Rootkit: Analysis, Detection and Protection

Igor Neri

Sicurezza Informatica – Prof. Bistarelli
Definition of Rootkit

A rootkit is malware which consists of a set of programs designed to hide or obscure the fact that a system has been compromised.
What does a Rootkit do?

- Hides Attacker Activities
What does a Rootkit do?

- Hides Attacker Activities
- Provides unauthorized access
What does a Rootkit do?

- Hides Attacker Activities
- Provides unauthorized access
- Cleans Logs
Classification

User Space  Kernel Space
Classification

- Ring 0 - full access to all memory and the entire instruction set
- Ring 3 - restricted memory access and instruction set availability
User Space

- Replace specific system program used to extract information from the system
- Can include additional tools like sniffers and password crackers
User Space: Hiding

- File Hiding: du, find, sync, ls, df, lsof, netstat
- Processes Hiding: killall, pidof, ps, top, lsof
- Connections Hiding: netstat, tcpd, lsof, route, arp
- Logs Hiding: syslogd, tcpd
- Logins Hiding: w, who, last
User Space: Grant Access

- Backdoors: inetd, login, rlogin, rshd, telnetd, sshd, su, chfn, passwd, chsh, sudo

- SNIFFING & data acquisitions: ifconfig (hide the PROMISC flag), passwd
User Space: Clean

- addlen: tool to fit the *trojaned* file size to the original one
- fix: changes the creation date and checksum of any program
- wted: has edit capabilities of wtmp and utmp log files
- zap: zeroes out log files entries
- zap2 (z2): erases log files entries: utmp, wtmp, lastlog
User Space: summary

- Easy to write/install
- Too many binaries to replace thus prone to mistakes
- Verifications through checksums is easy and OS dependent
- Old type
Kernel Space

- The goal of a kernel rootkit is placing the malicious code inside the kernel by manipulating the kernel source / structure
- No need to substitute binaries, kernel modification affects all binaries system call
- Complex to write
- Complex to identify
How is the flow of execution intercepted?

- The flow of execution needs to be intercepted or modified at some point
- The manipulation can take place at many different levels

Example: `ls` command

<table>
<thead>
<tr>
<th>Process</th>
<th><code>ls</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System library</td>
<td><code>getdents()</code></td>
</tr>
<tr>
<td>User mode</td>
<td>System call interface</td>
</tr>
<tr>
<td>Kernel mode</td>
<td><code>sys_getdents()</code> (<em>manipulated</em>)</td>
</tr>
<tr>
<td>Kernel functions</td>
<td></td>
</tr>
</tbody>
</table>

16/03/2009  Igor Neri - Sicurezza Informatica  14/34
Normal Execution Flow

Executing a syscall in the kernel:

- Interrupt handler consults the IDT
- System call handler consults Syscall Table
- Function implementing the system call execute other kernel functions
Manipulating the Syscall Table

- The rootkit is called instead of original function
- Rootkit acts as a wrapper
- Method used by first kernel rootkits
- Example: Adore
Copying the syscall table/handler

- Original syscall table is not modified
- Modified syscall handler uses manipulated copy
- Example: SucKIT

**Rootkit**
- sys_getdents()
  - access virtual filesystem
  - access actual filesystem
  - ...
Manipulating the IDT

- A different syscall handler is used, which calls rootkit
- No need to modify syscall handler or syscall table

Rootkit

- sys_getdents()
  - access virtual filesystem
  - access actual filesystem...

choose interrupt handler

choose system call

Kernel mode

Interrupt Descriptor Table

Syscall Table
Manipulation deeper inside the kernel

- Less central kernel structures are manipulated
- Hard to detect since many kernel structures need to be monitored
Kernel rootkit example
Target Program: *netstat*

*netstat* provide information about network connection

```
root@localhost# netstat -an
```

```
tcp 0 0 0.0.0.0:8080 0.0.0.0:* LISTEN
```
```
tcp 0 0 127.0.0.1:1025 0.0.0.0:* LISTEN
```
```
tcp 0 0 0.0.0.0:6000 0.0.0.0:* LISTEN
```
```
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
```

