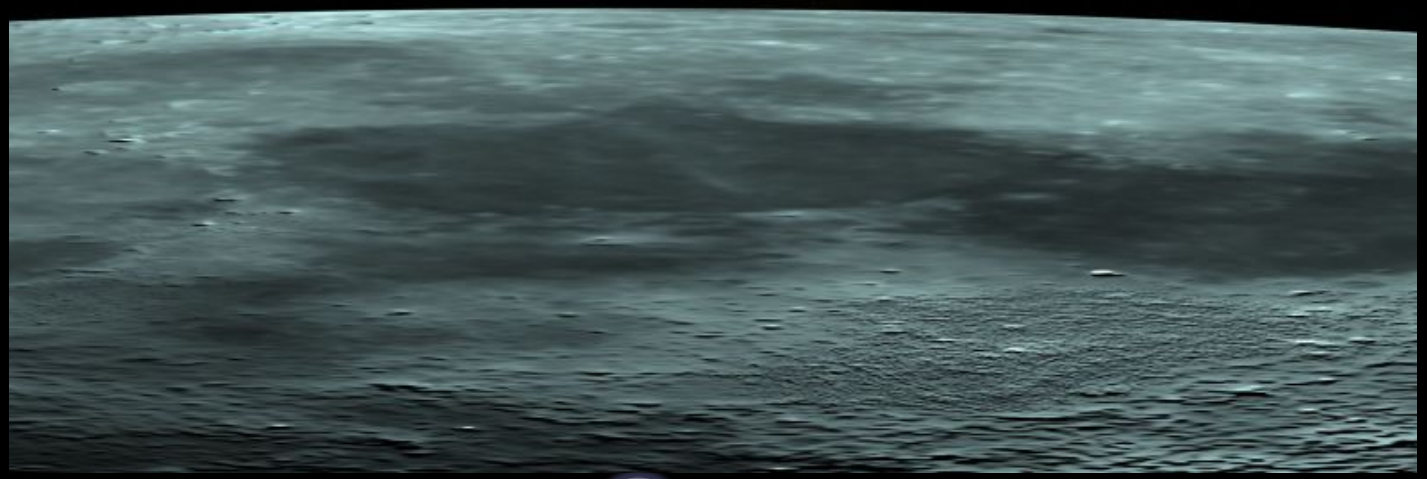
A stitch in time saves nine: a case of multiple OS vulnerability

Rafal Wojtczuk
rafal@bromium.com

Agenda

- CERT VU#649219 overview
- Crash course on ring transitions on x86_64
- Exploit techniques
- Related musings

PART 1. CERT VU#649219 overview



CERT VU#649219

- SYSRET 64-bit operating system privilege escalation vulnerability on Intel CPU hardware
  - Escalation from untrusted user to kernel
- Root cause: On Intel CPUs, “sysret” instruction executed with non-canonical return address throws exception in ring0
- Patches released on 12 June 2012

Known affected systems (in April 2012)

- Only 64bit OS versions running on Intel CPU are vulnerable
- Xen with PV guests
- Windows 7 and Windows 2008 R2
- FreeBSD
- NetBSD

Coordinating patches release

- Xen security team
- Other affected software vendors
- Intel
- US CERT
- Bromium
- Thank you all

Known non-affected systems

- Apple OSX
- OpenBSD >=5.0
  - Most likely, accidentally (?) fixed on 4 Jul 2011 during code cleanup
- Linux kernel >= 2.6.15.5
  - Consciously fixed the root cause in 2006
- So, why so many systems were vulnerable after 2006?

More on Linux case

- CVE-2006-0774: Linux kernel before 2.6.16.5 does not properly handle uncanonical return addresses on Intel EM64T CPUs, which reports an exception in the SYSRET instead of the next instruction, which causes the kernel exception handler to run on the user stack with the wrong GS.
- Impact not specified explicitly; Linux-specific

