A walk with Shannon
Walkthrough of a pwn2own baseband exploit.

@amatcama
Introduction

Amat Cama

• Independant Security Researcher.
• CTF player @ Shellphish.
• Exploitation and Reverse Engineering.
• Currently interested in Hypervisors and Baseband security reseach.

Previously

• Research Assistant – UCSB Seclab.
• Product Security Engineer – Qualcomm Inc.
• Senior Security Researcher – Beijing Chaitin Tech Co., Ltd.
Agenda

• Prior Work.
• Motivation.
• Cellular Networks ? Baseband ?
• The Shannon Baseband.
• Hunting for Bugs.
• Demo.
• Conclusions.
Prior Work
Prior Work

- “Breaking Band - reverse engineering and exploiting the shannon baseband” - Nico Golde and Daniel Komaromy.
- Very useful talk if you want to do research on the shannon baseband. Lots of scripts and information that will definitely be of help.
Motivation
Motivation

- Because it is fun.
- Unexplored area of research; great opportunity to learn.
- Many bugs.
- Big impact.
- Pwning a phone just by having it connect to a cellular network sounds pretty cool.
Cellular Networks ?
Baseband ?
What is a Cellular Network?

- Mobile communication network.
- “Cells” are land areas covered by a base transciever station (BTS).
- To cover a large area, the cells are used in junction: A Cellular Network.
- Technically could be any kind of network, today mostly Mobile Phone Network.
Cellular Networks ? Baseband ?

The technologies and standards (I)

- A number of technologies and standards developed.
- Different generations with improving speeds and capacity.
- Competing technologies for different generations.
The technologies and standards (II)
• Mainly two branches: **GSM** branch and **CDMA** branch

```
<table>
<thead>
<tr>
<th>Generation</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>2G</td>
<td>GSM, cdmaOne</td>
</tr>
<tr>
<td>2.5G</td>
<td>GPRS, IS-95 A/B</td>
</tr>
<tr>
<td>2.75G</td>
<td>EDGE, IS-95 A/B</td>
</tr>
<tr>
<td>3G</td>
<td>UMTS, CDMA2000</td>
</tr>
<tr>
<td>4G</td>
<td>LTE</td>
</tr>
</tbody>
</table>
```
The technologies and standards (III)

- **3GPP** is a collaboration agreement with a number of telecommunication standard bodies.
- Provides maintenance and development of the GSM *Technical Specifications (TS)*
  - GSM
  - GPRS / EDGE
  - UMTS
  - LTE
- Is Comprised of bodies such as the *European Telecommunications Standards Institute (ETSI)*.
- The technical standards provide detailed information on the structure of messages exchanged.
The Protocol Stack

<table>
<thead>
<tr>
<th>Layer</th>
<th>GSM</th>
<th>GPRS/EDGE</th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network Layer</strong></td>
<td><strong>CM - Connection Management</strong></td>
<td><strong>SM - Session Management</strong></td>
<td><strong>IP - Internet Protocol</strong></td>
</tr>
<tr>
<td></td>
<td><strong>MM - Mobility Management</strong></td>
<td><strong>GMM - GPRS Mobility Management</strong></td>
<td><strong>NAS - Non-Access Spectrum</strong></td>
</tr>
<tr>
<td><strong>Data Link Layer</strong></td>
<td><strong>LAPDm</strong></td>
<td><strong>SNDCP - Subnetwork Dependent Convergence Protocol</strong></td>
<td><strong>PDCP - Packet Data Convergence Protocol</strong></td>
</tr>
<tr>
<td></td>
<td><strong>RLC/LLC - Radio/Logical Link Control</strong></td>
<td><strong>MAC - Media Access Control</strong></td>
<td><strong>RLC/LLC - Radio/Logical Link Control</strong></td>
</tr>
<tr>
<td></td>
<td><strong>MAC - Media Access Control</strong></td>
<td></td>
<td><strong>MAC - Media Access Control</strong></td>
</tr>
</tbody>
</table>
Cellular Networks ? Baseband ?

