Mark Ermolov
Maxim Goryachy

Inside Intel Management Engine

34th Chaos Communication Congress, Leipzig, 2017

POSITIVE TECHNOLOGIES

ptsecurity.com

Research Team

- Mark Ermolov
  email: mermolov {at} ptsecurity {dot} com
  twitter: @_markel___
- Maxim Goryachy
  email: mgoryachy {at} ptsecurity {dot} com
  twitter: @h0t_max
- Dmitry Sklyarov
  email: dsklyarov {at} ptsecurity {dot} com
  twitter: @_Dmit

Roadmap

- Intel Management Engine: Quick Start
- Intel’s JTAG: Overview
- JTAG for ME: How Does It Work?
- Activation Without Intel Keys
- DFx Abstraction Layer
- Developing ME Core Configuration
- Demo

Intel Management Engine

Quick Start

Intel Management Engine (ME)

- **Poorly documented** Intel technology with proprietary firmware
- **Root of trust** for security features such as PAVP, PTT, and Boot Guard
- **Full access** to many Intel devices
- **Hardware capabilities for interception** of user activity
- Integral component for **all stages** of the platform operating cycle

Intel ME 11: Implementation Details

- Independent 32-bit processor core (x86)
- Runs its own modified MINIX [STW17]
- Has a built-in Java machine [IMS14]
- Interacts with CPU/iGPU/USB/DDR/PCI/…
- Operates when main CPU is powered down (M3 mode)
- Contains starter code in non-reprogrammable on-die memory

Intel’s JTAG

Overview

JTAG Overview

- JTAG, Joint Test Action Group IEEE 1149
- Essential mechanism for debugging electronic chips
- JTAG-based debugging is available immediately after processor core reset
- *Maxim Goryachy, Mark Ermolov, Where there's a JTAG there's a way: obtaining full system access via USB:* details about JTAG in modern Intel's platform

Intel DCI

- **Intel Direct Connect Interface (DCI)** is a debug transport technology designed to enable closed chassis debug through a **USB3** port from Intel silicon
- Intel DCI provides access to CPU/PCH JTAG via USB3.0
- **Software is available without NDA (Intel System Studio)**
- There are two types of DCI hosting interfaces in the platform:
  - ✓ USB3 Hosting DCI (USB-Debug cable)
  - ✓ BSSB Hosting DCI (Intel SVT Closed Chassis Adapter)

Available starting with 6th generation Intel® Core™ processor family

JTAG + ME = ?

Unlimited research of
a modern x86 architecture

How Does It Work?

JTAG for ME

Unlock Token

UTOK (unlock token) or STOK (security token) is a special partition in ME region:

- Integrated via FPT, HECI, DCI, or directly via an SPI programmer
- Unique for the platform and temporary
- Unlocking modes: ORANGE and RED
- Designed to activate DFx functionality for Intel Management Engine

About DFx

- DFx stands for design for manufacturability, testability, and debuggability
- DFx is a private implementation of JTAG (1149.1 and 1149.7) by Intel
- There are many integrated devices coupled to a DFx chain inside PCH and CPU
- Embedded DFx Interface (ExI) is used to access DFx
- ExI connects DFx and the external interface (such as USB)

ORANGE

- Provides access to IOSF*
- Unlocks JTAG for ISH core*
- Enables debugging of the ISH program via GDB-stub or DCI

N.B. UTOK partition must be signed by vendor's key.

* Our team has found a server firmware image with ORANGE unlock support (provides access to IOSF on the server's motherboard), but hasn't found a similar image for desktops.

RED

- Provides access to IOSF
- Unlocks JTAG for ME core
- Unlocks JTAG for ISH core
- Enables debugging from the reset vector (S0) before starting the main CPU
- Provides unlimited access to internal devices and memory

N.B. UTOK partition must be signed by Intel key

ME JTAG Activation Interface

- PCH has a special internal device DFX_AGGREGATOR that controls access to DFx
- BUP and ROM have direct access to the CSE zeroing register and DFX_AGGREGATOR device (via LDT selector)

```
Ext#8 MmioRanges[41]:
    ...
    sel= FF, base:F00B1050, size:00000004, flags:00000003 :: F00B1000:00001000 GEN_PCIP
    sel=107, base:F00B1004, size:00000004, flags:00000003 :: F00B1000:00001000 GEN_PCIP
    sel=10F, base:F5010000, size:00001000, flags:00000003 :: F5010000:00008000 DFX_AGGREGATOR_SBS
    ...
```

