Ouroboros:
Tearing Xen Hypervisor With the Snake

Augeust 04, 2016

Shangcong Luan
Alibaba Cloud Platform Security Team

Who am I?

2014 - 2015 at Vulnhunt Security Team for APT Defense

2015 - now at Alibaba Cloud for Cloud Security

2016.05 *Advanced Exploitation: Xen Hypervisor VM Escape*, HITB AMS 2016

Agenda

1. Introduction
2. XSA-182/CVE-2016-6258
3. Exploitation Technologies
4. The End

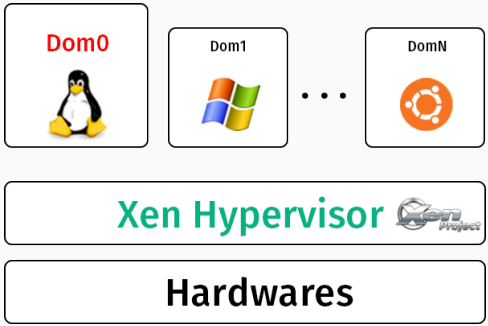
Introduction

The Xen Project

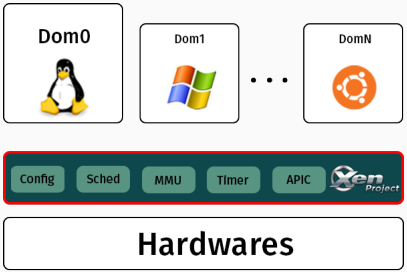
The Xen Project™ is the leading open source virtualization platform that is powering some of the largest clouds in production today.

from xenproject.org

Xen Architecture

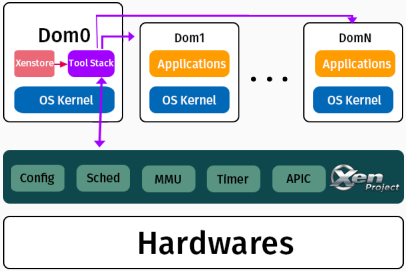


Xen Hypervisor



- CPU Scheduling
- Memory Management
- VM Executation
- ...

Xen Domain



Dom0:

- Privileged Domain
- Control Other Domains

domU:

- Dom1, Dom2, Dom3 ...
- Unprivileged Domains

Xen Domains Running Mode

PVM:

- paravirtualization machine
- modified OS kernel

HVM:

- hardware-assisted virtualization machine
- unmodified OS kernel
- CPU/MMU =¿ hardware assistance

x86 Paravirtualised Memory Management:

Direct Paging

Direct Paging: Address Spaces

guest pseudo-physical frame number (gpfn)

equals to

machine frame number (mfn)

Direct Paging: Page Type

mutually-exclusive page types:

- PGT_writable_page could be writable mapped into Guest
- PGT_l1_page_table L1 page table type
- PGT_l2_page_table L2 page table type
- PGT_l3_page_table L3 page table type
- PGT_l4_page_table L4 page table type

Direct Paging: Safe Invariants

1. A guest OS may always create readable mappings to its own page frames, regardless of their current types.
2. A frame may only safely be retasked when its reference count is zero.

PV Guest Cannot Read/Write Security-Sensitive Memorys,

e.g., page tables

Direct Paging: Safe Invariants

1. A guest OS may always create readable mappings to its own page frames, regardless of their current types.
2. A frame may only safely be retasked when its reference count is zero.

PV Guest Cannot Read/Write Security-Sensitive Memorys,

e.g., page tables

Direct Paging: Safe Invariants

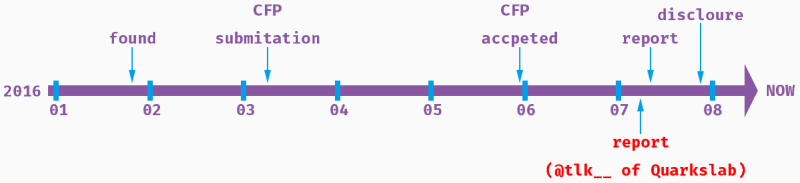
1. A guest OS may always create readable mappings to its own page frames, regardless of their current types.
2. A frame may only safely be retasked when its reference count is zero.

