

<fnord>

Loadable Kernel Integrity Tools

Eric Brandwine

ericb@uu.net

Todd MacDermid

tmacderm@uu.net



What is a kernel module?

<fnord>

- Code dynamically loaded into the operating system
- There is no interruption in OS services
- Code does not run IN the OS, it IS the OS
- New functionality can be provided, or existing functionality can be augmented or replaced

An MCI WorldCom Company

Kernel modules and security

<fnord>

- The Problem: The kernel is implicitly trusted
- Black Hat kernel mods are relatively recent
 - First public reference: Phrack 50-5, April 1997
- Detection of malicious kmods is difficult to impossible
 - System tools use (and trust) system services
 - Evil kmods can lie about their existence

Kernel modules and security (cont.)

<fnord>

- As black hat tools move into the kernel, so must white hat tools
- Few kmod tools exist, and very few in-kernel
 - kstat
 - chkrootkit
 - rkscan
 - carbonite
 - lionfind, etc.

Kernel mode vs. user mode

<fnord>

- No UIDs/PIDs in kernel
 - No permissions checks, everything is super-root
- Kernel has direct access to hardware
- Kernel can bypass memory mapping hardware
- Primary entry point for kernel land is system calls
 - Excellent choke point for monitoring kernel/user interaction
 - strace/truss

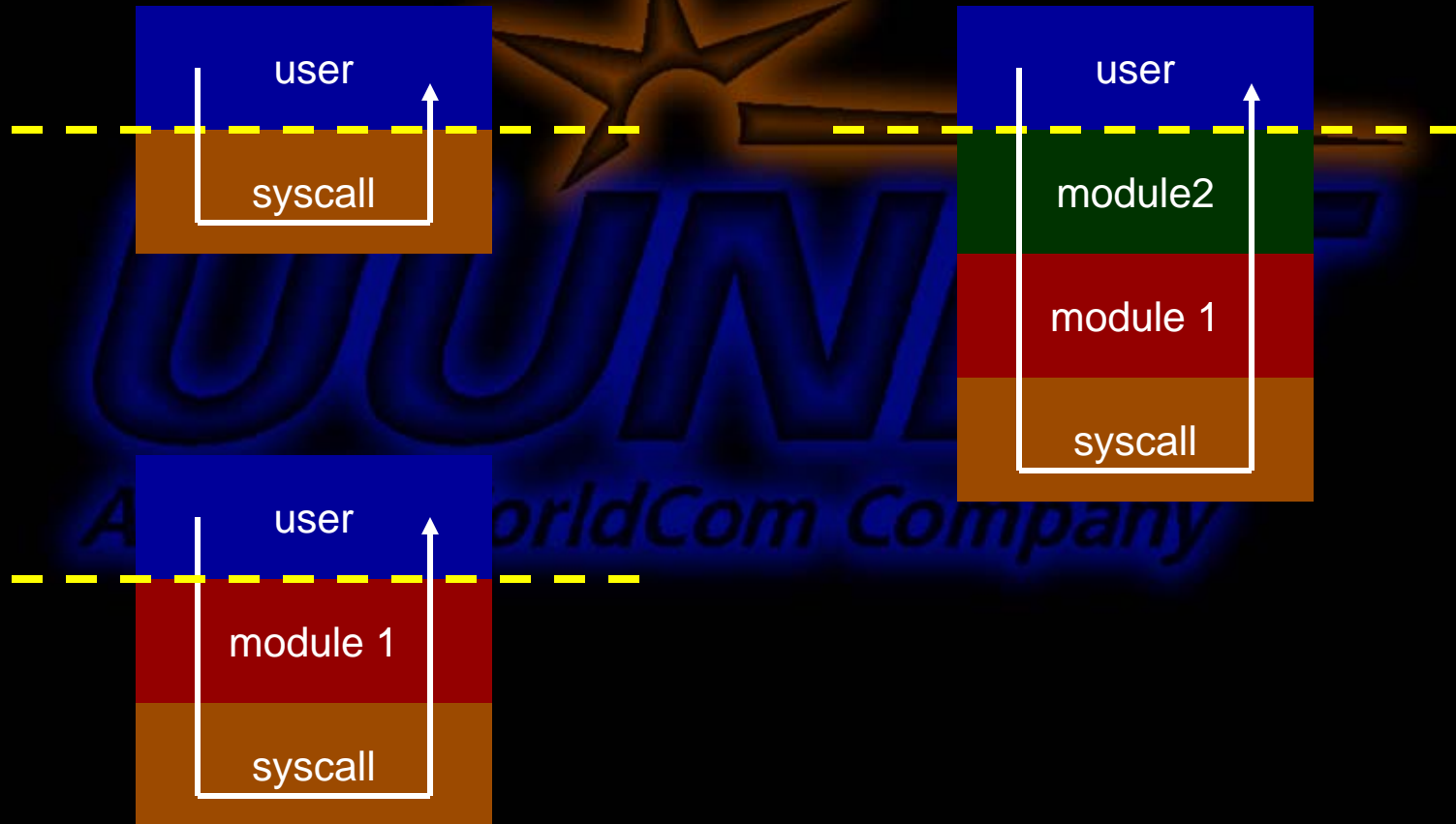
System call table and wrapping

<fnord>

- Addresses of all syscalls are listed in table
 - kernel calls table entries, NOT `sys_*()`
- Table entries can be replaced with address of module code
- Syscalls can be wrapped to alter, rather than replace functionality
 - Can alter arguments before passing on to original syscall
 - Can alter return value before returning to user
- A single system call can be wrapped multiple times
- Little evidence is visible in user space

Syscall wrapping and the stack

<fnord>



Examples of published KRK

<fnord>

- Adore
 - <http://www.team-teso.net>
 - Hides files based on owner uid
- kmod
 - <http://www.itsx.com/kernmod-0.2.tar.gz>
 - Presented at Black Hat 2000 by Job de Haas
 - Solaris based - issues with 64 bit kernels?
- http://packetstorm.securify.com/groups/thc/LKM_HACKING.html
 - Tutorial on Linux LKMs
 - Multiple kmods included

Objectives of fnord

<fnord>

- Kernel integrity checking
 - tripwire for the kernel
 - NOT to guarantee kernel/system integrity (impossible?)
 - Attempt to guarantee notification of integrity breach
- Stealth for itself
 - We assume that attackers are aware of fnord's existence
 - We must hide from active searchers, not just casual inspection
 - No security through obscurity

Objectives of fnord (cont.)

<fnord>

- Flexible, configurable stealth for others
 - Traditional security tools can be hidden (tripwire, etc.)
 - Administration of fnord must be as secure as possible
- Silent alarms and logging
 - Intruders know and circumvent system logging
 - They can't disable logging that they can't see
- In addition for general system integrity, this is ideal for a honeypot

How fnord differs from known kmods

<fnord>

- Designed for widespread deployment
- Designed to have no operational impact
- Separates security administration from systems administration
- Can be administrated and configured while still in stealth mode
- No special tools required to administrate
 - Does not use non-standard `ioctl()`s

fnord feature list

<fnord>

- TTY-like management interface
- Covert logging channel
- fnord alters it's behavior based on environment variables
- Hide files by dev/inode
- Alter selected file contents
- Kernel enforced (silent) append only files
- In-kernel verification of kernel data structures

How to talk with fnord (interactively)

<fnord>

- Motivation: Must be able to get status/configure module while “live”
- fnord can register a pseudo device for interaction with users
- TTY-like interface accepts human readable commands
 - Simple to work with
 - No special tools requires (e.g. cat works)

How to talk with fnord (interactively)

<fnord>

- Can specify files/processes/network connections to hide or show
- Can configure logging
- Can request status



Registering the TTY interface

<fnord>

- Problem: Occupying a char major number would be detectable, due to small number of char majors
- Solution: dynamically register and unregister the interface
 - If a (leet) process `mknod()`s a char special with a name that matches the hardcoded SECDEV string, the interface is registered on the major number of that node
 - If a (leet) process `unlink()`s a char special with the major number of the i/f, and a name matching SECDEV, the i/f is unregistered

How to talk with fnord (non-interactively)

<fnord>

- Problem: Must be able to gather status from module without polling (scalability!)
- Solution: logging channel parallel to standard kernel logging
 - provides `sprintk()` to kernel. Is not exported beyond module
 - provides `/proc/skmsg` to user space
 - similar to `printk()` and `/proc/kmsg`
- Is hidden by fnord

How to talk with fnord (non-interactively)

<fnord>

- Can attach a modified klogd/syslogd to /proc/skmsg
 - Reads private (hidden) config file
 - Is also hidden by fnord
- All changes to module configuration are logged for audit purposes

An MCI WorldCom Company

How to authenticate to fnord

<fnord>

- Special environment variable (LEETVAR) hardcoded into module
- MD5 of special value (LEETSTR) hardcoded into module
 - LEETSTR cannot be recovered from module
- If a process has LEETVAR=LEETSTR in its environment, it is “leet”
 - All hidden files are visible to the leet process
 - File contents are not modified
 - can cause (de)registration of admin interface

Keeping a low profile: filesystem

<fnord>

- Files are hidden based on dev/inode
 - Currently hidden dev/inode pairs are stored in a linked list
 - orig_stat called on each file access attempt
- Benefits of dev/inode hiding
 - All links to a file are hidden
 - No chance of string collision, no spurious files are hidden
 - Files in any location can be hidden, there is no special dir
- Problem: Raw disk access

Keeping a low profile: filesystem (cont.)

<fnord>

- `getdents()` is wrapped to remove entries for hidden files
 - we `orig_stat()` the `dirp` to obtain the `dev` value
 - mount points cannot be hidden
- All syscalls that take `char *path` are wrapped to return `-ENOENT` if the path is hidden
 - What to do for syscalls that take `char *path` for destination (e.g. `link()`)?
- Process hiding is a simple case of file hiding (`/proc/PID`)

Keeping a low profile: file contents

<fnord>

- `open()` is wrapped so that files meeting certain criteria cause the resulting fd to be marked
 - Based on different criteria, fds can be marked for different kinds of alteration
- `close()` is wrapped so that it removes the marked fds from the alteration list
- `lseek()` is wrapped to return appropriate offsets
 - This is a non-trivial problem
- Problem: `fork()` and `dup()` change PID/fd

Keeping a low profile: file contents (cont.)

<fnord>

- `read()` is wrapped to do the actual work
 - As needed, a buffer is read in, internal to the module
 - This buffer is modified (records deleted, added, or changed)
 - user request is satisfied from this buffer
 - `read()`ing a byte at a time will not fool us.
 - Sliding buffer window catches all interesting records
 - User is returned data from this buffer

The sliding `read()` buffer

<fnord>

User `read()`s n bytes, internal buffer filled



Buffer gets modified



User request satisfied from buffer



When buffer is down to 1000 bytes, shift left, and repeat



Keeping a low profile: file content alteration (cont.)

<fnord>

- Files that get altered:
 - /proc/modules
 - /proc/net/udp
 - /proc/net/tcp
 - /proc/device
 - /proc/PID/environ
 - /dev/kmem (fools kstat!)



Keeping a low profile: (misc.)

<fnord>

- /proc/module content alteration is not sufficient, due to `query_module()`
- `query_module(VOID, QM_MODULES, ...)` triggers content hiding as well.

UNINET
An MCI WorldCom Company


fnord & system integrity: logfiles

<fnord>

- Files can be marked append-only in fnord
 - Works regardless of fs type
- `open()` is wrapped such that opening `O_WRONLY` or `O_RDWR` without `O_APPEND` will trigger
 - Silent alarm via `/proc/skmsg`
 - Unaltered version of file saved off (and hidden)
 - `open()` then proceeds normally
- Lulls attackers into a false sense of security

fnord & system integrity: misc

<fnord>

- Traditional tools can be hidden
 - Tripwire
 - Host IDS
 - Honeypot monitoring
 - Snort
 - Root is no longer all-powerful
 - Systems administration and security administration are separated
- 

fnord & kernel integrity

<fnord>

- The real reason we wrote fnord, most of the rest is support for this
- Kernel data structures cannot be modified for an in-kernel viewer
- `sys_call_table[]` entries can be compared to exported kernel symbols
 - Will catch already installed kmod rootkits
- Can be extended to other kernel data structures

Syscall integrity checking

<fnord>

- All syscalls that can be wrapped are. Previous and new addrs are recorded
- Each time a syscall is called, the addr in the table is compared against the recorded addr. A change alarms
- Periodically, on the kernel timer, all syscall addrs are compared. All discrepancies alarm.
- Syscalls that are not easily wrapped:
 - execve
 - sigreturn
 - rt_sigsuspend

Attacks against fnord kernel integrity

<fnord>

- Renaming exported symbol before initial checks are run
- Unwrapping system calls before each check, and rewrapping (requires knowledge of timer)
- Still an arms race. Whoever gets there first wins.

An MCI WorldCom Company

Summary of fnord capabilities

<fnord>

- Can be loaded into an already running kernel in an unknown state
 - Will detect previously installed kmod root kits (if they're still installed), including itself
- Protects systems from future kmod based root kits
- Hides itself virtually without trace
 - Cannot be easily detected through the filesystem layer or standard OS services
 - Cannot be easily detected through /dev/kmem

fnord capabilities (cont.)

<fnord>

- Can hide other devices/processes/network connections, dynamically
- Provides covert logging channel that does not rely on system logging
- Protects log files, silently
- Can be administered without special tools, without disabling stealth
 - Hashed authentication string prevents easy reversal

Future work

<fnord>

- Tcpdump mangling, so that selected traffic is invisible to our machine
- Loose source routing, so all traffic gets routed through a selected box
- Verification of more kernel structures
- Solaris port

An MCI WorldCom Company

Stumbling blocks

<fnord>

- Overflowing the kernel stack
- Processes exiting without `_exit()`
- Re-wrapping syscalls (and overwriting exported symbols)
- Improperly exported `orig_stat`
- Calling syscalls via `int 0x80` within kernel, causing strace oddness
 - `syscall_wrap.o`
- Reading into kernel buffers

Alternative approaches to kernel integrity

<fnord>

- BSD securelevels
 - Control access to kernel and /dev/kmem
- Cryptographic signature checking of modules
- Linux Security Module interface
 - Currently under development
 - Extensible set of hooks
 - <http://mail.wirex.com/mailman/listinfo/linux-security-module>