VMware ESXi Use-After-Free / Out-Of-Bounds Access

Authored by Google Security Research, Cfir Cohen

Posted Jul 17, 2020

Several security issues have been identified in the VMware ESIx virtual machine monitor (VMM). A use-after-free (UAF) vulnerability in PVNVRAM, a missing return value check in EHCI USB controller leading to private heap information disclosure, and several out-of-bounds reads.

tags | advisory, info disclosure
advisories | CVE-2020-3960, CVE-2020-3963, CVE-2020-3964, CVE-2020-3965
SHA-256 | 9736a651dce3d31a53e949929fa5e638854317668ea1eefa6f0e52872f79d3a2 Download | Favorite | View

Related Files

Share This

Like Twee LinkedIn Reddit Digg StumbleUpon

Change Mirror Download

```
Overview
=======
We identified several security issues in the ESIx virtual machine
monitor (VMM): a use-after-free (UAF) vulnerability in PVNVRAM, a
missing return value check in EHCI USB controller leading to private
heap information disclosure, and several OOB reads.

All issues have been fixed by the vendor. Links to the patches are
provided below.

ESXi PVNVRAM Use After Free [CVE-2020-3963]
=========================================
The paravirtualized NVRAM device supports read / write / find-next
operations on variables stored in its variables store.

The find-next operation (opcode 0xD2) allows guest firmware to
enumerate all variables in the store by querying for the next variable
in the store. PVNVRAM stores a raw pointer to the last returned
variable. In most places that update the variable store, this pointer
is properly cleared, however, in the write operation (opcode 0xD3),
there's a flow that updates / deletes an existing variable, where this
last_search_value pointer is not cleared.

This leads to a situation where the dangling pointer is used in
subsequent find-next operations. We were able to trigger this UAF from
the guest, and confirmed (using gdb) that the dangling pointer is
indeed used after free.

https://www.vmware.com/security/advisories/VMSA-2020-0015.html

ESXi EHCI qTD data leak [CVE-2020-3964]
=================================
The EHCI USB controller processes queue element transfer descriptors
(qTD), as described in section 3.5 of the EHCI specification. We found
that the implementation in this case processes guest-controlled qTDs.

Each descriptor has up to 5 buffer pointers that together hold the USB
request block (URB):

+---------Queue Element Transfer Descriptor Block    +
|                Next qTD pointer           | 0 | T |
|        Alternate Next qTD Pointer         | 0 | T |
| Total Bytes to Transfer | C_Page | Cerr |  Status  |
| Buffer Pointer (page 0)          |  Current Offset |
| Buffer Pointer (page 1)          |     Reserved    |
| Buffer Pointer (page 2)          |     Reserved    |
| Buffer Pointer (page 3)          |     Reserved    |
| Buffer Pointer (page 4)          |     Reserved    |
+---------------------------------------------------+

The function EhciReadTDBuffer() (name identified from log string)
reads the URB contents into a heap allocated buffer. Unfortunately,
the return value of ReadBytes is not checked. A guest can cause this
function to fail by passing a GPA value of zero (or, in the 64bit
addressing case, a non-canonical address >= 0x8000'0000'0000). This
leads to a case where an attached USB device processes a URB with
uninitialized heap data.

We successfully exploited this to leak VMM heap data by sending a SCSI
WRITE command to a USB mass storage device.

Writes to a USB mass storage device are encoded in two qTDs: the first
holds the SCSI WRITE header, and the second holds the data to be
written.

For example, the following operations in the guest:

$ perl -e "print 'a'x2000;" > aaaa
$ sudo dd if=aaaa of=/dev/sdb1 bs=512 count=8

Result in the following qTDs:

#
# First qTD of size 0x1f:
#  "USBC" signature is the CBW packet header set by
#  usb_stor_Bulk_transport() in drivers/usb/storage/transport.c.
#
#  0x2A is SCSI WRITE (10) command in the CDB buffer.
#  0x08 is the transfer length in sectors (8 * 0x200 = 0x1000).
#
Thread 1 hit Breakpoint 1, xxxx in ?? ()
EhciReadTDBuffer, buffer pointer 1 = 34a89000, size = 1f

Thread 1 hit Breakpoint 2, xxxx in ?? ()
=>
0x65f38a8:  0x55    0x53    0x42    0x43    0x5f    0x03    0x00    0x00
0x65f38b0:  0x00    0x10    0x00    0x00    0x00    0x00    0x0a    0x2a
0x65f38b8:  0x00    0x00    0x00    0x00    0x20    0x00    0x00    0x08
0x65f38c0:  0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00

#
# Second qTD of size 0x1000 holds the "aaaaaa" data:
#
Thread 1 hit Breakpoint 1, xxxx in ?? ()
EhciReadTDBuffer, buffer pointer 1 = 33a82000, size = 1000

Thread 1 hit Breakpoint 2, xxxx in ?? ()
=>
0x65e1728:  0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x65e1730:  0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x65e1738:  0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x65e1740:  0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61

By failing ReadBytes() in the second qTD, we write previous heap data
to the disk. We verified VUsb_NewUrb() mallocs the URB buffer, and
doesn't memset the data buffer to zeros. We confirmed that by reading
back the disk contents, the hypervisor was leaking uninitialized heap
data.

https://www.vmware.com/security/advisories/VMSA-2020-0015.html


ESXi XHCI OOB read access [CVE-2020-3965]
=================================
XHCI USB controller reads the DCBs from the guest by mapping a guest
page and iterating over values in it based on a bit field value.

There was insufficient validation on the bit field value: the map size
may be out of sync with the loop counter. A guest can supply a size
value of 0x40, with a bit field value of 0xffffffff. This causes the
```


