

The logo for Nixu, featuring the word "nixu" in a white, lowercase, sans-serif font. The letters are spaced out, with "n" and "i" being smaller than "x" and "u". A thin white vertical line is positioned to the right of the text.

nixu

# Hardware and Operating System Security

# Contents

- Hardware security
- Unix operating system security
- Attacking Unix
- Installing Unix

# Hardware possession and control

- Generally computing takes place in some kind of computing hardware
- In traditional general purpose computing the authorized user is given access based on passwords or some other kind of user identification
- However anybody who has access to the physical computing device may perform different kinds of actions on it
  - Steal the computer or parts of it
  - Reboot the computer from the user's own media and bypass the operating system security
  - Break physical security measures to access various parts of the device

# Securing workstation computers

- Generally the computer must be protected from theft, vandalism, opening, booting with another media
- Sometimes an authorized user must be prevented from using transportable media devices
  - Disks, writable CD-ROMs
  - USB-, PCMCIA-, parallel port hard drives
- Remember that the operating system access permissions do not prevent accessing files on disks, if that computer is booted with the attacker's own media or if the disks are removed and installed to another computer
- Solutions are in the family of physical security and often include computer case locks, chains, epoxy and video monitoring

## Emission security

- All electrical devices that cause variations in the flow of electricity create some kind of electromagnetic radiation
- General purpose computers are really radio transmitters with several transmission frequencies
  - CPU and bus clock
  - Peripheral transmission rates
  - Especially the video display (of CRT type)
- The basic transmissions are modulated by the data being processed
- The connections to the peripherals are often also antennas
  - Mouse and keyboard cords

# Controlling the computer emissions

- Tempest is a non-public US military standard that is often used to refer to computing devices with emission control
  - Roughly can be said to require enclosing the whole computer in a Faraday cage
  - Controlling the display emissions is a special problem
- Mostly used by the military
- Emission control is currently considered too expensive for most civilian applications
  - Always a subject to change in the future

# Tamper resistant hardware

- Problem: hardware is given to end users, but the contents should remain in the control of the owner or originator of the hardware
  - Telephone SIM cards
  - Smart cards (used for access, TV decoders, ID, money...)
  - Cryptographic password tokens (eg. SecurID)
  - Car computers
  - Public ATM machines
- One solution is tamper detection with e.g. seals
  - Especially if the problem domain allows rollback, meaning that the effects of the tampering can be reversed

## Smartcards

- A plastic card with a CPU and non-volatile memory
- Can store information and perform low-end processing
- Draws power from the host terminal, often also a clock pulse
- Communicates with the host terminal

# Smartcard security

- The data on a smartcard (often a crypto key) can be extracted using several methods
  - Gaining access to the circuitry and by reading the data as it is transferred from the memory to the CPU
    - Requires shaving the protective layers of the card or careful drilling
  - Monitoring the power consumption of the card and deducing the key as it is used by clever mathematics (e.g. Chinese Remainder Theorem)
  - Interrupting the operation of the smart card by tampering with the operating voltages or the clock signal
- With money cards it is often enough to prevent change to the memory
  - The attacker can try to filter out the EEPROM write voltage
- Smartcards are getting better all the time, but they are not invulnerable

# Protecting data from hardware failures

- All mass media storage devices with moving parts are sure to fail
  - Backups are used to store the data in a secure location
  - Redundant Arrays of Inexpensive Disks (RAID) distribute the data to several physical disks using an error correcting code
    - Failure of a single disk should not cause any data loss
    - Beware of manufacturers with good quality control, multiple failures over a weekend are not unknown
- Power spikes or blackouts are no good for data
- All kinds of devices fail, e.g.. tape drive's writing heads
  - Experienced system administrators also read the tapes besides just writing to them

# Backup computing

- Part of continuity planning
- Backup media storage location is important
  - A physical location that is unaffected by anything that might destroy the main location is preferred
    - Physical security requirements are thus doubled
    - Bank vaults are usually a good choice for backup tapes
  - Magnetic media is sensitive to temperature
    - Ordinary safes are not enough to protect magnetic tapes from fire
- Entire backup computer systems are sometimes used
  - Expensive
  - Enable the business to continue almost immediately
  - Hot or cold backup

