Rootkits

CS 161/194-1
Anthony D. Joseph
December 2, 2005

Administrivia

• Final exam:
  – 1 LeConte Hall
  – Tuesday 12/13 12:30-3:30
  – Comprehensive
  – Open books, notes, …
  – No electronic devices
• No office hours for me next Mon/Tue
  – Substitute hours: Th 12-1, Fr 10-11
• Project 2 is on web page
Outline

• How to tell you’ve been owned?
• What is a rootkit?
• History of rootkits
• User-mode rootkits
• Kernel module/hooking rootkits

You’ve Been Owned!

• How can you tell when your machine has been compromised or taken over?
• “Odd” processes
• “Odd” windows
• “Extra” files
• Changed registry/configuration files
• “Extra” network connections, open ports
• …
What Is a Rootkit?

- Software or techniques that attempts to hide cracker’s software from detection
  - Cracker’s software can be anything
- Simple methods
  - Delete entries from login records, shell history
    - Then, **last** command won’t show intruder
- Cloaking methods (aka Ghostware)
  - Hide executables, libraries, config files, processes, …
  - Hide from **ls**, **dir**, **ps**, **taskmgr**, …

Rootkit Functions

1. Maintain access
2. Attack local or other systems
3. Destroy evidence

- **Which OS’es are vulnerable?**
Rootkit Function: Maintain Access

- Backdoor: telnet, rsh, ssh, irc, custom
  - UDP/TCP/ICMP protocol running on “high” port
  - Could require activation by “magic” TCP/IP packet, be a stealthy network sniffer, or use a covert channel, …

- Outbound connection
  - Works behind firewalls, no open inbound port to detect
  - Can be tunneled over outbound port 80

Rootkit Function: Attack Local or Other Systems

- Collect local information
- Install network sniffer
- Perform DDoS attack
- Attempt to propagate
- …
Rootkit Function: Destroy Evidence

- Execute a log cleaner
- Hide its files
- Hide its processes
- Hide its network connections
- …

How Rootkits Get On Your Machine

- Cracker scans for vulnerable hosts
  - Or uses privilege elevation exploit
  - Or uses a worm or virus payload
- Exploits vulnerability to gain shell access
- Then copies over and installs rootkit …
  - Hides existence, records
  - Modifies start files
  - Starts daemon
Some Rootkit History Highlights

• 1989: First log cleaners found on hacked systems
• 1994: Early SunOS kits detected
• 1996: First Linux rootkits released
• 1997: Linux Kernel Module Trojans proposed
• 1998
  – Non-LKM kernel patching proposed
  – “Cult of the Dead Cow” created Windows rootkit “Back Orifice”
• 1999
  – Adore LKM kit released by TESO
  – “Cult of the Dead Cow” releases BO2K
• 2000: T0rn rootkit released
• 2002: Sniffer backdoors start to show up in kits

Pre-Rootkits: Hiding Login Events

• Many systems display a user’s last login time when they login
• Early crackers covered their tracks by using tools to modify login and other db records
  – Modify or delete wtmp file
  – Kill syslogd, and modify or delete syslog.conf
• How to defend systems?
  – Use a remote syslogd
  – But, some tools report remote entries in syslog.conf
Binary Library Rootkits: T0rn v8

- User-mode rootkit
- Easy to use (precompiled binaries)
  - Just type ./t0rn.
  - Includes a log cleaner called t0rnsb
  - Also a network sniffer named t0rns and a log parser called t0rnp
- Replaces the tools that would show the rootkit:
  - /usr/bin/du, /usr/bin/find, /sbin/ifconfig, /usr/sbin/in.fingerd, /bin/login, /bin/ls, /bin/netstat, /bin/ls, /usr/bin/sz, /usr/bin/tcp
- Replaces system dynamic libraries to hide rootkit

Detecting T0rn v8

- Several serious implementation errors:
  - Different output from ps -eb than real one
  - Running netstat causes seg fault
- Wrong file sizes versus real files
- Easy to detect with lsof (list open files/ports)
  - Shows daemon listening on t0rn’s default port
  - Shows all processes running under t0rn daemon (since it has open files)
- Can also be remotely detected
  - Use nmap to detect open ports
  - This is a common detection mechanism for non-stealthy rootkits
- Libraries only work for dynamically linked programs
Kernel Module-based Rootkits

- Target Linux, Free/OpenBSD and Solaris
- Hook into the system kernel and replace/remap or modify/intercept various system calls
  - Ones used by file system tools, and core kernel components
- Operating system core is no longer trustworthy
- Config file or built-in filename regexps lists files to hide:
  - Its own files, process, and sub-processes
  - Any of its inbound/outbound network connections (by address, protocol, listening process)

Detecting Kernel Module Rootkits

- Challenge is detection “from within the box”
  - Rootkit controls the vertical and the horizontal
- Leverage implementation errors
- Look for inconsistencies between different views
  - Can use cryptographic hashes of all important files (but have to protect hash values…)
  - Use tcsh’s built-in ls: ls-F
  - Compare results from lower level interface
- Ideal solution:
  - Compare against known good system or CDROM
    - Boot from CDROM/remote system and then examine disk
User-Mode Windows Rootkit: Back Orifice

- Windows is also vulnerable to user and kernel rootkits…
- Back Orifice (Win98 and WinNT systems)
  - Hid by running as a “system service”
  - Modified a registry startup entry
  - Listened for remote commands
  - Wasn’t very stable under WinNT
- Didn’t really try to hide itself
  - Was visible to process list tools

Kernel Module
Windows Rootkit: BO2K

- Similar behavior as Unix kernel rootkits
  - Targeted W2K systems
- Installed itself into kernel memory
- Hooked kernel functions with its own modified functions
  - Blocked filesystem, process table and other attempts to find BO2K
Detecting Windows Kernel Rootkits

- Examine startup registry entries
  - Works for many rootkits
- In the box checks
  - Compare Win32 API results with results from low level kernel calls (e.g., process list, master file table,…)
  - Compare cryptographic hashes against known correct values
  - Look for hiding actions (create file/dir with prefixes)
- Out of the box checks
  - Compare against known good media/system

Rooting a Windows Kernel Rootkit

- Microsoft Research Tricks for using rootkit against itself
- Same name attack
  - Copy cmd.exe to same name/prefix as rootkit
  - Launch with start command
  - Rootkit can’t hook itself, so built-in commands can run and see rootkit files, processes, directories, …
- Tools same name attack
  - Pick tool of choice for removing rootkit
  - Use same name attack, as rootkit won’t block itself
Kernel Hooking Abuses

- Many anti-virus, firewall, anti-spyware and other tools use kernel hooking tricks
  - Can affect system stability when multiple programs are hooking kernel
  - MS Vista will block unsigned program hooking
- Sony XCP used kernel hooking to hide itself
- Problem is that crackers may be able to exploit cloaking to hide their tools!

The Future of Rootkits

- On going arms race between crackers and detection tools…
- Out of the box detection will always be possible
- In the box detection will increase in difficulty