# Intel AMT vulnerability. Life after CVE-2017-5689

The intention of this report is not only to show the story of "her majesty" Intel AMT vulnerability, or the CVE-2017-5689. This report describes possible ways and scenarios of exploiting the vulnerability as well. In addition this white paper outlines some new interesting "undocumented features" of Intel ME/AMT that can be used by an attacker. However, this will demonstrate how the capabilities of Intel ME/AMT can be used to their full extent to tinker with the very platform.

#### Beyond the Dark Portal:

Further research revealed, that there are some more things to worry about:

- Firmware (Intel ME/AMT) has security issues. Thus, web interface security weakness can be used to obtain remote access Intel AMT system.
- Hardware (Intel ME/AMT) has undocumented features. An illustrative example to it is MEI (HECI) communication protocol that can be reverse-engineered, thus making it possible for attackers to intercept the data and use it for their own benefit.
- There is a possible New stealth infecting computer system with malware that uses only common Intel AMT SoL capabilities to keep communication stealthy and evade security applications.
- If successfully exploited by attackers Intel ME/AMT capabilities become attackers' capabilities. Thus legit functionality will be used to perform non-legit actions

#### History:

Earlier this year our researchers detected an Intel AMT vulnerability, which was followed by CVE-2017-5689. The fix released by Intel was intended to directly address the corresponding vulnerability analyzed in the scope of the research and described in this white paper. Considering it successfully closed the given vulnerability, Embedi didn't test the fix for side effects it may cause.

According to Shodan, on May 2 there were 6,378 available IPs with Intel AMT. It should be taken into consideration that Shodan's data covers only the Internet, while a good deal of Intel AMTs is used in corporate networks. These corporate AMTs can be easily used by an attacker who has connected to a network.

A little reminder - exploitation is possible: Local (by using the LSM service) and Remote (via the open port)

#### The waves spreading:

It turned out that this vulnerability had affected ICS as well. For instance, Siemens industrial computers based on Intel chipset were susceptible to it. In other words, not only enterprises and public organizations but also critical facilities may be affected by the vulnerability exploitation.

#### Welcome to the jungle:

After the information about the vulnerability had become available to the public, various tools, from scanners and public exploits to automated AMT and honeypots disablers, emerged at once. Community was more than interested.

#### What goes around comes around:

Intel itself didn't stand far from the discussion making INTEL-SA-00075 Detection and Mitigation Tool and INTEL-SA-00075 Mitigation Guide. With official forum for general discussions.

# Introduction to Intel ME/AMT Architecture

Every modern computer system based on Intel architecture has Intel Management Engine (ME) - a built-in subsystem with a wide array of powerful capabilities (such as full access to operating memory, out-of-band access to a network interface, running independently of CPU even when it is in a shutdown state, etc.). On the one hand, these capabilities allow Intel to implement many features and technologies based on Intel ME. On the other hand, it makes Intel ME a tempting target for an attacker. Especially, if an attack can be conducted remotely. Here, Intel Active Management Technology (AMT) fits perfectly – it is based on Intel ME and intended for remote administration of a computer system.

As you surely know, although Intel CPU equipped with various integrated controllers and graphics controller (Fig. 1) is a major execution environment, it's not the only one. There is also a chipset with integrated controllers and subsystems supporting peripheral devices and some system functions. One of such subsystems is the Intel Management Engine (Intel ME). Intel ME is an isolated, stealth and highly powerful (meaning its capabilities) execution environment. UEFI BIOS, Intel ME firmware, and a few other binaries are system firmware, which is stored on common SPI flash memory.

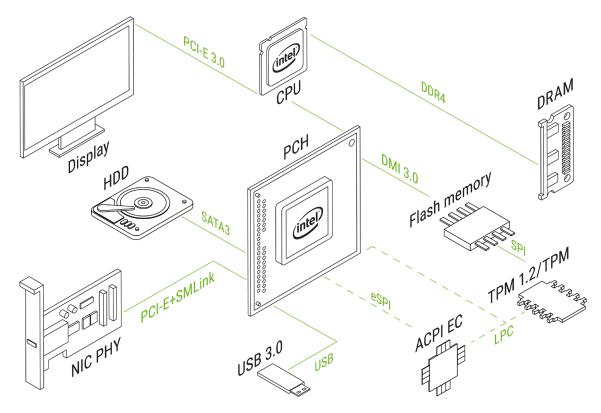


Fig. 1. Typical Intel system architecture

The common SPI flash contents are divided into regions (Fig. 2)

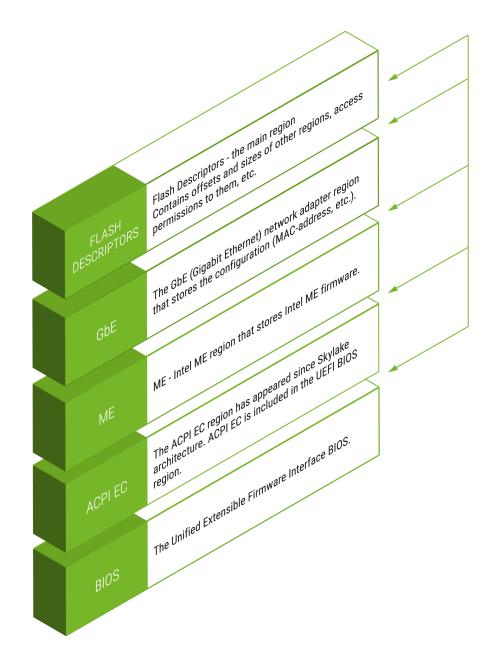


Fig. 2. SPI flash regions

Any program code that is running on a system is executed in one of the CPU protection rings (Fig.3):

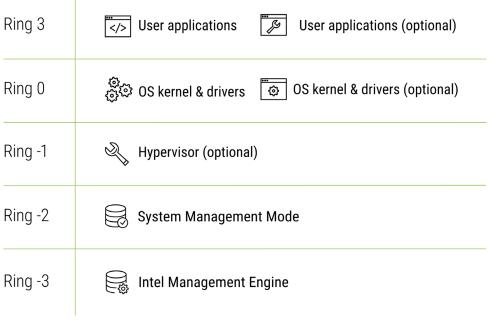


Fig.3 CPU protection rings

**Ring 3** (the user mode) is the highest level with minimal access privileges, needed for user applications.

**Ring 0** (the kernel mode) is the lowest level an OS may have.

**Ring -1** (the hypervisor mode) controls the workspace of multiple guest OSs running in parallel (e.g. to share hardware resources and memory between them).

**Ring -2** (the System Management Mode) is the lowest and most privileged execution mode for the CPU. Its code is located in hidden SMRAM, not visible from any of the above levels.

But it occurs to be that the CPU in the SMM is not the most powerful execution environment on a system. Intel ME subsystem (**Ring -3**) has even more available access capabilities and memory isolation techniques.

# Intel ME/AMT architecture

Intel ME is based on the MCU with ROM and SRAM. The subsystem consists of the following components (Fig.4):

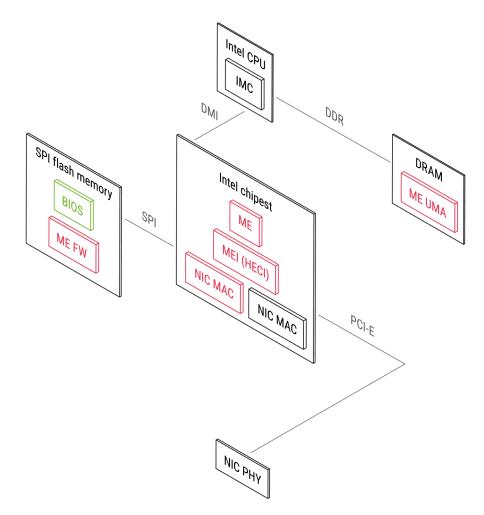


Fig. 4 Intel ME architecture

- 1. Intel chipset's own NIC MAC level controller;
- 2. MEI (HECI) an interface for software to communicate with Intel ME;
- 3. ME UMA a memory block inside DRAM isolated from the main CPU.
- 4. ME FW Intel ME firmware, stored on the SPI flash along with BIOS.