# Secure Computing Architectures

- Multilevel secure systems can limit information flow between levels
  - The levels might be different operating system levels or
  - Different users or user groups
- Generally the CPU should provide protection to these levels
  - Some CPUs have complex multilevel structures
- However the protection achieved depends also on the operating system
  - Most operating systems do not provide very good security or do not use the features of the CPU
  - Some operating systems support security levels by supporting for example virtual machines for the users

# Unix security paradigm

- In the Unix operating system there are two parts
  - Kernel
  - User space
- Any programming code in the kernel space has full access to the computer it is running on
- Code running in the user space has access rights based on the User ID (UID) it is running under
  - UID 0 is reserved for the super user or root and the kernel automatically gives this UID complete access
- Notice the difference between kernel and root access
  - Kernel processes can access anything
  - Root processes can order the kernel to access anything

# Unix Users and User Groups

- The system must be able to tell users apart
- User number (user id, uid) identifies an user
  - Internal information for the operating system
  - Usually not visible for users
  - Ordinary users have number  $> 0$  (1 to 99 are used for different system accounts such as mail, news and sync)
  - Largest available user number on many systems is 60000
- Every user belongs also to at least one group
- Group number (group id, gid) identifies the group
  - Also internal information

## User names and /etc/passwd file

- Users are referred to by user names
- Every user name points to a user number
- Several user names can have the same user number
  - The operating system holds these accounts (almost) identical
- /etc/passwd connects user names to numbers
  - Also NIS, Kerberos...
  - Passwords are often held in file /etc/shadow
  - Other mechanisms exist

# Format of /etc/passwd

- Fields are separated by colon

```
username:password:uid:gid:comment:home directory:shell
```

- Samples

```
root:z83c0d958LH2E:0:0:The Superuser:/:/bin/sh
```

```
bin:*:2:2:0000-Admin:/usr/bin:
```

```
ali:WiqIjY17WlPk6:202:100:Ali Baba:/home/ali:/bin/bash
```

```
jim:0w93R8u6nJ2NT:205:200:JimJones:/home/jim:/opt/bin/db-app
```

- Separate accounts for administrator's own use and administrator as root (shadow password system used here)

```
root:x:0:0:Big Brother Watching You:/:/bin/sh
```

```
rckent:x:0:108:Clark Kent as root:/:/bin/sh
```

```
ckent:x:108:108:Clark Kent:/home/ckent:/usr/bin/bash
```

## /etc/passwd

- User name (login name)
- Password, encrypted
  - usually modified DES (MD5 is becoming quite common)
  - one way function, it is impossible to decrypt the password
  - at login the entered password is encrypted and compared to file
- User id (number)
- Login group id (number)
- GCOS (Comment, usually real-life name)
- Home directory
- Program to be executed at login, usually shell

# System Administrator's Accounts

- Three basic alternatives:
- Everybody shares the root account (bad)
- Everybody has their own root account (uid 0, username something else)
- Root access enablers
  - sudo, ena

## Using Permissions

- A file has owner and group id (sometimes several)
- A process has owner and group id (sometimes several)
- Kernel verifies permissions before executing system calls
  - If owner uid=0 (root), everything is allowed
  - Otherwise the uid and gid of the process and object are compared in this order and permission for the operation is searched for based on owner, group and other (world) rights

# Unix file permissions

- Permissions
  - r Read
  - w Write
  - x Execute or reference (for directories)
- For
  - Owner (User)
  - Group
  - World (Others)
- This file may be edited (rw) by its owner, read by the members of the "titu" group and not read by others

```
-rw-r----- 1 kiravuo titu 7627 Oct 1 12:50 exam
```
- This file may be edited by anybody:

```
-rw-rw-rw- 1 root root 12987 Sep 7 19:34 /etc/passwd
```

# Unix permission levels and bits

- The information is coded in two bytes in the inode of the file

```

Bit 15                               Bit 0
xxxx s s t r w x rwx rwx
  |   |   |   |   |
  |   |   |   |   | permissions for others
  |   |   |   |   | permissions for group
  |   |   |   |   | owner's x-permission
  |   |   |   |   | owner's write permission
  |   |   |   |   | owner's read permission
  |   |   |   |   | sticky bit
  |   |   |   |   | sgid bit
  |   |   |   |   | suid bit
file type (-, d, c, b, l, s, p)

