

Users and Groups

unique identifier for each user is a *numeric UID* (user id),

UID=0 is super user, usually called `root`

a user is member of one or more groups

one group is the principal group, the one found in `/etc/passwd`
this group is used as group owner for files the user created, unless he uses `newgrp`

other group memberships are located in `/etc/group`

Who am I?

Which UID do I have? Command Shell:

```
$ id
uid=2030(sysi30) gid=1000(stud) groups=1000(stud)
```

Which UID do I have? C-program:

```
uid_t u; /* this usually is a 16-bit-integer */
u=getuid();
```

There is a command

```
$ who am I
```

but it doesn't really show who I am ...

```
root          ttyt1    Jun 13 23:20 (localhost)
```

...but who that terminal belongs to.

Which UIDs do processes have?

- RUID: real User-ID
who starts the process
this is also inherited from parent processes
- EUID: effective User-ID
decides about access to system resources

these two are different only if `setuid-bit` set

```
-r-sr-xr-x 2 root wheel 5828 Jan 12 08:41 /usr/bin/passwd
```

this is controlled by the system call `execve()`

Changing UIDs within a process (1)

Why should we want to do that?

- permission issues that are not solved by the filesystem
- security *principle of least privilege*

Changing UIDs within a process (2)

- from the command-shell: use command `su`
 - need username + password
 - with option `-,` simulate login
 - from a C-program: use functions `setuid()`, `seteuid()`
 - process has real UID and effective UID
 - only allowed to switch between original RUID and EUID
 - this is implemented storing the EUID on startup in a third (hidden) *saved UID*
- `set(e)uid()` fails, if... the user is not the super user and the ID specified is not the real, effective ID, or saved ID.

Managing UIDs: Programmer's View

functions that work with UIDs

`setuid()` different historical implementations

→ `setuid mess`, read article

http://yarchive.net/comp/setuid_mess.html

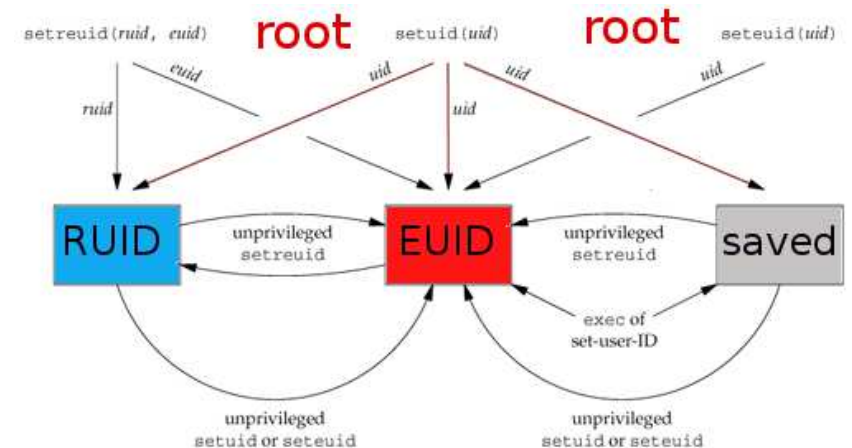
<code>getuid()</code>	return real UID
<code>geteuid()</code>	return effective UID
<code>setuid()</code>	set effective UID (root: EUID+RUID)
<code>seteuid()</code>	set effective UID
<code>setreuid()</code>	set real and effective UID

su-command, simulating login

```
# su dweber
[t] u@h(#)#

# su - dweber
[13:38:44] dweber@isl-c-01(1)$
```

Managing UIDs: Programmer's View (2)



Managing UIDs: Programmer's View (3)

consequence/portability:

- use `seteuid()` whenever possible
- use `setuid()` to drop all privileges

Managing Users: Files (Linux/Solaris)

- user database is `/etc/passwd`

```
root:x:0:0:root,,,:/root:/bin/bash
ftpadmin:x:1004:1003:FTP Administrator:/home/ftp:/bin/tcsh
ftp:x:40:2:ftp account:/usr/local/ftp:/bin/true
user1:x:1001:1000:Some User:/home/user1:/bin/bash
.....
name  x  UID   GID  description  home  dir  shell
```

note: a user may change his default shell by `chsh`.

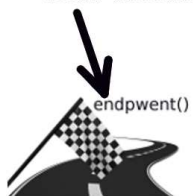
- encrypted passwords are stored in `/etc/shadow`

```
user1:PkEZq9UQ/d9v.:11991:0:99999:7:::
user2:Ye40QU1rqln8s:11992:0:99999:7:::
user3:aeFRGxtgth.s:12064:0:99999:7:::
user4:KsiYtUF6aGDUQ:12123:0:99999:7:::
.....
name, encrypted passwd, password aging info
```

/etc/passwd Entries: Programmer's View

`setpwent()`

```
■ root:$1$8Mj4lzRQ$...:0:0::0:0:Charlie &:/root:/bin/csh
toor:*:0:0::0:0:Bourne-again Superuser:/root:
daemon:*:1:1::0:0:system processes:/root:/usr/sbin/nologin
user1:*:1000:1000::0:0:System &:/bin/csh
```



`getpwent()`

Managing Users: Files (BSD)

