BASEBAND PLAYGROUND

ekoparty Security Conference
7° edición

Luis Miras
luis@ringzero.net
@_luism

Thanks

This talk is based on unlocks and information provided by the following

- Musclenerd @MuscleNerd
- iPhone Dev Team @iphone_dev
- Geohot http://geohot.com
- iPhone Wiki http://theiphonewiki.com

What is talk about

- Exploring basebands via code injection
- Creating a development environment
- Reverse Engineering, static and dynamic
- Code patching techniques

What is not covered

- Remote exploitation. See:
  - Ralf-Philipp Weinmann “The Baseband Apocalypse”
    - http://www.youtube.com/watch?v=CPPQ8vA6cRc
  - Grugq “Base Jumping”
    - https://media.blackhat.com/bh-ad-10/Grugq/BlackHat-AD-2010-Gurgq-Base-Jumping-slides.pdf
- Non public unlocks

Contents

- Baseband Background
- Static Reverse Engineering
- Development Environment
- Dynamic Reverse Engineering
- Patching Code
- Conclusion


BASEBAND BACKGROUND

What is a baseband?

- Baseband is short for baseband processor.
- For this talk only talking about smartphones
- Can be a separate chip or a separate core

What is a baseband?

- Controls/Interfaces with hardware
  - Audio, voice and mp3 codecs
  - Video display
  - Camera
  - USB
  - GPS
  - WiFi
  - Bluetooth, etc

What is a baseband?

- Provides communication protocols
  - GSM
  - GPRS
  - Edge
  - UMTS

What is a baseband?

- Generally ARM core
- Basebands run small RTOS
  - iPhone, iPhone 3G, iPhone 3GS uses Nucleus
  - iPhone 4 uses ThreadX
  - Qualcomm chips use OKL4
- OsmocoBB
  - Open Source GSM implementation
  - Motorola C123 and friends (feature phone)
  - http://bb.osmocom.org

What is a baseband?

- Application/Baseband communication
  - Shared memory
  - AT commands over high speed serial lines

What is a baseband?

- Application/Baseband communication
  - Shared memory
  - AT commands over high speed serial lines

AT commands ?

What is a baseband?

- Application/Baseband communication
  - Shared memory
  - AT commands over high speed serial lines

What is a baseband?

- Application/Baseband communication
  - Shared memory
  - AT commands over high speed serial lines

```
at+cnum
+CNUM: "","15555551212",129

OK
at+cgsn
011770000000000

OK
```

What is an unlock?

- A carrier lock prevents phone's use on other networks.
- An unlock removes this restriction.
- Some phones require numeric password to unlock.
- Other unlocks patch bootloader/firmware, removing checks. ex. original iPhone

iPhone 3G unlock

- First unlock was released by iPhone Dev Team, yellowsn0w.
- It exploits an AT command parser vuln executing arbitrary code in the baseband
- Payload uses Nucleus API to perform unlock
- The unlock is not permanent.

Infineon X-Gold 608 / XMM 6080

- Baseband used in iPhone 3G(S) and iPad1 3G
- ARM 926 Core
- Nucleus RTOS
- Selected as test case because of unlock (exploit) availability

Baseband Unlocks

Unlock	Baseband(s)	Firmware		Vector
yellowsn0w [1]	02.28.00	2.2	3G	AT+STKPROF
ultrasn0w [1]	04.26.08 05.11.07 05.12.01 05.13.04 06.15.00 [3]	3.0, 3.0.1 3.1, 3.1.2 3.1.3 4.0 – 4.0.2 3.2 – 3.2.2	3G(S) 3G(S) 3G(S) 3G(S) iPad1	AT+XLOCK AT+XAPP AT+XAPP AT+XAPP AT+XAPP
purplesn0w [2]	04.26.08	3.0, 3.0.1	3G(S)	AT+XLOCK
blacksn0w [2]	05.11.07	3.1, 3.1.2	3G(S)	AT+XEMM (heap)

1. iPhone dev team
2. Geohot
3. iPad1 baseband. iPhone 3G(S) will lose GPS functionality

NOTE: Downgrades are generally not possible except for a specific early release 3G bootloader.

What is Nucleus?

- It is a RTOS developed by Mentor Graphics.
  - Downloadable trial available
- Closed source, but clients get source code.
- Runs on multiple CPUs
- 2.84 Billion devices by end of 2010
- C/C++ development environment using Code Sourcery tools


UNLOCK WALKTHROUGH

AT+STKPROF Exploit

- This is the vector yellowsn0w used.
- Source code was released
- Payload disassembly provided:
  http://theiphonewiki.com/wiki/index.php?title=Yellowsn0w
- I will highlight some interesting areas
- Exploit has 3 parts
  - loader
  - stage2
  - payload

