



DECEMBER 4-7, 2017
EXCEL / LONDON, UK

Mark Ermolov
Maxim Goryachy

POSITIVE TECHNOLOGIES

How to Hack a Turned-Off Computer, or Running Unsigned Code in Intel Management Engine

🐦 #BHEU / @BLACKHATEVENTS

We Are Going To...

- Reveal an Intel ME vulnerability (CVE-2017-5705,6,7) allowing **arbitrary code execution**
- Emphasize the **dangers** of such bugs in digitally signed firmware
- Show how we **bypassed** built-in exploitation mitigations
- Disclose **architecture flaws** in Intel ME

Who is Mark Ermolov

System programmer interested in security aspects of hardware, firmware, and low-level system software. Currently researching inner workings of Intel platforms (PCH, IOSF, iGPU).

@_markel__
mermolov{at}ptsecurity.com



Who is Maxim Goryachy

System and embedded developer and security researcher. Interested in cryptography, virtualization technologies, reverse engineering, and hardware.

@h0t_max
mgoryachy{at}ptsecurity.com



Our Research Team

Maxim Goryachy



Tapping into the Core



Intel DCI Secret

Mark Ermolov

Dmitry Sklyarov



Intel ME: The Way of the
Static Analysis



Intel ME: Flash File System Explained

- Intel Management Engine 11 overview
- Known public vulnerabilities
- Potential attack vectors
- Vulnerability
- Bypassing mitigation
- Possible exploitation vectors
- Demo

Intel Management Engine 11 Overview

Intel ME 11

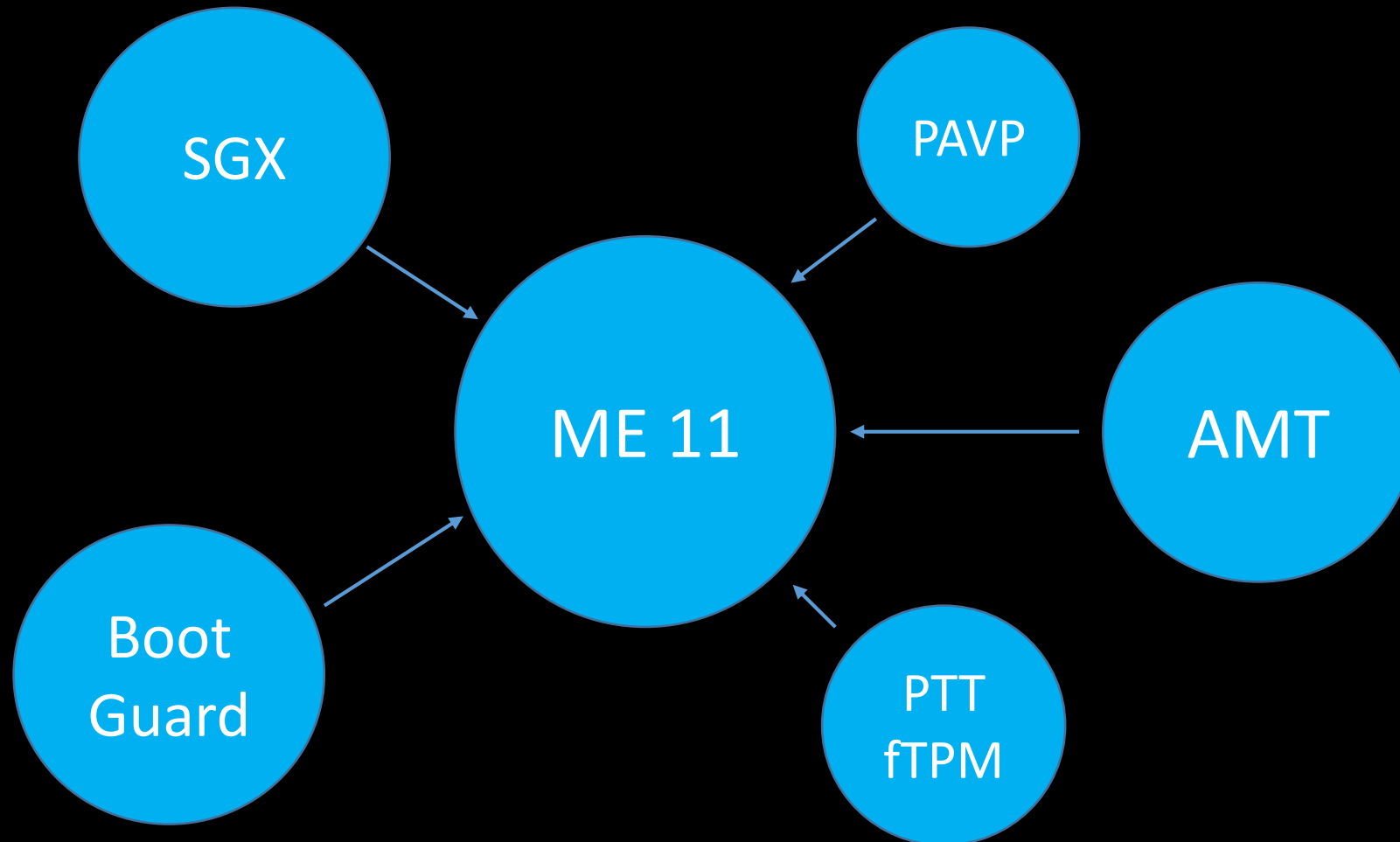
- Excluding publications such as Dr. Ruan's book *[PSTR14]*, ME is a **partially documented** Intel technology with proprietary firmware
- **Root of trust** for multiple security features such as PAVP, PTT and Boot Guard
- Has **full access** to many Intel hardware devices
- Has **hardware capabilities for interception** of some user activity
- An integral component for **all stages** of the platform operating cycle