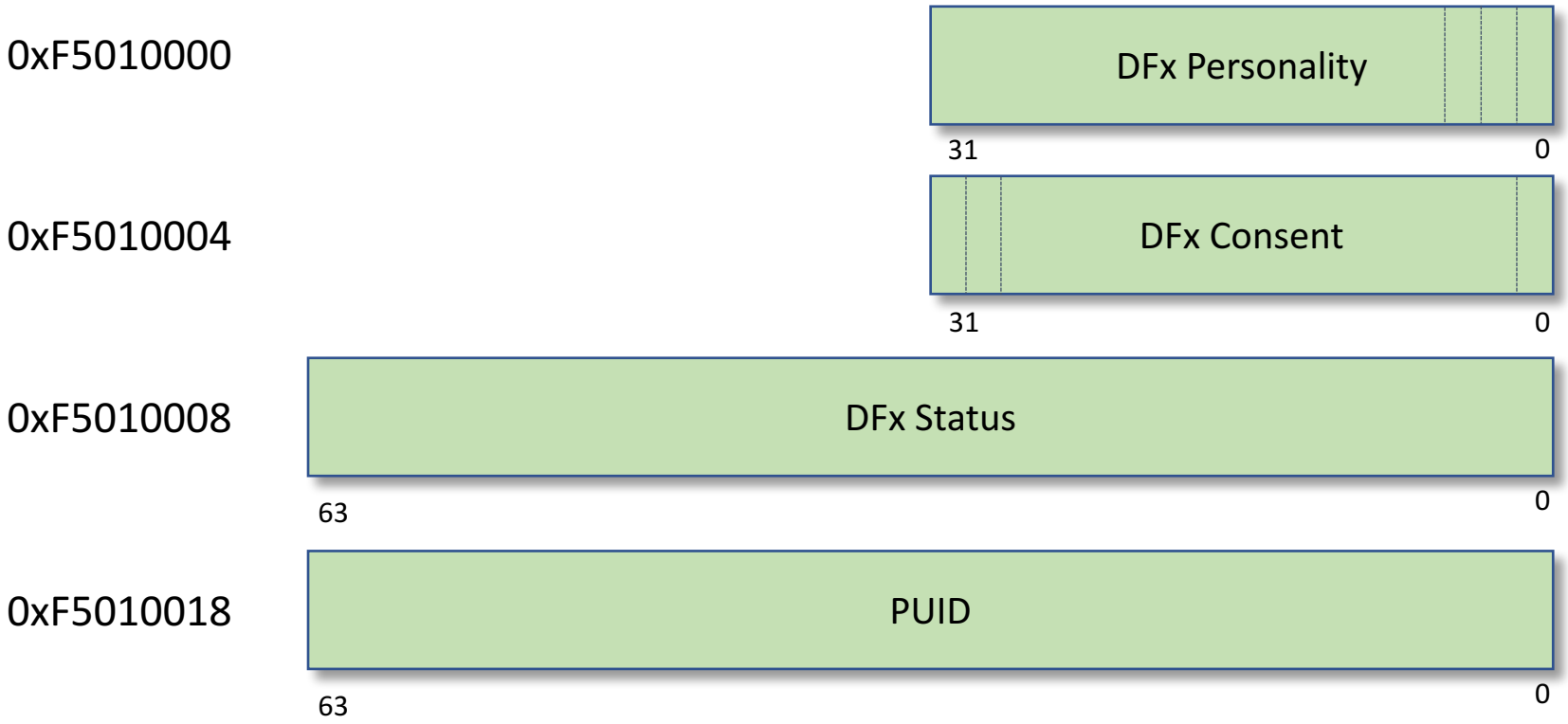
Activation (I)

0xF00B1050

CSE zeroing register (bit)	
0	Intel Unlock Request (R/W)
31..1	Reserved

Activation (II)