AT+STKPROF

```
at+stkprof=1,"064a541c044b1878222803d0107001320133f8e720470000bf
9f15400170100546e5640200000005c130100266e5640ddddddddeeeeeeeeb8
90512000000000101010102020202061130100 0c000000";"\x10\x32\x0F\x27
\xBA\x43\x17\x1C\x0E\xA4\x0B\xA5\x01\x35\x21\x78\x78\x29\x0C\xD0
\xA8\x47\x0B\x01\x61\x78\xA8\x47\xC0\x46\xC0\x46\xC0\x46\xC0\x46
\xC9\x18\x11\x70\x02\x34\x01\x32\xEF\xE7\xC0\x46\xC0\x46\x01\x37
\x38\x47\x30\x30\x41\x29\x01\xDA09pG79pG024803A1013101601FBD0000
4C711140F0B51C4B80268BB036011880089 11A4C301CA0470025099098 20A047
071CC56080204000A0478022144952 00144B041C9847099B0193442303930A23
013405930C23221C06930F490095029604953 81C00230D4CA047021C002804D1
0B4908980B4B984703E00B490898094B98470BB0F0BD000044B33B40AC201420
641A0100A0583C20481A010040B53F20541A010000DD4620581A010064657674
65616D31000000004F4B21004552524F52202564000000030B5114D85B0114B
281C6946FF229847009B0D2B11D101990D4B0A681A6004334A681A608A680B4B
13600B4B53600B4B93600123CB602023009328 1C6946FF22074B9847DFE70000
5427234098591620BC792F4000FF000101040204030404046 8D53E20xx”
```

AT+STKPROF

```
at+stkprof=1,"064a541c044b1878222803d0107001320133f8e720470000bf <- loader
9f154000170100546e5640200000005c130100266e5640ddddddddeeeeeeeeb8
90512000000000101010102020202061130100 0c000000";"\x10\x32\x0F\x27
\xBA\x43\x17\x1C\x0E\xA4\x0B\xA5\x01\x35\x21\x78\x78\x29\x0C\xD0
\xA8\x47\x0B\x01\x61\x78\xA8\x47\xC0\x46\xC0\x46\xC0\x46\xC0\x46
\xC9\x18\x11\x70\x02\x34\x01\x32\xEF\xE7\xC0\x46\xC0\x46\x01\x37
\x38\x47\x30\x30\x41\x29\x01\xDA09pG79pG024803A1013101601FBD0000
4C711140F0B51C4B80268BB036011880089 11A4C301CA0470025099098 20A047
071CC56080204000A0478022144952 00144B041C9847099B0193442303930A23
013405930C23221C06930F490095029604953 81C00230D4CA047021C002804D1
0B4908980B4B984703E00B490898094B98470BB0F0BD000044B33B40AC201420
641A0100A0583C20481A010040B53F20541A010000DD4620581A010064657674
65616D31000000004F4B21004552524F52202564000000030B5114D85B0114B
281C6946FF229847009B0D2B11D101990D4B0A681A6004334A681A608A680B4B
13600B4B53600B4B93600123CB602023009328 1C6946FF22074B9847DFE70000
5427234098591620BC792F4000FF000101040204030404 0468D53E20xx”
```

AT+STKPROF

at+stkprof=1,"064a541c044b1878222803d0107001320133f8e720470000bf
9f15400170100546e5640200000005c130100266e5640ddddddddeeeeeeeeb8
90512000000000101010102020202061130100 0c000000";"\x10\x32\x0F\x27
\xBA\x43\x17\x1C\x0E\xA4\x0B\xA5\x01\x35\x21\x78\x78\x29\x0C\xD0 <- stage2
\xA8\x47\x0B\x01\x61\x78\xA8\x47\xC0\x46\xC0\x46\xC0\x46\xC0\x46
\xC9\x18\x11\x70\x02\x34\x01\x32\xEF\xE7\xC0\x46\xC0\x46\x01\x37
\x38\x47\x30\x30\x41\x29\x01\xDA09pG79pG024803A1013101601FBD0000
4C711140F0B51C4B80268BB036011880089 11A4C301CA0470025099098 20A047
071CC56080204000A0478022144952001 44B041C9847099B0193442303930A23
013405930C23221C06930F490095029604953 81C00230D4CA047021C002804D1
0B4908980B4B984703E00B490898094B98470BB0F0BD000044B33B40AC201420
641A0100A0583C20481A010040B53F20541A010000DD4620581A010064657674
65616D31000000004F4B21004552524F52202564000000030B5114D85B0114B
281C6946FF229847009B0D2B11D101990D4B0A681A6004334A681A608A680B4B
13600B4B53600B4B93600123CB602023009328 1C6946FF22074B9847DFE70000
5427234098591620BC792F4000FF000101040204030404046 8D53E20xx”

