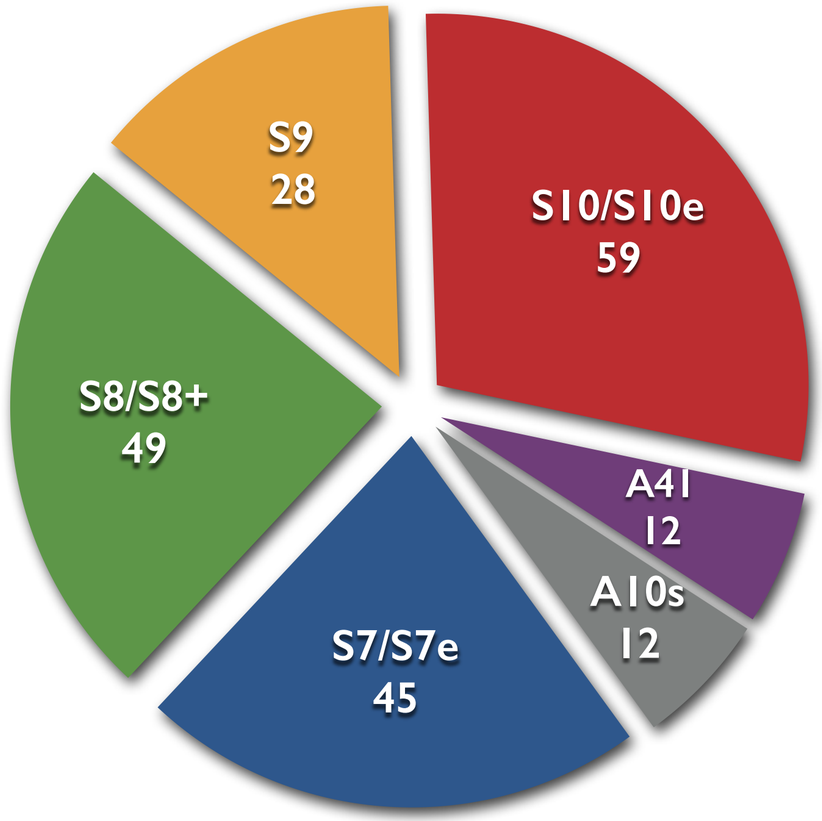
Scalability Testing



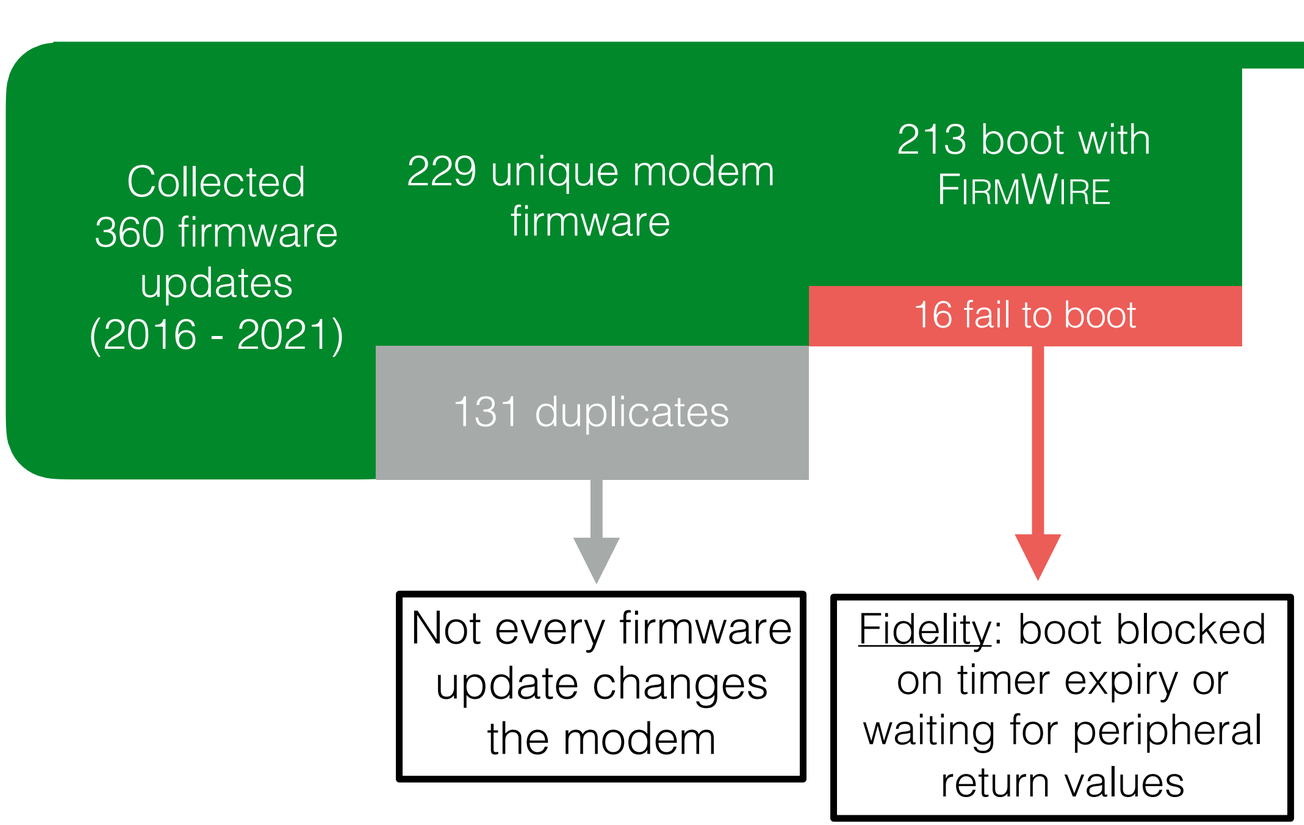
Scalability Testing



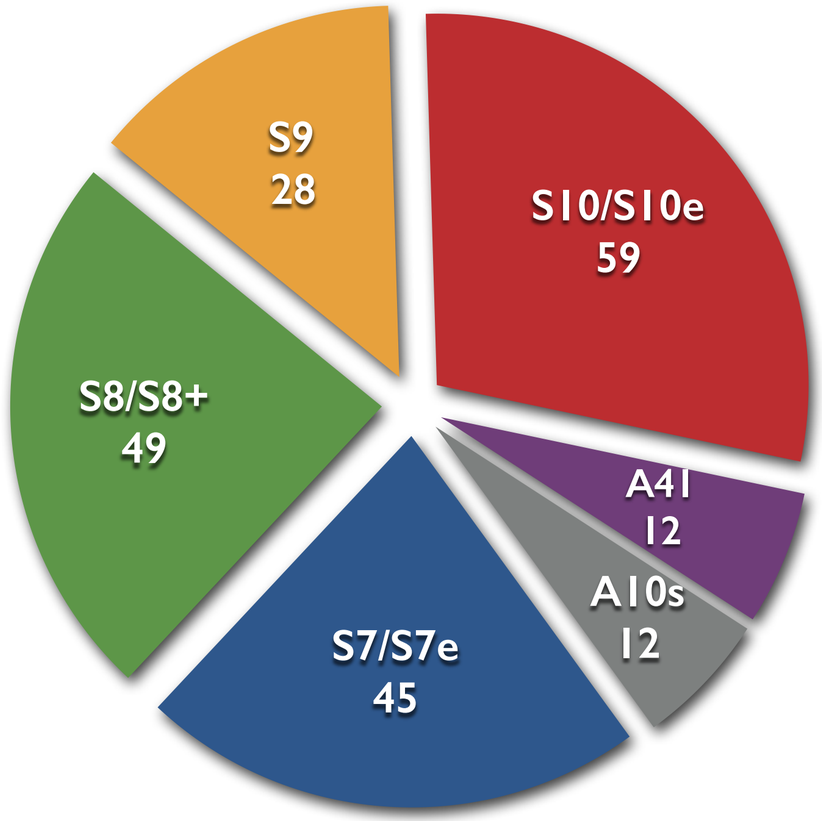
Booting firmware by model



Scalability Testing