This subsystem is generally thought of as the most privileged and hidden execution environment, which is reasonable enough while it:

- has the ME UMA;
- has full access to the whole DRAM (by using its own DMA engine);
- continues its work as long as a system is plugged in;
- has out-of-band access to the network interface.

Moreover, the only way for communication between software and Intel ME is the Management Engine Interface (the Host Embedded Controller Interface), but the protocol is undocumented.

Intel ME was integrated into Q-type chipsets (960-series and higher) in 2006. Back then it was used in Intel ME 2.x. - 5.x. chipsets. Since 2010 any Intel-based system (series 6.x and higher) is equipped with Intel ME:

- Intel ME 6.x 11.x
- Intel TXE 1.x 3.x
- Intel SPS 1.x 4.x

The table with more details is presented below (see Table 1.)

Table 1. Intel ME/AMT versions in chipsets

PCH:	ME/AMT vershion:
5 series chipset	ME 6.x (AMT 6.x)
6 series chipset:	ME 7.x (AMT 7.x)
7 series chipset	ME 8.x (AMT 8.x)
8 series chipset	ME 9.x (AMT 9.x)
9 series chipset	ME 9.5.x/10.x (AMT 9.5.x/10.x)
100 series chipset	ME 11.0.x (AMT 11.0.x) ME 11.6.x (AMT 11.6.x)
200 series chipset	ME 11.7.x (AMT 11.7.x)

However, the name and firmware implementation are specific to a platform type.

Desktop/laptop – the Intel Management Engine (ME) Server – the Intel Server Platform Services (SPS) Mobile – the Intel Trusted Execution Engine (TXE)

## Intel ME RE problems

Year by year, the interest in researching Intel ME and its firmware is increasing. Nevertheless, it proves to be not an easy task due to the following reasons:

- Intel ME firmware code modules use syslib functions implemented in the bootcode inside the ME ROM. Therefore, it is quite difficult to understand the function behavior without knowing what syslib functions are called from it. Fortunately, ROM images can be found in debug versions of ME firmware (used for the ROM bypass debug capability).
- A certain part of Intel ME firmware code modules is compressed using the Huffman algorithm. But the decompression dictionary is unknown, so the decoding is not a trivial task. Although, still possible: the dictionaries for 6x - 10.x versions were reconstructed.
- One of the best-known problems is the undocumented MEI (HECI) communication protocol. However, the details of this protocol can be reverse-engineered.

The ME UMA memory in DRAM is inaccessible for the main CPU.

 Open-source society created a heading to disable Intel ME subsystem. This is quite a difficult thing to do, but there is a method to restrict the functionality of the subsystem by cutting unnecessary (for system boot up) firmware components.

To reverse engineer the firmware one needs a disassembler and the following scripts:

- me\_unpack.py to parse Intel ME firmware images and extract all partitions/modules;
- me\_heci.py to send commands to Intel ME through HECI;
- dump\_me.py;
- **unhuffme** to unpack Huffman-compressed modules from Intel ME firmware image.

# Intel AMT architecture

# Intel ME firmware components

Intel AMT architecture is a broad and quite a challenging topic to comprehend, moreover it is tightly related to Intel ME firmware. Here, you can find essentials on the latter required to understand the following part of the research.

The code is divided into modules. A precise list varies, depending on a system. Every firmware version contains basic code modules (e.g. bringup module, kernel, etc.) and application modules that implement various Intel technologies (e.g. PTT, AMT, etc.).

The applied technologies define what firmware type is used. There are 3 types of firmware:

- 1. ignition firmware (ME 6.x only) which has minimal size and contents;
- 2. 1.5MB firmware which has incomplete contents of modules;
- 3. 5MB firmware with full firmware contents.

## Getting down to Intel AMT architecture

Putting it in a nutshell, Intel AMT is an application implemented as a module inside Intel ME firmware. The very technology is intended to remotely control and administer computer systems.

With Intel AMT features, it is possible to:

- power up/off, reset a system, access BIOS setup through Serial-Over-LAN (SOL);
- get the information about a system hardware through the web-interface;
- boot a system from the custom boot image/file in recovering purposes;
- acquire the full control of the monitor/keyboard and mouse used in a system.

Any of the above listed features requires a target system to be plugged in and to have the official AMT support (the "vPro" brand). These capabilities do not depend on an OS of a target system at all. Moreover, they can be used to delete or reinstall it.

Applications running locally on the platform communicate with Intel AMT Release 2.0 and later releases the same way network applications do: WS-Management over SOAP over HTTP. When a local application sends a message addressed to the local Intel AMT host name, the Local Manageability Service (LMS), which listens to traffic directed to the host name, intercepts the message and routes it to the Intel Management Engine Interface.

### Access to Intel AMT

Intel AMT has 2 types of interfaces which can be used to access it.

The first type is network interfaces (Intel AMT Releases 2.5, 2.6, 4.0, and 6.0 and later releases support a wireless, along with a wired, network interface). TCP/UDP messages addressed to certain registered ports are routed to Intel AMT when those ports are enabled. Messages received on a wired LAN interface go directly to Intel AMT.

The second one is a local interface. Various local applications are able to access Intel AMT features the same way network applications do – WS-Management over SOAP over HTTP. When a local application sends a message addressed to the local Intel AMT host name, the Local Manageability Service (LMS), which listens to traffic directed to the host name, intercepts the message and routes it to the Intel Management Engine Interface (see Fig. 5.)

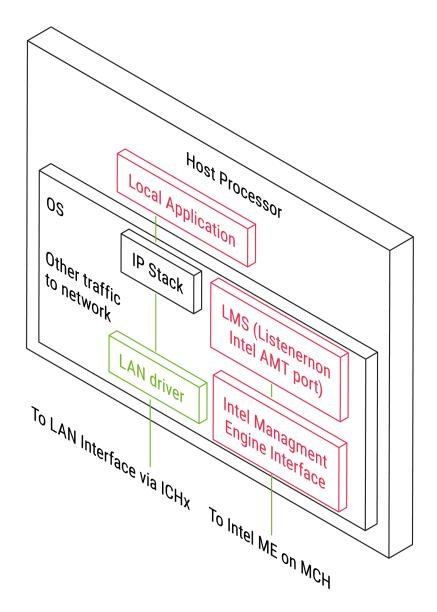


Fig. 5 Local access to Intel AMT

## Intel AMT manageability ports

Getting access to Intel AMT manageability ports provides an attacker with an advantage. You can find the list of widely known ports used by a remote administrator to access the AMT:

5900 - AMT VNC-server without encryption;

16992 - AMT web-server, HTTP protocol;

16993 - AMT web-server, HTTPS protocol;

16994 - AMT redirection for SOL, IDE-R, KVM without encryption;

16995 - AMT redirection for SOL, IDE-R, KVM with TLS.

The access to the Intel AMT Manageability Ports (save the VNC-server at 5900) is protected with the Intel AMT Access Control List (ACL) managed by the system administrator.

The two supported mechanisms of authentication are Digest and Kerberos. Users added to the Intel AMT ACL are either digest or Kerberos users respectively. A notable, and the most interesting, exception from that is the 'admin' user which is present by default and uses the Digest authentication.

### Unauthorized remote access to Intel AMT system

When accessed with a regular web-browser Intel AMT redirects a user to a logon page (Fig. 6) and greets him/her with the "Authentication required" request. By using mitmproxy it is possible to see what is happening right now:

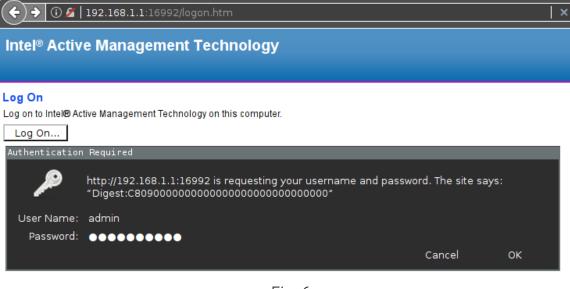


Fig. 6 Intel AMT logon page

In the scope of our research we gave it a closer look and used a proxy server. As for RFC 2617 "Digest Authentication", the first request gets a respond with 401 Unauthorized (see Fig. 7).