AT+STKPROF

```
at+stkprof=1,"064a541c044b1878222803d0107001320133f8e720470000bf
9f154000170100546e5640200000005c130100266e5640ddddddddeeeeeeeeb8
90512000000000101010102020202061130100 0c000000";"\x10\x32\x0F\x27
\xBA\x43\x17\x1C\x0E\xA4\x0B\xA5\x01\x35\x21\x78\x78\x29\x0C\xD0
\xA8\x47\x0B\x01\x61\x78\xA8\x47\xC0\x46\xC0\x46\xC0\x46\xC0\x46
\xC9\x18\x11\x70\x02\x34\x01\x32\xEF\xE7\xC0\x46\xC0\x46\x01\x37
\x38\x47\x30\x30\x41\x29\x01\xDA09pG79pG024803A1013101601FBD0000 <- payload
4C711140F0B51C4B80268BB03601188008911A4C301CA04700250990 9820A047
071CC56080204000A04780221449520 0144B041C9847099B0193442303930A23
013405930C23221C06930F490095029604953 81C00230D4CA047021C002804D1
0B4908980B4B984703E00B490898094B98470BB0F0BD000044B33B40AC201420
641A0100A0583C20481A010040B53F20541A010000DD4620581A010064657674
65616D31000000004F4B21004552524F52202564000000030B5114D85B0114B
281C6946FF229847009B0D2B11D101990D4B0A681A6004334A681A608A680B4B
13600B4B53600B4B93600123CB6020230093281C6946FF22074B9847DFE70000
5427234098591620BC792F4000FF000101040204030404 0468D53E20xx”
```

Loader

```
RAM:00011360    loader
RAM:00011360            LDR     R2, =0x11700     ; unused ram to place code
RAM:00011362            ADDS    R4, R2, #1       ; thumb switch
RAM:00011364            LDR     R3, =0x40159FBF  ; location of stage2+payload in mem
RAM:00011366    copy_loop
RAM:00011366            LDRB    R0, [R3]         ; copy till quotes
RAM:00011368            CMP     R0, #0x22
RAM:0001136A            BEQ     run
RAM:0001136C            STRB    R0, [R2]
RAM:0001136E            ADDS    R2, #1
RAM:00011370            ADDS    R3, #1
RAM:00011372            B       copy_loop
RAM:00011374    run
RAM:00011374            BX      R4               ; stage2
RAM:00011376                    DCW    0
RAM:00011378  dword_11378       DCD 0x40159FBF
RAM:0001137C  dword_1137C       DCD 0x11700
```

Original disasm on iPhone wiki

Loader

- Loader is a standard copy loop
- Dwords following loader are interesting:

```
RAM:00011380      DCD 0x40566E54
RAM:00011384      DCD 0x20
RAM:00011388      DCD 0x1135C     ; R4 used in bytecopy
RAM:0001138C      DCD 0x40566E26  ; R5 used in bytecopy, has loader code
RAM:00011390      DCD 0xDDDDDDDD  ; padding
RAM:00011394      DCD 0xEEEEEEEE  ; padding
RAM:00011398      DCD 0x205190B8  ; <- ys_bytecode_rop(R4 + 4, R5 + 0x12, 0x50)
RAM:0001139C      DCD 0           ; padding
RAM:000113A0      DCD 0x10101010  ; padding
RAM:000113A4      DCD 0x20202020  ; padding
RAM:000113A8      DCD 0x11361     ; return into loader code (0x11360+Thumb)
RAM:000113AC      DCD 0xC
```

Loader

- When entering the following code snippet,
- R4 = 0x1135C and R5 = 0x40566E26,

```
ROM:205190B8          ADD     R1, R5, #0x12    ; R1 = 0x40566E26 + 0x12 = 0x40566E38
ROM:205190BC          ADD     R0, R4, #4       ; R0 = 0x1135C + 4 = 0x11360
ROM:205190C0          BL      bytecpy
ROM:205190C4          LDMFD   SP!, {R4-R6,PC}
```

Performs bytecpy(0x11360, 0x40566E38, R2)

R2 happens to be 0x50

Loader

- Loader is a standard copy loop
- Dwords following loader are interesting:

```
RAM:00011380      DCD 0x40566E54
RAM:00011384      DCD 0x20
RAM:00011388      DCD 0x1135C     ; R4 used in bytecopy
RAM:0001138C      DCD 0x40566E26  ; R5 used in bytecopy, has loader code
RAM:00011390      DCD 0xDDDDDDDD  ; padding
RAM:00011394      DCD 0xEEEEEEEE  ; padding
RAM:00011398      DCD 0x205190B8  ; <- ys_bytecode_rop(R4 + 4, R5 + 0x12, 0x50)
RAM:0001139C      DCD 0           ; padding
RAM:000113A0      DCD 0x10101010  ; padding
RAM:000113A4      DCD 0x20202020  ; padding
RAM:000113A8      DCD 0x11361     ; return into loader code (0x11360+Thumb)
RAM:000113AC      DCD 0xC
```

Stage2

- Stage2 decodes, aka unhexlify(), the payload and jumps into it.
- Can be viewed on iPhone Wiki.
- Note: payload loading address is based on size of payload.
- This is fixed in ultrasn0w

Payload

Payload consists of three functions

- **handler_replace()** – changes soft reset handler ptr to new_handler

- **new_handler()** – creates a new task using NU_Create_Task()

- **task_loop()** – sends a message to the Security Mailbox (IPC)

Payload

- The **new_handler()** function is ideal for modification.
- It will be called during soft reset

Disassembly tips

- ARM must be 4 byte aligned, Thumb 2 byte aligned
- If disasm not aligning, make 3 copies with extra character in front:

```
\x06\x4a\x54\x1c\x04\x4b\x18…
A\x06\x4a\x54\x1c\x04\x4b\x18…
AA\x06\x4a\x54\x1c\x04\x4b\x18…
AAA\x06\x4a\x54\x1c\x04\x4b\x18…
```