The Baseband (I)

- Component of the phone in charge of handling communication with the mobile network.
- Deals with low level radio signal processing.
- Supports a number of standards (GSM, 3G, 4G, 5G, cdmaOne, CDMA2000, ...).
- Basically the main “interface” to the mobile network.
The Baseband (II)

- A number of different implementations.
- Qualcomm owns most of the market.
- Qualcomm: Galaxy, iPhone, OnePlus, Pixel, Xperia, HTC, LG, ASUS, Motorola, ...
- Huawei: Mate 10, P20, Honor 9, ...
- Samsung: Galaxy S6, S7, S8, S9, ...
- Intel: iphone X, iphone 8, ...
The Baseband (III)

- The most common architecture today: baseband firmware runs on a dedicated chip; the *cellular processor* (CP).
- This chip is tasked with all of the radio processing.
- The code is generally written in low level languages such as C/C++.
- A communication interface between CP and AP (Application Processor) such as shared memory, serial or interrupts.
The Baseband (IV)

- Getting code execution on the CP doesn’t necessarily result in owning the whole device.
- A number of attacks can be performed:
  - Redirect/Intercept phone calls.
  - Redirect/Intercept SMS.
  - Modify Internet traffic.
  - ...
- A step further; attack the AP through the IPC mechanisms and gain full control of the device.
The Shannon Baseband
The Shannon Baseband

About Shannon

- Samsung’s Baseband implementation.
- Typically ships with phones featuring the Exynos SoC.
- e.g: most non-US Galaxy phones.
- A RTOS running on an ARM Cortex R7.
The Shannon Baseband

Obtaining the code (I)

- The modem firmware can be obtained from the phone’s firmware images.
- However it is encrypted and doesn’t seem to be an easy way to decrypt it.
- Luckily it is possible to make the phone generate modem RAM dumps.
- Dialing the code *#9900# brings up the SYSDUMP menu.
Obtaining the code (II)

SYSDUMP
- RUN DUMPSTATE/LOGCAT/MODEM LOG
- DELETE DUMPSTATE/LOGCAT
- RUN DUMPSTATE/LOGCAT
- COPY KERNEL LOG TO SD CARD
- RUN CP BASED LOG
- COPY TO SDCARD(INCLUDE CP RAMDUMP)
- COPY TO EXTERNAL SDCARD(INCLUDE CP RAMDUMP)
- DEBUG LEVEL ENABLED/HIGH
- CP DEBUGGING POPUP UI : DISABLED
- SILENT LOG : OFF
- ENABLE SILENT LOGGING FROM BOOT(ONLY QCOM)
- ENABLE ONLY AP SILENT LOGGING FROM BOOT
- TRANSLATION ASSISTANT : OFF

CP RAM LOGGING: ON

LOW BATTERY DUMP : OFF
TCP DUMP START
IMS LOGGER
Obtaining the code (III)

- Tap on the `DEBUG LEVEL ENABLED/` option and set it to `High`. The phone will reboot.
- Reopen the SYSDUMP menu, scroll down and tap on the `CP RAM LOGGING` option and set it to `On`. The phone will reboot.
- Reopen the SYSDUMP menu and scroll all the way down, tap the `RUN FORCED CP CRASH DUMP` option. The phone will reboot and go into the ram upload mode. Hold the power and volume down button for 10 seconds to turn the phone off and then power it back on.
- Reopen the SYSMDUMP menu and tap the `COPY TO SDCARD(INCLUDE CP RAMDUMP)` option.
- Now in the folder `/sdcard/log` of the device, we have the log files including the ram dump. Largest file in the folder and has a name of the following format `cpcrash_dump_YYYYMMDD_HHSS.log`
Obtaining the code (IV)

MODEM CRASH!!

