TDIU11 – Föreläsning 6

Säkerhet och skydd

Filip Strömbäck

LINKÖPING UNIVERSITY

Planering

Vecka	Föreläsning	Seminarie
3	Processer och trådar	—
4	Minneshantering	Challenge 1: Schemaläggning
5	Virtuellt minne	Artikel 1: Schemaläggning
6	Filsystem och lagring	Challenge 2: Virtuellt minne
7	Säkerhet	Challenge 3: Filsystem
8	Repetition/utblickar	Artikel 2: Filsystem
9	—	Challenge 4: Säkerhet
10	Tentaförberedelse	Challenge 5: Repetition
11	(omtenta-p)	Artikel 3: Säkerhet (reserv)

LINKÖPING UNIVERSITY

Mål

Vi har *objekt* vi vill skydda.

De ska användas av rätt
personer, på rätt sätt.

Mål

Vi har *objekt* vi vill skydda.

De ska användas av rätt personer, på rätt sätt.

Objekt: resurser i systemet som behöver skyddas. Varje objekt har ett antal *operationer*.

Exempelvis: filer, hårdvara, ...

Domän samling av rättigheter.

Exempelvis: användare, grupp, process, ...

Modell: Åtkomstmatris (Access matrix)

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Bild från material till *Operating System Concepts*

Rader: domäner
Kolumner: objekt

Att fundera på:

- Vad ska matrisen innehålla? Hur ska vi tänka?
- Matrisen blir stor. Hur lagrar vi den?

Hur ska en åtkomstmatris konstrueras?

- Minsta möjliga privilegier (principle of least privilege):
  En domän ska ha minsta möjliga åtkomst för att utföra sin uppgift.
- Vid behov (need to know):
  En process ska, *vid varje tidpunkt*, bara ha de rättigheter den behöver just då.

Ofta avvägning mellan säkerhet och användarvänlighet

Hur detaljerad ska vi vara?

- Grovkornig kontroll
  Ofta enklare, men vi får ofta "för mycket" privilegier under "för lång" tid.
  Exempelvis: En process är antingen i domänen av en användare, eller `root` med *alla* rättigheter
- Finkornig kontroll
  Minimerar risker, men mer komplext att hantera och använda.
  Exempelvis: Finkorninga filåtkomster, tillfällig byte av domän, individuella rättigheter för program, ...

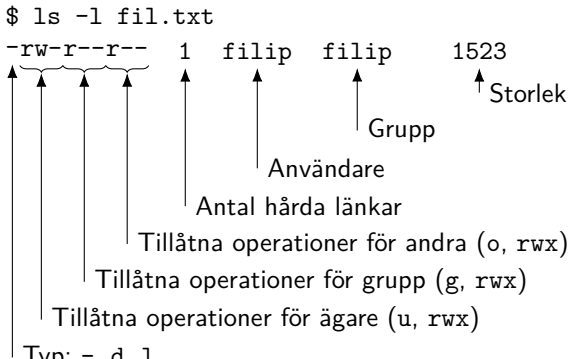
Implementation: Access-Control Lists

Idé: Lagra kolumner hos varje objekt.

⇒ Filrättigheter (UNIX)

- ACL (UNIX + Windows NT)

Enkelt att veta vilka domäner som har tillgång till objekt



Implementation: Access-Control Lists

Idé: Lagra kolumner hos varje objekt.

- Filrättigheter (UNIX)

$\Rightarrow$ ACL (UNIX + Windows NT)

Enkelt att veta vilka domäner som har tillgång till objekt

```
$ getfacl fil.txt
# file: fil.txt
# owner: filip
# group: filip
user::rw-
group::r--
group:sudo:rw-
other::r--
```

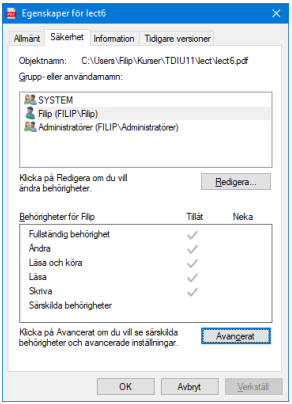
Implementation: Access-Control Lists

Idé: Lagra kolumner hos varje objekt.