Intel ME 11 Design

- Independent 32-bit processor core (x86)
- Running own modified MINIX
- Has a built-in Java machine *[IMS14]*
- Interacts with CPU/iGPU/USB/DDR/PCI/...
- Works when main CPU is powered down (M3 mode)
- Starter code is burned in non-reprogrammable on-die memory

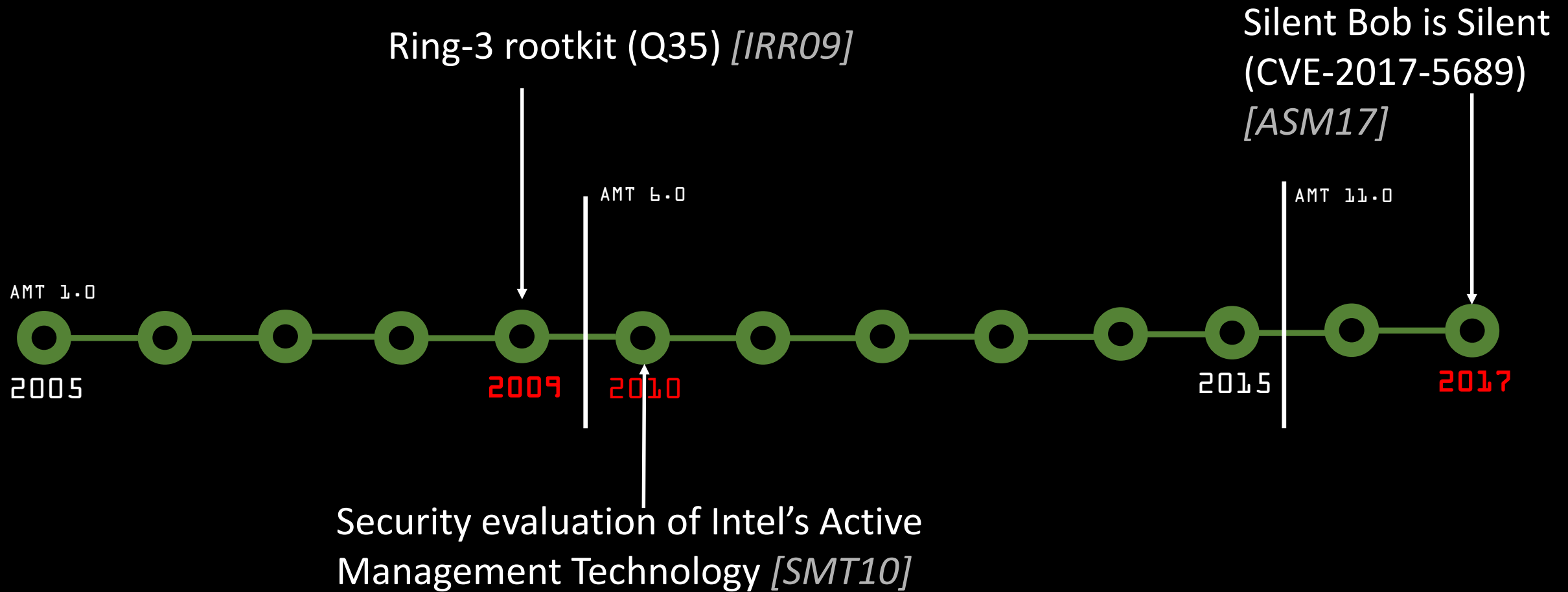
- Intel Active Management Technology
- Intel Protected Audio Video Path
- Intel Platform Trust Technology
- Intel Software Guard Extensions *[PSTR14]*
- Intel Boot Guard
- ...