File Archive: December 2022 <

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Top Authors In Last 30 Days

Red Hat 150 files
Ubuntu 68 files
LiquidWorm 23 files
Debian 16 files
malvuln 11 files
nu11secur1ty 11 files
Gentoo 9 files
Google Security Research 6 files
Julien Ahrens 4 files
T. Weber 4 files

File Tags

ActiveX (932)
Advisory (79,754)
Arbitrary (15,694)
BBS (2,859)
Bypass (1,619)
CGI (1,018)
Code Execution (6,926)
Conference (673)
Cracker (840)
CSRF (3,290)
DoS (22,602)
Encryption (2,349)
Exploit (50,359)
File Inclusion (4,165)
File Upload (946)
Firewall (821)
Info Disclosure (2,660)
Intrusion Detection (867)
Java (2,899)
JavaScript (821)
Kernel (6,291)
Local (14,201)
Magazine (586)
Overflow (12,419)
Perl (1,418)
PHP (5,093)
Proof of Concept (2,291)
Protocol (3,435)
Python (1,467)
Remote (30,044)
Root (3,504)
Ruby (594)
Scanner (1,631)
Security Tool (7,777)
Shell (3,103)
Shellcode (1,204)
Sniffer (886)

File Archives

December 2022
November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
Older

Systems

AIX (426)
Apple (1,926)
BSD (370)
CentOS (55)
Cisco (1,917)
Debian (6,634)
Fedora (1,690)
FreeBSD (1,242)
Gentoo (4,272)
HPUX (878)
iOS (330)
iPhone (108)
IRIX (220)
Juniper (67)
Linux (44,315)
Mac OS X (684)
Mandriva (3,105)
NetBSD (255)
OpenBSD (479)
RedHat (12,469)
Slackware (941)
Solaris (1,607)

```
loop to copy 32 elements out of the mapped page, which is outside the
bounds of the mapped region

https://www.vmware.com/security/advisories/VMSA-2020-0015.html

ESXi NVME OOB read access [CVE-2020-3960]
=========================================
The NVME controller does not properly handle namespace 0 in the
nvme_admin_identify handler. NSID of 0 minus one underflows and leads
to an OOB read access.

https://www.vmware.com/security/advisories/VMSA-2020-0012.html


Credits
===========
These vulnerabilities were discovered and reported to VMware by Cfir
Cohen of the Google Cloud security team.


Timeline
========

4-20 - Vulnerabilities disclosed to VMware security team
4-26 - Vendor confirms the issues
6-09 - VMSA-2020-0012 fixes NVME issue
6-23 - VMSA-2020-0015 fixes PVNVRAM, EHCI, XHCI issues
7-16 - Public disclosure

We'd like to thank the VMware security team for their prompt response.
```