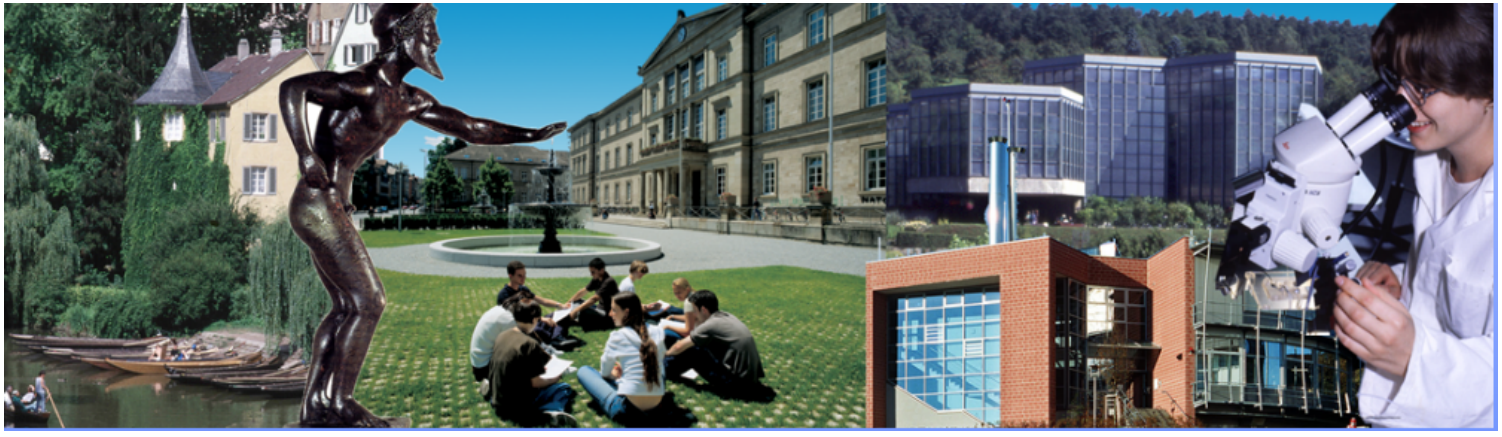
Eberhard Karls
Universität
Tübingen



Introduction to Computer Security

UNIX Security

Pavel Laskov
Wilhelm Schickard Institute for Computer Science

Genesis: UNIX vs. MULTICS

- MULTICS (Multiplexed Information and Computing Service)
  - a high-availability, modular, multi-component system
  - secure design from ground up: implementation of BLP
  - initial development from 1963 to 1969; continued until 1985; last system decommissioned in 2000
- UNIX: the opposite of MULTICS
  - initial assembler implementation by Ken Thompson and Dennis Ritchie for PDP-7 and PDP-11
  - rewritten in C in 1973: the first operating system written in a high-level language
  - continuous evolution of various dialects of UNIX and its routines for almost 40 years

UNIX security architecture

UNIX security architecture

None!

Security and UNIX design

Security and UNIX design

- Security was not a primary design goal of UNIX; dominant goals were modularity, portability and efficiency.

Security and UNIX design

- Security was not a primary design goal of UNIX; dominant goals were modularity, portability and efficiency.
- UNIX provides sufficient security mechanisms that have to be properly configured and administered.

Security and UNIX design

- Security was not a primary design goal of UNIX; dominant goals were modularity, portability and efficiency.
- UNIX provides sufficient security mechanisms that have to be properly configured and administered.
- The main security strength of UNIX systems comes from open source implementation which helps improve its code base.

Security and UNIX design

- Security was not a primary design goal of UNIX; dominant goals were modularity, portability and efficiency.
- UNIX provides sufficient security mechanisms that have to be properly configured and administered.
- The main security strength of UNIX systems comes from open source implementation which helps improve its code base.
- The main security weakness of UNIX systems comes from open source implementation resulting in a less professional code base.

