There's a party at ring0...

(...and you're invited)

Tavis Ormandy, Julien Tinnes

Introduction

- All systems make some assumptions about kernel security.
  - Sometimes a single kernel flaw can break the entire security model.
  - Things like sandboxing in Google Chrome and Android make us even more dependent on kernel security.
- Over the last year we've been involved in finding, fixing and mitigating some fascinating kernel bugs, and we want to share some of the details.
- We will discuss some of the ways to protect the kernel from malicious userland code.

The kernel as a target

Local privilege escalation

- You have arbitrary code execution on a machine
- You want to escalate (or change) privileges
- What can you target?
  - Processes with more/other privileges (Running deamons, suid binaries you can execute on Unix)
  - The kernel
    - Big code base
    - Performs complex, error-prone tasks
    - Responsible for the security model of the system

The Linux kernel as a local target

- The Linux kernel has been a target for over a decade
- Memory / memory management corruption vs. logical bug

- The complexity of a kernel makes for more diverse and interesting logical bugs

- Fun logical bugs include:
  - ptrace() / suidexec (Nergal, CVE-2001-1384)
  - ptrace() / kernel threads (cliph / Szombierski, CVE-2003-0127)
  - /proc file give-away (H00lyshit, CVE-2006-3626)
  - prctl suidsafe (CVE-2006-2451)

Linux kernel mm corruption bugs

- Tend to be more interesting and diverse than userland counterpart
  - Complexity of memory management
  - Interesting different paradigm (the attacker finely controls a full address space)

- cliph / ihaquer do_brk() (CVE-2003-0961)
- cliph / ihaquer / Devine / others "Lost-VMA"-style bugs (check isec.pl)
- Couple of "classic" overflows
- Null (or to-userland) pointer dereferences

Escapes through the kernel

- Exploiting the kernel is often the easiest way out of:
  - chroot() jails
  - Mandatory access control
  - Container-style segregation (vserver etc..)
- Using those for segregation, you mostly expose the full kernel attack surface
  - Virtualization is a popular alternative

- MAC makes more sense in a full security patch such as grsecurity.

Windows and local kernel bugs

- Traditionally were not considered relevant on Windows
- Changed somewhat recently
    - Increased reliance on domain controls
    - Use of network services
    - introduction of features like protected mode / integrity levels

- This has changed in the last few years and Windows is roughly in the same situation as Linux now
    - With a bit less focus on advanced privilege separation and seggregation (Lacks MACs for instance)

Remotely exploitable kernel bugs

- Public exploits are still quite rare on Linux
- Notable exceptions
  - Wifi drivers (big attack surface, poorly written code)
    - See few exploits by Stéphane Duverger, sgrakkyu or Julien
    - Read Stéphane's paper
  - *s*grakkyu's impressive SCTP exploit
    - (Read his article co-written with twiz in Phrack)
  - Few others

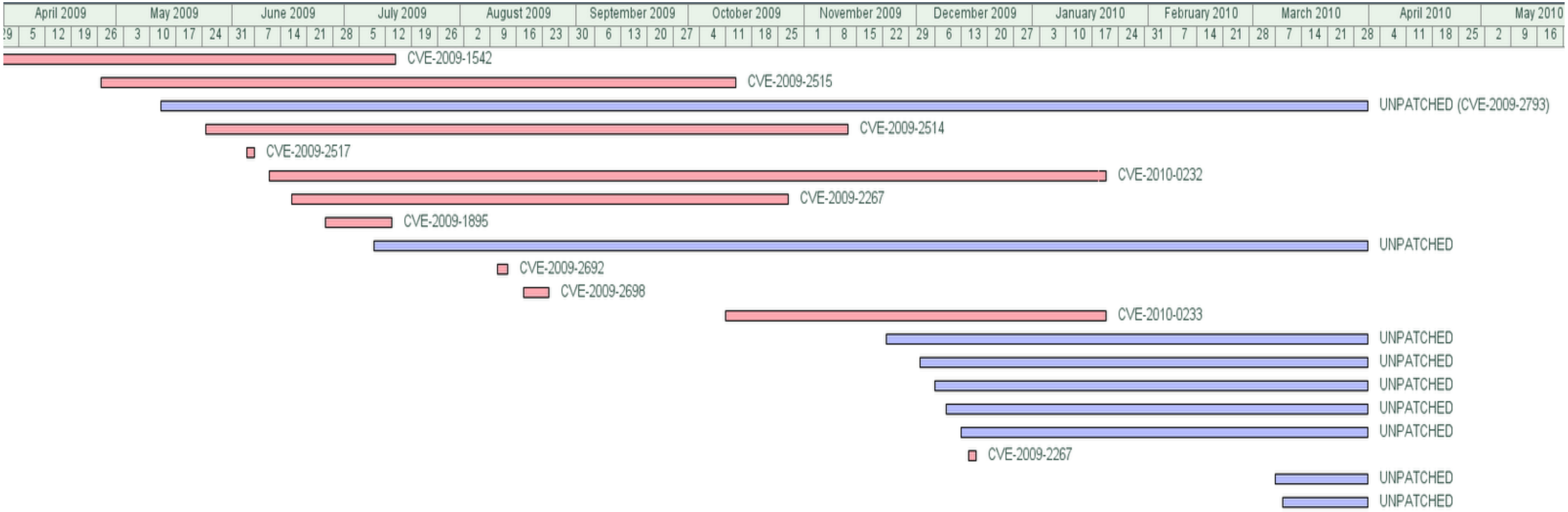
Remotely exploitable kernel bugs (2)