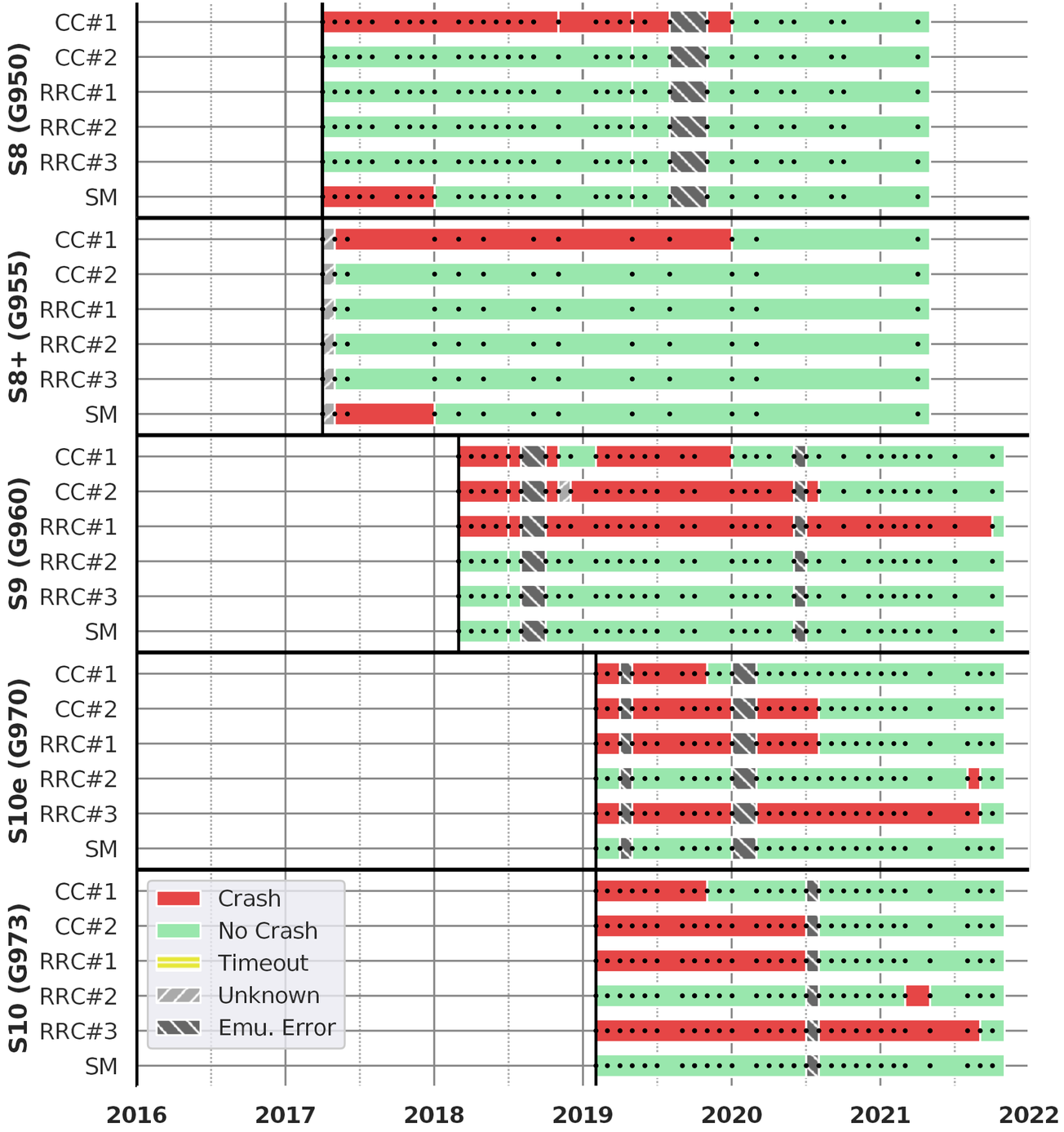


Booting firmware by model

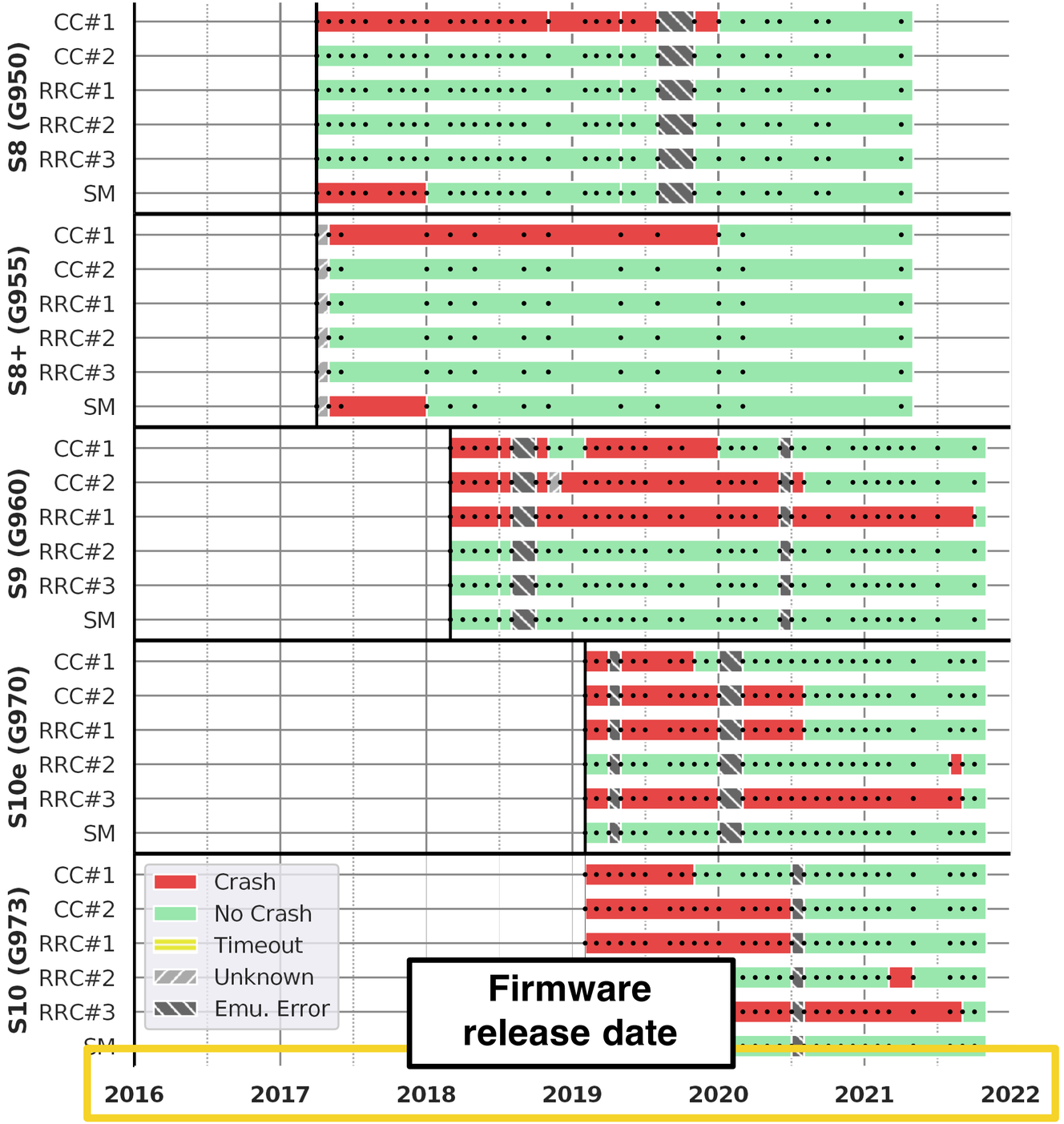


Yes. FirmWire boots 213 unique firmware across two platforms and six models with no manual intervention