PV Guest Cannot Read/Write Security-Sensitive Memorys,

e.g., page tables.

XSA-182/CVE-2016-6258

Timeline


CFP
found
submitation
CFP
accpeted
report
discloure
2016
01
02
03
04
05
06
07
08
NOW
report
(@tlk__ of Quarkslab)

Xen Security Advisoriy[1]

Information

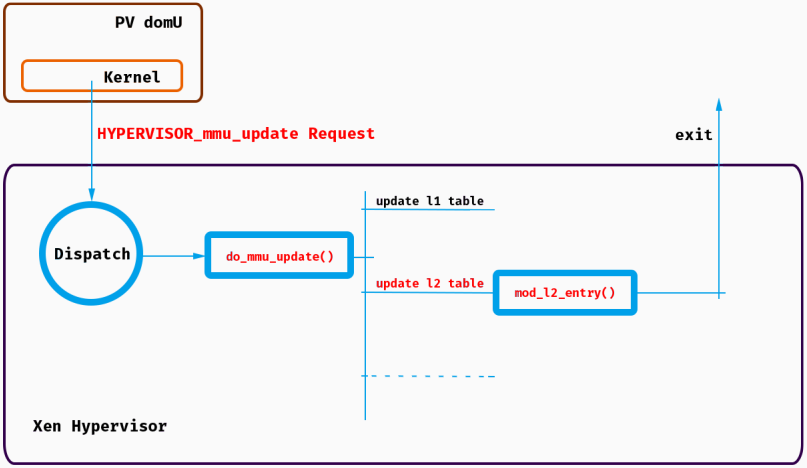
Advisory	XSA-182
Public release	2016-07-26 11:32
Updated	2016-07-26 11:32
Version	3
CVE(s)	CVE-2016-6258
Title	x86: Privilege escalation in PV guests

Advisory

```
                Xen Security Advisory CVE-2016-6258 / XSA-182
                                version 3

                     x86: Privilege escalation in PV guests

ISSUE DESCRIPTION
=================

The PV pagetable code has fast-paths for making updates to pre-existing
pagetable entries, to skip expensive re-validation in safe cases
(e.g. clearing only Access/Dirty bits).  The bits considered safe were too
broad, and not actually safe.

IMPACT
======

A malicous PV guest administrator can escalate their privilege to that
of the host.

VULNERABLE SYSTEMS
==================

All versions of Xen are vulnerable.

The vulnerability is only exposed to PV guests on x86 hardware.

The vulnerability is not exposed to x86 HVM guests, or ARM guests.
```

[1] http://xenbits.xen.org/xsa/advisory-182.html

_mmu_update hypercall request



mod_l2_entry()

```
1813 /* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
1814 static int mod_l2_entry( ... )
1819 {
         // skip some checks...
1835     if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
1836     {
1837         if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
1838         {
                 // check fault
1842         }
1843
             // "fast-path update" check
1844         /* Fast path for identical mapping and presence. */
1845         if ( !l2e_has_changed(ol2e, nl2e,
1846                               unlikely(opt_allow_superpage)
1847                               ? _PAGE_PSE | _PAGE_RW | _PAGE_PRESENT
1848                               : _PAGE_PRESENT) )
1849         {
                 // update operations ...
1854         }
1855
             // page type check
1856         if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
1857             return rc;
1858
              // update operations ...
1866     }
         // skip ...
1875 }
```

mod_l2_entry(): page type check

```
1813 /* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
1814 static int mod_l2_entry( ... )
1819 {
         // skip some checks...
1835     if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
1836     {
1837         if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
1838         {
                 // check fault
1842         }
1843
             // “fast-path update” check
1844         /* Fast path for identical mapping and presence. */
1845         if ( !l2e_has_changed(ol2e, nl2e,
1846                               unlikely(opt_allow_superpage)
1847                               ? _PAGE_PSE | _PAGE_RW | _PAGE_PRESENT
1848                               : _PAGE_PRESENT) )
1849         {
                 // update operations ...
1854         }
1855
             // page type check
1856         if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
1857             return rc;
1858
             // update operations ...
1866     }
         // skip ...
1875 }
```

