Concurrency: A problem and opportunity in the exploitation of memory corruptions

Ralf-Philipp Weinmann, Comsecuris
<ralf@comsecuris.com>

2014-03-13
Vancouver, Canada

$ whoami

- Founder of Comsecuris, a research and consulting company in Germany
- Previously at University of Luxembourg
- Cryptography and vulnerability research background

Comsecuris

Overview

- Motivation
- Characterization & patterns of concurrency bugs
- Relevant academic studies
- Bug finding
- Exploitation strategies
- Constructing concurrency bugdoors

Motivation

- Modern machines, whether big or small, are multi-core
- Writing correct multi-threaded code is hard
- Interesting bugs all the way through the stack: application/server → OS kernel → hardware (CPUs)
- Many codebases of interest to explore (both server-side and client-side)
  - Web servers, Database servers
  - Browsers (mostly events, but also web workers)

Comsecuris

Objective of Talk

- Large body of academic work on concurrency bugs
- Mostly disconnected from security community
- Survey of practical concurrency issues and academic approaches to them
- Focus on issues triggering *memory corruptions*
- Exploration how and when these can be leveraged
- Concurrency providing *advantage* to attacker (use-after-frees)

Comsecuris

Root Causes of Concurrency Bugs

- *Atomicity violations:*
  Intended serializability between multiple memory accesses is violated
- *Order violations:*
  Intended order between groups of memory accesses is flipped
- Other cases exist (not covered here)

Atomicity Violation

Thread 1

```
[...]
if (gctx->f != NULL)
   free(gctx->f);
[...]
[...]
```

Thread 2

```
[...]
gctx->f = someptr;
[...]
```

Thread 2 is scheduled right after check of **`gctxt->`**f, executing **`free()`** on **`someptr`** instead of **`gctxt->f`**.

Comsecuris

Order Violation

Thread 1

```
[...]
T = new T();
[...]
[...]
```

Thread 2

```
[...]
y = T->x
[…]
```

`T.x` gets read into local variable **`y`** before object **`T`** has been created, accessing uninitialized memory

Data Races

- Two memory accesses *conflict* iff:
  - They are to the same location
  - At least one is a write operation
- *Data race*: concurrent conflicting memory accesses
- C++11 mandates that programmer's must take care to avoid data races

Important reminder:

- Data races not necessarily are concurrency bugs
- Code that is data-race free may contain concurrency bugs

Comsecuris

Bug Finding

- Spotting concurrency bugs by code inspection in real-world code bases: moderately hard, but possible
- Look for state that is accessed without locking
- Also useful to inspect patched bugs to see patterns specific to code base.
- Finding remotely exploitable concurrency bugs is a formidable challenge/pastime
  - Actual exploitation often advances state of the art

Statistics to Keep in Mind

From "*Learning from Mistakes — A Comprehensive Study on Real World Concurrency Bug Characteristics* (S. Lu, S. Park, E. Seo and Y. Zhou):

- 92% of examined bugs guaranteed to manifest if certain partial order on <= 4 memory accesses is enforced
- 96% of examined bugs guaranteed to manifest if certain partial order enforced on 2 threads
- 66% of examined non-deadlock bugs involve concurrent access to 1 variable

Comsecuris

More Relevant Data from Study

- Cited study investigated 105 real-world concurrency bugs from MySQL, Apache, Mozilla and OpenOffice
- 70% of those bugs were non-deadlock bugs
- Fixes for deadlock bugs often were observed to bear potential to introduce non-deadlock bugs

Manifestations of Concurrency Bugs

- According to Zhang, Sun, Lu (2010):

  Intermediate memory error:

  Buggy interleaving causes execution order of memory accesses that directly leads to memory error

  Intermediate semantic error:

  Buggy interleaving causes new and unexpected program states (e.g. length field is corrupted)

Types of Concurrency Memory Errors

- `NULL` pointer deferences (less useful)
- Dangling pointers
- Uninitialized variables
- Buffer overflows due to intermediate semantic errors

Comsecuris

Interesting Browser Bugs

- Surprisingly low number of concurrency-based memory corruptions found in Chromium and Firefox bug trackers.
- UAF in SpeechRecognizerImpl (IPC), May 2013
  Chromium issue #244415
- Firefox 956327 (open since January 2014):
  Audit use of weak pointers regarding thread-safety:

```
"This considers both XPCOM's
ns(Supports)WeakReference and mfbt/(Supports)WeakPtr.

There might be a problem when multiple threads
access/reference/dereference an object implementing
nsSupportsWeakReference or SupportsWeakPtr."
```