Intel ME 11 – Related Technologies



Known Public Vulnerabilities

Known Public Vulnerabilities



Known Public Vulnerabilities



- Alexander Tereshkin and Rafal Wojtczuk of Invisible Things Lab: Introducing Ring-3 Rootkits (**code execution**)
- Vassilios Ververis of the Royal Institute of Technology: Security Evaluation of Intel's Active Management Technology (**AMT authentication bypass**)
- Dmitriy Evdokimov, Alexander Ermolov, and Maksim Malyutin of Embedi: Silent Bob is Silent (**AMT authentication bypass**)

Over the past 12 years, **only one**
vulnerability allowing execution of
arbitrary code on ME has been found!

Now we have two of them!

Potential attack vectors (ways to impact)

How we can impact Intel ME



- Local communication interface (HECI)
- Network (vPro only)
- IPMI/MCTP
- Host memory (UMA)
- Firmware SPI layout
- Internal file system

- Main interface for communication between host and ME
- Represented as PCI device
- Transports dozens of ME service protocols
- Undocumented; some protocol formats can be found in coreboot
- MEBx and BIOS use HECI to set up ME
- Used by Intel tools for updating and manufacture-line configuring

Network (vPro only)

- ME implements various industry-standard protocols (IP, HTTP, WPA2, KERBEROS)
- Has built-in full-fledged web and VNC servers
- Complete platform control is exposed in XML-based WSMAN protocol
- Most functionality is in one large module (AMT)

Currently UMA is encrypted by AES with integrity checking via unknown cyclic redundancy code

- Has complex structure
- We found bugs in parsing procedures of signed data (not exploitable if you don't have Intel's private key)
- Firmware code is generally not vulnerable to "evil SPI flash" attack



Have you attended
Intel ME: Flash File System Explained
by Dmitry Sklyarov? ;)

Potential attack vectors (which modules?)

Access Control Model

- Process is a subject of access control
- A process has statically defined access rights
 - User and groups identity for file system access
 - List of allowed hardware resources
 - List of allowed kernel syscalls

Architecture Problems

- **A process with permission to create new processes can spawn one more privileged than itself**
- **Access to some internal devices completely breaks the security model**

Modules With High Privileges



- “ROM”
- RBE
- KERNEL
- BUP
- LOADMGR
- PM

BUP is very tempting

- First user-mode process
- Exists on all platforms
- Has access to security-sensitive hardware (e.g., DMA controller)
- Can create new processes
- Performs early platform initialization
- Can bypass MFS protection (via SPI controller)
- Builds basic configuration for all other processes

BUG: More Reasons

- One of the largest modules
- Duplicates a lot of other modules' functionality
- Processes large amount of configuration data
- Interacts with the host via HECI