- Have been quite popular on Windows for at least 6/7 years
  - Third party antivirus and personal firewall code
  - GDI-related bugs
  - TCP/IP stack related ones (Neel Mehta et al.)
  - Immunity's SMBv2 exploit

- Web browsers changed the game
  - The threat model for in-kernel GDI is now different
  - See also the remotely exploitable NVidia drivers bug on Linux
  - Stay tuned...

Some bugs from the last year

Timeline


April 2009
May 2009
June 2009
July 2009
August 2009
September 2009
October 2009
November 2009
December 2009
January 2010
February 2010
March 2010
April 2010
May 2010
CVE-2009-1542
CVE-2009-2515
UNPATCHED (CVE-2009-2793)
CVE-2009-2514
CVE-2009-2517
CVE-2010-0232
CVE-2009-2267
CVE-2009-1895
UNPATCHED
CVE-2009-2692
CVE-2009-2698
CVE-2010-0233
UNPATCHED
UNPATCHED
UNPATCHED
UNPATCHED
UNPATCHED
CVE-2009-2267
UNPATCHED
UNPATCHED

Exposing Kernel Attack Surfaces

- There are many entrypoints for attackers to expose kernel attack surface, apart from system calls there are also
  - Ioctls, devices, kernel parsers
  - Filesystems, network protocols
  - Fonts, Bitmaps, etc. (primarily Windows)
  - Executables formats (COFF, ELF, a.out, etc.)
  - And so on.

- Perhaps one under appreciated entrypoint is dpl3 interrupt handlers, so we decided to take a look.

Windows 2003 KiRaiseAssertion Bug

- In Windows Server 2003, Microsoft introduced a new dpl3 (accessible to ring3 code) IDT entry (KiRaiseAssertion in the public symbols).

```
0008:f9d91d22  66baff0c                  mov          dx,0cff
FrameEBP—RetEIP——Syms—Symbol
80894600 808873A2   N   intelppm!.text+1CA2
808970C0 00000000   Y   ntoskrnl!@KiIdleLoop+000A
(DISPATCH)-KTEB(80896E40)-TID(0000)—intelppm!.text+1CA1
:filter "DPL=3" white+blue
:idt 2c
002C   IntG32    0008:80883600  DPL=3   P    _KiRaiseAssertion
:
     Enter a command (H for help)                                          Idle
```