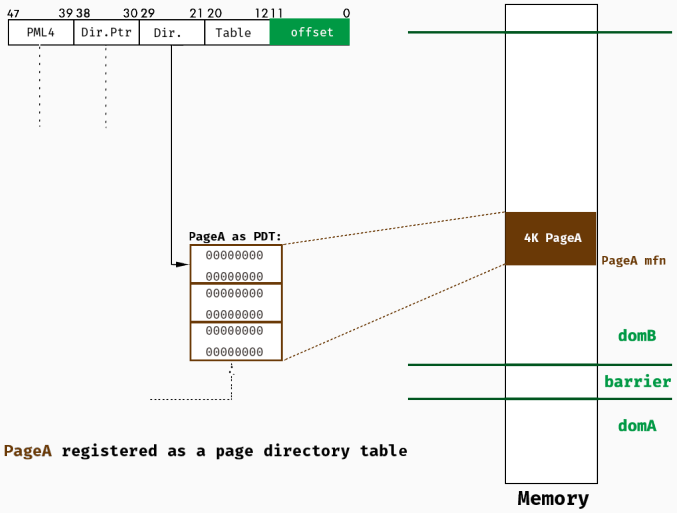
mod_l2_entry(): page type check

```
1813 /* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
1814 static int mod_l2_entry( ... )
1819 {
        // skip some checks...
1835    if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
1836    {
1837        if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
1838        {
                // check fault
1842        }
1843
            // "fast-path update" check
1844        /* Fast path for identical mapping and presence. */
1845        if ( !l2e_has_changed(ol2e, nl2e,
1846                              unlikely(opt_allow_superpage)
1847                              ? _PAGE_PSE | _PAGE_RW | _PAGE_PRESENT
1848                              : _PAGE_PRESENT) )
1849        {
                // update operations ...
1854        }
1855
            // page type check
1856        if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
1857            return rc;
1858
             // update operations ...
1866    }
        // skip ...
1875 }
```

```
static int get_page_from_l2e( ... )
{
    // skip ...
    rc = get_page_and_type_from_pagenr(mfn, PGT_l1_page_table, d, 0, 0);
    if ( unlikely(rc == -EINVAL) && get_l2_linear_pagetable(l2e, pfn, d) )
        rc = 0;
    return rc;
    // skip ...
}
```

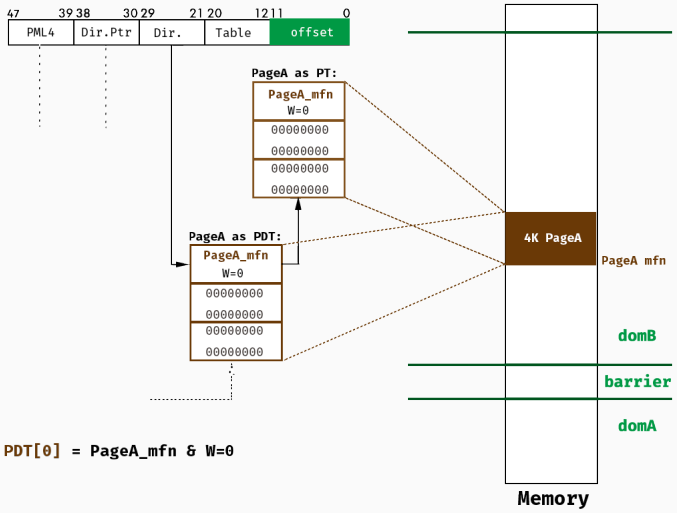
```
#define define_get_linear_pagetable(level)                                  \
static int                                                                   \
get_##level##_linear_pagetable(                                              \
    level##_pgentry_t pde, unsigned long pde_pfn, struct domain *d)          \
{                                                                            \
    // skip ...                                                              \
    if ( (level##e_get_flags(pde) & _PAGE_RW) )                              \
    {                                                                        \
        MEM_LOG("Attempt to create linear p.t. with write perms");           \
        return 0; // failed                                                  \
    }                                                                        \
                                                                             \
    if ( (pfn = level##e_get_pfn(pde)) != pde_pfn )                          \
    {                                                                        \
        // skip more checks ...                                              \
    }                                                                        \
    return 1; // success                                                     \
}
```