Trace Hub Initialization

```
void __cdecl bup_init_trace_hub()
{
    int err; // eax
    signed int npk_reg_idx; // ebx
    unsigned int bytes_read; // [esp+0h] [ebp-350h]
    unsigned int file_size; // [esp+4h] [ebp-34Ch]
    int si_features[5]; // [esp+8h] [ebp-348h]
    int ct_data[202]; // [esp+1Ch] [ebp-334h] 808 bytes
    int cookie; // [esp+344h] [ebp-Ch]

    cookie = gRmlbCookie;
    memset(si_features, 0, 0x14u);
    bytes_read = 0;
    file_size = 0;
    if ( !(getDW_sel(0xBF, 0xE0u) & 0x1000000)
        && !bup_get_si_features(si_features)
        && !bup_dfs_get_file_size("/home/bup/ct", &file_size) )
    {
        if ( file_size )
        {
            LOBYTE(err) = bup_dfs_read_file("/home/bup/ct", 0, ct_data, file_size, &bytes_read);
            npk_reg_idx = 0;
            if ( !err )
            {
                while ( npk_reg_idx < HIWORD(ct_data[1]) )
                {
                    if ( HIBYTE(ct_data[2 * npk_reg_idx + 2]) == 1 )
                        putDW_sel(0xB7, ct_data[2 * npk_reg_idx + 2] & 0xFFFF, ct_data[2 * npk_reg_idx + 3]);
                    if ( HIBYTE(ct_data[2 * npk_reg_idx + 2]) == 2 )
                        putDW_sel(0xBF, ct_data[2 * npk_reg_idx + 2] & 0xFFFF, ct_data[2 * npk_reg_idx + 3]);
                    ++npk_reg_idx;
                }
                bup_switch_tracer(0xB7, 0xBFu);
            }
        }
    }
    if ( gRmlbCookie != cookie )
        sys_fault();
}
```


The Vulnerability

```
void __cdecl bup_init_trace_hub()
{
...
    int ct_data[202]; // [esp+1Ch] [ebp-334h] 808 bytes
    int cookie; // [esp+344h] [ebp-Ch]

    cookie = gRmlbCookie;
...
    if ( !(getDW_sel(0xBF, 0xE0u) & 0x1000000)
        && !bup_get_si_features(si_features)
        && !bup_dfs_get_file_size("/home/bup/ct", &file_size) )
    {
        if ( file_size )
        {
            LOBYTE(err) = bup_dfs_read_file("/home/bup/ct", 0, ct_data, file_size, &bytes_read);
...
        if ( gRmlbCookie != cookie )
            sys_fault();
    }
}
```

808-byte Mystery

```
void __cdecl bup_init_trace_hub()
{
...
    int ct_data[202]; // [esp+1Ch] [ebp-334h] 808 bytes
...
}
```

Trace Hub Configuration Binary



name	mode	opt	cb	uid	gid	offset	mode	opt	path
..	13C0	0000	0000	0003	0000	00003388			/home/
ct	01E0	0009	0000	0003	015F	0000338B	--- rwxr ----	? -- F	/home/bup/ct
df_cpu_info	01FF	0009	0004	0003	00CE	0000338B	- rwxrwxrwx -	? --	/home/bup/df_cpu_info

Not signed

From fitc.cfg

```
void __cdecl bup_init_trace_hub()
{
...
    int ct_data[202]; // [esp+1Ch] [ebp-334h] 808 bytes
    int cookie; // [esp+344h] [ebp-Ch]

    cookie = gRmlbCookie;
...
    if ( !(getDW_sel(0xBF, 0xE0u) & 0x1000000)
        && !bup_get_si_features(si_features)
        && !bup_dfs_get_file_size("/home/bup/ct", &file_size) )
    {
        if ( file_size )
        {
            LOBYTE(err) = bup_dfs_read_file("/home/bup/ct", 0, ct_data, file_size, &bytes_read);
...
        if ( gRmlbCookie != cookie )
            sys_fault();
    }
}
```

Stack Guard Implementation



- Each process has unique value for stack cookie
- Value is obtained from hardware random number generator
- Stored in nonvolatile process memory
- If stack's copy of cookie is changed, process terminates

Bypass mitigations

How to Bypass Stack Guard?



- Break random number generator
- Intercept code flow before cookie checking

Break Random Number Generator

```
signed int InitRandDev()
{
    signed int i; // edx@1
    signed int result; // eax@4

    dev_rnd_seed = 0xDC80;
    dev_rnd_conf = 0x44050;
    RandDevPriming(0x190u);
    i = 1001;
    while ( !(dev_rnd_sts & 1) )
    {
        if (!--i)
            return DEVEERROR;
    }
    result = DEVEERROR;
    if ( (dev_rnd_sts & 0xF) == 15 )
        result = NOERROR;
    return result;
}
```

If Random Number Generator is broken, RBE doesn't start at all

How to Bypass Stack Guard?