```

# Structure of process

- Address space
  - Program code (text)
  - Data
  - Stack
  - Possible shared memory
- Properties
- Slice of processor time

# Suid and Sgid Mechanisms

- Usually a process (program) inherits the properties of the parent process
  - In most cases shell
- Suid and sgid bits in file permissions allow executing a program with the file owner and group IDs
  - Necessary for Unix to work, for example the passwd command, which is owned by root and allows an user to change his or her password

# Creating a Suid Program

- Easy:  
`chmod +s program`
- Or  
`chmod 4111 program`  
– no read permission
- The program is responsible for limiting the acts of the user
- A shell script should never be made a suid program
  - Read it again: NEVER
  - This a hole that is trivial to exploit

# Changing passwords

- Passwords are changed using the program `/bin/passwd`
  - Users can change only their own password, requires old password
  - root user can change any password
    - # `passwd username`
- Password files not writable (not even root), how does changing password work?

```
-r-s--x--x 1 root sys 15688 Oct 25 1995 /bin/passwd
-r--r--r-- 1 root sys 7627 Oct 1 12:50 /etc/passwd
-r----- 1 root sys 3292 Oct 2 15:14 /etc/shadow
```

## Different UIDs

- Real UID (RUID): UID of the user running program
- Effective UID (EUID): UID of user with whose privileges the program runs
  - SUID bit effects this
  - Access permission tests in system calls etc. use this
- These allow quite complicated swapping between UIDs

## Sample

- User riku executes the following program:  
`-r-s--x--x 1 root sys 15688 Oct 25 1995 /bin/passwd`
- The UIDs will be set as follows
  - RUID: riku
  - EUID: root

## UID-related system calls

- `getuid()` - returns RUID
- `geteuid()` - returns EUID
- `setuid(uid)` - set UID
  - If `EUID == root`, sets RUID, EUID
  - If not root, sets EUID if certain conditions are met
- `setruid(uid)` - sets RUID
- `seteuid(uid)` - sets EUID

# Handling names vs. IDs

```
#include <pwd.h>
struct passwd *pwp;
char *user_name;
```

- Get first user with given UID:

```
pwp = getpwuid(getuid());
user_name = pwp->pw_name;
```

- Get first user with given name:

```
pwp = getpwnam(user_name);
uid = pwp->pw_uid;
```

- Likewise, with groups

- The data structures are a bit more complicated because of multiple membership

# Implementing multilevel security in Unix

- All the users are in same level, only root and kernel are different
- Unix has no support for basic multilevel security
  - The more advanced multilevel models break down also, because Unix has only two security levels and because Unix does not support data classification
- Unix can be used as a trusted computing base (TCB) for systems that implement multilevel security using e.g.. separate computers for separate levels
  - However the mechanisms for data transfer between security levels do not exist
- Some versions of Unix have more advanced features and can provide multilevel security

## Unix and multilateral security

- Unix provides separation of users if file permissions are used correctly
  - Enforcing this is difficult, users can re-set the permissions
    - Discretionary Access Control
- Access to other user's resources (files) is provided using the permission groups, where each user must be listed for access to a resource
  - Access Control Lists (ACL) are available as an additional feature
  - Users who have access to a certain file can be named in a list

# Security vulnerabilities in Unix

- A security vulnerability potentially gives a user possibility to do things that are not allowed to him
- Reasons for vulnerabilities are
  - installation errors
  - implementation flaws on system programs
  - design flaws
- Typical vulnerabilities are bugs in suid-programs or network daemons
- Major problem in Unix is weak design and reliance on root
  - Everything must run as root, root has access to everything
  - No easy ways to limit risks

## Bugs in Unix programs

- Bugs that endanger security have been found from all layers of system: kernel, utilities and applications
- More are found occasionally
- All programs that are not trivial contain bugs and some of these cause security holes
- Most typical mistake is buffer overflow
  - Input is not treated with enough suspicion

## Different attackers

- Outside intruder who tries to cause harm to system
- Outside intruder who tries to get access to system (root privileges are most wanted)
- Local user who wants to get more rights than he has
- Local user who misuses his privileges
- Malware: virus, worm, trojan horse