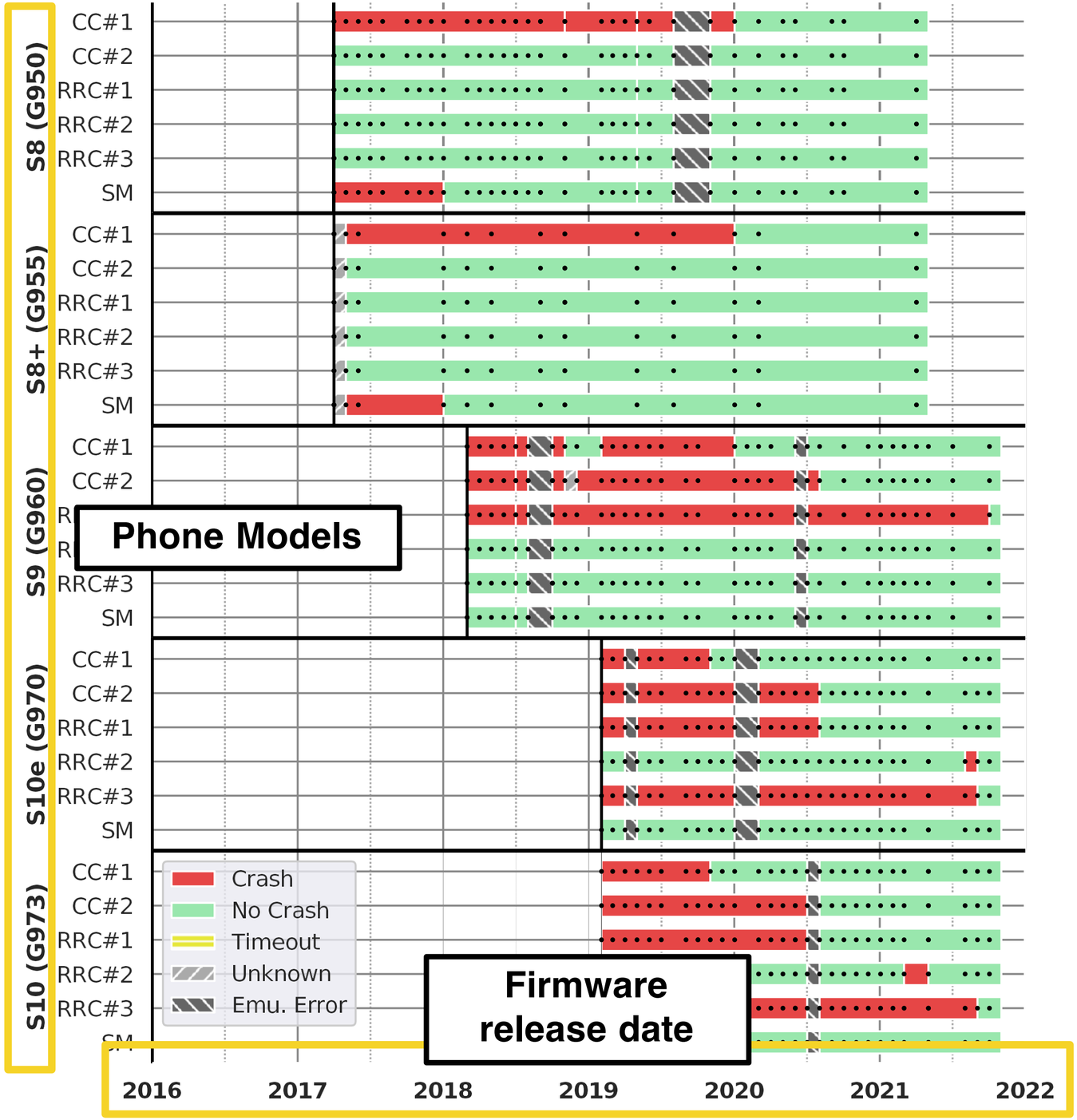
Samsung Patch Timeline



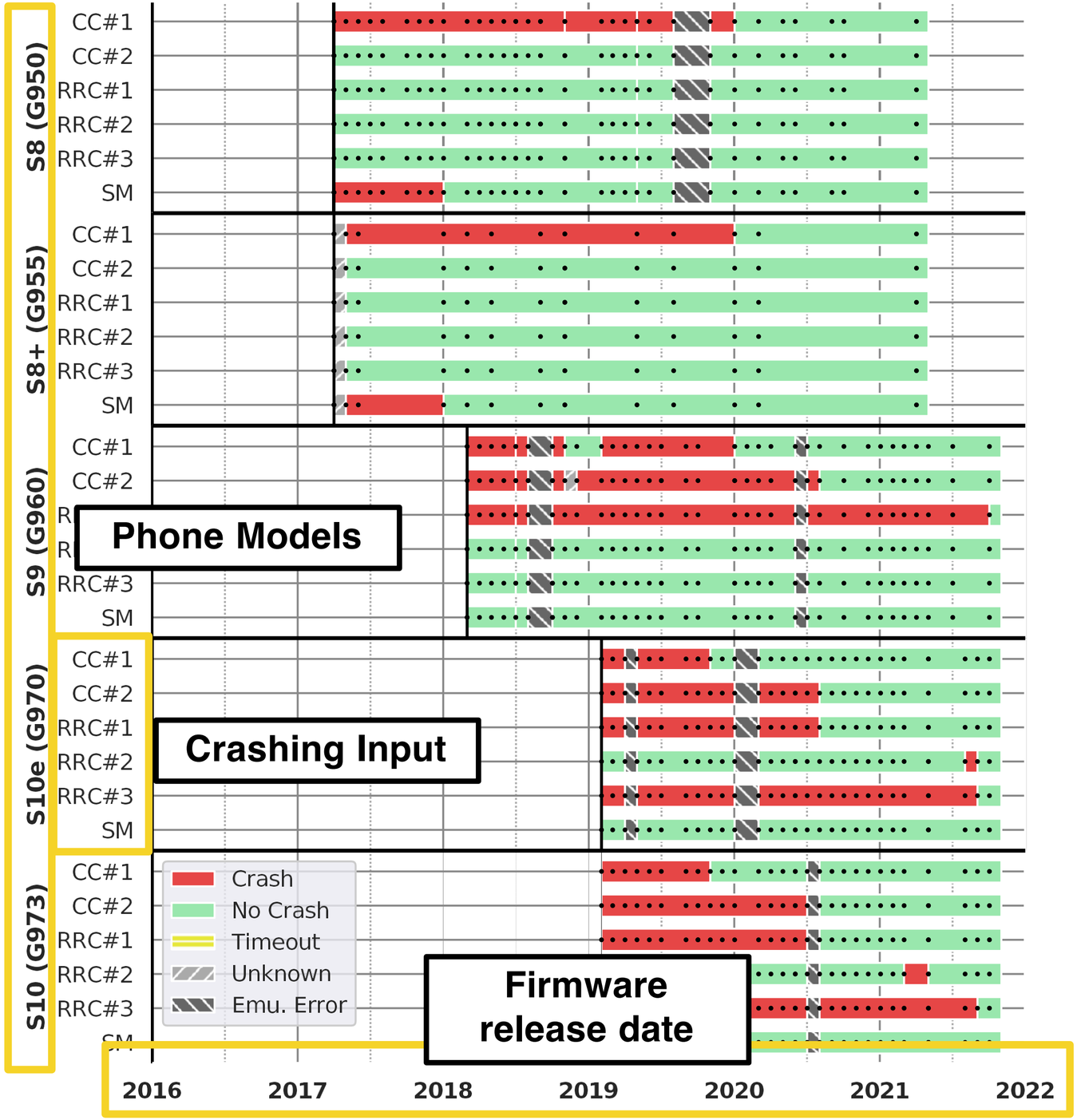
Samsung Patch Timeline



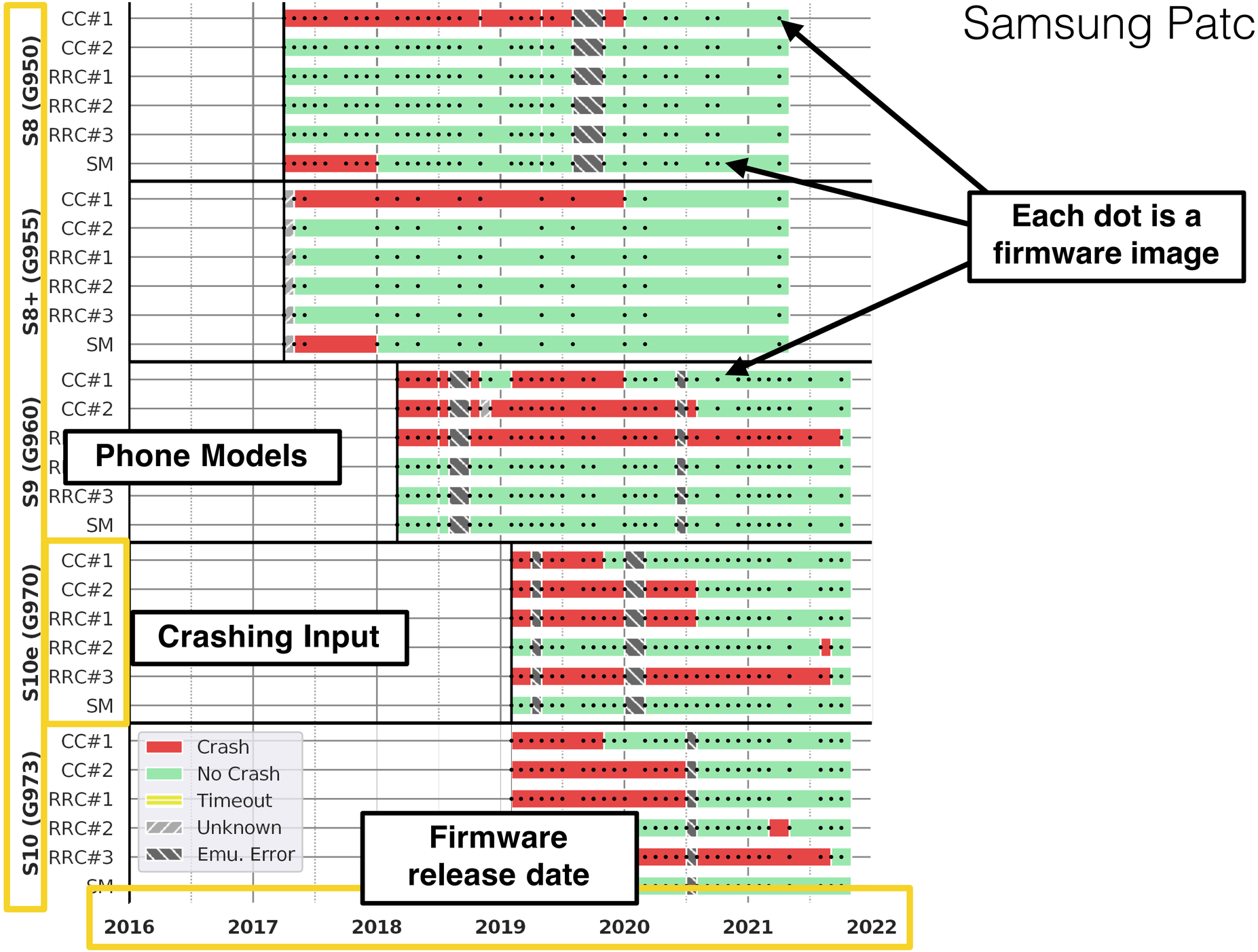
Samsung Patch Timeline



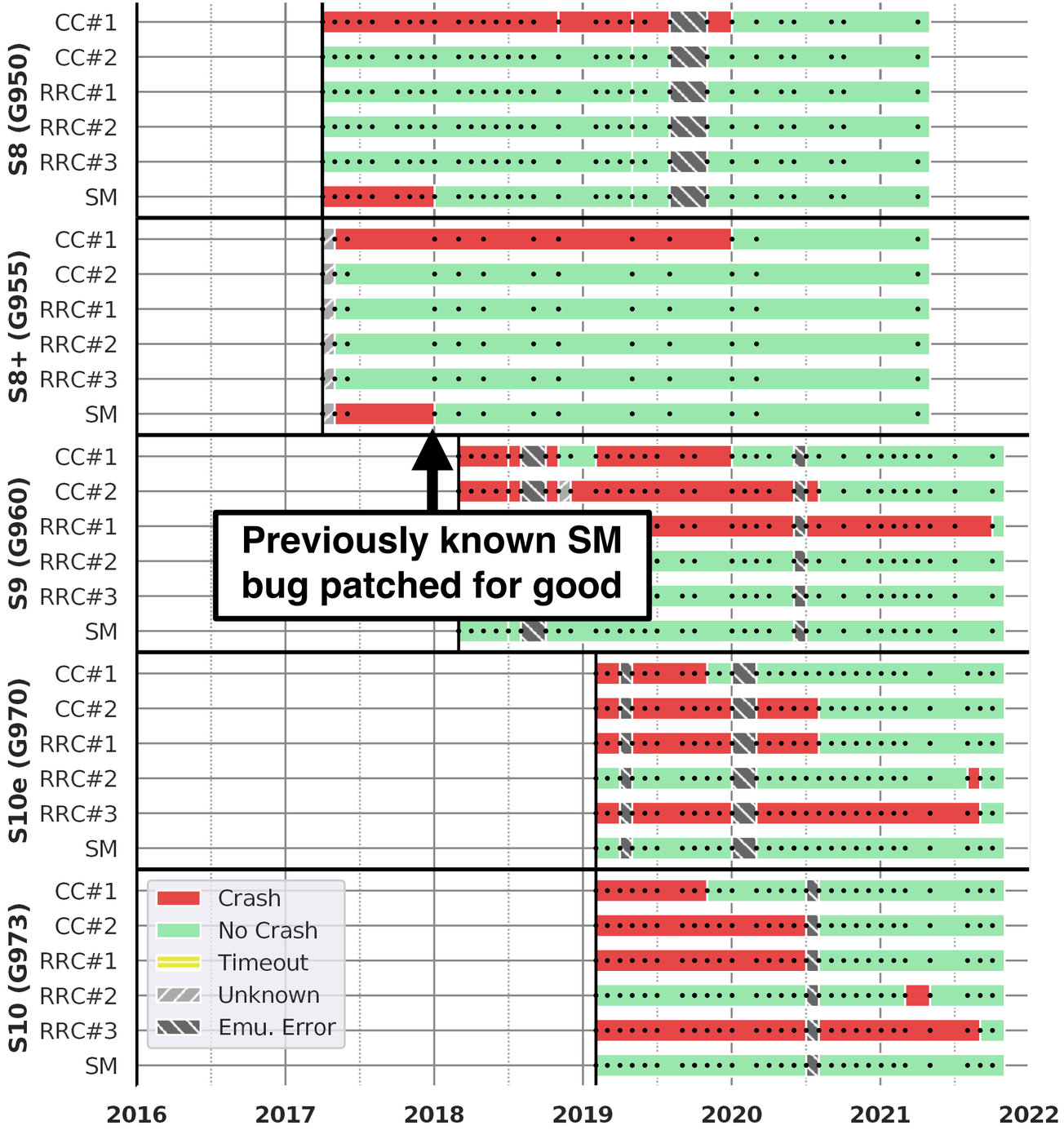
Samsung Patch Timeline



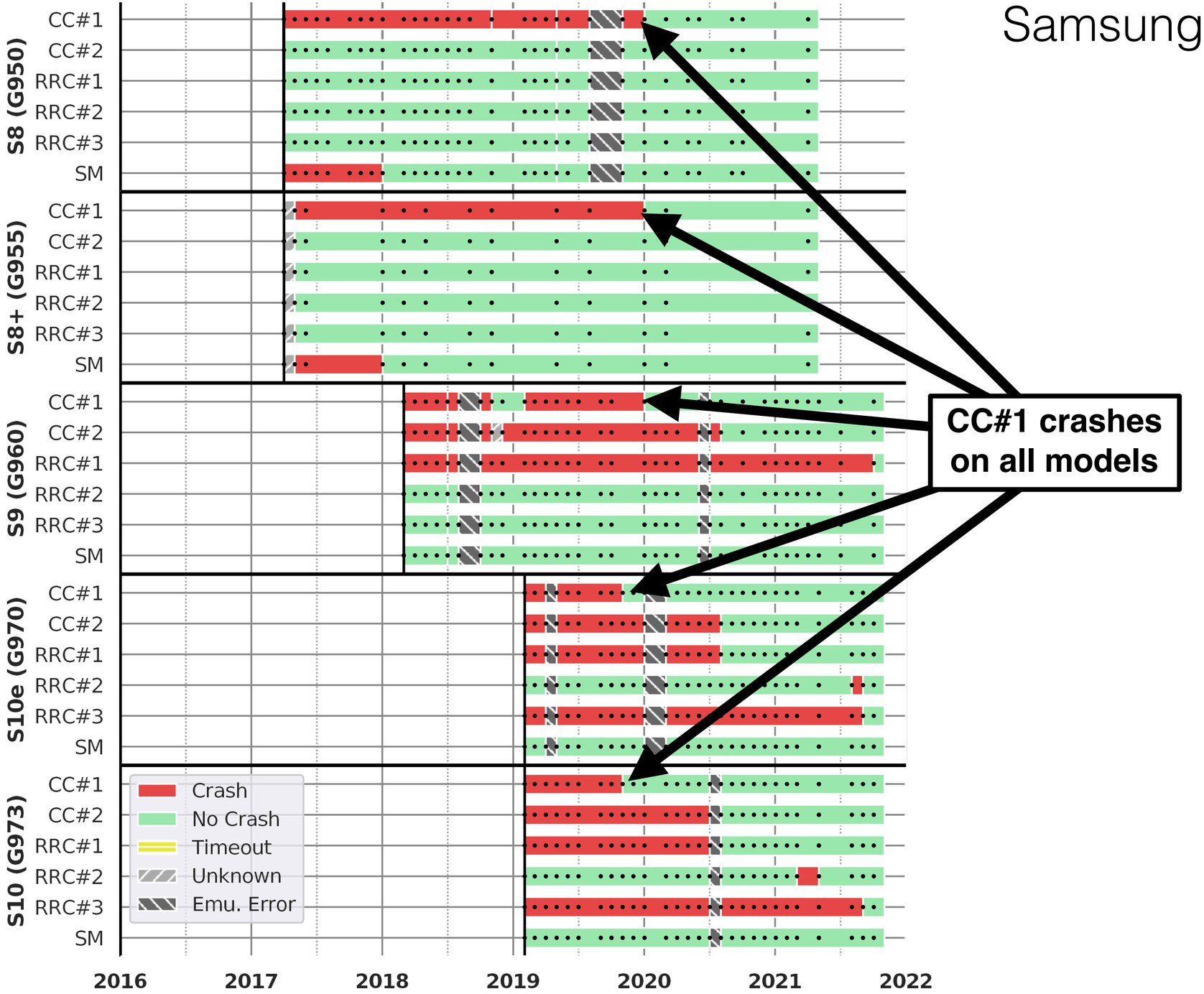
Samsung Patch Timeline



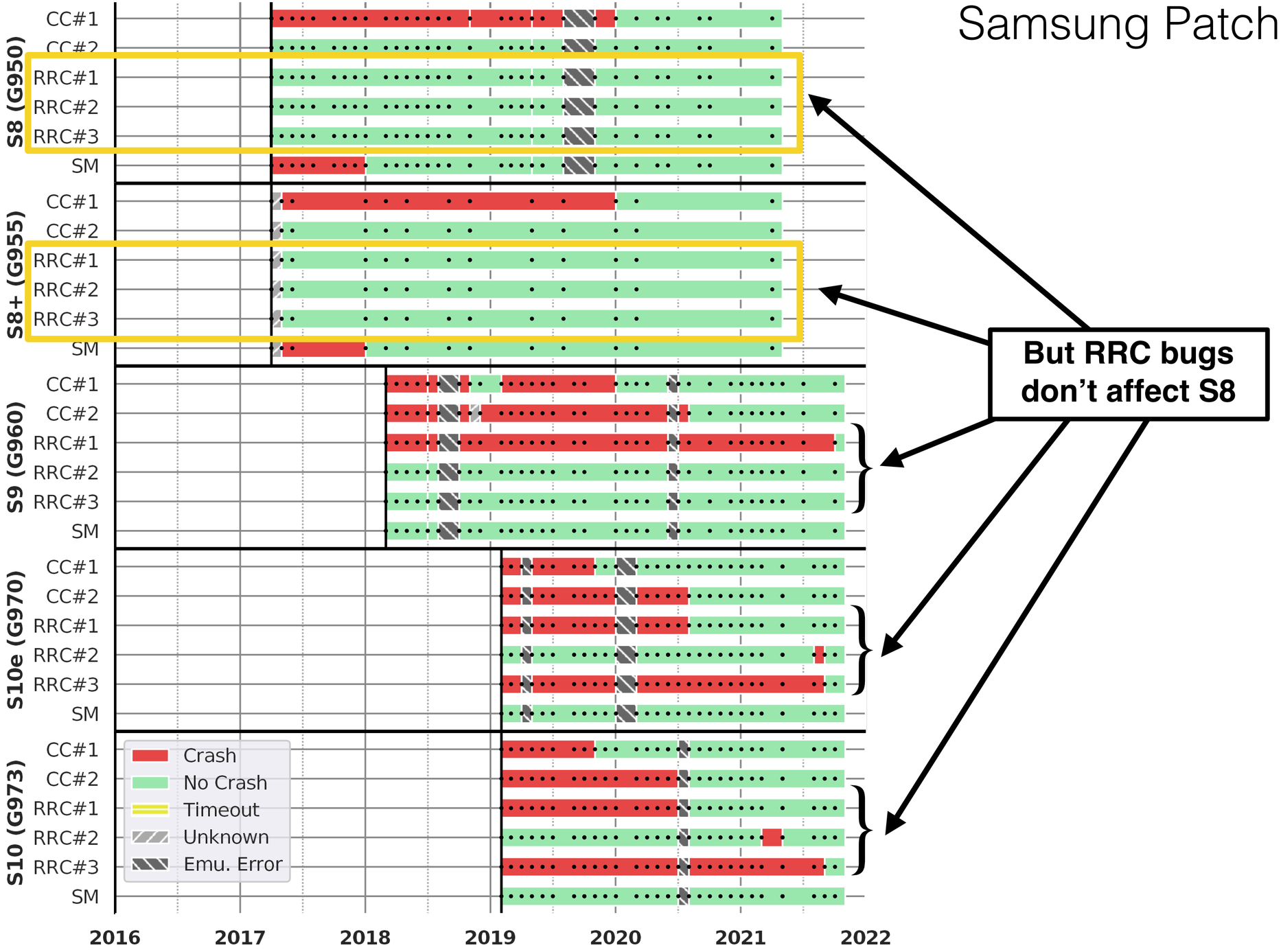
Samsung Patch Timeline



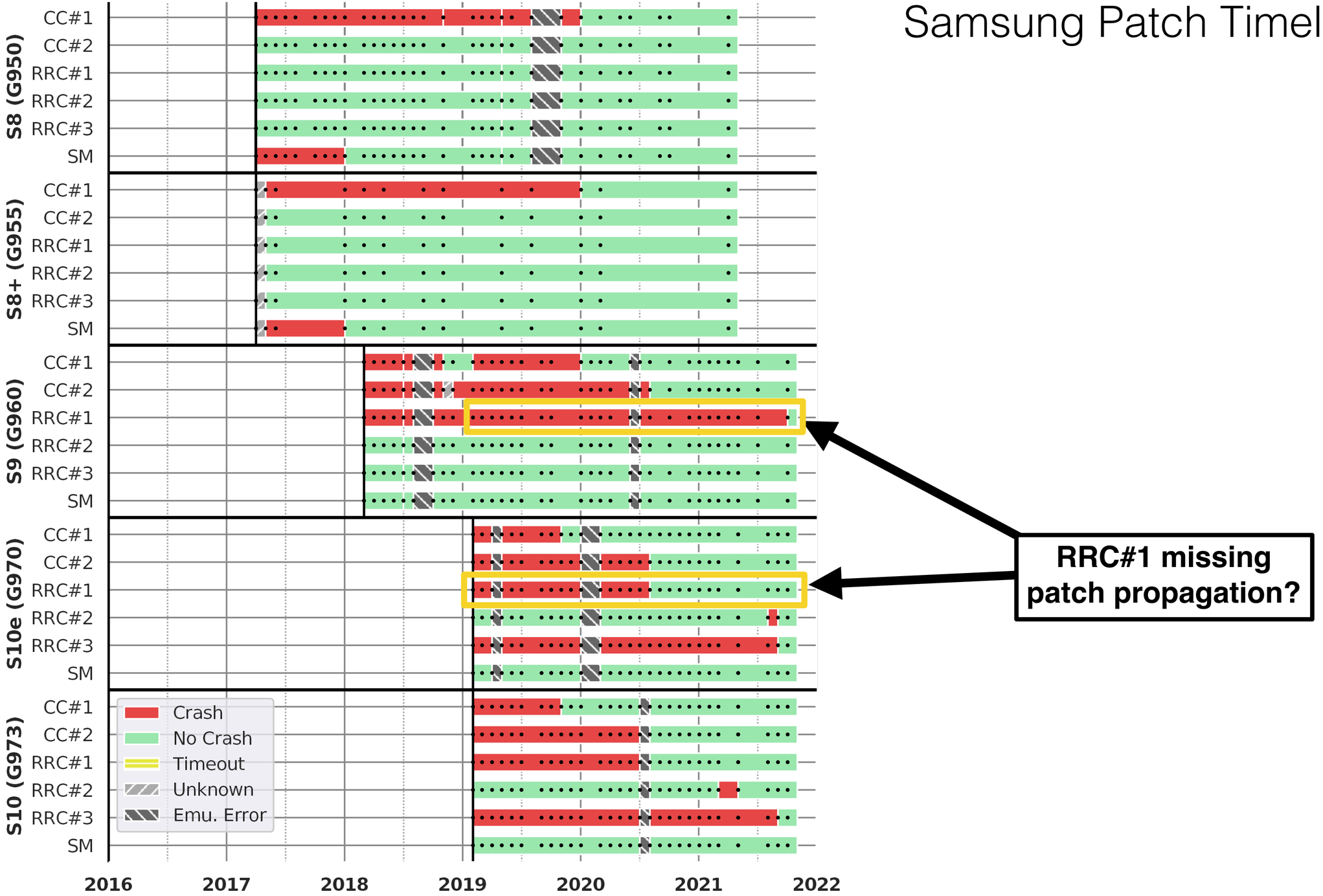