- Break random number generator
- Intercept code flow before cookie checking

Code Flow (For C Programmers)

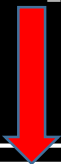
```
void __cdecl bup_init_trace_hub()
{
...
    LOBYTE(err) = bup_dfs_read_file("/home/bup/ct", 0, ct_data, file_size, &bytes_read);
...
}
```



```
char __cdecl bup_dfs_read_file(char *file_name, int offset, char *buffer, unsigned int read_size, unsigned int *out_bytes_read)
{
...
    *out_bytes_read = read_size;
    LOBYTE(res) = bup_read_mfs_file(7, fitc_file_desc.data_offset + offset, out_bytes_read, sm_mem_id, 0);
...
}
```

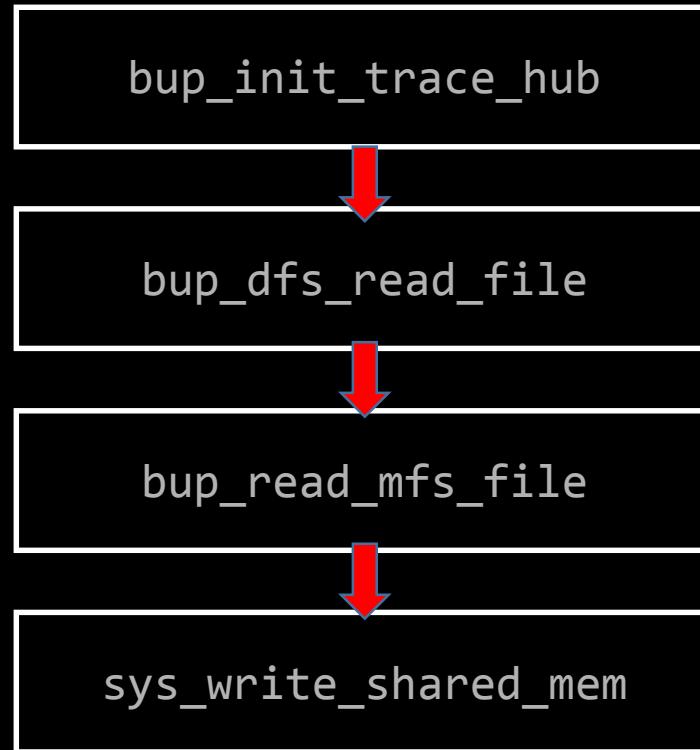


```
int __cdecl bup_read_mfs_file(BUP_MFS_DESC *mfs_desc, int file_number, unsigned int offset, unsigned int *size, int sm_block_idx, __int16 proc_thread_id)
{
...
    while ( 1 ) {
        if ( cur_offset >= read_size ) break;
...
        err = bup_mfs_read_data_chunks(mfs_desc, buffer, mfs_desc->data_chunks_offset + ((read_start_chunk_id - mfs_desc->total_files) << 6), block_chunks_count);
...
        err = sys_write_shared_mem(proc_thread_id, sm_block_idx, cur_offset, &buffer[chunk_offset], copy_size, copy_size);
...
    }
...
}
```



```
signed int __cdecl sys_write_shared_mem(__int16 owner_proc_thread_id, int block_idx, int offset, char *src_data, unsigned int src_size, unsigned int write_size)
{
...
    sm_block_desc = sys_get_shared_mem_block(block_idx);
...
    memcpy_s((sm_block_desc->start_addr + offset), sm_block_size - offset, src_data, write_size);
...
}
```

Code Flow (For People)