translation 1/4

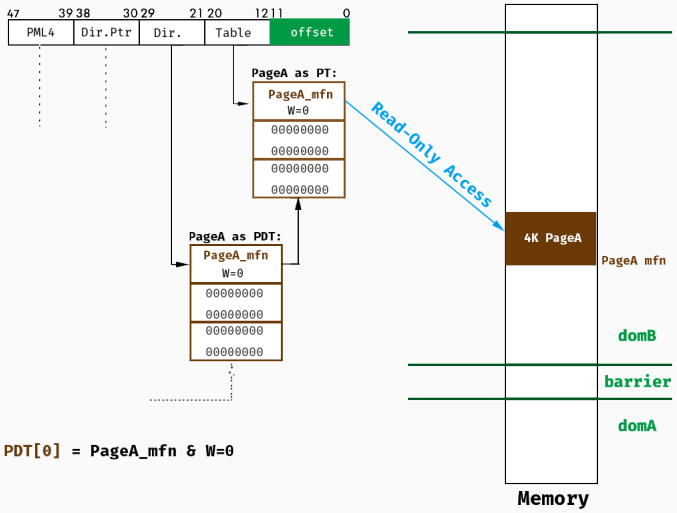


PageA registered as a page directory table

translation 2/4



translation 2/4



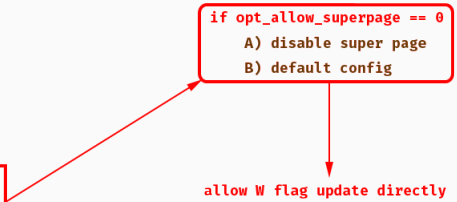
mod_l2_entry(): fast-path update check

```
1813 /* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
1814 static int mod_l2_entry( ... )
1819 {
         // skip some checks...
1835     if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
1836     {
1837         if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
1838         {
                 // check fault
1842         }
1843
             // "fast-path update" check
1844         /* Fast path for identical mapping and presence. */
1845         if ( !l2e_has_changed(ol2e, nl2e,
1846                               unlikely(opt_allow_superpage)
1847                               ? _PAGE_PSE | _PAGE_RW | _PAGE_PRESENT
1848                               : _PAGE_PRESENT) )
1849         {
                 // update operations ...
1854         }
1855
             // page type check
1856         if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
1857             return rc;
1858
              // update operations ...
1866     }
         // skip ...
1875 }
```

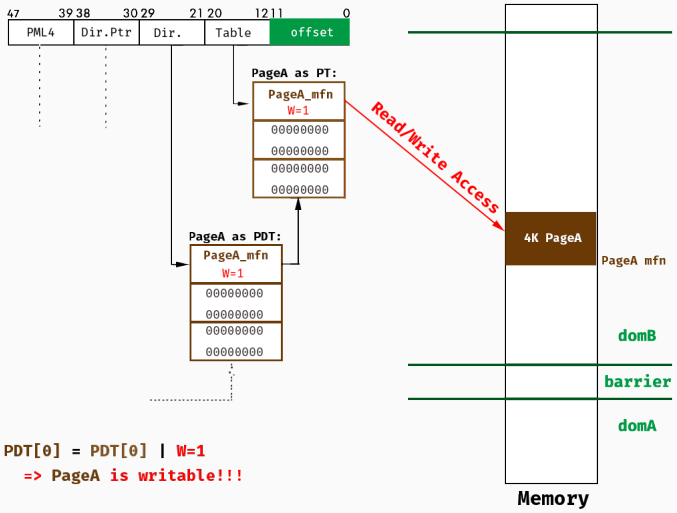
```
if opt_allow_superpage == 0
    A) disable super page
    B) default config
```

