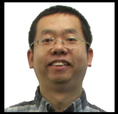
black hat®
USA 2023

AUGUST 9-10, 2023
BRIEFINGS

Over the Air, Under the Radar

Attacking and Securing the Pixel Modem


Xuan Xing

Eugene Rodionov

Xiling Gong

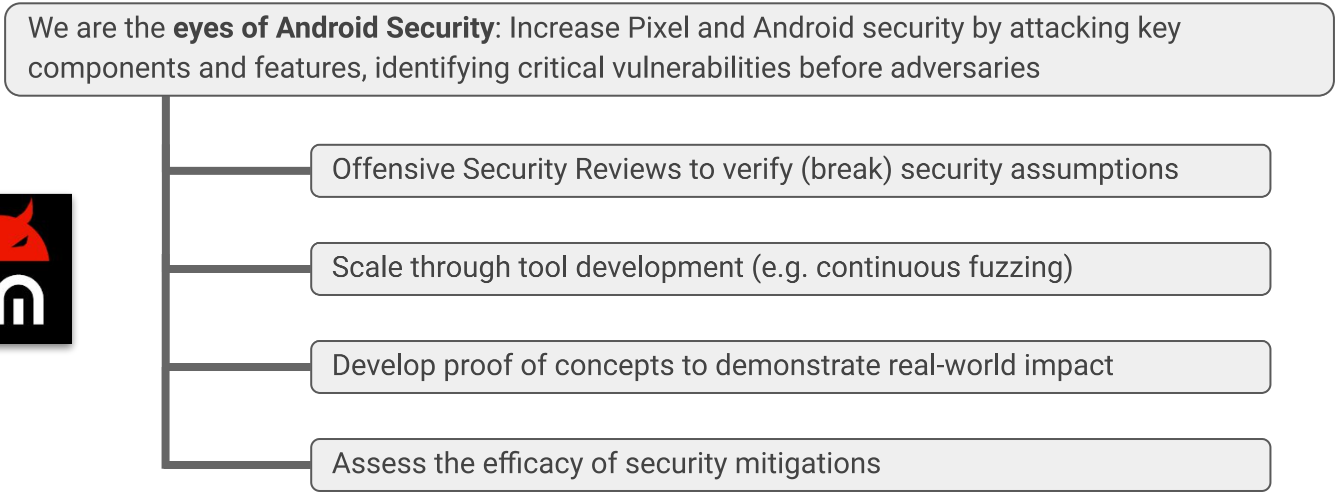
Farzan Karimi

Agenda

- Who We Are
- Pixel Modem Red Team Engagement Overview
    - Why Modem?
    - Goals & Methodology
- Proof of Concept Demonstrations
    - CVE-2022-20170
    - CVE-2022-20405
- How we secure the next generation of Pixel

All vulnerabilities mentioned in this presentation have been fixed

Mission



We are the **eyes of Android Security**: Increase Pixel and Android security by attacking key components and features, identifying critical vulnerabilities before adversaries

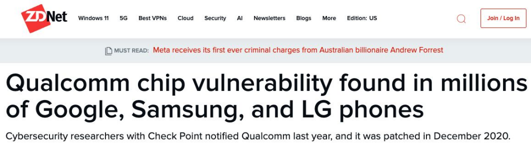
- Offensive Security Reviews to verify (break) security assumptions
- Scale through tool development (e.g. continuous fuzzing)
- Develop proof of concepts to demonstrate real-world impact
- Assess the efficacy of security mitigations

ANDROID RED TEAM

Why Modem?

2019-2023

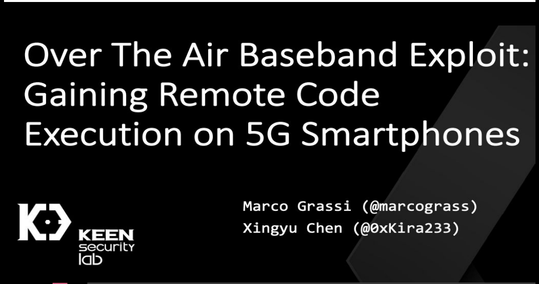
Modem has been an emerging area of risk



MUST READ: Meta receives its first ever criminal charges from Australian billionaire Andrew Forrest

Qualcomm chip vulnerability found in millions of Google, Samsung, and LG phones

Cybersecurity researchers with Check Point notified Qualcomm last year, and it was patched in December 2020.

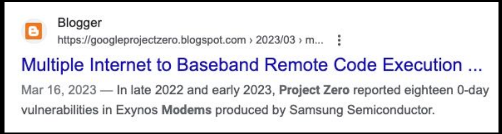


Over The Air Baseband Exploit: Gaining Remote Code Execution on 5G Smartphones

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

Advanced SMS Phishing Attacks Against Modern Android-based Smartphones

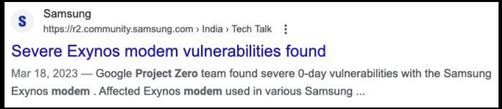
September 4, 2019



Blogger
https://googleprojectzero.blogspot.com › 2023/03 › m...

Multiple Internet to Baseband Remote Code Execution ...

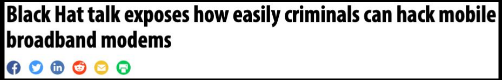
Mar 16, 2023 — In late 2022 and early 2023, **Project Zero** reported eighteen 0-day vulnerabilities in Exynos **Modems** produced by Samsung Semiconductor.



Samsung
https://r2.community.samsung.com › India › Tech Talk

Severe Exynos modem vulnerabilities found

Mar 18, 2023 — Google **Project Zero** team found severe 0-day vulnerabilities with the Samsung Exynos **modem** . Affected Exynos **modem** used in various Samsung ...



Black Hat talk exposes how easily criminals can hack mobile broadband modems

Samsung Smartphones Already Received Modem Vulnerability Patch

By Teja Chedalla May 12, 2021

The vulnerability in Qualcomm modems affects 30 percent of mobile phones

So What?

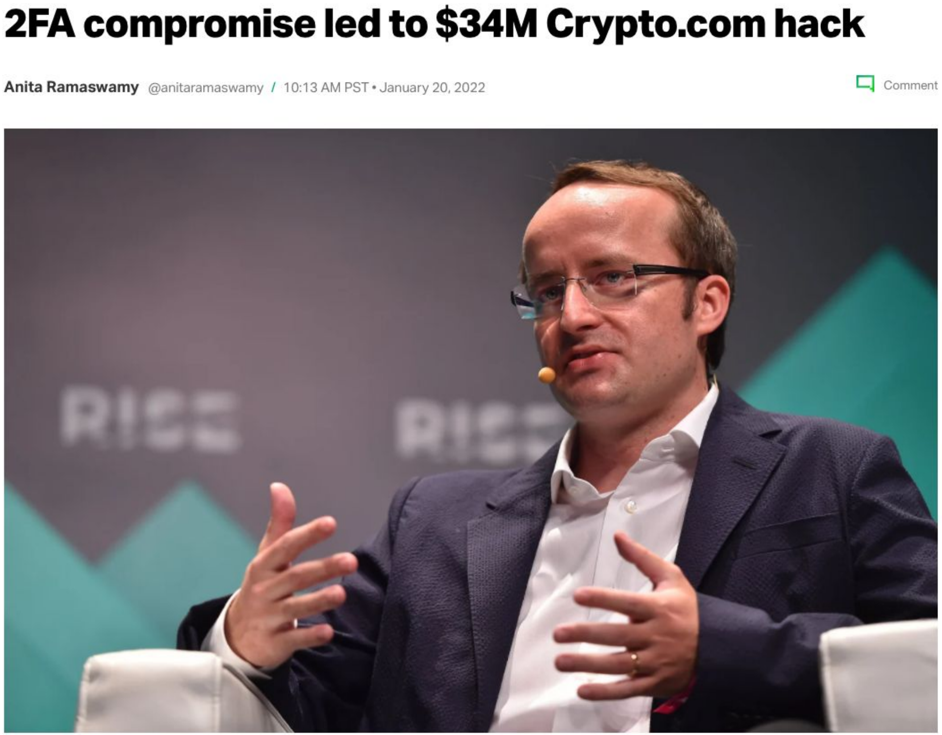
What an attacker would get:

- Over-the-air Remote Code Execution
- Running in Privileged Context

What that means:

- DDoS Botnet
- SMS/RCA Sniffing and Spoofing
- MFA Compromise
- Pivot Opportunities to Kernel

So What?