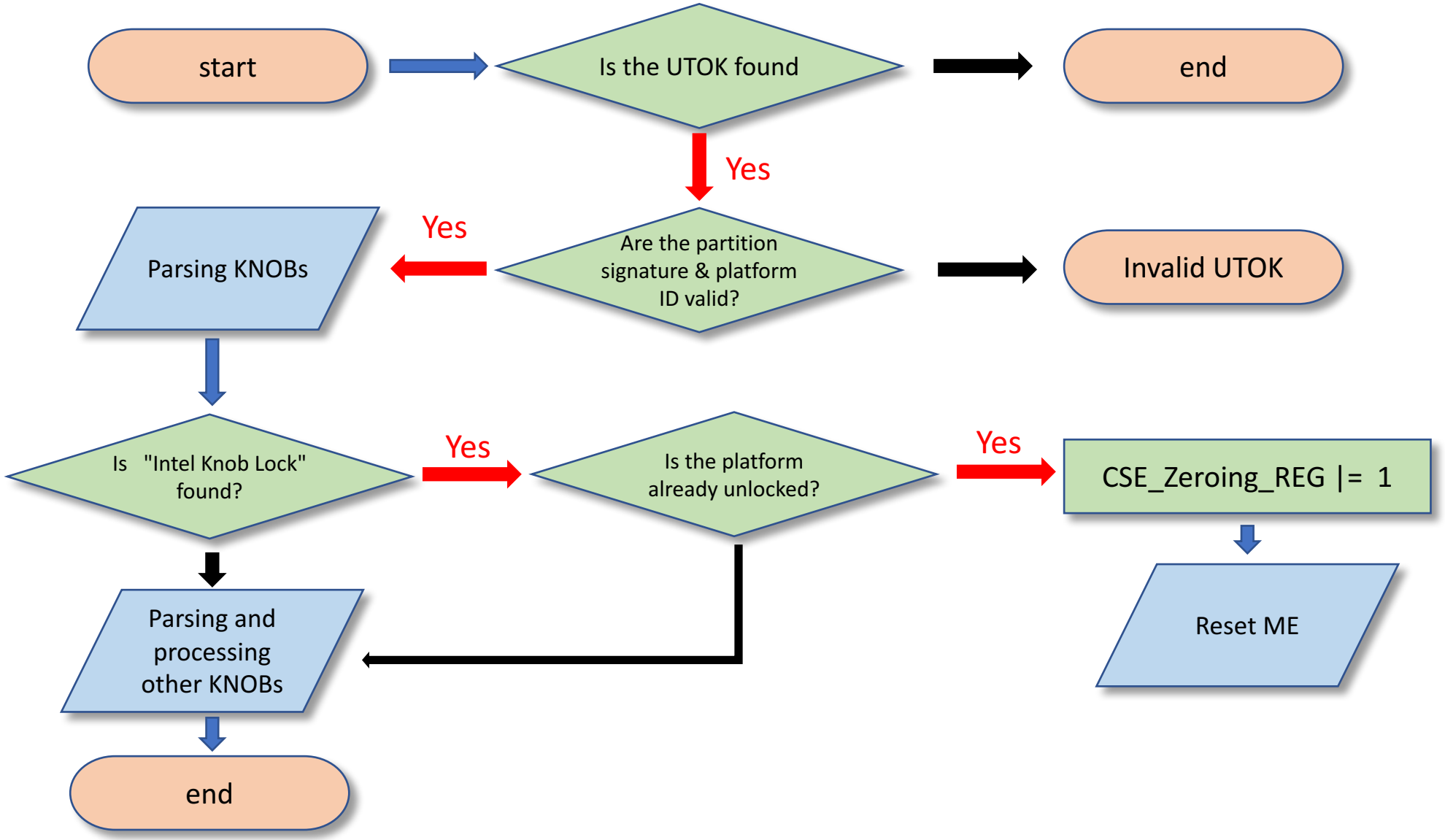
DFx Aggregator MMIO:



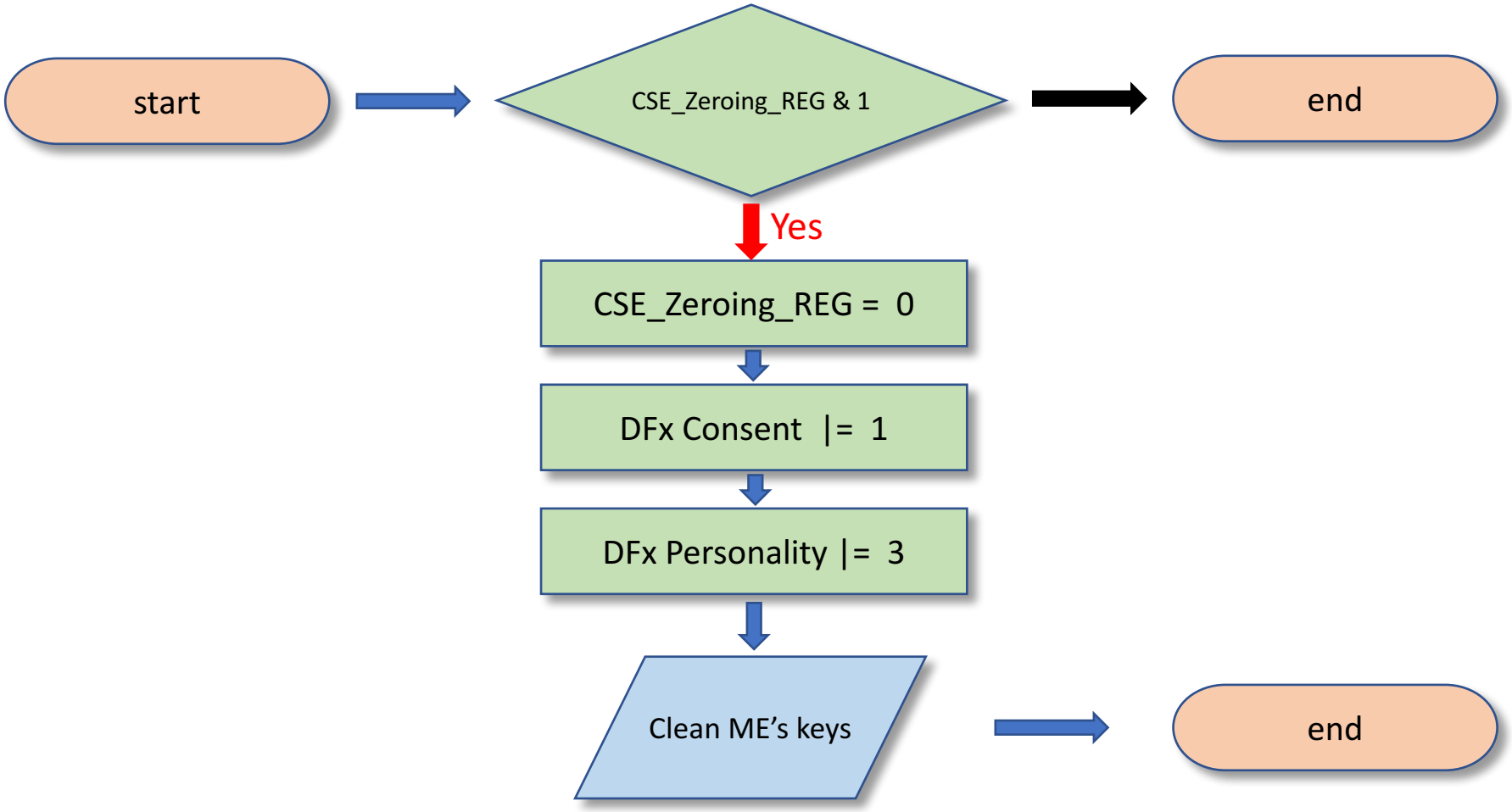
DFx Personality value (2..0)	Unlock type
101	ORANGE
011	RED

DFx Consent bits	Unlock type
0	Unlock Consent
...	
30	Lock Bit

RED Unlock: BUP



RED Unlock: ROM



Latching Consent Register

```
void bup_switch_on_dci()
{
...
  eom = 0;
  bup_get_pch_straps(0, &pch_desc_rec0);
  LOBYTE(eom_err) = bup_read_eom(&eom); // Is the platform in Manufacture Mode?
  if ( !(BYTE2(pch_desc_rec0) & 2) || (dfx_data |= 2u, eom_err) || eom )
    bup_disable_dci_by_strap();
  else
    bup_enable_dci_by_strap();
  if ( bup_is_dci_active() == 1 ) // If dci is active ME doesn't latch DFx consent register
    bup_set_dfx_agg_consent();
  else
    bup_lock_dfx_agg_consent();
  if ( gRmlbCookie != cookies )
    sys_fault();
}
```

Is it a design flaw or not?