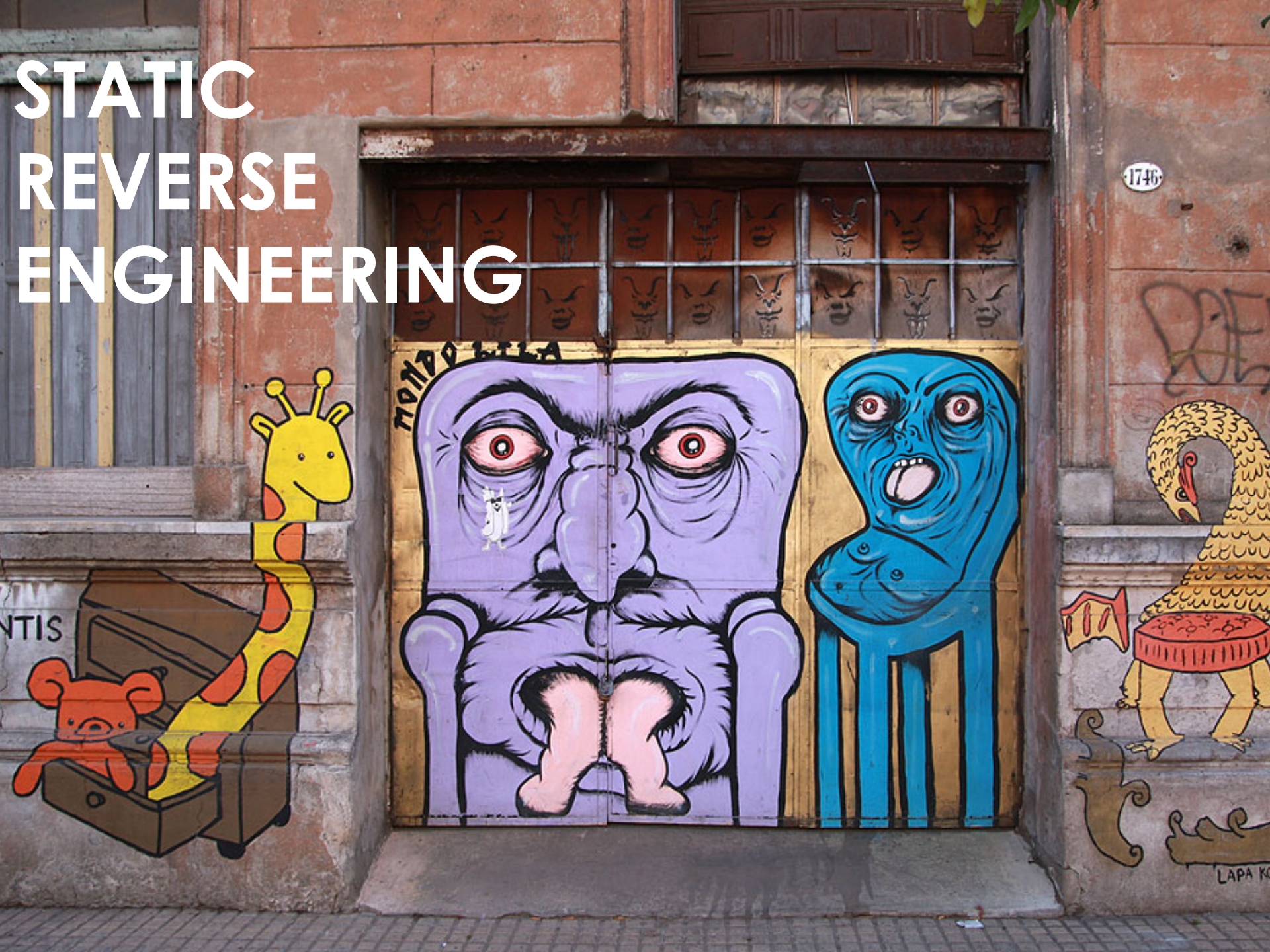
- See which makes best code
  - Try both ARM and Thumb mode (ALT-G)

Disassembly Tips

- Make segments for loader, stage2, etc
  - Load *Additional Binary File* with different offsets
  - Or make manual segments and copy bytes
- Use IDA and QEMU to run code snippets
  - Works great for testing custom injected code later on.

    http://www.hexblog.com/?p=111

STATIC REVERSE ENGINEERING



Choosing a Baseband Version

- We need an unlockable version
  - Generally can't downgrade
- This talk is using an old version 02.28.00
  - Started after yellowsn0w release, stopped working on this, and recently restarted
  - Test phone was never upgraded
  - Techniques applicable to other versions

Extracting Baseband Image

- iPhone firmware comes in ipsw (aka zip) files.
- Baseband firmware is in the “restore” dmg
- The restore image needs to be decrypted using img3decrypt or xpwntool
- Keys are listed on iPhone wiki
- Path is usr/local/standalone/firmware/
- File extension is .fls, ex: ICE2_04.26.08.fls