Inside sys_write_shared_mem

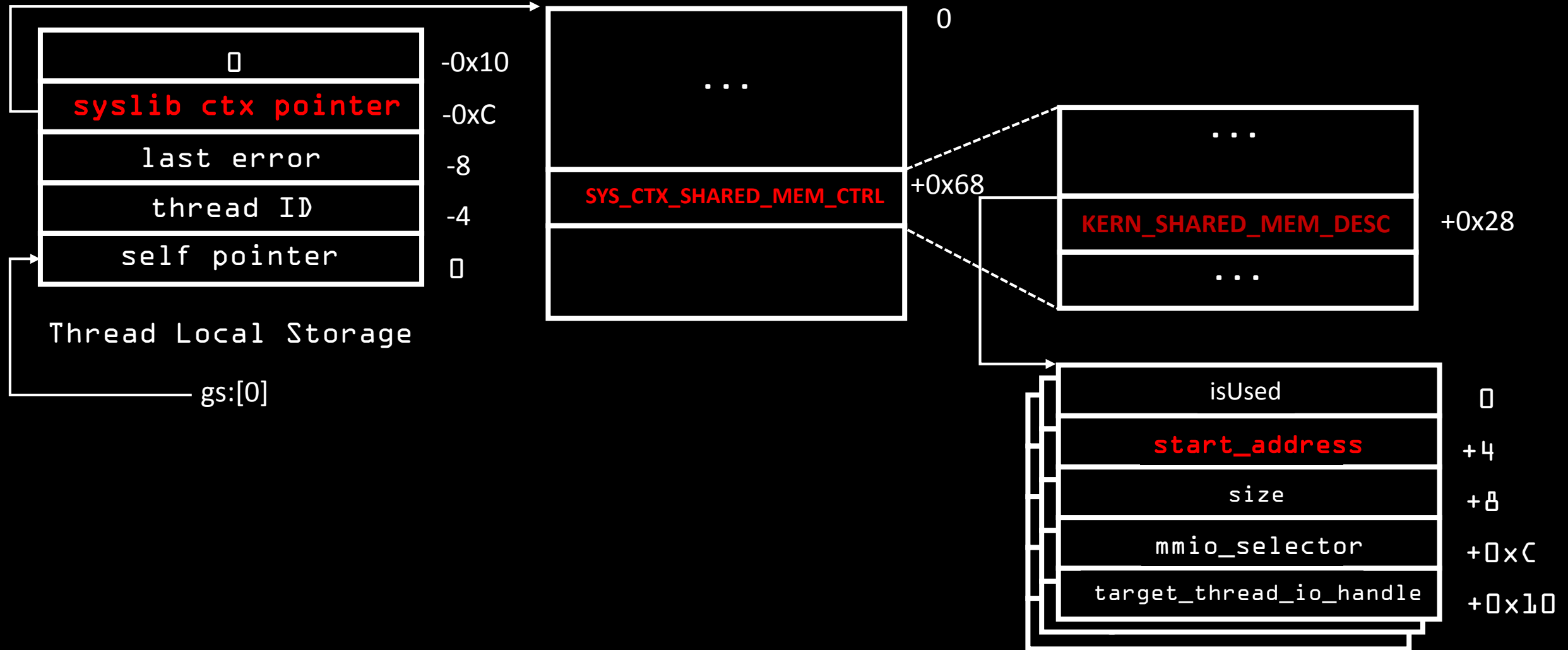
```
signed int __cdecl sys_write_shared_mem(...)  
{  
...  
    sm_block_desc = sys_get_shared_mem_block(block_idx);  
...  
    memcpy_s((sm_block_desc->start_addr_linked_block_idx + offset), sm_block_size - offset, src_data,  
write_size);  
...  
}
```

[...]

```
int __cdecl sys_get_ctx_struct_addr(SYS_LIB_CTX_STRUCT_ID struct_id)  
{  
...  
    sys_ctx_start_ptr = sys_get_tls_data_ptr(SYSLIB_GLB_SYS_CTX);  
    switch ( struct_id ) {  
        case SYS_CTX_SHARED_MEM:  
            addr = *sys_ctx_start_ptr + 0x68;  
            break;  
...  
    }  
    return addr;  
}
```

```
sys_get_tls_data_ptr proc near  
tls_idx = dword ptr 8  
    push    ebp  
    mov     ebp, esp  
    mov     eax, large gs:0 ???  
    mov     ecx, [ebp+tls_idx]  
    pop     ebp  
    lea     edx, ds:0[ecx*4]  
    sub     eax, edx  
    retn  
sys_get_tls_data_ptr endp
```