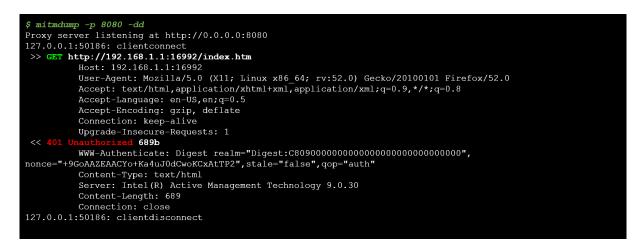


Fig. 7 401 Unauthorized

Given a username and password, the client responds with a new request, including the Authorization header field (Fig. 8) and the server accepts a user as a legitimate one.

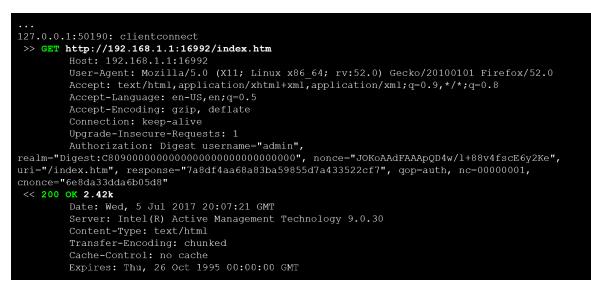


Fig. 8 Server response

At first glance, nothing extraordinary has happened during the authentication process. A closer look, however, gives a clue of what to look for in the ME firmware. To be precise the name of the fields sent in the Authorization Headers. These strings will help us to pinpoint the very code that is responsible for the digest authentication (Fig. 9).

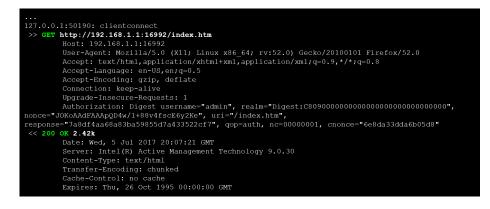
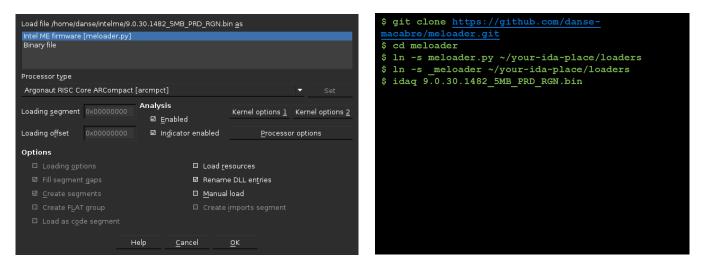


Fig. 9 Authentication field

For the reverse engineering purposes, we use the IDA disassembler and a special loader script (Fig. 10, Fig. 11)







They proved to be useful enough. After the loader finished its work, we got a nice and clean look onto the code and data (Fig. 12).