IDA Pro

- iPhone wiki has loading instructions
  http://theiphonewiki.com/wiki/index.php?title=IDA_Pro_Setup
- Unfortunately IDA Pro doesn't find very many functions.
- The baseband has both ARM and Thumb code.
- ALT-G changes the CPU mode. 0 = ARM, 1 = Thumb

IDA Pro

- A script is needed to find common prologues (ARM and Thumb)
- Script should
  - Change mode if needed
  - Create code
  - Create a function

IDA Pro

- Original disassembly has 3440 functions

- After running the script, disassembly has 33284 functions

IDA Pro

- The script isn't perfect.
- Lots of fixups still needed
- itsme idc scripts are useful for data
  - Table("od"); // creates offset, data
- Many strings aren't defined
  - itsme script assumes 4 byte aligned strings
- Remember Thumb function pointer are address + 1

IDA Pro

- Start using known function address from exploit disasm, ex NU_Create_Task
- Locate nucleus library files, preferably ARM
  - I believe there is now a downloadable demo
- Bindiff can be used
- Look for specific constants used as immediates in IDA
  - TASK is 0x5441534B
  - HISR is 0x48495352

Some Interesting Strings

```
ROM:20623462  fbi_bits
ROM:204B33D4  I don't like you.  Go 'way.
ROM:203315B7  I2S1 Bluetooth
ROM:204BE8AC  heap internal error ( memory overwriting? )
ROM:20109794  IMEI check failed for IMEI[%d]=%c\n
ROM:20547DF4  Not implemented functionality.
ROM:203AA51C  WARNING: NMEA buffer %s is too small %d<100\n
ROM:20412E60  UMTS Serving Cell:
ROM:204546BC  GPS~ E911 Agps Allowed Time set to %d s.\n
ROM:205133A0  ../../umts/design/ue-rrc/text/urrcm_l3_handle_cells.c
```


¿O RLMENTE?


DEVELOPMENT
ENVIRONMENT

Development Environment

Need to :

1. Compile our assembly code
2. Strip out the essential code and do reloc fixups
3. Package code in proper format for exploit

Compiler

- Using Code Sourcery Lite under Linux
  - Code Sourcery runs under Windows as well
  - X Code will compile (mach-o vs. ELF object files)

  http://www.codesourcery.com/sgpp/lite/arm/portal/release1802
- The tar version works

Makefile

```
CROSS_COMPILE=~/arm-2011.03/bin/arm-none-eabi-
CC=${CROSS_COMPILE}gcc

CFLAGS=-mcpu=arm926ej-s
CFLAGS+=-mthumb

all:   asm.o
    @echo "[*] building payload.bin"
    @python build_payload.py ys asm.o
    @echo "[*] done."

asm.o: asm.s
    @echo "[*] compiling asm.s => asm.o"
    @$ $(CC)  $(CFLAGS) -c asm.s

clean:
    rm -f *.o *~ *.bin
```

Build script

- A python script is called during the build
  - Finds .text section
  - Gets raw code
  - Performs relocation fixes
    - Must know loading address, exploit specific
  - Packages payload in usable exploit format

Build script

```
Usage: build_payload.py [options] payload_type objectfile.o

Required args:
  payload_type - type of payload, valid types:
    ys  - yellowsn0w
    ysr - yellowsn0w raw
    idt - injection dev tool (must set a base address, '-b')

  objectfile.o - file to strip payload_type from

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -b BASE_ADDR, --base_addr=BASE_ADDR
                        base address (idt payload_type only)
  -a, --arm_entry       starting addr is arm, default is Thumb
  -q, --quiet
```

Code injection

- Modified version of yellowsn0w reads in payload.bin file.
- The file contains the AT+STKPROF command
- Used in single shot mode

Sample assembly

```
    .text
    .align 2
    .code 16
_new_handler:
    push    {r4-r7,lr}
    ldr r3, jumptable_addr
    strh    r0, [r3]
    mov r0, r1
    ldr r1, ptr_my_response
    ldr r3, sprintf
    blx r3
    pop {r4-r7,pc}
```

```
/*  .datas */
    .align 4
jumptable_addr:
    .long 0x403BB344
sprintf:
    .long 0x2046DD00
ptr_my_response:
    .long my_response
my_response:
    .ascii "y0h! all up in ur
basebands!!!\0”
```