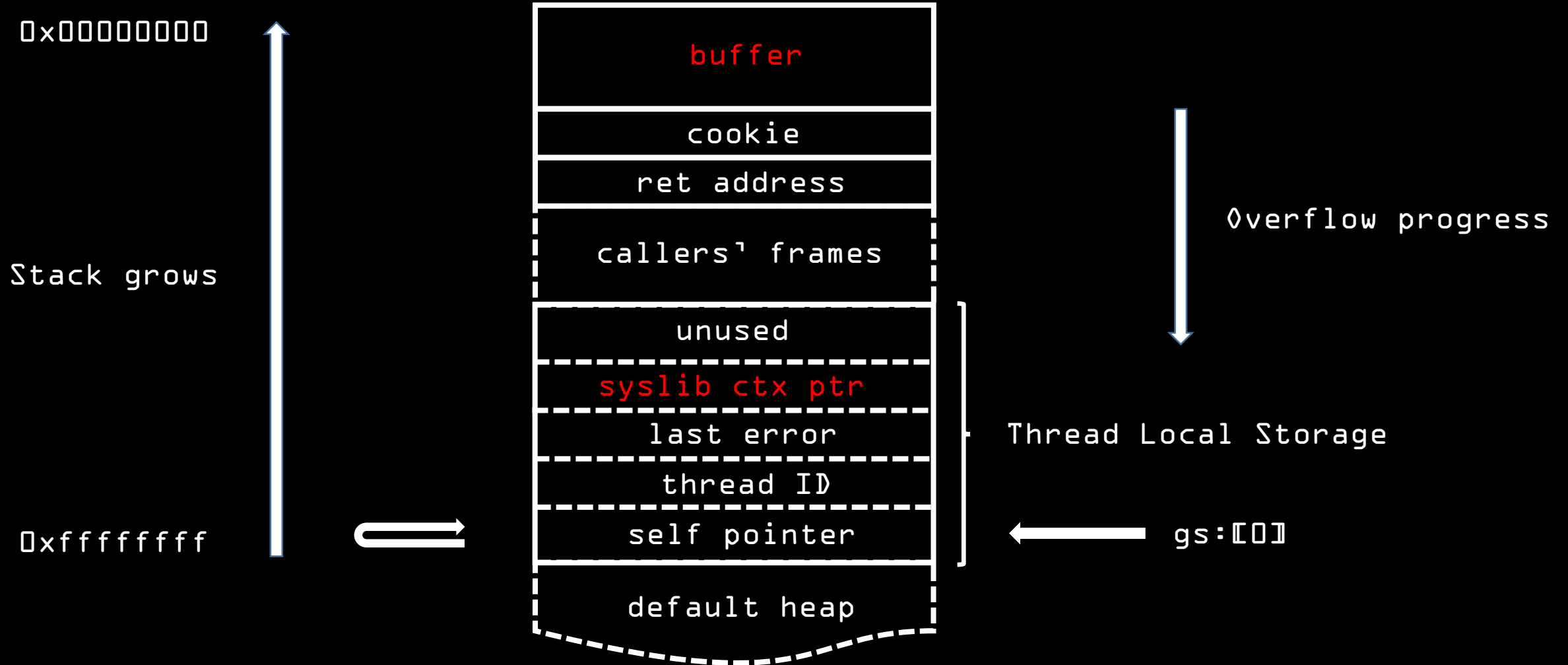
What is gs:[0]?





**Thread Local Storage
resides at thread stack bottom**

Stack Layout



- Function `bup_read_mfs_file` gets address of destination buffer from `syslib_context`
- Reads file data and writes to destination by chunks iteratively
- We can get arbitrary write primitive replacing `syslib_context` pointer
- We can rewrite `memcpy`'s return address



Non-Executable Stack

Non-Executable Stack

Problem:

Stack segment doesn't intersect with code segment

Solved:

We found ROP gadgets for own process creation



- Create own code module and integrate it into firmware
- Using ROP, load the module into memory
- Using ROP, create new process with highest privileges

Vulnerability Overview



- CVSSv3: **AV:L**/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H (8.2 High)
- Attacker with local access to the system can load and execute arbitrary code
- Affected Intel® Management Engine (ME), Intel® Server Platform Services (SPS), and Intel® Trusted Execution Engine (TXE)

Affected Products

- 6th, 7th & 8th Generation Intel® Core™ Processor Family
- Intel® Xeon® Processor E3-1200 v5 & v6 Product Family
- Intel® Xeon® Processor Scalable Family
- Intel® Xeon® Processor W Family
- Intel® Atom® C3000 Processor Family
- Apollo Lake Intel® Atom Processor E3900 series
- Apollo Lake Intel® Pentium™
- Celeron™ N and J series Processors

Possible Exploitation Vectors



Restriction:

Attacker needs write access to MFS partition of
ME SPI region

Ways to Rewrite ME SPI Region



- Mistakes in SPI flash region settings in SPI flash descriptor
- Via HMR-FPO HECI message
 - ✓ Manufacture mode
 - ✓ Attack on UEFI setup variable
 - ✓ DMA attack
- Security Descriptor Override jumper
- SPI programmer
- ...

