Xen API Archaeology

Creating a Full-Featured VMI Debugger for the Xen Hypervisor

Spencer Michaels[1]

Xen Project Developer and Design Summit 2019

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

$ whoami

- Security Consultant at NCC Group
- Unikernel security researcher
- Author of `xendbg`, a full-featured Xen VMI debugger
- Ardent Sinophile[1]

[1] `https://www.nccgroup.trust/sg/about-us/newsroom-and-events/blogs/2019/april/assessing-unikernel-security-cn/`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

- I’ve been doing security research on unikernels since summer 2017
  - Most popular ones are Xen-based
- Focused mostly on low-level implementation details
  - Stack & heap protections, entropy, ELF section initialization…

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

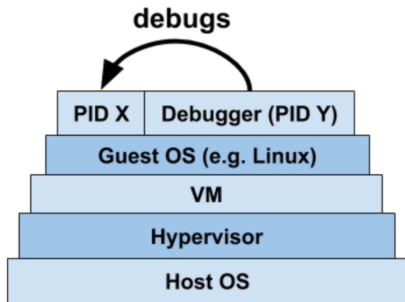
Background — Why make a Xen VMI debugger?

- I couldn't rely on simple test programs & `printf` debugging for what I was doing
  - Early initialization
  - Exploit development
  - Empirical testing of complex code
- Need introspection (i.e. debugging) capability

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

- Why not just run a debugger alongside the primary program?
- You can't! Unikernels have no concept of processes
  - Only one program
  - Baked in at compile time
  - Runs directly in kernel space
    - Yes, this is as scary as it sounds. See my Toorcon talk.[2]

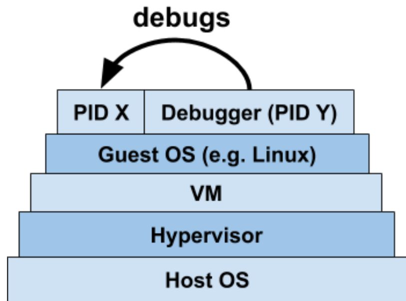


[2] https://www.youtube.com/watch?v=b68VFuB_y5M

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

- Why not just run a debugger alongside the primary program?
- You can't! Unikernels have no concept of processes
  - Only one program
  - Baked in at compile time
  - Runs directly in kernel space
    - Yes, this is as scary as it sounds. See my Toorcon talk.[2]

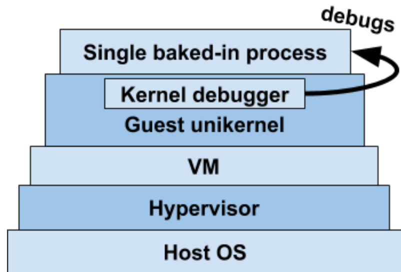


[2] https://www.youtube.com/watch?v=b68VFuB_y5M

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

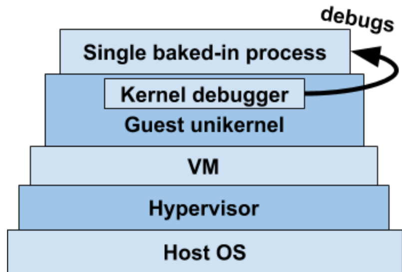
- Solution 1: Use a built-in debug shell/bridge (e.g. kdb)
- Bad for several reasons
  - Runs at the same level as the debugged application
    - Must avoid damaging itself when manipulating the VM
  - Can't run until system has started up
  - Large performance drop due to frequent traps & blocking
  - Has to be specifically implemented by the target unikernel



© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

- Solution 1: Use a built-in debug shell/bridge (e.g. kdb)
- Bad for several reasons
  - Runs at the same level as the debugged application
    - Must avoid damaging itself when manipulating the VM
  - Can't run until system has started up
  - Large performance drop due to frequent traps & blocking
  - Has to be specifically implemented by the target unikernel

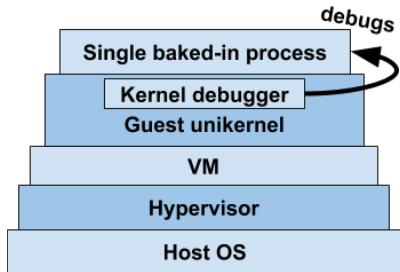


© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Background — Why make a Xen VMI debugger?

- Solution 1: Use a built-in debug shell/bridge (e.g. kdb)
- Bad for several reasons
  - Runs at the same level as the debugged application
    - Must avoid damaging itself when manipulating the VM
  - Can't run until system has started up
  - Large performance drop due to frequent traps & blocking
  - Has to be specifically implemented by the target unikernel

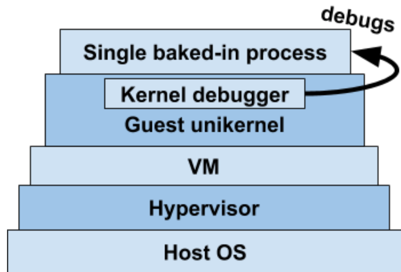


© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Background — Why make a Xen VMI debugger?

- Solution 1: Use a built-in debug shell/bridge (e.g. kdb)
- Bad for several reasons
  - Runs at the same level as the debugged application
    - Must avoid damaging itself when manipulating the VM
  - Can't run until system has started up
  - Large performance drop due to frequent traps & blocking
  - Has to be specifically implemented by the target unikernel

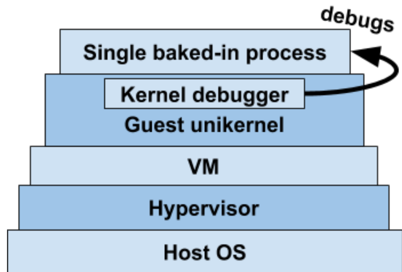


© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Background — Why make a Xen VMI debugger?

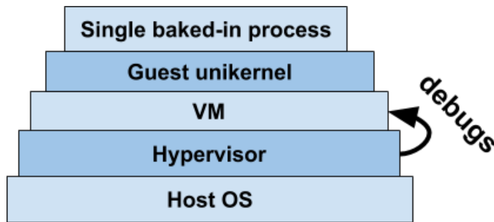
- Solution 1: Use a built-in debug shell/bridge (e.g. kdb)
- Bad for several reasons
  - Runs at the same level as the debugged application
    - Must avoid damaging itself when manipulating the VM
  - Can't run until system has started up
  - Large performance drop due to frequent traps & blocking
  - Has to be specifically implemented by the target unikernel



© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

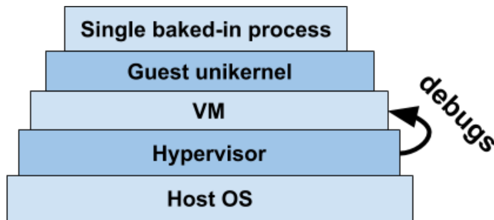
- Solution 2: Debug from the outside via **Virtual Machine Introspection (VMI)**
  - Inspect and manipulate the VM via the hypervisor
- Lots of benefits
  - Runs one level below the target
    - Can mess with VM contents at no risk to itself
  - Can debug from the very first instruction
  - Smaller performance hit
    - Event subscription
    - Non-blocking operations
  - Works for all VMs no matter their contents



© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

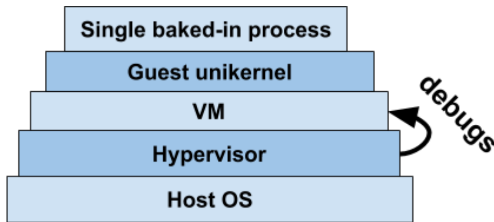
- Solution 2: Debug from the outside via **Virtual Machine Introspection (VMI)**
  - Inspect and manipulate the VM via the hypervisor
- Lots of benefits
  - Runs one level below the target
    - Can mess with VM contents at no risk to itself
  - Can debug from the very first instruction
  - Smaller performance hit
    - Event subscription
    - Non-blocking operations
  - Works for all VMs no matter their contents



© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

- Solution 2: Debug from the outside via **Virtual Machine Introspection (VMI)**
  - Inspect and manipulate the VM via the hypervisor
- Lots of benefits
  - Runs one level below the target
    - Can mess with VM contents at no risk to itself
  - Can debug from the very first instruction
  - Smaller performance hit
    - Event subscription
    - Non-blocking operations
  - Works for all VMs no matter their contents

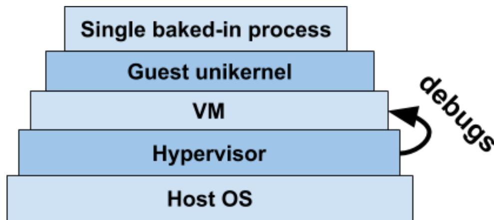


© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Background — Why make a Xen VMI debugger?

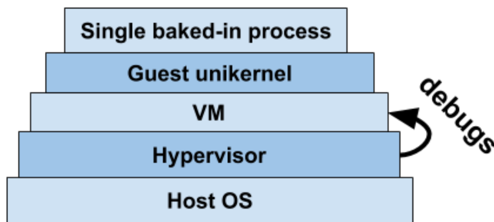
- Solution 2: Debug from the outside via **Virtual Machine Introspection (VMI)**
  - Inspect and manipulate the VM via the hypervisor
- Lots of benefits
  - Runs one level below the target
    - Can mess with VM contents at no risk to itself
  - Can debug from the very first instruction
  - Smaller performance hit
    - Event subscription
    - Non-blocking operations
  - Works for all VMs no matter their contents



© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a Xen VMI debugger?