## Different attacks

- Getting user name and password (by guessing or by some other means)
  - clean guess or systematically trying
  - Social engineering (strong weapon)
  - It is easier to get more privileges from inside a computer
- Portscanning
  - Tries to find out services that are running on a machine (and security vulnerabilities)
- Finding implementation flaws from programs
- Sniffing
  - listening to net traffic and catching passwords
  - all unprotected communication can be sniffed

# Guessing passwords

- About 25% of passwords are easy to guess
  - names, birth dates, phone numbers...
  - words (from the English or a local language)
  - etc.
- Login name and users name can be resolved e.g. using finger
- Terminal connection and pure guess might succeed
- More efficient way is to get password file and compare it to own list of encrypted passwords (using crack)
- Sometimes password can be found from a note (e.g. below keyboard)

## Root kits

- Intruders typically install a "root kit"
  - Replaces standard system binaries with ones that hide the intrusion
  - System administrator should have correct binaries on non-writable media (CD, write protected diskette)
- Changed system binaries can be detected with programs like Tripwire
  - Unless the attacker installs a weak root kit as deception and the actual root kit is a kernel module...
  - Must be kept on secure media itself

# Installing Unix

- Standard installations are now days somewhat reasonable
  - Blatant holes are rare
    - Debug-accounts
    - + in hosts.equiv
    - World-writable /etc/passwd
  - Usually too many services are installed by default
  - Default user profiles may have . in \$PATH
- The system should be checked after installation
  - Read manuals and READMEs
  - Go through /etc-directory
  - Check out user profiles

# Installing (free) Software

- Verify the source and distribution of the software
  - Be careful on sources
  - Most network distributed software comes now with cryptographically signed checksums
    - Do not get the key from the same source
- Plan a directory structure that supports several versions
- Read the documents
- Have a separate host and network segment for testing the software
- Follow discussions, do not be the first to test

# Upgrading Unix and Software

- Many upgrades mess up settings
  - /etc/passwd usually unchanged
  - Anything else might go...
    - sendmail configuration
    - inetd.conf
- While upgrades may fix old problems they may introduce new ones
  - Upgrades should be reversible
  - Soft links are a handy way to manage different versions
- Verify the source of upgrade
  - We have not seen fake OS upgrades sent to customers, yet

# Following Announcements

- Security problems are announced either when they are found or when they are fixed
  - Often several months in between
- Reading CERT mailing list is the minimum requirement
  - CERT announces problems only when a solution is available
- Bugtraq and other lists are faster
  - More work to weed through
- OS vendors might announce fixes also
  - User's product and vendor-specific mailing lists are worth joining
- Mailing lists can be directed to intranet, filtered etc.

## Be Careful of These

- Accounts with no password, group ID 0
- Unnecessary suid-programs
- PATH-environment variable
- Protection of directories and files
  - Home directories
  - Device files
  - System programs and configuration files
- Hidden directories
- Physical protection of servers and workstations

# Internal Threats

- Can all users be trusted?
  - Shell access gives an internal view to system, makes breaking security easier
- Unix was not designed from start to be secure
- Unix vendor's own special features
  - Access control lists (ACL)
  - Hidden passwords
  - Usually add-ons
  - Not always reliable and problem free

# File Permissions

- Monitoring system file permissions is recommended
  - Cops
  - SUID tai SGID:

```
find / -perm -4000 -o -perm -2000 -print
find / -perm +4000 -o -perm +2000 -print
find / -user root -perm +0022 -print
```
  - Find has some variations between Unixes
  - For monitoring open files and IP-connections lsof:

```
ftp://vic.cc.purdue.edu/pub/tools/unix/lsof
```
- Some files should not be readable or writable to world (e.g. /etc/shadow, /dev/hda)

## User leaves organization

- Management problem, with system administration issues
- User account should be closed either immediately when intentions are confirmed or at latest when user leaves
- Changing shell to /bin/false makes reactivation easier
  - Leaves holes in security like FTP, intranet systems, e-mail etc.
- A person who is no longer in organization should not have e-mail address in that organization
- All this should be in security policy

# Different approaches to OS security

- All operating systems have usually a kernel
  - The size and functionality of the kernel is important
  - Mach removes the file system, virtual memory and other components from the kernel to the user space, reducing the exposure to attacks
- Multiple security layers
  - E.g.. the backup system has access to all the user's files, but not to the system files
- Virtual machines
  - The user is very forcibly limited to the virtual instance of the computer