

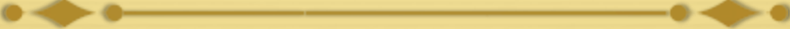


Kernel Rootkits

◆ — ◆

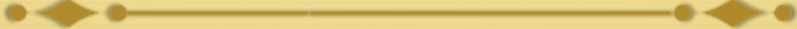
Don Porter

Motivation (bad guys)



- ✦ Why take over a computer system?
 - ✦ To get it to do (potentially illicit) work for you for free!
 - ✦ E.g., send spam

Stages of an attack



- ✦ Get on the system
- ✦ Get administrator privilege
- ✦ Install your software, and possibly a way to get back in later

How does one get in?



- ✦ Common attack vectors:
 - ✦ Take over an account
 - ✦ Weak passwords
 - ✦ Colluding, legitimate user
 - ✦ Exploit a bug in a network service
 - ✦ E.g., buffer overflow, shell code injection
 - ✦ These can be trickier to pull off

How does one get privilege?

- ✦ For free
 - ✦ Attack a network service with root privilege (ssh)
 - ✦ Or take over an account with 'sudo' permission
- ✦ Or, find an exploitable bug in a privileged service on the system
 - ✦ Setuid binaries and system daemons common targets
 - ✦ Input parsing bugs, time-of-check-to-time-of-use (TOCTTOU) race conditions

How to come back later?



- ✦ These attacks are elaborate (a lot of hassle)
 - ✦ And vulnerabilities could be patched by an upgrade
- ✦ Ideas?
 - ✦ Install an ssh or telnet daemon that uses different credentials, listens on an unusual port, etc.
 - ✦ A.k.a. a “back door” into the system
 - ✦ No fuss, no hassle to come back later

Problem?



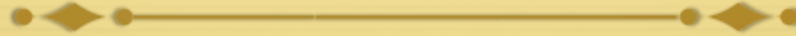
- ✦ What if the administrator discovers your malware, or your back door?
- ✦ This is where rootkits generally come in---they hide the malware from the administrator

High-level Goal



- ✦ Hide all traces of malware
 - ✦ 'ls' doesn't show the binary files
 - ✦ 'ps' doesn't show running processes
 - ✦ 'lsmmod' doesn't show the rootkit
- ✦ Hard to leave no trace
 - ✦ E.g., system is slow, but 'top' says completely idle
- ✦ Also, need to be persistent across reboots
 - ✦ Usually a (hidden) init script to reload the rootkit

Strategies



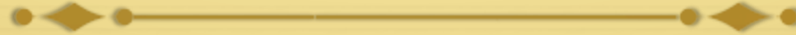
✦ Suggestions?

Common strategies



- ✦ Replace entries in the system call table
 - ✦ Filter return values
- ✦ Replace function pointers on file inodes
 - ✦ E.g., filter readdir output
- ✦ Replace libc in user level
- ✦ Actually overwrite the first instruction of a kernel function with a jump to other code

How to mitigate this?



Countermeasures



- ✦ Generally enforced in a hypervisor/VMM
- ✦ Mark kernel code pages as read only
- ✦ Look for inconsistencies in function pointers, etc.

Another way to think about the problem

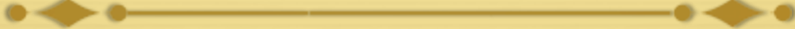
- ✦ This is really an issue of code *integrity*
- ✦ In other words, by changing key data structures or code modules, the attacker violates an assumed invariant of the rest of the code
- ✦ Most countermeasures attempt to prevent or detect broken invariants

Example: File integrity checks



- ✦ Have a database of file checksums on read-only media or another system
- ✦ Periodically check the file system for checksum changes
 - ✦ When the system is powered down, if necessary

Example 2: OSck



- ✦ The hypervisor creates a “sibling” VM that has a read-only view of kernel data
- ✦ Developer specifies a bunch of data structure invariants
 - ✦ All tasks should be in the scheduler queue and in /proc
 - ✦ All inodes on an ext4 FS should point to an ext4 operations struct
- ✦ Sibling VM periodically walks all kernel data structures, checking for inconsistencies

Summary



- ✦ Rootkits hide the presence of other malware
 - ✦ Lab 4: You will build one to hide some “fake” malware
- ✦ Ultimately work by undermining an assumption in the running code (integrity)
- ✦ Most countermeasures focus on detecting inconsistencies or changes in the system