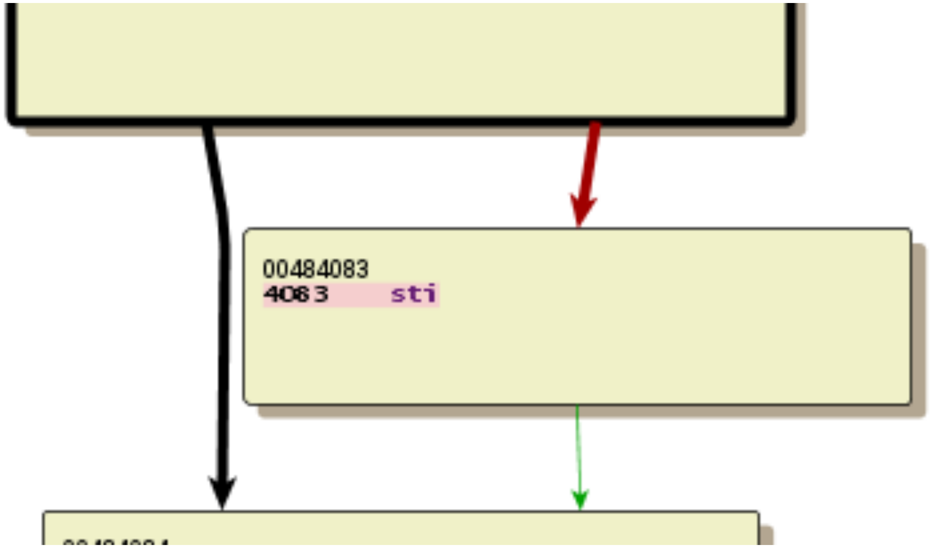
- This makes int 0x2c roughly equivalent to RaiseException (STATUS_ASSERTION_FAILED).
- I've never seen this feature used, but analysis revealed an interesting error; interrupts were not enabled before the exception dispatch!
- This bug has two interesting characteristics...

Windows 2003 KiRaiseAssertion Bug

- Tiny exploit (4 bytes)...

```
00000000  31E4            xor esp,esp
00000002  CD2C            int 0x2c
```

- Tiny patch (1 byte)...



```
00484083
4063      sti
```

Page Fault Exceptions

- A page fault exception occurs when code...
  - Attempts to access a non-present page
  - Has insufficient privilege to access a present page
  - Various other paging related errors

The handler is passed a set of flags describing the error:

- `I/D` - Instruction / Data Fetch
- `U/S` - User / Supervisor Mode
- `W/R` - Read / Write access
- `P` - Present / Not present

Supervisor Mode

- If the processor is privileged when the exception occurs, the supervisor bit is set.
- Operating system kernels use this to detect when special conditions occurs
  - This could mean a kernel bug is encountered.
  - Oops, BugCheck, Panic, etc.
  - Or some other unusual low-level event
- Can also happen in specific situations (copy-from-user etc...)
- If the processor can be tricked into setting the flag incorrectly, ring3 code can confuse the privileged code handling the interrupt.

VMware Invalid #PF Code

- By studying the machine state while executing a Virtual-8086 mode task, we found a way to cause VMware to set the supervisor bit for user mode page faults.
- Far calls in Virtual-8086 mode were emulated incorrectly.
  - When the cs:ip pair are pushed onto the stack, this is done with supervisor access.
  - We were able to exploit this to gain ring0 in VMware guests.
- The linux kernel checks for a magic CS value to check for PNPBIOS support.
  - But...
  - Because we're in Virtual-8086 mode we must be permitted any value cs.

Exploiting Incorrect U/S Bit

- We can exploit this error :-)
- We mmap() our shellcode at NULL, then enter vm86 mode.
    - mmap_min_addr was beginning to gain popularity at the time we were working on this, so we bypassed that as well (CVE-2009-1895) :-)
- When we far call with a non-present page at ss:sp, a #PF is delivered.
- Because we can spoof arbitrary cs, we set a value that the kernel recognises as a PNPBIOS fault.
- The kernel tries to call the PNPBIOS fault handler.
- But because this is not a real fault, the handler will be NULL.
- => r00t :-)

Exploiting Incorrect U/S Bit

- Triggering this issue was simple, we used a code sequence like this:

```
vm.regs.esp = 0xDEADBEEF;
vm.regs.eip = 0x00000000;
vm.regs.cs = 0x0090;
vm.regs.ss = 0xFFFF;

CODE16("call 0xaabb:0xccdd", code, codesize);

memcpy(REAL(vm.regs.cs, vm.regs.eip), code, codesize);

vm86(Vm86Enter, &vm);
```