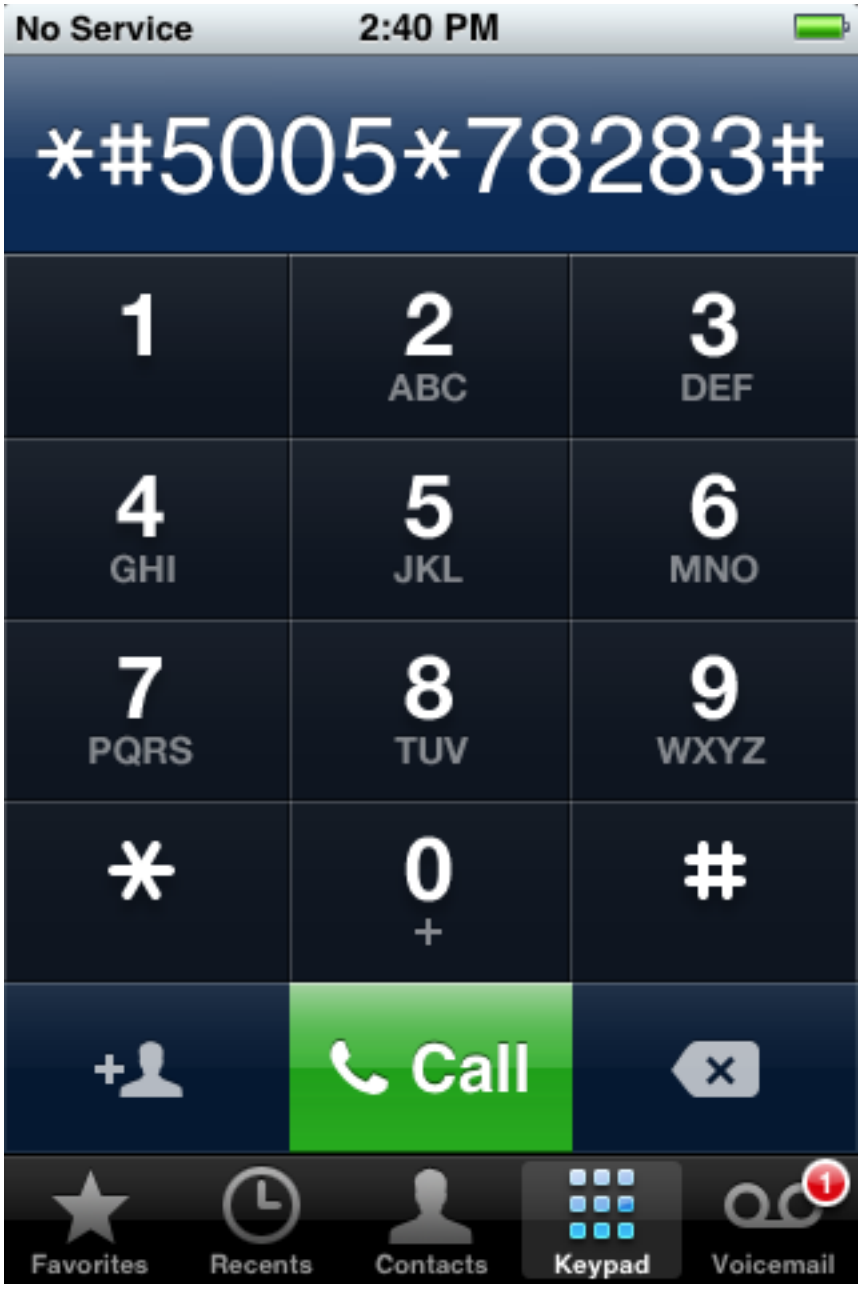
Sample C

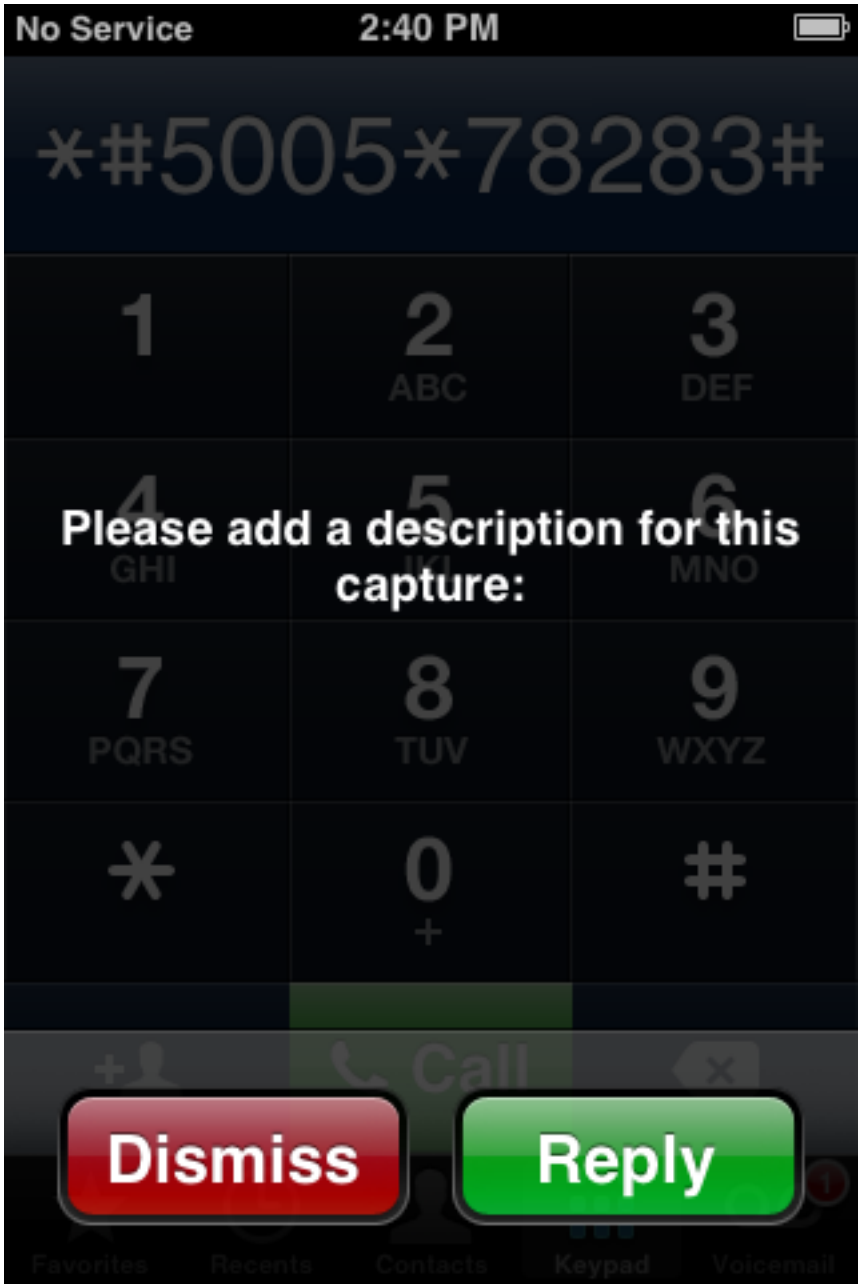
```
#define BYTECPY(x) void* (* x)(void*, void*, unsigned int)
#define BYTECPY_ADDR (void* (*)(void*, void*, unsigned int)) 0x203C58A0

int test(void)
{
    void* dest = (void*)0x11300;
    void* src =  (void*)0x2026B0F8;
    BYTECPY(my_bytecpy) = BYTECPY_ADDR;

    my_bytecpy(dest,src,4);
    return 0;
}
```