Name         Start         End         R         W X         D         L         Align         Bas: Type         Class         AD         rVds           ID         JOM_BSS         200DA000         200DB000         ??         ?         ?         L         page         00         public         32         00           ID         VCODTAYLOR_CODE         200DC000         ?         ?         ?         L         page         00         public         32         00           ID         VCODTAYLOR_CODE         200DC000         ?         ?         ?         L         page         00         public         32         00           ID         VCODTAYLOR_CODE         2010C000         ?         ?         ?         L         page         00         public         32         00           ID         WCODTAYLOR_BSS         2014C000         KERNEL_CODE         2019F000         KERNEL_CODE         2019	😝 Program Segmentation	I												
# WCODTAYLOR_KAPI       200DE000       ? ? ? L page 00 public       32 00         # WCODTAYLOR_CODE       200DC000       ? ? ? L page 00 public       32 00         # WCODTAYLOR_CODE       201371F2       TDA View-B         # WCODTAYLOR_DATA       201371F2       Data         # WCODTAYLOR_DATA       201371F2       TDA View-B         # WCODTAYLOR_BSS       2014000       # EFNEL_CODE: 2019F000         # ROMP_CODE       20185400       KERNEL_CODE: 2019F000         # ROMP_CODE       20187000       KERNEL_CODE: 2019F000         # BUP_CODE       20187000       KERNEL_CODE: 2019F000         # BUP_DATA       20196A00       KERNEL_CODE: 2019F004       push         # KERNEL_CODE       2019F000       KERNEL_CODE: 2019F004       push         # KERNEL_CODE       2019F000       KERNEL_CODE: 2019F024       push         # KERNEL_CODE       2019F000       KERNEL_CODE: 2019F024       push         # KERNEL_CODE       2019F000       KERNEL_CODE: 2019F024       push         # KERNEL_CODE       2019F000       KERNEL_CODE: 2019F026       mov       r14, r0         # KERNEL_CODE       2019F000       KERNEL_CODE: 2019F026       mov       r1, r13         # SESSMGRPRIV_CODE       201F6000       KERNEL_CODE	Name	Start	End						Align	Base	Туре	Class	AD	r∨ds
Image: Constant C	😝 JOM_BSS	200DA000	200DB000						page	00	public		32	00
# wcoDTAYLOR_DATA       201371F2         # wcoDTAYLOR_BSS       2014C000         # wcoDTAYLOR_BSS       2014C000         # ROMP_CODE       20185000         KERNEL_CODE: 2019F000       KERNEL_CODE: 2019F000         # ROMP_DATA       20185480         # ROMP_BSS       20186000         # ROMP_BSS       20186000         # ROMP_BSS       20186000         # BUP_CODE       20187000         # BUP_CODE       20187000         # KERNEL_CODE: 2019F002       push         # BUP_DATA       20196ADC         KERNEL_CODE: 2019F004       KERNEL_CODE: 2019F005         # KERNEL_CODE       2019F000         KERNEL_CODE: 2019F005       mov         # KERNEL_DATA       2018F060         KERNEL_CODE       2019F000         KERNEL_CODE: 2019F016       mov         # KERNEL_DATA       2018F060         KERNEL_CODE: 2019F016       mov         # KERNEL_CODE       2019F000         KERNEL_CODE: 2019F016       kERNEL_CODE: 2019F018         # SESSMGRPRIV_CATA       20204000         KERNEL_CODE: 2019F018       pop         # SESSMGRPRIV_DATA       203C0400         KERNEL_CODE: 2019F016       kERNEL_CODE: 2019F016		200DB000	200DC000						page	00	public			00
F:       WCODTAYLOR_BSS       2014C000       KHNLL_CODE: 2019F000       KENNEL_CODE: 2019F000         F:       MCMD_CODE       20185000       KENNEL_CODE: 2019F000       KENNEL_CODE: 2019F000         F:       ROMP_DATA       20185480       KENNEL_CODE: 2019F000       push       r13         F:       BUP_CODE       20187000       KENNEL_CODE: 2019F000       push       r14         F:       BUP_DATA       20196ADC       KENNEL_CODE: 2019F004       push       blink         F:       BUP_DATA       20196ADC       KENNEL_CODE: 2019F006       mov       r14, r0         F:       BUP_DATA       20196ADC       KENNEL_CODE: 2019F008       mov       r14, r0         F:       KENNEL_CODE       2019F000       KENNEL_CODE: 2019F068       mov       r14, r0         F:       KENNEL_CODE       2019F000       KENNEL_CODE: 2019F068       mov       r1, r13         F:       KENNEL_CODE       2019F000       KENNEL_CODE: 2019F016       pop       blink         F:       SESSMGRPRN_CODE       2019F000       KENNEL_CODE: 2019F016       pop       r13         F:       KENNEL_CODE       SESSMGRPRN_CODE       2019F000       KENNEL_CODE: 2019F016       pop       r13         F: <td< td=""><td></td><td>200DC000</td><td>IDA View-B</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></td<>		200DC000	IDA View-B											
P:       WCODTAYLOR BSS       2014C000       KERNEL_CODE: 2019F000       KERNEL_CODE: 2019F000         P:       ROMP_CATA       20185400       KERNEL_CODE: 2019F000       KERNEL_CODE: 2019F000         P:       ROMP_BSS       20186000       KERNEL_CODE: 2019F000       KERNEL_CODE: 2019F000         P:       BUP_CODE       20187000       KERNEL_CODE: 2019F000       FURMEL_CODE: 2019F000         P:       BUP_CODE       20187000       KERNEL_CODE: 2019F000       mov       r14         P:       BUP_CODE       20187000       KERNEL_CODE: 2019F006       mov       r14, r0         P:       BUP_DATA       20196ADC       KERNEL_CODE: 2019F008       mov       r13, r1         P:       BUP_DATA       20196ADC       KERNEL_CODE: 2019F008       mov       r0, r14         P:       KERNEL_CODE       2019F000       KERNEL_CODE: 2019F012       mov       r1, r13         P:       KERNEL_BSS       20196000       KERNEL_CODE: 2019F018       pop       r14         P:       SESSMGRPRIV_KAPI       201F6000       KERNEL_CODE: 2019F018       pop       r14         P:       SESSMGRPRIV_CDATA       2020210       KERNEL_CODE: 2019F012       mov       r14         P:       SESSMGRPRIV_DATA       20202				KERNE	- C	ODE	: 201	9FC	000 # ==			= SUB	ROU	T T N F ================================
PROMP_DATA       20185480       KENNEL_CODE:2019F000       KENNEL_CODE:2019F000         PROMP_BSS       20186000       KENNEL_CODE:2019F002       push       r13         PROMP_BSS       20186000       KENNEL_CODE:2019F002       push       r14         PROMP_BSS       20196000       KENNEL_CODE:2019F004       push       p13         PROMP_BSS       20196000       KENNEL_CODE:2019F004       push       p14         PROMP_BSS       20196000       KENNEL_CODE:2019F006       mov       r14, r0         PROMP_BSS       20196000       KENNEL_CODE:2019F006       mov       r13, r1         KENNEL_CODE       20196000       KENNEL_CODE:2019F006       mov       r0, r14         FX       KENNEL_DATA       201E7000       KENNEL_CODE:2019F016       mov       r14, r0         FX       KENNEL_DATA       201E9000       KENNEL_CODE:2019F016       p00       r13         FX       KENNEL_OATA       201E9000       KENNEL_CODE:2019F016       p00       r14         FX       SESSMGRPRIV_CATA       20204000       KENNEL_CODE:2019F016       p00       r14         F3       SESSMGRPRIV_DATA       20204000       KENNEL_CODE:2019F016       p00       r13         F4       HOTHAM_KAPI				KERNE	L_C	ODE	:201	9FC	000					
iii Norm_ONIX       20180000       KERNEL_CODE:2019F000       push       r13         iii Norm_ONIX       20180000       KERNEL_CODE:2019F002       push       r13         iii Norm_ONIX       20187000       KERNEL_CODE:2019F002       push       r14         iii BUP_DATA       20196A0C       KERNEL_CODE:2019F008       mov       r14, r0         iii BUP_BSS       20194000       KERNEL_CODE:2019F008       mov       r14, r0         iii BUP_BSS       20194000       KERNEL_CODE:2019F008       mov       r14, r0         iii KERNEL_CODE       2019F000       KERNEL_CODE:2019F018       mov       r14, r0         iii KERNEL_CODE       2019F000       KERNEL_CODE:2019F018       mov       r1, r13         iii KERNEL_CODE       2019F000       KERNEL_CODE:2019F012       bl       KERNEL_CODE:2019F08C         iii SESSMGRPRIV_CODE       201F6000       KERNEL_CODE:2019F016       pop       blink         iii SESSMGRPRIV_CODE       201F6000       KERNEL_CODE:2019F016       pop       r14         iii SESSMGRPRIV_CODE       201F6000       KERNEL_CODE:2019F016       pop       r13         iii SESSMGRPRIV_CODE       2020410       KERNEL_CODE:2019F016       pop       r13         iiii HOTHAM_CAPI       203C8000												5000		
Hit ROMP_BSS       20186000       KEFNEL_CODE: 2019F002       push       r14         Hit ROMP_BSS       20196000       KEFNEL_CODE: 2019F004       push       blink         Hit ROMP_BSS       20196000       KEFNEL_CODE: 2019F004       push       blink         Hit ROMP_BSS       20196000       KEFNEL_CODE: 2019F004       mov       r14, r0         Hit RomP_BSS       20196000       KEFNEL_CODE: 2019F004       mov       r13, r1         KERNEL_CODE       2019F000       KEFNEL_CODE: 2019F004       bl       KEFNEL_CODE: 2019F0048         KERNEL_DATA       201E7C60       KEFNEL_CODE: 2019F016       mov       r0, r14         KERNEL_BSS       201E9000       KEFNEL_CODE: 2019F016       mov       r1, r13         KERNEL_CODE       201F6000       KEFNEL_CODE: 2019F016       pop       r14         SESSMGRPRIV_CATA       202020100       KEFNEL_CODE: 2019F016       pop       r14         SESSMGRPRIV_DATA       202020100       KEFNEL_CODE: 2019F016       pop       r14         SESSMGRPRIV_DATA       202020100       KEFNEL_CODE: 2019F016       pop       r13         SESSMGRPRIV_DATA       203C9000       KEFNEL_CODE: 2019F016       mop       r13         HOTHAM_KAPI       203C9000       KEFNEL_CODE										EL_CO	DE_2019		r13	