Samsung Patch Timeline



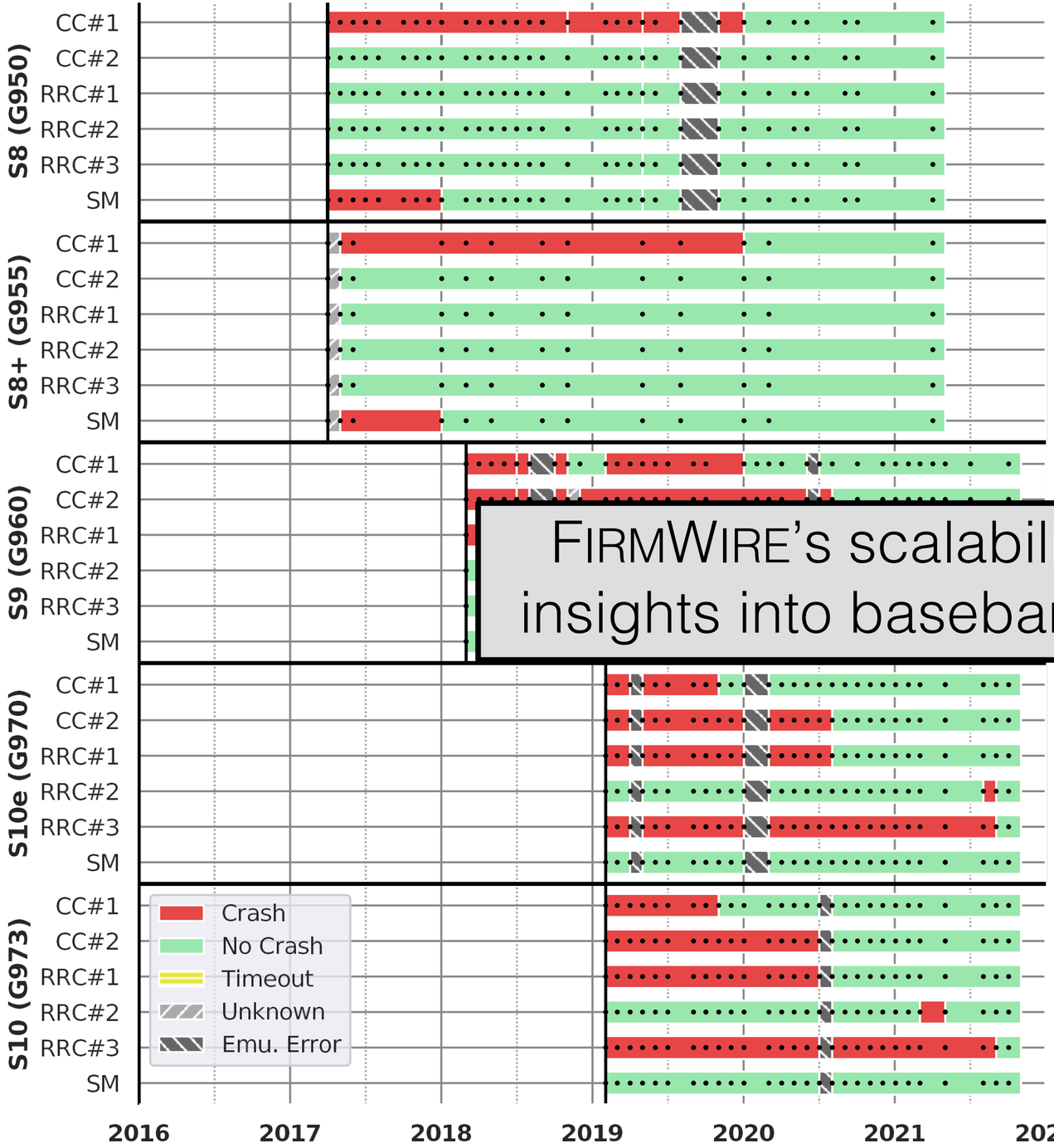
Samsung Patch Timeline



Samsung Patch Timeline



Samsung Patch Timeline



FIRMWIRE's scalability enables long-term insights into baseband health and security

FirmWire Outcomes

Released on GitHub – currently over 600 stars
Actively maintained (thanks Marius!)

Samsung Rewards Program Hall of Fame 2021
#4 overall of all security researchers