Principals

- User identifiers (UID)
- Group identifiers (GID)
- A UID (GID) is always a 16-bit number
- A superuser (root) always has UID 0.
- UID information is stored in /etc/passwd
- GID information is stored in /etc/group

User account information: /etc/passwd

1. Username: used when user logs in, 1–32 characters long
2. Password: 'x' indicates that encrypted password is stored in /etc/shadow
3. User ID (UID): 0 reserved for root, 1-99 for other predefined accounts, 100-999 for system accounts/groups
4. Group ID (GID): the primary group ID
5. User ID Info: a comment field
6. Home directory: The absolute path to the directory the user will be in when they log in
7. Command/shell: The absolute path of a command or shell (/bin/bash)

/etc/passwd examples

```
root:x:0:0:root:/root:/bin/bash
dhcp:x:101:102::/nonexistent:/bin/false
syslog:x:102:103::/home/syslog:/bin/false
laskov:x:1000:1000:Pavel Laskov,,,:/home/laskov:/bin/bash
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
```

Shadow password file

1. Username: the user name
2. Passwd: the encrypted password
3. Last: days since Jan 1, 1970 that password was last changed
4. May: days before password may be changed
5. Must: days after which password must be changed
6. Warn: days before password is to expire that user is warned
7. Expire: days after password expires that account is disabled
8. Disable: days since Jan 1, 1970 that account is disabled

Examples:

```
root:!:14118:0:99999:7:::
laskov:$1$/et/grJh$xssVNwpdA35TwsSt7Yjvb/:14118:0:99999:7
```

Password encryption on UNIX

- DES
  - prepend password with 2-bit salt
  - take 7 lowest bits from first 8 characters
  - encrypt a fixed 64-bit string with DES using 56 bits as a key
  - convert the resulting 64 bits into 11 ASCII characters using 6 bits for character (2 bits padded with zeros)
- MD5
  - originally written for FreeBSD to avoid export restrictions
  - no limit on password size
  - is indicated by the starting 1 in the shadow file

Group file

1. Groupname: the group name
2. Password: an x indicates that a password is set and if left blank no password has been set
3. GID: the group ID number
4. Members: current members of the group separated by a comma

Examples:

```
root:x:0:
adm:x:4:laskov
laskov:x:1000:
```

Root privileges

- Almost no security checks:
  - all access control mechanisms turned off
  - can become an arbitrary user
  - can change system clock
- Some restrictions remain but can be overcome:
  - cannot write to read-only file system but can remount them as writable
  - cannot decrypt passwords but can reset them
- Any user name can be root!

```
root:x:0:1:root:/:/bin/sh
funnybunny:x:0:101:Nice Guy:/home/funnybunny:/bin/sh
```

Subjects

- The subjects in UNIX are processes identified by a process ID (PID).
- New process creation
  - fork: spawns a new child process which is an identical process to the parent except for a new PID
  - vfork: the same as fork except that memory is shared between the two processes
  - exec family: replaces the current process with a new process image
- Processes are mapped to UIDs (principal-subject mapping) in either of the following ways:
  - real UID is always inherited from the parent process
  - effective UID is either inherited from the parent process or from the owner of the file to be executed

Objects

- Files, directories, memory devices, I/O devices etc. are uniformly treated as resources subject to access control.
- All resources are organized in tree-structured hierarchy
- Each resource in a directory is a pointer to the inode data structure that describes essential resource properties.

Inode Structure

mode	file type and access control rights
uid	user name
gid	group name
atime	last access time
mtime	last modification time
itime	last inode change time
block count	size of the file in blocks
ptr	pointers to physical blocks with file contents

Mode field in detail

- File/resource type

'-' file	
'd' directory	's' socket
'b' block device file	'l' symbolic link
'c' character device file	'p' FIFO

- Access control rules (permissions)

owner rights	'r', 'w', 'e', '-'
group rights	'r', 'w', 'e', '-'
"world" rights	'r', 'w', 'e', '-'