2FA compromise led to $34M Crypto.com hack

Anita Ramaswamy @anitaramaswamy / 10:13 AM PST • January 20, 2022

Comment

Engagement Overview

Timeline:

- Multi-month Android Red Team engagement from late 2021 to early 2022

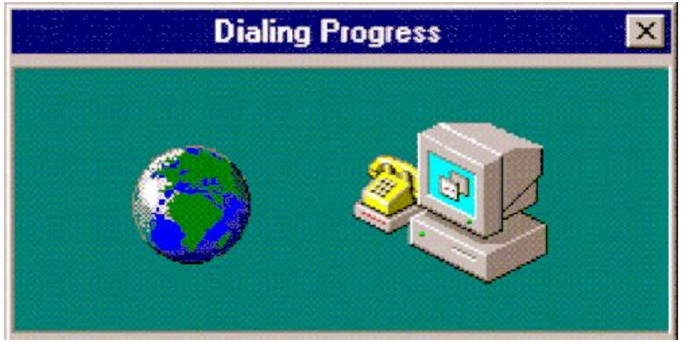
Mission

- Gain remote code execution on baseband via the Pixel 6 modem stack ✓
- Suggest systemic security improvements to harden the Pixel 6+ modem ✓
- Bonus: Get everything patched before debrief ✓

Modem Overview

Modem Overview

Modem at a glance:



Modem Overview

Modem at a glance:

- A critical component with access to sensitive user data
- Remotely accessible with various radio technologies
- A high profile target which could benefit from security hardening mitigations
- A historical source of vulnerabilities from external researchers and modem owners
- Many legacy protocols with outdated security practice

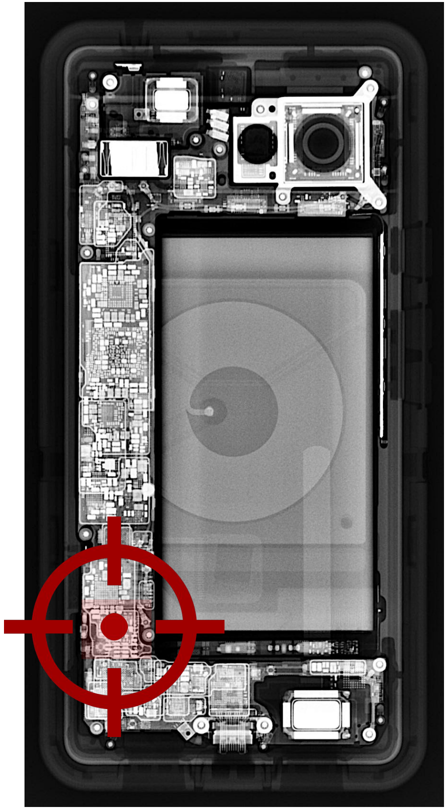
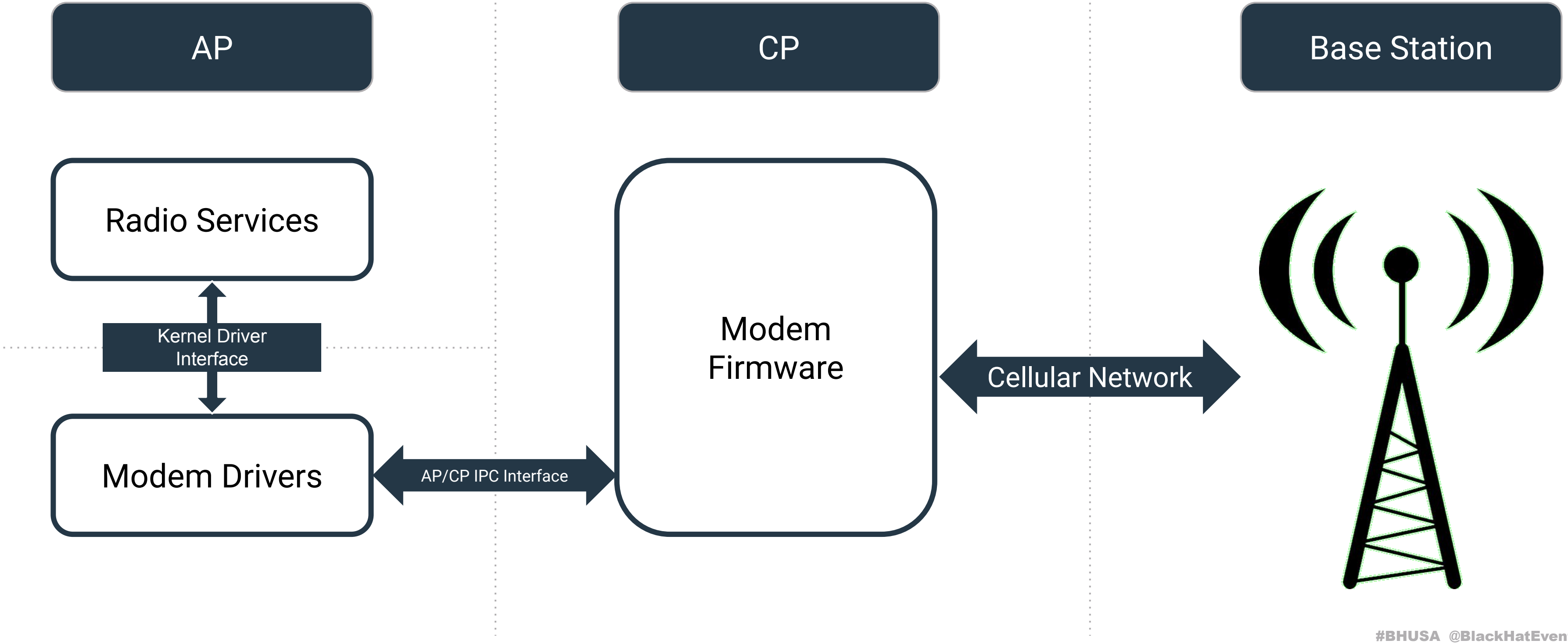
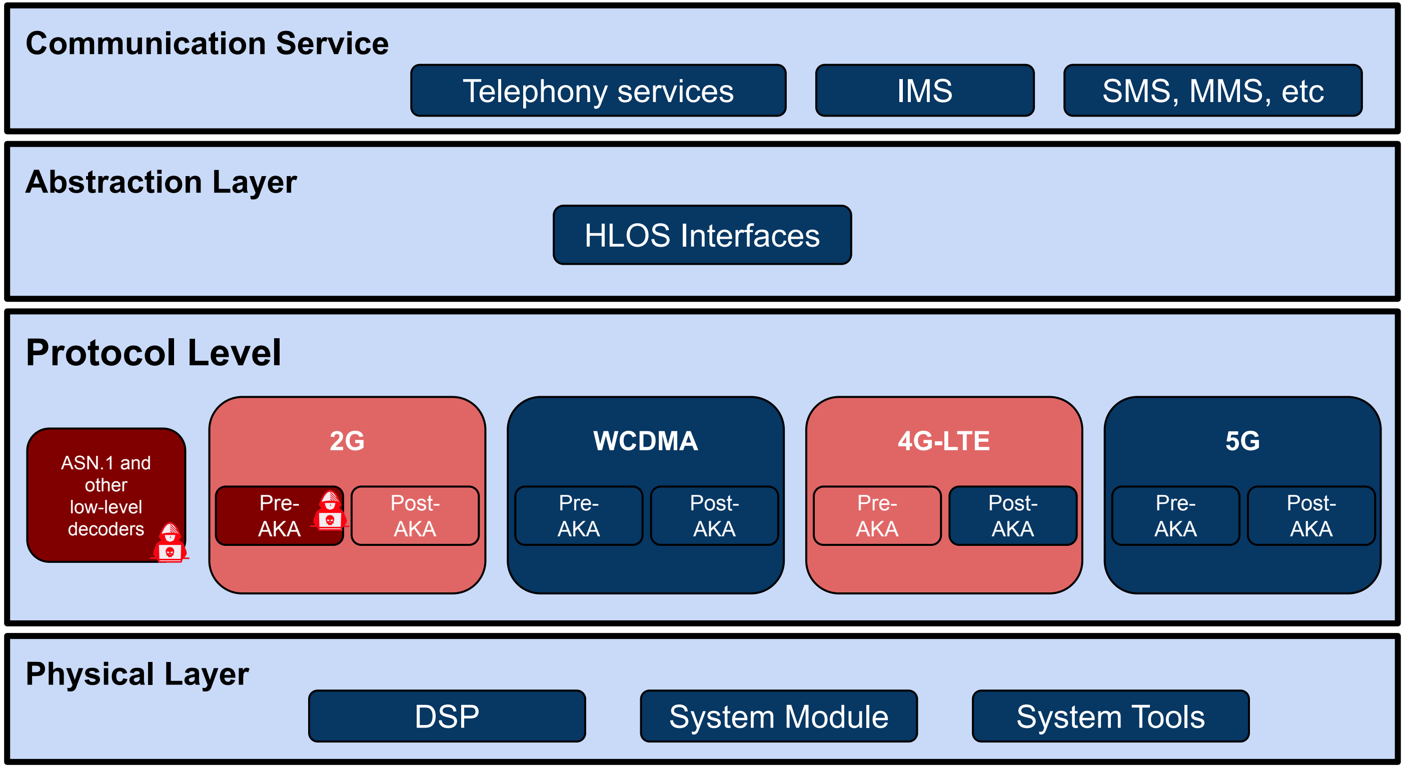