Doing Modem RAM Dump!
Do not touch anything for 10 minutes.
null
Loading Code in IDA

- The CP Boot Daemon (/sbin/cbd) handles powering on the modem and processing RAM dumps amongst other things.
- Boot code can be found at the start of the encrypted modem image in the firmware packages.
- By reversing the cbd and boot, we can translate the file offsets of the RAM dump to virtual addresses:
  
  0x40000000
  0x8000000
  0x4000000
  0x20000
  0x4800000
  0x4000
  0x3E00
  0x200
Identifying Tasks

- We need to identify the different tasks run by the RTOS.
- Start reversing from RESET Exception Vector Handler…
- Look at the start of the different memory regions and you recognize the Exception Vector Table in one of them.
- A linked list contains all the different tasks’ entry points, corresponding stack frames and task names (very useful).
- Traverse the list and identify all the tasks.
The Shannon Baseband

The Tasks (I)

- We end up with a list of tasks with different names, some of them self-explanatory, some of them misleading, some of them hard to understand.
- **MM** *(Mobility Management ?)*
- LLC
- SMS_SAP
- GRR
- SS
- SAEL3
- SNDCP
- **CC** *(Call Control ?)*
- **SM** *(Session Management ?)*
- LLC
- ...
Cellular Networks ? Baseband ?

- The Tasks (II)

<table>
<thead>
<tr>
<th>Network Layer</th>
<th>GSM</th>
<th>GPRS/EDGE</th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM - Connection Management</td>
<td>SM - Session Management</td>
<td>IP - Internet Protocol</td>
<td>GMM - GPRS Mobility Management</td>
</tr>
<tr>
<td>MM - Mobility Management</td>
<td>GMM - GPRS Mobility Management</td>
<td>NAS - Non-Access Spectrum</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Link Layer</th>
<th>LAPDm</th>
<th>RLC/LLC - Radio/Logical Link Control</th>
<th>SNDCP - Subnetwork Dependent Convergence Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC - Media Access Control</td>
<td>MAC - Media Access Control</td>
<td>PDCP - Packet Data Convergence Protocol</td>
<td></td>
</tr>
</tbody>
</table>

| Physical Layer | | | |
|----------------| | | |
The Shannon Baseband

The Tasks (III)

- Different tasks are used for different components and layers of the protocol stacks.
- Tasks communicate with each other using a *mailbox* system.
- Tasks are pretty much *while* loops waiting to process messages (from other tasks).
The Shannon Baseband

The Tasks (IV)

- Pick a task and start reversing.
- The Code is pretty generous in that it contains a lot of strings.
Hunting for Bugs
Setting up an environment (I)

- The goal is to be able to send arbitrary data to the baseband.
- Need to operate our own cellular network.
- Can be achieved with a *Software Defined Radio* (SDR).
- The Mobile Network Stack / Standard is implemented in software that runs on our computers.
- The SDR (device) is a general purpose transceiver that supports different frequencies.
Hunting for Bugs

Setting up an environment (II)

- A number of different options for the SDRs.
  - *BladeRF x40*: $420.00
  - *BladeRF x115*: $650.00
  - *USRP B200*: $675.00
  - *LimeSDR*: $300.00
  - *UmTRX*: $950.00 - $1300.00
Setting up an environment (III)

- A number of different options for software implementation of the standards.

- **YateBTS:**
  - Clean code, easy to modify.
  - Good support for *bladeRF*.
  - GSM and GPRS.
  - Easy to compile and run.

- **OpenBTS** (*OpenBTS-UMTS)*:
  - Clean code, easy to modify.
  - Good support for *USRP* and *UmTRX*.
  - GSM, GPRS, 3G.
  - Easy to compile and run.
Setting up an environment (IV)

- **OpenBSC** (OsmoNITB, OsmBTS, ...):
  - Good support for *USRP*, *LimeSDR* and *UmTRX*.
  - Compiling wasn’t easy.
  - Clean code, easy to modify.
  - GSM + GPRS.

- **OpenAirInterface**:
  - Hard to compile and run.
  - Good support for *USRP*.
  - 4G.