- Solution 2: Debug from the outside via **Virtual Machine Introspection (VMI)**
  - Inspect and manipulate the VM via the hypervisor
- Lots of benefits
  - Runs one level below the target
    - Can mess with VM contents at no risk to itself
  - Can debug from the very first instruction
  - Smaller performance hit
    - Event subscription
    - Non-blocking operations
  - Works for all VMs no matter their contents



© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a *new* Xen VMI debugger?

- Xen comes with a basic GDB debug bridge, `gdbsx`
- But it's terrible
  - Written in 2007, well before Xen's current VMI API existed[3]
    - Uses `gdbsx`-branded magic hypercalls for guest interactions
  - Missing basic functionality, e.g. breakpoints
  - Basically a PEEK/POKE debugger, if you can get it to work at all
    - When I was originally writing xendbg, its GDB remote protocol implementation didn't work with GDB (which doesn't comply with its own spec)
    - "Fixed" much later [4]

[3] https://xenproject.org/2009/10/21/debugging-on-xen/
[4] https://github.com/xen-project/xen/commit/0c9821d5

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a *new* Xen VMI debugger?

- Xen comes with a basic GDB debug bridge, `gdbsx`
- But it's terrible
  - Written in 2007, well before Xen's current VMI API existed[3]
    - Uses `gdbsx`-branded magic hypercalls for guest interactions
  - Missing basic functionality, e.g. breakpoints
  - Basically a PEEK/POKE debugger, if you can get it to work at all
    - When I was originally writing xendbg, its GDB remote protocol implementation didn't work with GDB (which doesn't comply with its own spec)
    - "Fixed" much later [4]

[3] `https://xenproject.org/2009/10/21/debugging-on-xen/`
[4] `https://github.com/xen-project/xen/commit/0c9821d5`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Background — Why make a *new* Xen VMI debugger?

- Xen comes with a basic GDB debug bridge, `gdbsx`
- But it's terrible
  - Written in 2007, well before Xen's current VMI API existed[3]
    - Uses `gdbsx`-branded magic hypercalls for guest interactions
  - Missing basic functionality, e.g. breakpoints
  - Basically a PEEK/POKE debugger, if you can get it to work at all
    - When I was originally writing xendbg, its GDB remote protocol implementation didn't work with GDB (which doesn't comply with its own spec)
    - "Fixed" much later [4]

[3] `https://xenproject.org/2009/10/21/debugging-on-xen/`
[4] `https://github.com/xen-project/xen/commit/0c9821d5`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Background — Why make a *new* Xen VMI debugger?

- Xen comes with a basic GDB debug bridge, `gdbsx`
- But it's terrible
  - Written in 2007, well before Xen's current VMI API existed[3]
    - Uses `gdbsx`-branded magic hypercalls for guest interactions
  - Missing basic functionality, e.g. breakpoints
  - Basically a PEEK/POKE debugger, if you can get it to work at all
    - When I was originally writing `xendbg`, its GDB remote protocol implementation didn't work with GDB (which doesn't comply with its own spec)
    - "Fixed" much later [4]

[3] `https://xenproject.org/2009/10/21/debugging-on-xen/`
[4] `https://github.com/xen-project/xen/commit/0c9821d5`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Alternatives

- There are no generic alternatives to `gdbsx`
- `libvmi` supports only Windows and Linux VMs[5]
  - Not an option for exotic systems like unikernels!

[5] `http://libvmi.com/docs/gcode-intro.html`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Alternatives

- Why isn't there a better alternative to `gdbsx` already?

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Documentation

- The Xen VMI APIs are almost completely undocumented
- For the most part,
  - Many basic APIs are not mentioned anywhere online
    - Undiscoverable by keyword search on Google
    - Function names not always what you'd expect
  - Little to no example code
    - Not much on the public Internet
    - At most a few uses in Xen & `libvmi` combined
    - Many uses in `libvmi` are copied from Xen verbatim
    - Some calls only used for special cases, with the general case unclear
  - Code comments, if any, are rare

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Documentation

- The Xen VMI APIs are almost completely undocumented
- For the most part,
  - Many basic APIs are not mentioned anywhere online
    - Undiscoverable by keyword search on Google
    - Function names not always what you'd expect
  - Little to no example code
    - Not much on the public Internet
    - At most a few uses in Xen & `libvmi` combined
    - Many uses in `libvmi` are copied from Xen verbatim
    - Some calls only used for special cases, with the general case unclear
  - Code comments, if any, are rare

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Documentation

- The Xen VMI APIs are almost completely undocumented
- For the most part,
  - Many basic APIs are not mentioned anywhere online
    - Undiscoverable by keyword search on Google
    - Function names not always what you'd expect
  - Little to no example code
    - Not much on the public Internet
    - At most a few uses in Xen & `libvmi` combined
    - Many uses in `libvmi` are copied from Xen verbatim
    - Some calls only used for special cases, with the general case unclear
  - Code comments, if any, are rare

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Documentation

- The Xen VMI APIs are almost completely undocumented
- For the most part,
  - Many basic APIs are not mentioned anywhere online
    - Undiscoverable by keyword search on Google
    - Function names not always what you'd expect
  - Little to no example code
    - Not much on the public Internet
    - At most a few uses in Xen & `libvmi` combined
    - Many uses in `libvmi` are copied from Xen verbatim
    - Some calls only used for special cases, with the general case unclear
  - Code comments, if any, are rare

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Documentation

- **My conclusion:** So few people use these APIs because no one knows how to use them
- Developers may not even know they exist

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Documentation

- I’m here to fix that!
- This talk will detail all the APIs you need to make a basic Xen VMI tool

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

The Basics — PEEK/POKE

- Memory I/O
- Register I/O
- …and then it gets complicated

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Beyond The Basics — HVM & PV

- Xen guests come in two types:[6] hardware- and para-virtualized (HVM & PV)
- **HVM**
  - Guests are NOT virtualization-aware
  - Passes certain instructions straight to the processor
  - Relies on specialized CPU features to maintain guest/host isolation
- **PV**
  - Guest are virtualization-aware
  - Performs privileged operations via “hypercalls” to the hypervisor
  - Relies on the hypervisor to implement these ops (e.g. network and disk I/O)
    - Easier for devs, who don’t have to write their own low-level drivers
    - Most unikernels are PV

[6] Or a mix between the two: `https://wiki.xen.org/wiki/PV_on_HVM`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Beyond the Basics

- Beyond PEEK/POKE operations, PV and HVM diverge substantially
- PV VMI is terrible…
  - Relies on decade-old `gdbsx` hacks
  - Supports very little
- …but HVM VMI is great!
  - Has a full-fledged event channel API
  - Supports lots of event types

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Features covered — Extras

- Get domain metadata
- Listen for domain start/stop

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Assumptions: Architecture

- My sample code assumes x86/x86-64 guests only
- Everything in this talk applies generally to other architectures
- However, specifics may differ…
  - Breakpoint instructions
  - Register names
  - Differences in struct layout due to arch-specific typedefs
  - Some APIs may not work on other architectures — I didn't test others

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Assumptions: Error checking

- For brevity's sake, my sample code does not do error checking, but **yours** should!
- Almost every Xen VMI function returns an `int`
  - Zero: **OK**
  - Negative: **Error**, usually either (a) a negative `errno` value or (b) `-1`, setting the global `errno`
  - Positive: **A valid value** whose meaning depends on the function
- Exception: functions that return a pointer return `NULL` on error and set the global `errno`
  - Positive `errno` values are used here

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Xen VMI API

Introduction to the Sub-Libraries

Xen VMI API — Libraries

- `xenctrl` (`xc_*`): Manipulate & monitor domains & Xen itself
  - Most functionality is in this module; kind of a ”god object”
  - `xc_domain_*` functions affect individual domains
- `xenstore`: Read/write domain metadata, monitor
- `xencall`: Make arbitrary hypervisor calls
- `xendevicemodel`: Manage/access virtual hardware, inject events
- `xenevtchn`: HVM event channel (subscribe to events, e.g. breakpoints)
- `xenforeignmemory`: Map guest memory into Dom0
- `xenlight` & `xlutil` (`libxl_*`): Create, destroy and reboot domains
- `xengnttab`: Grant tables (share memory between guests)
- `xenttollog`: Inject custom loggers
- `xentoolcore`: ???

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Xen VMI API — Libraries

- Most library functions take a handle as their first parameter.
- For each library,
  - The handle type is `$LIB_handle`
  - Open with `$LIB_handle *handle = $LIB_open(/* args differ */);`
  - Close with `$LIB_close(handle);`
  - Exceptions: `xenctrl` (`xc_*`) and `xenstore` (`xs_*`)
- Remember to close the handle when you're done!

```
xc_interface *xenctrl = xc_interface_open(NULL, NULL, 0); // xenctrl uses acronym prefix
int version = xc_version(xenctrl, XENVER_version, NULL);
xc_interface_close(xenctrl);

xencall_handle *xencall = xencall_open(NULL, 0); // All other libs use this naming scheme
int err = xencall1(xencall, __HYPERVISOR_domctl, HYPERCALL_BUFFER_AS_ARG(domctl));
xencall_close(xencall);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O

Memory I/O — The old way: `XEN_DOMCTL_gdbsx_guestmemio`

- `gdbsx` uses the raw `XEN_DOMCTL_gdbsx_guestmemio` hypercall [7]
- This works, but I recommend against it
  - It's a hack from before Xen had proper VMI APIs
  - Just copies a region of guest memory to/from a buffer in dom0
    - Un-performant compared to the modern API

[7] See `xg_read_mem()` (L. 770) and `xg_right_mem()` (L. 801) from `https://github.com/mirage/xen/blob/master/tools/debugger/gdbsx/xg/xg_main.c#L770`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — The new way: `xenforeignmemory_map()`

- Instead, use `xenforeignmemory_map()` to map guest memory directly into Dom0
- Must specify the pages to map using **guest frame numbers**
  - Will explain how to calculate these in a bit…
- Remember to unmap when you're done!