- Examples

```
-rw-r--r-- 1 laskov laskov 10652 ... 08-unix.tex
lrwxrwxrwx 1 root root 15 ... stdin -> /proc/self/fd/0
crw------- 1 laskov tty 136 ... /dev/pts/1
```

Directory permissions

- read: searching a directory using e.g. `ls`
- write: modifying directory contents, creating and deleting files and directories
- execute: making a directory current and/or opening files in it

Managing permissions

- Octal encoding of permissions

  read-only: $100_B \Rightarrow 4$
  read-write: $110_B \Rightarrow 6$
  read-write-execute: $111_B \Rightarrow 7$

- Modifying permissions

  chmod 777 filename
  chmod u+rwx,g+rx,o-w filename

- Changing file owner (root only)

  chown user:group filename

Default permissions

- Default permissions are usually 666 for files and 777 for programs.
- umask command changes default permissions
  - synopsis: `umask mask`
  - the inverse of `mask` is added to the current permissions
- Examples:

def. perm.	mask	inv. mask	result
777	022	755	755
777	027	750	750
666	033	744	644
666	077	700	600

Controlled invocation

- Certain actions, e.g. using system ports (1-1023) or changing a password, require root privileges.
- We don’t want to give users a general root privilege by telling them a root password, but only the right to run selected commands as root.

Controlled invocation

- Certain actions, e.g. using system ports (1-1023) or changing a password, require root privileges.
- We don't want to give users a general root privilege by telling them a root password, but only the right to run selected commands as root.
- Solution: set a special flag indicating that a program can be run under the privilege of its owner rather than that of a calling user.

Controlled invocation

- Certain actions, e.g. using system ports (1-1023) or changing a password, require root privileges.
- We don't want to give users a general root privilege by telling them a root password, but only the right to run selected commands as root.
- Solution: set a special flag indicating that a program can be run under the privilege of its owner rather than that of a calling user.
- Disadvantage: this right cannot be given to selected users: all users in the "world" (or in a group) can run a program under its owner's privilege.

SUID, SGID and sticky flags

- A fourth octal number is added to permissions with the following bit designations:
  - SUID: set UID (allow all users to run a program)
  - SGID: set GID (allow all users in a specific group to run a program)
  - sticky flag: only an owner (or root) can remove files in a directory
- Use `chmod` with four octal digits to set the extra flags:

```
chmod 7644 08-unix.tex
ls -l 08-unix.tex
-rwSr-Sr-T 1 laskov laskov 13031 ... 08-unix.tex
```

Secure mounting of filesystems

- By mounting an external file system we cannot guarantee that it is free from malicious programs, e.g. SUID to root programs.
- As a result, access control setting may need to be redefined for mounted media:
- Security options to the `mount` command:
  - `-r`: read-only mount
  - `-o nosuid`: turn off SUID flags for all data in a mounted file system
  - `-o noexec`: no program can be run from a mounted file system
  - `-o nodev`: no character or block device can be accessed from a mounted file system

Search paths

- A potential danger lies attacker's diverting of execution of a wrong program with the same name.
- Rules of conduct:
  - If possible, specify full paths when calling programs, e.g. `/bin/sh` instead of `sh`.
  - The same applied to programs to be run locally: use `./program` instead of `program`.
  - Make sure . is the first symbol in the `PATH` variable. This will at least prevent calling a "remote" version of a program if what you really want is a "local" invocation.

Security features missing in UNIX

- ACLs in general (`getfacl` only gets permissions)
- Data labeling, e.g. secret, classified etc.
- Mandatory access control, so that individuals are unable to overrun certain security decisions made by an admin (e.g. `chmod 777 $HOME` is always possible)
- Capabilities are supported by only a small subset of UNIX-like operating systems (e.g. Linux with kernel versions above 2.4.19)
- Standardized auditing

Summary

- UNIX provides a set of flexible security mechanisms; however, their efficacy relies on careful and knowledgable administration.
- UNIX does not provide several key features suggested by security models, e.g. no ACLs or security levels.
- The main security strength lies in its open source implementation; hence, security flaws are discovered and fixed early.