mod_l2_entry(): fast-path update check

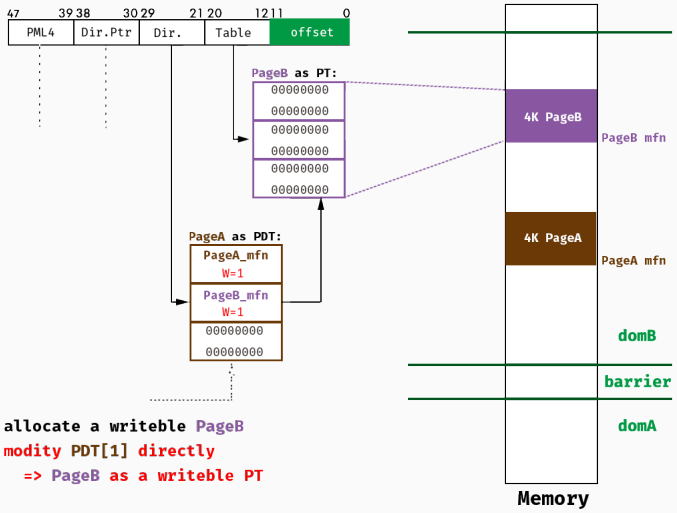
```
1813 /* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
1814 static int mod_l2_entry( ... )
1819 {
         // skip some checks...
1835     if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
1836     {
1837         if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
1838         {
                 // check fault
1842         }
1843
             // “fast-path update” check
1844         /* Fast path for identical mapping and presence. */
1845         if ( !l2e_has_changed(ol2e, nl2e,
1846                               unlikely(opt_allow_superpage)
1847                               ? _PAGE_PSE | _PAGE_RW | _PAGE_PRESENT
1848                               : _PAGE_PRESENT) )
1849         {
                 // update operations ...
1854         }
1855
             // page type check
1856         if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
1857             return rc;
1858
              // update operations ...
1866     }
         // skip ...
1875 }
```



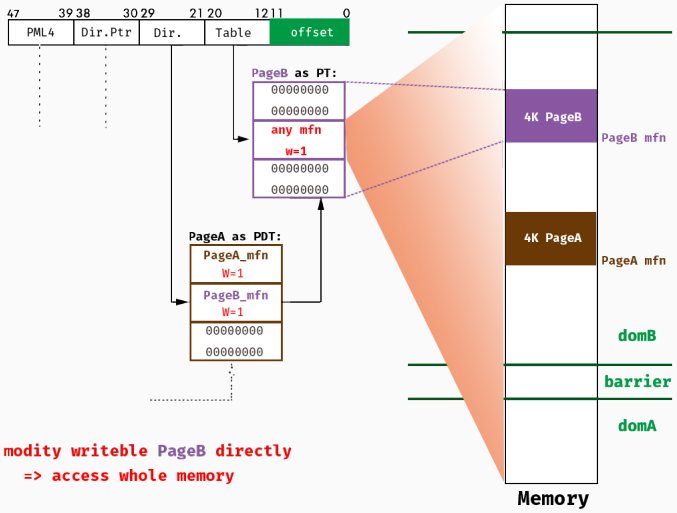
translation 3/4



translation 4/4



Whole Memory Access



Arbitrary Machine Memory Read/Write

four steps translate the XSA-182 to

Arbitrary Machine Memory Read/Write

Exploitation Technologies

Expectation

- Arbitrary Code Execution Within Domain Context
  - get dom0 root shell
  - get other domains root shell
- Arbitrary Code Execution Within VMM Context
  - fake hypercall inject

Expectation: get dom0 root shell

Hijack Vectors:

- VDSO/vsyscall Page (Linux)
- SharedUserData Page (Windows)
- Hypercall Page
- ...