```
void *xenforeignmemory_map(
    xenforeignmemory_handle *fmem,    /* Library handle */
    uint32_t dom,                     /* Domain ID */
    int prot,                         /* Prot flags (PROT_READ, PROT_WRITE...) */
    size_t num_pages,                 /* Size of `pages` array */
    const xen_pfn_t pages[/*pages*/], /* Array of guest frame numbers */
    int errors[/*pages*/]);           /* Array of errnos (same length as pages) */

int xenforeignmemory_unmap(
    xenforeignmemory_handle *fmem,    /* Library handle */
    void *addr,                       /* Buffer address returned by XFM_map() */
    size_t num_pages);                /* Number of pages mapped */
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — xenforeignmemory_map() args

```
void *xenforeignmemory_map(
    xenforeignmemory_handle *fmem,    /* Library handle */
    uint32_t dom,
    int prot,
    size_t num_pages,
    const xen_pfn_t pages[/*pages*/],
    int errors[/*pages*/]);
```

- Pass in the library handle obtained from `xenforeignmemory_open()`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Memory I/O — `xenforeignmemory_map()` args

```
void *xenforeignmemory_map(
    xenforeignmemory_handle *fmem,
    uint32_t dom,                    /* Domain ID */
    int prot,
    size_t num_pages,
    const xen_pfn_t pages[/*pages*/],
    int errors[/*pages*/]);
```

- `domid` is the numeric ID of the target domain
- Obtained from xenstore metadata (later…)
- All domain-affecting functions (e.g. `xc_domain_*`) take one
    - Usually as the second parameter

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Memory I/O — `xenforeignmemory_map()` args

```
void *xenforeignmemory_map(
    xenforeignmemory_handle *fmem,
    uint32_t dom,
    int prot,                          /* PROT_READ, PROT_WRITE, PROT_EXEC, etc. */
    size_t num_pages,
    const xen_pfn_t pages[/*pages*/],
    int errors[/*pages*/]);
```

- PROT flags are exactly the same as regular POSIX

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — xenforeignmemory_map() args

```
void *xenforeignmemory_map(
    xenforeignmemory_handle *fmem,
    uint32_t dom,
    int prot,
    size_t num_pages,                /* Size of `pages` array */
    const xen_pfn_t pages[/*pages*/],
    int errors[/*pages*/]);
```

- Use Xen constants to calculate the number of pages you need to map

```
static inline size_t calc_num_pages(size_t num_bytes) {
  return (size + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Memory I/O — `xenforeignmemory_map()` args

```
void *xenforeignmemory_map(
    xenforeignmemory_handle *fmem,
    uint32_t dom,
    int prot,
    size_t num_pages,
    const xen_pfn_t pages[/*pages*/], /* Array of guest frame numbers */
    int errors[/*pages*/]);
```

- Guest frame numbers are the indices of pages on the guest
- Guest vaddrs must be *translated* into a GFN via `xc_translate_foreign_address()`

"The foreign map API uses a GFN, even if the underlying API describes the parameter name as MFN. This is a consequence of PV guests having been developed long before hardware virt extensions came along, and noone having gone through and retroactively updated the terminology." — Andrew Cooper [8]

[8] https://lists.xenproject.org/archives/html/xen-devel/2018-10/msg00947.html

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — `xenforeignmemory_map()` args

```
void *xenforeignmemory_map(
    xenforeignmemory_handle *fmem,
    uint32_t dom,
    int prot,
    size_t num_pages,
    const xen_pfn_t pages[/*pages*/],
    int errors[/*pages*/]);           /* Array of errnos, same size as `pages` */
```

- Each page can fail to map individually, so there's one errno per page

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — `xc_translate_foreign_address()`

- `xc_translate_foreign_address()` translates guest virtual addresses to GFNs
  - See the implementation for how to read page tables![9]

```
unsigned long xc_translate_foreign_address(
  xc_interface *xch,        /* XenCtrl handle */
  uint32_t domid,           /* Domain ID */
  int vcpu,                 /* VCPU ID (0-indexed) */
  unsigned long long virt); /* Guest virtual address */
```

[9] https://github.com/xen-project/xen/blob/0ca7624/tools/libxc/xc_pagetab.c#L29

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — `xenforeignmemory_map()` Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — `xenforeignmemory_map()` Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — xenforeignmemory_map() Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — xenforeignmemory_map() Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — xenforeignmemory_map() Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — `xenforeignmemory_map()` Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — `xenforeignmemory_map()` Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Memory I/O — `xenforeignmemory_map()` Example

```
void *map_memory(xenforeignmemory_handle *fmem, xenctrl_handle *xch,
                 uint32_t domid, int vcpu, addr_t vaddr, size_t num_bytes, int prot) {
  const size_t num_pages = (num_bytes + XC_PAGE_SIZE - 1) >> XC_PAGE_SHIFT;
  xen_pfn_t *pages = (xen_pfn_t*)malloc(num_pages * sizeof(xen_pfn_t));
  int *errors = (int*)malloc(num_pages * sizeof(int));

  const xen_pfn_t base_gfn = xc_translate_foreign_address(xch, domid, vcpu, vaddr);
  for (size_t i = 0; i < num_pages; ++i)
    pages[i] = base_gfn + i;

  void *mem = xenforeignmemory_map(fmem, domain_id, prot, num_pages, pages, errors);

  for (size_t i = 0; i < num_pages; ++i)
    if (errors[i]) // Failed to map i-th page with errno errors[i]
      return NULL;
  return mem;
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Memory I/O — `xenforeignmemory_unmap()`

- Remember to unmap the shared memory once you're done using it
- Use `xenforeignmemory_unmap()`
- You'll need the base pointer (returned by `_map()`) and the number of pages mapped

```
int xenforeignmemory_unmap(
    xenforeignmemory_handle *fmem,  /* Library handle */
    void *addr,                     /* Shared memory base address (returned by _map()) */
    size_t num_pages);              /* Number of pages mapped */
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O on PV

Register I/O — PV Read & Write

- Get/set register state with `xc_vcpu_{get,set}context()`
- Represented by `vcpu_guest_context_any_t`
  - Xen will populate one of two union members depending on guest width

```
int xc_vcpu_getcontext(
    xc_interface *xch,
    uint32_t domid,
    uint32_t vcpu,
    vcpu_guest_context_any_t *ctxt);

int xc_vcpu_setcontext(
    xc_interface *xch,
    uint32_t domid,
    uint32_t vcpu,
    vcpu_guest_context_any_t *ctxt);
```

```
typedef union {
  vcpu_guest_context_x86_64_t x64; // rip...
  vcpu_guest_context_x86_32_t x32; // eip...
  vcpu_guest_context_t c;          // internal
} vcpu_guest_context_any_t;

// Get the guest width (in BYTES, not bits)
int xc_domain_get_guest_width(
    xc_interface *xch,
    uint32_t domid,
    unsigned int *guest_width);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Register I/O — PV Read & Write

- Get/set register state with `xc_vcpu_{get,set}context()`
- Represented by `vcpu_guest_context_any_t`
  - Xen will populate one of two union members depending on guest width

```
int xc_vcpu_getcontext(
    xc_interface *xch,
    uint32_t domid,
    uint32_t vcpu,
    vcpu_guest_context_any_t *ctxt);

int xc_vcpu_setcontext(
    xc_interface *xch,
    uint32_t domid,
    uint32_t vcpu,
    vcpu_guest_context_any_t *ctxt);
```

```
typedef union {
  vcpu_guest_context_x86_64_t x64; // rip...
  vcpu_guest_context_x86_32_t x32; // eip...
  vcpu_guest_context_t c;          // internal
} vcpu_guest_context_any_t;

// Get the guest width (in BYTES, not bits)
int xc_domain_get_guest_width(
    xc_interface *xch,
    uint32_t domid,
    unsigned int *guest_width);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — PV Read & Write

- Get/set register state with `xc_vcpu_{get,set}context()`
- Represented by `vcpu_guest_context_any_t`
  - Xen will populate one of two union members depending on guest width

```
int xc_vcpu_getcontext(
    xc_interface *xch,
    uint32_t domid,
    uint32_t vcpu,
    vcpu_guest_context_any_t *ctxt);

int xc_vcpu_setcontext(
    xc_interface *xch,
    uint32_t domid,
    uint32_t vcpu,
    vcpu_guest_context_any_t *ctxt);
```

```
typedef union {
  vcpu_guest_context_x86_64_t x64; // rip...
  vcpu_guest_context_x86_32_t x32; // eip...
  vcpu_guest_context_t c;          // internal
} vcpu_guest_context_any_t;

// Get the guest width (in BYTES, not bits)
int xc_domain_get_guest_width(
    xc_interface *xch,
    uint32_t domid,
    unsigned int *guest_width);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — PV Read & Write Example

```c
// Get the current register state
vcpu_guest_context_any_t ctx;
xc_vcpu_getcontext(handle, domid, vcpu_id, &ctx);

// Modify it
unsigned width;
xc_domain_get_guest_width(handle, domid, &width);
if (width == 8)
  ctx.x64.rip = 0xDEADBEEFDEADBEEF;
else if (width == 4)
  ctx.x32.eip = 0xDEADBEEF;

// Write it back to the VM
xc_vcpu_setcontext(handle, domid, vcpu_id, &ctx);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — PV Read & Write Example