- **OpenLTE**:
  - Hard to compile and run.
  - 4G.
  - Good support for *USRP*.
  - Clean code, easy to modify.
Hunting for Bugs

Setting up an environment (V)

- Provisionned or programmable SIM Cards because 3G and 4G do not support open authentication.
- Faraday Cage / RF Enclosure because in most countries, operating a cell network without permission is Illegal!
Hunting for Bugs

Debugging The Phone

- Everytime the modem crashes we get a RAM dump.
- Luckily the dump contains the state of the registers at the time of the crash, therefore we have pretty decent post-mortem debugging capabilities.
- Write a script to process the dumps and do useful stuff (registers, peeking at memory).
Digging into the code (I)

- Back to picking a task to have a closer look at.
- An interesting approach is the following:
  - Layer 3 Messages are comprised of *Information Elements (IEs).*
  - IEs are *V, LV, TLV.*
  - What are the different messages that can be sent to different components?
  - Cross Reference the Technical Standards to know the different message types sent to different components.
  - Read the description of the different messages and the content of the Information Elements. Are there (T)LVs? Then reverse the corresponding task and try to find the code processing that particular IE.
  - A number of trivial bugs can be found this way...
Digging into the code (II)

- Let’s clarify with an example.
- The CC task most likely stands for Call Control.
- Call control is a part of Connection Management in the GSM protocol stack.
- What are the different CC messages?
# Digging into the code (III)

<table>
<thead>
<tr>
<th>Call establishment messages:</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALERTING</td>
<td>9.3.1</td>
</tr>
<tr>
<td>CALL CONFIRMED (NOTE)</td>
<td>9.3.2</td>
</tr>
<tr>
<td>CALL PROCEEDING</td>
<td>9.3.3</td>
</tr>
<tr>
<td>CONNECT</td>
<td>9.3.5</td>
</tr>
<tr>
<td>CONNECT ACKNOWLEDGE</td>
<td>9.3.6</td>
</tr>
<tr>
<td>EMERGENCY SETUP (NOTE)</td>
<td>9.3.8</td>
</tr>
<tr>
<td>PROGRESS</td>
<td>9.3.17</td>
</tr>
<tr>
<td>CC-ESTABLISHMENT</td>
<td>9.3.17a</td>
</tr>
<tr>
<td>CC-ESTABLISHMENT CONFIRMED</td>
<td>9.3.17b</td>
</tr>
<tr>
<td>START CC</td>
<td>9.3.23a</td>
</tr>
<tr>
<td>SETUP</td>
<td>9.3.23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call information phase messages:</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODIFY (NOTE)</td>
<td>9.3.13</td>
</tr>
<tr>
<td>MODIFY COMPLETE (NOTE)</td>
<td>9.3.14</td>
</tr>
<tr>
<td>MODIFY REJECT (NOTE)</td>
<td>9.3.15</td>
</tr>
<tr>
<td>USER INFORMATION</td>
<td>9.3.31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call clearing messages:</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCONNECT</td>
<td>9.3.7</td>
</tr>
<tr>
<td>RELEASE</td>
<td>9.3.18</td>
</tr>
<tr>
<td>RELEASE COMPLETE</td>
<td>9.3.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Messages for supplementary service control</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACILITY</td>
<td>9.3.9</td>
</tr>
<tr>
<td>HOLD (NOTE)</td>
<td>9.3.10</td>
</tr>
<tr>
<td>HOLD ACKNOWLEDGE (NOTE)</td>
<td>9.3.11</td>
</tr>
<tr>
<td>HOLD REJECT (NOTE)</td>
<td>9.3.12</td>
</tr>
<tr>
<td>RETRIEVE (NOTE)</td>
<td>9.3.20</td>
</tr>
<tr>
<td>RETRIEVE ACKNOWLEDGE (NOTE)</td>
<td>9.3.21</td>
</tr>
<tr>
<td>RETRIEVE REJECT (NOTE)</td>
<td>9.3.22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miscellaneous messages</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONGESTION CONTROL</td>
<td>9.3.4</td>
</tr>
<tr>
<td>NOTIFY</td>
<td>9.3.16</td>
</tr>
<tr>
<td>START DTMF (NOTE)</td>
<td>9.3.24</td>
</tr>
<tr>
<td>START DTMF ACKNOWLEDGE (NOTE)</td>
<td>9.3.25</td>
</tr>
<tr>
<td>START DTMF REJECT (NOTE)</td>
<td>9.3.26</td>
</tr>
<tr>
<td>STATUS</td>
<td>9.3.27</td>
</tr>
<tr>
<td>STATUS ENQUIRY</td>
<td>9.3.28</td>
</tr>
<tr>
<td>STOP DTMF (NOTE)</td>
<td>9.3.29</td>
</tr>
<tr>
<td>STOP DTMF ACKNOWLEDGE (NOTE)</td>
<td>9.3.30</td>
</tr>
</tbody>
</table>
Hunting for Bugs