Hypercall Page Hijack

Hypercall Page

- A 4K page alloced by Guest Kernel filled with 0xCC
- Need to be initialized with hypercall stub codes
- Guest Kernel use it to do hypercall request
- Each Guest Kernel only has one hypercall page

Hypercall Page Stub Codes

```
569  static void hypercall_page_initialise_ring3_kernel(void *hypercall_page)
570  {
571      char *p;
572      int i;
573      /* Fill in all the transfer points with template machine code. */
574      for ( i = 0; i < (PAGE_SIZE / 32); i++ )
575      {
576          if ( i == __HYPERVISOR_iret )
577              continue;
578          p = (char *)(hypercall_page + (i * 32));
579          *(u8  *)(p+ 0) = 0x51;    /* push %rcx */
580          *(u16 *)(p+ 1) = 0x5341;  /* push %r11 */
581          *(u8  *)(p+ 3) = 0xb8;    /* mov  $<i>,%eax */
582          *(u32 *)(p+ 4) = i;
583          *(u16 *)(p+ 8) = 0x050f;  /* syscall */
584          *(u16 *)(p+10) = 0x5b41;  /* pop  %r11 */
585          *(u8  *)(p+12) = 0x59;    /* pop  %rcx */
586          *(u8  *)(p+13) = 0xc3;    /* ret */
587      }
588      /*
589       * HYPERVISOR_iret is special because it doesn't return and expects a
590       * special stack frame. Guests jump at this transfer point instead of
591       * calling it.
592       */
593      p = (char *)(hypercall_page + (__HYPERVISOR_iret * 32));
594      *(u8  *)(p+ 0) = 0x51;    /* push %rcx */
595      *(u16 *)(p+ 1) = 0x5341;  /* push %r11 */
596      *(u8  *)(p+ 3) = 0x50;    /* push %rax */
597      *(u8  *)(p+ 4) = 0xb8;    /* mov  $__HYPERVISOR_iret,%eax */
598      *(u32 *)(p+ 5) = __HYPERVISOR_iret;
599      *(u16 *)(p+ 9) = 0x050f;  /* syscall */
600  }
```

Hypercall Page Signature

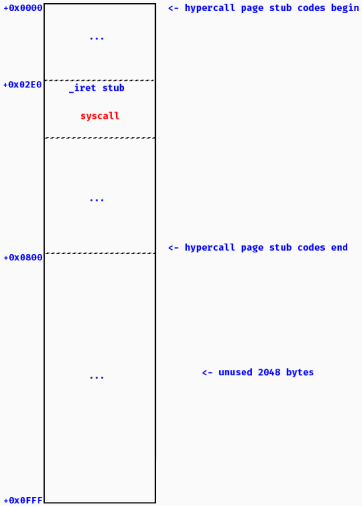
```
static void hypercall_page_initialise_ring3_kernel(void *hypercall_page)
{
    char *p;
    int i;
    /* Fill in all the transfer points with template machine code. */
    for ( i = 0; i < (PAGE_SIZE / 32); i++ )
    {
        if ( i == __HYPERVISOR_iret )
            continue;
        p = (char *)(hypercall_page + (i * 32));
        *(u8  *)(p+ 0) = 0x51;    /* push %rcx */
        *(u16 *)(p+ 1) = 0x5341;  /* push %r11 */
        *(u8  *)(p+ 3) = 0xb8;    /* mov  $<i>,%eax */
        *(u32 *)(p+ 4) = i;
        *(u16 *)(p+ 8) = 0x050f;  /* syscall */
        *(u16 *)(p+10) = 0x5b41;  /* pop  %r11 */
        *(u8  *)(p+12) = 0x59;    /* pop  %rcx */
        *(u8  *)(p+13) = 0xc3;    /* ret */
    }
    /*
     * HYPERVISOR_iret is special because it doesn't return and expects a
     * special stack frame. Guests jump at this transfer point instead of
}
```

Hypercall Page Signature:

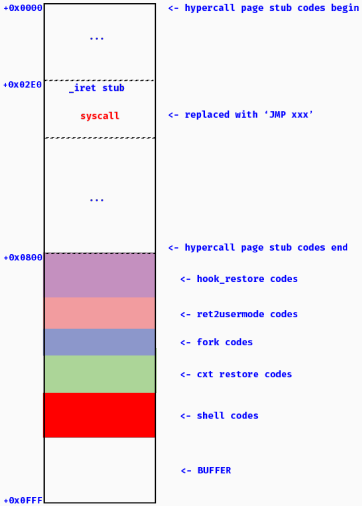
0x00000000B8534151

0xCCCC3595B41050F

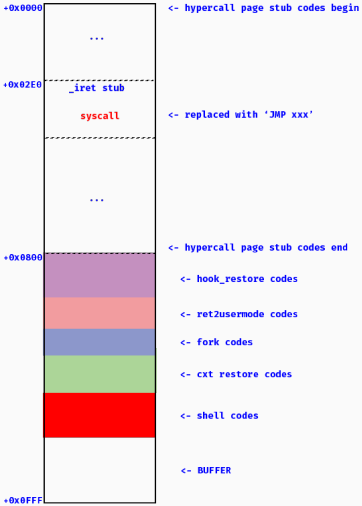
Hypercall Page Layout



Hypercall Page Hijacked Layout



Hypercall Page Hijacked Layout



Expectation: get dom0 root shell

- Arbitrary Code Execution Within Domain Context
  - get dom0 root shell 🙂
  - get other domains root shell
- Arbitrary Code Execution Within VMM Context
  - fake hypercall

Expectation: get other domains root shell

- Arbitrary Code Execution Within Domain Context
  - get dom0 root shell 🙂
  - get other domains root shell 😐
- Arbitrary Code Execution Within VMM Context
  - fake hypercall

Expectation: VMM code inject

- Arbitrary Code Execution Within Domain Context
  - get dom0 root shell 🙂
  - get other domains root shell 😐
- Arbitrary Code Execution Within VMM Context
  - fake hypercall

hypercall table

```
/* Xen 4.6 xen/x86/x86_64/entry.S */
726 ENTRY(hypercall_table)
727         .quad do_set_trap_table     /*  0 */
728         .quad do_mmu_update
729         .quad do_set_gdt
730         .quad do_stack_switch
731         .quad do_set_callbacks
732         .quad do_fpu_taskswitch     /*  5 */
            // skip ...
766         .quad do_ni_hypercall       /* reserved for XenClient */
767         .quad do_xenpmu_op          /* 40 */
768         .rept __HYPERVISOR_arch_0-((.-hypercall_table)/8)
769         .quad do_ni_hypercall
770         .endr
771         .quad do_mca                /* 48 */
772         .quad paging_domctl_continuation
773         .rept NR_hypercalls-((.-hypercall_table)/8)
774         .quad do_ni_hypercall
775         .endr
776
```

hypercall args table

```
/* Xen 4.6 xen/x86/x86_64/entry.S */
726 ENTRY(hypercall_table)
727         .quad do_set_trap_table     /*  0 */
            // skip ...
773         .rept NR_hypercalls-((.-hypercall_table)/8)
774         .quad do_ni_hypercall
775         .endr
776
777 ENTRY(hypercall_args_table)
778         .byte 1 /* do_set_trap_table    */  /*  0 */
779         .byte 4 /* do_mmu_update        */
780         .byte 2 /* do_set_gdt           */
781         .byte 2 /* do_stack_switch      */
782         .byte 3 /* do_set_callbacks     */
783         .byte 1 /* do_fpu_taskswitch    */  /*  5 */
784         .byte 2 /* do_sched_op_compat   */
785         .byte 1 /* do_platform_op       */
786         .byte 2 /* do_set_debugreg      */
787         .byte 1 /* do_get_debugreg      */
788         .byte 2 /* do_update_descriptor */  /* 10 */
789         .byte 0 /* do_ni_hypercall      */
790         .byte 2 /* do_memory_op         */
791         .byte 2 /* do_multicall         */
792         .byte 3 /* do_update_va_mapping */
793         .byte 1 /* do_set_timer_op      */  /* 15 */
794         .byte 1 /* do_event_channel_op_compat */
795         .byte 2 /* do_xen_version       */
796         .byte 3 /* do_console_io        */
797         .byte 1 /* do_physdev_op_compat */
```

hypercall args table

```
/* Xen 4.6 xen/x86/x86_64/entry.S */
726 ENTRY(hypercall_table)
727         .quad do_set_trap_table     /*  0 */
            // skip ...
773         .rept NR_hypercalls-((.-hypercall_table)/8)
774         .quad do_ni_hypercall
775         .endr
776
777 ENTRY(hypercall_args_table)
778         .byte 1 /* do_set_trap_table    */  /*  0 */
779         .byte 4 /* do_mmu_update        */
780         .byte 2 /* do_set_gdt           */
781         .byte 2 /* do_stack_switch      */
782         .byte 3 /* do_set_callbacks     */
                                                    5 */
792         .byte 3 /* do_update_va_mapping */
793         .byte 1 /* do_set_timer_op      */  /* 15 */
794         .byte 1 /* do_event_channel_op_compat */
795         .byte 2 /* do_xen_version       */
796         .byte 3 /* do_console_io        */
797         .byte 1 /* do_physdev_op_compat */
```

hypercall_args_table signature:
0x0102010302020401
0x0103020200020102
0x0004020301030201
0x0202020201040203

fake hypercall steps

Step 1: allocate a memory area and deploy codes in it:

padding padding

padding padding

padding padding

padding padding

padding padding

padding padding

padding padding

padding padding

```
shellcodes set current->domain->is_privileged to 1:
mov $0xfffffffffff8000,%rax
and %rsp,%rax
mov 0x7fe8(%rax),%rax
mov 0x10(%rax),%rax
movb $0x1,0x116(%rax)
retq
```

fake hypercall steps

Step 1: allocate a memory area and deploy code in it

Step 2: search hypercall_table and modify slot N to refer to the memory area

padding padding

padding padding

padding padding

padding padding

padding padding

padding padding

padding padding

padding padding

fake hypercall steps

Step 1: allocate a memory area and deploy code in it

Step 2: search hypercall_table and modify slot N to refer to the memory area

Step 3: bypass SMEP and SMAP features:

padding set PTE.U/S = 0

padding padding

padding padding

padding padding

padding padding

padding padding

padding padding

fake hypercall steps

Step 1: allocate a memory area and deploy code in it

Step 2: search hypercall_table and modify slot N to refer to the memory area

Step 3: bypass SMEP and SMAP features

Step 4: request this hypercall:

padding MOV $N, %RAX

padding SYSCALL

padding padding

padding padding

padding padding

padding padding

Expectation

- Arbitrary Code Execution Within Domain Context
  - get dom0 root shell 🙂
  - get other domains root shell 😐
- Arbitrary Code Execution Within VMM Context
  - fake hypercall 🙂

The End

10 years old

$ git show f87f8a7110e5dd57091b8484685953414693e2a3

```
Date:   Tue Feb 8 15:13:45 2005 +0000

+
+    if ( l2_pgentry_val(nl2e) & _PAGE_PRESENT )
+    {
+        /* Differ in mapping (bits 12-31) or presence (bit 0)? */
+        if ( ((l2_pgentry_val(ol2e) ^ l2_pgentry_val(nl2e)) & ~0xffe) == 0 )
+            return update_l2e(pl2e, ol2e, nl2e);
+
```

Thanks!

阿里云 aliyun.com

Question?

Shangcong Luan from Alibaba Cloud Platform Security Team