Systemy operacyjne

Wykład 6:
Tożsamość, upoważnienie i uwierzytelnianie

Pojęcia podstawowe

W systemach z wieloma użytkownikami wymagana jest **kontrola dostępu** (ang. *access control*) do zasobów.

Procesy mają **tożsamość** (ang. *identity*) użytkownika, z reguły tego który je utworzył. W bloku kontrolnym procesu jądro trzyma niepodrabialne **kredencjały** (ang. credential).

Upoważnienie (ang. *authorization*) to proces sprawdzania, czy proces o danej tożsamości ma dostęp do zasobu.

Tożsamość należy potwierdzić przy pomocy mechanizmu **uwierzytelniania** (ang. *authentication*), np. podanie hasła, klucza prywatnego, odcisku palca, zdjęcia siatkówki.

Tożsamość procesu Uniksowego

- **użytkownik** → **`getuid(2)`**
- **grupa podstawowa** → **`getgid(2)`**
- **grupy rozszerzone** → **`getgroups(2)`**

Jądro przechowuje identyfikatory w postaci **numerycznej**!

Wyróżniony użytkownik **`root`** (uid=0) jest uprzywilejowany!

```
# id
uid=1000(cahir) gid=1000(cahir) grupy=1000(cahir),
24(cdrom),27(sudo),29(audio),44(video),46(plugdev),
108(netdev),123(vboxusers),999(docker)
```

Skąd się biorą nazwy użytkowników i grup?

Przypisanie nazw do identyfikatorów oddelegowane całkowicie do przestrzeni użytkownika. Obsługuje to baza danych usługi Name Service Switch.

Zazwyczaj źródło danych znajduje się w plikach: **`/etc/passwd`**, **`/etc/groups`**, **`/etc/shadow`**, ale dane można też pobierać z usług sieciowych (LDAP).

Bazę danych użytkowników / grup można wydrukować przy pomocy `getent`. Istnieje zestaw narzędzi systemowych do modyfikacji bazy danych np.: `useradd`.

PAM: uwierzytelnianie w świecie Unix

PAM to biblioteka systemowa służąca do implementacji programów, które muszą między innymi **uwierzytelniać**.

Dla każdego programu w **`/etc/pam.d`** zdefiniowane reguły:

- **`account`** → upoważnienie (np. czy konto aktywne?)
- **`auth`** → uwierzytelnianie (np. zapytanie o hasło)
- **`password`** → zmiana hasła (np. czy hasło jest zbyt słabe?)
- **`session`** → akcje podejmowane przed i po zakończeniu sesji (np. ustawienie zmiennych środowiskowych, zamontowanie zaszyfrowanego systemu plików, a na koniec jego odmontowanie)

Upoważnienie w systemach uniksowych

Proces może prosić jądro o przydzielenie zasobu z danej **przestrzeni nazw** → system plików, procesy, …

Jądro udziela lub odmawia dostępu (upoważnienie) do:

- zasobu plikowego na podstawie kredencjałów, trybu dostępu (**`O_RDWR`**), limitów systemowych, oraz oczywiście metadanych pliku;
- wysłania sygnału do innego procesu, wyłącznie na podstawie kredencjałów procesu źródłowego i docelowego;

Zmiana tożsamości przez `setuid` / `setgid`

Jądro utrzymuje trzy identyfikatory użytkownika:

1. **rzeczywisty** (ang. *real id*)
2. **obowiązujący** (ang. *effective id*)
3. **zachowany** (ang. *saved set-user id*)

W trakcie sprawdzania uprawnień brany jest pod uwagę tylko *effective id*, przy pomocy `setuid` / `setgid` użytkownik:

- `root` → zmienia wszystkie trzy identyfikatory na dowolny
- pozostali → *effective-id* może być ustawiony na *real-id* albo *saved-id*

Szczegółom przyjrzymy się później...

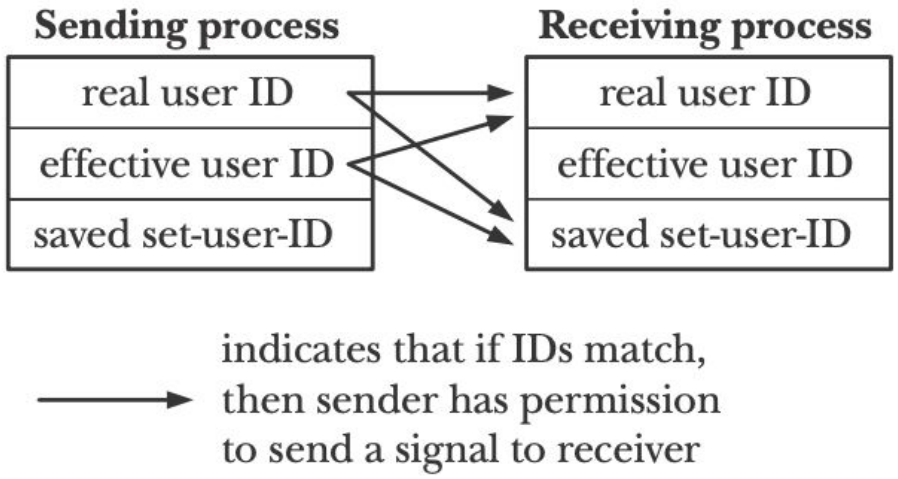
Przykład: program `login(1)`

Uruchamiany na konsolach systemowych (`Ctrl+Alt+Fx`) z uprawnieniami użytkownika `root`.

Pyta o nazwę użytkownika i hasło, jeśli poprawne tworzy proces powłoki na uprawnieniach użytkownika.

1. Czyta `/etc/{passwd,shadow,group}`.
2. Uwierzytelnia użytkownika → wyłącza **`ECHO`**, czeka na hasło i sprawdza z tym wczytanym z `/etc/shadow`.
3. Zmienia grupy dodatkowe (`setgroups`).
4. Zmienia uid i gid (`setuid`, `setgid`).
5. Uruchamia powłokę z wpisu w `/etc/passwd`.

Upoważnienie: wysyłanie sygnałów (LPI 20-5)



Dotyczy **wyłącznie** sygnałów wysyłanych przy pomocy `kill(2)`. Jądro jest uprzywilejowane, tak jak użytkownik **`root`**, więc zawsze może wysyłać sygnał do procesu (np. **`SIGTSTP`** albo **`SIGSEGV`**).

Jeśli sprawdzenie według powyższej tabelki nie wyjdzie → **`EPERM`**. Wyjątkiem jest **`SIGCONT`** w obrębie tej samej sesji. **Dlaczego?**

Upoważnienie: dostęp do pliku (LPI 15.4.3)

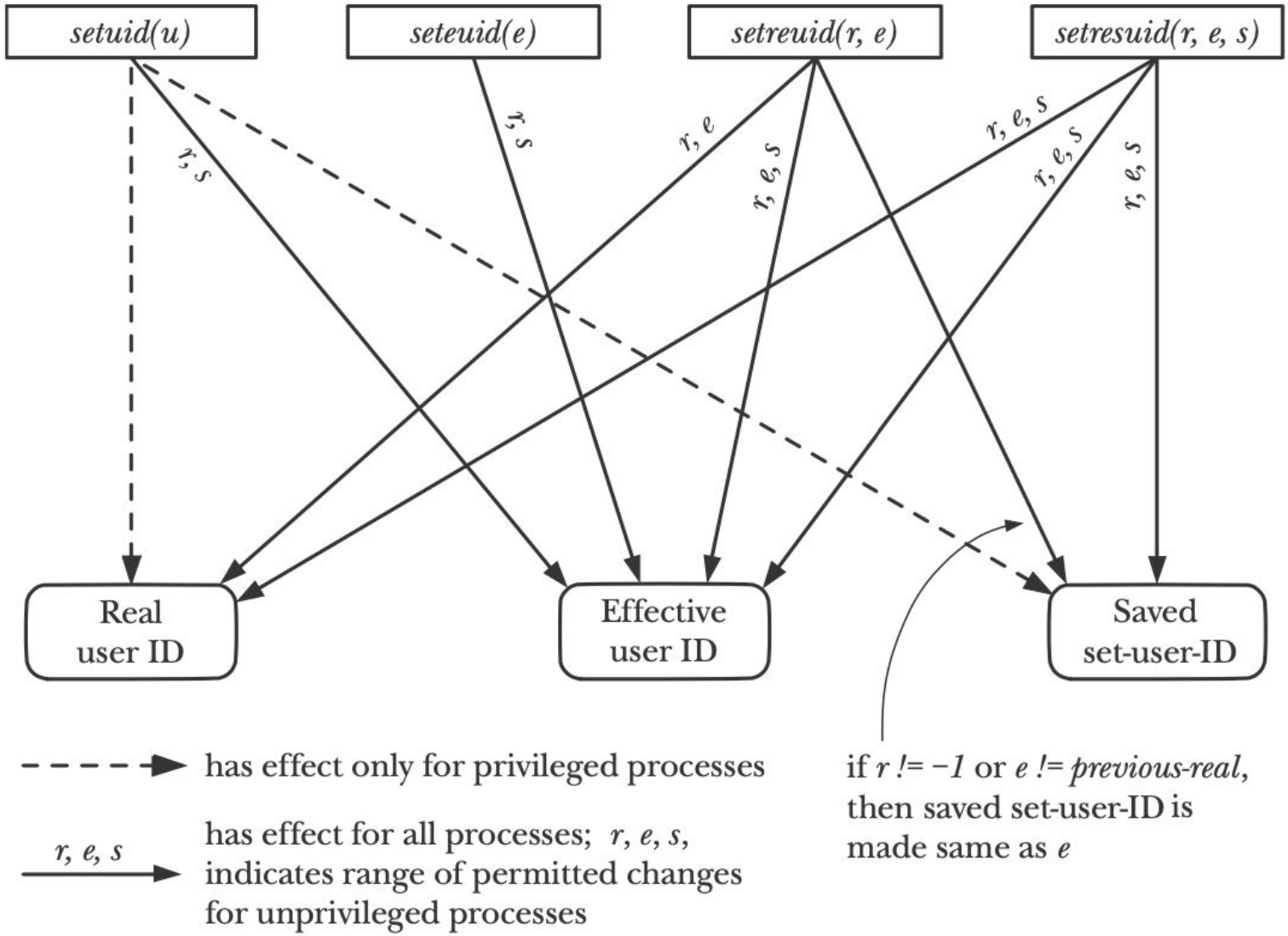
Zachodzi wyłącznie w momencie otwierania pliku!