# BUP_DATA       20196ADC       KENNEL_CODE       2019F004         # BUP_BSS       2019A000       KENNEL_CODE: 2019F008       mov       r13, r1         # KENNEL_CODE       2019F000       KENNEL_CODE: 2019F008       mov       r0, r13, r1         # KENNEL_CODE       2019F000       KENNEL_CODE: 2019F004       bl       KENNEL_CODE: 2019F004         # KENNEL_CODE       2019F000       KENNEL_CODE: 2019F010       mov       r0, r14         # KENNEL_SS       2019F000       KENNEL_CODE: 2019F016       mov       r1, r13         # SESSMGRPRIV_KAPI       2019F000       KENNEL_CODE: 2019F016       pop       r14         # SESSMGRPRIV_CATA       2020210       KENNEL_CODE: 2019F016       pop       r14         # SESSMGRPRIV_DATA       2020210       KENNEL_CODE: 2019F016       pop       r14         # SESSMGRPRIV_DATA       202020100       KENNEL_CODE: 2019F016       pop       r13         # SESSMGRPRIV_DATA       203C000       KENNEL_CODE: 2019F016       mov       mov         # HOTHAM_BASS       203C9000       KENNEL_CODE: 2019F016       mop       r13         # HOTHAM_BASS       203CP000       KENNEL_CODE: 2019F020       #       mop         # HOTHAM_BSS       203CP000       KENNEL_CODE: 2019F020			•											
P       BUP_BSS       2019A000       KEPNEL_CODE: 2019F008       mov       r13, r1         KERNEL_CODE       2019F000       KENNEL_CODE: 2019F0048       kENNEL_CODE: 2019F048         KERNEL_BSS       2019F000       KEPNEL_CODE: 2019F006       mov       r0, r14         KERNEL_BSS       2019F000       KEPNEL_CODE: 2019F012       bl       KEPNEL_CODE: 2019F08C         F       KERNEL_CODE: 2019F010       mov       r1, r13       KEPNEL_CODE: 2019F08C         F       SESSMGRPRIV_KAPI       201F5000       KEPNEL_CODE: 2019F018       pop       p1ink         F       SESSMGRPRIV_CADE       201F6000       KEPNEL_CODE: 2019F018       pop       r13         F       SESSMGRPRIV_BSS       20202410       KEPNEL_CODE: 2019F016       pop       r14         KEPNEL_CODE: 2019F018       pop       r13       p13       p35555MGRPRIV_BSS       20202410       KEPNEL_CODE: 2019F016       p0p       r13         F       SESSMGRPRIV_BSS       20204000       KEPNEL_CODE: 2019F01E #       mov       mov       mov         F       HOTHAM_CAPI       203C8000       KEPNEL_CODE: 2019F01E #       mov       mov       mov         F       HOTHAM_DATA       203CP604       KEPNEL_CODE: 2019F020       KEPNEL_CODE: 2019F020 <td></td>														
BDD_B33       2019F000       KEFNEL_CODE       2019F000         KEFNEL_CODE       2019F000       KEFNEL_CODE       000       r0, r13         KEFNEL_CODE       2019F010       mov       r1, r13         KEFNEL_CODE       2019F016       p00       p13         KEFNEL_CODE       2019F010       mov       r13         SESSMGRPRIV_CODE       20202010       KEFNEL_CODE:2019F012       mop       r13         SESSMGRPRIV_CODE       2036000       KEFNEL_CODE:2019F012       mop       r13         Fill HOTHAM_CAPI       203C0000       KEFNEL_CODE:2019F012       mop       r14         KEFNEL_CODE       203F000       KEFNEL_CODE:2019F012       mop       r13         Fill HOTHAM_CAPI       203C0000       KEFNEL_CODE:2019F012       mop       r13         KEFNEL_CODE       2019F012       KEFNEL_CODE:2019F012       mop       r13         Fill HOTHAM_CAPI       203C9000       KEFNEL_CODE:2019F012       mop       r14         Fill HOTHAM_BSS       203CP600       KEFNEL_CO														
IP       KERNEL_CODE       2019F000       KERNEL_CODE       2019F000       KERNEL_CODE       2019F000         IP       KERNEL_BSS       2019F000       KERNEL_CODE:2019F010       mov       rl, rl3         IP       KERNEL_BSS       2019F000       KERNEL_CODE:2019F012       bl       KERNEL_CODE:2019F03C         IP       SESSMGRPRIV_KAPI       201F5000       KERNEL_CODE:2019F018       pop       blink         IP       SESSMGRPRIV_CAPI       201F5000       KERNEL_CODE:2019F018       pop       rl4         IP       SESSMGRPRIV_DATA       20202410       KERNEL_CODE:2019F016       pop       rl3         IP       SESSMGRPRIV_DATA       203204000       KERNEL_CODE:2019F01E #       nop         IP       HOTHAM_CAPI       203C8000       KERNEL_CODE:2019F01E #       nop         IP       HOTHAM_DATA       203CB000       KERNEL_CODE:2019F01E #       nop         IP       HOTHAM_DATA       203CDE64       KERNEL_CODE:2019F020       KERNEL_CODE:2019F020         IP       HOTHAM_BSS       203CA000       KERNEL_CODE:2019F020       KERNEL_CODE:2019F020         IP       POLICY_KAPI       203EA000       KERNEL_CODE:2019F020       KERNEL_CODE:2019F020			•											
F:       KERNEL_BSS       201E9000       KERNEL_CODE: 2019F012       bl       KERNEL_CODE: 2019F08C         F:       SESSMGRPRIV_KAPI       201F5000       KERNEL_CODE: 2019F018       pop       blink         F:       SESSMGRPRIV_CODE       201F5000       KERNEL_CODE: 2019F018       pop       r14         F:       SESSMGRPRIV_CODE       2016000       KERNEL_CODE: 2019F01C       j       [blink]         F:       SESSMGRPRIV_BSS       20204000       KERNEL_CODE: 2019F01C       j       [blink]         F:       SESSMGRPRIV_BSS       20204000       KERNEL_CODE: 2019F01E       mop       r13         F:       SESSMGRPRIV_BSS       2030900       KERNEL_CODE: 2019F01E       mop       nop         F:       HOTHAM_CODE       20309000       KERNEL_CODE: 2019F01E       mop       nop         F:       HOTHAM_DATA       2030C664       KERNEL_CODE: 2019F020       KERNEL_CODE: 2019F020         F:       HOTHAM_BSS       203CF000       KERNEL_CODE: 2019F020       KERNEL_CODE: 2019F020         F:       POLICY_KAPI       203EA000       KERNEL_CODE: 2019F020       KERNEL_CODE: 2019F020			•											
iii SESSMGRPRIV_KAPI       201F5000       KEPNEL_CODE: 2019F016       pop       blink         iii SESSMGRPRIV_CODE       201F6000       KEPNEL_CODE: 2019F018       pop       r14         iii SESSMGRPRIV_DATA       20202410       KEPNEL_CODE: 2019F01A       pop       r13         iii SESSMGRPRIV_DATA       20202410       KEPNEL_CODE: 2019F01C       j       [blink]         iii SESSMGRPRIV_DATA       20202400       KEPNEL_CODE: 2019F01E       j       [blink]         iii SESSMGRPRIV_BSS       20204000       KEPNEL_CODE: 2019F01E       mop         iii HOTHAM_CAPI       203C8000       KEPNEL_CODE: 2019F01E       mop         iii HOTHAM_DATA       203CDE64       KEPNEL_CODE: 2019F020       KEPNEL_CODE: 2019F020         iii HOTHAM_BSS       203CP000       KEPNEL_CODE: 2019F020       #Emester       SU B R 0 U T I N E         iii POLICY_KAPI       203EA000       KEPNEL_CODE: 2019F020       #Emester       SU B R 0 U T I N E														
H       SESSMGRPRIV_CADE       201F0000       KEFNEL_CODE: 2019F018       pop       r14         F       SESSMGRPRIV_CODE       201F0000       KEFNEL_CODE: 2019F01A       pop       r14         F       SESSMGRPRIV_DATA       2020210       KEFNEL_CODE: 2019F01C       j       j         F       SESSMGRPRIV_BSS       20204000       KEFNEL_CODE: 2019F01C       j       [blink]         F       HOTHAM_KAPI       203C8000       KEFNEL_CODE: 2019F01E       nop         KEFNEL_CODE: 2019F01E       Nop       KEFNEL_CODE: 2019F01E       Nop         F       HOTHAM_CODE       203C9000       KEFNEL_CODE: 2019F01E       Nop         KEFNEL_CODE: 2019F01E       Nop       KEFNEL_CODE: 2019F01E       Nop         KEFNEL_CODE: 2019F020       KEFNEL_CODE: 2019F020       KEFNEL_CODE: 2019F020       KEFNEL_CODE: 2019F020         F       HOTHAM_BSS       203CF600       KEFNEL_CODE: 2019F020       KEFNEL_CODE: 2019F020         F       POLICY_KAPI       203EA000       KEFNEL_CODE: 2019F020       KEFNEL_CODE: 2019F020	-													
12       SESSMGRAPRIV_CODE       20176000       KEPNEL_CODE: 2019F01A       pop       r13         13       SESSMGRAPRIV_DATA       2002410       KEPNEL_CODE: 2019F01C       j       [blink]         13       SESSMGRAPRIV_BSS       20204000       KEPNEL_CODE: 2019F01C       j       [blink]         15       SESSMGRAPRIV_BSS       20204000       KEPNEL_CODE: 2019F01E       mop         15       HOTHAM_CAPI       203C8000       KEPNEL_CODE: 2019F01E       mop         15       HOTHAM_CODE       203C9000       KEPNEL_CODE: 2019F01E       mop         15       HOTHAM_DATA       203CP600       KEPNEL_CODE: 2019F020       KEPNEL_CODE: 2019F020         16       HOTHAM_BSS       203CF000       KEPNEL_CODE: 2019F020       #====================================			•											
# SESSMGRPRIV_BSS       20204000       KEPNEL_CODE:2019F01E #         # HOTHAM_KAPI       203C8000       KEPNEL_CODE:2019F01E #         # HOTHAM_CODE       203C9000       KEPNEL_CODE:2019F01E #         # HOTHAM_CODE       203C9000       KEPNEL_CODE:2019F01E #         # HOTHAM_DATA       203CDE64       KEPNEL_CODE:2019F020         # HOTHAM_BSS       203CF000       KEPNEL_CODE:2019F020         # HOTHAM_BSS       203CF000       KEPNEL_CODE:2019F020         # POLICY_KAPI       203EA000       KEPNEL_CODE:2019F020														
B         DOTHAM_KAPI         203C8000         KENNEL_CODE: 2019F01E         nop           B         HOTHAM_CAPI         203C8000         KENNEL_CODE: 2019F01E # End of function KERNEL_CODE_2019F000           B         HOTHAM_CODE         203C9000         KERNEL_CODE: 2019F01E           B         HOTHAM_DATA         203CP000         KERNEL_CODE: 2019F020           B         HOTHAM_BSS         203CP000         KERNEL_CODE: 2019F020           B         POLICY_KAPI         203EA000         KERNEL_CODE: 2019F020			· ·										[blin	k]
#) HOTHAM_KAPI       203C9000       KEPNEL_CODE: 2019F01E # End of function KEPNEL_CODE_2019F000         #) HOTHAM_CODE       203C9000       KEPNEL_CODE: 2019F01E         #) HOTHAM_DATA       203CDE64       KEPNEL_CODE: 2019F020         #) HOTHAM_BSS       203CF000       KEPNEL_CODE: 2019F020         #) POLICY_KAPI       203EA000       KEPNEL_CODE: 2019F020         # POLICY_KAPI       203EA000       KEPNEL_CODE: 2019F020			•											
IDTHAM_DATA         203CDE64         KEPNEL_CODE: 2019F020           IDTHAM_BSS         203CF000         KEPNEL_CODE: 2019F020 #											functi			2019F000
P HOTHAM_BSS         203CF000         KEPNEL_CODE: 2019F020 # =========================         S U B R O U T I N E =================================														-
POLICY_KAPI     203EA000     KERNEL_CODE: 2019F020       KERNEL_CODE: 2019F020     KERNEL_CODE: 2019F020														Τ.Τ.Ν.Γ
E POLICY_KAPI 203EA000 KERNEL CODE: 2019F020	-													T I N E
KERNEL CODE: 2019F020 KERNEL CODE: 2019F020; # CODE XREF; KERNEL CODE: 2019F08C+AEL				KERNE				9FC						
										EL_CO				# CODE XREF: KERNEL_CODE_2019F08C+AE↓p
POLICY_DATA 20404E66 KEFNEL_CODE:2019F020 D push blink     File     Delucy_DSC 00404E66 KEFNEL_CODE:2019F022 bl KEFNEL_CODE:2019F022												push bl		
POLICY_BSS 20407000 • KERNEL_CODE: 2019E026 Id r2 -KERNEL_DATA 201E9098 # r2 < unk 201E9098 @ 201E			•											
er utilities KAPI 20409000 • KERNEL_CODE: 2019F028 ld r1, =aPreapisemaphor # r1 <- aPreapisemaphor @ 201E80	-													aPreapīsemaphor # rl <- aPreapisēmaphor @ 201E8008
tilities_CODE 2040A000 kEFNEL_CODE: 2019F02A cmp r2, r1												cmp		
Itilities_DATA         204115D0         r =         KERNEL_CODE: 2019F02C         bls         loc_2019F036           Image: trilition_PSS         20412000         KERNEL_CODE: 2019F02E         ld         r0, =KERNEL_BSS 201E9000 # r0 <- unk 201E9000 @ 201E9			<u>, 1</u>											
H Utilities_BSS 20413000 / KERNEL CODE: 2019F030 sub r2, r2, r1 # r2 <- 00000C90	-		•									sub		
P         MCTP_KAPI         20414000         •         KERNEL_CODE: 2019F032         bl         RAPT_memcpy			•									bl	RAPI_	memcpy
Imp 27 of 133         KERNEL CODE: 2019F036         # CODE XREF: KERNEL CODE: 2019F020+C1		20415000								20105	006			# CODE YREE, KERNEL CODE 2010E02010t
Line 27 of 133 # CODE XREF: KERNEL_CODE: 2019F036 Loc_2019F036: # CODE XREF: KERNEL_CODE: 2019F020+CT] KERNEL_CODE: 2019F036 b KERNEL_CODE 2019F020+CT]	Line 27 of 133		<b>9</b> 0							2019	036:	b	KERNE	
KETNEL_CODE:2019F03A #														

Fig. 12 Obtained code and data

A quick search for the strings like "username", "qop" and "cnonce" gave us the exact strings within the NETSTACK module. All these strings are all cross-refed from the one particular function (Fig. 13)

😴 Strings windov	v					
Address	Length	Туре	String			
S NETSTACK	0000000B	С	, cnonce=\"			
S NETSTACK	00000007	с	cnonce			
CONFSTAC		c	McNonce			
	IDA View-B					
	•	NETSTA	CK DATA:2048C56C	aUsername:	.ascii "username"	# DATA XREF: NETSTACK CODE 20431E74+14to
			CK DATA: 2048C56C			# NETSTACK CODE 20431E74+2Eto
			CK_DATA:2048C56C		.byte O	
			CK_DATA:2048C575	aRealm:	.ascii "realm"	# DATA XREF: NETSTACK_CODE_20431E74+2E <sup>1</sup> 0
			CK_DATA:2048C575		.byte O	
			CK_DATA:2048C57B	aNonce:	.ascii " <mark>nonce</mark> "	# DATA XREF: NETSTACK_CODE_20431E74+40to
			CK_DATA: 2048C57B		hut a	# NETSTACK_CODE_20432B90+12↑0
			CK_DATA:2048C57B CK_DATA:2048C581		.byte O .ascii "uri"	# DATA XREF: NETSTACK CODE 20431E74+54to
			CK_DATA:2048C581	a011.	.bvte 0	# DATA AREF. NETSTACK_CODE_20431E/4+3410
			CK DATA: 2048C585	aResponse 0:	.ascii "response"	# DATA XREF: NETSTACK CODE 20431E74+66to
			CK DATA: 2048C585		.byte o	
			CK_DATA:2048C58E		.ascii "qop"	# DATA XREF: NETSTACK_CODE_20431E74+7Eto
			CK_DATA:2048C58E		.byte O	
			CK_DATA:2048C592		.ascii "nc"	# DATA XREF: NETSTACK_CODE_20431E74+92to
			CK_DATA:2048C592		.byte O	·····
			CK_DATA: 2048C595		.ascii "cnonce"	# DATA XREF: NETSTACK_CODE_20431E74+A2to
			CK_DATA: 2048C595		.byte 0 048C59C:.ascii "%8x"	# DATA XREF: NETSTACK CODE 20453BC8+14to
			CK_DATA:2048C59C		0480390astii 36%	# NETSTACK CODE:NETSTACK CODE 204536C8+1410 # NETSTACK CODE:NETSTACK CODE 20453CC810
			CK DATA: 2048C59C		.bvte o	
					048C5A0:.ascii "\b"	# DATA XREF: NETSTACK CODE 20458984+Cto
		NETSTA	CK_DATA:2048C5A0			# NETSTACK_CODE:NETSTACK_CODE_204589BCto
			CK_DATA:2048C5A0		.byte 🛛	
			CK_DATA:2048C5A2		.align 4	
					.bvte l	

Fig. 13 Quick search for "cnonce"

Our research went on with examining NETSTACK\_CODE\_20431E74() subroutine (Fig. 14). This function is responsible for processing Authorization header fields and the overall digest authentication. After the careful examination of the function's code we found an interesting bug. The exact place of that bug is in the final comparison, which is meant to yield whether the authorization was successful by comparing the provided and the computed response. The two values were tested against each other to see if they match, but the actual number of bytes to be tested was taken from the user-provided response, not from the computed one. Thus, in case one provides an empty response string, strncmp() returns zero (as the number of bytes to be compared is zero), and zero means a successful authorization (Fig. 15).

```
NETSTACK CODE:20431ED4
  add r13, sp, 0x7C
  mov
       r0, r17
       r1, r18
  mov
       r2, r14, (aResponse_0 - aUsername) # "response"
  add
       r3, r13, 0x24 # R3 = SP + 0xA0 = &response
  add
       NETSTACK AuthGetValue
  bl
      r0, 0
  cmp
  bne error
NETSTACK CODE:20431FC8
       r1, [sp,0x10C+user response]
  ld
  mov
       r0, r13
                       # computed response
       r2, [sp,0xA4]
                       # response.length
  ld
       RAPI strncmp
  bl
       r0, 0
  cmp
  bne
       error
  mov
       r0, 0
                           # zero means success!
       sp, sp, 0x108
  add
       RAPI 2000DA4
                       # ret
  b
```

Fig. 14 NETSTACK\_CODE\_20431E74() subroutine

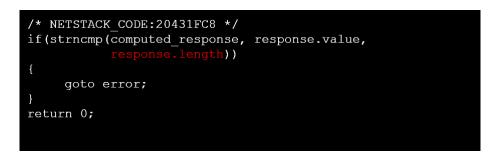


Fig. 15 Accepting empty response as a valid one

CVE-2017-5689

To exploit the newly found vulnerability it is possible to use the mitmproxy tool with a simple script that blanks the "response" field in the Authorization head of the outgoing request (Fig. 16).



Fig. 16 MiTM proxy tool

We configured the web-browser to access the network through the local proxy at 8080. The password we typed in was obviously incorrect, because Intel AMT did not allow passwords shorter than 8 characters. Still, we gave it a try (Fig. 17).

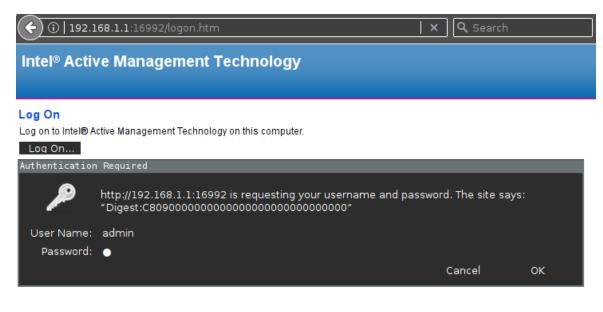


Fig. 17 "Still, we gave it a try."

Just like in the previous case no Authorization header field was sent, so the server responded with 401 Unauthorized (Fig. 18).

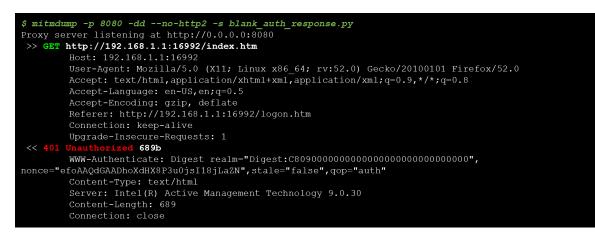


Fig. 18 Server response

Then we got "200 OK", altough the response field was empty (Fig. 19).

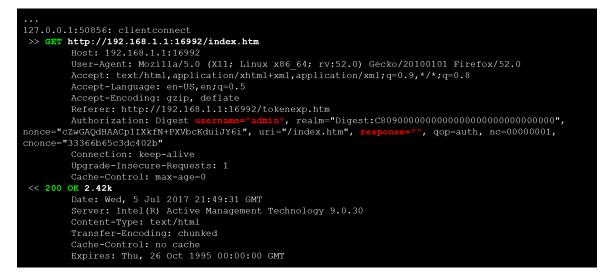


Fig. 19 "200 OK" & empty value

In other words every feature the AMT had became available for an attacker as if the admin password was known (Fig. 20, Fig. 21).

()   192.168.1.1:1	6992/index.htm 🛛 🤁 🗘 😭 🖡 🏠	◙ ≡			
Intel <sup>®</sup> Active Mar <sup>Computer:</sup>	nagement Technology (	(intel)			
System Status Hardware Information System Processor Memory Disk Event Log Remote Control Power Policies Network Settings IPv6 Network Settings System Name Settings User Accounts	System Status           Power         On           IP address         192.168.1.1           IPv6 address         Disabled           System ID         03880288-0488-0588-8706-880700080009           Date         7/5/2017           Time         9:52 pm           Refresh         Copyright © 2005-2013 Intel Corporation. All Rights Reserved. Intel® Active Management Technology firmware version: \$0.30-build 1482				
Fig. 20 AMT's features are available					
System Status Hardware Information System Processor Memory Disk Event Log Remote Control Power Policies Network Settings IPv6 Network Settings User Accounts	Remote Control         Power state: On         Send a command to this computer:         Turn power off*         Turn power off*         Cycle power off and on*         Reset*         *Caution: These commands may cause user application data loss.    Send Command				

Fig. 21 AMT's features are available

Sure thing the bug had been reported and then the vendor offered us to participate in their bug bounty program (Fig. 21, Fig. 22, Fig. 23).

From Intel Product Security Incident Response Team <intel.product.se th="" 📩<=""></intel.product.se>
Subject Intel Announces Bug Bounty Program 03/16/2017 12:10 AM
To Management in the second state of the secon
Maksim, Intel announced a bug bounty program at CanSecWest today in Vancouver B.C. Here is the Intel link and it includes requirements <u>https://security-center.intel.com/BugBountyProgram.aspx</u> . Can you please review and let us know if you'd be interested in participating we could use the AMT vulnerability you discovered as a starting point. Sincerely,
Intel Product Security Incident Response Team <u>www.intel.com/security</u> <u>secure@intel.com</u>
Fig. 21 Intel's bug bounty program

#215598 Intel AMT authentication bypass vulnerability						
State	• Triaged (Open)	Severity	Critical (9.8)			
Reported To	Intel Corporation	Participants	P 🔲 👎			
Scope		Visibility	Private			
Weakness	Improper Authentication - Generic					
Bounty	\$10,000					

Fig. 22 Intel's bug bounty program



Fig. 23 Vulnerability details

#### Exploitation of CVE-2017-5689

There is a vulnerability that allows attackers to log as "admin" user in the AMT.

- The only thing needed is open 16992 port
- No dependence on hardware or OS
- Attackers can use all the Intel AMT capabilities for their own good
- Turned off devices may be attacked as well
- Some systems are accessible through the Internet

Just like in the previous case there are 2 attack methods:

- Local (by using the LSM service)
- Remote (via the open port)

The reasonable question here is, "What an impact can CVE-2017-5689 have?" According to Shodan, on May 2 there were 6,378 available IPs with Intel AMT. However, it should be taken into consideration that Shodan's data covers only the Internet, while a good deal of Intel AMTs is used in corporate networks. These corporate AMTs can be easily used by an attacker who has connected to a network (Fig. 24, Fig. 25).



Fig. 24 Shodan's statistics

Top Countries	
1. United States	2,433
2. Germany	763
3. Canada	566
4. United Kingdom	408
5. Australia	325
6. Russian Federation	289
7. Romania	222
8. Norway	159
9. Korea, Republic of	118
10. Poland	110

Fig. 25 Shodan's statistics

It turned out that this vulnerability affected ICS. For instance, Siemens industrial computers based on Intel chipset were susceptible to it. In other words, not only enterprises and public organizations but also critical facilities may be affected by the vulnerability exploitation.

After the information about the vulnerability had become available to the public, various tools, from scanners and public exploits to automated AMT and honeypots disablers, emerged at once.

## Possible attack scenarios

One should get used to the idea that attackers' possibilities and Intel AMT capabilities are the same thing. Specifically, they can use Intel AMT legitimate functionality to achieve their malicious purposes.

To exploit the vulnerability, an attacker may stick to the FW downgrade scenarios:

1. just swap current firmware blob with the older one

the experiment: swap the FW 11.0.25.3001 with the FW 11.0.24.1000

the result: doesn't work if the SVN of the firmware was incremented

2. change just one code module from the FW blob

the experiment: replace the FW 11.0.25.3001 -> nfc code module with the FW 11.0.24.1000 -> nfc code module

## Spreading out coverage. Part I

It begs the question, "Is it possible for attackers to spread out the coverage of this vulnerability and Intel AMT to achieve their own goals?" How exactly the vPro-system differed from the non-vPro one? We checked that non-vPro and vPro system might have absolutely similar Intel ME firmware images, so Intel AMT implementation was often present on a non-vPro system. Maybe there were hardware restrictions?

The only difference is the MEBx module that used HECI to configure Intel AMT. The HECI interface is a registers set in PCI CFG and MMIO. The messages should be sent through the circular buffer in MMIO.

The messages should contain a command. After the message is sent to Intel Me, the acknowledge message (with the completion status) is to be responded.

The message protocol itself is based on DCMI-HI protocol. There are clients (code modules) that use HECI inside Intel ME firmware. To connect them you need to know the GUID of the client. Here are known GUIDS:

ICC	42b3ce2f-bd9f-485a-96ae-26406230b1ff
MKHI	8e6a6715-9abc-4043-88ef-9e39c6f63e0
LMS	3d98d9b7-1ce8-4252-b337-2eff106ef29f
AMTHI	12f80028-b4b7-4b2d-aca8-46e0ff65814c

To send the message through the HECI an attacker connects to the client using the GUID. The format of the messages (with commands) varies depending on the HECI client you are communicating with. So, for the AMTHI client the following format is used:

```
struct
{
unsigned int groupID; // the AMTHI client code, 0x12
unsigned int command; // command code
unsigned int isResponse;
unsigned int reserved;
unsigned int result)
};
```

The messages have the groupID field, meaning the command group identifier (each group have a specific set of commands). To configure the AMT the 0x12 groupID should be used.

The following commands must be sent one-by-one (after the sending the message, an attacker receives the acknowledge) to configure the AMT:

AMT_INIT	groupID 0x12	command 0x05	ack 0x85
AMT_SET_PWD	groupID 0x12	command 0x09	ack 0x89
AMT_SET_IVP4	groupID 0x12	command 0x0C	ack 0x8C

Intel MEI can also be used to retrieve some data or check the state of Intel ME subsystem (Fig. 26):

- FWSTATUS registers;
- Status request to MKHI;
- Intel PT

t 3 Reserved	0x00000000		
Autotest	Enabled		
ink Status	Enabled		
alized Language	English		
ependent Firmware Recovery	Disabled		
Group ID	0xF85		
Public Key Hash FPF	Not set		
Public Key Hash ME	000000000000000000000000000000000000000	000000000000000000000000000000000000000	
SVN FPF	0x0		
SVN FPF	0x0		
1 SVN FPF	0x0		
Encryption Key FPF	Not set		
Encryption Key ME	000000000000000000000000000000000000000	000000000000000000000000000000000000000	
	FPF	ME	
ce Boot Guard ACM	Not set	Disabled	
tect BIOS Environment	Not set	Disabled	
Debugging Initialization	Not set	Enabled Enabled	
sured Boot	Not set Not set	Disabled	
ified Boot	Not set	Disabled	
Manifest ID	Not set	0x0	
proment Policy	Not set	0×0	
or cementer Politey		Enabled	
Revoke State		LINDZEG	
Revoke State RTC Clear Detection FPF	Not set Not Revoked Not set	Enabled	

Fig. 26 Intel ME subsystem state check

As a PoC we have created a tool to activate the AMT on vPro and non-vPro systems. To do this:

- Run the "AMTactivator" on OS
- Configure the KVM feature of Intel AMT (with a random generated password)
- Remote pwn (password is unknown for anybody, but we don't need one)

The activator consists of the following program components:

AMTactivator:

- 1. mei.sys 32-bit kernel driver to work with MEI;
- 2. mei64.sys 64-bit kernel driver to work with MEI;
- 3. AMTactivator.exe the application.

The workflow:

- 1. Find the MEI device in the PCI CFG and get the base address if the MEI MMIO;
- 2. Use the MEI MMIO to send activation/configuration commands to Intel ME.

The AMT activator was tested on the following systems:

Intel ME version	System and chipset	CPU
7	Intel DQ67SW (vPro), Intel Q67	Intel Core i7-2600 (vPro)
8	Gigabyte GA-H77-D3H (non-vPro), Intel H77	Intel Core i7-3770 (vPro)
9	Gigabyte GA-Q87N (vPro), Intel Q87	Intel Core i3-4300 (without Intel vPro); Intel Core i5-4590 (with Intel vPro)
	Gigabyte GA-H97-D3H (non-vPro), Intel H97	Intel Core i5-4590 (with Intel vPro)

As for a malicious code using Intel AMT, it is worth to be mentioned that:

- First, LegbaCore researchers have already come up with this idea in their research.

- Second, quite recently PLATINUM, a malicious program, has been detected in the ITW. PLATINUM uses Intel AMT SoL to secretly communicate with the CNC.

In both cases, the malware does not use a vulnerability in Intel AMT. Contrary to it, the malware uses only common Intel AMT SoL capabilities to keep communication stealthy and evade security applications.

As Microsoft stated: "This channel works independently of the operating system (OS), rendering any communication over it invisible to firewall and network monitoring applications running on the host device"

So, the combination of the kind is no fiction, but fact. So, it is conceivable enough that it would occur more frequently.

#### **Mitigations**

To protect from malware that is remotely using Intel AMT of your system, you should:

- periodically check that Intel AMT is disabled;
- use a firewall to block any external requests to Intel AMT known network ports.

#### Limitations of AMTactivator

In the scope of our research we have experimentally found out what are the limitations AMTactivator:

1. Only 6 - 9 Intel desktop chipset series can be activated by AMTactivator (on 100/200 series chipsets it is not yet achieved).

2. Intel AMT configures to Standard Manageability mode (without the KVM feature) if your CPU is non-vPro.

3. Intel AMT activation is possible on the systems with Intel ME 5MB firmware (1,5MB firmwares do not have such functionality)

## Spreading out coverage. Part II

Is it possible to swap the 1.5MB FW to 5MB FW to add the absent Intel AMT implementation to a system? An obvious limitation here is the fact the new FW should fit the SPI flash size.

Systems with 6 - 9 series chipsets: system won't boot (resets during the early phases of boot process).

Systems with 100 series chipsets: system boots (but currently we haven't achieved the activation to check if added functionality is working.

One of the greatest challenges for an attacker trying to hide the usage of a remote connection to AMT-enabled is a blinking color frame on the screen. To delete it an attacker uses the VCP DDC/CI commands to change the visible space on the screen: ???forcedly change the resolution of the screen: 1920x1080 -> 1930->1090???

By using Intel AMT an attacker can perform the following actions:

- 1. Exploit CVE-2017-5689 if a system uses an outdated Intel AMT.
- 2. Downgrade a system even if an up-to-date Intel AMT is used.
- 3. Use ActivatorAMT a system does not use Intel AMT.
- 4. Add Intel AMT if there is no Intel AMT in a system

#### Takeaways

- 1. An attacker can do everything a system can.
- 2. Ring-3 firmware (Intel ME/AMT) has security issues.
- 3. Ring-3 hardware (Intel ME/AMT) has undocumented features.
- 4. New stealth infecting technique of computer system.
- 5. Legit functionality for non-legit actions.