Red Activation Without Intel Keys

JTAG for ME

CVE-2017-5705,6,7

```
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();
}
```

Vulnerability in BUP module [HTH17]

ME JTAG How-To

Arbitrary code execution in the BUP module (CVE-2017-5705,6,7)

Activation of RED UNLOCK without Intel keys

JTAG access to ME core

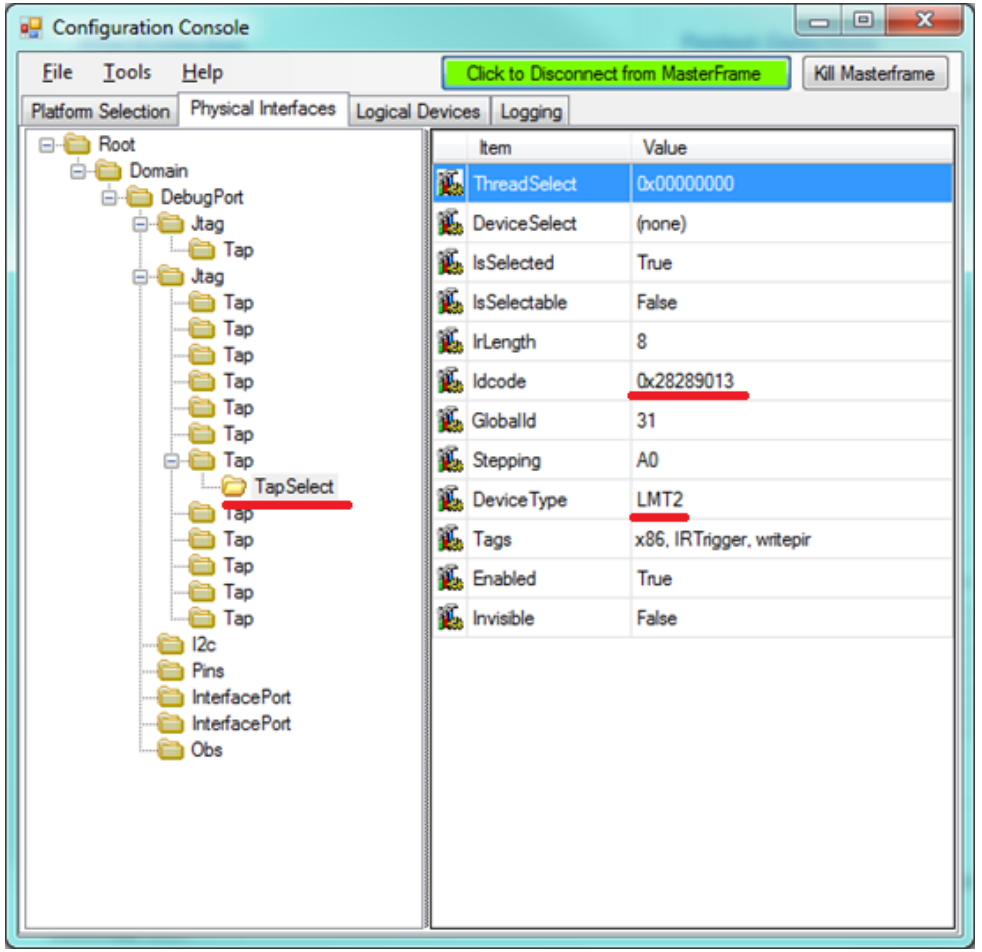
Full control over the target

ME is no longer a "black box"