Image Credit: Pixel 6 X-ray Imablog

Modem Overview



Modem Overview



Communication Service

- Telephony services
- IMS
- SMS, MMS, etc

Abstraction Layer

- HLOS Interfaces

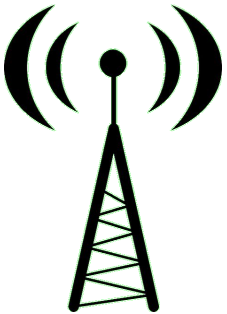
Protocol Level

- ASN.1 and other low-level decoders
- 2G: Pre-AKA, Post-AKA
- WCDMA: Pre-AKA, Post-AKA
- 4G-LTE: Pre-AKA, Post-AKA
- 5G: Pre-AKA, Post-AKA

Physical Layer

- DSP
- System Module
- System Tools

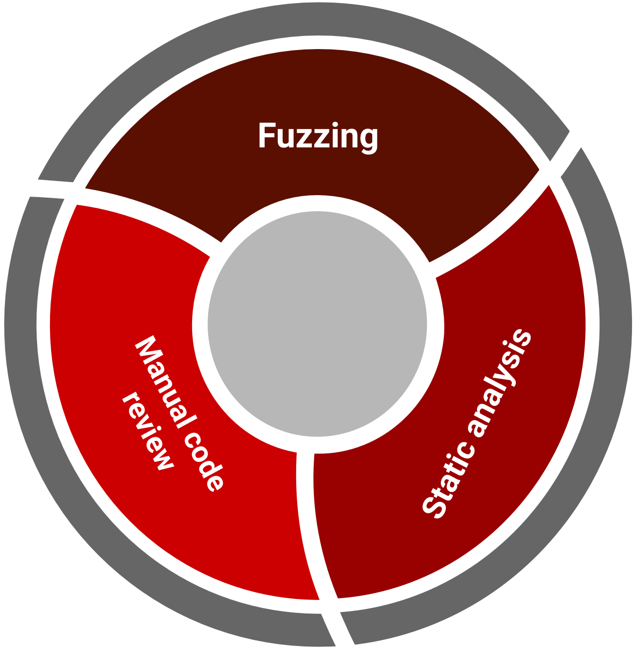
LEGEND
Layer
Component
Attack Surface Covered by Android Red Team
Proof of Concept



Our Methodology

Evaluation Approaches

- **Fuzzing as the primary approach**
  - Host based fuzzing has been proven effective during first modem engagement
  - Full system emulation is complete
  - On-device fuzzing was cut due to schedule constraint
- **Static analysis using CodeQL**
  - Exploring modem codebase
  - Variant analysis
- **Manual code review**
  - Only for areas identified by fuzzing or external researches



Fuzzing Overview

Progress:

- 10 fuzzers created during the engagement and running on our internal at-scale device fuzzing platform.
- Fuzzers not only find great bugs, but also identify high risk areas for manual code review.
- Developing an easy to use framework for host based modem fuzzing.

Fuzzing Challenges:

- Low severity bugs blocking fuzzing from continuing
- Complex dependencies to other components
- Tasks dealing with internal messages no value for fuzzing

Fuzzer Name	**Description**
AsnDecoder	Targets ASN.1 decoder which reads and translates data encoded in ASN.1 format by feeding malformed inputs. ASN.1 is widely used in various protocols and data formats
CdParseMsg	Targets parser responsible for processing and interpreting messages received by the modem from external sources
More fuzzers…	More protocols…

CodeQL

CodeQL Overview:

CodeQL is a static analysis tool with powerful data-flow and taint analysis engine to find code errors, check code quality, and identify vulnerabilities.