4	Team FirmWire	SVE-2021-22051	SVE-2021-22079	SVE-2021-22199

github.com/FirmWire/FirmWire

Finalist, Best Cybersecurity Research, 2022

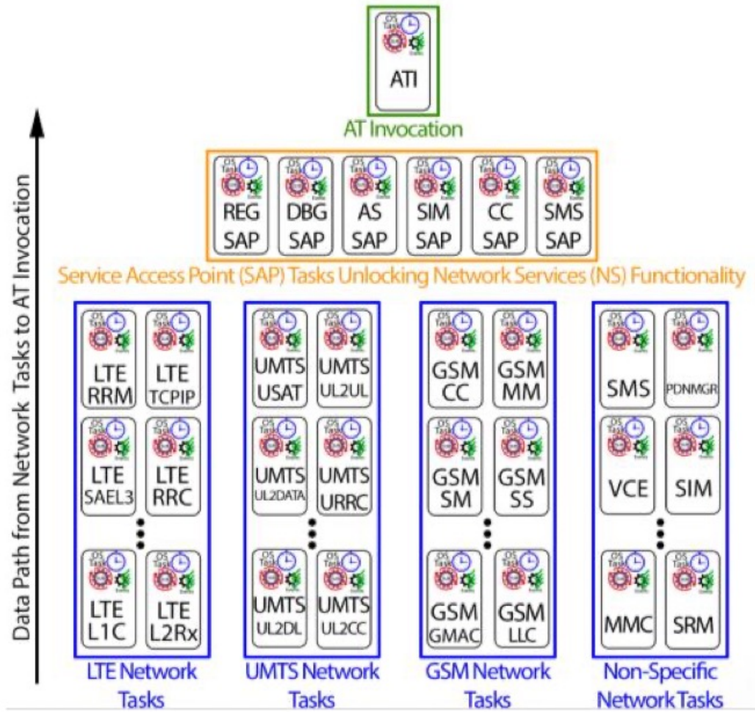
Remaining Attack Surface

- We only tested a very small fraction of protocols. Most of the time was spent on building a system that *others* could use
- Potential next targets
  - Remaining 2G/3G CS messages, EMM, ESM, 5GMM,5GSM, etc.
  - RR/RRC - CSN.1 decoding and further ASN.1 coverage SMSPP
  - IP core of modems, IP services in modems (TLS,HTTP, DNS)
  - IMS - SIP, RTP, RTCP, SDP, etc.

AT Commands Revisited

- Previous work – we know some AT commands are executed by the baseband
- Can we get more insights by using FirmWire to fuzz the AT distributor directly from the remote interface tasks?

- Fuzz protocol handlers and triage recorded crashes, look for evidence of AT command invocation from the logs



AT Commands Revisited

- Seed selection:
- 1. AT commands and crashing inputs as seeds
- 2. "Multi-message fuzzing": sequence of 10 messages send to a handler task (initialization faster than with reverse engineering)
- 3. Code coverage guiding towards AT invocation task
- 4. Concolic engine (SymQEMU)

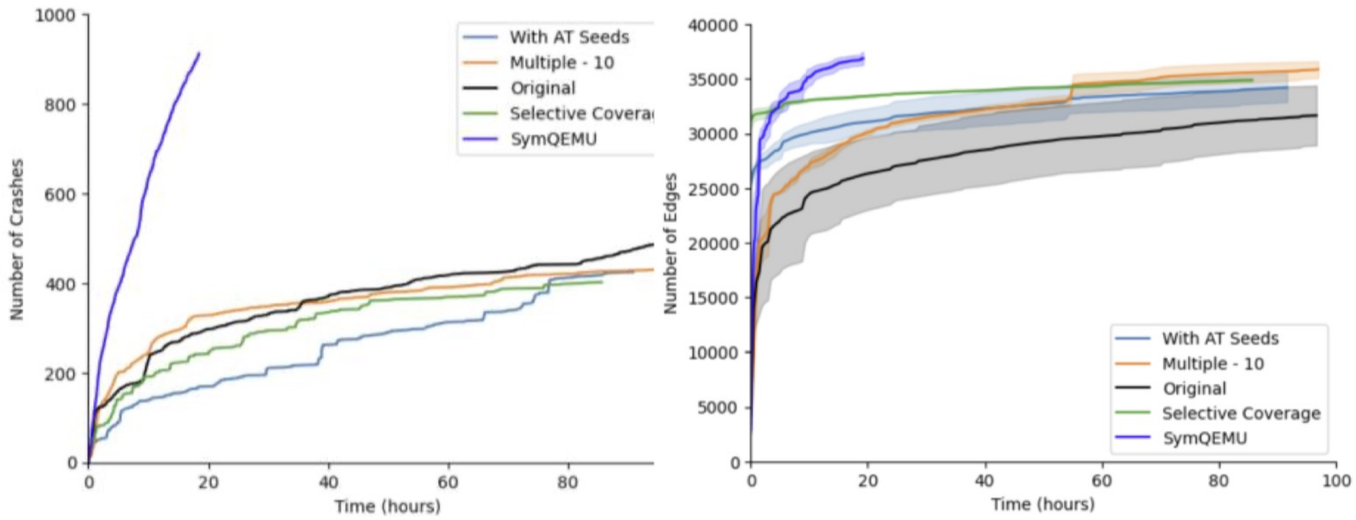


Figure: Results of fuzzing the LTE SAE L3 task.