Red Activation Without Intel's Crypto Keys

1. Activate Manufacture Mode for the target
2. Set DCI strap in a flash descriptor
3. Use the vulnerability to load the value 3 to DFx Personality register
4. Done ;)

RED is Activated for Target



ME core JTAG device ID

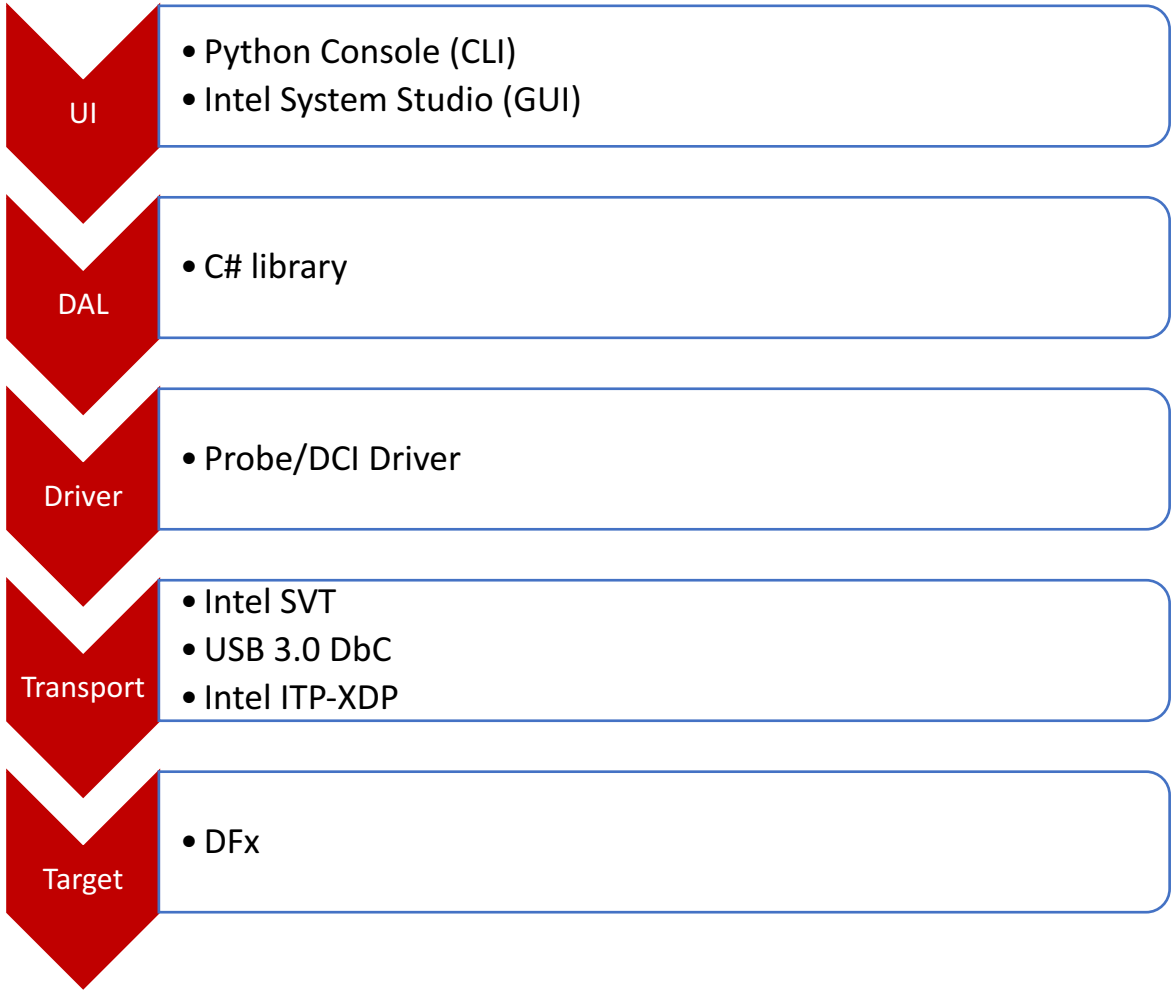
What About Host Side?

DFx Abstraction Layer

Intel DAL: What Is It?