1. Gdy użytkownik uprzywilejowany → może wszystko.
2. Jeśli **`euid`** zgadza się z właścicielem pliku → sprawdź czy tryb dostępu pokrywa się z bitami **`S_IRWXU`**; koniec.
3. Jeśli **`egid`** zgadza się z grupą pliku → sprawdź czy tryb dostępu pokrywa się z bitami **`S_IRWXG`**; koniec.
4. W p.p. sprawdź czy tryb dostępu pokrywa się z bitami **`S_IRWXO`**; koniec.

Co znaczą bity uprawnień dla katalogów?

- Bit **`R`**: Można czytać wyłącznie zawartość pliku katalogu → **`getdirents`** działa, ale **`stat`** nie.
- Bit **`W`**: Można modyfikować zawartość pliku katalogu
  → działają **`creat`**, **`unlink`**, **`rename`**;
  Uwaga: Sprawdzany jest wyłącznie właściciel katalogu!
- Bit **`X`**: Można czytać metadane plików w katalogu, czyli działa **`stat`**. Można przeczytać zawartość pliku lub przejść do podkatalogu pod warunkiem, że znamy nazwę.
- Bit **`sticky:`** Plik może usunąć tylko jego właściciel.

Wywołania systemowe do zmiany tożsamości



Wymiana tożsamości

Wywołaniami systemowymi możemy tylko zmieniać obowiązującą tożsamość (`euid`) na rzeczywistą (`ruid`) lub zachowaną (`suid`).

Co jeśli potrzebujemy tymczasowo zmienić tożsamość, żeby wykonać zadania administracyjne, np.: zmienić hasło, wydrukować użytkowników zalogowanych do systemu, zainstalować pakiet?

W trakcie instalacji systemu niektórym narzędziom zostają nadane dodatkowe bity uprawnień: **`set-uid`** / **`set-gid`**. W trakcie zastępowania przestrzeni adresowej `execve` jądro wymienia tożsamość procesowi na zgodną z właścicielem lub grupą pliku pliku wykonywalnego (przykład: `su`, `passwd`).

Zmiana tożsamości przez wywołanie `execve`

ID	exec		setuid(*uid*)	
	set-user-ID bit off	set-user-ID bit on	superuser	unprivileged user
real user ID	unchanged	unchanged	set to *uid*	unchanged
effective user ID	unchanged	set from user ID of program file	set to *uid*	set to *uid*
saved set-user ID	copied from effective user ID	copied from effective user ID	set to *uid*	unchanged

Rozważmy wykonanie programu `wall(1)` (bit `set-gid`, grupa `tty`, `gid=4`) użytkownikiem cahir `uid=1000`, `gid=1000`.

- przed: `rgid=1000`, `egid=1000`, `sgid=1000`.
- po: `rgid=1000`, `egid=4`, `sgid=4`.

Proces może przełączać między `rgid` i `sgid`, np. żeby odczytać plik konfiguracyjny z katalogu użytkownika.

Po co zmieniać tożsamość?

Konstruujemy aplikacje zgodnie z modelem przyznawania najmniejszego możliwego zestawu uprawnień.

Innymi słowy: ile uprawnień można odebrać procesowi, żeby jeszcze był się w stanie wykonać swoje zadanie.

Pożądany rezultatem jest zmniejszenie prawdopodobieństwa wystąpienia podatności na atak złośliwych użytkowników.

Podział programu na podprocesy, które mogą mniej niż proces główny. Delegowanie zadań przez gniazda lub potoki do wykonania w piaskownicy (ang. sandbox) o ograniczonych uprawnieniach.

Inne metody ograniczania uprawnień

1. Zmiana korzenia systemu plików → `chroot(2)`
   Użytkownik `root` przed zmianą tożsamości może zawęzić widoczność systemu plików do danego katalogu.
2. Ograniczanie możliwości (ang. *capabilities*) procesu poprzez wyłączanie dostępu do wywołań systemowych, np. nie można korzystać z `open(2)`, albo używać podzbioru operacji na deskryptorach plików `fcntl(2)`.
3. OpenBSD: `pledge(2)` i `unveil(2)`

Capsicum: practical capabilities for UNIX

Watson et al., USENIX Security Symposium 2010

Pytania?