- Filrättigheter (UNIX)

$\Rightarrow$ ACL (UNIX + Windows NT)

Enkelt att veta vilka domäner som har tillgång till objekt



Implementation: Capabilities

Idé: Lagra rader hos varje domän
Exempel:

- Fildeskriptorer
- Specifika capabilities
- `pledge` och `unveil` i OpenBSD

Enkelt för domänen att veta vad som kan användas

Exempel: En process som har öppnat en fil kan komma åt den oavsätt om rättigheterna ändras

Exempel: Vem som helst får inte skapa "råa" sockets:

```
$ getcap /usr/bin/ping
/usr/bin/ping cap_net_raw=ep
```

ACL eller Capability?

ACL eller capability?

- Öppna en fil
- Fildeskriptorer

Hur kan vi ta bort rättigheter till ett objekt i fallen ovan?

ACL eller Capability?

ACL eller capability?

- Öppna en fil
- Fildeskriptorer

Hur kan vi ta bort rättigheter till ett objekt i fallen ovan?

ACL: Gå till objektet, ta bort rättigheter för relevanta domäner

Capability: Gå till domänerna, ta bort rättigheter för objekt

Byte av domän

För att följa *need-to-know* (och ibland *principle of least privilege*) behöver vi ibland byta domän.

Exempel:

- Ändra systeminställningar
- Ändra nätverkskonfiguration (ansluta till trådlösa nätverk)
- Montera filsystem
- Installera program

Vi vill inte ge rättigheterna till en användare. Vi vill inte att alla program ska ha rättigheterna ovan. Hur gör vi?

Setuid och Setgid

Ber OS att köra ett program som en annan användare eller grupp:

- Tillåter mer flexibilitet – kan bygga logik som inte går att åstadkomma på annat sätt
- `chmod u+s`
- `chmod g+s`
- Relevant för körbara filer
- Buggar i setuid-program leder till stora problem!
- Notera: `root` får göra vad som helst

Exempel:

- `sudo`
- `su`
- `mount`

Kodexempel:
`setuid.cpp`

Capabilities

För att bättre uppfylla *least privilege* kan vi ge en process delar av `root`:

- `setcap`
- `man capabilities`
- Reducerar risken för problem om något går fel
- Kräver fortfarande försiktighet

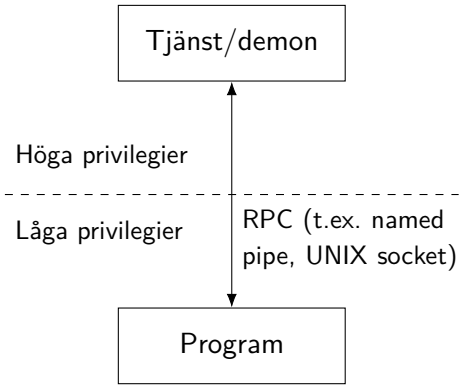
Exempel:
`/usr/bin/ping`

Tjänster/Demonprocesser

En annan strategi: be en annan process om hjälp

- Kräver bara rättigheter till *RPC*-mekanism
- Tjänst/demon måste köras
- Risk för buggar...

Exempel: displayserver, kommunikation med hårdvara, ...



LINKÖPING UNIVERSITY

Autentisering: Lösenord

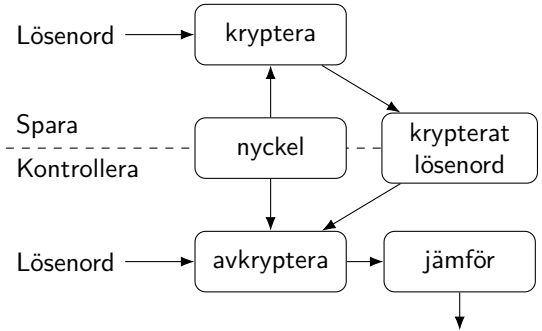
Hur lagrar vi lösenord?

- I klartext?

Autentisering: Lösenord

Hur lagrar vi lösenord?