- DAL stands for DFx Abstraction Layer, a software stack for DFx
- DAL is the core of all recent Intel HW debugging/checking tools (System Debugger, System Trace, Platform Debugging Toolkit)
- Supports a wide range of Intel platforms/CPUs
- Supports multiple Intel HW probe types
- **DAL is available without NDA**

Overview of Intel DAL



Sources of Information About DAL

Documentation / White Papers / Patents

See also:

Problem

Trial version of Intel System Studio
doesn’t include configuration options for ME core

Crafting ME Core Configuration

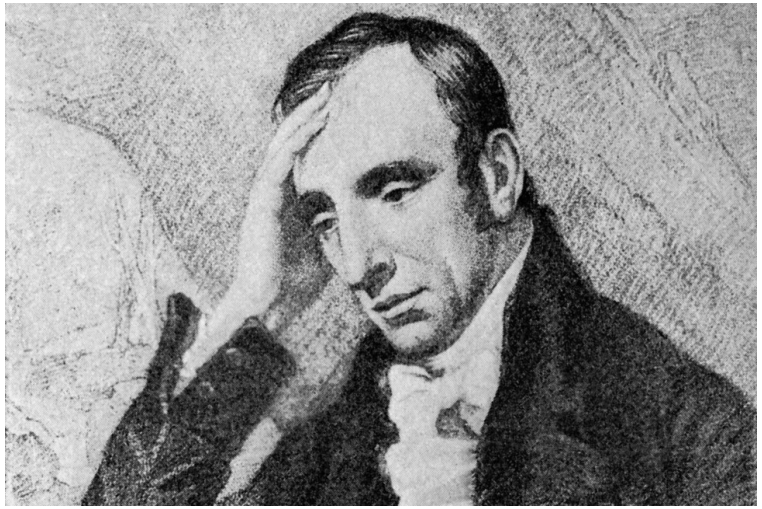
DFx Abstraction Layer

Encrypted XML Files

- DAL configuration is included in encrypted XML files
- Encryption is performed using PBKDF2 and AES
- Key and salt are hardcoded in DAL (*Intel.DAL.Common.Decryption.dll*)

Salt = "I wandered lonely as a cloud,\r\n
That floats on high o'er vales and hills,\r\n
When all at once I saw a crowd,\r\n
A host of golden daffodils "

Key = "ITP"