Modem Exploration Queries:

- Finding all task entry points
- Finding all Low-level Interrupt Service Routines (LISRs)
- Finding all High-level Interrupt Service Routines (HISRs)
- Graphing IPC between different tasks

General purpose bug finding queries:

- Identifying `memcpy` which write to a fixed-size buffer, but use a non-constant size argument
- Identifying `for` loops writing to buffers, where the loop could iterate more times than the size of the buffer

Modem Emulator

Technical Spec

- **Unicorn-base full-stack emulation**
  - Supports 5G Modem Chipset (Shannon 5123)
- **Emulates some hardware layers**
  - Hardware Registers
  - PCIE interface
  - OTP
  - Flash Memory (RFS)
- **Software layer functionalities**
  - Process snapshot and restore - useful for high-speed fuzzing
  - ASAN-style instrumentation

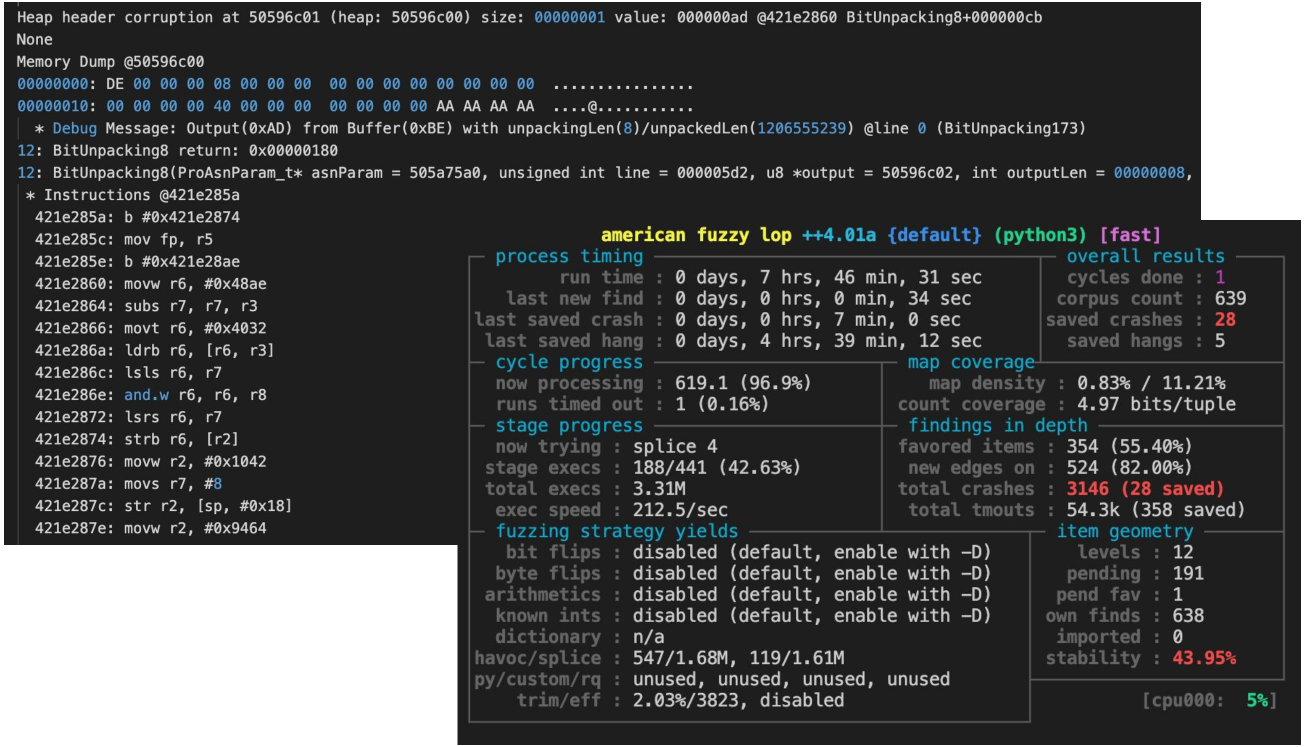
Benefits & Usages

- Accurate emulation with full symbols vs FirmWire with guessed limited symbols

- Fuzzing - AFLPlusPlus unicorn mode integration
  - Better code coverage

- Root Cause Analysis
  - Triaging & Investigation
  - Accurate and fast crash investigation

Modem Emulator
Root Cause Analysis



```
Heap header corruption at 50596c01 (heap: 50596c00) size: 00000001 value: 000000ad @421e2860 BitUnpacking8+000000cb
None
Memory Dump @50596c00
00000000: DE 00 00 00 08 00 00 00  00 00 00 00 00 00 00 00  ................
00000010: 00 00 00 00 40 00 00 00  00 00 00 00 AA AA AA AA  ....@...........
 | * Debug Message: Output(0xAD) from Buffer(0xBE) with unpackingLen(8)/unpackedLen(1206555239) @line 0 (BitUnpacking173)
12: BitUnpacking8 return: 0x00000180
12: BitUnpacking8(ProAsnParam_t* asnParam = 505a75a0, unsigned int line = 000005d2, u8 *output = 50596c02, int outputLen = 00000008,
 | * Instructions @421e285a
   421e285a: b #0x421e2874
   421e285c: mov fp, r5
   421e285e: b #0x421e28ae
   421e2860: movw r6, #0x48ae
   421e2864: subs r7, r7, r3
   421e2866: movt r6, #0x4032
   421e286a: ldrb r6, [r6, r3]
   421e286c: lsls r6, r7
   421e286e: and.w r6, r6, r8
   421e2872: lsrs r6, r7
   421e2874: strb r6, [r2]
   421e2876: movw r2, #0x1042
   421e287a: movs r7, #8
   421e287c: str r2, [sp, #0x18]
   421e287e: movw r2, #0x9464
```

```
              american fuzzy lop ++4.01a {default} (python3) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 7 hrs, 46 min, 31 sec     │  cycles done : 1     │
│   last new find : 0 days, 0 hrs, 0 min, 34 sec      │ corpus count : 639   │
│last saved crash : 0 days, 0 hrs, 7 min, 0 sec       │saved crashes : 28    │
│ last saved hang : 0 days, 4 hrs, 39 min, 12 sec     │  saved hangs : 5     │
├─ cycle progress ─────────────────────┬─ map coverage┴──────────────────────┤
│  now processing : 619.1 (96.9%)      │    map density : 0.83% / 11.21%    │
│  runs timed out : 1 (0.16%)          │ count coverage : 4.97 bits/tuple   │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : splice 4               │ favored items : 354 (55.40%)       │
│ stage execs : 188/441 (42.63%)       │  new edges on : 524 (82.00%)       │
│ total execs : 3.31M                  │ total crashes : 3146 (28 saved)    │
│  exec speed : 212.5/sec              │  total tmouts : 54.3k (358 saved)  │
├─ fuzzing strategy yields ────────────┴─────────────┬─ item geometry ───────┤
│   bit flips : disabled (default, enable with -D)   │    levels : 12        │
│  byte flips : disabled (default, enable with -D)   │   pending : 191       │
│ arithmetics : disabled (default, enable with -D)   │  pend fav : 1         │
│  known ints : disabled (default, enable with -D)   │ own finds : 638       │
│  dictionary : n/a                                  │  imported : 0         │
│havoc/splice : 547/1.68M, 119/1.61M                 │ stability : 43.95%    │
│py/custom/rq : unused, unused, unused, unused       ├───────────────────────┘
│    trim/eff : 2.03%/3823, disabled                 │          [cpu000:  5%]
└────────────────────────────────────────────────────┘
```

Our Findings

Re: ASN.1
“Maybe all the bugs are gone…?”

How to Hack Shannon Baseband (from a Phone)
OffensiveCon Presentation by Google Project Zero (May, 2023)

(~12 months after the Android Red Team Engagement)

Findings Summary

By the numbers:

122
Total Issues

18%
Critical/High Severity

50
Fuzzer Bugs

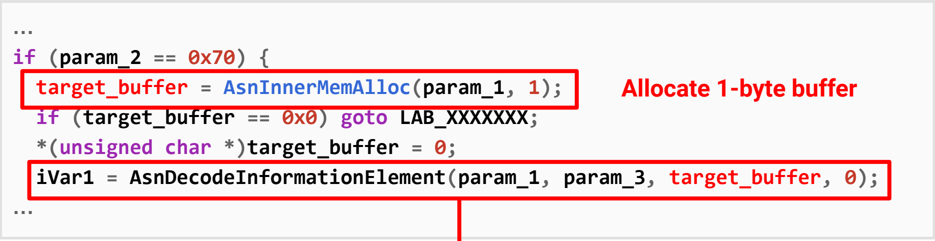
Two bugs in particular stood out in this engagement, and when chained, led to a Modem RCE.