Digging into the code (IV)

<table>
<thead>
<tr>
<th>IEI</th>
<th>Information element</th>
<th>Type/Reference</th>
<th>Presence</th>
<th>Format</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Call control protocol discriminator</td>
<td>Protocol discriminator</td>
<td>M</td>
<td>V</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transaction identifier</td>
<td>Transaction identifier</td>
<td>M</td>
<td>V</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.3.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Progress message type</td>
<td>Message type</td>
<td>M</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Progress indicator</td>
<td>Progress indicator</td>
<td>M</td>
<td>LV</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.5.4.21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7E</td>
<td>User-user</td>
<td>User-user</td>
<td>O</td>
<td>TLV</td>
<td>3-131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.5.4.25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hunting for Bugs

Digging into the code (V)

- Using this approach it is possible to find a number of trivial vulnerabilities.
- A previous bad experience competing at P2O taught me that trivial bugs are bad.
- Dig a bit deeper in order to find something less trivial and reduce the chance for collisions.
Hunting for Bugs

The Mobile Pwn2Own Bug (I)

- Decided to look at GPRS since it seems complicated?
- Start by reading the standards and looking at the GPRS Session Management Messages.

GPRS/EDGE

SM - Session Management

GMM - GPRS Mobility Management

GRR - GPRS Radio Resource Management
Hunting for Bugs

The Mobile Pwn2Own Bug (II)

- The ACTIVATE PDP CONTEXT ACCEPT message looks good.

<table>
<thead>
<tr>
<th>Table 9.5.2/3GPP TS 24.008: ACTIVATE PDP CONTEXT ACCEPT message content</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEI</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>2B</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>39</td>
</tr>
<tr>
<td>B-</td>
</tr>
<tr>
<td>C-</td>
</tr>
<tr>
<td>33</td>
</tr>
</tbody>
</table>
The Mobile Pwn2Own Bug (III)

- By reversing the SM task, we find the handlers for the different messages.
- One of these messages is the ACTIVATE PDP CONTEXT ACCEPT message.
- One part of it that seems to be interesting is the Protocol Configuration Options, the function processing that IE seems complicated.
Hunting for Bugs

The Mobile Pwn2Own Bug (IV)

```c
#include <stdlib.h>

int main() {
    unsigned int buf[10];
    unsigned int buf2[10];
    while (idx < bufsize) {
        unsigned int idx;
        v10 = buf[idx];
        v11 = buf[v10];
        idx = v10 + 1;
        proto = (unsigned int)[buf+1] + (*buf+2);
        v11 = v13;
        if (v11) {
            if (proto == 0x8021) {
                subprot = buf[idx];
                idx = idx + 2;
                if (subprot == 2 || subprot == 3 || subprot == 4 || subprot == 1) {
                    ntr = buf[idx];
                    idx = idx + 2;
                    v18 = (unsigned int)[nptr+1] + (*nptr+2);
                    if (v18 == 4) {
                        v19 = v18 - 4;
                        while (2) {
                            v20 = v19;
                            while (i) {
                                if (v20) {
                                    goto LABEL_217;
                                }
                                v20 = buf[idx];
                                v21 = idx + 1;
                                if (v20 == 3) {
                                    break;
                                }
                                v22 = buf[v21];
                                idx = v21 + 1;
                                v24 = v23;
                                if (len > 0x10) {
                                    if (byte_41695FA4) {
                                        v68 = 0x18186344;
                                        v61 = (unsigned int)byte_41695FA4 << 18 + 0x46521;
                                        v25 = len;
                                    } else {
                                        v23 = len;
                                        v68 = 0x18186372;
                                        v61 = 0x46521;
                                    }
                                } BugRelatedFcn(0x8068, v25, 0x46521);
                                for (i = 0; i < (signed int)(v24 - 2); i = (i + 1) & 0xFF ) {
                                    c = buf[idx+1];
                                    v22[i] = c;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return 0;
}
```