More on Linux case

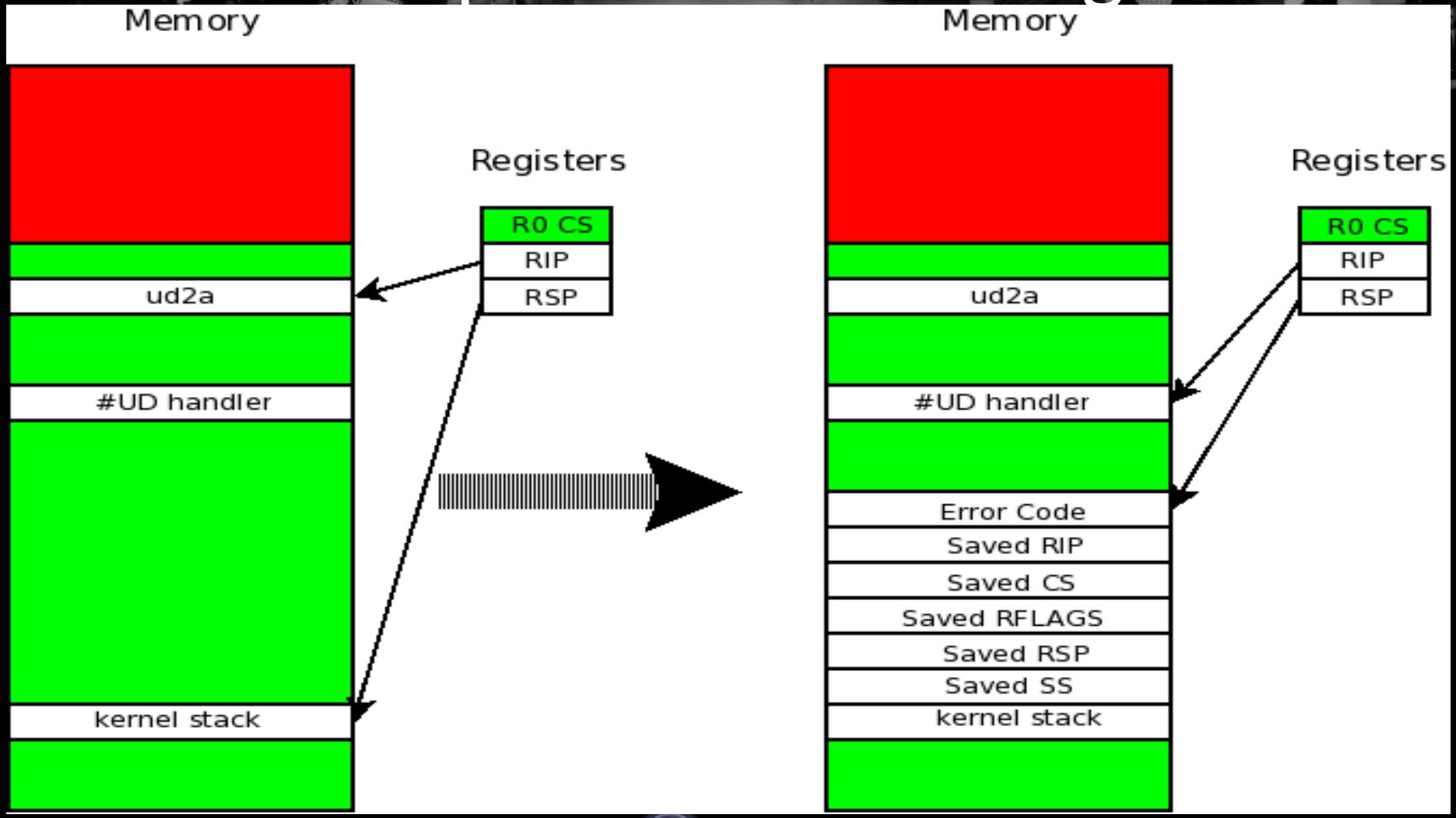
- BID 17541: Linux Kernel Intel EM64T SYSRET Local Denial of Service Vulnerability
- CVE-2006-0774 and BID 17541 suggest this is Linux-specific problem, DoS only
  - We will see it is not
- Apparently, other vendors did not notice the problem, and were not warned
- Hopefully, this talk is an explicit warning

Are escalations to kernel important?

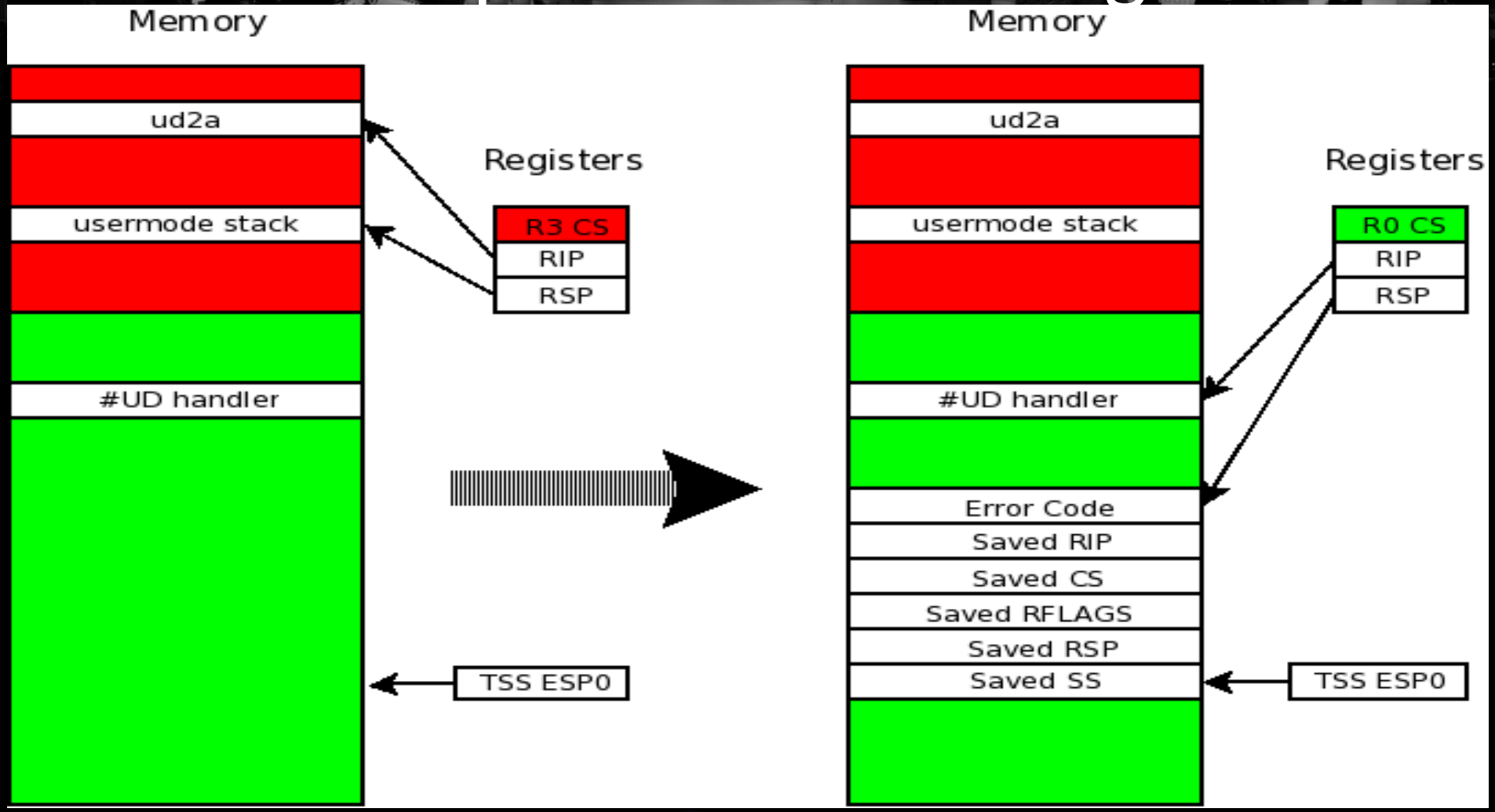
- On server systems, with untrusted users, obviously yes
- On desktop systems, they allow to escape from many sandboxing solutions
  - Really, no need to chain 10 different bugs
- Multiple OS issues are very rare, so this case is interesting

PART 2: Crash course on ring transitions on x86_64

Exception while in ring0



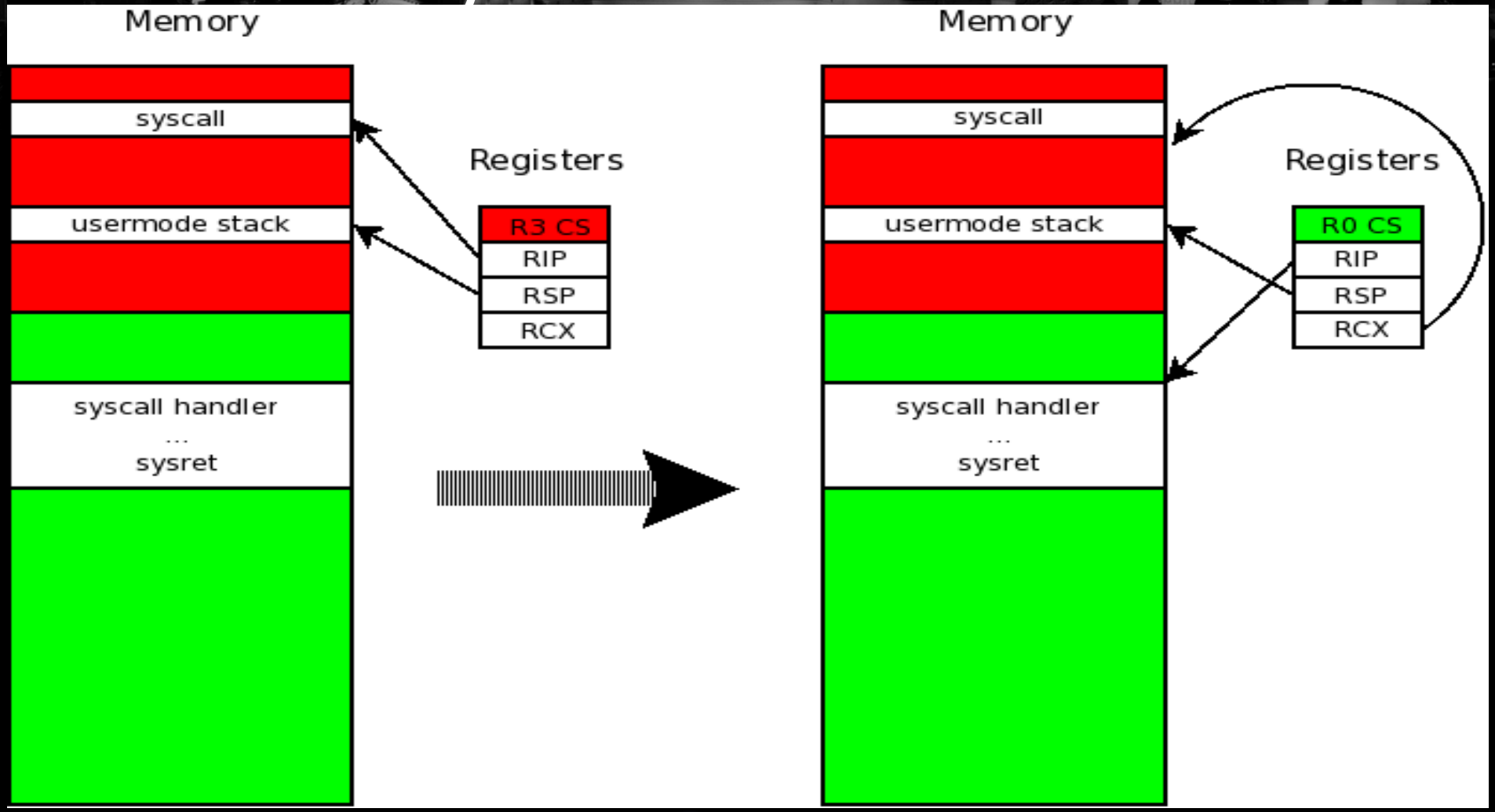
Exception while in ring3



More on stack switch mechanism

- Always used when changing ring
- Usually, not used when not changing ring
- Interrupt Stack Table feature allows to force stack switch even when exception happens in ring0
  - Normally used only for catastrophic exceptions like #MC, #DF and NMI

“syscall” instruction



“syscall” handler lifecycle

- Save usermode RSP
- Set RSP to kernel stack
- Do the actual job
- Restore usermode RSP
- Sysret

Exception in syscall handler…

- … when RSP is still usermode-provided, is dangerous
- RSP assignments should not fault
- Can “sysret” throw an exception?

Sysret manual entry

AMD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
M	M	M	M		0	M	M	M	M	M	M	M	M	M	M	M
21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The SYSCALL and SYSRET instructions are not supported, as indicated by EDX bit 11 returned by CPUID function 8000_0001h.
	X	X	X	The system call extension bit (SCE) of the extended feature enable register (EFER) is set to 0. (The EFER register is MSR C000_0080h.)
General protection, #GP	X	X		This instruction is only recognized in protected mode.
			X	CPL was not 0.

Sysret manual entry, Intel

64-Bit Mode Exceptions

#UD	If IA32_EFER.SCE bit = 0.
	If the LOCK prefix is used.
#GP(0)	If CPL ≠ 0.
	If ECX contains a non-canonical address.

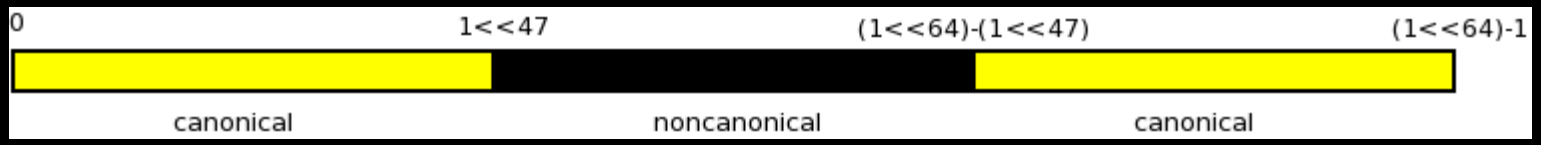
SYSRET—Return From Fast System Call

Impact?

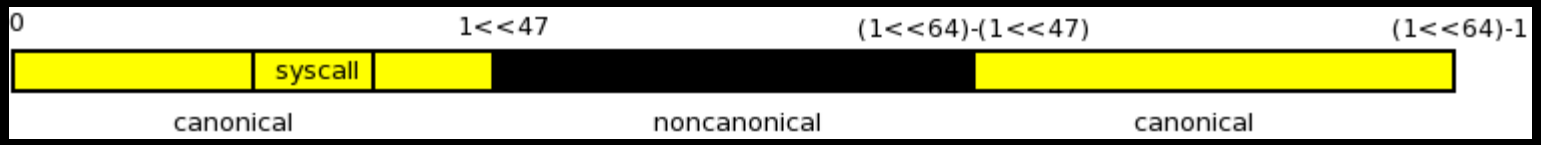
- Ring3 to ring0 escalation

Part 3: Exploit techniques

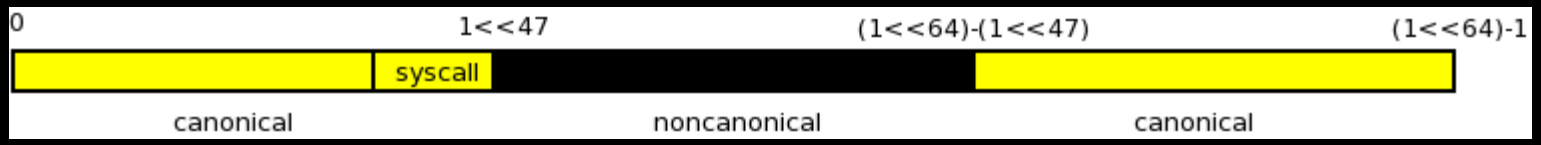
What is a non-canonical address?



Why is #GP normally never seen?



How to force non-canonical address?



Which OS allows required mapping?

- Linux – no
- Windows – no
- OpenBSD, NetBSD – no
- FreeBSD – yes
- Xen with PV guests– yes

DoS is straightforward…

- … place “syscall” at (1<<47)-2, set RSP to something unmapped, e.g. 0

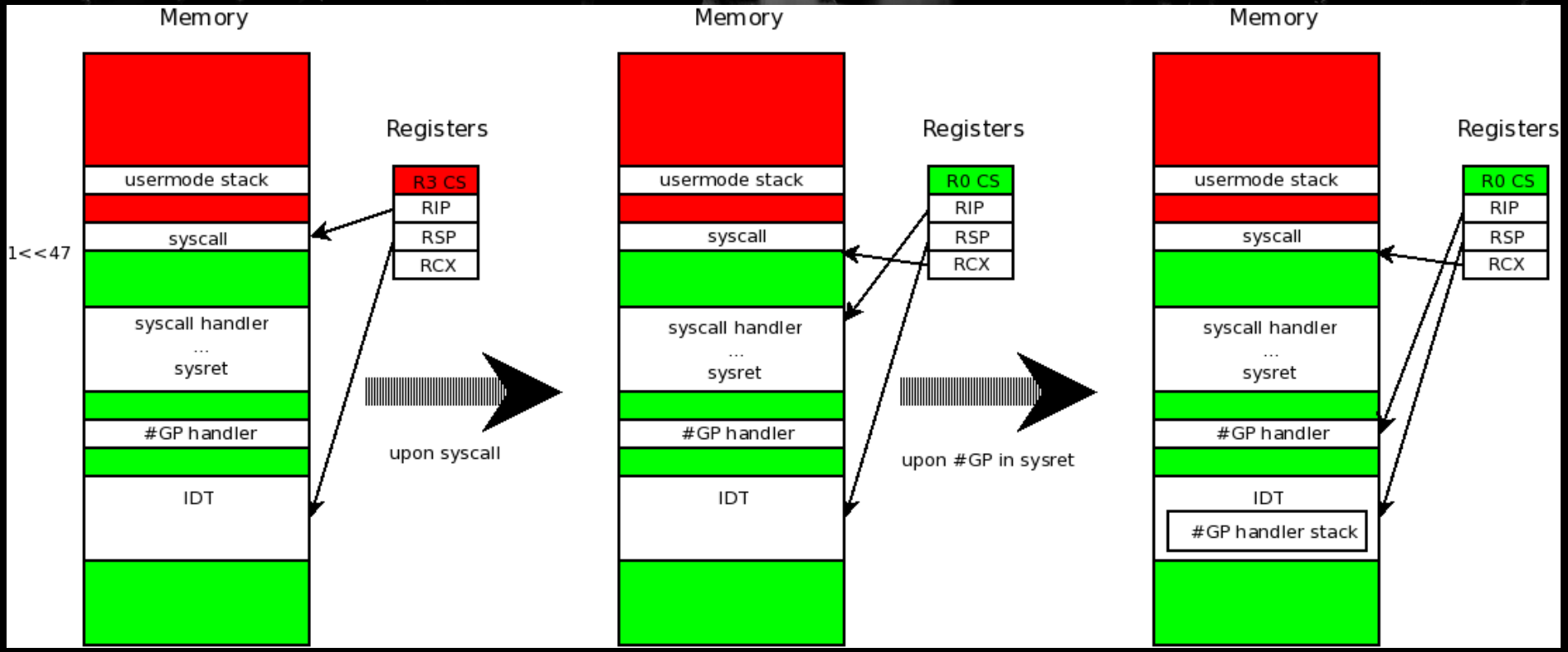
Code execution in ring0?

- One possible trick: before executing “syscall”, set RSP to point to some important kernel data structure
- When #GP is raised, processor will overwrite this structure with the exception record
- Any subsequent stack pushes done by #GP handler will scribble over it, too

We need to hurry…

- Usually, the job of #GP handler invoked from unexpected location in ring0 is to panic/ bugcheck the machine
- Also, #GP handler may crash because of running in an unexpected environment
  - Particularly, usermode gs_base

FreeBSD exploit scenario



FreeBSD exploit demo

- Exploit is reliable, and does not require knowledge of any kernel absolute addresses
  - IDT base leaks via “sidt”
- IDT overwrite is a very generic method

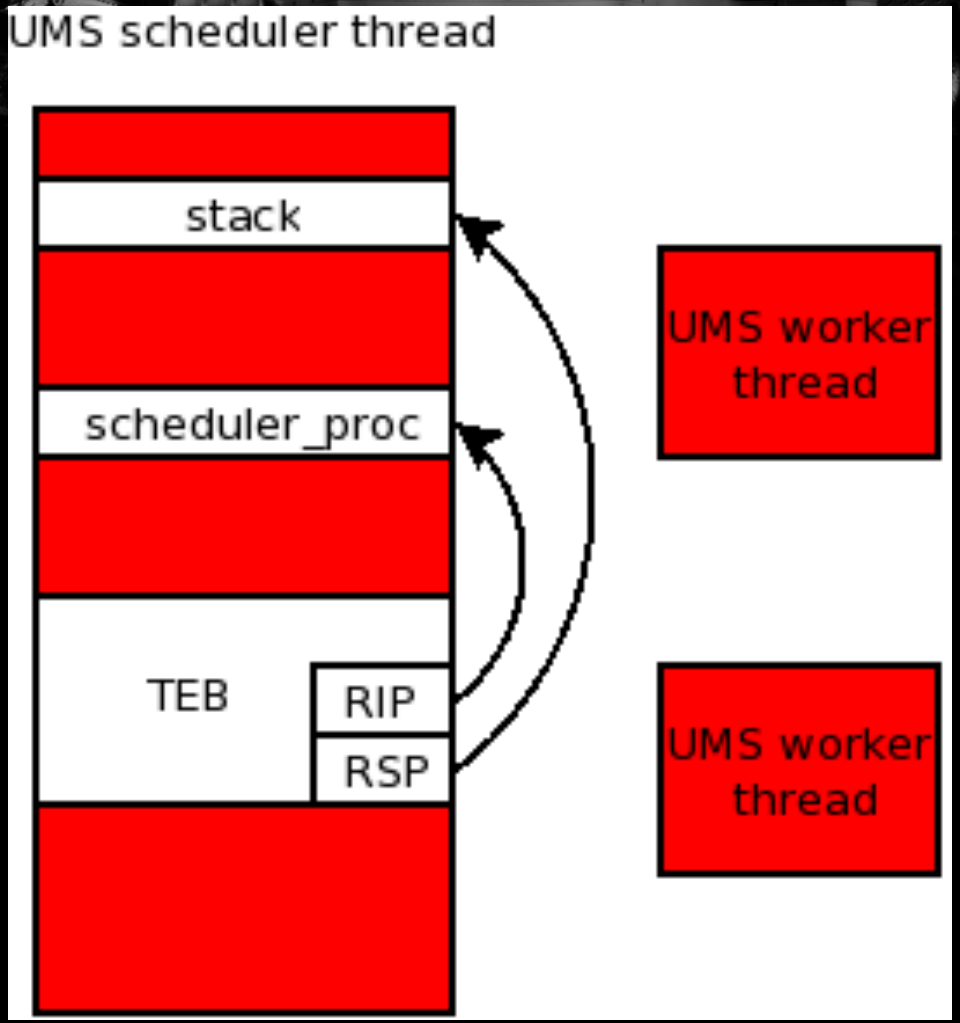
Windows 7 case

- Usermode not allowed to map/access memory at (1<<47)-2
- Any other way to force “sysret” with non-canonical address?

“sysret” not always returns after “syscall”

- Jan Beulich of Suse has spotted it first, in Xen context
- Sys_sigreturn, NtContinue
  - Not working – they return via “iret”
- Let’s search ntoskrnl.exe for “sysret” occurences

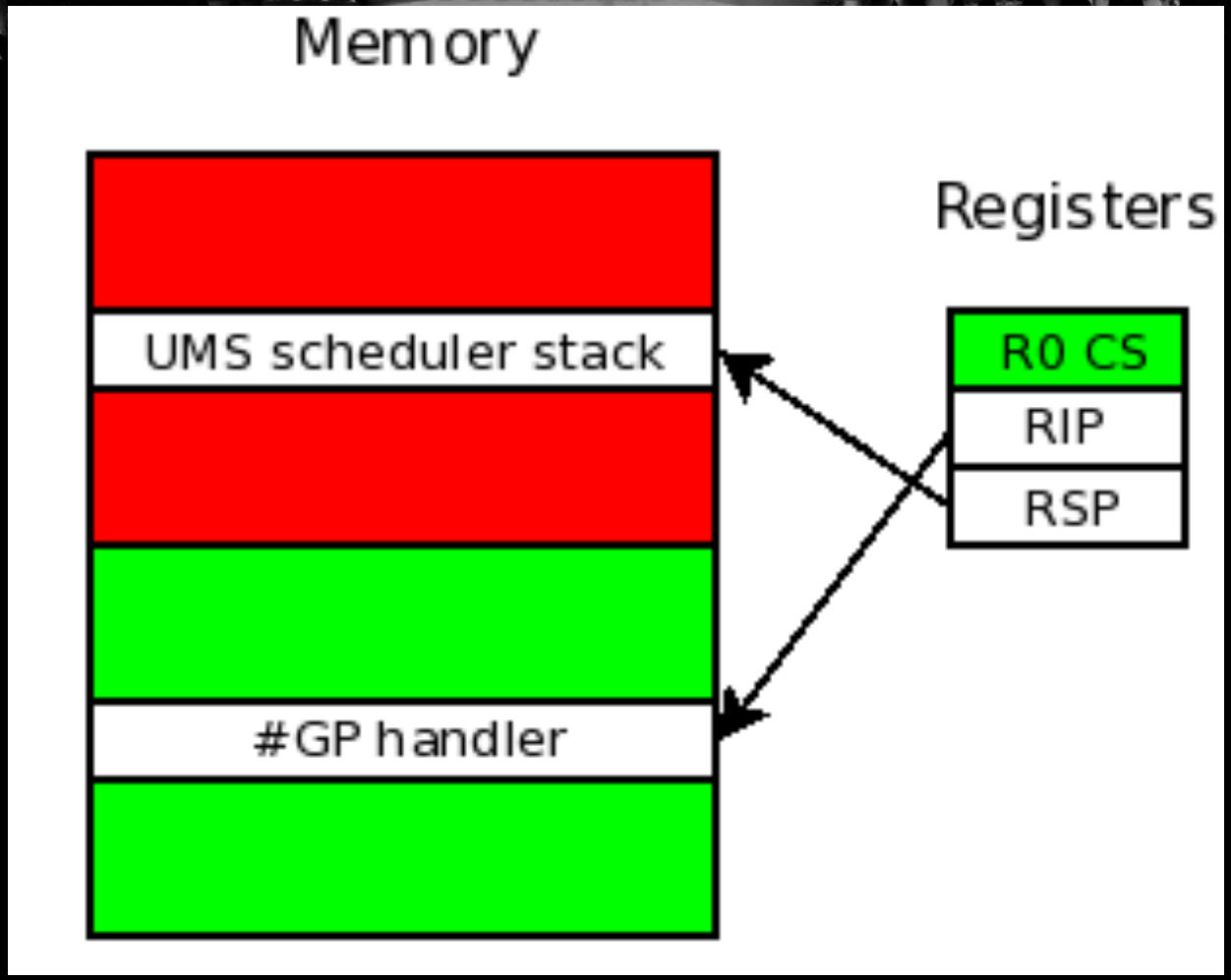
Windows User Mode Scheduling



Windows 7 exploit

- Just as before, we can trigger #GP handler running with arbitrary RSP
- Unfortunately, in this case, the trick of pointing RSP to IDT is problematic
  - Windows 7 #GP handler uses much more stack space, and before #PF is triggered, kernel memory before IDT is corrupted
- What if we point RSP to usermode area, and preload this memory with _really_ evil stuff?

#GP with usermode RSP



#GP with usermode RSP, cnt

- Stack is treated as uninitialized memory; code never reads from any location before write
- We need to overwrite some stack location after it was initialized by #GP handler, but before it is used by it
- We need to create a race condition – run another thread concurrently with exploit thread; so it works only on SMP system

Windows 7 exploit

- #GP thread actions:
  - Call some_function
  - Some_function does its job
  - "ret" to some_function's return address
- "overwriter" thread actions
  - Continuously overwrite some_function's return address with the address of kernel shellcode

Is it reliable?

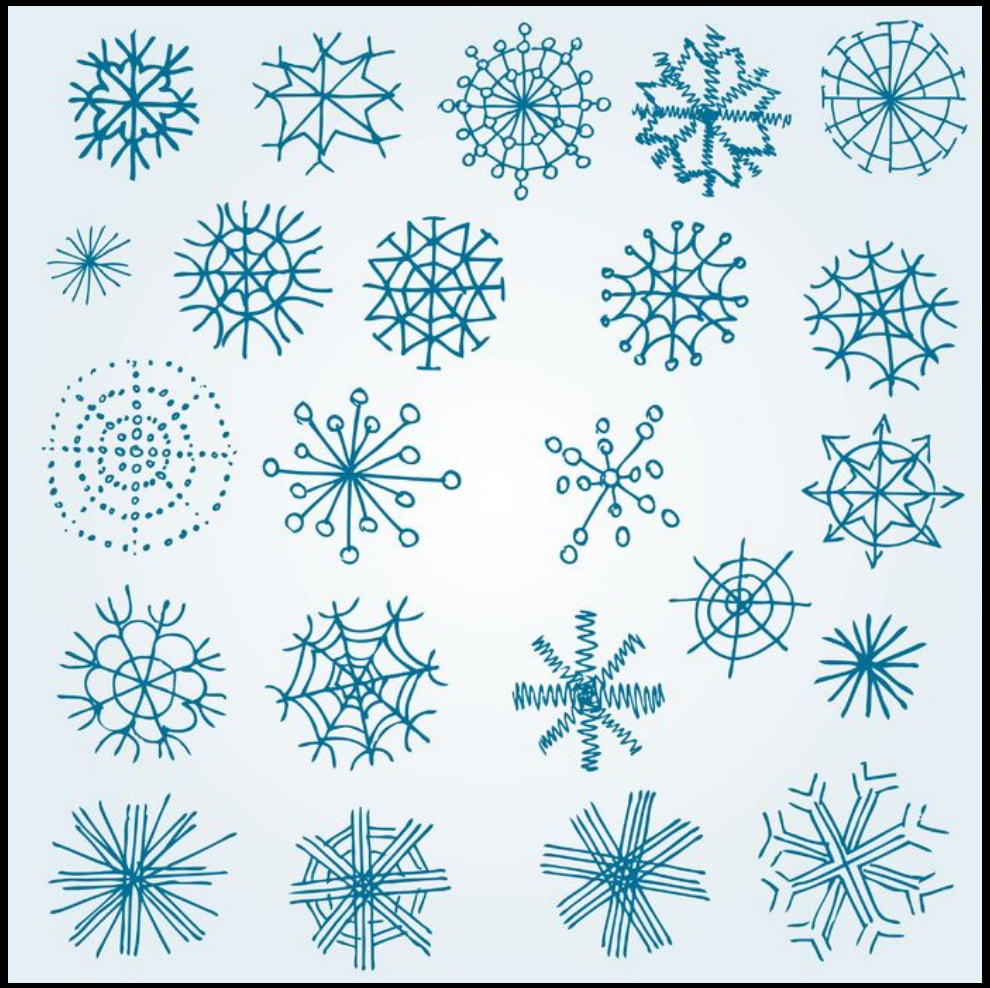
- "overwriter" must write in the short time window
  - It is only 1 assembly instruction in a loop
- "overwriter" must be scheduled to run when #GP handler runs
  - We can spawn many of them
- We have only one shot
- Exploit works 100% on bare metal; not in VM

Windows 7 demo

Related research

- Derek Soeder, “VMware Emulation Flaw x64 Guest Privilege Escalation”, Nov 2008
- Nate Eldredge, CVE-2008-3890
- In this presentation, to fit in the timeslot, I ignored the issue of “swapgs” desynchronization; see the above for details
  - BTW, Xen does not use “swapgs”

Part 4: Related musings



Witchhunt – whose fault is it?

- 3 answers, find the one that is a joke

Whose fault is it?

- Those pesky security researchers!
- “Security people are leeches”.
- "I can tell you I wish those people just would be quiet. It would be best for the world. That's not going to happen, so we have to work in the right fashion with these security researchers”

Whose fault is it?

- OS developers?
- “Sysret” semantics is explicitly documented in Intel SDM
- after CVE-2006-0774, all developers should have checked if it is applicable to their OSes

Whose fault is it?

- Intel?
- allowing “sysret” to raise an exception, with user-controlled stack, is a design error
- After CVE-2006-0774, Intel should have realized the problem, notify everyone, and update SDM with an explicit warning

Mitigation?

- Typically, kernel escalation exploits attempt to run with ring0 privilege some arbitrary code stored in usermode pages
- SMEP feature, present in Ivy Bridge - prevents this
- Properties very similar to NX/DEP…
- … with similar bypass techniques
- Still, a first step in the right direction

The state of mainline kernels security

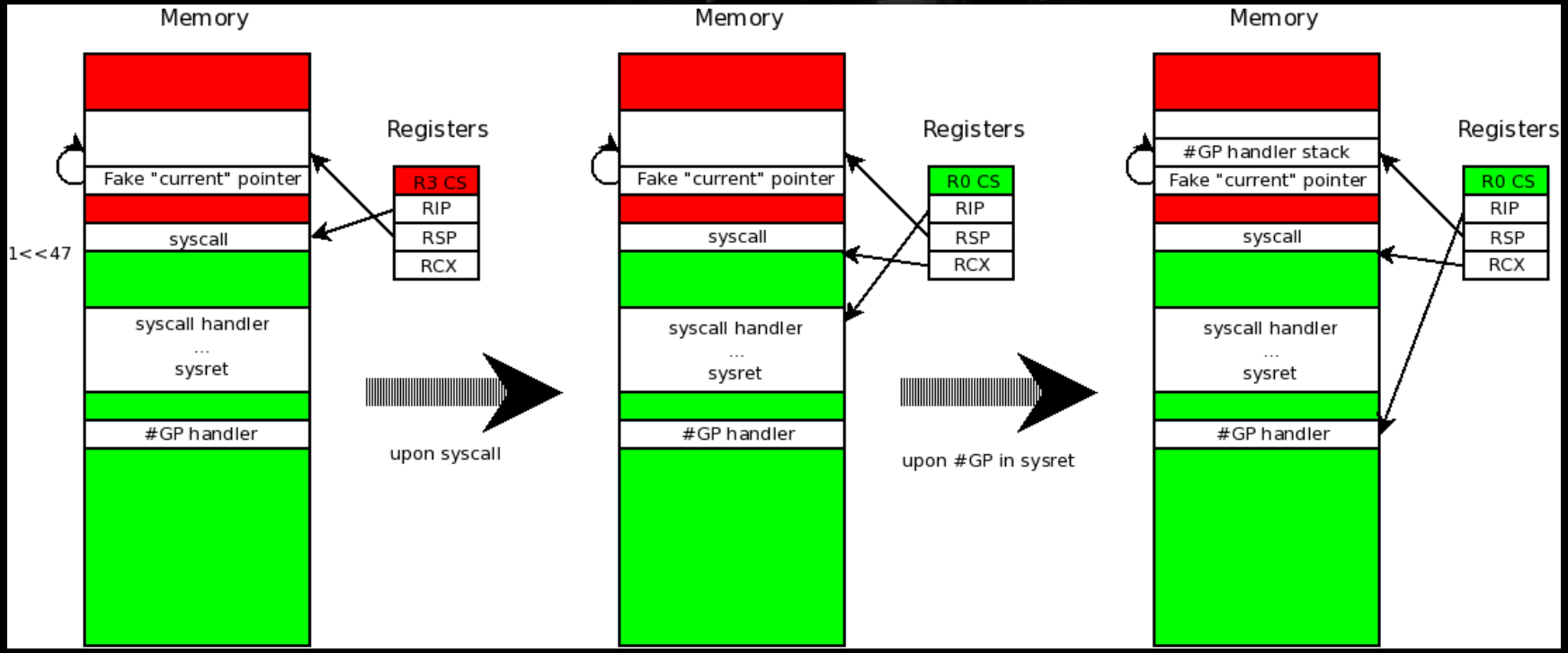
- Large code base, implies many vulnerabilities
- Often reliably exploitable
  - A lot of state shared with untrusted usermode, e.g. virtual memory mappings
- Encapsulating untrusted code with hardware-assisted virtualization, if done correctly, seems better
  - What happens in V*, stays in V*

Pls provide feedback to organizers…

Q&A

- Send questions to rafal@bromium.com
- Updated versions of the paper (if any) at http://www.bromium.com/misc/astitchintimesavesnine.pdf

Bonus track: Xen exploit



Bonus track: Xen exploit demo

Bibliography

- 1. CERT Vulnerability Note VU#649219, http://www.kb.cert.org/vuls/id/649219/
- 2. Philip Guenther, "Force the sigreturn syscall to return to userspace via iretq", http://www.openbsd.org/cgi-bin/cvsweb/src/sys/arch/amd64/amd64/locore.S.diff?r1=1.47;r2=1.48
- 3. Andi Kleen, "[PATCH] [18/30] x86_64: When user could have changed RIP always force IRET", http://www.x86-64.org/pipermail/discuss/2006-April/008271.html
- 4. CVE-2006-0744, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0744
- 5. BID 17541, http://www.securityfocus.com/bid/17541
- 6. Intel SDM, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- 7. Derek Soeder , “VMware Emulation Flaw x64 Guest Privilege Escalation”, http://www.securityfocus.com/archive/1/498150
- 8. Nate Eldredge, “amd64 swapgs local privilege escalation”, http://security.freebsd.org/advisories/FreeBSD-SA-08:07.amd64.asc
- 9. “User Mode Scheduling”, http://msdn.microsoft.com/en-us/library/windows/desktop/dd627187(v=vs.85).aspx