- **CVE-2022-20170** is a critical severity issue. This is an OOB write issue that occurs when decoding the OTA packets from 2G (GSM).
- **CVE-2022-20405** is a moderate severity issue that is the result of a mis-configuration in modem code makes most of the memory space with RWX.

All vulnerabilities mentioned in this presentation have been fixed

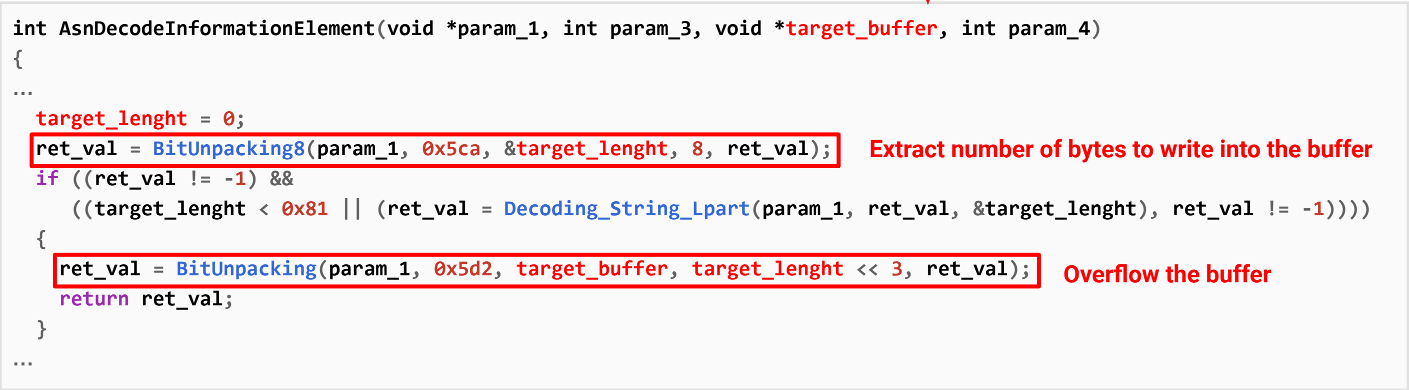
CVE-2022-20170 Details

- Linear OOB write in the heap
- Happens during ASN.1 parsing of Information Element during call setup stage in 2G stack
- The attacker fully controls up to 255 bytes written into 1-byte buffer in the heap



```
...
if (param_2 == 0x70) {
  target_buffer = AsnInnerMemAlloc(param_1, 1);
  if (target_buffer == 0x0) goto LAB_XXXXXXX;
  *(unsigned char *)target_buffer = 0;
  iVar1 = AsnDecodeInformationElement(param_1, param_3, target_buffer, 0);
...
```

Allocate 1-byte buffer



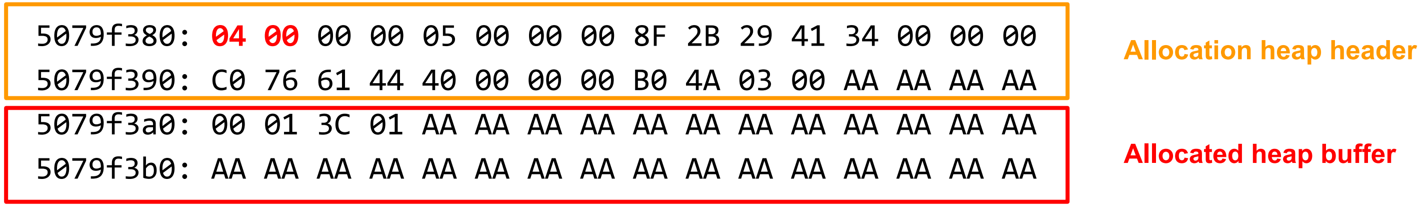
```
int AsnDecodeInformationElement(void *param_1, int param_3, void *target_buffer, int param_4)
{
...
  target_lenght = 0;
  ret_val = BitUnpacking8(param_1, 0x5ca, &target_lenght, 8, ret_val);
  if ((ret_val != -1) &&
     ((target_lenght < 0x81 || (ret_val = Decoding_String_Lpart(param_1, ret_val, &target_lenght), ret_val != -1))))
  {
    ret_val = BitUnpacking(param_1, 0x5d2, target_buffer, target_lenght << 3, ret_val);
    return ret_val;
  }
...
```

Extract number of bytes to write into the buffer

Overflow the buffer

Heap Management Overview

- Every heap allocation is prepended with a 0x20-byte header with the metadata
  - Allocation driver ID: partitioned memory driver, system dynamic memory driver, etc
  - Size of allocated chunk
  - Allocation-driver-specific metadata



```
5079f380: 04 00 00 00 05 00 00 00 8F 2B 29 41 34 00 00 00
5079f390: C0 76 61 44 40 00 00 00 B0 4A 03 00 AA AA AA AA
5079f3a0: 00 01 3C 01 AA AA AA AA AA AA AA AA AA AA AA AA
5079f3b0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
```

Allocation heap header

Allocated heap buffer

Partitioned Memory Driver:

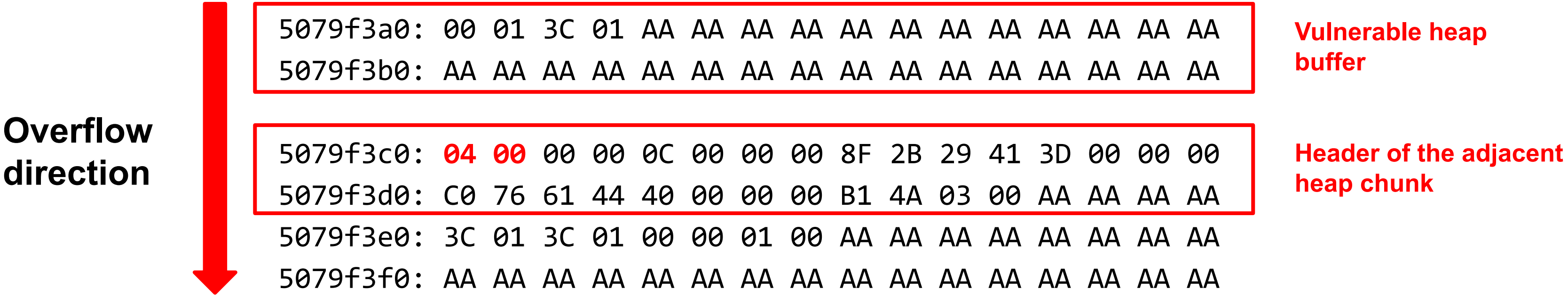
- manages arrays of fixed-size memory blocks
- tracks state of the memory blocks using a separate bitmap
- not very convenient for exploitation

System Dynamic Memory Driver:

- uses a double-linked list to manage allocated/free chunks
- **heap header contains the double-linked list and free function pointer!!!**

Getting Arbitrary Write Primitive

- Leverage the linear OOB write in the heap to obtain write-what-where primitive:
  - CVE-2022-20170 enables us to overwrite heap header of the next adjacent chunk with the fully controlled data
- The overwritten adjacent heap chunk is:
  - Conveniently allocated by ASN.1 parsing code **before** the buffer overflow happens
  - Reliably freed **after** the overflow
- Use the “classic” heap unlink technique to overwrite `free` function pointer