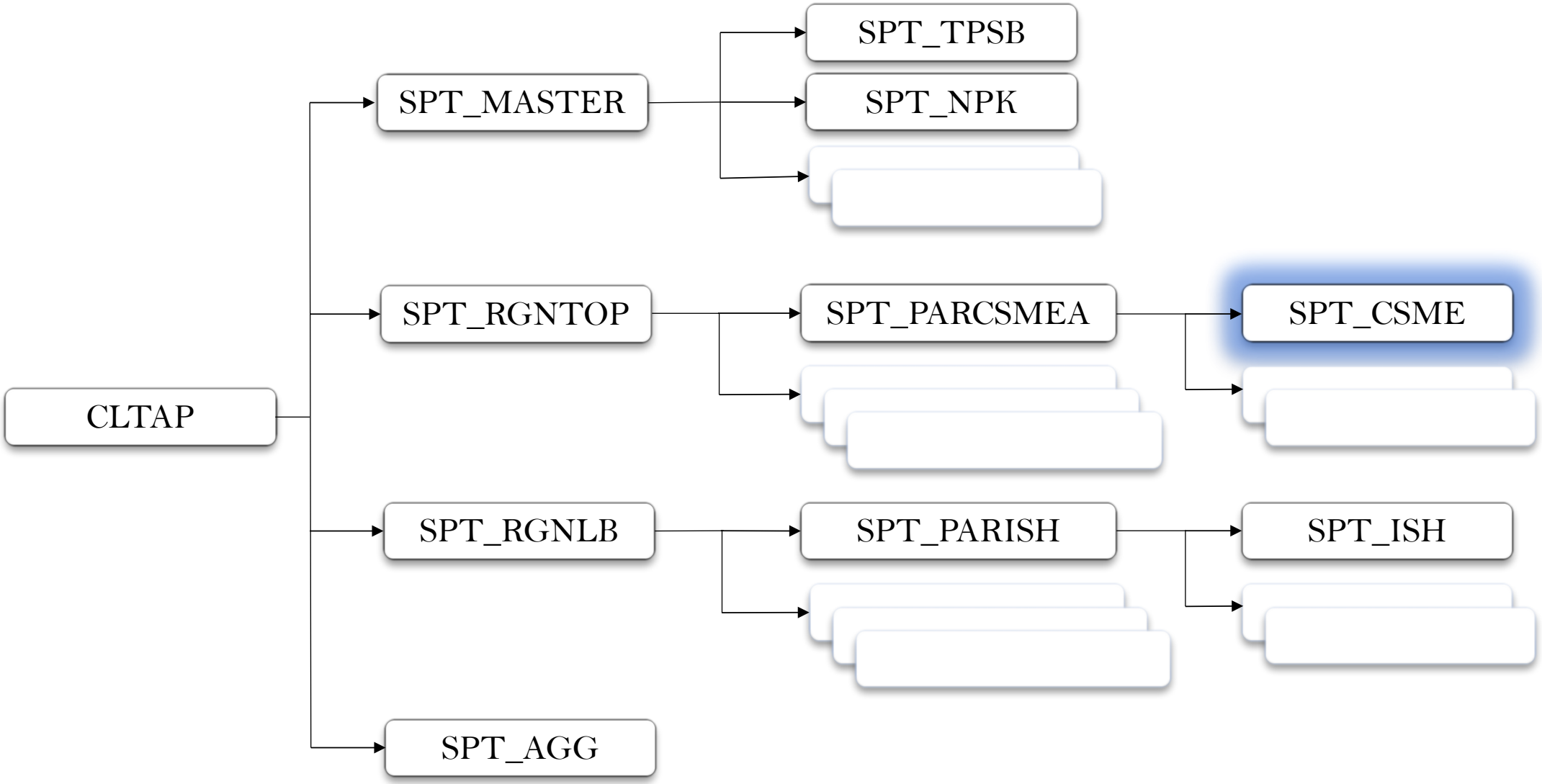


William Wordsworth

ME Core Device Configuration

- Configuration options for ME core are missing in public XML files
- ME core is an LMT2 device (by JTAG ID code)
- LMT2 is included in XML files

DFx Chain to ME LMT2 Core (LP series)



Craft Custom Configuration (for Skylake)

1. Decrypt XML files
2. Add the following lines to "Topo.SPT.xml":

```
<Device Name="SPT_PARCSMEA" LogicalType="CHIPSET" IrLength="8" IdCode="0x00000000" Mask="0x00000000" IsIndependentTap="false" Subtypes="_INHERIT">
<_tag key="Invisible" value="False" />
      <SubDevices>
        <SubDevice Name="LMT2" TapRegister="idcode" Field="idcode" Val="0x1" Mask="0x1" IsLogicalChild="true" SerializePreScan="TapSerializationSTAP0.Serialized"
PhysicalEnable="True" />
        <SubDevice Name="SPT_PARCSMEA_RETIME" TapRegister="idcode" Field="idcode" Val="0x1" Mask="0x1" IsLogicalChild="true"
SerializePreScan="TapSerializationSTAP5.Serialized" PhysicalEnable="True" />
      </SubDevices>
    </Device>
```

3. Use standard DAL environment for ME debugging
4. Make your computer personal again

Demo

Our achievements so far

- JTAG activated for Intel ME
- Starter code (aka ROM) dumped
- Complete Huffman code recovered for ME 11
- Integrity and Confidentiality Platform Keys [FFS17] extracted

Links

- GitHub:

  https://github.com/ptresearch/

- Blogs:

  http://blog.ptsecurity.com/

References

[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.

[IDS17] Mark Ermolov, Maxim Goryachy, Intel DCI Secrets, HITBSecConf 2017 CommSec, Amsterdam, 2017.

[HTH17] Mark Ermolov, Maxim Goryachy, How to Hack a Turned-Off Computer, or Running Unsigned Code in Intel Management Engine, Black Hat Europe, 2017.

[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.

Thank you!
Questions?

Mark Ermolov
Maxim Goryachy

POSITIVE TECHNOLOGIES