UNIX Security: Threats and Solutions

Matt Bishop
Dept. of Computer Science
University of California at Davis
Davis, CA 95616-8562
phone: (916) 752-8060
email: bishop@cs.ucdavis.edu

Goal of Talk

- Describe goals of attackers
- Describe relevant UNIX history, features
- Outline of types of flaws, and examples
  - These include current serious problems as well as oldies
  - Present solutions and/or suggestions for strengthening security
- Conclusion

To Those Who Advocate No Disclosure

Three can keep a secret,
if two of them are dead.

— B. Franklin

Overview of UNIX System

- Designed by programmers for programmers
- Open design (no security through obscurity)
- Lots of free, easily available software
- Lots of development done at universities

Privileges in UNIX

- “superuser” or root user, to which ***no*** access control rules apply
- Each user has own protection domain
- Each user in 1 or more groups which also have own protection domain

Attackers

- “bad guys and gals”
- End goal: get something, do damage
- How? get (user or superuser) privileges
- So … first, need access to system

Example Attack

```
% telnet victim 25
220 victim sendmail ready Jan 24 05:06:07 1995
helo nsa.gov
250 victim Hello nsa.gov (kgb.su), Pleased to meet you
wiz
220 Enter, oh mighty wizard
shell
%
```

What Went Wrong

- technical
  - inadequate validation of peer
  - incorrect access setting (allowing anyone in)
  - sendmail program far too complex
- organizational
  - developer not given adequate access to needed resources
  - distributors did not disable “back door”
- human
  - configuration errors (no password, wrong privileges)

Threats to UNIX Systems

- human
  - errors in administration
  - errors in configuration
  - lack of knowledge
- organizational
  - unrealistic expectations of workers
  - lack of resources
  - unwillingness to provide resources

Technical Threats

- not desgned with security as primary goal
- new features/programs being added
  - usually not written with security in mind
  - nonsecure interactions with other programs
- lack of knowledge of older holes

Example #1

Program “login” logs you in; it runs as superuser

Dynamic loading lets programs load routines from libraries as needed

First: allow users to say where the libraries are (using special variable LD_PATH)

Attack: login uses a routine called “getpwnam”; write a routine by this name giving you a shell, and run login

Example #2

Fix: Ignore LD_PATH if program runs as superuser

Attack: User "sync" executes program "sync" on login. This program calls routine "sync"; write a routine by this name giving you a shell. No password needed, so log in as sync.

Result: login does not use bogus "getpwnam" as is run as superuser; runs sync as user sync with privileges of a system group, and you get a shell with those privileges

Moral Of The Story

- Don’t forget the past; first problem was seen in another form, much earlier
- Interaction (between privileged login program and unprivileged sync program) caused the problem

Classes of Security Flaws

- Flaw taxonomies developed in 1970s by Program Analysis and RISOS projects, as well as by NSA
- Resulting classes are isomorphic
- What follows is based on PA work

Improper Choice of Initial Protection Domain

ftp files owned by user ftp

have anonymous ftp enabled

Why: anonymous ftp user can delete, create files and give itself privileges by altering files

Network

Network may be in protection domain — unprotected

- no privacy
- no authenticity

  Can read data (such as passwords) off the telnet/rlogin/ftp sessions

Solutions

- Have a well-defined security policy
- Know your system!
  - how it should be configured to enforce security policy
  - what procedures should be in place, etc.
- Use tools
  - COPS does a general check
  - tripwire checks file integrity

Improper Isolation of Implementation Detail

NIS: password and user info stored on a central server, clients invoke client program ypchfn to change user info

Attack: place appropriate characters into user info and you could create a bogus account with no password and arbitrary privileges

Fix: have client check for those characters

Problem: write your own client

Solutions

- Look for multiple ways to name one object
  - example: main memory and /dev/mem
- Figure out how a resource (database) should be accessed at abstract level
  - security manual
  - contrast with way programs do access it
- No easy solutions for this

Improper Change

time of check to time of use flaws

superuser program xterm checks that user owns file, then writes to it; trick is to have it check one file, then alter what name refers to before the open for writing

How hard: very easy!

Solutions

- Slicing
  - Determine which pairs of system calls create problem
  - Analyze program for those pairs
- Less formal: look for the pairs of suspect functions
  - access / open
  - access / chroot
  - creat / chown
  - open / rename

Improper Naming

- Look for Trojans with name of system commands
  - A very simple program to write; COPS does this
- Look for 2 names for 1 object
  - Different privileges for each name
- Look for processes which communicate with another process

Solutions

- Get COPS
- Confine search path to directories not generally writeable and whose owners you trust
- Look for one file with multiple links

Improper Deallocation or Deletion

Input data put into kernel memory (called “cbuf”s)

If this can be read, attacker can get confidential data (like passwords)

cbufs are reused, but not cleared, so the data read in lingers

Solutions

- In programs, be sure any memory allocated is cleared before deallocation
- Clear files before deleting them

Improper Validation

- WWW problem: expects data stream to be no more than 2048 characters long
- So send more, and make excess be a program to spawn a shell
- Presto! You can access the system
- Reincarnation of a1988 bug used by Internet worm

On the Network

- IP spoof attack Mitnick used to get to Tsutomu’s machine
- assumed the IP address in the messages was valid
- allowed access based on the trust Tsutomu had in the host with that address

No good way around this, as strong authentication does not exist on the Internet for general use

Solutions

- Things to look for
  - Use enumerated types instead of integers or macros
  - Check all arguments for illegal or unexpected values
  - Do not make assumptions you can’t verify; if in doubt, stop
  - Check all return values
  - Use error-checking library functions
- Know your environment
  - If data is inherently untrustworthy, verify it some other way or don’t repose trust in it

Improper Indivisibility

To create a directory, make it (requires superuser privileges) and then change its ownership

- Operation ***not*** atomic

Create directory named x

- Attacker deletes it and creates a link named x to some protected file

Change ownership of x

Solutions

- Kernel mods so the operations are indeed atomic
- Analyze program steps to see what should be atomic
- Bernstein conditions (at most one writer active, and no reading while writing going on) need to be enforced

Improper Sequencing

- Two people changing the password file simultaneously
- The make directory bug above
- The xterm bug

Solutions

- Same as for improper indivisibility
- Also check for violaions of Bernstein conditions

Improper Choice of Operand or Operation

- Allowing a mail message to be used as input to a standard shell
  - Sender can get the shell to act on his/her behalf
  - This may allow creation of privileged programs which will then give access to attackers (uudecode bug)

Solution

- Check operations carefully against policy
- Check arguments, results of function calls and program outputs

Ten Biggest Threats

- Improper configuration
  - Probably 90-95% of breakins occur because of this
- Improper placement of trust
  - Most network breaking involve this
- Improper validation
  - Make bogus assumptions, like basing security on IP address
- Improper change
  - With UNIX, it’s real easy to do this one
- Using the network to send confidential material
  - Read this as: passwords

More Threats

- Improperly trained system administrators
  - “please change my password over the phone”
- Apathetic or hostile users
  - These are the first line ofdefence
- Too little resources
  - Can stretch things only so far
- Availability of security testing programs
  - I don’t consider this a bug; others to
- Existence of superuser
  - Can be eliminated, but requires kernel rewrite

Conclusion #1

Zymurgy’s First Law of Evolving System Dynamics:

When you open a can of worms,
the only way to recan them is to use a larger can

Conclusion #2

- UNIX is fun
- It has security problems
- You can minimize their threat
  - you must be alert to attacks
  - have a plan to deal with attacks