Memory Sniffer

Smart Spaces REU 2015
Lehigh University
Seth Bartynski
Advisor: Prof. M. Chuah
Project Goals

- Stage one: create a program able to steal the memory of other programs on a Linux machine
- Stage two: find ways to determine the existence of such a program on any Linux machine
Overview

- Background
- Memory Sniffer
- Concealment
- Detection
- Applications
- Future Work
Background

- **Userspace vs. kernelspace**
  - Kernel deals with physical memory, whereas userspace deals with virtual memory

- **Modules**
  - Dynamically loaded into the kernel
  - Typically device drivers

- **Root**
  - “System administrator” on Unix and Unix-like systems
  - `sudo`: executes command as superuser
Processes / threads
- Linux does not distinguish between processes and threads
- Each task has a corresponding task_struct in the kernel

Pages
- How memory is represented by the kernel
- Size varies depending on architecture
  - 4KB for x86-32
Memory Sniffer: Terms

- **Kernel structures**
  - task_struct
  - mm_struct
  - vma

- **Process ID**

- **Layout of a process in memory**
Memory Sniffer: Terms

Layout of a process in memory:

- **Kernel space**: User code CANNOT read from or write to these addresses, doing so results in a Segmentation Fault.

- **Stack (grows down)**

- **Memory Mapping Segment**: File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

- **Heap**

- **BSS segment**: Uninitialized static variables, filled with zeros. Example: static char *username;

- **Data segment**: Static variables initialized by the programmer. Example: static char "gonzo = "God's own prototype";

- **Text segment (ELF)**: Stores the binary image of the process (e.g., /bin/gonzo)

Memory Sniffer: Details

- Once set up, sniffer periodically scans the computer for running processes
- Evaluates processes to sniff
- Finds the associated task_struct based on the process ID
- Navigates to appropriate memory area
- Scans each area and withdraws any sensitive information
Memory Sniffer: Testing

- **0xdeadbeef:**
  - 2 page block filled with value 0xdeadbeef
  - Looking for 32768 bytes starting near 0x85c0008
    - \[0x85c8001 - 0x85c0002 = 140279809 - 140247042 = 32767 \text{ (+1 for inclusive subtraction)}\]
MySQL:
- Tried heap, data, and text areas: no success
- Other thoughts: not stored as integer value?
- Search for strings instead → no success
- Change input method
- Searched for both “Bob” and 999999999
- Further research into MySQL memory storage is needed
Concealment: Terms

- **sysfs**
  - Is a virtual filesystem created by kernel
  - Contains information about kernel data structures

- **kobjects**
  - Fundamental structure for Linux device model
  - Are embedded within other data structures to give “object oriented” abilities to C

- **lsmod**
  - Lists all dynamically loaded modules from sysfs
  - Note: ignores built-in modules
Delete the kobject for the module
Removes the module from sysfs
No longer shows up in lsmod
Note: rmmod no longer works, since it cannot find the module
Detection: Kernel Tracing

- The Linux kernel has built-in tracing
- Can enable tracing for modules and exclude certain ones
- Memory sniffer will show up in the trace
- Drawbacks:
  - Assumes kernel has debugging enabled
  - Must be setup before the module is initiated

```
seth@seth-VirtualBox:~/Documents/KernelModules/HideAndSeek/hide$ sudo cat /sys/kernel/debug/tracing/trace_pipe | tee tracelog.txt
insmod-2172 [000] .N.. 646.283931: module_load: hide OE
insmod-2172 [000] .... 646.296545: module_put: hide call_site=Sys_f
init_module refcnt=0
```
Detection: uevent Tracing

- uevents notify userspace of kernel-level changes
- kobjects generate uevents with KOBJ_ADD
- Every module has an associated kobject
- Sniffer is never deleted
  - Never sends KOBJ_REMOVE
- Can monitor uevents for modules that have sent KOBJ_ADD
  - Would find sniffer due to lack of presence in lsmod
Drawbacks:
- Again, must be setup before the module is initiated
- Could be system intensive

Countermeasures:
- The sniffer could just send KOBJ_REMOVE

```
seth@seth-VirtualBox:~/Documents/KernelModules/HideAndSeek/hide$ sudo udevadm monitor --kernel --udev --property
monitor will print the received events for:
UDEV - the event which udev sends out after rule processing
KERNEL - the kernel uevent

KERNEL[645.729771] add
ACTION=add
DEVPATH=/module/hide
SEQNUM=1546
SUBSYSTEM=module

UDEV [645.863384] add
ACTION=add
DEVPATH=/module/hide
SEQNUM=1546
SUBSYSTEM=module
USEC_INITIALIZED=5749364
```
The sniffer’s memory is never freed
It is also not marked as used by kernel
It exists in some indeterminate third state
By scanning all the system’s memory, one could theoretically find the sniffer

Drawbacks:
◦ System intensive
◦ Possible race condition

More research necessary before testing
Applications

- Security
  - Prevent against this type of attack
  - Detect and eradicate such an attack
- Virtual machines (the cloud)
  - Supposed to be “secure”
  - Side-channel attacks
  - As of December 2014, 74 percent of operating systems on Amazon Web Services were Linux
Future Work

- Sniff the memory from other programs
  - Server software, bash, other databases
- Use detection results to improve concealment
- Explore more methods for detection of concealed kernel module
- Investigate possibility of side-channel attacks using this module
- Investigate exploiting difference between built-in and dynamically loaded modules
Acknowledgements

- Thanks to:
  - Professor Chuah
  - Heidi Wegrzyn
  - Lehigh University
  - National Science Foundation
Works Cited

- http://blog.intronis.com/bid/a-peek-inside-amazon-web-services
- https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt
- http://www.makelinux.net/ldd3/chp-14-sect-1
- http://stackoverflow.com/a/15494206
- Linux man pages