Is Remote Exploitation Possible?



- Yes, if:
 - ✓ AMT is enabled on the target and attacker knows password*
 - ✓ BIOS has “Flash Rewrite Enable” option
 - ✓ BIOS password is blank or known
- *Attacker can use AMT authentication bypass vulnerability (CVE-2017-5689)

BAD NEWS



HAP is **no cure-all** (for CVE-2017-5705,6,7)

Hence HAP protects against vulnerabilities present in all modules except RBE, KERNEL, SYSLIB, ROM, and BUP. However, unfortunately this mode does not protect against exploitation of errors at earlier stages.

*Mark Ermolov, Maxim Goryachy
Disabling Intel ME 11 via undocumented mode *[DMU17]*

Bad News – Firmware Downgrade

Intel Firmware Update against CVE-2017-5705,6,7
doesn't help because **ROM allows ME downgrading**



ME 11.8.50.3399

TLS is still **in at the same place**



Demo time

AMT on non-vPro platform

JTAG for Intel ME

Hello from Intel ME

Intel ME 11 – Related Technologies

- Intel Active Management Technology
- Intel Protected Audio Video Path
- Intel Platform Trust Technology (fTPM)
- Intel Software Guard Extensions
- Intel Boot Guard

IN DANGER

**Waiting for Intel Management Engine 13!
(maybe Intel will remove it from PCH... ;)**

Our Achievements

- Switched-on AMT on non-vPro systems
- Activated JTAG for Intel ME via the vulnerability
- Dumped starter code (aka ROM)
- Recovered complete Huffman code for ME 11
- Extracted Integrity and Confidentiality Platform Keys [FFS17]
- Bypassed Intel Boot Guard

- Positive Technologies for allowing us to spend part of our working time on it!
- Dmitry Sklyarov
- Plato Mavropoulos
- People who have helped us but don't want their names to be published ;)

Disclosure timeline

- ✓ 27/06/2017 - Bug reported to Intel PSIRT
- ✓ 28/06/2017 - Intel started initial investigation
- ✓ 05/07/2017 - Intel requested proof-of-concept
- ✓ 06/07/2017 - Additional information sent to Intel PSIRT
- ✓ 17/07/2017 - Intel acknowledged the vulnerability
- ✓ 28/07/2017 - Bounty payment received
- ✓ 20/11/2017 - Intel published SA-00086 security advisory

- [IMS14] Igor Skochinsky, Intel ME Secrets. Hidden code in your chipset and how to discover what exactly it does. Hex-Rays. RECON 2014.
- [STW17] Dmitry Sklyarov, ME: The Way of the Static Analysis. Troopers 2017.
- [FFS17] Dmitry Sklyarov, Intel ME: flash file system explained, Black Hat Europe, 2017.
- [IRR09] Alexander Tereshkin, Rafal Wojtczuk, Introducing Ring-3 Rootkits. Black Hat USA, 2009 Las Vegas, NV.
- [SMT10] Vassilios Ververis, Security Evaluation of Intel's Active Management Technology, Sweden 2010 TRITA-ICT-EX-2010:37.
- [PSTR14] Xiaoyu Ruan, Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine, 2014, Apress, ISBN 978-1-4302-6572-6.
- [ASM17] Dmitriy Evdokimov, Alexander Ermolov, Maksim Malyutin, Intel AMT Stealth Breakthrough, Black Hat USA, 2017 Las Vegas, NV.
- [DMU17] Mark Ermolov, Maxim Goryachy, Disabling Intel ME 11 via undocumented mode, <https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Intel-ME-disable-eng.pdf>, 2017.

Thank you!

Questions?

Mark Ermolov
Maxim Goryachy

POSITIVE TECHNOLOGIES