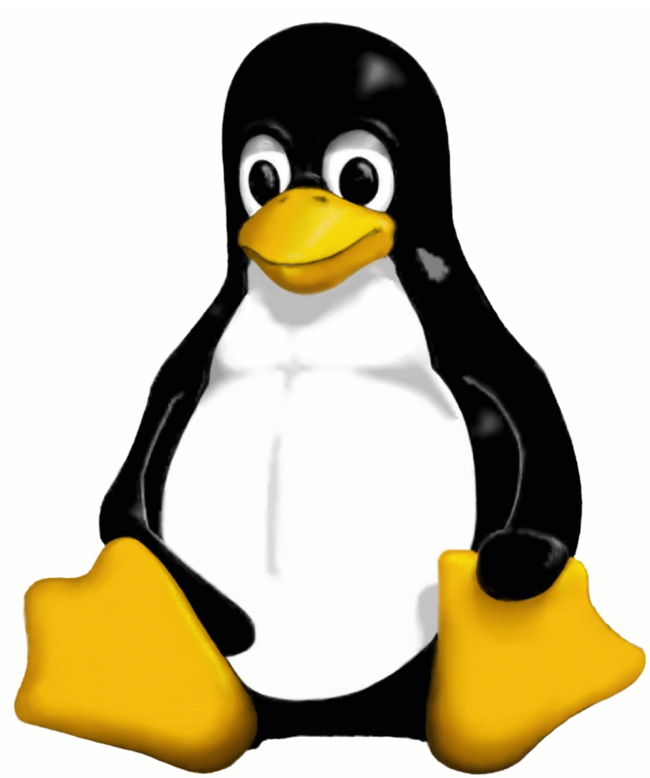
EXPLOITING THE LINUX KERNEL: MEASURES AND COUNTERMEASURES

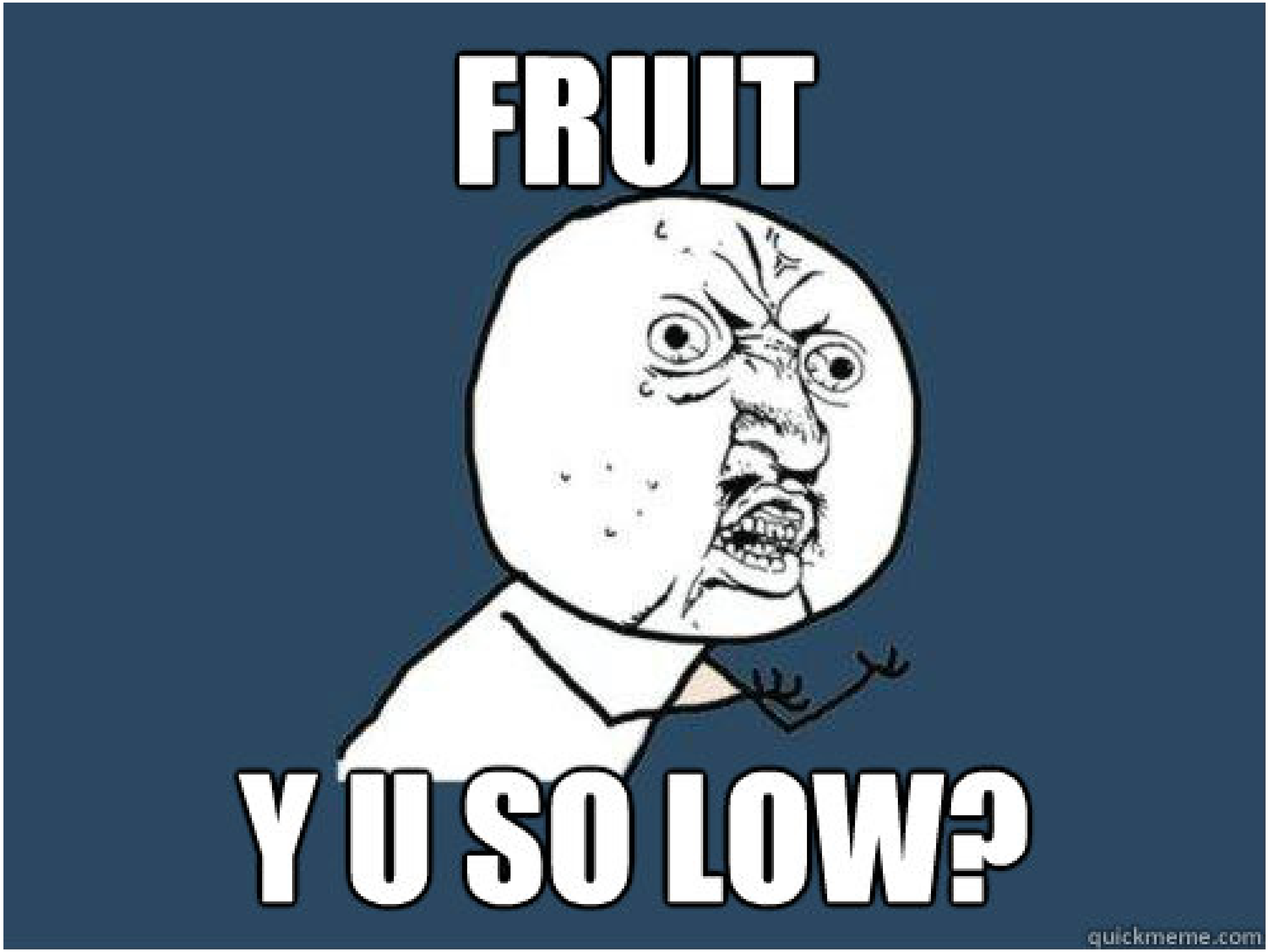


JON OBERHEIDE
DUO SECURITY

INTRODUCTION

- Who are you?
  - Jon Oberheide
  - I enjoy the Linux kernel
- What is this?
  - High-level look at Linux kernel mitigations
  - What has changed with respect to exploitation
  - Both on vanilla and on hardened kernels
  - *SPOILER: not as depressing as Chris/Tarjei's talk!*

WHY LINUX?