More Page Fault Fun

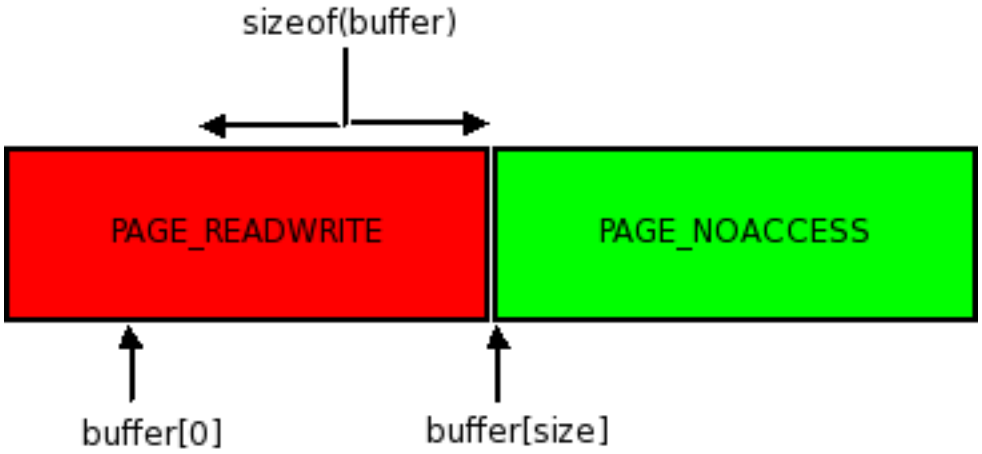
- If the kernel ever trusts data from userspace, a security issue may exist.

- However, it's worth remembering that it's not just the data that users control, it's also the presence or absence of data.

- By claiming to have more data available than we really do, we can reach lots of unusual error paths.
  - This is especially true on Windows where the base system types are large inter-dependent structures.
- We found an interesting example of this problem on Windows NT, resulting in a privilege escalation.
- MS10-015, a double-free in NtFilterToken()

Windows NT NtFilterToken() Bug

- NtFilterToken() is the system service that makes routines like CreateRestrictedToken() work.
- NtFilterToken() would pass a (void **) to a helper routine, which would be used to store the captured data.
- I can force the capture to fail by claiming the SID is bigger than it really is, and forcing the structure to straddle a page boundary.



Windows NT NtFilterToken() Bug

- On error, the helper routine releases but doesn't reset the (void **) parameter, which NtFilterToken() will release again!
- The kernel detects a double free and BugChecks, so we only get one attempt to exploit this...
- We need to get the buffer reallocated a small window. This is possible, but unfortunately is unavoidably unreliable.

Example Code: http://bit.ly/b9tPqn

Windows NT TTF Parsing Vulnerability

"Moving [...] the GDI from user mode to kernel mode has provided improved performance without any significant decrease in system stability or reliability."

(Windows Internals, 4th Ed., Microsoft Press)

- GDI represents a significant kernel attack surface, and is perhaps the most easily accessible remotely.
- We identified font parsing as one of the likely weak points, and easily accessible via Internet Explorer's `@font-face` support.
- This resulted in perhaps our most critical discovery, remote ring0 code execution when a user visits a hostile website (even for unprivileged or protected mode users).

Windows NT TTF Parsing Vulnerability

- The font format supported by Internet Explorer is called EOT (Embedded OpenType), essentially a trivial DRM layer added to TTF format fonts.
- EOT also defines optional sub-formats called CTF and MTX (in which we also identified ring3 vulnerabilities, see MS10-001 and others), but are essentially TTF with added compression and reduced redundancy.
    - See http://www.w3.org/Submission/2008/SUBM-EOT-20080305/
- EOT also adds support for XOR encryption, and other advanced DRM techniques to stop you pirating Comic Sans.
- The t2embed library handles reconstructing TTF files from EOT input, including decryption and so on, at which point GDI takes over.

Windows NT TTF Parsing Vulnerability

- We found multiple integer errors when GDI parses TTF directories (these directories simply describe the position of each table in the file).
- This code is executed at ring0, and was essentially unchanged since at least NT4.
- Microsoft wasn't alone, most other implementations we tested were vulnerable, but as the decoder ran at ring0 on Microsoft platforms, the impact was far more serious.

