Analyzing Linux Rootkits with Volatility

Andrew Case / @attrc

Who Am I?

- Digital Forensics Researcher @ Terremark
- Volatility Core Developer & Registry Decoder Co-Developer
- Former Blackhat, DFRWS, BSides, and SOURCE speaker

Linux Support for Volatility

- New in 2.2
- Over 30 plugins
- Supports x86 and x86_64
- Profiles for common kernel versions [4]
  - You can also make your own [5]

Analyzing Average Coder [1]

- Loads as an LKM
- Hides processes, logged in users, and kernel modules
- Operates by overwriting *file_operation* structures in the kernel

file_operations

- One for each active file in the kernel
- Has function pointers *open, close, read, readdir, write,* and so on
- Referenced every time a file is accessed by the kernel
- By hooking a file’s ops structure, a rootkit can control all interactions with the file

Hiding the Kernel Module

- Average Coder hides itself by hooking the *read* member of */proc/modules*
- This is the file used by *lsmod* to list modules
- This effectively hides from *lsmod* and the majority of other userland tools

Hiding Processes

- There is one directory per-process under */proc,* named by the PID
  - e.g. *init* has a directory of /proc/1/
- To hide processes, the *readdir* member of */proc* is hooked
- PIDs to be hidden are filtered out

Communicating with Userland

- Average coder receives commands from the attacker through */proc/buddyinfo*
- Hooks the *write* member which normally is unimplemented

Possible Commands

- hide – hide the LKM
- hpid – hide process
- hdport / hsport – hide network ports
- huser – hide user
- root – elevate process to uid 0

Detecting *f_op* hooks

- The *linux_check_fop* plugin enumerates the */proc* filesystem and all opened files and verifies that each member of every file ops structure is valid
- Valid means the function pointer is either in the kernel or in a known (not hidden) loadable kernel module

 # python vol.py -f avgcoder.mem --profile=LinuxCentOS63x64 linux_check_fop

```
Volatile Systems Volatility Framework 2.2_rc1
Symbol Name              Member          Address
------------------------ --------------- ------------------
proc_mnt: root           readdir         0xffffa05ce0e0
buddyinfo                write           0xffffa05cf0f0
modules                  read            0xffffa05ce8a0
```

Hiding Users

- */var/run/utmp* stores logged in users
- Avg Coder uses *path_lookup* to find the *inode* structure for this file
- It then hooks the *read* member of the *i_fop* structure to filter out hidden users from *w* and *who*

Detecting *utmp* Tampering – Pt 1

- To determine if the file is hooked, we need to find it in memory
- We use the *linux_find_file* plugin with the –F option
- This simulates *path_lookup*

```
# python vol.py -f avgcoder.mem --profile=LinuxCentOS63x64 linux_find_file -F "/var/run/utmp"
Volatile Systems Volatility Framework 2.2_rc1
Inode Number          Inode
----------------      ------------------
      130564    0x88007a85acc0
```

Detecting *utmp* Tampering – Pt 2

- We now know where the inode is in memory
- We can use the -i option to *linux_check_fop* to check a particular inode

 # python vol.py -f avgcoder.mem --profile=LinuxCentOS63x64 linux_check_fop -i 0x88007a85acc0

```
Volatile Systems Volatility Framework 2.2_rc1
Symbol Name                 Member          Address
--------------------------- --------------- -----------------
inode at 88007a85acc0       read            0xffffa05ce4d0
```

Detecting *utmp* Tampering – Pt 3

- We know *utmp* is hooked
- Our live system analysis, whether manual or scripted, will have been lied to
- So we want to recover the real file

Recovering utmp

 # python vol.py -f avgcoder.mem --profile=LinuxCentOS63x64 linux_find_file -i 0x88007a85acc0 -O utmp

 # who utmp

centoslive tty1 2013-08-09 16:26 (:0)

centoslive pts/0 2013-08-09 16:28 (:0.0)

.bash_history

- Stores the commands entered by users on the bash command line
- Invaluable forensics artifact
- Often the focus of anti-forensics:
  - unset HISTFILE
  - export HISTFILE=/dev/null
  - export HISTSIZE=0
  - ssh -T

Bash History in Memory [2]

- All commands in the current session are stored in-memory regardless of the previous anti-forensics tricks used
- The times the commands were executed are also stored in memory regardless if timestamps are enabled!
- Recovering this information would be interesting…

Recovering Bash

 # python vol.py -f avgcoder.mem --profile=LinuxCentOS63x64 linux_bash -H 0x6e0950

```
Volatile Systems Volatility Framework 2.2_rc1
Command Time        Command
-------------------- -------
#1376085128        sudo insmod rootkit.ko
#1376085176        echo "hide" > /proc/buddyinfo
#1376085180        lsmod | grep root
#1376085194        w
#1376085218        echo "huser centoslive" > /proc/buddyinfo
#1376085220        w
#1376085229        sleep 900 &
#1376085241        echo "hpid 2872" > /proc/buddyinfo
#1376085253        ps auwx | grep sleep
```

</Average Coder>

- Detected the rootkit many ways
- The techniques shown are applicable to a number of rootkits

Analyzing KBeast [3]

- Loads as an LKM
- Hides processes, files, directories, and network connections and provides keylogging capabilities
- Gains control by hooking the system call table and */proc/net/tcp*
- Hides itself from modules list

Hiding the Module

- Removes itself from the *modules* list
- Rootkit stays active but is not detected by *lsmod*
- Many other rootkits use this technique

Detection through sysfs

- *sysfs* provides a kernel-to-userland interface similar to */proc*
- */sys/module* contains a directory per kernel module, named by the name of the module

linux_check_modules

- The *linux_check_modules* plugin leverages *sysfs* to detect the hidden module
- Gathers the *modules* list and every directory under */sys/modules* and compares the names
- No known rootkit hides itself from sysfs

 # python vol.py -f kbeast.this --profile=LinuxDebianx86 linux_check_modules

Volatile Systems Volatility Framework 2.2_rc1

Module Name

----

ipsecs_kbeast_v1

System Call Table Hooking

- KBeast hooks a number of system calls in order to hide attacker activity
- *read, write, getdents, kill, open, unlink,* and more…
- These hooks allow the rootkit to alter control flow over a wide range of userland activity

 # python vol.py -f ../this.k.lime --profile=Linuxthisx86 linux_check_syscall > ksyscall

 # head -6 ksyscall

```
Table Name      Index Address    Symbol
---------- ---------- ---------- ------------------------------
32bit            0x0 0xc103ba61 sys_restart_syscall
32bit            0x1 0xc103396b sys_exit
32bit            0x2 0xc100333c ptregs_fork
32bit            0x3 0xe0fb46b9 HOOKED
```

 # grep –c HOOKED ksyscall

10

Hiding Network Connections

- KBeast hooks the *show* member of *tcp4_seq_afinfo*
- This is a sequence operations structure used to populate */proc/net/tcp*
- *netstat* uses this to list connections
- Hidden connections are simply filtered out from reading

Validating Network Ops Structures

- The *linux_check_afinfo* plugin checks the file operations and sequence operations of:
  - tcp6_seq_afinfo
  - tcp4_seq_afinfo
  - udplite6_seq_afinfo
  - udp6_seq_afinfo
  - udplite4_seq_afinfo
  - udp4_seq_afinfo

 # python vol.py -f kbeast.lime --profile=LinuxDebianx86 linux_check_afinfo

```
Volatile Systems Volatility Framework 2.2_rc1
Symbol Name        Member     Address
-----------        ------     ----------
tcp4_seq_afinfo    show       0xe0fb9965
```

</KBeast>

Jynx / LD_PRELOAD

- LD_PRELOAD is an env variable that, when set, loads a shared library into every process
- Any function defined in the pre-loaded library is called before the real function
- Very powerful for debugging purposes and abused by many malware samples

Jynx/Jynx 2

- Popular LD_PRELOAD based malware sample
- Hooks all functions related to reading the filesystem and network
  - open/opendir/stat/fstat/fopen
  - unlink/access
  - accept
- Uses the *accept* hook to implement a network-based backdoor

 # python vol.py -f jynx.mem --profile=LinuxUbuntu1204x64 linux_proc_maps > all_proc_maps

 # grep -c jynx2.so all_proc_maps

364

 # grep jynx2.so all_proc_maps | head -3

```
0x7fb809b61000-0x7fb809b67000 r-x          0  8: 1 655368 /XxJynx/jynx2.so
0x7fb809b67000-0x7fb809d66000 ---      24576  8: 1 655368 /XxJynx/jynx2.so
0x7fb809d66000-0x7fb809d67000 r--      20480  8: 1 655368 /XxJynx/jynx2.so
```

 # python vol.py -f jynx.lime --profile=Linuxthisx86 linux_pstree

```
<snip>
.nc          3047     0
..bash         3048     0
<snip>
```

 # python vol.py -f jynx.lime --profile=Linuxthisx86 linux_netstat -p 3047,3048

Volatile Systems Volatility Framework 2.2_rc1

TCP 0.0.0.0:**12345** 0.0.0.0:0 LISTEN **nc**/3047

TCP 0.0.0.0:**12345** 0.0.0.0:0 LISTEN **bash**/3048

TCP 192.168.181.128:12345 192.168.181.129:42 ESTABLISHED nc/3047

TCP 192.168.181.128:12345 192.168.181.129:42 ESTABLISHED bash/3048

Recovering the Shared Object

- *linux_find_file* can recover the entire shared object
- Can then do binary analysis to determine what functions are hooked, password to the backdoor, etc [6]

Other Plugins

- A number of other Volatility plugins can be used to perform and to aid in malware analysis
- Use in conjunction with each other to get the best results!

Networking Plugins

- linux_ifconfig
  - Lists if interface is in promiscuous mode
- linux_arp
  - Prints the ARP cache (detect lateral movement)
- linux_route_cache
  - Prints the routing cache (external IP addresses communicated with)

Networking Plugins Cont.

- sk_buff_cache
  - Recover packets from the kmem_cache
- pkt_queues
  - Recover queued packets on open/active sockets

File Access & Mappings

- linux_dentry_cache
  - Recover the full path and metadata of accessed files
- linux_vma_cache
  - Recovering files mapped into processes (shared libraries, *mmap*'d data files, etc)

Processes

- linux_psaux
  – Recover command line arguments
- linux_pslist_cache
  – Recovers processes from the kmem_cache (including exited ones)
- linux_pidhashtable
  – Recovers processes from the *pid* hash table
- linux_psxview
  – Lists all processes and if they are found in process list, cache, and/or hash table

Conclusion

- Volatility's Linux support provides powerful rootkit & IR analysis
- We did not even cover all the plugins…
- Exciting features to come soon related to Android processing!

The End

- Volatility:
  - http://volatility-labs.blogspot.com/
  - http://code.google.com/p/volatility/
  - @volatility
- Me
  - http://www.memoryanalysis.net
  - @attrc

References

[1] http://average-coder.blogspot.com/2011/12/linux-rootkit.html

[2] http://volatility-labs.blogspot.com/2012/09/movp-14-average-coder-rootkit-bash.html

[3] http://volatility-labs.blogspot.com/2012/09/movp-15-kbeast-rootkit-detecting-hidden.html

[4] http://code.google.com/p/volatility/wiki/LinuxProfiles

[5] http://code.google.com/p/volatility/wiki/LinuxMemoryForensics

[6] http://volatility-labs.blogspot.com/2012/09/movp-24-analyzing-jynx-rootkit-and.html