ENHANCE VIRTUALIZATION STACK WITH INTEL CET AND MPX

Xiaoning Li
Chief Security Architect
Alibaba Cloud

Ravi Sahita
Principal Engineer
Intel Corporation
Agenda

- Full Virtualization Stack
- QEMU Vulnerabilities
- Intel CET
- Intel MPX and PKU
- VM Escape Case Study
- Mitigation with CET/MPX
- Other Mitigations
Full Virtualization Stack

- **Dom0/Host OS**
  - CPU (VT-x)
  - Hypervisor
  - Guest OS
    - Device Driver
    - Pass-through Device Driver

- **HVM Guest OS with Device Pass-through**

- **Dom0/Host OS**
  - CPU (VT-x)
  - Hypervisor
  - Guest OS
    - Device Emulation
    - Device Driver

- **HVM Guest OS Without Device Pass-through**

- **Dom0/Host OS**
  - CPU (VT-x)
  - Hypervisor
  - Guest OS
    - Frontend Driver/VirtIO
    - Device Driver
Why Device Emulation?

- Supports more guest devices with virtual devices
  - If physical devices number are not enough - limited GPU resource
  - If physical devices don’t support device virtualization – Not every device support SRIOV
  - If physical devices don’t exist – some outdated devices
- Popular usage in cloud environment to support many VMs
- QEMU can provide device emulation for KVM/XEN, but it brings new attack surface on virtualization stack
QEMU VULNERABILITIES
QEMU Vulnerabilities by 2018 Jan

QEMU Vulnerability Numbers by Year

<table>
<thead>
<tr>
<th>CVE Year</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>1</td>
</tr>
<tr>
<td>2008</td>
<td>6</td>
</tr>
<tr>
<td>2009</td>
<td>1</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
</tr>
<tr>
<td>2011</td>
<td>7</td>
</tr>
<tr>
<td>2012</td>
<td>3</td>
</tr>
<tr>
<td>2013</td>
<td>23</td>
</tr>
<tr>
<td>2014</td>
<td>20</td>
</tr>
<tr>
<td>2015</td>
<td>29</td>
</tr>
<tr>
<td>2016</td>
<td>86</td>
</tr>
<tr>
<td>2017</td>
<td>49</td>
</tr>
</tbody>
</table>
```c
if (proto == ETH_P_IP)
{
    DPRINTF("+++ C+ mode has IP packet\n");

    /* not aligned */
    eth_payload_data = saved_buffer + ETH_HLEN;
    eth_payload_len = saved_size - ETH_HLEN;

    ip = (ip_header*)eth_payload_data;

    if (IP_HEADER_VERSION(ip) != IP_HEADER_VERSION_4) {
        DPRINTF("+++ C+ mode packet has bad IP version %d \\
            "expected %d\n", IP_HEADER_VERSION(ip),
            IP_HEADER_VERSION_4);
        ip = NULL;
    } else {
        hlen = IP_HEADER_LENGTH(ip);
        ip_protocol = ip->ip_p;
        ip_data_len = be16_to_cpu(ip->ip_len) - hlen;
    }
} « end if proto==ETH_P_IP »
```

```c
struct ip_header {
    uint8_t ip_ver_len;     /* version and header length */
    uint8_t ip_tos;         /* type of service */
    uint16_t ip_len;        /* total length */
    uint16_t ip_id;         /* identification */
    uint16_t ip_off;        /* fragment offset field */
    uint8_t ip_ttl;         /* time to live */
    uint8_t ip_p;           /* protocol */
    uint16_t ip_sum;        /* checksum */
    uint32_t ip_src, ip_dst; /* source and destination address */
};

#define IP_HEADER_LENGTH(ip) ((ip->ip_len)&0xf) << 2)
```

```c
/* ETH_MTU = ip header len + tcp header len + payload */
int tcp_data_len = ip_data_len - tcp_hlen;
int tcp_chunk_size = ETH_MTU - hlen - tcp_hlen;
```
CVE-2015-7504 – Heap Overflow

```c
} else if (s->looptest == PCNET_LOOPTEST_CRC || !CSR_DXMTFCS(s) || size < MIN_BUF_SIZE + 4) {
    uint32_t fcs = ~0;
    uint8_t *p = src;

    while (p != &src[size])
        CRC(fcs, *p++);
    *(uint32_t *)p = htonl(fcs);
    size += 4;
} else {
    uint32_t fcs = ~0;
    uint8_t *p = src;

    while (p != &src[size-4])
        CRC(fcs, *p++);
    crc_err = (*(uint32_t *)p != htonl(fcs));
```
INTEL CET
Intel® Control-flow Enforcement Technology

Intel CET = Shadow Stack + Indirect Branch Tracking
Shadow Stack

Return Address 4
Return Address 3
Parameter
Parameter
Return Address 2
Parameter
Return Address 1

RET

Return Address 4
Return Address 3
Return Address 2
Return Address 1
Shadow Stack Control Protection Exception

Return Address 4
Return Address 3
Parameter
Return Address 2
Parameter
Return Address 1

Return Address 4
Return Address 3
Return Address 2.1
Return Address 1

#CP
RET
Indirect Branch Tracking

- IND JMP
- IND CALL

ENDBR32/ENDBR64

- Instruction1
- Instruction2
- Instruction...