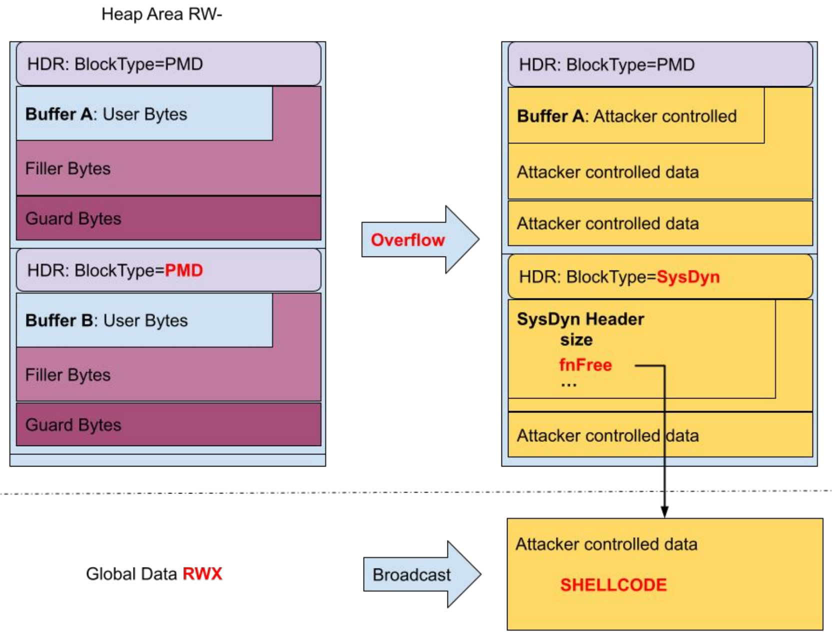
```
5079f380: 04 00 00 00 05 00 00 00 8F 2B 29 41 34 00 00 00
5079f390: C0 76 61 44 40 00 00 00 B0 4A 03 00 AA AA AA AA
5079f3a0: 00 01 3C 01 AA AA AA AA AA AA AA AA AA AA AA AA
5079f3b0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA

5079f3c0: 04 00 00 00 0C 00 00 00 8F 2B 29 41 3D 00 00 00
5079f3d0: C0 76 61 44 40 00 00 00 B1 4A 03 00 AA AA AA AA
5079f3e0: 3C 01 3C 01 00 00 01 00 AA AA AA AA AA AA AA AA
5079f3f0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
```

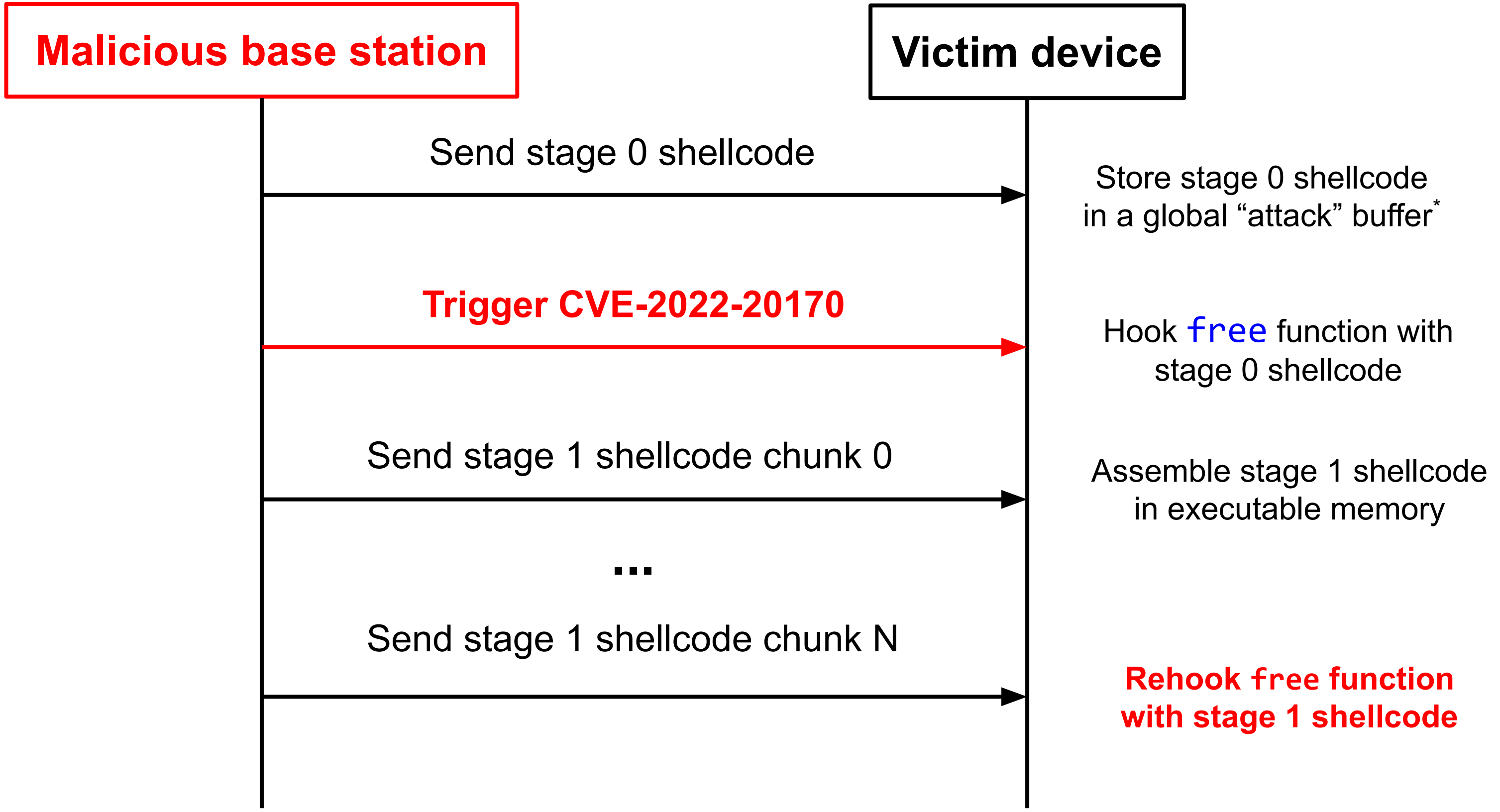
Getting RCE on Modem

CVE-2022-20170 + CVE-2022-20405 Overview

- Out-of-bounds write occurs in the **ASN decoder within the 2G stack** (CVE-2022-20170). This allows us to write a limited number of controlled bytes in the heap and corrupt adjacent heap objects.
- Corrupted adjacent heap objects give us arbitrary pointer write primitive when those objects are freed.
- **Misconfiguration in MMU** (CVE-2022-20405) allows us to stage executable shellcode in the heap.
- Overwrite the function pointer pointing to the free function of the heap allocator to point to our shell code
- When a heap object is freed, it will execute our shellcode.



Shellcode Delivery



Malicious base station

Victim device

Send stage 0 shellcode

Store stage 0 shellcode in a global “attack” buffer*

Trigger CVE-2022-20170

Hook `free` function with stage 0 shellcode

Send stage 1 shellcode chunk 0

Assemble stage 1 shellcode in executable memory

...