- user database is `/etc/master.passwd`

```
-rw----- 1 root wheel /etc/master.passwd
root:$1$8Mj4lzRQ$...:0:0::0:0:Charlie &:/root:/bin/csh
toor:*:0:0::0:0:Bourne-again Superuser:/root:
daemon:*:1:1::0:0:system processes:/root:/usr/sbin/nologin
user1:*:1000:1000::0:0:System &:/bin/csh
.....
name  PWD  UID   GID  class  pwd-change  expire  descr.  home  dir  shell
```

- copy w/o passwords is stored in `/etc/passwd`

```
-rw-r--r-- 1 root wheel 1357 Mar 12 12:35 /etc/passwd
root:*:0:0:Charlie &:/root:/bin/csh
toor:*:0:0:Bourne-again Superuser:/root:
daemon:*:1:1:system processes:/root:/usr/sbin/nologin
user1:*:1000:1000:System &:/bin/csh
```

The Password Encryption/Hash

iterated versions of the following encryption- or hash-methods

- DES cipher:
encrypt 000...0
with key = password (25 iterations)
- Blowfish cipher
- MD5 hash (1000 iterations)
- SHA-256 hash
- SHA-512 hash



cryptographic hash functions work like this:



see manual page crypt(3) for more information

Attacks

invert Hashing / Encryption

~>analyze algorithm, very hard (crypto research topic)

dictionary attack (variations of dictionary words)

brute force (= exhaustive search)

Examples

outdated DES is default method (and fallback) for passwords

```
openssl passwd -salt AbCdEfG secret_password
```

Warning: truncating password to 8 characters

```
AbKLsS6u5sAh6
```

several systems today use MD5

```
openssl passwd -1 -salt AbCdEfG secret_password
```

```
$1$AbCdEfG$PPiziSx3vbgV1HnIvpJAZO
```

Dictionary Attack

```
Aachen      250d6e3dc34afb195271904349fcf790
```

```
Aachener    bb6fae8a70240eb9f26b0c8a53345d08
```

```
Aachenerin  107b911e2cec78856a4676ea3ce16f92
```

```
Aachenerinnen 657b25a7aff45f9434c36d4b1479cde3
```

```
Aachenern   6bc4b0cbdda46a3c30b19d3a1a6fbf5c
```

...

```
zytotoxischer 9b64262fe97427370242dbc4061722ba
```

```
zytotoxisches 1efec802b37771252068b36ee1ce0067
```

```
zzgl         71832d182a57a01f13b11014a1264cf7
```

135,000 words in German Duden (2¹⁷)

Brute Force Attack

```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000002
00000000000000000000000000000003
00000000000000000000000000000004
...
ffffffffffffffffffffffffffffffffffff
```

2^{128} bit strings of length 128

Other Authentication Methods (1/2)

- Challenge/Response
 - server sends x , client sends $MD5(x + pass)$ to server
 - used in APOP, POP3–authentication
 - attacked in 2008 (Leurent)
 - drawback: clear-text passwords on the server
- One–Time–Passwords
 - a random password list (strong PRNG needed)
 - used in PIN/TAN, S/Key, OPIE
 - OPIE (library) *One time Passwords In Everything*
 - drawback: store password lists

Defenses

against inverting: strong cryptographic hash

against dictionary/brute force:

- salt
- long passphrases / special characters
- * CPU–/RAM–intensive operation (GPUs have small RAM)
- password shadowing
- per user files (/etc/tcb), in OpenWall Linux, 2005


(*) proposal from OpenBSD: bcrypt

Other Authentication Methods (2/2)

- Public–Key–Crypto
 - used in SSH
 - explained later in this course
- Secure Remote Password protocol
- Kerberos ticket–granting–ticket

A Note on Secure One-Time-Passwords and TANs (1)

need a **secure hash function**: SHA-3, SHA-256, Blake2, Grøstl,...



One-Time-Password Generation:

x3 ← Einbahnstraße x2 ← Einbahnstraße x1 ← Einbahnstraße x0

User gets following password list: **x3, x2, x1, x0**

Hacker's Problem:

x3 → [red circle with white bar] → x2 → [red circle with white bar] → x1 → [red circle with white bar] → x0

Other Password Use Cases than Login

~key-derivation function transforms password into key

- disk encryption
- securing ZIP-, RAR-files
- wireless networks (WPA2)
- GPG, PGP e-mail encryption
- password vaults

A Note on Secure One-Time-Passwords and TANs (2)

Implementation:

- INIT: system stores x_4
- the user enters x_3 as his first password
- the system compares $h(x_3) = x_4$, if unequal, permission denied
- the system stores x_3
- next time the user enters x_2
- the system compares $h(x_2) = x_3, \dots$

~system does not need to store the whole list, only the last used password

Which Future Key-Derivation-Function?

GPU- and ASIC-unfriendly, the brute-force-attacking devices

- not 32-bit-based
- huge memory requirements (more than a GPU-thread can handle)
- lots of data dependent branching (no similar results in each thread)

~not necessarily standard hash functions (~scrypt?)

May 2014:
specialized ASIC mining hardware for scrypt-based cryptocurrencies.

Back to UNIX-Usermanagement: Concept of Groups

each user belongs to *exactly one* principal group (\sim /etc/passwd)

the group ID and name defined in /etc/group

users may belong to additional groups

```
$ id theobald
```

```
uid=55177(theobald) gid=1111(stl)
```

```
groups=1111(stl), 1113(stlnagios),60001(cuda)
```

corresponding entries in /etc/group

```
cuda:*:60001:dweber,bohr,theobald
```