Hunting for Bugs

The Mobile Pwn2Own Bug (V)

- Processes Protocol Configuration Options which are sent by the Network in a `ACTIVATE PDP CONTEXT ACCEPT`.
- PDP stands for Packet Data Protocol
- The purpose of the protocol configuration options information element is to:
  - transfer external network protocol options associated with a PDP context activation, and
  - transfer additional (protocol) data (e.g. configuration parameters, error codes or messages/events) associated with an external protocol or an application.
Hunting for Bugs

The Mobile Pwn2Own Bug (VI)

- One of the supported protocols is **IPCP** (Internet Protocol Control Protocol).

**IPCP header:**

| Offsets | Octet | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---------|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Octet   |       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Bit     | Code  | Identifier | Length |

**Code.**

8 bits. Specifies the function to be performed.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Vendor Specific.</td>
<td>RFC 2153</td>
</tr>
<tr>
<td>1</td>
<td>Configure-Request.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Configure-Ack.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Configure-Nak.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Configure-Reject.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Terminate-Request.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Terminate-Ack.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Code-Reject.</td>
<td></td>
</tr>
</tbody>
</table>
Hunting for Bugs

The Mobile Pwn2Own Bug (VII)

```c
int sm_ProcessProtConfigOpts(unsigned __int8 *buf, unsigned int bufsize, unsigned __int8 *a3) {
    unsigned __int8 tbuf[16];
    int idx;
    int len;
    unsigned __int8 c;

    ...

    len = buf[idx++];
    ...
    if ( !len || len > 0x10 )
        // Do nothing about it
    }
    for ( i = 0; i < len - 2; i = (i + 1) & 0xFF )
        { 
        c = buf[idx++];
        tbuf[i] = c;
        }

    ...
}
```
The Mobile Pwn2Own Bug (VIII)

- The plan looks like this:
Hunting for Bugs

The Mobile Pwn2Own Bug (IX)

• Not so easy...
• Problem is the phone will only process this message if it is in the correct state.
• This happens when the phone sends a `ACTIVATE PDP CONTEXT REQUEST` message.
• Which in turn happens if the phone is manually configured to include an APN in the connection settings.
• However this is a problem for the P2O...
Hunting for Bugs

The Mobile Pwn2Own Bug (X)

- Read more of the technical standards...
- We can force the MS to get in the correct state (i.e. perform PDP activation procedure) by sending a `REQUEST PDP CONTEXT ACTIVATION`.

9.5.7 Request PDP context activation

This message is sent by the network to the MS to initiate activation of a PDP context. See table 9.5.7/3GPP TS 24.008.

- Message type: REQUEST PDP CONTEXT ACTIVATION
- Significance: global
- Direction: network to MS
The Mobile Pwn2Own Bug (XI)

- The actual plan looks like this:
The Mobile Pwn2Own Bug (XII)

- In order to actually implement the attack we need to modify the source code of *YateBTS*.
- Add code to send the `REQUEST PDP CONTEXT ACTIVATION` message to the phone.
- Modify the `ACTIVATE PDP CONTEXT ACCEPT` messages to trigger the bug.
- As said earlier the code is pretty clean and actually reading it will help you better understand the GSM protocol stack.
- For this attack, the file to modify is: mbts/SGSNGGSN/Ggsn.cpp
Hunting for Bugs

The Mobile Pwn2Own Bug (XIII)

- ROP is needed for the first stage of your payload due to ARM cache-fu.
- Copy shellcode to some arbitrary RWX address and invalidate/flush the i-cache/d-cache.
- Jump to win.
- Payload can do any number of things, for P2O I chose to write to the Android filesystem by leveraging the RFS (Remote? File System), a mechanism which allows the baseband to store data such as NV Items to the android filesystem.
- Payload can even be a custom “debugger” that can be used to find other bugs and write more involved exploits (e.g. heap memory corruption).
Conclusions
Conclusions

• Baseband exploitation isn’t as hard as it is perceived to be.
• You don’t need to know much about cellular networks in order to exploit them.
• When will we see the first full remote compromise through baseband?
• Many targets out there, Huawei, Intel, Qualcomm...
?'s