AT Commands Revisited

- Early findings:
- 3 unique crashes when fuzzing LTE SAE L3 task (shown below)
- 3 unique crashes when fuzzing PDN manager task
- SMS task results in unique AT command invocation involved in a series of crashes
- 328 unique crashes when directly fuzzing the AT invocation task

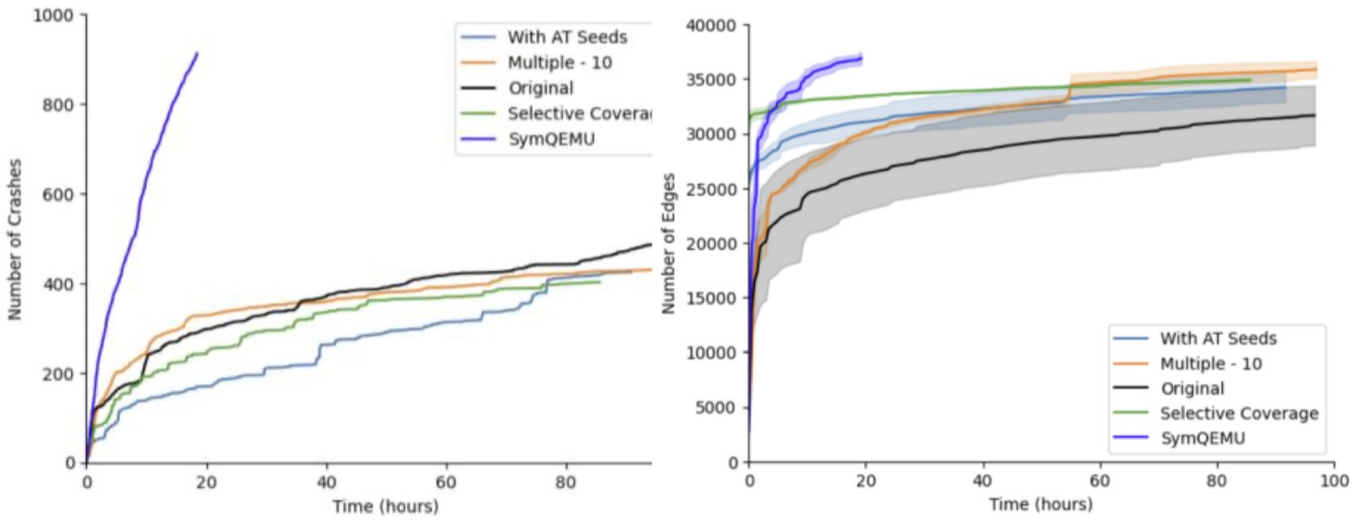


Figure: Results of fuzzing the LTE SAE L3 task.

What Have We Learned?

- New 3GPP releases ensure that new, less tested, code is always been written. Basebands support 30+ years of standards. Both of these are great for security researchers (large attack surface).

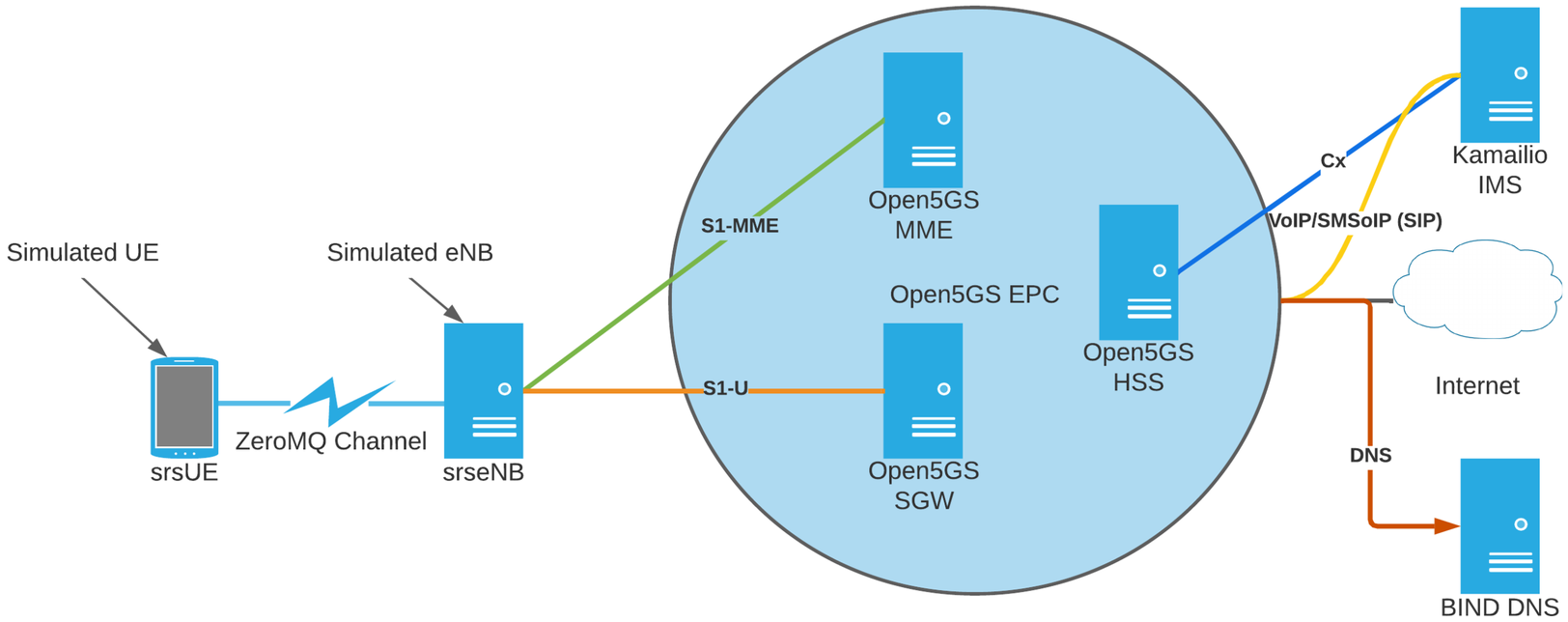
- Many critical baseband vulnerabilities are the result of memory corruption. Memory safety is key.
- Type-safe languages?

- Baseband mitigations (ASLR, NX/DEP/XN, CFI, StackGuard)?

- Time for open-source basebands? Foster a larger community?

Change the Focus

- Much of the recent research has focused on the device
- What about if we consider examining the network instead?



- Code-centric approach to understanding the core

Collaborators

Grant Hernandez

Dave (Jing) Tian

Marius Muench

Dominik Maier

Tobias Scharnowski

Alyssa Milburn

Tyler Tucker

Patrick Traynor

Collaborators/Acknowledgements

- Pirouz Naghavi
- Raghav Gupta
- Saijayanth Chidirala
- Vanessa Frost
- Sri Chandra Devarakonda
- Lee Harrison
- Mike Grace
- Jigar Patel
- Prakhar Saxena
- Yash Mundra
- Christie Ruales
- Hayawardh Vijaykumar
- Amir Rahmati

- Funding acknowledgements:
  - National Science Foundation CNS-1815883
  - Office of Naval Research ONR-OTA N00014-20-1-2205
  - Semiconductor Research Corporation
  - Air Force Office of Scientific Research
  - Dutch Research Council NOW 628.001.0303 (“TROPICS”)

Conclusion

- The closed nature of telephony networks and device has made them exciting areas for hacking over the past 50 years
- Opening the black boxes of implementations and deployments can ensure safer and more secure communications for everyone
- Lots of technical challenges for the community!

github.com/FirmWire/FirmWire

Contact:
butler@ufl.edu
https://fics.institute.ufl.edu

https://atcommands.org