```
// Get the current register state
vcpu_guest_context_any_t ctx;
xc_vcpu_getcontext(handle, domid, vcpu_id, &ctx);

// Modify it
unsigned width;
xc_domain_get_guest_width(handle, domid, &width);
if (width == 8)
  ctx.x64.rip = 0xDEADBEEFDEADBEEF;
else if (width == 4)
  ctx.x32.eip = 0xDEADBEEF;

// Write it back to the VM
xc_vcpu_setcontext(handle, domid, vcpu_id, &ctx);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — PV Read & Write Example

```
// Get the current register state
vcpu_guest_context_any_t ctx;
xc_vcpu_getcontext(handle, domid, vcpu_id, &ctx);

// Modify it
unsigned width;
xc_domain_get_guest_width(handle, domid, &width);
if (width == 8)
  ctx.x64.rip = 0xDEADBEEFDEADBEEF;
else if (width == 4)
  ctx.x32.eip = 0xDEADBEEF;

// Write it back to the VM
xc_vcpu_setcontext(handle, domid, vcpu_id, &ctx);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Register I/O — PV Read & Write Example

```
// Get the current register state
vcpu_guest_context_any_t ctx;
xc_vcpu_getcontext(handle, domid, vcpu_id, &ctx);

// Modify it
unsigned width;
xc_domain_get_guest_width(handle, domid, &width);
if (width == 8)
  ctx.x64.rip = 0xDEADBEEFDEADBEEF;
else if (width == 4)
  ctx.x32.eip = 0xDEADBEEF;

// Write it back to the VM
xc_vcpu_setcontext(handle, domid, vcpu_id, &ctx);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Register I/O — PV Read & Write Example

```
// Get the current register state
vcpu_guest_context_any_t ctx;
xc_vcpu_getcontext(handle, domid, vcpu_id, &ctx);

// Modify it
unsigned width;
xc_domain_get_guest_width(handle, domid, &width);
if (width == 8)
  ctx.x64.rip = 0xDEADBEEFDEADBEEF;
else if (width == 4)
  ctx.x32.eip = 0xDEADBEEF;

// Write it back to the VM
xc_vcpu_setcontext(handle, domid, vcpu_id, &ctx);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O on HVM

Register I/O — HVM Read (No Write)

- Use `xc_domain_hvm_getcontext_partial()`
- Can get more than just reg state, so we need to specify what we want

```
HVM_SAVE_TYPE(CPU) registers; // .rax, .rbx, .rcx, etc...

int xc_domain_hvm_getcontext_partial(
    xc_interface *xch,
    uint32_t domid,
    uint16_t typecode, // HVM_SAVE_CODE(CPU)
    uint16_t instance, // VCPU ID
    void *ctxt_buf,    // HVM_SAVE_TYPE(CPU)*
    uint32_t size);    // sizeof(HVM_SAVE_TYPE(CPU))
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read (No Write)

- Use `xc_domain_hvm_getcontext_partial()`
- Can get more than just reg state, so we need to specify what we want

```
HVM_SAVE_TYPE(CPU) registers; // .rax, .rbx, .rcx, etc...

int xc_domain_hvm_getcontext_partial(
    xc_interface *xch,
    uint32_t domid,
    uint16_t typecode, // HVM_SAVE_CODE(CPU)
    uint16_t instance, // VCPU ID
    void *ctxt_buf,    // HVM_SAVE_TYPE(CPU)*
    uint32_t size);    // sizeof(HVM_SAVE_TYPE(CPU))
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Register I/O — HVM Read (No Write)

- Use `xc_domain_hvm_getcontext_partial()`
- Can get more than just reg state, so we need to specify what we want

```
HVM_SAVE_TYPE(CPU) registers; // .rax, .rbx, .rcx, etc...

int xc_domain_hvm_getcontext_partial(
    xc_interface *xch,
    uint32_t domid,
    uint16_t typecode, // HVM_SAVE_CODE(CPU)
    uint16_t instance, // VCPU ID
    void *ctxt_buf,    // HVM_SAVE_TYPE(CPU)*
    uint32_t size);    // sizeof(HVM_SAVE_TYPE(CPU))
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read (No Write)

- Use `xc_domain_hvm_getcontext_partial()`
- Can get more than just reg state, so we need to specify what we want

```
HVM_SAVE_TYPE(CPU) registers; // .rax, .rbx, .rcx, etc...

int xc_domain_hvm_getcontext_partial(
    xc_interface *xch,
    uint32_t domid,
    uint16_t typecode, // HVM_SAVE_CODE(CPU)
    uint16_t instance, // VCPU ID
    void *ctxt_buf,    // HVM_SAVE_TYPE(CPU)*
    uint32_t size);    // sizeof(HVM_SAVE_TYPE(CPU))
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read (No Write)

- Use `xc_domain_hvm_getcontext_partial()`
- Can get more than just reg state, so we need to specify what we want

```
HVM_SAVE_TYPE(CPU) registers; // .rax, .rbx, .rcx, etc...

int xc_domain_hvm_getcontext_partial(
    xc_interface *xch,
    uint32_t domid,
    uint16_t typecode, // HVM_SAVE_CODE(CPU)
    uint16_t instance, // VCPU ID
    void *ctxt_buf,    // HVM_SAVE_TYPE(CPU)*
    uint32_t size);    // sizeof(HVM_SAVE_TYPE(CPU))
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read (No Write)

- Use `xc_domain_hvm_getcontext_partial()`
- Can get more than just reg state, so we need to specify what we want

```
HVM_SAVE_TYPE(CPU) registers; // .rax, .rbx, .rcx, etc...

int xc_domain_hvm_getcontext_partial(
    xc_interface *xch,
    uint32_t domid,
    uint16_t typecode, // HVM_SAVE_CODE(CPU)
    uint16_t instance, // VCPU ID
    void *ctxt_buf,    // HVM_SAVE_TYPE(CPU)*
    uint32_t size);    // sizeof(HVM_SAVE_TYPE(CPU))
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read (No Write)

- Use `xc_domain_hvm_getcontext_partial()`
- Can get more than just reg state, so we need to specify what we want

```
HVM_SAVE_TYPE(CPU) registers; // .rax, .rbx, .rcx, etc...

int xc_domain_hvm_getcontext_partial(
    xc_interface *xch,
    uint32_t domid,
    uint16_t typecode, // HVM_SAVE_CODE(CPU)
    uint16_t instance, // VCPU ID
    void *ctxt_buf,    // HVM_SAVE_TYPE(CPU)*
    uint32_t size);    // sizeof(HVM_SAVE_TYPE(CPU))
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write

- Use `xc_domain_hvm_{get,set}context()`
- Context is a series of header-prefixed blocks of binary data **of variable total length**
- No way to craft valid HEADER data or get it via `_getcontext_partial()`[10]

```c
int xc_domain_hvm_setcontext(
    xc_interface *xch,
    uint32_t domid,
    uint8_t *hvm_ctxt,
    uint32_t size);

int xc_domain_hvm_getcontext(
    xc_interface *xch,
    uint32_t domid,
    uint8_t *ctxt_buf,
    uint32_t size);
```

```c
struct hvm_save_descriptor {
    uint16_t typecode; // HVM_SAVE_CODE(...)
    uint16_t instance; // VCPU ID
    uint32_t length;   // num bytes following
};
struct cpu_context_update { // User-defined example
    struct hvm_save_descriptor header_d;
    HVM_SAVE_TYPE(HEADER) header;
    struct hvm_save_descriptor cpu_d;
    HVM_SAVE_TYPE(CPU) cpu;
    struct hvm_save_descriptor end_d;
    HVM_SAVE_TYPE(END) end;
};
```

[10] https://github.com/mirage/xen/blob/ee92c92/tools/libxc/xc_dom_x86.c#L948

© NCC Group
nccgroup

Register I/O — HVM Read & Write

- Use `xc_domain_hvm_{get,set}context()`
- Context is a series of header-prefixed blocks of binary data **of variable total length**
- No way to craft valid HEADER data or get it via `_getcontext_partial()`[11]

```
int xc_domain_hvm_setcontext(
    xc_interface *xch,
    uint32_t domid,
    uint8_t *hvm_ctxt,
    uint32_t size);

int xc_domain_hvm_getcontext(
    xc_interface *xch,
    uint32_t domid,
    uint8_t *ctxt_buf,
    uint32_t size);
```

```
struct hvm_save_descriptor {
    uint16_t typecode; // HVM_SAVE_CODE(...)
    uint16_t instance; // VCPU ID
    uint32_t length;   // num bytes following
};
struct cpu_context_update { // User-defined example
    struct hvm_save_descriptor header_d;
    HVM_SAVE_TYPE(HEADER) header;
    struct hvm_save_descriptor cpu_d;
    HVM_SAVE_TYPE(CPU) cpu;
    struct hvm_save_descriptor end_d;
    HVM_SAVE_TYPE(END) end;
};
```

[11] https://github.com/mirage/xen/blob/ee92c92/tools/libxc/xc_dom_x86.c#L948

© NCC Group

Register I/O — HVM Read & Write

- Use `xc_domain_hvm_{get,set}context()`
- Context is a series of header-prefixed blocks of binary data **of variable total length**
- No way to craft valid HEADER data or get it via `_getcontext_partial()`[12]

```
int xc_domain_hvm_setcontext(
    xc_interface *xch,
    uint32_t domid,
    uint8_t *hvm_ctxt,
    uint32_t size);

int xc_domain_hvm_getcontext(
    xc_interface *xch,
    uint32_t domid,
    uint8_t *ctxt_buf,
    uint32_t size);
```

```
struct hvm_save_descriptor {
    uint16_t typecode; // HVM_SAVE_CODE(...)
    uint16_t instance; // VCPU ID
    uint32_t length;   // num bytes following
};
struct cpu_context_update { // User-defined example
    struct hvm_save_descriptor header_d;
    HVM_SAVE_TYPE(HEADER) header;
    struct hvm_save_descriptor cpu_d;
    HVM_SAVE_TYPE(CPU) cpu;
    struct hvm_save_descriptor end_d;
    HVM_SAVE_TYPE(END) end;
};
```

[12] https://github.com/mirage/xen/blob/ee92c92/tools/libxc/xc_dom_x86.c#L948

© NCC

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```c
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```c
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
         offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
         offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
         offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
         offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
         offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
         offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
         offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Register I/O — HVM Read & Write Example

- So to write regs, we have to get the entire context anyway just for the header
- `_getcontext()` works like `snprintf()`: if buffer/size is NULL/0, return min buf size needed

```c
// Get required size
uint32_t size =
  xc_domain_hvm_getcontext(
    xch, domid, NULL, 0);

// Alloc buffer
uint8_t *ctx =
    (uint8_t*)calloc(1, size);

// Get the current register state
xc_domain_hvm_getcontext(xch, domid,
    &ctx, size);
```

```c
uint32_t offset = 0;
do {
  struct hvm_save_descriptor *desc =
      (struct hvm_save_descriptor *)(ctx + offset);
  offset += sizeof(struct hvm_save_descriptor);
  if (desc->typecode == HVM_SAVE_CODE(CPU)) {
    HVM_SAVE_TYPE(CPU) *cpu =
        (HVM_SAVE_TYPE(CPU)*)(ctx+offset);
    cpu->rip = 0xDEADBEEF;
  }
  offset += desc->length;
} while (desc->typecode != HVM_SAVE_CODE(END) &&
          offset < size);
xc_domain_hvm_setcontext(xch, domid, &ctx, size);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Advanced VMI on PV

Debugging Mode

PV VMI — Debugging mode

- PV only has a kludgy “debugging mode”
- Makes the guest pause instead of crash on INT3 (“trap to debugger”) instructions [13]
- Enable it before you do any debugging operations!

```
int xc_domain_setdebugging(
    xc_interface *xch,
    uint32_t domid,
    unsigned int enable);
```

[13] https://github.com/xen-project/xen/blob/0ca7624/xen/include/asm-x86/debugger.h#L62

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

PV VMI — Single step

- Write to the `rflags` (x64) or `eflags` (x32) register of the VCPU you want to single-step
- Set the x86 trap flag `0x100`
- This makes the VCPU trap on every instruction

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

PV VMI — Software breakpoints

- Just write an INT3 (`0xCC`) over an instruction
- Caveat: PV guests pause on the instruction *after* the breakpoint
  - Set `rip -= 1` after a breakpoint is hit
- Remember to save the byte that was overwritten to restore later!
- To continue past a BP,
  - Replace the original byte
  - Single-step once
  - Overwrite with 0xCC again
  - Unpause the guest

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

PV VMI — Checking VCPU pause status

- Poll VCPU pause state using the XEN_DOMCTL_gdbsx_domstatus domctl hypercall

```
DECLARE_HYPERCALL_BUFFER(xen_domctl, domctl);
xen_domctl *domctl = (xen_domctl*)(xc_hypercall_buffer_alloc(
    xenctrl_handle, domctl, sizeof(*domctl)));

domctl->domain = domid;
domctl->interface_version = XEN_DOMCTL_INTERFACE_VERSION;
domctl->cmd = XEN_DOMCTL_gdbsx_domstatus;
xencall1(xencall_handle, __HYPERVISOR_domctl,
    HYPERCALL_BUFFER_AS_ARG(domctl));

if (domctl->u.gdbsx_domstatus.paused)                  // 1 if the guest is paused
  printf("VCPU ID %zu is paused\n",
         domctl->u.gdbsx_domstatus.vcpu_id);           // The ID of the VCPU that paused

xc_hypercall_buffer_free(_xenctrl.get(), domctl);  // Remember to free the buffer!
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

PV VMI — Checking VCPU pause status

- Poll VCPU pause state using the XEN_DOMCTL_gdbsx_domstatus domctl hypercall

```
DECLARE_HYPERCALL_BUFFER(xen_domctl, domctl);
xen_domctl *domctl = (xen_domctl*)(xc_hypercall_buffer_alloc(
    xenctrl_handle, domctl, sizeof(*domctl)));

domctl->domain = domid;
domctl->interface_version = XEN_DOMCTL_INTERFACE_VERSION;
domctl->cmd = XEN_DOMCTL_gdbsx_domstatus;
xencall1(xencall_handle, __HYPERVISOR_domctl,
    HYPERCALL_BUFFER_AS_ARG(domctl));

if (domctl->u.gdbsx_domstatus.paused)              // 1 if the guest is paused
  printf("VCPU ID %zu is paused\n",
         domctl->u.gdbsx_domstatus.vcpu_id);       // The ID of the VCPU that paused

xc_hypercall_buffer_free(_xenctrl.get(), domctl);  // Remember to free the buffer!
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

PV VMI — Checking VCPU pause status

- Poll VCPU pause state using the `XEN_DOMCTL_gdbsx_domstatus` domctl hypercall

```
DECLARE_HYPERCALL_BUFFER(xen_domctl, domctl);
xen_domctl *domctl = (xen_domctl*)(xc_hypercall_buffer_alloc(
    xenctrl_handle, domctl, sizeof(*domctl)));

domctl->domain = domid;
domctl->interface_version = XEN_DOMCTL_INTERFACE_VERSION;
domctl->cmd = XEN_DOMCTL_gdbsx_domstatus;
xencall1(xencall_handle, __HYPERVISOR_domctl,
    HYPERCALL_BUFFER_AS_ARG(domctl));

if (domctl->u.gdbsx_domstatus.paused)              // 1 if the guest is paused
  printf("VCPU ID %zu is paused\n",
         domctl->u.gdbsx_domstatus.vcpu_id);       // The ID of the VCPU that paused

xc_hypercall_buffer_free(_xenctrl.get(), domctl);  // Remember to free the buffer!
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

PV VMI — Checking VCPU pause status

- Poll VCPU pause state using the `XEN_DOMCTL_gdbsx_domstatus` domctl hypercall

```
DECLARE_HYPERCALL_BUFFER(xen_domctl, domctl);
xen_domctl *domctl = (xen_domctl*)(xc_hypercall_buffer_alloc(
    xenctrl_handle, domctl, sizeof(*domctl)));

domctl->domain = domid;
domctl->interface_version = XEN_DOMCTL_INTERFACE_VERSION;
domctl->cmd = XEN_DOMCTL_gdbsx_domstatus;
xencall1(xencall_handle, __HYPERVISOR_domctl,
    HYPERCALL_BUFFER_AS_ARG(domctl));

if (domctl->u.gdbsx_domstatus.paused)                // 1 if the guest is paused
  printf("VCPU ID %zu is paused\n",
         domctl->u.gdbsx_domstatus.vcpu_id);        // The ID of the VCPU that paused

xc_hypercall_buffer_free(_xenctrl.get(), domctl);  // Remember to free the buffer!
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

PV VMI — Checking VCPU pause status

- Poll VCPU pause state using the `XEN_DOMCTL_gdbsx_domstatus` domctl hypercall

```
DECLARE_HYPERCALL_BUFFER(xen_domctl, domctl);
xen_domctl *domctl = (xen_domctl*)(xc_hypercall_buffer_alloc(
    xenctrl_handle, domctl, sizeof(*domctl)));

domctl->domain = domid;
domctl->interface_version = XEN_DOMCTL_INTERFACE_VERSION;
domctl->cmd = XEN_DOMCTL_gdbsx_domstatus;
xencall1(xencall_handle, __HYPERVISOR_domctl,
    HYPERCALL_BUFFER_AS_ARG(domctl));

if (domctl->u.gdbsx_domstatus.paused)             // 1 if the guest is paused
  printf("VCPU ID %zu is paused\n",
         domctl->u.gdbsx_domstatus.vcpu_id);       // The ID of the VCPU that paused

xc_hypercall_buffer_free(_xenctrl.get(), domctl);  // Remember to free the buffer!
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

PV VMI — Checking VCPU pause status

- Poll VCPU pause state using the XEN_DOMCTL_gdbsx_domstatus domctl hypercall

```
DECLARE_HYPERCALL_BUFFER(xen_domctl, domctl);
xen_domctl *domctl = (xen_domctl*)(xc_hypercall_buffer_alloc(
    xenctrl_handle, domctl, sizeof(*domctl)));

domctl->domain = domid;
domctl->interface_version = XEN_DOMCTL_INTERFACE_VERSION;
domctl->cmd = XEN_DOMCTL_gdbsx_domstatus;
xencall1(xencall_handle, __HYPERVISOR_domctl,
    HYPERCALL_BUFFER_AS_ARG(domctl));

if (domctl->u.gdbsx_domstatus.paused)            // 1 if the guest is paused
  printf("VCPU ID %zu is paused\n",
         domctl->u.gdbsx_domstatus.vcpu_id);       // The ID of the VCPU that paused

xc_hypercall_buffer_free(_xenctrl.get(), domctl);  // Remember to free the buffer!
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

PV VMI — That’s all, folks

- …and that’s it.
- PV VMI is *very* limited.

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

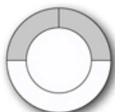
Advanced VMI on HVM

The Event API

HVM VMI — The Event API

- Dom0 maps a ring buffer page, shared with the guest
  - Guest emplaces “requests” (event notifications); Dom0 ACKs with “responses”

DomU Writes Request 1


DomU Writes Request 2

Dom0 Writes Response 1

Dom0 Reads Response 1

Dom0 Writes Response 2

Dom0 Reads Response 2 [14]

[14] http://www.informit.com/articles/article.aspx?p=1160234&seqNum=3

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Ring buffer setup

- Map the ring buffer via `xc_monitor_enable()`
  - **Note**: The monitor can only be active for one VMI client at a time
- Returns the address of the shared ring buffer mapped in Dom0
- Also provides a remote port number associated with the guest
- Unmap when done; use `xc_monitor_disable()`

```
void *xc_monitor_enable(
    xc_interface *xch,
    domid_t domain_id,
    uint32_t *port);      /* Remote port will be written to `port` */

int xc_monitor_disable(
    xc_interface *xch,
    domid_t domain_id);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Ring buffer setup

- Pass the remote port to `xenevtchn_bind_interdomain()`
- Will return a local port number
  - Pending events are batched by port number
  - Each port number associates with a domain
- Unbind when done; use `xenevtchn_unbind()`

```
xenevtchn_port_or_error_t           /* int */
xenevtchn_bind_interdomain(
    xenevtchn_handle *xce,
    uint32_t domid,
    evtchn_port_t remote_port);     /* uint32_t */

int xenevtchn_unbind(
    xenevtchn_handle *xce,
    evtchn_port_t port);            /* The local port */
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events  ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE,     1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET:  2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE,     1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET: 2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE,      1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET: 2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE, 1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET: 2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events  ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE,     1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET: 2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE,     1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET: 2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE,     1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET:  2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Ring buffer setup example

```
uint32_t remote_port;
vm_event_sring_t *ring_page =
    (vm_event_sring_t*)xc_monitor_enable(xenctrl_handle, domid, &remote_port);

uint32_t local_port = xenevtchn_bind_interdomain(xenevtchn_handle, domid, remote_port);

vm_event_back_ring_t back_ring;
SHARED_RING_INIT(ring_page);
BACK_RING_INIT(&back_ring, ring_page, XC_PAGE_SIZE);

/* ... Enable desired events  ... */
/* ... Read from the buffer in a loop ... */

xenevtchn_unbind(xenevtchn_handle, local_port); // WHEN DONE,     1. Release the port
xc_monitor_disable(xenctrl_handle, domid);      // DON'T FORGET:  2. Disable the monitor
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Monitoring event types

- Check supported event types with `xc_monitor_get_capabilities()`[15]
- Enable each event type via `xc_monitor_$EVENT()`
- A given event type will only be emitted if you enable monitoring for it!

```
uint32_t capabilities;
xc_monitor_get_capabilities(xenctrl_handle, domid, &capabilities);

if (capabilities & VM_EVENT_REASON_SOFTWARE_BREAKPOINT)
  xc_monitor_software_breakpoint(xenctrl_handle, domid, 1); // 0 to disable
```

[15] See the list of event reason flags on the next slide

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Monitoring event types

- Check supported event types with `xc_monitor_get_capabilities()`[16]
- Enable each event type via `xc_monitor_$EVENT()`
- A given event type will only be emitted if you enable monitoring for it!

```
uint32_t capabilities;
xc_monitor_get_capabilities(xenctrl_handle, domid, &capabilities);

if (capabilities & VM_EVENT_REASON_SOFTWARE_BREAKPOINT)
  xc_monitor_software_breakpoint(xenctrl_handle, domid, 1); // 0 to disable
```

[16] See the list of event reason flags on the next slide

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Monitoring event types

- Check supported event types with `xc_monitor_get_capabilities()`[17]
- Enable each event type via `xc_monitor_$EVENT()`
- A given event type will only be emitted if you enable monitoring for it!

```
uint32_t capabilities;
xc_monitor_get_capabilities(xenctrl_handle, domid, &capabilities);

if (capabilities & VM_EVENT_REASON_SOFTWARE_BREAKPOINT)
  xc_monitor_software_breakpoint(xenctrl_handle, domid, 1); // 0 to disable
```

[17] See the list of event reason flags on the next slide

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Monitoring event types

- Check supported event types with `xc_monitor_get_capabilities()`[18]
- Enable each event type via `xc_monitor_$EVENT()`
- A given event type will only be emitted if you enable monitoring for it!

```
uint32_t capabilities;
xc_monitor_get_capabilities(xenctrl_handle, domid, &capabilities);

if (capabilities & VM_EVENT_REASON_SOFTWARE_BREAKPOINT)
  xc_monitor_software_breakpoint(xenctrl_handle, domid, 1); // 0 to disable
```

[18] See the list of event reason flags on the next slide

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — VM_EVENT_REASON_* flags

- CPUID: CPUID executed
- DEBUG_EXCEPTION: A debug exception was caught
- GUEST_REQUEST: An event has been requested via HVMOP_guest_request_vm_event
- MEM_ACCESS: Memory access violation
- MEM_PAGING: Memory paging event
- MEM_SHARING: Memory sharing event
- MOV_TO_MSR: An MSR was updated
- SINGLESTEP: Single-step (e.g. monitor trap flag)
- SOFTWARE_BREAKPOINT: Debug operation executed (e.g. int3)
- WRITE_CTRLREG: A control register was updated

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — VM_EVENT_REASON_* flags

- CPUID: CPUID executed
- DEBUG_EXCEPTION: A debug exception was caught
- GUEST_REQUEST: An event has been requested via HVMOP_guest_request_vm_event
- MEM_ACCESS: Memory access violation
- MEM_PAGING: Memory paging event
- MEM_SHARING: Memory sharing event
- MOV_TO_MSR: An MSR was updated
- SINGLESTEP: Single-step (e.g. monitor trap flag)
- SOFTWARE_BREAKPOINT: Debug operation executed (e.g. int3)
- WRITE_CTRLREG: A control register was updated

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Reading from the ring buffer

- Dom0 blocks on a file descriptor (`xenevtchn_fd()`) to wait for events — no polling!
- `xenevtchn_pending()` returns the local port of the next event
- Once finished reading events, release the channel with `xenevtchn_unmask()`
  - This lets event notifications to Dom0 continue.

```
int fd = xenevtchn_fd(xenevtchn_handle);

while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  if (xenevtchn_pending(xenevtchn_handle) == local_port)
    read_events();
  xenevtchn_unmask(xenevtchn_handle, local_port);
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Reading from the ring buffer

- Dom0 blocks on a file descriptor (`xenevtchn_fd()`) to wait for events — no polling!
- `xenevtchn_pending()` returns the local port of the next event
- Once finished reading events, release the channel with `xenevtchn_unmask()`
  - This lets event notifications to Dom0 continue

```
int fd = xenevtchn_fd(xenevtchn_handle);

while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  if (xenevtchn_pending(xenevtchn_handle) == local_port)
    read_events();
  xenevtchn_unmask(xenevtchn_handle, local_port);
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading from the ring buffer

- Dom0 blocks on a file descriptor (`xenevtchn_fd()`) to wait for events — no polling!
- `xenevtchn_pending()` returns the local port of the next event
- Once finished reading events, release the channel with `xenevtchn_unmask()`
  - This lets event notifications to Dom0 continue

```
int fd = xenevtchn_fd(xenevtchn_handle);

while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  if (xenevtchn_pending(xenevtchn_handle) == local_port)
    read_events();
  xenevtchn_unmask(xenevtchn_handle, local_port);
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Reading from the ring buffer

- Dom0 blocks on a file descriptor (`xenevtchn_fd()`) to wait for events — no polling!
- `xenevtchn_pending()` returns the local port of the next event
- Once finished reading events, release the channel with `xenevtchn_unmask()`
  - This lets event notifications to Dom0 continue

```
int fd = xenevtchn_fd(xenevtchn_handle);

while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  if (xenevtchn_pending(xenevtchn_handle) == local_port)
    read_events();
  xenevtchn_unmask(xenevtchn_handle, local_port);
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Reading from the ring buffer

- Dom0 blocks on a file descriptor (`xenevtchn_fd()`) to wait for events — no polling!
- `xenevtchn_pending()` returns the local port of the next event
- Once finished reading events, release the channel with `xenevtchn_unmask()`
  - This lets event notifications to Dom0 continue

```
int fd = xenevtchn_fd(xenevtchn_handle);

while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  if (xenevtchn_pending(xenevtchn_handle) == local_port)
    read_events();
  xenevtchn_unmask(xenevtchn_handle, local_port);
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Reading from the ring buffer

- Dom0 blocks on a file descriptor (`xenevtchn_fd()`) to wait for events — no polling!
- `xenevtchn_pending()` returns the local port of the next event
- Once finished reading events, release the channel with `xenevtchn_unmask()`
  - This lets event notifications to Dom0 continue

```
int fd = xenevtchn_fd(xenevtchn_handle);

while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  if (xenevtchn_pending(xenevtchn_handle) == local_port)
    read_events();
  xenevtchn_unmask(xenevtchn_handle, local_port);
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                     /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                     /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                       /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                     /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                      /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading events

```c
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                     /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                      /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(rsp, &back_ring);                     /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — Reading events

```
while (RING_HAS_UNCONSUMED_REQUESTS(&back_ring)) {
  vm_event_request_t req = get_request(&back_ring);  /* `get` boilerplate on next slide */
  if (req.version != VM_EVENT_INTERFACE_VERSION)
    die("version mismatch");

  switch (req.reason) {
    case VM_EVENT_REASON_SINGLE_STEP:
      /* ... Read req.u.single_step ... */
  }

  vm_event_response_t rsp;
  memset(&rsp, 0, sizeof(rsp));
  rsp.reason = req.reason;
  rsp.version = VM_EVENT_INTERFACE_VERSION;
  rsp.vcpu_id = req.vcpu_id;
  rsp.flags = req.flags & VM_EVENT_FLAG_VCPU_PAUSED; // VCPU auto-unpauses if this is set
  put_response(&back_ring, rsp);                     /* `put` boilerplate on next slide */
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM Event API — `get_request()` & `put_response()` boilerplate

```
vm_event_request_t get_request(
    vm_event_back_ring_t *back_ring) {
  vm_event_request_t req;
  RING_IDX req_cons = back_ring->req_cons;

  // Copy the request out of the ring
  memcpy(&req,
    RING_GET_REQUEST(back_ring,
      req_cons), sizeof(req));
  req_cons++;

  // Update the ring position
  back_ring->req_cons = req_cons;
  back_ring->sring->req_event = req_cons+1;

  return req;
}
```

```
void put_response(vm_event_response_t rsp,
  vm_event_back_ring_t *back_ring) {
  RING_IDX rsp_prod = back_ring->rsp_prod_pvt;

  // Copy response
  memcpy(RING_GET_RESPONSE(back_ring,
      rsp_prod), &rsp, sizeof(rsp));
  rsp_prod++;

  // Update ring
  back_ring->rsp_prod_pvt = rsp_prod;
  RING_PUSH_RESPONSES(back_ring);
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — The Request/Response Struct

- Requests and responses use the same type, `struct vm_event_st`
- You can get a lot of data out of requests, and perform ops via responses!

```
typedef struct vm_event_st {
  uint32_t version;   /* VM_EVENT_INTERFACE_VERSION */
  uint32_t flags;     /* VM_EVENT_FLAG_* */
  uint32_t reason;    /* VM_EVENT_REASON_* */
  uint32_t vcpu_id;
  uint16_t altp2m_idx; /* may be used during request and response */
  uint16_t _pad[3];

  union { /* ... event structs ... */ } u;
  union { /* ... register & other data */ } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — The Request/Response Struct

- `version` represents the API version of the request
- Check against `VM_EVENT_INTERFACE_VERSION`

```
typedef struct vm_event_st {
  uint32_t version;   /* VM_EVENT_INTERFACE_VERSION */
  uint32_t flags;     /* VM_EVENT_FLAG_* */
  uint32_t reason;    /* VM_EVENT_REASON_* */
  uint32_t vcpu_id;
  uint16_t altp2m_idx; /* may be used during request and response */
  uint16_t _pad[3];

  union { /* ... event structs ... */ } u;
  union { /* ... register & other data */ } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — The Request/Response Struct

- `flags` carries request/response metadata
- Certain flags affect the guest when sent in responses

```
typedef struct vm_event_st {
  uint32_t version;   /* VM_EVENT_INTERFACE_VERSION */
  uint32_t flags;     /* VM_EVENT_FLAG_* */
  uint32_t reason;    /* VM_EVENT_REASON_* */
  uint32_t vcpu_id;
  uint16_t altp2m_idx; /* may be used during request and response */
  uint16_t _pad[3];

  union { /* ... event structs ... */ } u;
  union { /* ... register & other data */ } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM VMI — VM_EVENT_FLAG_* Flags

- Event flags are well-documented in header comments[19]
- **VCPU_PAUSED**: In req, means VCPU was paused. In resp, unpauses VCPU.
- **TOGGLE_SINGLESTEP**: Toggles singlestep. Requires the VCPU to be paused.
- **SET_REGISTERS**: Set registers to the values in `data.regs`.
- **EMULATE**: Emulate the fault-causing instruction
- **EMULATE_NOWRITE**: Same as EMULATE, plus disables operation side effects
- **SET_EMUL_READ_DATA**: Fake data is being provided to “read” in an emulated instruction.
- **SET_EMUL_INSN_DATA**: A fake instruction cache is bring provided.
- **ALTERNATE_P2M**: In req, means event occurred in alt p2m specified by altp2m_idx. In resp, means VCPU should resume in the specified alternate p2m.
- **DENY**: Deny completion of the operation that triggered the event.

[19] https://github.com/xen-project/xen/blob/0ca7624/xen/include/public/vm_event.h#L42

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — The Request/Response Struct

- `reason` specifies the reason the event occurred

```
typedef struct vm_event_st {
  uint32_t version;   /* VM_EVENT_INTERFACE_VERSION */
  uint32_t flags;     /* VM_EVENT_FLAG_* */
  uint32_t reason;    /* VM_EVENT_REASON_* */
  uint32_t vcpu_id;
  uint16_t altp2m_idx; /* may be used during request and response */
  uint16_t _pad[3];

  union { /* ... event structs ... */ } u;
  union { /* ... register & other data */ } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — The Request/Response Struct

- `vcpu_id` specifies the VCPU on which the event occurred

```
typedef struct vm_event_st {
  uint32_t version;   /* VM_EVENT_INTERFACE_VERSION */
  uint32_t flags;     /* VM_EVENT_FLAG_* */
  uint32_t reason;    /* VM_EVENT_REASON_* */
  uint32_t vcpu_id;
  uint16_t altp2m_idx; /* may be used during request and response */
  uint16_t _pad[3];

  union { /* ... event structs ... */ } u;
  union { /* ... register & other data */ } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM VMI — The Request/Response Struct

- `altp2m_idx` is used when the `ALTERNATE_P2M` flag is set

```
typedef struct vm_event_st {
  uint32_t version;   /* VM_EVENT_INTERFACE_VERSION */
  uint32_t flags;     /* VM_EVENT_FLAG_* */
  uint32_t reason;    /* VM_EVENT_REASON_* */
  uint32_t vcpu_id;
  uint16_t altp2m_idx; /* may be used during request and response */
  uint16_t _pad[3];

  union { /* ... event structs ... */ } u;
  union { /* ... register & other data */ } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — The Request/Response Struct

- The u union holds structs for event-specific data, corresponding to VM_EVENT_REASON_*

```
typedef struct vm_event_st {
  /* ... omitted ... */

  union {
      struct vm_event_singlestep singlestep;
      struct vm_event_debug      software_breakpoint;
      struct vm_event_mem_access mem_access;
      /* ... etc ... */
  } u;

  union { /* ... register & other data */ } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — The Request/Response Struct

- In requests, `data.regs` holds the register state when the event occurred
- In responses, the `flags` determines which member of `data` Xen uses, if any

```
typedef struct vm_event_st {
  /* ... omitted ... */
  union { /* ... event structs ... */ } u;
  union {
    union {
      struct vm_event_regs_x86 x86;        // Used when SET_REGISTERS flag is set,
      struct vm_event_regs_arm arm;        // and stores register state in requests
    } regs;
    union {
      struct vm_event_emul_read_data read; // Used when SET_EMUL_READ_DATA is set
      struct vm_event_emul_insn_data insn; // Used when SET_EMUL_INSN_DATA is set
    } emul;
  } data;
} vm_event_request_t, vm_event_response_t;
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — A note on pausing

- `req.flags & VM_EVENT_FLAG_VCPU_PAUSED` means the VCPU has been paused
- If this flag is passed in the response, Xen will unpause the VCPU
- VCPU pausing is reference-counted
- To stay paused, pause the VCPU in the request handling code, before sending the response

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — A note on pausing

- Pause VCPUs with the `XEN_DOMCTL_gdbsx_pausevcpu` domctl hypercall
- Unpause with `XEN_DOMCTL_gdbsx_unpausevcpu`
    - In both cases, set `domctl.u.gdbsx_pauseunp_vcpu.vcpu = <VCPU_ID>`
    - There's no modern API to do this (that I know of)
- **NOTE:** Before pausing a VCPU, the domain itself must be paused via `xc_domain_pause()`

```
int xc_domain_pause(
    xc_interface *xch,
    uint32_t domid);
```

```
int xc_domain_unpause(
    xc_interface *xch,
    uint32_t domid);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM Event API — A note on pausing

- If you pause the whole domain (`xc_domain_pause()`), everything stops
  - This is what Xen uses when you run `xl pause`
  - Separate "layer" of pausing, distinct from VCPUs
- If you pause the domain & one or more VCPUs, then unpause the domain…
- …the non-paused VCPUs will keep running!