IN THE ENTERPRISE

MOBILE AND EMBEDDED

LUNATICS ON DESKTOPS

“The sound doesn't work on my Linux desktop *for security reasons*”

AGENDA

- **Vanilla kernel mitigations**
- Hardened kernel mitigations
- Future mitigations

WHY UPSTREAM SECURITY FAILS

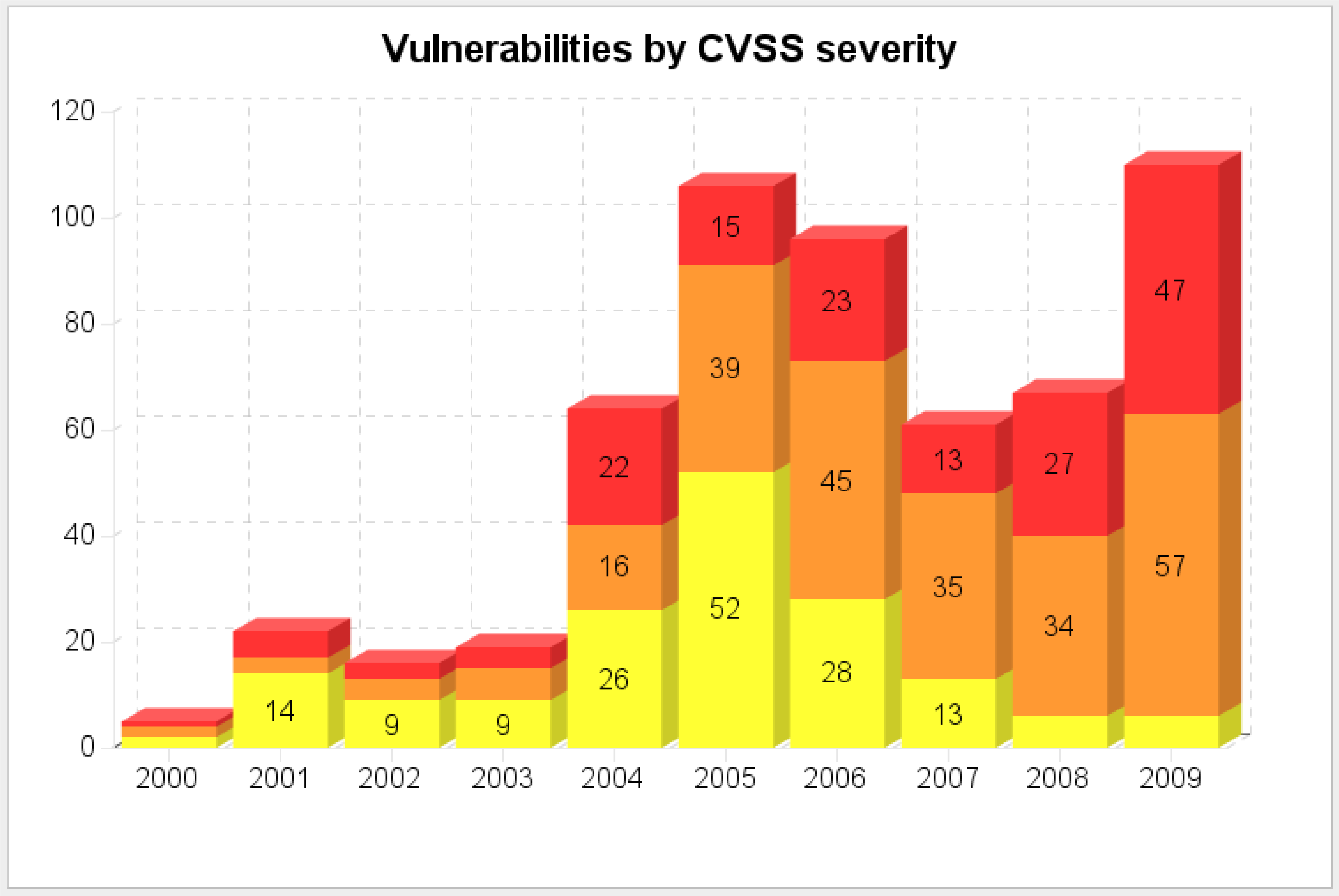
```
Btw, and you may not like this, since you are so focused on security, one
reason I refuse to bother with the whole security circus is that I think
it glorifies - and thus encourages - the wrong behavior.

It makes "heroes" out of security people, as if the people who don't just
fix normal bugs aren't as important.

In fact, all the boring normal bugs are _way_ more important, just because
there's a lot more of them. I don't think some spectacular security hole
should be glorified or cared about as being any more "special" than a
random spectacular crash due to bad locking.
```

A misguided view of security...although Linus has been getting better recently.

LINUX KERNEL SECURITY IN THE 2000s



DISTRO PROGRESS: RHEL

Features	Red Hat Enterprise Linux			
	3	4	5	6
	2003 Oct	2005 Feb	2007 Mar	2010 Nov
Firewall by default	Y	Y	Y	Y
Signed updates required by default	Y	Y	Y	Y
NX emulation using segment limits by default	Y(since 9/2004)	Y	Y	Y
Support for Position Independent Executables (PIE)	Y(since 9/2004)	Y	Y	Y
Address Randomization (ASLR) for Stack/mmap by default	Y (since 9/2004)	Y	Y	Y
ASLR for vDSO (if vDSO enabled)	no vDSO	Y	Y	Y
Support for NULL pointer dereference protection	Y(since 11/2009)	Y(since 9/2009)	Y(since 5/2008)	Y
NX for supported processors/kernels by default	Y(since 9/2004)	Y	Y	Y
Support for block module loading via cap-bound sysctl tunable or /proc/sys/kernel/cap-bound	Y	Y	Y	no cap-bound
Restricted access to kernel memory by default		Y	Y	Y
Support for SELinux		Y	Y	Y
SELinux enabled with targeted policy by default		Y	Y	Y
glibc heap/memory checks by default		Y	Y	Y
Support for FORTIFY_SOURCE, used on selected packages		Y	Y	Y
Support for ELF Data Hardening		Y	Y	Y
All packages compiled using FORTIFY_SOURCE			Y	Y
All packages compiled with stack smashing protection			Y	Y
SELinux Executable Memory Protection			Y	Y
glibc pointer encryption by default			Y	Y
Enabled NULL pointer dereference protection by default			Y(since 5/2008)	Y
Enabled write-protection for kernel read-only data structures by default			Y	Y
FORTIFY_SOURCE extensions including C++ coverage				Y
Support for block module loading via modules_disabled sysctl tunable or /proc/sys/kernel/modules_disabled				Y
Support for SELinux to restrict the loading of kernel modules by unprivileged processes in confined domains				Y
Enabled kernel -fstack-protector buffer overflow detection by default				Y
Support for sVirt labelling to provide security over guest instances				Y
Support for SELinux to confine users' access on a system				Y
Support for SELinux to test untrusted content via a sandbox				Y
Support for SELinux X Access Control Extension (XACE)				Y

DISTRO PROGRESS: UBUNTU

feature	8.04 LTS (Hardy Heron)	10.04 LTS (Lucid Lynx)	11.04 (Natty Narwhal)	11.10 (Oneiric Ocelot)	12.04 LTS (Precise Pangolin)
No Open Ports	policy	policy	policy	policy	policy
Password hashing	md5	sha512	sha512	sha512	sha512
SYN cookies	--	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl
Filesystem Capabilities	--	kernel	kernel	kernel	kernel
Configurable Firewall	ufw	ufw	ufw	ufw	ufw
PR_SET_SECCOMP	kernel	kernel	kernel	kernel	kernel
AppArmor	2.1	2.5	2.6.1	2.7.0~beta1	2.7.0
SELinux	universe	universe	universe	universe	universe
SMACK	--	kernel	kernel	kernel	kernel
Encrypted LVM	alt installer	alt installer	alt installer	alt installer	alt installer
eCryptfs	--	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames
Stack Protector	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch
Heap Protector	glibc	glibc	glibc	glibc	glibc
Pointer Obfuscation	glibc	glibc	glibc	glibc	glibc
Stack ASLR	kernel	kernel	kernel	kernel	kernel
Libs/mmap ASLR	kernel	kernel	kernel	kernel	kernel
Exec ASLR	kernel (-mm patch)	kernel	kernel	kernel	kernel
brk ASLR	kernel (exec ASLR)	kernel	kernel	kernel	kernel
VDSO ASLR	kernel	kernel	kernel	kernel	kernel
Built as PIE	--	package list	package list	package list	package list
Built with Fortify Source	--	gcc patch	gcc patch	gcc patch	gcc patch
Built with RELRO	--	gcc patch	gcc patch	gcc patch	gcc patch
Built with BIND_NOW	--	package list	package list	package list	package list
Non-Executable Memory	PAE only	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation
/proc/$pid/maps protection	kernel & sysctl	kernel	kernel	kernel	kernel
Symlink restrictions	--	--	kernel	kernel	kernel
Hardlink restrictions	--	--	kernel	kernel	kernel
ptrace scope	--	--	kernel	kernel	kernel
0-address protection	kernel & sysctl	kernel	kernel	kernel	kernel
/dev/mem protection	kernel (-mm patch)	kernel	kernel	kernel	kernel
/dev/kmem disabled	kernel (-mm patch)	kernel	kernel	kernel	kernel
Block module loading	drop CAP_SYS_MODULES	sysctl	sysctl	sysctl	sysctl
Read-only data sections	kernel	kernel	kernel	kernel	kernel
Stack protector	--	kernel	kernel	kernel	kernel
Module RO/NX	--	--	kernel	kernel	kernel
Kernel Address Display Restriction	--	--	kernel	kernel	kernel
Blacklist Rare Protocols	--	--	kernel	kernel	kernel
Syscall Filtering	--	--	--	kernel	kernel

THE NEW RELEASE HAS MORE GREEN!

But what's actual relevant to kernel exploitation?

ONLY A HANDFUL RELEVANT TO KERNEL

Mitigating vuln classes

- Stack protector (2008)
- *mmap_min_addr (2008)*

Hampering exploitation

- RO/NX for kernel text/data (2008, 2011)

Reducing attack surface

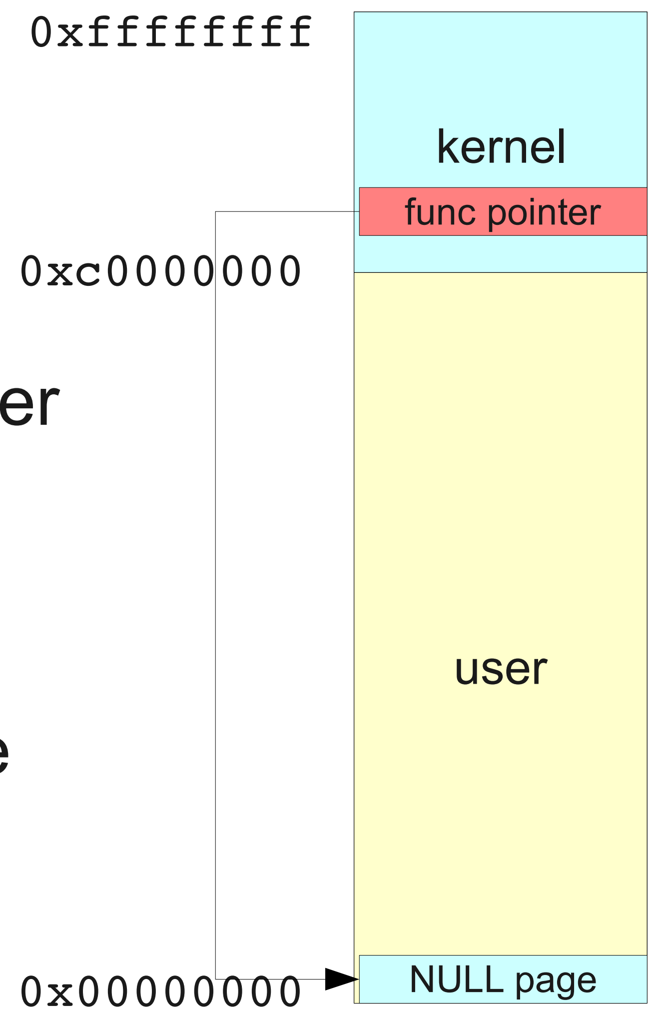
- *Packet family blacklisting (2011)*
- Syscall filtering (2012)

Plugging info leaks

- kptr_restrict (2011)
- dmesg_restrict (2011)
- *kallsyms (2011)*
- slabinfo (2011)

MMAP_MIN_ADDR MITIGATION

- NULL pointer dereferences
  - Used to be very exploitable on Linux kernels
  - mmap payload at NULL page, trigger
- mmap_min_addr
  - Limits lowest allow mmap region
  - Can't map the NULL page anymore
- (Mostly) mitigated in 2008
  - Tarjei: Win kernel in 2011?!?



FUN EXPLOITS IN PACKET FAMILIES

- Linux kernel will happily load ancient, obsolete, unmaintained packet family modules
  - Opens up HUGE attack surface
  - Just call socket(2) from unprivileged app
- Exploit-o-rama
  - Econet - LAN protocol from 1981
  - RDS - Proprietary transport protocol for Oracle
  - CAN - Internal broadcast bus in automobiles
- Distros _finally_ started blacklisting old modules

KERNEL SYMBOLS

- Kernel symbols
  - Favorite example of upstream kernel dysfunction
  - Most kernel exploits depend on them
    - Although sometimes out of convenience than necessity
    - prepare_kernel_cred/commit_creds combo
  - Exported through world-readable /proc/kallsyms

```
jono@apollo ~ $ ls -l /proc/kallsyms
-r--r--r-- 1 root root 0 Apr 25 03:23 /proc/kallsyms
```

TRYING TO RESTRICT KALLSYMS

- Recently, an attempt to make it non-world readable
  - What a concept!
- This LKML thread is full of gold:

```
[PATCH] kernel: make /proc/kallsyms mode 400 to
reduce ease of attacking
```

- For a simple one-liner patch:

```
-   proc_create("kallsyms", 0444, NULL, &kallsyms_operations);
+   proc_create("kallsyms", 0400, NULL, &kallsyms_operations);
```

LIFECYCLE OF A SECURITY PATCH ON LKML

"Hey, check out this totally reasonable security enhancement!"

"Sounds reasonable, but we should probably fix X, Y, and Z also!"

"Yeah, but if we do X, Y, and Z, we should probably boil the ocean too while we're at it!"

"Boiling the ocean is crazy talk, this will never work."

"...I give up."

“I GUESS I'LL REVERT IT”

- In this case, the patch was accepted!
  - For a couple days...
  - Until a user reported that the change broke klogd
  - Linus: “I guess I'll revert it”

```
ksyms = fopen(KSYMS, "r");
if (ksyms == NULL) {
    ...
    fclose(ksyms);
    return 0;
}
```

- So, the security patch that exposed a bug in an unmaintained log daemon was reverted...
  - Ubuntu included a slightly different kallsyms restriction in their next release

KALLSYMS ON LATEST UBUNTU

- Ubuntu LTS 12.04 (in final beta)
- Privileged user

```
root@ubuntu:/home/vm# id
uid=0(root) gid=0(root) groups=0(root)
root@ubuntu:/home/vm# cat /proc/kallsyms | grep commit_creds
ffffffff81091660 T commit_creds
ffffffff81a8fed0 r __ksymtab_commit_creds
ffffffff81aa6f70 r __kcrctab_commit_creds
ffffffff81ab46d2 r __kstrtab_commit_creds
root@ubuntu:/home/vm#
```

KALLSYMS ON LATEST UBUNTU

- Ubuntu LTS 12.04 (in final beta)
- Unprivileged user

```
vm@ubuntu:~$ id
uid=1000(vm) gid=1000(vm) groups=1000(vm),4(adm),24(c
lugdev),109(lpadmin),124(sambashare)
vm@ubuntu:~$ cat /proc/kallsyms | grep commit_creds
0000000000000000 T commit_creds
0000000000000000 r __ksymtab_commit_creds
0000000000000000 r __kcrctab_commit_creds
0000000000000000 r __kstrtab_commit_creds
vm@ubuntu:~$ 
```

KSYM SOURCES

- Thwarted!
- Where else are ksyms available?
  - System.map in /boot, /usr/src/linux, /lib/modules
  - vmlinux in /boot, /usr/src/linux, /usr/lib/debug

```
vm@ubuntu:~$ cat /boot/System.map-3.2.0-20-generic | grep commit_creds
cat: /boot/System.map-3.2.0-20-generic: Permission denied
```

  - /me shakes fist at Kees!

KSYMS FROM VMLINUZ

- One last local source
  - The vmlinuz kernel image itself in /boot
  - Compressed tokenized symbol table for the kernel internal resolution...not pretty to extract!
  - Even legit debug tools fail to find vmlinuz ksyms

```
-rw-r--r-- 1 root root  4965584 Mar 27 16:37 vmlinuz
```

- How to find these automatically?
  - ksymhunter
  - https://github.com/jonoberheide/ksymhunter

KSYMHUNTER

- ksymhunter on Ubuntu LTS 12.04

```
vm@ubuntu:~$ ./ksymhunter commit_creds 2>/dev/null
[+] trying to resolve commit_creds...
[+] resolved commit_creds using /boot/vmlinuz-3.2.0-20-generic
[+] resolved commit_creds to 0xffffffff81091660
```

- Can't Ubuntu fix this with a chmod?
  - Yes...but no...other “unfixable” ways to get ksyms
  - All distros run the same stock binary kernel image
- ksymhunter supports remote symbol lookups for common distros/kernels

CURRENT STATE OF KERNEL EXPLOITATION

- A decade ago...
  - Only required a write4 to escalate privileges
- How about in 2012 with all of those recent kernel mitigations?
  - Still only a write4!
  - And in many cases, even a weaker null write will work just fine
- Let's show this on Ubuntu 12.04

USING A WRITE4 AGAINST UBUNTU 12.04

- ksymhunt for apparmor_ops
  - apparmor_ops at 0xffffffff81c62fa0
- Pick any of the ~180 security ops function pointers to overwrite
  - Say, ptrace_access_check
- mmap privesc payload in userspace
  - ksymhunt for prepare_kernel_cred, commit_creds
- Trigger the poisoned func ptr
  - In this case by ptrace'ing a process

NULL WRITE WORKS TOO

- Can use weaker NULL write primitive
  - Partial overwrite of high order bytes of a fptr
- Or, without a partial overwrite
  - ksymhunt for mmap_min_addr
  - Reset mmap_min_addr to 0 with the NULL write
  - mmap privesc payload at NULL page
  - Overwrite fptr with NULL write
  - Trigger fptr to get code exec at NULL page

VANILLA SUMMARY

- Current upstream mitigations are incomplete
  - A write4 primitive is still as effective as it was a decade ago
- Upstream dysfunction is biggest hurdle to Linux kernel security
  - Brave souls get rejected upstream, push things into distros like Ubuntu, then hope for later upstream acceptance

AGENDA

- Vanilla kernel mitigations
- **Hardened kernel mitigations**
- Future mitigations

ON TO A HARDENED KERNEL

- How about a modern hardened kernel with PaX and grsecurity?
- A few of the relevant mitigations
  - KERNEXEC, UDEREF, HIDESYM, MODHARDEN, LOCKOUT, TPE, RANDKSTACK, REFCOUNT, USERCOPY, STACKLEAK
- More recently, via gcc plugins
  - kernexec, constify, stackleak, overflow

THE PAX TEAM

Visual approximation of pipacs

PAX VS WRITE4

- How does PaX fare against a write4 primitive?
  - KERNEXEC
    - Can't modify or introduce new code into kernel memory
  - UDEREF
    - Can't dereference any userspace pointers (whether code or data accesses)
  - HIDESYM
    - Can't discover any useful addresses or ksyms that could be used during exploitation

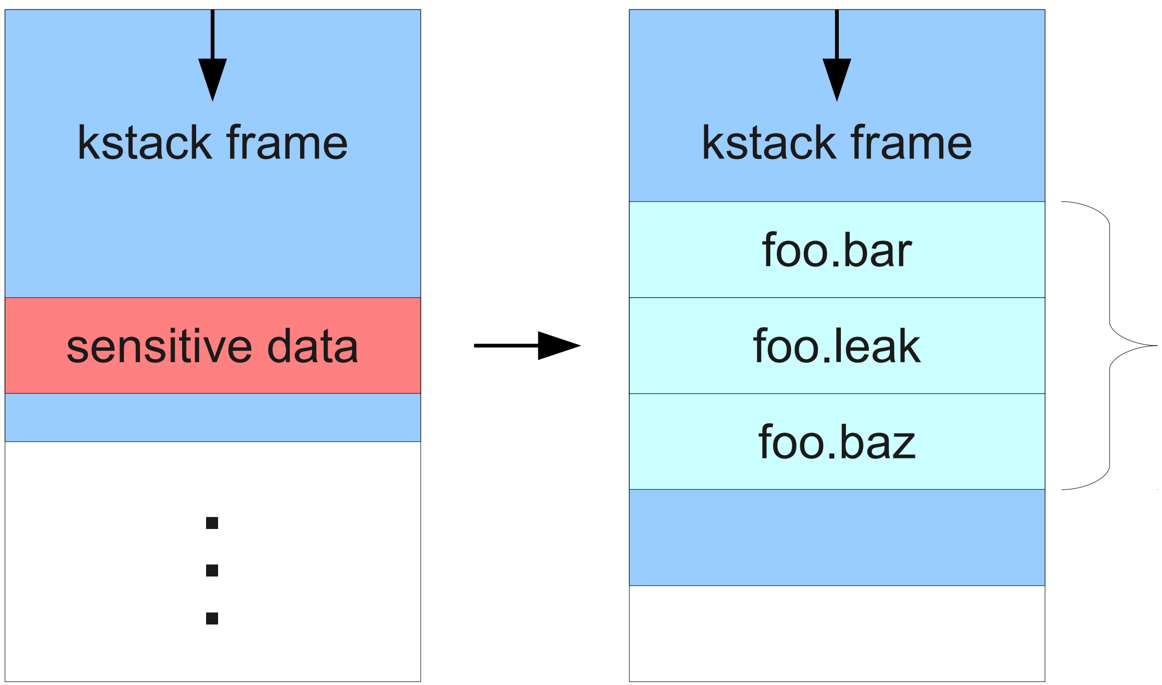
NEED TO KNOW SOMETHING

- So, write4 is pretty useless in the dark
- One way: arbitrary kmem disclosure
  - procfs (2005), sctp (2008), move_pages (2009), pktcdvd (2010)
- Just dump a bunch arbitrary kmem
  - But these are rare!
  - And in many instances, mitigated by grsec/PaX
- So far, busted by PaX

STACKJACKING

- In 2011, we came up with the stackjacking technique
- Combine primitives to defeat PaX
  - Arb write + kstack mem disclosure → arb read
  - Arb write + arb read → game over
- Kstack mem disclosures are relatively common unlike arbitrary reads
  - WTF is a kstack mem disclosure?

WHAT'S A KSTACK MEM DISCLOSURE?




```
struct foo {
  uint32_t bar;
  uint32_t leak;
  uint32_t baz;
};

syscall() {
  struct foo;
  foo.bar = 1;
  foo.baz = 2;
  copy_to_user(foo);
}
```

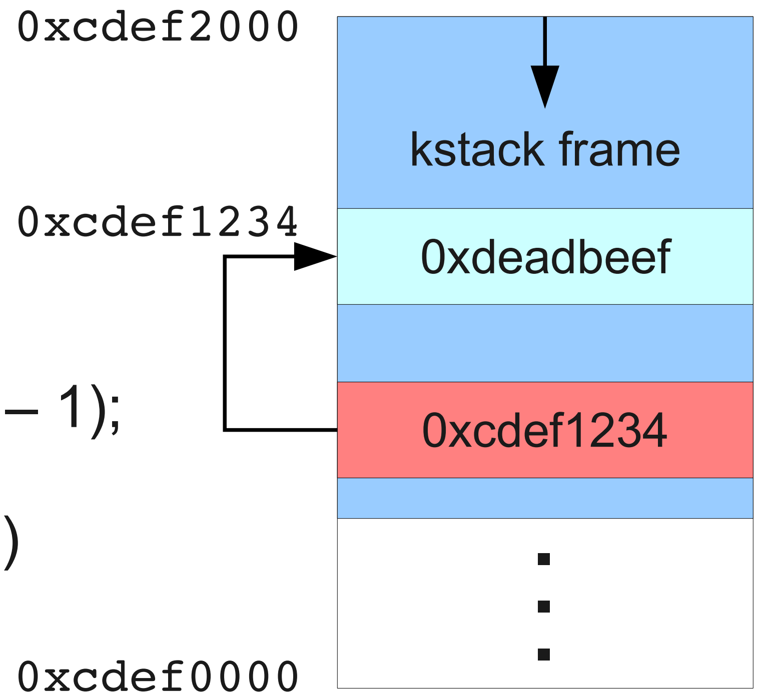
1) process makes syscall and leaves sensitive data on kstack

2) kstack is reused on subsequent syscall and struct overlaps with sensitive data

3) foo struct is copied to userspace, leaking 4 bytes of kstack through uninitialized foo.leak member

KSTACK SELF-DISCOVERY

- If we can leak an pointer to the kstack off the kstack, we can calculate the base address of the kstack



kstack_base = addr & ~(THREAD_SIZE – 1);

kstack_base = 0xcdef1234 & ~(8192 – 1)

kstack_base = 0xcdef0000

We call this ***kstack self-discovery***

HOW TO GET AN ARBITRARY READ

- We now have a known reference point in kernel memory, our own kstack
  - Couple of complicated techniques to turn the write+kleak into an arbitrary read
  - Obergrope and Rosengrope techniques
- See SummerCon slides for full details

http://jon.oberheide.org/files/stackjacking-summercon11.pdf

PAX RESPONSE

- Moved thread_info off kstack
  - Kills Rosengrope technique
- RANDKSTACK enhancements
  - Randomizes kesp on each syscall
  - Make Obergrope a bit unreliable
- USERCOPY enhancements
  - Hardened to prevent task_struct → userspace
- STACKLEAK plugin
  - Clears the kstack after each syscall

PIPACS IS GREAT AT RUINING PARTIES

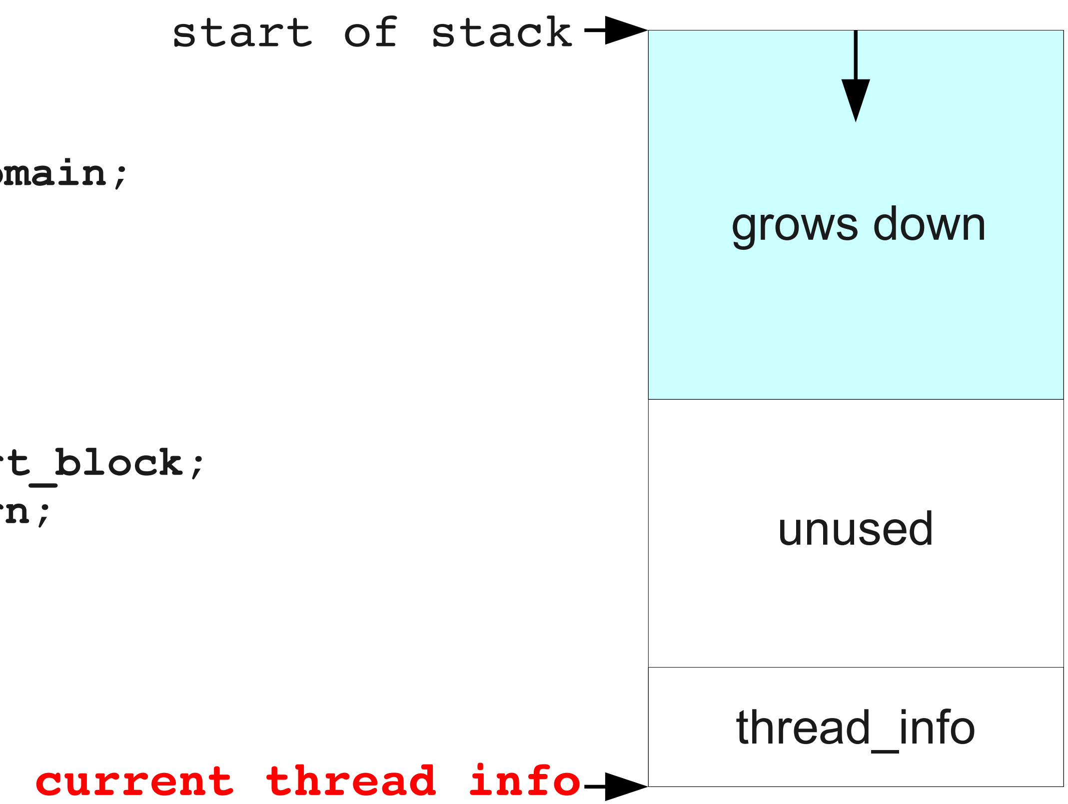
- Stackjacking technique on life support thanks to the fury of pipacs
  - But lives on as an effective technique against vanilla kernels...probably for a long long time.
  - ^ Why you should focus on exploiting PaX, then reap the years of benefits against vanilla
- A good example of scary-thorough mitigation response

KERNEL STACK OVERFLOWS

- Stackjacking against PaX got me hot on kernel stack bizness
  - Heap research is for suckers
- Discovered that stack overflows are exploitable in the Linux kernel
  - _Not_ stack-based buffer overflows
- Again, pipacs quickly ruined my kernel stack overflow party :-(

METADATA ON THE KERNEL STACK

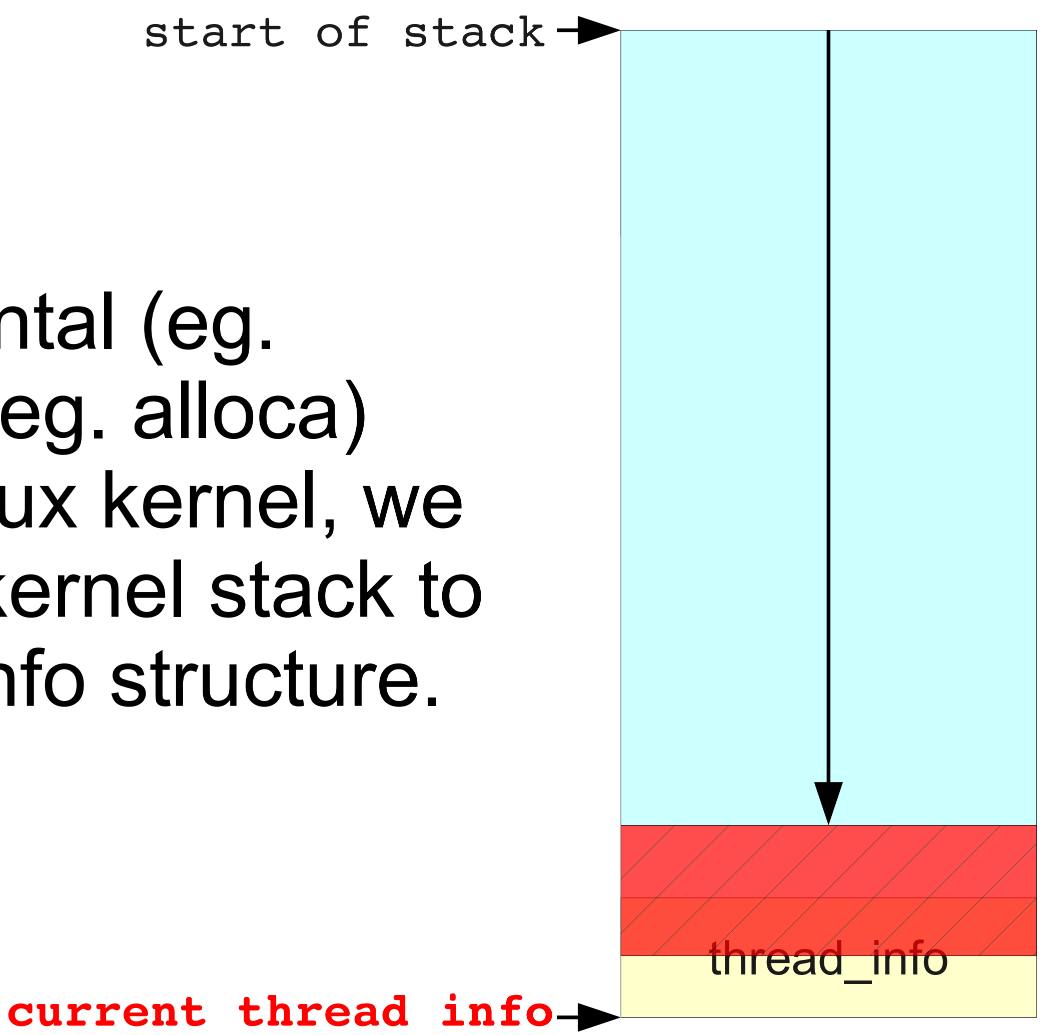
```
struct thread_info {
    struct task_struct *task;
    struct exec_domain *exec_domain;
    __u32 flags;
    __u32 status;
    __u32 cpu;
    int preempt_count;
    mm_segment_t addr_limit;
    struct restart_block restart_block;
    void __user *sysenter_return;
#ifdef CONFIG_X86_32
    unsigned long previous_esp;
    __u8 supervisor_stack;
#endif
    int uaccess_err;
};
```



thread_info struct is at the base of kstack!

EXPLOITING STACK OVERFLOWS

If we control an incremental (eg. recursion) or allocation (eg. alloca) stack overflow in the Linux kernel, we can cause our thread's kernel stack to collide with the thread_info structure.



TARGETING THREAD INFO

- What would the overflow collide with?
  - uaccess_err
    - No security impact, but safe to clobber
  - **restart_block**
    - A function pointer, BINGO!
  - **addr_limit**
    - Define u/k boundary, BONGO!
  - preempt_count .. task_struct
    - Pretty sensitive, avoid clobbering

```
struct restart_block {
    long (*fn)(struct restart_block *);
    union {} /* safe to clobber */
};
```

```
access_ok()/__range_not_ok():

Test whether a block of memory
is a valid user space address.

addr + size > addr_limit.seg
```

CONTROLLING THE CLOBBER

- Can we control the clobbering value?
  - Incremental overflow: tip of the stack, unlikely
  - Allocation overflow: VLA values, maybe
- Good news, don't need *much* control!
- Two categories:
  - Value represents a kernel space address
    - Value > TASK_SIZE
  - Value represents a user space address
    - Value < TASK_SIZE

CLOBBER TO CODE EXEC

- If value < TASK_SIZE
  - Clobber restart_block fptr with userspace value
  - mmap privesc payload at that address in userspace
  - Trigger fptr via syscall(SYS_restart_syscall);
- If value > TASK_SIZE
  - Clobber addr_limit with a high kernel space value
  - You can now pass copy_from_user()/access_ok() checks up to that kernel address
  - So we can read(2) from a fd and write into kmem

STACK OVERFLOW EXPLOITATION

- thread_info clobbering technique
  - Will work in the common case for Linux kernel stack overflows (recursion or VLAs)
- More advanced stack overflow techniques are possible
  - See Infiltrate slides for half-nelson.c exploit

http://jon.oberheide.org/files/infiltrate12-thestackisback.pdf

AGAIN, PIPACS HURTS MY FEELINGS

- STACKLEAK plugin enhancements
  - Instruments any functions with alloca functionality with sanity checks to prevent stack overflows.
- Recursive vulns should still be in play though
  - Especially on vanilla kernels
  - Again, target PaX and you're reap the rewards against vanilla

HARDENED SUMMARY

- Write primitive not sufficient against modern PaX-hardened kernel
  - Need info leak, but can be rare
- Hardened kernel years ahead of vanilla in mitigations
- A couple new exploitation techniques
  - Stackjacking and stack overflows
  - Promptly demolished by the PaX team
  - But still very applicable to vanilla kernel

AGENDA

- Vanilla kernel mitigations
- Hardened kernel mitigations
- **Future mitigations**

FUTURE MITIGATIONS

- Within the next few months
  - SMEP on Intel's Ivy-Bridge
  - More GCC plugins for PaX (overflow)
- Within a year or two
  - More NX/RODATA/const progress
  - More address leakage plugging
  - In-kernel ASLR
- Within a decade
  - KERNSEAL ;-P

SMEP IN INTEL IVY-BRIDGE

x86, cpu: Enable/disable Supervisor Mode Execution Protection

```
author     Fenghua Yu <fenghua.yu@intel.com>
           Wed, 11 May 2011 23:51:05 +0000 (16:51 -0700)
committer  H. Peter Anvin <hpa@linux.intel.com>
           Wed, 18 May 2011 04:22:00 +0000 (21:22 -0700)
commit     de5397ad5b9ad22e2401c4dacdf1bb3b19c05679
tree       e8c612c4f84efe458d8ec3c23f1bfd834f20c0f2      tree | snapshot
parent     dc23c0bccf5eea171c87b3db285d032b9a5f06c4      commit | diff
```

```
x86, cpu: Enable/disable Supervisor Mode Execution Protection

Enable/disable newly documented SMEP (Supervisor Mode Execution Protection) CPU
feature in kernel. CR4.SMEP (bit 20) is 0 at power-on. If the feature is
supported by CPU (X86_FEATURE_SMEP), enable SMEP by setting CR4.SMEP. New kernel
option nosmep disables the feature even if the feature is supported by CPU.

[ hpa: moved the call to setup_smep() until after the vendor-specific
  initialization; that ensures that CPUID features are unmasked.  We
  will still run it before we have userspace (never mind uncontrolled
  userspace). ]

Signed-off-by: Fenghua Yu <fenghua.yu@intel.com>
LKML-Reference: <1305157865-31727-1-git-send-email-fenghua.yu@intel.com>
Signed-off-by: H. Peter Anvin <hpa@linux.intel.com>
```

Documentation/kernel-parameters.txt	diff | blob | history
arch/x86/kernel/cpu/common.c	diff | blob | history

Supposedly proposed to Intel by Joanna/ITL in 2008:

http://theinvisiblethings.blogspot.com/2011/06/from-slides-to-silicon-in-3-years.html

SMEP ILLUSTRATED

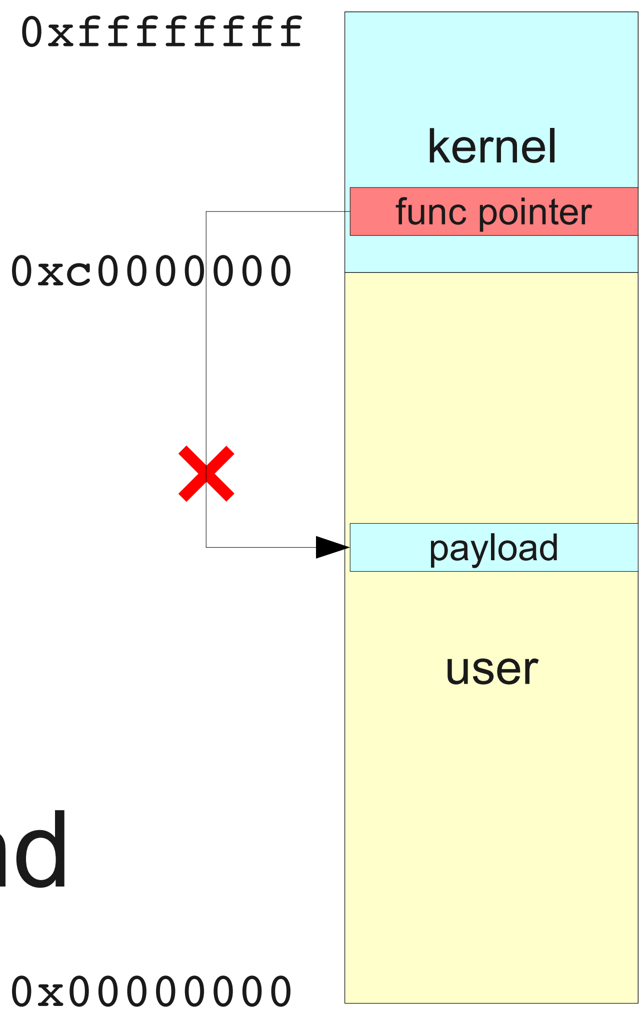
Supervisory Mode Execute Protection (SMEP)

- **Ivy Bridge introduces SMEP to help prevent Escalation of Privilege (EoP) security attacks**
  - Prevents execution out of untrusted application memory while operating at a more privileged level
  - If CR4.SMEP set to 1 and in supervisor mode (CPL<3), instructions may not be executed from a linear address for which the user mode flag is 1
  - Available in both 32- and 64-bit operating modes
  - SMEP is enumerated via CPUID.7.0.EBX[7]

Simply put: SMEP blocks the kernel from unsafely dereferencing code in userspace

SMEP MITIGATION

- Common exploit flow
  - Use write primitive to modify a function pointer in kmem
  - Mmap privesc payload in userland
  - Trigger function pointer and redirect control flow to userspace
- SMEP blocks this userland code access from ring0



SMEP WEAKNESSES

- ROP beats SMEP
  - Also, writing payload into W+X kmem
  - Also, stackjacking
- Doesn't prevent user data derefs (r/w)
  - Basically a subset of PaX's UDEREF
  - But at least hardware supported (w/o segmentation)
- Haven't seen SMEP in the wild yet
  - Windows 8 support? Yes according to Tarjei.
  - But Ivy-Bridge CPUs just launched 2 days ago (April 23rd, 2012)

http://vulnfactory.org/blog/2011/06/05/smep-what-is-it-and-how-to-beat-it-on-linux/

CONCLUSIONS

- Not a whole lot has changed
  - Offense drives defense drives offense, etc
  - But when there's no defense...
- Vanilla
  - As usual, years behind in mitigations
  - write4 (or weaker) sufficient a few years to come
  - Upstream resistance, lack of completeness
- PaX
  - As usual, a crystal ball into the future

QUESTIONS?

Jon Oberheide
jon@oberheide.org
@jonoberheide