We want to hide the service on **8080**
How *netstat* works

root@localhost# strace netstat -an

[cut]

open("/proc/net/tcp", O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40191000
read(3, " sl local_address rem_address ...", 4096) = 900
write(1, "tcp  0  0 0.0.0.0:8080"", 81tcp  0 0 0.0.0.0:8080
  0.0.0.0:* LISTEN) = 81
write(1, "tcp  0  0 127.0.0.1:10"", 81 [cut]
close(3)
Altering open and read syscall

Hijacking on init module phase:

old_open=sys_call_table[__NR_open];
sys_call_table[__NR_open]=new_open;
old_read=sys_call_table[__NR_read];
sys_call_table[__NR_read]=new_read;

Check on file opening:

if (strstr (filename,"/proc/net/tcp")) ACTIVA = 1;
r=old_open(filename,flags,mode);

Variable ACTIVA useful on read syscall
Altering open and read syscall

Check on file reading, if process netstat and file /proc/net/tcp

```c
r=old_read(fd,buf,count);
if(r<0)return r;
if ((strcmp(current->comm,"netstat")!=0) || (ACTIVA==0))
return r;
```

Then we'll search for occurrence to hide and we'll remove that from r
Load kernel module & try

Load module

```
root@localhost# insmod hide_netstat.ko
```

re-run `netstat`

```
root@localhost# netstat -an
[cut]
tcp 0 0 127.0.0.1:1025 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:6000 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
[cut]
```
Detection

- Checksums of important files (aide, tripwire, …)
- Rootkit detector programs using signatures (chkrootkit, rootkit hunter, …)
- Backups of central kernel structures (kstat)
- Runtime measurement of system calls (patchfinder)
- Anti-rootkit kernel modules (St Michael)
- Offline / forensic analysis (TCT, …)
- Watching the network traffic-flows from 3rd system
- Manual logfile analysis and search
DEMO

- Login on remote host via SSH using Debian OpenSSL vulnerability (DSA-1571)
- Installation of homemade rootkit and Adore-NG rootkit with example of use
- Detection via system analysis and detection tools: chkrootkit e rkhunter+skdet
DEMO: What's SSH

- SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices.

- Key Based Authentication:
  - First, a pair of cryptographic keys is generated.
  - One is the private key; the other is the public key. The public key is installed on the remote machine and is used by ssh to authenticate users which use private key.
Luciano Bello discovered that the random number generator in Debian's openssl package is predictable. This is caused by an incorrect Debian-specific change to the openssl package (CVE-2008-0166). As a result, cryptographic key material may be guessable.
Protecting the system

- Applying runtime detection methods
- OS / Kernel Hardening
- Patching the vulnerabilities
- Restricted operations and capabilities
- LKM Protection
Famous case:
Ken Thompson vs. Naval Lab.

```c
compile(s)
char *s;
{
    if(match(s,"pattern1")){
        compile("bug1");
        return;
    }
    if(match(s,"pattern2")){
        compile("bug2");
        return;
    }
    ... 
}
```

Reflections on Trusting Trust Ken Thompson
Famous Case: Sony BMG CD copy protection

- The copy protection scandal concerns the copy protection measures included by Sony BMG on compact discs in 2005.
- This software was automatically installed on Windows desktop computers when customers tried to play the CDs.
References

- “SHADOW WALKER” Raising The Bar For Rootkit Detection
- UNIX and Linux based Kernel Rootkits (DIMVA 2004 - Andreas Bunten)
- Rootkits: Subverting the Windows Kernel
- Countering Trusting Trust through Diverse Double-Compiling (DDC), David A. Wheeler
- Reflections on Trusting Trust Ken Thompson
- Analysis of Rootkits: Attack Approaches and Detection Mechanisms - Alkesh Shah
- http://packetstormsecurity.org/UNIX/penetration/rootkits/
- Come costruire un mini-rootkit I - Nascondiamoci da Netstat - blAAd!