NULL pointer dereferences

- To-userland pointer dereferences
  - If at any time the kernel trusts data in user space, privilege escalation is likely

- NULL dereferences are a common error
  - Common initialization value / error-returned as pointers
  - NULL is a special value in C, but has no special meaning to the underlying hardware on x86

NULL pointer dereferences

- Interestingly, they used to not be exploitable in Linux 2.0 / i386
  - Segmentation was used
  - A dereferenced pointer without a segment override would not reach userland
  - Wrong pointer dereferences didn't become "to-userland" pointer dereferences
  - thus their destination would be harder to control
- Interesting threads in ~2004/2005, where many Linux kernel developers did not understand the security consequences
- Was still the case for some of them until recently
- Will talk about mmap_min_addr later

Linux kernel sock_sendpage

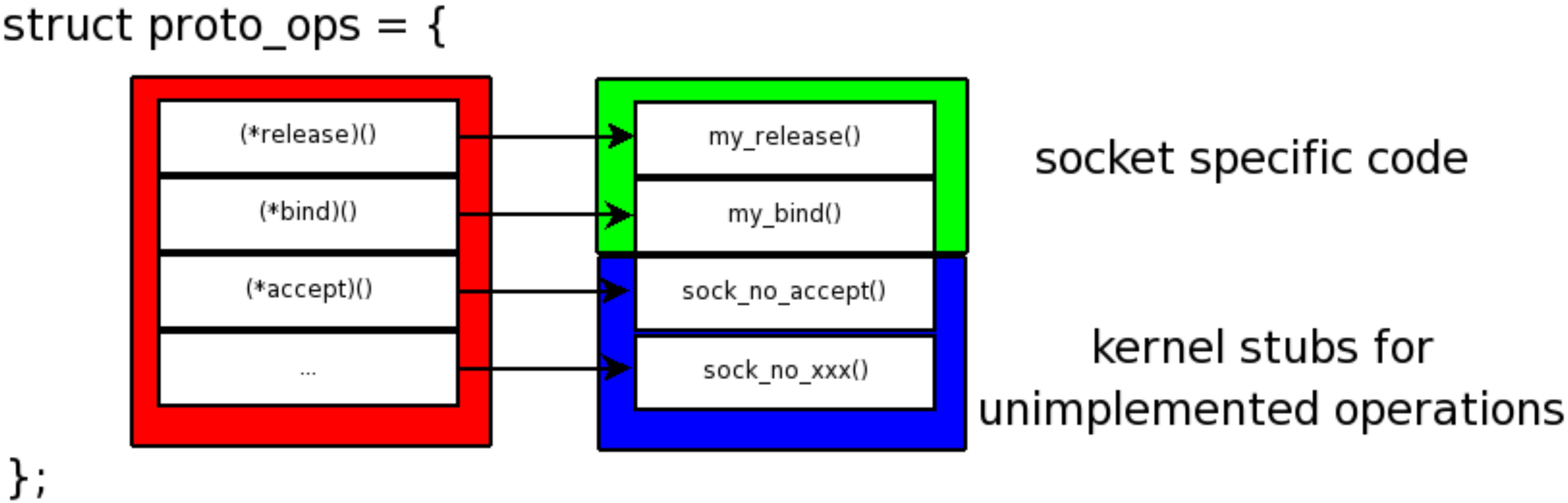
- CVE-2009-2692, found it last August
- Affected all 2.4 and 2.6 kernels to date
- Every major distribution shipped vulnerable kernels
- NULL function pointer dereference
- Trivial to exploit

Linux kernel sock_sendpage

- Every socket in the Linux kernel has a set of function pointers associated with it called `proto_ops` (Protocol Operations).
- Implement the various operations that can be performed on a socket, e.g. accept, bind, shutdown, and so on.
- The general socket management code doesn't have to know about the underlying transport or protocol, because this is all abstracted away.

Linux kernel sock_sendpage

- The proto_ops definition is available in include/linux/net.h



struct proto_ops = {

(*release)()

(*bind)()

(*accept)()

...