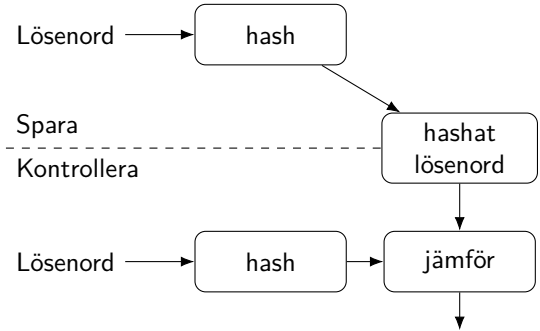
- I klartext?
- Krypterade?



Autentisering: Lösenord

Hur lagrar vi lösenord?

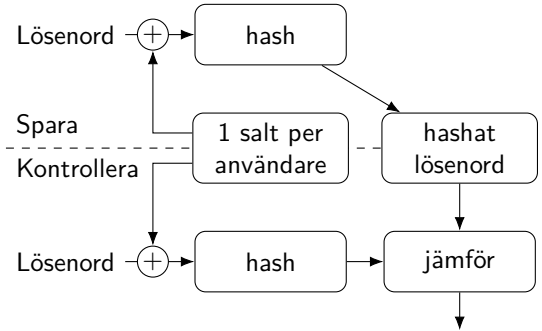
- I klartext?
- Krypterade?
- Hashade?



Autentisering: Lösenord

Hur lagrar vi lösenord?

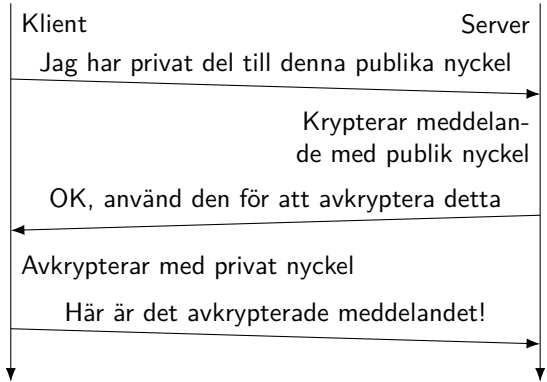
- I klartext?
- Krypterade?
- Hashade?
- Hash + salt!



Autentisering: Kryptografi

Exempelvis: SSH-nycklar

- Generera ett nyckelpar
- *Publik* nyckel kan distribueras
- *Privat* nyckel hålls hemlig
- Kan autentisera utan att ge bort vår privata nyckel!



LINKÖPING UNIVERSITY

Injektionsattacker

Följande program körs som `root`

```
int main(int argc, const char *argv[]) {
    std::string input = argv[1];
    std::string command = "cat " + input;
    system(command.c_str());
}
```

`read_line` läser indata från användaren

`system` kör kommando som om det skrivits i terminalen

Vad kan gå fel?

Injektionsattacker

```
int main(int argc, const char *argv[]) {
    std::string username = argv[1];
    std::string password = hash_password(argv[2]);
    std::string query = "SELECT COUNT(*) FROM users "
        + "WHERE username = '" + input
        + "' AND password = '" + hashed + "';";
    int count = execute_db_query(query);
    if (count == 0) return 0;
    // Success! ...
}
```

Vad kan gå fel?

Skydd mot injektionsattacker

- Var försiktig med strängar!
- Strängdata tolkas ibland som ett språk (bash och SQL i exemplen)
- Måste i så fall vara försiktig med s.k. *escape*-tecken
- Finns ofta mekanismer för att hantera användardata separat:
  - `exec(<array>)`
  - Prepared statements

Buffer overflow

Följande program läser indata från användaren:

```
int main() {
    char buffer[128];
    int total = readline(buffer);
    std::cout << "Läste " << total << " tecken: "
        << buffer << std::endl;
    return 0;
}
```

Vad kan gå fel?

Skydd mot buffer overflow

Finns olika tekniker:

- Sentinelvärden på stacken (stack cookies)
- Ej exekeverbar stack
- Plats för stack, program, heap slumpas vid varje körning

Men:

- Gör det bara *svårare* att ta sig runt. Det går fortfarande.
- Kan fortfarande *ändra* på data, modifiera datastrukturer.
- Kan fortfarande *läcka* data (vid läsning, heartbleed).

Nästa föreläsning

Repetition/utblickar

Vad ska vi fokusera på?

- Schemaläggning
- Minneshantering
- Lagring
- Säkerhet
- Kompilering, länkning, körning
- Virtualisering
- Kryptografi

Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY