Advanced Exploitation: Xen Hypervisor VM Escape

Shangcong Luan

May 27, 2016

Alibaba Cloud Platform Security Team

Who am I?

2014 - 2015	at Vulnhunt Security Team for APT Defense
2015 - now	at Alibaba Cloud Platform Security Team for Cloud Security
Twitter:	@hikerell

Agenda

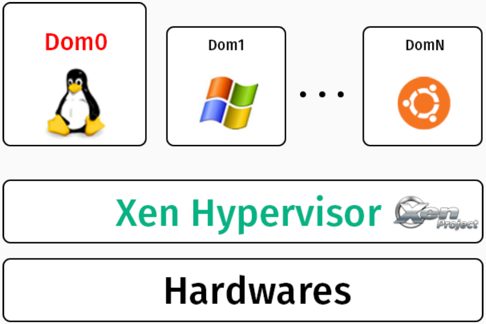
Introduction

The Xen Project

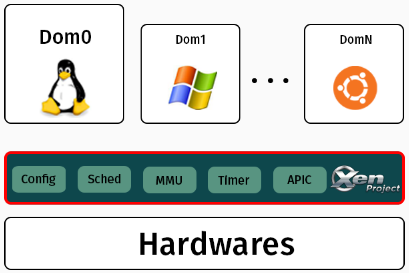
“ The Xen Project^{TM} is the leading open source virtualization platform that is powering some of the largest clouds in production today. ”

from xenproject.org

Xen Architecture

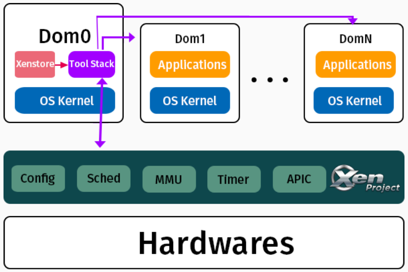


Xen Hypervisor



- 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

Xen Memory Management

x86 Paravirtualised Memory Management:

Direct Paging

Direct Paging: Address Spaces

machine frame number (mfn)

||

guest pseudo-physical frame number (gpfn)

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.

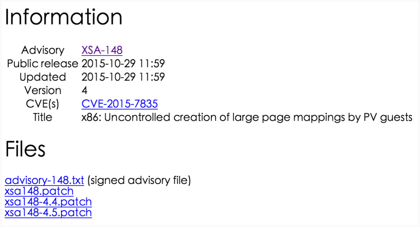
PV Guest Cannot Read/Write Security-Sensitive Memorys,

e.g., page tables.

XSA-148/CVE-2015-7835

Xen Security Advisoriy

Offical Vulnerablity Advisoriy[1]



Information

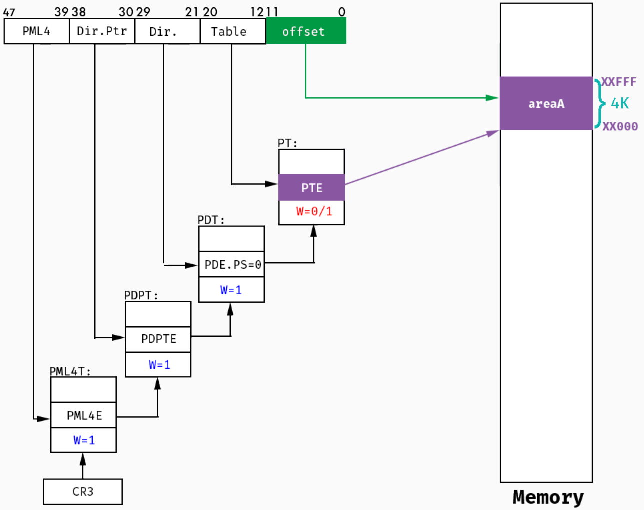
Advisory	XSA-148
Public release	2015-10-29 11:59
Updated	2015-10-29 11:59
Version	4
CVE(s)	CVE-2015-7835
Title	x86: Uncontrolled creation of large page mappings by PV guests

Files

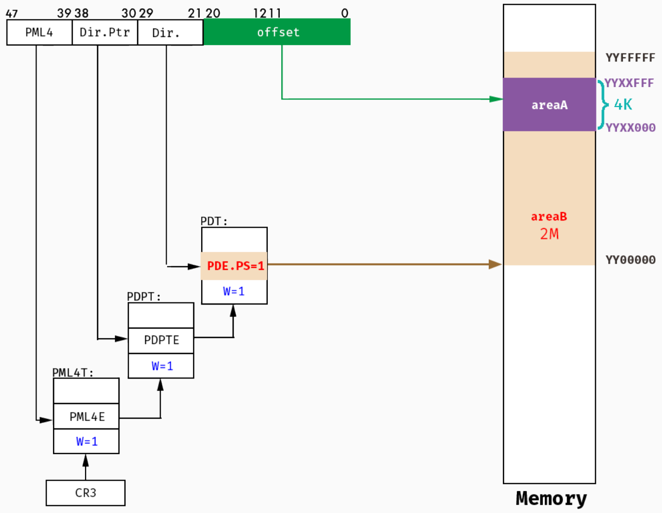
advisory-148.txt (signed advisory file)
xsa148.patch
xsa148-4.4.patch
xsa148-4.5.patch

[1]http://xenbits.xen.org/xsa/advisory-148.html

Intel IA-32e 4K Paging Mode



Intel IA-32e 2M Paging Mode



Page Tables Update Validation

xen-4.4.0 xen/arch/x86/mm.c

```
/* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
static int mod_l2_entry(l2_pgentry_t *pl2e,
                        l2_pgentry_t nl2e,
                        unsigned long pfn,
                        int preserve_ad,
                        struct vcpu *vcpu)
{
    l2_pgentry_t ol2e;
    struct domain *d = vcpu->domain;
    struct page_info *l2pg = mfn_to_page(pfn);
    unsigned long type = l2pg->u.inuse.type_info;
    int rc = 0;

    check-1:
    if ( unlikely(!is_guest_l2_slot(d, type, pgentry_ptr_to_slot(pl2e))) )
    {
        MEM_LOG("Illegal␣L2␣update␣attempt␣in␣Xen-private␣area␣%p", pl2e);
        return -EPERM;
    }

    if ( unlikely(__copy_from_user(&ol2e, pl2e, sizeof(ol2e)) != 0) )
        return -EFAULT;

```

Page Tables Update Validation

```
  check-2: PDE.P == 1 ?
  if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
  {
    check-3: PDE.reserved_bits == 0 ?
    if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
    {
      MEM_LOG("Bad␣L2␣flags␣%x",
          l2e_get_flags(nl2e) & L2_DISALLOW_MASK);
      return -EINVAL;
    }

    check-4: Old PDE.PADDR == New PDE.PADDR and Old PDE.P == New PDE.P ?
    /* Fast path for identical mapping and presence. */
    if ( !l2e_has_changed(ol2e, nl2e, _PAGE_PRESENT) )
    {
      adjust_guest_l2e(nl2e, d);
      if ( UPDATE_ENTRY(l2, pl2e, ol2e, nl2e, pfn, vcpu, preserve_ad) )
        return 0;
      return -EBUSY;
    }

   // check-5: do other audit, such as super page audit
    if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
      return rc;
    // ...
  }
  // ...
}
```

Page Tables Update Validation

```
    check-2: PDE.P == 1 ?
    if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
    {
        check-3: PDE.reserved_bits == 0 ?
        if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
        {
            MEM_LOG("Bad␣L2␣flags␣%x",
                    l2e_get_flags(nl2e) & L2_DISALLOW_MASK);
            return -EINVAL;
        }

        check-4: Old PDE.PADDR == New PDE.PADDR and Old PDE.P == New PDE.P ?
        /* Fast path for identical mapping and presence. */
        if ( !l2e_has_changed(ol2e, nl2e, _PAGE_PRESENT) )
        {
            adjust_guest_l2e(nl2e, d);
            if ( UPDATE_ENTRY(l2, pl2e, ol2e, nl2e, pfn, vcpu, preserve_ad) )
                return 0;
            return -EBUSY;
        }

       // check-5: do other audit, such as super page audit
        if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
            return rc;
        // ...
    }
    // ...
}
```

Page Tables Update Validation

```
    check-2: PDE.P == 1 ?
    if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
    {
        check-3: PDE.reserved_bits == 0 ?
        if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
        {
            MEM_LOG("Bad␣L2␣flags␣%x",
                    l2e_get_flags(nl2e) & L2_DISALLOW_MASK);
            return -EINVAL;
        }

        check-4: Old PDE.PADDR == New PDE.PADDR and Old PDE.P == New PDE.P ?
        /* Fast path for identical mapping and presence. */
        if ( !l2e_has_changed(ol2e, nl2e, _PAGE_PRESENT) )
        {
            adjust_guest_l2e(nl2e, d);
            if ( UPDATE_ENTRY(l2, pl2e, ol2e, nl2e, pfn, vcpu, preserve_ad) )
                return 0;
            return -EBUSY;
        }

       // check-5: do other audit, such as super page audit
        if ( unlikely((rc = get_page_from_l2e(nl2e, pfn, d)) < 0) )
            return rc;
        // ...
    }
    // ...
}
```

How to Trigger It?

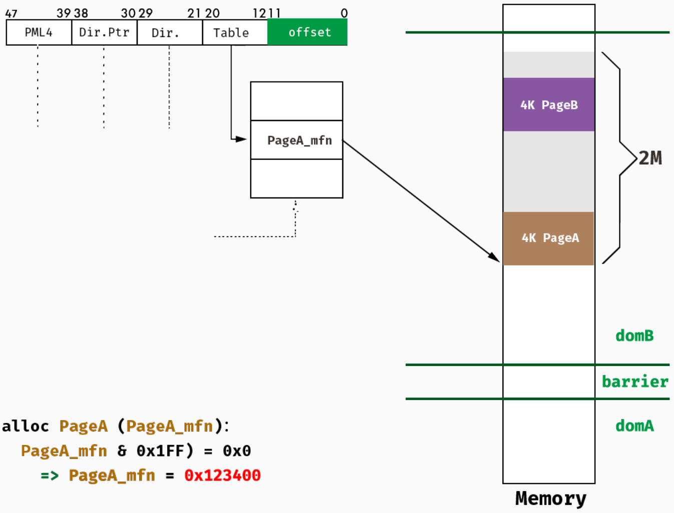
How to trigger it?

Arbitrary Physical Memory Read/Write

translate the XSA-148 to

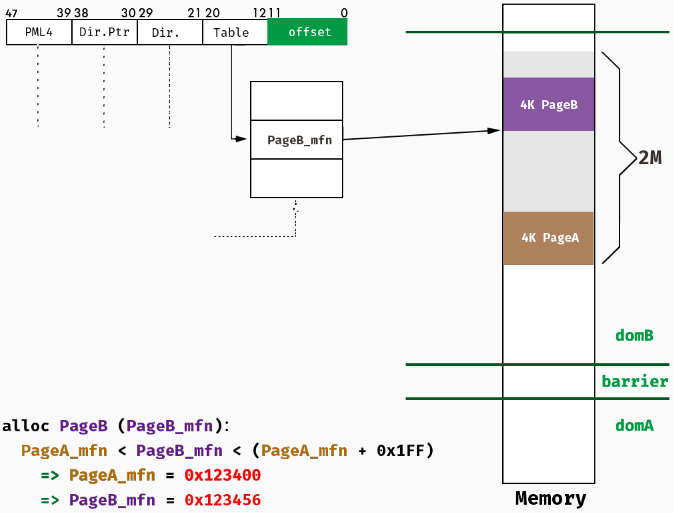
Arbitrary Physical Memory Read/Write

Step-1



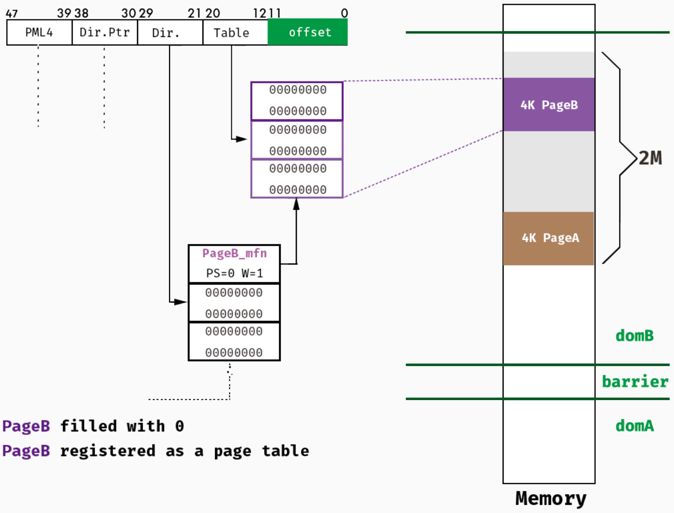
```
alloc PageA (PageA_mfn):
  PageA_mfn & 0x1FF) = 0x0
    => PageA_mfn = 0x123400
```

Step-2



```
alloc PageB (PageB_mfn):
  PageA_mfn < PageB_mfn < (PageA_mfn + 0x1FF)
    => PageA_mfn = 0x123400
    => PageB_mfn = 0x123456
```

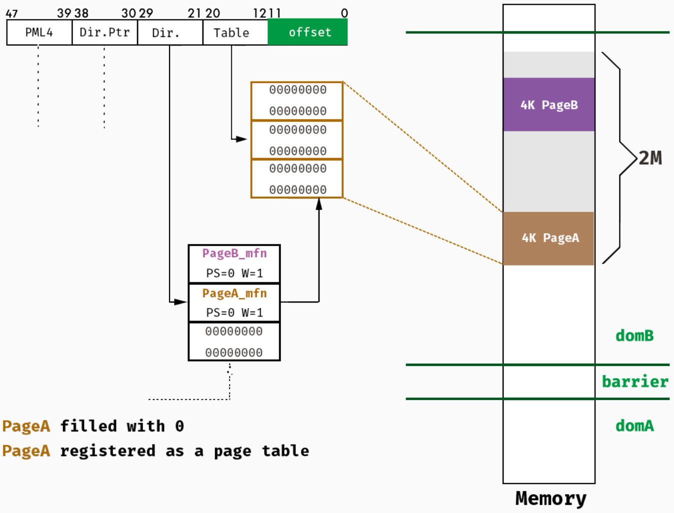
Step-3



PageB **filled with 0**

PageB **registered as a page table**

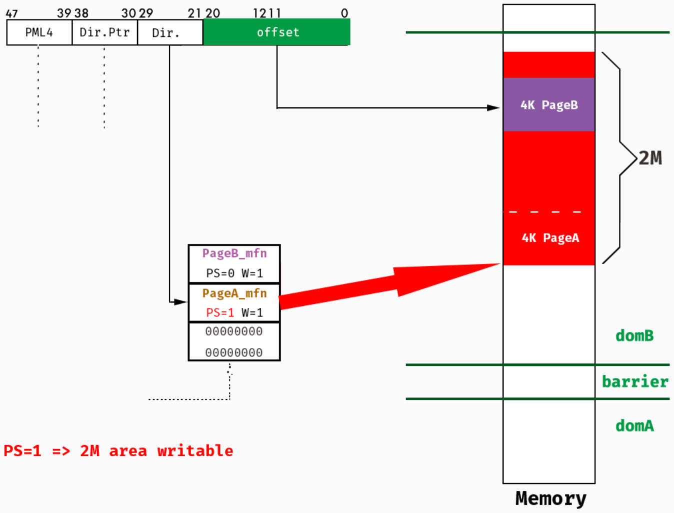
Step-4



PageA filled with 0

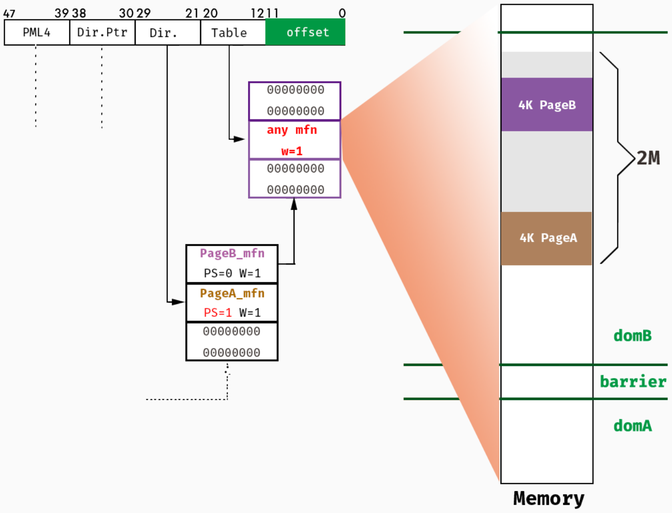
PageA registered as a page table

Step-5



PS=1 => 2M area writable

Step-6



47
39 38
30 29
21 20
12 11
0
PML4
Dir.Ptr
Dir.
Table
offset
00000000
00000000
any mfn
w=1
00000000
00000000
PageB_mfn
PS=0 W=1
PageA_mfn
PS=1 W=1
00000000
00000000
4K PageB
2M
4K PageA
domB
barrier
domA
Memory

Exploitation Technologies

Control Flow Hijack

Hijack Vectors:

- Hypercall Page
- VDSO/vsyscall Page
- Hypercall Table
- ...

Control Flow Hijack

Hijack Vectors:

- Hypercall Page
- VDSO/vsyscall Page
- Hypercall Table
- ...

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

- 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

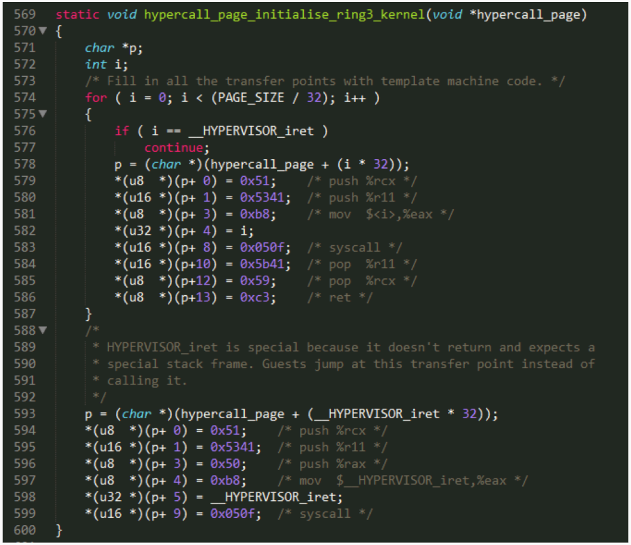
Hypercall Page

- 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

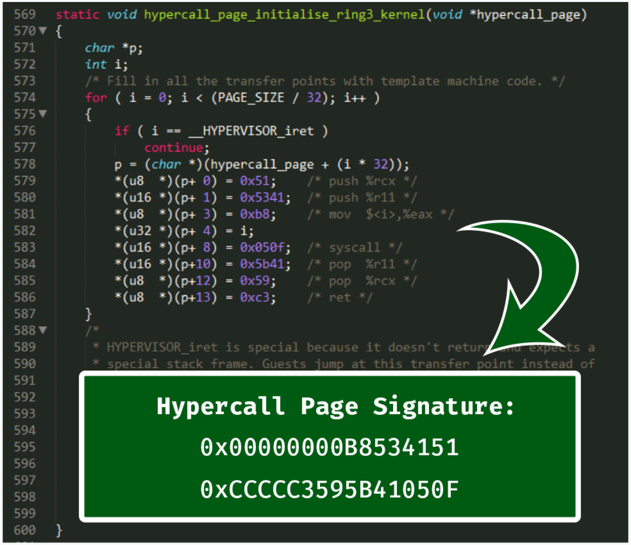
- A 4K page alloced by Guest Kernel filled with 0xCC
- Need to be initialized with hypercall stub codes
- Guest Kernel use it to do hypercall request
- Each Guest Kernel only has one hypercall page

Hypercall Page Stub Codes



```c
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
     * calling it.
     */
    p = (char *)(hypercall_page + (__HYPERVISOR_iret * 32));
    *(u8  *)(p+ 0) = 0x51;    /* push %rcx */
    *(u16 *)(p+ 1) = 0x5341;  /* push %r11 */
    *(u8  *)(p+ 3) = 0x50;    /* push %rax */
    *(u8  *)(p+ 4) = 0xb8;    /* mov  $__HYPERVISOR_iret,%eax */
    *(u32 *)(p+ 5) = __HYPERVISOR_iret;
    *(u16 *)(p+ 9) = 0x050f;  /* syscall */
}
```

Hypercall Page Signature



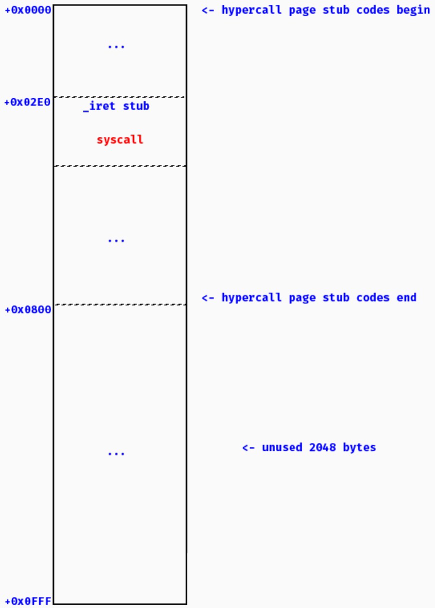
```
static void hypercall_page_initialise_ring3_kernel(void *hypercall_page)
{
    char *p;
    int i;
    /* Fill in all the transfer points with template machine code. */
    for ( i = 0; i < (PAGE_SIZE / 32); i++ )
    {
        if ( i == __HYPERVISOR_iret )
            continue;
        p = (char *)(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:

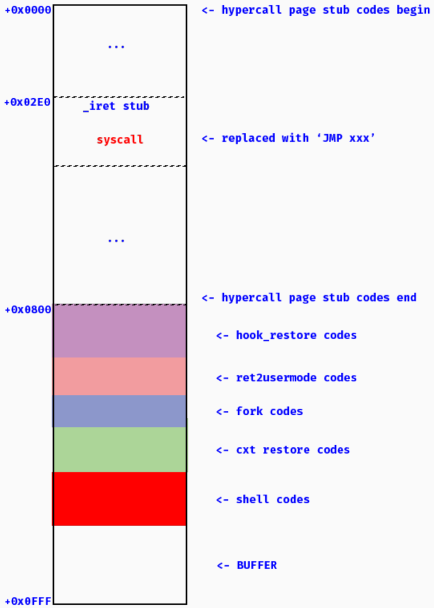
0x00000000B8534151

0xCCCCC3595B41050F

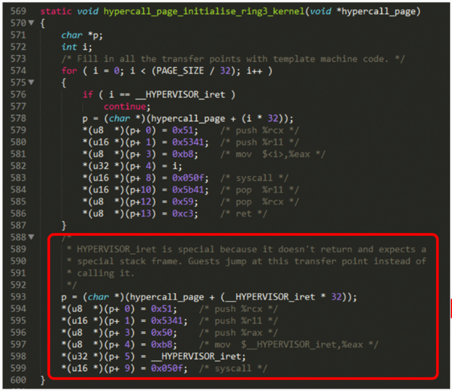
Hypercall Page Layout



Hypercall Page Hijacked Layout



Hypercall Page _IRET Stub Codes

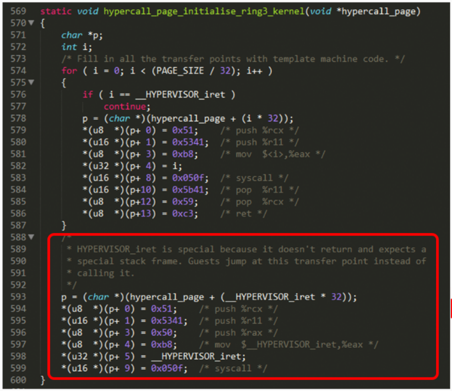


```c
static void hypercall_page_initialise_ring3_kernel(void *hypercall_page)
{
    char *p;
    int i;
    /* Fill in all the transfer points with template machine code. */
    for ( i = 0; i < (PAGE_SIZE / 32); i++ )
    {
        if ( i == __HYPERVISOR_iret )
            continue;
        p = (char *)(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
     * calling it.
     */
    p = (char *)(hypercall_page + (__HYPERVISOR_iret * 32));
    *(u8  *)(p+ 0) = 0x51;    /* push %rcx */
    *(u16 *)(p+ 1) = 0x5341;  /* push %r11 */
    *(u8  *)(p+ 3) = 0x50;    /* push %rax */
    *(u8  *)(p+ 4) = 0xb8;    /* mov  $__HYPERVISOR_iret,%eax */
    *(u32 *)(p+ 5) = __HYPERVISOR_iret;
    *(u16 *)(p+ 9) = 0x050f;  /* syscall */
}
```

Hypercall Page:

```
+00*32: pushq %rcx
        pushq %r11
        movq $0x0, %rax
        syscall
        popq %r11
        popq %rcx
        ...
+23*32: push %rcx
        push %r11
        push %rax
        mov $_HYPERVISOR_iret, %rax
        syscall
        ...
```

Hypercall Page _IRET Hijack

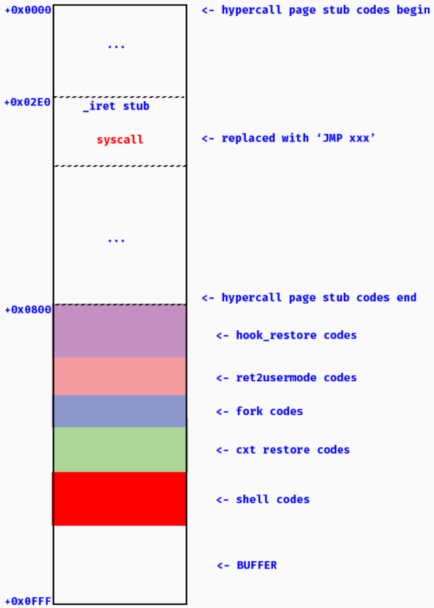


```c
static void hypercall_page_initialise_ring3_kernel(void *hypercall_page)
{
    char *p;
    int i;
    /* Fill in all the transfer points with template machine code. */
    for ( i = 0; i < (PAGE_SIZE / 32); i++ )
    {
        if ( i == __HYPERVISOR_iret )
            continue;
        p = (char *)(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
     * calling it.
     */
    p = (char *)(hypercall_page + (__HYPERVISOR_iret * 32));
    *(u8  *)(p+ 0) = 0x51;    /* push %rcx */
    *(u16 *)(p+ 1) = 0x5341;  /* push %r11 */
    *(u8  *)(p+ 3) = 0x50;    /* push %rax */
    *(u8  *)(p+ 4) = 0xb8;    /* mov  $__HYPERVISOR_iret,%eax */
    *(u32 *)(p+ 5) = __HYPERVISOR_iret;
    *(u16 *)(p+ 9) = 0x050f;  /* syscall */
}
```

Hypercall Page:

```
+00*32: pushq %rcx
        pushq %r11
        movq $0x0, %rax
        syscall
        popq %r11
        popq %rcx
        ...
+23*32: push %rcx
        push %r11
        push %rax
        mov $_HYPERVISOR_iret, %rax
        ~~syscall~~ JMP XXXX
        ...
```

Hypercall Page Hijacked Layout



The End

7-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);
+
```

only dom0 shell?

- dom0, dom1, dom2 …
- PV or HVM guest …
- linux or windows …

Demo Time

A Exploitation Demo

Host:	Debian 7 with Xen 4.4.0
Guest:	Debian 8
Video:	Get Dom0 Shell (XSA-148)

Thanks!

Question?

Shangcong Luan from Alibaba Cloud Platform Security Team