my_release()

my_bind()

sock_no_accept()

sock_no_xxx()

socket specific code

kernel stubs for unimplemented operations

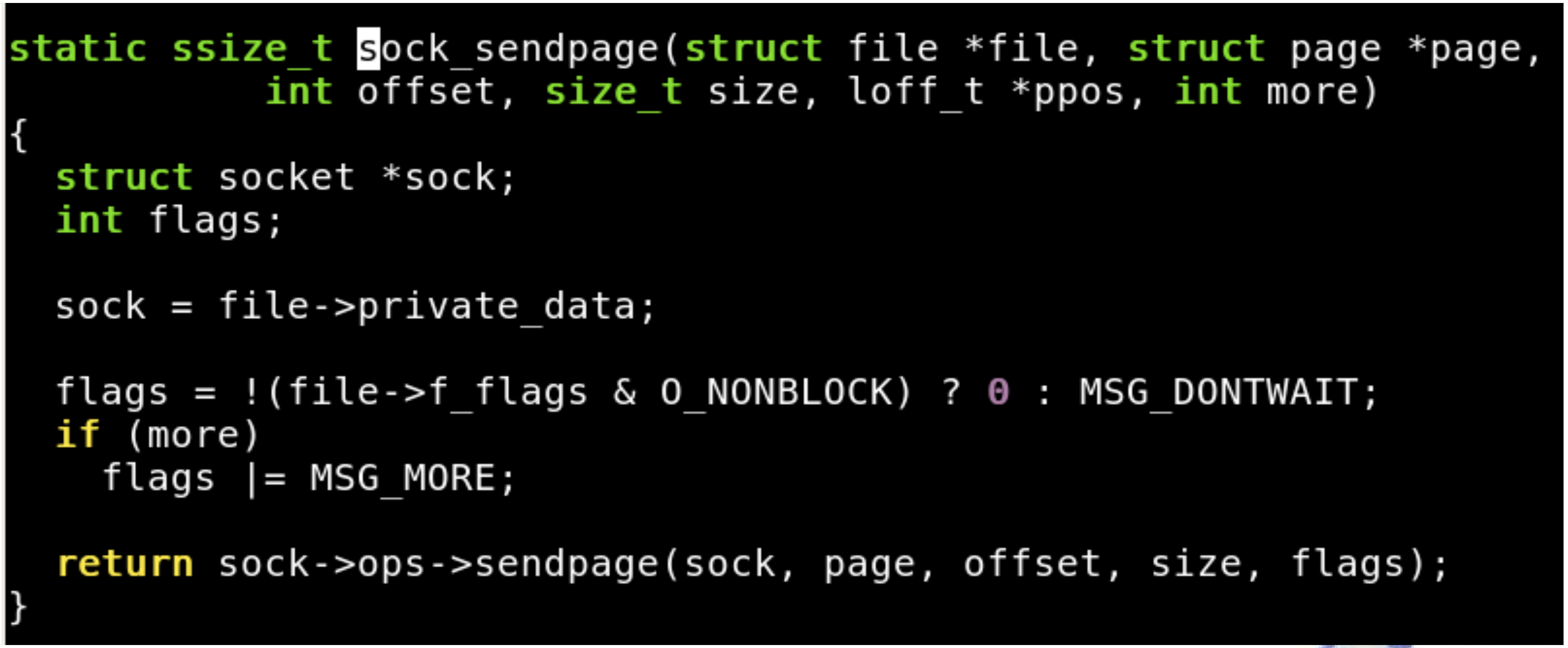
};

Linux kernel sock_sendpage

- Drivers implement the operations they support and point operations they don't support to pre-defined kernel stubs
- This model is very fragile if you add a new operation:
  - You need to update all drivers and point the new operation to a stub (or implement it)
  - It's a lot of code to update, including macros used for initialization

Linux kernel sock_sendpage

When sock_sendpage() was added, it assumed the corresponding proto_ops field would always be correctly initialized



```c
static ssize_t sock_sendpage(struct file *file, struct page *page,
           int offset, size_t size, loff_t *ppos, int more)
{
  struct socket *sock;
  int flags;

  sock = file->private_data;

  flags = !(file->f_flags & O_NONBLOCK) ? 0 : MSG_DONTWAIT;
  if (more)
    flags |= MSG_MORE;

  return sock->ops->sendpage(sock, page, offset, size, flags);
}
```

Linux kernel sock_sendpage

- Unfortunately, a lot of drivers did not get properly updated
- The SOCKOPS_WRAP macro had a bug
    - Used by many drivers to initialize proto_ops
    - Making them vulnerable in any case

- .sendpage was implicitly initialized to NULL for many drivers
- And sock_sendpage() would start executing code at NULL
- Map your shellcode at NULL and it'll get executed