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM VMI — Single-step

- Listen with `xc_monitor_singlestep()`
- Enable single-step mode with `xc_domain_debug_control()`
- `VM_EVENT_REASON_SINGLESTEP` events will be fired on each instruction

```
int xc_domain_debug_control(
    xc_interface *xch,
    uint32_t domid,
    uint32_t sop,     /* Debug control operation --- only 2 for now */
    uint32_t vcpu);

// Enable:  sop = XEN_DOMCTL_DEBUG_OP_SINGLE_STEP_ON
// Disable: sop = XEN_DOMCTL_DEBUG_OP_SINGLE_STEP_OFF
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM VMI — Software breakpoints

- Listen with `xc_monitor_software_breakpoint()`
- Write INT3 (0xCC) over an instruction
- `VM_EVENT_REASON_SOFTWARE_BREAKPOINT` events will be fired when INT3s are hit
  - **NOTE**: `rip` will be left pointing to the INT3, **not** after it like in PV
- Continue past BP: same as PV

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — Emulated memory watchpoints

- **Note**: Watchpoints are software-based, memory-only
  - Can't do register WPs in general; only control regs via `VM_EVENT_REASON_WRITE_CTRLREG`
- Listen with `xc_monitor_mem_access()`
- `VM_EVENT_REASON_MEM_ACCESS` events will be fired…
  - …but only on *failed* memory access attempts

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — Emulated memory watchpoints

- Use `xc_set_mem_access()` to set permissions on the target memory region
  - Access must **fail** on the operation we want to listen for (read, write, etc.)[20]
- Save original permissions to restore later; get with `xc_get_mem_access()`
- Use `xc_translate_foreign_address()` to translate the guest virtual addr to a PFN

```
int xc_set_mem_access(
  xc_interface *xch,
  domid_t domain_id,
  xenmem_access_t access, // Permissions
  uint64_t first_pfn,     // Page frame number
  uint32_t nr);           // Number of pages

// All fail:    XENMEM_access_n;
// Reads fail:  XENMEM_access_wx;
// Writes fail: XENMEM_access_rx;
```

```
int xc_get_mem_access(
    xc_interface *xch,
    domid_t domain_id,
    uint64_t pfn,
    xenmem_access_t *access);
```

[20] https://github.com/xen-project/xen/blob/0ca7624/xen/include/public/memory.h#L419

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — Emulated memory watchpoints

- Notable `MEM_ACCESS` event members:
  - `.gla`: Guest virtual address of the failed access*
  - `.flags`: Metadata including type(s) of access attmpted (`MEM_ACCESS_{R,W,X}`)[21]
- **NOTE**: `gla` only contains the guest VA if `flags` includes `MEM_ACCESS_GLA_VALID`. Otherwise, see:
  - `.gfn`: Guest frame number of the failed access
  - `.offset`: Offset into the page, in bytes

[21] https://github.com/xen-project/xen/blob/0ca7624/xen/include/public/vm_event.h#L239

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — Emulated memory watchpoints

- To remove the WP, use `xc_set_mem_access()` to restore the original permissions
- To retain it while continuing,
  - Restore the region's old permissions
  - Single-step once
  - Put the watchpoint (fail on access) permissions back
  - Unpause the guest

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — Trap injection

- The event API will consume faults/traps associated with monitored events
  - So guests **will not crash** as a result
- But what if you get a *real* memory access violation?
  - i.e. one not associated with one of your watchpoint regions

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

HVM VMI — Trap injection

- Use `xendevicemodel_inject_event()` to inject traps back into the guest
- This will force the guest to handle the trap (i.e. crash)

```
int xendevicemodel_inject_event(
    xendevicemodel_handle *dmod,
    domid_t domid,
    int vcpu,
    uint8_t vector,       // The interrupt vector
    uint8_t type,         // The event type (see the definition of enum x86_event_type)
    uint32_t error_code,  // The error code or ~0 to skip
    uint8_t insn_len,     // The instruction length
    uint64_t cr2);        // The value of CR2 for page faults
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

HVM VMI — Trap injection example

```
if (req.reason == VM_EVENT_REASON_SOFTWARE_BREAKPOINT) {
  xendevicemodel_inject_event(
      dmod,
      domain_id,
      req.vcpu_id,
      X86_TRAP_INT3,                        // Interrupt vector
      req.u.software_breakpoint.type,       // Event type
      ~0,                                   // Error code (skip)
      req.u.software_breakpoint.insn_length, // Instruction length
      0);                                   // Value of CR2 (not used for BPs)
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

VMI Extras

XenStore & Watches

Extras — XenStore

- A filesystem-like transactional DB that holds domain metadata
- Read node (directory) contents with `xs_directory()`
- Read leaf (file) contents with `xs_read()`
- **NOTE**: You are responsible for freeing the returned arrays!

```
char **xs_directory(
    struct xs_handle *h,
    xs_transaction_t t, // transaction
    const char *path,
    unsigned int *num); // num. of entries
```

```
void *xs_read(
    struct xs_handle *h,
    xs_transaction_t t, // transaction
    const char *path,
    unsigned int *len); // length of data
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore

- Manage transactions with `xs_transaction_{start,end}()`
- **NOTE**: Changes are not propagated until `_end()` is called!
- For `end()`, if `abort` is true, the transaction will be discarded, else committed

```
xs_transaction_t xs_transaction_start(
    struct xs_handle *h);
```

```
bool xs_transaction_end(
    struct xs_handle *h,
    xs_transaction_t t,
    bool abort);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore example: listing domain names

```
xs_transaction_t t = xs_transaction_start(xs_handle);

unsigned num_files;
char **contents = xs_directory(xs_handle, t, "/local/domains", &num_files);

char *domid_str, path[128];
while (domid_str = *contents++) {
  snprintf(&path, 128, "/local/domains/%s/name", contents[i]);

  unsigned len;
  char *name = (char*)xs_read(xs_handle, t, path, &len);
  printf("DomID %s is named '%s'\\n", domid_str, name);
  free(name);
}

xs_transaction_end(xs_handle, t, false);
free(contents);
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore watches

- Can use `xs_watch()` to listen for changes to the XenStore DB
- Pass a XenStore path, or one of two special path strings, triggered when…
  - `@introduceDomain`: a new domain is created
  - `@releaseDomain`: an existing domain is destroyed
- **Note**: `path` and `token` are copied; compare by value, not by pointer.

```
bool xs_watch(
    struct xs_handle *h,
    const char *path,     // "Filesystem" path to watch
    const char *token);   // User-defined unique identifier
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore watches

- Wait on XenStore's FD (`xs_fileno()`) for watch events
- Once data is available, call `xs_check_watch()` repeatedly until it returns NULL
- Each return value is a `(path, token)` pair indicating a node change

```
int fd = xs_fileno(xenstore_handle);
while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  char **pair;
  while ((pair = xs_check_watch(xenstore_handle))) {
    char *path = pair[0];
    char *token = pair[1];
    printf("Path '%s' (token '%s') changed\n", path, token);
  }
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Extras — XenStore watches

- Wait on XenStore's FD (`xs_fileno()`) for watch events
- Once data is available, call `xs_check_watch()` repeatedly until it returns `NULL`
- Each return value is a `(path, token)` pair indicating a node change

```
int fd = xs_fileno(xenstore_handle);
while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  char **pair;
  while ((pair = xs_check_watch(xenstore_handle))) {
    char *path = pair[0];
    char *token = pair[1];
    printf("Path '%s' (token '%s') changed\n", path, token);
  }
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup

Extras — XenStore watches

- Wait on XenStore's FD (`xs_fileno()`) for watch events
- Once data is available, call `xs_check_watch()` repeatedly until it returns `NULL`
- Each return value is a `(path, token)` pair indicating a node change

```
int fd = xs_fileno(xenstore_handle);
while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  char **pair;
  while ((pair = xs_check_watch(xenstore_handle))) {
    char *path = pair[0];
    char *token = pair[1];
    printf("Path '%s' (token '%s') changed\n", path, token);
  }
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore watches

- Wait on XenStore's FD (`xs_fileno()`) for watch events
- Once data is available, call `xs_check_watch()` repeatedly until it returns `NULL`
- Each return value is a `(path, token)` pair indicating a node change

```
int fd = xs_fileno(xenstore_handle);
while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  char **pair;
  while ((pair = xs_check_watch(xenstore_handle))) {
    char *path = pair[0];
    char *token = pair[1];
    printf("Path '%s' (token '%s') changed\n", path, token);
  }
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore watches

- Wait on XenStore's FD (`xs_fileno()`) for watch events
- Once data is available, call `xs_check_watch()` repeatedly until it returns `NULL`
- Each return value is a `(path, token)` pair indicating a node change

```
int fd = xs_fileno(xenstore_handle);
while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  char **pair;
  while ((pair = xs_check_watch(xenstore_handle))) {
    char *path = pair[0];
    char *token = pair[1];
    printf("Path '%s' (token '%s') changed\n", path, token);
  }
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore watches

- Wait on XenStore's FD (`xs_fileno()`) for watch events
- Once data is available, call `xs_check_watch()` repeatedly until it returns `NULL`
- Each return value is a `(path, token)` pair indicating a node change

```
int fd = xs_fileno(xenstore_handle);
while (/* fd is not closed */) {
  /* ... select() `fd` and wait for data ... */
  char **pair;
  while ((pair = xs_check_watch(xenstore_handle))) {
    char *path = pair[0];
    char *token = pair[1];
    printf("Path '%s' (token '%s') changed\n", path, token);
  }
}
```

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Extras — XenStore watches

- No extra data is carried with the watch events
- To know what the newly added/deleted data is, you have to read the node
- e.g. if you get `@introduceDomain`, just re-read `"/local/domains"`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Congratulations!

Congratulations!

- You can now write your own Xen VMI debugger!
- For a copy of my slides, see `spencermichaels.net/projects` → `Talks`
- For a working debugger / MIT-licensed C++ reference implementation, check out Xendbg
  - `github.com/nccgroup/xendbg`

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

Questions?

spencer.michaels@nccgroup.com

spencermichaels.net

@sxmichaels

© NCC Group 2019. All rights reserved

Xen API Archaeology

Creating a Full-Featured VMI Debugger for the Xen Hypervisor

Spencer Michaels[1]

Xen Project Developer and Design Summit 2019

© NCC Group 2019. All rights reserved. Code snippets free for use with attribution under the MIT license.

nccgroup