Send stage 1 shellcode chunk N

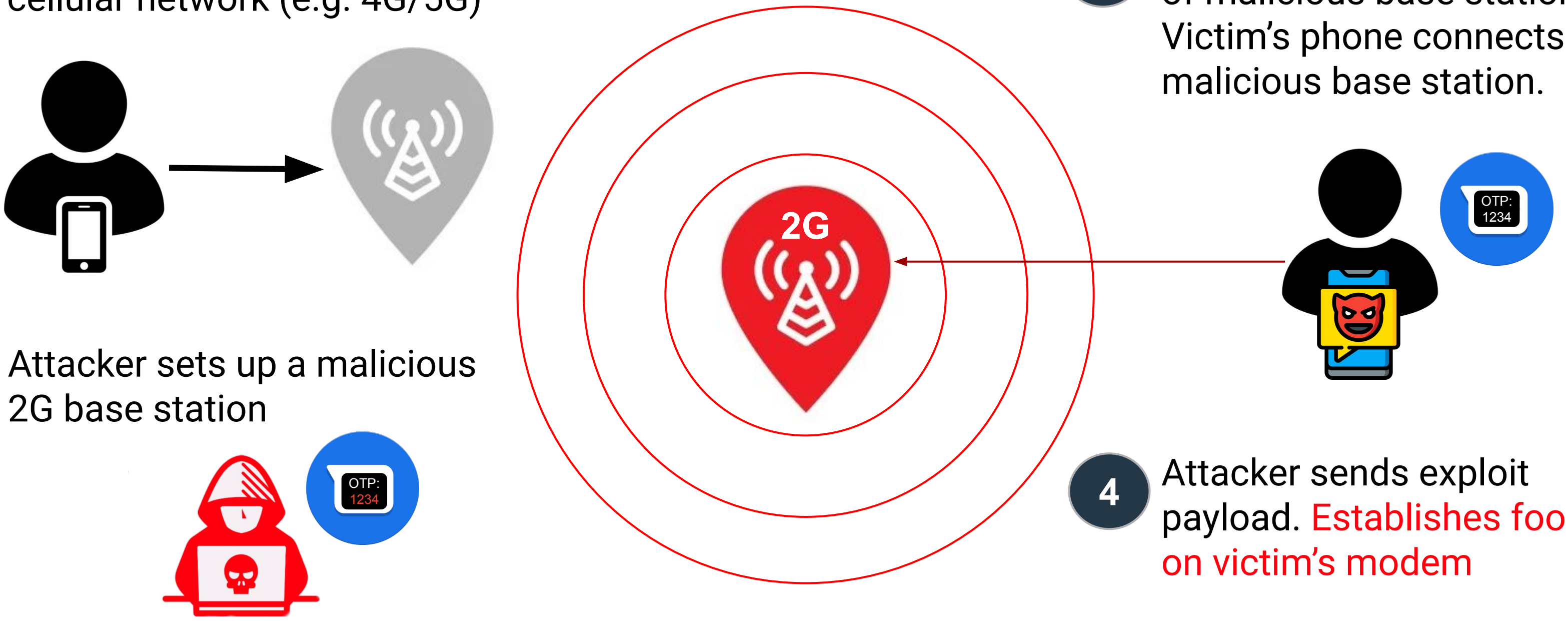
Rehook `free` function with stage 1 shellcode

** Global array of ~80 bytes at a known address used for storing stage 0 and chunks of stage 1 shellcode*

ANDROID RED TEAM

Modem RCE Proof of Concept

Attack Chain



1. User connects their phone to a cellular network (e.g. 4G/5G)

2. Attacker sets up a malicious 2G base station

3. User (victim) comes in proximity of malicious base station. Victim's phone connects to the malicious base station.

4. Attacker sends exploit payload. Establishes foothold on victim's modem

5. Attacker can capture and forward SMS messages (+more)

```
[illegible] $ adb devices
List of devices attached
[illegible] device
[illegible] device

[illegible] $ adb -s [illegible] shell
[illegible]
```

Attacker can now target victim's apps supporting SMS MFA

Exploitation Details

Prerequisites:

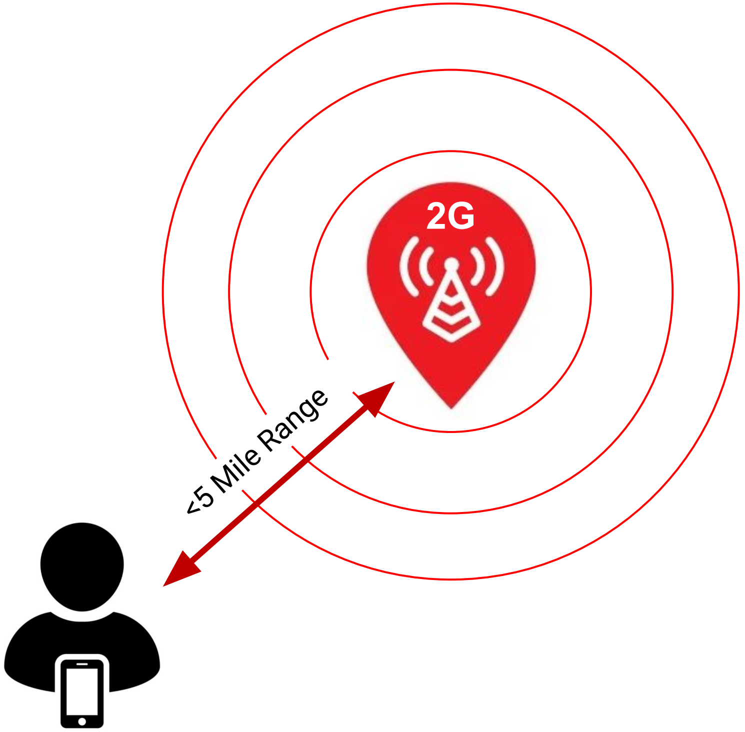
- 2G stack is enabled (default on Pixel 6)
- “Nearby range” to deploy the attack (<5 miles)

Impact:

- Total modem firmware compromise
- Possible Android OS compromise with radio driver/HAL side issues

Issues utilized for this exploit:

- An attacker controlled heap OOB write in GSM code (CVE-2022-20170)
- A mis-configuration of MMU allowing writable and executable memory (CVE-2022-20405)
- Lack of standard security mitigations making the exploiting easier



Proof of Concept Setup

Required hardware:

- SDR
- Cables and USB hubs
- Faraday cage (not needed for real attack)

Required software:

- OpenBTS (free, open source)

Total cost: <$2,800

Exploitation Challenges

- Not that easy to pack SDR, the attacker and victim devices into the Faraday cage to avoid interference
    - Subject to the value of the radio wavelength
- Reliability of the exploitation and time between iterations
    - Multiple complex systems involved into the exploitation: SDR + OpenBTS & modem
- Debugging shellcode on the production modem image
    - Collect ramdump when modem crash and then check the memory status
    - Patched an AT command handler in modem to confirm success of the exploitation locally on the victim device
- 80 bytes of thumb2 instructions is very tight to implement stage 0 shellcode
    - Effective shellcode area is less than 80 bytes due to specifics on heap “unlink” primitive

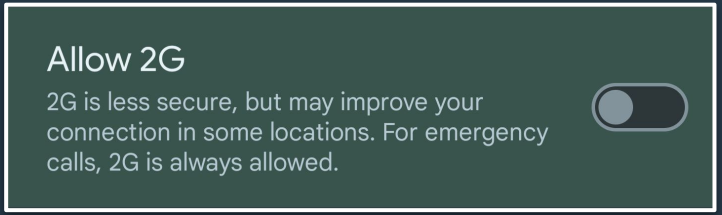
Remediation & What Comes Next

What You Can Do

Google is committed to making the Pixel modem as secure as possible. Here’s what you can do:

- 2G security is obsolete. The 2G standards didn’t take in account rogue cell towers as an attack vector (lack of mutual auth)
- Weak encryption combined with no authentication between device and tower means impersonation is easy over 2G.
- 2G is optional on Pixel devices. Disable the “Allow 2G” toggle on your device. This feature is supported in all Android (12+) devices with Radio HAL >1.6
- 2G disablement isn’t enforced as it’s required in certain locations

The best mitigation is to disable 2G on your device



Allow 2G

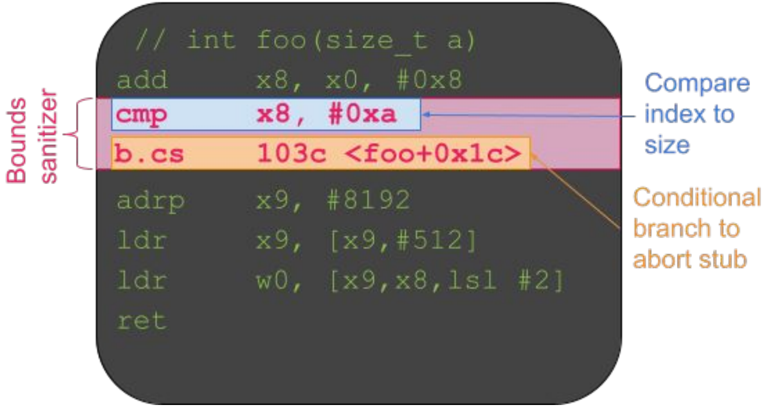
2G is less secure, but may improve your connection in some locations. For emergency calls, 2G is always allowed.

Bare Metal Mitigations

Android Security prioritizes hardening bare metal firmware

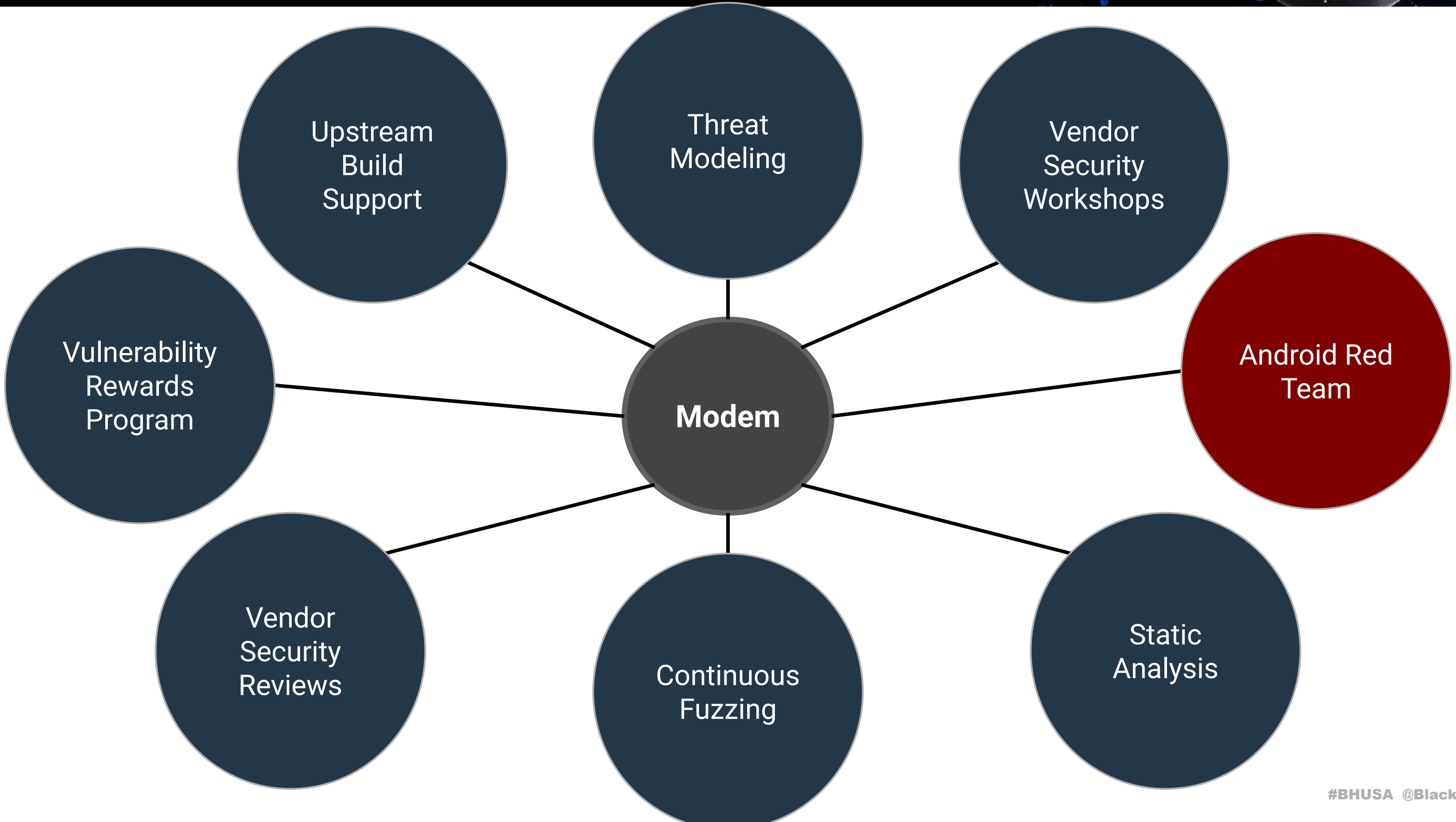
- System hardening and exploit mitigations
- Exploring and enabling compiler-based sanitizers (BoundSan, IntSan) and other exploit mitigations (CFI, kCFI, Shadow Call Stack, Stack Canaries) in firmware.
- Enabling further memory safety features (Auto-initialize Memory) in firmware.
- Exploring the application of Rust in bare metal firmware.

```
int bar[10];
int foo(size_t a) {
  return bar[a+8];
}
```



```
  // int foo(size_t a)
add        x8, x0, #0x8
cmp        x8, #0xa
b.cs       103c <foo+0x1c>
adrp       x9, #8192
ldr        x9, [x9,#512]
ldr        w0, [x9,x8,lsl #2]
ret
```

Cross-Functional Coverage



Conclusion

Concluding Thoughts

Red Team to Secure Pixel

~100 security issues were identified and **fixed in Pixel 6 before its release**

Exploit development helps articulate impact

Fuzzing is the Way

We heavily invested in fuzzing, developing 8 fuzzers identifying >60% of bugs logged during the engagement. These fuzzers run continuously and find issues today.

2G security is outdated

Google has protections in place to limit the outdated security and lack of mutual authentication of 2G. **Turning off 2G protects you from most attacks.**

Modem mitigations

We applied various mitigations to eradicate entire classes of vulnerabilities, with more hardening measures to come.

Our Work is Never done

Many Google teams came together on these security investments prioritizing security and remediation

We're never done! The team continues testing new features and releases

Acknowledgements

- Android Red Team
- Connectivity Security Team
- Pixel Engineering & Security Team
- Android Security
- Project Zero
- External Partners

ANDROID RED TEAM

Thanks!