- We wrote a trivial exploit that we shared with vendors, spender released one with a fully featured shellcode

Linux udp_sendmsg()

- CVE-2009-2698, released in August
- It's possible to trigger a codepath in udp_sendmsg() that will result in calling ip_append_data() with a NULL routing table

- This time, it's a data NULL pointer dereference
  - An attacker will control kernel's data (rtable) through address NULL
  - Still exploitable

Linux fasync use after free

- Drivers which want to provide asynchronous IO notification have a linked list of fasync_struct containing fds (and the corresponding *file* structure) to notify
- The same *file* structure could be in multiple fasync_struct lists
  - Most notably a special one for locked files
- If the *file* was locked, and then closed, a logical bug would remove the *file* structure only from the special locked files linked list and free the file structure
- The driver would still have a reference to this freed *file* structure

- Gabriel Campana wrote an exploit
  - Tricky to make it reliable

NetBSD's Iret #GP handling failure

- An inter-privilege iret can fail before the privilege switch occurs
- For instance, if restored EIP is past the code segment limit
  - #GP will occur
  - ... while in kernel mode
  - No privilege switch occurs, so no stack switch
  - No saved stack information on the trap frame
- But NetBSD expects a full trap frame
- Due to the non executable stack emulation, this can happen during a legitimate program's execution

Windows NT #GP Trap Handler Bug

- After discovering these fun bugs in interrupt handlers, we audited the remaining interrupt handlers.
- One section of code in KiTrap0D (the name of the #GP trap handler in the public symbols) appeared to trust the contents of the trap frame.
- The code itself is a component of the Virtual-8086 monitor, introducing lots of fun special cases that few people are familiar with.
- It took another two weeks of research to figure out how to reach the code and write a reliable exploit, but the end result was a fascinating and ancient vulnerability in the core of Windows NT.

BIOS Calls and Sensitive Instructions

- If you can remember programming MS-DOS, you'll be familiar with int 0x21 to invoke system services.
- BIOS calls were then used to interact with hardware, most people will remember int 0x10 was used for video related services.
- In Virtual-8086 mode, these services are intercepted by the monitor code.
- "Sensitive Instructions" is the term given by Intel to any action in Virtual-8086 mode that real mode programs expect to be able to perform, but cannot be permitted in protected mode.
- These actions trap, and the kernel is given an opportunity to decide how to proceed.

Windows NT #GP Trap Handler Bug

- The design of the Virtual-8086 monitor in Windows NT has barely changed since it's original implementation in the early nineties.
- In order to support BIOS service routines, a stub exists in the #GP trap handler that restores execution context from the trap frame.
- Access to this code is authenticated, but by magic values that I knew we could forge from our work on vmware.
- However, There were several hurdles we needed to overcome before we could reach this code, but each one was an interesting exercise.

Windows NT #GP Trap Handler Bug

- The Virtual-8086 monitor is exposed via the undocumented system service NtVdmControl().
    - This call is authenticated, a process is required to have a flag called VdmAllowed in order to access it.
- We found that the VdmAllowed flag can only be set with SeTcbPrivilege (which is only granted to the most privileged code).
- We were able to defeat this check by requesting the NTVDM subsystem, and then using CreateRemoteThread() to execute within the authorised subsystem process.
- Now that we were authorised to access NtVdmControl(), we could try to reach the vulnerable code...

Windows NT #GP Trap Handler Bug

- The vulnerable code was guarded by a test for a specific cs:eip pair in the trap frame.
- We can forge trap frames by making iret fail, but we still can't request iret return into arbitrary code segments, as this would be an obvious privilege escalation (rpl0).
- But...cs loses it's special meaning in Virtual-8086 mode, which is guaranteed to always be cpl3, so it's reasonable to request any value.
- We still need to cause iret to #GP, we did this by setting eflags.TF=1, when returning. This is considered "sensitive", and we get #GP instead.
- This is poorly documented by Intel, but is self-evident from experimentation.

Automation and fuzzing

System Call Exploration

- On Windows, the system call interface is complex, unstable, unsupported and undocumented.
    - It's also vast, with ~1400 entries (cf. Linux ~300).
- They are designed to only ever be called by Microsoft code.
- Rarely see exposure to malformed parameters, so simple fuzzing will generally expose interesting bugs.
- The parameters are often complex objects, multiple levels deep with large inter-dependencies. Pathological parameters will often reach rarely exercised code.
- Of course, the kernel also parses fonts, pixmaps, and other complex formats all at ring0...
    - All excellent fuzz candidates!

System Call Fuzzing

- Trivial fuzzing will find Windows bugs.
- Fuzzing will find Linux bugs, but the task is not so trivial.

- We've developed some interesting techniques for fuzzing on Linux, and have had some success finding minor bugs.

Protecting the kernel and its attack surface

TPE (trusted path executables)

- A reasonably old concept to prevent local privilege escalation
- Aims to prevent gaining arbitrary code execution in the first place
- A naïve way of doing it on Linux was to mount user-writable PATHs "noexec"
  - Easy bypass by going through the dynamic loader
  - grsecurity had a good gid/uid based one for years
  - Now *could* actually works ("noexec" prevents file mappings as PROT_EXEC)
- This approach is gaining popularity on the Windows platform (white listing)

TPE (drawbacks)

- "Arbitrary code execution" should not only mean "arbitrary opcodes"
    - You can exploit lots of bugs from a Python or Ruby interpreter
- gdb
- The threat model is changed for many binaries
    - a local vulnerability in 'nethack' now becomes useful
    - or those zsh / make vulnerabilities

- Of course, useless if the attacker already has arbitrary code execution
    - Browser sandbox
    - OpenSSH / vsftpd 'privilege-separated' sandbox

Sandboxing and attack surface reduction

- Ideally, a process could opt-out from some kernel features it does not require
- Linux does not have any real "discretionary privilege dropping facility"
  - Most of the focus is on Mandatory Access Control
  - Programmer defined vs. Administratively defined policies debate
- Windows has more privilege-dropping like features (control over tokens)
  - But still nothing to really protect the kernel's attack surface

Options are limited

- On Linux, things such as chroot() to an empty directory remove a small chunk of attack surface
  - cf. Chrome's Linux suid sandbox design
- ptrace() based sandbox
  - Good choice but slow (and not trivial to get right)
- SECCOMP-based sandbox
  - Chrome Linux' future ?

- If we can't protect the kernel let's reduce it's privileges
  - Virtualization is an interesting alternative for seggregation

UDEREF

- Unexpected to userland pointer dereferences are an issue
- We've mentioned Linux/i386 used to have separate logical address space for Kernel/Userland
    - The Kernel's segment descriptors bases were above PAGE_OFFSET
- PaX' UDEREF makes data segments expand-down, limit them above PAGE_OFFSET
- KERNEXEC takes care of the code segment

- What to do on AMD_64 ?
    - No segmentation
    - Full address space switching (Xen does it) ?

mmap_min_addr

- mmap_min_addr is a pragmatic attempt to tackle this problem portably
    - Focusing on NULL pointers dereferences
- system-wide minimum address that can be used at a process

- This has been plagued with many bugs in the past
- In much better shape now
    - We've found one bypass using personalities and suid binaries
    - Another one we need to investigate

Other kernel protection

- From PaX
  - RANDKSTACK
  - KERNEXEC
  - Permission tightening
    - Data in kernel non executable
    - Make some sensitive structures read-only
  - Misc
    - Reference counters overflow
    - Slab object size checks

Conclusion

- There are lots of bugs to find in kernels
    - And the attack surface is growing in general
    - And easier to reach from remote
- Their exploitation difficulty goes from very easy to very challenging
- It's hard to get rid of the kernel's attack surface
    - Remains even in systems designed with security in mind
    - May evolve soon
- Userland exploitation prevention is maturing
    - Kernel exploitation prevention is immature

Bonus Slides

MiCreatePagingFileMap() Vulnerability

- MiCreatePagingFileMap() contained an interesting optimisation in PAE kernels.
- This routine accepts a PLARGE_INTEGER parameter, and is the kernel code responsible for things like CreateFileMapping().
- We noticed that part of the routine realised the parameter was 64bits, and part assumed it was 32bits.
- We could bypass the sanity checks by hiding bits in the upper dword.
- This results in an obvious heap overflow, a minimal testcase would be something like this.

```
CreateFileMappingA(NULL, NULL, PAGE_WRITECOPY, 0x6c, 0, NULL);
```