Xen Test Framework

Testing from a guest's perspective

Andrew Cooper

Citrix XenServer

Thursday 13th July 2017

Views on testing

Views on testing

- Ever thought?
  - "This bugfix is so simple, it is clearly correct"

Views on testing

- Ever thought?
  - "This bugfix is so simple, it is clearly correct"
  - "I've only got time for a dev test at the moment"

Views on testing

- Ever thought?
  - "This bugfix is so simple, it is clearly correct"
  - "I've only got time for a dev test at the moment"
  - "Writing a test for that will be too hard"

Views on testing

- Ever thought?
  - "This bugfix is so simple, it is clearly correct"
  - "I've only got time for a dev test at the moment"
  - "Writing a test for that will be too hard"
- Who actually likes testing?

Views on testing

- Ever thought?
  - "This bugfix is so simple, it is clearly correct"
  - "I've only got time for a dev test at the moment"
  - "Writing a test for that will be too hard"
- Who actually likes testing?
- Testing is frequently a lower priority activity than it should be
  - Therefore, it tends not to get done
  - Especially if there is no pressure to do so

Testing is perceived as being hard

Testing is perceived as being hard

- Because it is...
  - **... without good infrastructure**

Testing is perceived as being hard

- Because it is...
  - **... without good infrastructure**
- Technical problems boil down to the fact that:
  - Writing tests is hard
  - Automating written tests is hard

Testing is perceived as being hard

- Because it is...
  - **... without good infrastructure**
- Technical problems boil down to the fact that:
  - Writing tests is hard
  - Automating written tests is hard
- The Xen Test Framework, and associated infrastructure, intend to remove these obstacles

“Writing tests is hard”

- Easy 4-step guide:

“Writing tests is hard”

- Easy 4-step guide:

```
root@testbox:~/xtf# ./make-new-test.sh mytest
```

“Writing tests is hard”

- Easy 4-step guide:

```
root@testbox:~/xtf# ./make-new-test.sh mytest
root@testbox:~/xtf# $EDITOR tests/mytest/main.c
```

“Writing tests is hard”

- Easy 4-step guide:

```
root@testbox:~/xtf# ./make-new-test.sh mytest
root@testbox:~/xtf# $EDITOR tests/mytest/main.c
root@testbox:~/xtf# make
```

“Writing tests is hard”

- Easy 4-step guide:

```
root@testbox:~/xtf# ./make-new-test.sh mytest
root@testbox:~/xtf# $EDITOR tests/mytest/main.c
root@testbox:~/xtf# make
root@testbox:~/xtf# ./xtf-runner mytest
```

"Writing tests is hard"

- Easy 4-step guide:

```
root@testbox:~/xtf# ./make-new-test.sh mytest
root@testbox:~/xtf# $EDITOR tests/mytest/main.c
root@testbox:~/xtf# make
root@testbox:~/xtf# ./xtf-runner mytest

Combined test results:
test-pv32pae-mytest                      SUCCESS
test-pv64-mytest                         SUCCESS
test-hvm32-mytest                        SUCCESS
test-hvm32pse-mytest                     SUCCESS
test-hvm32pae-mytest                     SUCCESS
test-hvm64-mytest                        SUCCESS
```

"Writing tests is hard"

```
#include <xtf.h>

const char test_title[] = "XSA-203 PoC";
bool test_needs_fep = true;

void test_main(void)
{
    asm volatile (_ASM_XEN_FEP
                  "1: vmfunc; 2:"
                  /* Ignore #UD on older Xen versions. */
                  _ASM_EXTABLE(1b, 2b)
                  :: "a" (0));

    /* If Xen is alive, it didn't hit the NULL pointer. */
    xtf_success("Success: Not vulnerable to XSA-203\n");
}
```

“Automating written tests is hard”

- Easy 2-step guide:

“Automating written tests is hard”

- Easy 2-step guide:

  Get patch to me via xen-devel

“Automating written tests is hard”

- Easy 2-step guide:

  Get patch to me via xen-devel

  ```
  me@box:~/xtf$ git push upstream master
  ```

“Automating written tests is hard”

- Easy 2-step guide:

  Get patch to me via xen-devel

  ```
  me@box:~/xtf$ git push upstream master
  ```

- New test will be picked up automatically by OSSTest
- Will start blocking pushes to master if a regression is detected

History of XTF

- XSA-106 was initially reported to XenServer (August 2014)
  - "`x86_emulate()` doesn't perform DPL checks for software exceptions"
  - These are: `int3`, `into`, `int $x` and `icebp`
  - A windows userspace PoC existed, which would cause a BSOD

History of XTF

- XSA-106 was initially reported to XenServer (August 2014)
  - "`x86_emulate()` doesn't perform DPL checks for software exceptions"
  - These are: `int3`, `into`, `int $x` and `icebp`
  - A windows userspace PoC existed, which would cause a BSOD
- On inspection:
  - No checks of the IDT Entry at all
  - Also important to check the descriptor type and present bit
  - All software events injected as hardware exceptions

History of XTF

- XSA-106 was initially reported to XenServer (August 2014)
  - "`x86_emulate()` doesn't perform DPL checks for software exceptions"
  - These are: `int3`, `into`, `int $x` and `icebp`
  - A windows userspace PoC existed, which would cause a BSOD
- On inspection:
  - No checks of the IDT Entry at all
  - Also important to check the descriptor type and present bit
  - All software events injected as hardware exceptions
- First stab at a fix changed the BSOD, but didn't fix it

History of XTF

- XSA-106 was initially reported to XenServer (August 2014)
  - "`x86_emulate()` doesn't perform DPL checks for software exceptions"
  - These are: `int3`, `into`, `int $x` and `icebp`
  - A windows userspace PoC existed, which would cause a BSOD
- On inspection:
  - No checks of the IDT Entry at all
  - Also important to check the descriptor type and present bit
  - All software events injected as hardware exceptions
- First stab at a fix changed the BSOD, but didn't fix it
- Needed an easier repro!
  - Started with `hvmloader`
  - Removed all BIOS-related bits, added an IDT
  - Borrowed the Force Emulation Prefix from PV guests
- A mess, but it did work

History of XTF

- The successful case semantics were wrong
  - Should have had Trap semantics, actually had Fault semantics
  - Fixed the test code up to detect and break infinite loops

History of XTF

- The successful case semantics were wrong
  - Should have had Trap semantics, actually had Fault semantics
  - Fixed the test code up to detect and break infinite loops
- The `icebp` instruction bypasses DPL checks
  - Also sets the External bit in error codes

History of XTF

- The successful case semantics were wrong
  - Should have had Trap semantics, actually had Fault semantics
  - Fixed the test code up to detect and break infinite loops
- The `icebp` instruction bypasses DPL checks
  - Also sets the External bit in error codes
- The first proposed fix didn't actually work on non-Intel hardware
  - Without NRIPS, AMD hardware can't correctly inject a software exception which faults for IDT-related reasons
  - With NRIPS, AMD hardware experimentally still can't inject `icebp` properly if it would fault

History of XTF

- The successful case semantics were wrong
  - Should have had Trap semantics, actually had Fault semantics
  - Fixed the test code up to detect and break infinite loops
- The `icebp` instruction bypasses DPL checks
  - Also sets the External bit in error codes
- The first proposed fix didn't actually work on non-Intel hardware
  - Without NRIPS, AMD hardware can't correctly inject a software exception which faults for IDT-related reasons
  - With NRIPS, AMD hardware experimentally still can't inject `icebp` properly if it would fault
- XSA-106 was released (September 2014)

History of XTF

- The successful case semantics were wrong
  - Should have had Trap semantics, actually had Fault semantics
  - Fixed the test code up to detect and break infinite loops
- The `icebp` instruction bypasses DPL checks
  - Also sets the External bit in error codes
- The first proposed fix didn't actually work on non-Intel hardware
  - Without NRIPS, AMD hardware can't correctly inject a software exception which faults for IDT-related reasons
  - With NRIPS, AMD hardware experimentally still can't inject `icebp` properly if it would fault
- XSA-106 was released (September 2014)
- XSA-156 was released (November 2015)
  - The fix was buggy, and caused infinite loops when the guest used `int3`
  - Should have been caught during development
  - Would have been caught if the XSA-106 code had been in automation

So what is XTF, exactly?

- A microkernel core:
  - Performs minimal setup at boot (pagetables, exceptions, console, etc)
  - Non-essential infrastructure available from the library
  - Uniform reporting mechanism (success, skip, error, failure)

So what is XTF, exactly?

- A microkernel core:
  - Performs minimal setup at boot (pagetables, exceptions, console, etc)
  - Non-essential infrastructure available from the library
  - Uniform reporting mechanism (success, skip, error, failure)
- A multi-guest build system:
  - Write one main.c, compile for any/all environments
  - PV or HVM, 32bit or 64bit, unpaged/PSE/PAE/Long mode

So what is XTF, exactly?

- A microkernel core:
  - Performs minimal setup at boot (pagetables, exceptions, console, etc)
  - Non-essential infrastructure available from the library
  - Uniform reporting mechanism (success, skip, error, failure)
- A multi-guest build system:
  - Write one main.c, compile for any/all environments
  - PV or HVM, 32bit or 64bit, unpaged/PSE/PAE/Long mode
- A utility to run one or more tests from dom0:
  - `./xtf-runner $MYTEST`
  - Scriptable

So what is XTF, exactly?

- A microkernel core:
  - Performs minimal setup at boot (pagetables, exceptions, console, etc)
  - Non-essential infrastructure available from the library
  - Uniform reporting mechanism (success, skip, error, failure)
- A multi-guest build system:
  - Write one main.c, compile for any/all environments
  - PV or HVM, 32bit or 64bit, unpaged/PSE/PAE/Long mode
- A utility to run one or more tests from dom0:
  - `./xtf-runner $MYTEST`
  - Scriptable
- A suite of tests:
  - Sorted into broad categories (functional, XSA, utilities)

So what is XTF, exactly?

- A microkernel core:
  - Performs minimal setup at boot (pagetables, exceptions, console, etc)
  - Non-essential infrastructure available from the library
  - Uniform reporting mechanism (success, skip, error, failure)
- A multi-guest build system:
  - Write one main.c, compile for any/all environments
  - PV or HVM, 32bit or 64bit, unpaged/PSE/PAE/Long mode
- A utility to run one or more tests from dom0:
  - `./xtf-runner $MYTEST`
  - Scriptable
- A suite of tests:
  - Sorted into broad categories (functional, XSA, utilities)
- Quick and easy to use:
  - `time ./xtf-runner --all --quiet`
  - Haswell test box, 7.2s for all current tests (62) to run sequentially
  - My Edit/Compile/Rerun cycle is a matter of seconds

Examples of tests

- CPUID Faulting (all environments)
  - Xen 4.8 introduced guest CPUID faulting support
  - 71 LoC (137 inc. docs), test/probe/enable/retest/disable/retest cycle
  - Many subsequent CPUID changes in Xen, made with full confidence

Examples of tests

- CPUID Faulting (all environments)
  - Xen 4.8 introduced guest CPUID faulting support
  - 71 LoC (137 inc. docs), test/probe/enable/retest/disable/retest cycle
  - Many subsequent CPUID changes in Xen, made with full confidence
- NMI Taskswitch Priv (hvm32pae)
  - Xen 4.9 regression with task switching (fixed around rc9!)
  - 93 LoC (186 inc. docs), NMI Task Gate, userspace-initiated self-NMI

Examples of tests

- CPUID Faulting (all environments)
  - Xen 4.8 introduced guest CPUID faulting support
  - 71 LoC (137 inc. docs), test/probe/enable/retest/disable/retest cycle
  - Many subsequent CPUID changes in Xen, made with full confidence
- NMI Taskswitch Priv (hvm32pae)
  - Xen 4.9 regression with task switching (fixed around rc9!)
  - 93 LoC (186 inc. docs), NMI Task Gate, userspace-initiated self-NMI
- XSA-203 CVE-2016-10025 (hvm32, shown before)
  - NULL pointer dereference with `vmfunc` emulation
  - 12 LoC (42 inc. docs)

Examples of tests

- CPUID Faulting (all environments)
  - Xen 4.8 introduced guest CPUID faulting support
  - 71 LoC (137 inc. docs), test/probe/enable/retest/disable/retest cycle
  - Many subsequent CPUID changes in Xen, made with full confidence
- NMI Taskswitch Priv (hvm32pae)
  - Xen 4.9 regression with task switching (fixed around rc9!)
  - 93 LoC (186 inc. docs), NMI Task Gate, userspace-initiated self-NMI
- XSA-203 CVE-2016-10025 (hvm32, shown before)
  - NULL pointer dereference with `vmfunc` emulation
  - 12 LoC (42 inc. docs)
- XSA-186 CVE-2016-7093 (hvm32, hvm64)
  - Bad truncation of `%eip`, underflowing the instruction cache
  - 119 LoC (241 inc. docs), 16bit code segment, executing above 64k
  - Suspected Broadwell TLB erratum

Pagetable emulation test

- Started when cleaning up the XSA-176 PoC
  - Mysteriously descheduled itself and ceased executing

Pagetable emulation test

- Started when cleaning up the XSA-176 PoC
  - Mysteriously descheduled itself and ceased executing
- Found a large number of pagetable walking bugs
  - Comprehensive test of Xen's pagewalk against real hardware
  - Skylake with PKRU, 4.2s to run, 2252800 unique pagewalks

Pagetable emulation test

- Started when cleaning up the XSA-176 PoC
  - Mysteriously descheduled itself and ceased executing
- Found a large number of pagetable walking bugs
  - Comprehensive test of Xen's pagewalk against real hardware
  - Skylake with PKRU, 4.2s to run, 2252800 unique pagewalks

```
Issues fixed:
 * 2-level PSE36 superpages now return the correct translation.
 * 2-level L2 superpages without CR0.PSE now return the correct translation.
 * SMEP now inhibits a user instruction fetch even if NX isn't active.
 * Supervisor writes without CR0.WP now set the leaf dirty bit.
 * L4e._PAGE_GLOBAL is strictly reserved on AMD.
 * 3-level l3 entries have all reserved bits checked.
 * 3-level entries can no longer alias Xen's idea of paged or shared.
```

Pagetable emulation test

- Started when cleaning up the XSA-176 PoC
  - Mysteriously descheduled itself and ceased executing
- Found a large number of pagetable walking bugs
  - Comprehensive test of Xen's pagewalk against real hardware
  - Skylake with PKRU, 4.2s to run, 2252800 unique pagewalks

```
Issues fixed:
 * 2-level PSE36 superpages now return the correct translation.
 * 2-level L2 superpages without CR0.PSE now return the correct translation.
 * SMEP now inhibits a user instruction fetch even if NX isn't active.
 * Supervisor writes without CR0.WP now set the leaf dirty bit.
 * L4e._PAGE_GLOBAL is strictly reserved on AMD.
 * 3-level l3 entries have all reserved bits checked.
 * 3-level entries can no longer alias Xen's idea of paged or shared.
```

- Still under development. Outstanding issues:
  - Xen leaks EFER.NX into guests on Intel hardware
  - AMD Zen doesn't order A/D updates with loads
  - Intel and AMD's implementation of SMAP differs

Further testing ideas

- Currently, tests are "boot microkernel, wait for it to exit"
  - Easy, and very effective for a lot of tasks

Further testing ideas

- Currently, tests are "boot microkernel, wait for it to exit"
  - Easy, and very effective for a lot of tasks
- Device Model / Introspection testing
  - Dom0 test agent connects to the IOREQ/VM_EVENT ring
  - Microkernel makes a set of specific actions
  - Test agent checks for correct requests in the ring, and responds
  - Microkernel checks for correct results of the responses

Further testing ideas

- Currently, tests are "boot microkernel, wait for it to exit"
  - Easy, and very effective for a lot of tasks
- Device Model / Introspection testing
  - Dom0 test agent connects to the IOREQ/VM_EVENT ring
  - Microkernel makes a set of specific actions
  - Test agent checks for correct requests in the ring, and responds
  - Microkernel checks for correct results of the responses
- Configuration testing
  - In dom0, iterate over VM configuration options
  - Boot the microkernel for each configuration, and check

Further testing ideas

- Currently, tests are "boot microkernel, wait for it to exit"
  - Easy, and very effective for a lot of tasks
- Device Model / Introspection testing
  - Dom0 test agent connects to the IOREQ/VM_EVENT ring
  - Microkernel makes a set of specific actions
  - Test agent checks for correct requests in the ring, and responds
  - Microkernel checks for correct results of the responses
- Configuration testing
  - In dom0, iterate over VM configuration options
  - Boot the microkernel for each configuration, and check
- Performance testing
  - Microbenchmarking or multi-domain stress testing

Further testing ideas

- Currently, tests are "boot microkernel, wait for it to exit"
  - Easy, and very effective for a lot of tasks
- Device Model / Introspection testing
  - Dom0 test agent connects to the IOREQ/VM_EVENT ring
  - Microkernel makes a set of specific actions
  - Test agent checks for correct requests in the ring, and responds
  - Microkernel checks for correct results of the responses
- Configuration testing
  - In dom0, iterate over VM configuration options
  - Boot the microkernel for each configuration, and check
- Performance testing
  - Microbenchmarking or multi-domain stress testing
- Combine with coverage
  - See which paths are actually covered by a test

Any Questions?

- Source:
  - `git://xenbits.xen.org/xtf.git`
  - http://xenbits.xen.org/gitweb/?p=xtf.git
- Documentation:
  - http://xenbits.xen.org/docs/xtf/
- Examples:
  - Already 37 tests upstream and running automatically
  - http://xenbits.xen.org/docs/xtf/test-index.html
- Discussion:
  - xen-devel@lists.xenproject.org
  - irc://#xendevel@chat.freenode.net