Indirect Branch

- Instruction...
- RET
Indirect Branch Tracking

IND JMP
IND CALL

ENDBR32/ENDBR64
Instruction1
Instruction2
Instruction...
Indirect Branch
Instruction...
RET

ENDBR32/ENDBR64
Instruction...
RET
Indirect Branch Tracking #CP

- IND JMP
- IND CALL
- ENDBR32/ENDBR64
  - Instruction1
  - Instruction2
  - Instruction...
  - Indirect Branch
  - Instruction...
  - RET

- Push Eax
- Instruction...
- RET

#CP
Cross Mode Indirect Branch Tracking

Same binary built by compiler

32bit Mode
- ENDBR32
- Instruction1 32
- Instruction2 32

64bit Mode
- ENDBR32
- Instruction1 64
- Instruction2 64

Opcode | Instruction
--- | ---
F3 0F 1E FA | ENDBR64
F3 0F 1E FB | ENDBR32
INTEL MPX
Intel® Memory Protection Extensions

MPX = Bound Table + Bound Check ISA
Figure 17-4. Bound Paging Structure and Address Translation in 64-Bit Mode
## Bound Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNDMK</td>
<td>Create a LowerBound and a UpperBound in a register.</td>
</tr>
<tr>
<td>BNDCL</td>
<td>Check the address of a memory reference against a LowerBound.</td>
</tr>
<tr>
<td>BNDCU</td>
<td>Check the address of a memory reference against an UpperBound in 1’s compliment form.</td>
</tr>
<tr>
<td>BNDCN</td>
<td>Check the address of a memory reference against an UpperBound not in 1’s compliment form.</td>
</tr>
<tr>
<td>BNDMOV</td>
<td>Copy or load from memory of the LowerBound and UpperBound to a register.</td>
</tr>
<tr>
<td>BNDMOV</td>
<td>Store to memory of the LowerBound and UpperBound from a register.</td>
</tr>
<tr>
<td>BNDLDX</td>
<td>Load bounds using address translation.</td>
</tr>
<tr>
<td>BNDSTX</td>
<td>Store bounds using address translation.</td>
</tr>
</tbody>
</table>
QEMU VM ESCAPE CASE STUDY
Escape From The Docker-KVM-QEMU Machine

Shengping Wang, Xu Liu
Qihoo 360 Marvel Team

#HITB2016AMS D1T1 - Escape From The Docker KVM QEMU Machine - Shengping Wang and Xu Liu
CVE-2015-5165 Memory Disclosure

Host

QEMU

Vulnerable Code

Host Memory Data

Guest OS

Malformed Package

Malformed Package

Host Memory Data

QEMU Memory Address
Guest OS Memory Address
CVE-2015-7504 Code Execution

Host

QEMU

Vulnerable Code

Critical Pointer

Guest OS

Malformed Package
CVE-2015-7504 Code Execution

```c
struct PCNetState_st {
    NICState *nic;
    NICConf conf;
    QEMUTimer *poll_timer;
    int rap, isr, lnkst;
    uint32_t rdra, tdra;
    uint8_t prom[16];
    uint16_t csr[128];
    uint16_t bcr[32];
    int xmit_pos;
    uint64_t timer;
    MemoryRegion mmio;
    uint8_t buffer[4096];
    qemu_irq irq;
    void (*phys_mem_read)(void *dmaOpaque, hwaddr addr,
        uint8_t *buf, int len, int do_swap);
    void (*phys_mem_write)(void *dmaOpaque, hwaddr addr,
        uint8_t *buf, int len, int do_swap);
    void *dmaOpaque;
    int tx_busy;
    int looptest;
} « end PCNetState_st »;
```

```c
struct IRQState {
    Object parent_obj;
    qemu_irq_handler handler;
    void *opaque;
    int n;
};

void qemu_set_irq(qemu_irq irq, int level) {
    if (!irq)
        return;
    irq -> handler(irq -> opaque, irq -> n, level);
}
```
void qemu_set_irq(qemu_irq irq, int level)
{
    if (!irq)
        return;
    irq->handler(irq->opaque, irq->n, level);
}
CVE-2015-7504 Exploit in HITB

```c
void qemu_set_irq(qemu_irq irq, int level)
{
    if (!irq)
        return;

    irq->handler(irq->opaque, irq->n, level);
}
```

```
xchg rax, rsp;
ret
```
CVE-2015-7504 Exploit in Phrack

```c
void qemu_set_irq(qemu_irq irq, int level)
{
    if (!irq)
        return;
    irq->handler(irq->opaque, irq->n, level);
}
```

qemu_set_irq
Mprotect

shellcode
MITIGATION WITH CET/MPX
Enable MPX on packet memory access
Enable MPX on buffer[4096] memory access
Shadow Stack can stop "xchg rax,rsp;ret"
Indirect Branch Tracking can stop irq->handler calling qemu_set_irq without valid tag
OTHER MITIGATIONS
Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise

### Figure 2-9. Protection Key Rights Register for User Pages (PKRU)

<table>
<thead>
<tr>
<th>RDPKRU—Read Protection Key Rights for User Pages</th>
<th>WRPKRU—Write Data to User Page Key Register</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Opcode</strong></td>
<td><strong>Instruction</strong></td>
</tr>
<tr>
<td>NP OF 01 EE</td>
<td>RDPKRU</td>
</tr>
</tbody>
</table>
Other Mitigation - PMI

PMI Handler with Target Tag Check

IND JMP
IND CALL

LBR_FROM
LBR_TO

ENDBR32/ENDBR64
Summary

- VM escape is practical and impacts cloud foundation security
- Intel CET/MPX can enhance mitigation on ROP/JOP/COP and buffer overflow, specifically on cloud virtualization stack
Call For Actions

- Apply new CPU mechanisms such as CET/MPX/PMU/PKU on exploit defense
THANK YOU! QUESTIONS?
XIAONING.LI@ALIBABA-INC.COM