Dynamic Recreation of Kernel Data Structures for Live Forensics

Andrew Case, **Lodovico Marziale**,
Golden G. Richard III

DFRWS 2010

Memory Forensics

- Acquire a bitwise copy of physical RAM
  - Snapshot of live (volatile) state of machine
    - Running processes
    - Live network connections
    - Open files
  - Jumble of pages
  - Lost once plug is pulled
- Perform analysis for:
  - General forensics
  - Memory resident malware detection
  - Passwords, crypto keys
- Focus for today: Linux

Existing Analysis Techniques

- Strings, grep
  - Baseline
  - Not user friendly
- Crash kernel debugger
  - Tons of info
  - Only specific distros / kernels (Red Hat)
- RAMPARSER
  - From DFRWS 2008
  - More broad range of kernels but still very limited
    - No adaptation to unseen kernels
  - 32-bit x86 only

Kernel Basics

- Written in C and assembly
- Open source
- Constantly distributed updates
- Customized: patches, conditional compilation
  - Different distros (Ubuntu, Red Hat, …)
  - Architectures (x86, x86-64, …)
  - Functionality (Bluetooth, SATA, filesystems, …)
  - Performance requirements

Forensic Targets

- Interested in C structures in kernel
  - *task_struct* (processes - Windows EPROCESS)
  - *inet_sock* (network connections)
  - *file* (file path, owner, open permissions)
  - *dentry* (filename, inode)
  - *vm_area_struct* (mapped memory)
  - *mm_struct* (mapped memory)
  - Many, many others …

Why is Coverage Difficult?

- Many kernel versions
  - Constant updates
  - 2.6.35-rc6 (374 2.6.x.x kernels)
- Many distros
  - Different for Debian / Ubuntu, Red Hat, SuSe …
  - For potentially each above kernels
- Many architectures
  - x86, x86-64
  - PPC, ARM
- Custom user-compiled kernels
- Result:
  - Freely available source = Good
  - Combinatoric explosion = Bad

Effects of Config Options

- *task_struct* (2.6.27.9)
  - 150+ members
  - 50+ conditional members
  - 20+ #ifdef constructs
- Changes structure definition
  - Don’t always have config used
- Changes in-memory layout!

```
struct task_struct {
        volatile long state;        /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;         /* per process flags, defined below */
        unsigned int ptrace;
        int lock_depth;             /* BKL lock depth */
#ifdef CONFIG_SMP
#ifdef __ARCH_WANT_UNLOCKED_CTXSW
        int oncpu;
#endif
#endif
        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
#ifdef CONFIG_PREEMPT_NOTIFIERS/* list of struct preempt_notifier: */
        struct hlist_head preempt_notifiers;
#endif
        unsigned char fpu_counter;
```

Goal

- Build on RAMPARSER
- Wide coverage
  - Kernel versions
  - Distros, custom configs
  - Architectures (x86, x86-64, PPC)
- Minimal requirements
  - Some arch specific knowledge
  - *System.map*, but …
  - Do not require
    - Version
    - Distro
    - Config

Goal

- Provide information on machine state
  - Running processes
    - Name, owner uid and gid, pid
  - Open files
    - Path, name, owner, permissions
  - Live network connections
    - Source and destination ip address and port
  - Current memory mappings
    - Per process libraries, stack, heap, code

Process Listing

NAME	UID	GID	PID
swapper	0	0	0
Init	0	0	1
Kthreadd	0	0	2
migration/0	0	0	3
ksoftirqd/0	0	0	4
evolution-alarm	1000	1000	6010
update-notifier	1000	1000	6016
tracker-applet	1000	1000	6018
nm-applet	1000	1000	6019
python	1000	1000	6020
trackerd	1000	1000	6021

System.map

- Created at kernel compile time
  - For each exported kernel symbol
    - Name, type, address
- Used by klogd for debugging oopses
  - Maps kernel addresses to names
- Used by us
  - Links source code with location of kernel code
- Included in all tested distros /archs

System.map

Address	Type	Name
c041bc90	b	packet_sklist
c041bc94	b	packet_sklist_lock
c041bc94	b	packet_socks_nr
c041bc98	A	__bss_stop
c041bc98	A	_end
c041c000	A	pg0
ffffe400	A	__kernel_vsyscall
ffffe410	A	SYSENTER_RETURN
ffffe420	A	__kernel_sigreturn
ffffe440	A	__kernel_rt_sigreturn

Architecture Specific Info

- Minimal
  - Word size (size of integer, pointer …)
  - Endianness
  - Kernel load address
  - PAGE_OFFSET (for virt -> phys)
  - Methods for member offsets encoding
  - Opcodes used in above
    - Load, store, compare

Approach

- Want to parse structures
  - Regardless of arch, distro, kernel version
- Members accessed by
  - binary (compiled) code
  - Using one of a handful of formats of
    - Structure-base address
    - Plus offset of specific member
- Use known formats for parsing offsets
  - Some assembly composed of load, store, compare

Algorithm

- Less algorithm, more ad-hoc static analysis
  - For functions in *System.map*
  - Which access member of interesting structure
    - Want many short functions
  - Locate function in memory dumps
    - Across distros, kernel versions
  - Determine method for member access
    - Tons of commonality
  - Code RAMAPRSER to snarf member offset

Compilation Across Architectures

Fragment from *sys_remap_file_pages() :*

```
struct mm_struct *mm = current->mm;
```

Ubuntu 2.6.28 PPC64:

e9 2d **01 b0** ld r9,**432**(r13)

Ubuntu 2.6.27-7 x86_64:

4C 8B A0 **40 02 00 00** mov r12,[rax+**0x240**]

Debian 2.6.18-6 x86:

8B A8 **84 00 00 00** mov ebp,[eax+**0x84**]

Compilation Across Kernels

From *insert_vm_struct()* function. This function was used to help find the offset of the *vm_file* member of *struct vm_area_struct*.

if (!vma->**vm_file**)

Ubuntu 2.6.27-11:

```
8b 5a 48        mov     ebx,DWORD PTR [edx+0x48]
85 db           test    ebx,ebx
```

Debian 2.6.18-6:

```
83 7a 48 00     cmp     DWORD PTR [edx+0x48],0x0
```

Putting It Together

System.map contains:

```
c01e8c00    T        insert_vm_struct
```

File mm/mmap.c contains:

```
int insert_vm_struct(struct mm_struct * mm, struct
        vm_area_struct * vma) {
…

        if (!vma->vm_file) {

…
```

Putting It Together

- Analyze how access is compiled (previous slide) using memory address in System.map
- Code to extract offset
- Do this for useful members of interesting structures to build dynamically
- Repeat
- That's it

Results

- Duplicates (some) functionality of
  - ps
  - netstat
  - /proc/<pid>/maps
  - lsof
- Tested on
  - x86, x86-64, and PPC64
  - Kernels 2.6.9 to 2.6.27
- Extensible to new architectures
  - Just need new arch module

Moving Target

- Continuous changes
  - Starting with 2.6.28
    - Minor changes
  - In 2.6.29
    - uid, gid, euid, egid removed from task_struct
    - Entirely new user accounting system
- Can we cope?
  - Yes, but requires continuing effort

What's Next?

- Memory forensics panel later today ☺

Questions?

Vico Marziale
Senior Forensic Investigator
Digital Forensics Solutions
vico@digdeeply.com