Comsecuris

ThreadSanitizer

- Data race detector for C/C++
- PIN-based for Windows, Valgrind based for Linux/OSX
- Used by Google's Chrome security team
- Dynamic annotations for explaining app-specific synchronization mechanisms to detector
- Supports different level of aggressiveness
  - Resulting in different percentage of false positives
- Memory overhead of 3-4x

Firefox Chaos-Mode

- Very recently (March 2014) landed in Mozilla trunk*
- Intentionally introduce non-determinism
  - Assigns random thread priorities
  - Yield directly before firing XPCOM event
  - Scale timer firing times by random amounts
  - Etc.
- Has found bugs already

* http://robert.ocallahan.org/2014/03/introducing-chaos-mode.html

Comsecuris

Cuzz: Microsofts Concurrency Fuzzer

- Intuition behind approach:
  - Exponentially many thread schedules possible
  - Concurrency bugs: unexpected interaction between *small* number of threads and operations
- Uses binary instrumentation on unmodified PE binaries
- Found two unknown bugs in Firefox and Internet explorer (however, unclear from paper whether these have security impact)

Bug Depth

Def. depth of a concurrency bug:

Minimum number of scheduling constraints necessary to exhibit bug

- Generally, expected small depth expected
- Ordering bugs: usually *d=1*
- Atomicity violations: usually *d=2*

Comsecuris

Probabilistic Concurrency Testing

Ground rules:

- Lower number = lower priority
- Only one thread scheduled to execute in each step
- Passing *priority change point i*: thread priority is changed to *i*
- Priority change points only inserted for synchronization operation
  - e.g. system calls, pthread calls, hardware sync instructions
  - Also: volatile accesses, thread-shared memory operations

Comsecuris

Probabilistic Concurrency Testing

PCT Algorithm (patented, US20110131550):

- At thread creation: Assign random priority values *d, d+1, …, d+n* to each created thread
- Pick *d-1* random priority change points. Each of these gets assigned a unique value between *1, …, d-1*
- Run highest priority runnable thread at each scheduling step
- When thread reaches *i*-th change point, change thread priority to *i*

Comsecuris

More on Cuzz/PCT

- Significant improvement over adding randomized amount of sleep after each synchronization point
- Practically applicable on large-scale program
- No priority change points inserted for *d=1*
- *Main result:* Guaranteed detection probability of bugs of depth *d* in each run with probability ≥ $1/nk^{d-1}$

Comsecuris

Puzz: Poor Man's Cuzz for Unix

- Poor man's cuzz: ELF binary patching instead of binary instrumentation
- First iteration: only hook libpthread and system calls
- Prototype for Linux done in less than 1000 LOC
- Found that real-world code of interest to me has too many synchronization points
- Currently adding API to select synchronization points of interest.
- Thinking about switching PIN to handle “shared-memory” synchronization as well (non-thread local memory operations).
- Binary instrumentation (or compiler plugin) needed for browser fuzzing

Comsecuris

Ingredients for Exploitation

- Primitives for slowing down or blocking threads
  - For local escalation, see next slide
- Other cases: Since memory corruption will most likely be on heap, some Heap Feng-Shui needs to be applied.
  - If possible: pre-groom threads to a point right before critical point, then craft heap with another thread, then try to trigger vulnerability. Rinse and repeat.
  - If above strategy not possible (usually for remote vulnerabilities): simulate heap allocator, use simulated annealing to obtain good layout. Run allocation strategy against target. Needs to make assumption about thread schedules!
- Open problem: Better strategies for the remote case.

Concurrency Opportunities

- Server-side code also contains use-after frees
- Common situation:
  - In sequential execution no allocation is possible between deallocation and use of object
  - Use-after-free may be masked when freed object is not cleared by deallocation routine.
- Additional threads provide leverage to place object
- Still tricky to exploit, but in some cases makes latent bugs exploitable.

Public Examples of Slowdowns/Blocks

- Mateusz Jurczyk and Gynvael Coldwind: Bochspwn [exploits double-fetch vulnerabilities] (Syscan 2013)
  - Slowing down memory fetches using TLB flushing, cache-line/page boundaries and non-cacheability
- Exploitation of iOS 6 **`posix_spawn`** kernel bug by Stefan Esser (HITB 2013)
  - Used file locking to win race reliably
- Robert M. Watson: Exploiting Concurrency Vulnerabilities in System Call Wrappers (WOOT 2007):
  - copying arguments to syscalls that span pages may trigger page faults
  - connect() sleeps while waiting for SYNIACK

Going Deeper

- Concurrency problems exist in silicon too, but usually very tricky to trigger
  - No primitives for slowdowns
- Example: Errata #761319 in ARM Cortex A9
- See Encore to main presentation

Concurrency Bugdoors

- Introduce bugs that are hard to trigger without knowing schedule (use high *d* to make it hard to find)
- For bugdoors, usually logic bug better than memory corruption
- But, concurrency bugdoors can easily provide use-after-frees: if well-placed, both infoleak and arbitrary-write primitive
- Memory corruption may be preferred over logic bugdoor because it can provide RCE capabilities.

References

- J. Erickson, M. Musuvathi, S. Burckhardt, and K. Olynyk: *Effective Data-Race Detection for the Kernel*, in Operating, USENIX OSDI 2010
- M. Musuvathi, S. Burckhardt, P. Kothari, and S. Nagarakatte: *A Randomized Scheduler with Probabilistic Guarantees of Finding Bugs*, ACM ASPLOS 2010.
- S. Lu, S. Park, E. Seo, Y. Zhou: *Learning from mistakes: a comprehensive study on real world concurrency bug characteristics,* ACM ASPLOS 2008.
- Wei Zhang, Chong Sun, Shan Lu: *ConMem: Detecting severe concurrency bugs through an effect-oriented approach,* ASPLOS 2010.

Comsecuris

The Encore (Work in Progress): Hardware-assisted Memory Corruptions

Ralf-Philipp Weinmann, Comsecuris
<ralf@comsecuris.com>

2014-03-13
Vancouver, Canada

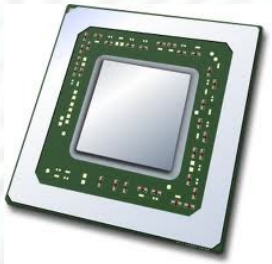
Overview

- Classes of software flaws triggering memory corruptions
- Part I: CPU bugs / silicon errata
  - Broken promises & countermeasures
- Part II: spurious DMA transfers
  - Countermeasures
- Conclusions

Comsecuris

What Hardware Can Corrupt Memory?

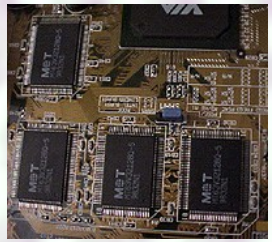
- CPU (malfunction)



- Devices with direct memory access

- HDD/SSD if paging is used

- Off-chip cache memory (some L3 cache on x86) and cache controller



- RAM itself

Comsecuris

Hardware/Firmware Memory Corruptions

- Memory corruptions caused by spurious/buggy DMA transfers
- NIC, GPU, HDD/SSD controller, sound card
- Bus mastering (PCIe, ARM AMBA)
- Microsoft has a support document on firmware-induced corruptions during sleep transitions:

  "During Windows 7 development, multiple Windows-based platforms have encountered memory corruption in the lowest 1 MB of physical memory after resuming from sleep."
- Windows 7 does not store system code and data in lowest 1MB of physmem
- Silicon errata (CPU bugs)
  - but checks whether corruption has occurred

Comsecuris

Related academic work

Sudhakar Govindavajhala, Andrew W. Appel:
Using Memory Errors to Attack a Virtual Machine,
IEEE S & P 2003.

Arbitrary code execution using heat
(70% success rate)

RAM itself

Eli Biham, Yaniv Carmeli, Adi Shamir:
Bug Attacks, CRYPTO 2008.

Using (hypothetical) bugs in hardware implementations to break public-key crypto algorithms

CPU bugs

Comsecuris

Previous discussion: Intel CPU bugs

- Theo de Raadt (OpenBSD project founder) in June 2007 [http://marc.info/?l=openbsd-misc&m=118296441702631]:

  "Various developers are busy implimenting (sic!) workarounds for serious bugs in Intel's Core 2 cpu.These processors are buggy as hell, and some of these bugs don't just cause development/debugging problems, but will *ASSUREDLY* be exploitable from userland code.[...]For instance, AI90 is exploitable on some operating systems (but not OpenBSD running default binaries)."

- Kris Kaspersky, Alice Chang: Remote Code Execution through Intel CPU Bugs,HackInTheBox Malaysia 2010
  - Bold claims (about AI39, AI43, AI79, AX52)
  - Somewhat skeptical, haven't actually seen proof (working code)
- Cache coherence correctly identified as big issue

Comsecuris

Our focus: ARM processors

- Intel “specification updates” provide little detail
- Apparently intentional to make issues non-reproducible from description
- No microcode to deal with
- Hypothesis: bugs are easier to understand/reproduce
- Focus on recent ARMv7 cores:
  - Cortex A8 (Apple A4 in iPhone 4, OMAP 3130 in N900/Motorola Milestone)
  - Cortex A9 (most multi-core smartphones & tablets)
  - Cortex A15 (Samsung Series 3 Chromebook, Google Nexus 10)

Comsecuris

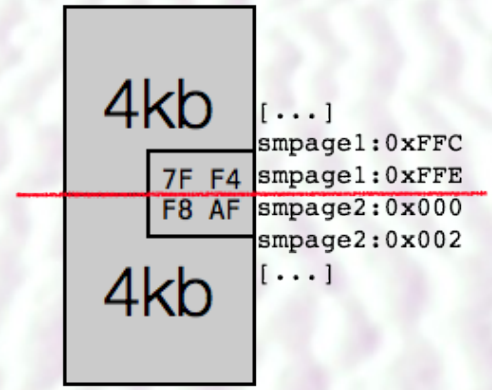
ARM errata

- Until late 2012 not publicly available from ARM
  - Detailed descriptions leaked in SoC vendor errata (FreeScale, Texas Instruments)
- Classification into 3 categories (A, B subdivided into common and rare):
  - category A: critical error (no workaround)
  - category B: critical error (with workaround)
  - category C: minor error
- Usually workaround/fixes set bit(s) in undocumented diag. control register to disable CPU features
- a bit to disable every single optimization?

ARM Erratum #657417

- Present in early revisions of Cortex-A8 core
- Concerns Thumb-2 direct branches with instruction on 4kb region boundary:

"If, while executing code in Thumb or ThumbEE state, a 32-bit Thumb-2 branch instruction is executed that spans two 4KB regions (i.e. the PC of the branch is xxxxxFFE), and the target address of the branch falls within the first region, it is possible for the processor to incorrectly determine that the branch instruction and the branch target are both within the same 4KB region and therefore a TLB lookup is not required on the target address. This failure to do a TLB lookup and instead use the physical address of the branch instruction for bits [31:12] causes the processor to branch to the wrong instruction address. It is also possible in some cases for the processor to enter a deadlock state when this incorrect fetch address is generated."



- Jump to incorrect target address can be caused
- Requires virtual to physical addresses to not be 1:1 to be exploitable
- Reproducible on Nokia N900 (OMAP 3130)

```
BNE.W smpage1:0xFF2
```

Comsecuris

WebKit/JavaScriptCore

- Surprise: original implementer aware of erratum:

```
static bool canBeJumpT3(const uint16_t* instruction, const void* target, bool& mayTriggerErrata) {
  ASSERT(!(reinterpret_cast<intptr_t>(instruction) & 1))
  ASSERT(!(reinterpret_cast<intptr_t>(target) & 1));
  intptr_t relative = reinterpret_cast<intptr_t>(target -
(reinterpret_cast<intptr_t>(instruction));
  // From Cortex-A8 errata:
  // If the 32-bit Thumb-2 branch instruction spans two 4KiB regions and
  // the target of the branch falls within the first region it is
  // possible for the processor to incorrectly determine the branch
  // instruction, and it is also possible in some cases for the processor
  // to enter a deadlock state.
  // The instruction is spanning two pages if it ends at an address ending 0x002
  bool spansTwo4K = ((reinterpret_cast<intptr_t>(instruction) & 0xfff) == 0x002);
  bool mayTriggerErrata = spansTwo4K;
  // The target is in the first page if the jump branch back by [3..0x1002] bytes
  bool targetInFirstPage = (relative >= -0x1002) && (relative < -2);
  bool wouldTriggerA8Errata = spansTwo4K && targetInFirstPage;
  return ((relative << 11) >> 11) == relative && !wouldTriggerA8Errata;
}
```

- Bigger surprise: Above code removed from WebKit in November 2012
- Some Android devices potentially vulnerable (wasn't able to reproduce erratum on Apple A4)
- Attacker needs to predict virt. to phys. mapping
  - seems to be possible under some assumptions

Comsecuris

Implications of ARM #657417

- Assume the attacker can generate code... Javascript!
  - May allow an attacker to corrupt memory (interpreter hijack) or change execution flow to his liking
- Checked Javascript engines
  - V8 began implementing Thumb-2 backend in 2009/2010, killed a short while later
  - Firefox: Thumb-2 “under consideration” according to https://wiki.mozilla.org/Javascript:SpiderMonkey:2013Projects
  - JavaScriptCore emits Thumb-2, yay!
  - Haven’t checked MS JIT compiler yet
- Alternative: pNaCl (LLVM bitcode)
  - Does generate Thumb-2 code currently

Comsecuris

ARM Erratum #743622

- Faulty logic in the Store Buffer may lead to data corruption:

```
STR A
STR A'
STR A''
STR B
STR A''' (or STR B')
```

- A, A', A" in the same cache line, B and B' in different cache line
- Fifth STR may write into faulty cache line (requires A or B to be invalidated by coherent request from another CPU)
- Thus far unable to reproduce this issue

Comsecuris

ARM Erratum #761319

- Read-after-read hazard on Cortex A9

```
STR <valueA>, [loc]
…
STR <valueB>, [loc]
```

CPU core #1

```
LDR Rx, [loc]
…
LDR Ry, [loc]
```

CPU core #2

- Problem: core #2 might see **Rx == valueB**, **Ry == valueA**
- reproducible on ST-Ericsson NovaThor U8500 CPU (1.7% success rate)
- Only seems to affect code using lock-free programming

Comsecuris

Example Cortex A15 Erratum

- Errata have become more sparse in some cases
- ARM erratum #773022:

```
Incorrect instructions may be executed from loop buffer
Category B
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r0p4
Description:

In certain rare sequences of code, the loop buffer may
deliver incorrect instructions. NOTE: This erratum
matches bug #5121 in the ARM internal Jira database.
```

- ARM may have realized security implications?

Comsecuris

Debugging ARM CPU errata

- JTAG on ARM allows to see deeply into CPU (thank you, Travis Goodspeed!)
- Mostly undocumented
- Allows access to store buffer, branch target buffer etc.
- Incentive: may allow derivatives of known errata
- Main problem foreseen: JTAG speed limited
- Research: need sampling strategy

Comsecuris

Broken Promises & Countermeasures

- Formally verified micro-kernels exist (seL4)
- Promises provided HOL-verified Haskell specification can be broken at hardware level
- Defense against unknown errata: hard
- Simplicity may allow formal verification of full current ARM cores (has last been done for ARM3)
- First steps by Fox and Myreen: *A Trustworthy Monadic Formalization of the ARMv7 Instruction Set Architecture*, Interactive Theorem Proving 2010
- ARMv7 in HOL on github: https://github.com/mn200/HOL
- Will most likely require some optimizations removed

Comsecuris

UEFI: 802.11 DMA memory corruption

- Problem found by Ubuntu developer Matthew Garrett*
- Memory allocated for UEFI boot services: Per UEF spec available after boot environment exits, i.e. to operating system
- UEFI firmware on Macs sets up DMA transfers RAM (into said memory region)
- Developer forgot to disable DMA
- Result: memory corruption at fixed physical addresses until OS driver reprograms WiFi chipset
- Exploitability in close-proximity case?

*http://mjg59.dreamwidth.org/11235.html

Comsecuris

ChromiumOS: Bluetooth DMA

Patch on February 25th, 2013:

```
CHROMIUM: bluetooth/usb: hack to cover up dma after free
corruption.This is a horrible, gross hack to cover up
the fact that we sometimes get an apparent DMA from the
interrupt endpoint on the bluetooth module after
shutting it down.  This ends up manifesting as random
kernel crashes, often from other users of kmalloc-16.
Since we don't have root cause (yet?) and can't do much
about it, put in this band-aid which will leak around 64
bytes of memory every time we suspend/resume a usb
bluetooth device.  We make no attempt to free this
memory because we don't know when the DMA could come in
and it's a relatively small amount so probably not worth
trying to free it later.”
```

Comsecuris

Summary DMA Memory Corruptions

- Various errata sheets for ARM SoCs list cases that sound like DMA memory corruption
- Vendors seem to downplay issue
- Security conscience for these issues underdeveloped
- Vendors see them as nuisance to developer
  - as stability issues by developers
- Control over corrupting data very often greatly limited
- Counterpoint: off-by-one buffer overflows were once deemed “unexploitable” as well
- Better use architectural defenses

Comsecuris

Countermeasures against DMA memory corruption

- IOMMU
  - Supported by Samsung Exynos4/5, NVidia Tegra 2/3
- Intel VT-d (virtualization for directed I/O)
  - DMA address translation/interrupt remapping
- Problems
  - primary use scenario of VT-d is isolation in virtualized environments
  - ability for OS to protect from hostile/malicious devices: side-effect
  - **need to ensure that DMA writable range does not contain pointers**

Comsecuris

Conclusions for Encore

- Using hardware/firmware to induce memory corruptions in main memory is possible
- Studying simpler designs pays off
- Creating exploits for CPU errata can be tricky
- JTAG may allow deeper understanding of errata
- Don’t trust your hardware!

Comsecuris