Debugging

- No interactive debugging
- Crash exception if we are lucky

1. Call the magic number, *#5005*78283#
2. Reply
3. Pick a description
4. Look in `/var/logs/CrashReporter/Baseband`


No Service
2:40 PM
*#5005*78283#
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
*
0 +
#
Call
Favorites
Recents
Contacts
Keypad
Voicemail
1


No Service
2:40 PM
*#5005*78283#
1
2
ABC
3
DEF
4
GHI
5
JKL
6
MNO
Please add a description for this capture:
7
PQRS
8
TUV
9
WXYZ
*
0
+
#
Call
Dismiss
Reply
Favorites
Recents
Contacts
Keypad
Voicemail


No Service
2:40 PM
Cancel
Reply
Send
Please add a description for this capture:
ekoparty
174 characters remaining
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
.?123
space
return

Crash dump (cleaned up)

```
+XLOG: Exception Number: 1
Trap Class: 0xBBBB (HW PREFETCH ABORT TRAP)
System Stack:

[  snip  ]
0x2030A740
0x20402049
0x20407ABD
0x2026D300
0x202CB7A3
Date: 10.09.2011
Time: 03:59:15
Register:
r0:   0x0000000C   r1:   0x1C544A06 r2:   0x00000050
r3:   0x20512A51   r4:   0x5F04044A r5:   0x5F05054A
r6:   0x5F06064A   r7:   0x5F07074A r8:   0x00000000
r9:   0x00000000   r10:  0x40565AA8 r11:  0x4056BAA8
r12:  0x45564E54   r13:  0x40566E98 r14:  0x20512A57
r15:  0x5F080848
SPSR: 0x00000013  DFAR:  0xFFFFCFFF DFSR: 0x00000005
OK
```

Debugging

Debugging

- As can be seen, Debugging is pretty caveman
- Once we can modify code, exceptions could be forced (invalid instruction) and registers inspected.
- Ideally could patch vector table in high memory and make our own exception handler.

DYNAMIC REVERSING

Dynamic Reversing

- The first thing we should do is dump memory.
- I found a memory dump of 02.28.00 online which appears to be gone.
- Within the dump I found code to a very useful tool.
- I don’t know who wrote it, but thanks :)

Dynamic Reversing

- I refer to the tool as IDT, injection development tool.
- The usage is:

```
usage: at@reset(cmd,addr,size,{byte array})
cmd: 1-read,2-write,3-call
```

- I packaged the tool into a AT+STKPROF payload

Dynamic Reversing

- Memory can now be dumped
- Memory map courtesy of iPhone wiki:

address	physical	d	size	permissions	glb	shr	access
C:00000000--00006FFF	A:00000000--00006FFF	00	00001000	P:noaccess U:noaccess	yes	no	uncached/buffered
C:00007000--0001FFFF	A:00007000--0001FFFF	00	00001000	P:readwrite U:noaccess	yes	no	uncached/buffered
C:00020000--1FFFFFFF							
C:20000000--200FFFFF	A:20000000--200FFFFF	00	00100000	P:readwrite U:noaccess	yes	no	write-through/buffered
C:20100000--20DFFFFF	A:20100000--20DFFFFF	00	00100000	P:noaccess U:noaccess	yes	no	write-through/buffered
C:20E00000--20FFFFFF	A:20E00000--20FFFFFF	00	00100000	P:readwrite U:noaccess	yes	no	uncached/unbuffered
C:21000000--3FFFFFFF							
C:40000000--400FFFFF	A:40000000--400FFFFF	00	00100000	P:readwrite U:noaccess	yes	no	uncached/unbuffered
C:40100000--43FFFFFF	A:40100000--43FFFFFF	00	00100000	P:readwrite U:noaccess	yes	no	write-back/buffered
C:44000000--5FFFFFFF							
C:60000000--600FFFFF	A:60000000--600FFFFF	00	00100000	P:readwrite U:noaccess	yes	no	uncached/unbuffered
C:60100000--7FFFFFFF							
C:80000000--800FFFFF	A:80000000--800FFFFF	00	00100000	P:readwrite U:noaccess	yes	no	uncached/unbuffered
C:80100000--EFFFFFFF							
C:F0000000--FFFFFFFF	A:F0000000--FFFFFFFF	00	00100000	P:readwrite U:noaccess	yes	no	uncached/unbuffered

Dynamic Reversing

- Interesting memory is in low memory, RAM, and high memory
- The goal is to add these into IDA
  - Keep in mind that RAM is just a snapshot in time

Combining dumps into IDA

This example loads a file dump_FFFF0000_10000.bin

1. File -> LoadFile -> Additional Binary File
2. Presented with a menu (changed the following)

   Loading segment: 0xffff000 // paragraphs aka /0x10

   Number of bytes: 0xfffc // cant have address wrap to 0
3. Change selectors

   This makes displayed addresses appear correctly

   View -> Open Subviews -> Selectors

   Change values to zero

Exploring memory

- One of the original yellowsn0w demos was a ps listing of running tasks.
- I implemented this in idapython in the combined idb.
- Simple code after learning the NU_TASK structure
- NU_TASK includes process name and address for entry function
- Script renames entry function using process name

Exploring Memory

```
[+] Collecting NU_TASKs
0x205675b9 dll:1[35]
0x205675b9 dll:2[35]
0x20594189 rrc:1[55]
0x202cb1c5 atc:1[85]
0x205b5ddd xdr:1[50]
0x2056a8c9 gps:1[90]
0x204d8a15 mon[120]
0x204043dd ata[84]
0x204429a9 io_evt[60]
0x202fcd2c sec[69]
0x202c8f4c xdrv_dat[150]
0x203e5e1c DMA[255]          NOTE: select set of tasks for display
```

Exploring Memory

- Similar things can be done with other resources
  - HISR
  - Semaphores
  - Etc.

Exploring Memory

- This is a rough overview of what is found in memory

Address	Contents
0x0000-0xFFFF	TLB, RAM, code
0x40000000-0x407fffff	RAM, Global Variables, General RAM
0xffff0000-0xffffffff	Vector Table, API functions, pointers


PATCHING
CODE

Patching Code

- Up to this point only have been able to run code that uses provided APIs
- Ideally we want to modify existing code.
- The code we want to modify is in ROM (flash).
- purplesn0w uses code patching to perform its unlock

```
source code: http://apt.geohot.com/purplesn0w_source.zip
```

Patching Code

1. Disable interrupts
2. Copy the page
3. Patch it
4. MMU disable
5. Write to TLB
6. Invalidate TLB
7. MMU enable
8. Flush instruction cache
9. Enable interrupts

Patching Code

```
/* PATCHES here: */

    LDR R0, =0x4306B0F8
    LDR R1, =0x646c6f6c
    LDR R2, =0x7a676e30
    LDR R3, =0x7265765f
    LDR R4, =0x6e6f6973
    LDR R5, =0x7325203a
    MOV R6, #0
    STR R1, [R0]
    ADD R0, R0, #4
    STR R2, [R0]
    ADD R0, R0, #4
    STR R3, [R0]
    ADD R0, R0, #4
    STR R4, [R0]
    ADD R0, R0, #4
    STR R5, [R0]
    ADD R0, R0, #4
    STR R6, [R0]

/* ************************** */
```

Patching Code

Original:

```
at+xgendata
+XGENDATA: "
"ICE2_MODEM_02.28.00",
"EEP_VERSION:526",
"EEP_REVISION:0",
“BOOTLOADER_VERSION: 5.9_M3S2",
1,0
OK
```

Patched:

```
at+xgendata
+XGENDATA: "
"ICE2_MODEM_02.28.00",
"EEP_VERSION:526",
"EEP_REVISION:0",
"loldOngz_version: 5.9_M3S2",1,0
OK
```

NOTE: edited and trimmed for display purposes

Patching Code

- At this point, we can add hooking code.
- Can add debugging logging removed in release build
- Can modify the vector table in high memory and write our own exception handler, aka our own debugger

Patching Code

- Many interesting areas of code to modify
- The main Tasks are good starting points.
- The GSM and UMTS code has many strings, making identification easy
- Appear to be internal testing tools and scripting in firmware.
- Should only be done on a test network
- Check with any laws in your area

CONCLUSION

Conclusion

- Baseband is just another embedded system.
- Using unlocks allows for runtime access.
- Combining runtime access with a development environment and existing RE methods allows for easy exploration.

Questions?