MER PROGRAMMERING

USER/GROUP ID

```
#include <sys/types.h>
#include <unistd.h>
int setuid(uid_t uid);
```

- Om processen är superuser
  - sätter real/effective/saved-suid = *uid*
- Otherwise
  - euid=*uid* om *uid* == real user id eller *uid*== saved-set-user-id

```
int setgid(gid_t gid);
```

- Samma som setuid fast för grupper

USER/GROUP ID

- Endast superuser processer kan ändra real UID – görs i normala fall av login-programmet.
- EUID sätts av `exec` endast om setuid-biten är satt för programfilen.
- `exec` kopierar EUID till saved-suid (efter att ha satt EUID om setuid-biten är satt).
- Se Figur 8.18, Sid 238

/etc/passwd

- Innehåller namn, krypterat passord, idnummer, gruppnummer, ett kommentarfält, hembibliotek och vilket shell som skall startas

```
#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
struct passwd *getpwnam(const char *name);
```

- Eventuellt:

```
struct passwd *getpwent(void);/*jmf readdir/opendir*/
void setpwent(void); /* jmf rewinddir/opendir */
void endpwent(void); /* jmf closedir */
```

/etc/group

- Innehåller namn, krypterat passord, idnummer och en vektor av pekare till de användarkonton som ingår i gruppen.

```
#include <sys/types.h>
#include <grp.h>
struct group *getgrgid(gid_t gid); /*jmf getpwuid*/
struct group *getgrnam(const char *name);/*jmf
getpwnam */
```

- Gå igenom hela filen:

```
struct group *getgrent(void); /*jmf getpwent*/
void setgrent(void); /* jmf setpwent */
void endgrent(void); /* jmf endpwent */
```

VARIABELT ANTAL ARGUMENT

- I C finns möjligheten att använda ett variabelt antal argument i anropet av en funktionen. Det finns några macron definierade så att man kan arbeta med dessa.

```
va_start
va_arg
va_end
```

- Notera att den första parametern till `va_start` är det sista argumentet innan den variabla listan och det andra argumentet till `va_arg` är den typ som man förväntar sig.

HUR VET MAN ANTALET ARGUMENT?

- Funktionen måste på något vis veta hur många argument den har fått.
- Man kan använda ett särskilt värde för att ange att listan är slut (t.ex. `NULL`).
  - T ex `execlp`
- Ett annat sätt är att först skicka ett heltal som anger antalet argument.
  - Här använder `printf` en speciell metod: antalet argument är lika med antalet konverteringsspecifierare (`%d %s` osv.) som ingår i formatsträngen

SOCKETS

SOCKETS

- Ytterligare ett sätt att kommunicera.
- Med sockets så kan vi både kommunicera över nätverk och lokalt på en maskin.
- Lite bökiga att komma igång med då vi kan välja många olika protokoll för kommunikationen etc. För internetförbindelser kan vi t.ex. välja mellan TCP (Transmission Control Protocol) och UDP (User Datagram Protocol) .
- Många av funktionerna som jobbar med fildeskriptorer funkar också med sockets

DOMÄN, TYP OCH PROTOKOLL

- **Domänen** tillhandahåller ett adresseringssystem och ett antal protokoll. Sockets ansluter bara till andra sockets i samma domän
  - AF_INET
    - Internet
  - AF_UNIX
    - Internt på en unixdator

DOMÄN, TYP OCH PROTOKOLL

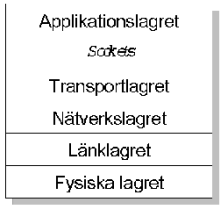
- **Typen** bestämmer semantiken för kommunikationen
  - SOCK_STREAM
    - Tillförlitlig tvåvägs byteström. Socketarna måste vara hopkopplade med connect. Vi kan använda read och write
  - SOCK_DGRAM
    - otillförlitlig meddelande (fix längd) kommunikation. Vi skickar meddelanden med sendto och tar emot med recvfrom

DOMÄN, TYP OCH PROTOKOLL

- **Protokoll**
  - kan oftast bestämmas utifrån domän och typ. T.ex. AF_INET och SOCK_STREAM ger protokollet TCP

Sockets i TCP/IP-stacken

- Applikationslagret
  - HTTP, SMTP, DNS
- Transportlagret
  - TCP, UDP
- Nätverkslagret
  - IP(v4/v6), ICMP, IPSec
- Länklagret
  - Ethernet, WLAN, GPRS, PPP
- Fysiska lagret
  - Sköter den faktiska kodningen och överföringen av bitar



- Sockets: ett gränssnitt mellan applikationen och transportlagret
- Identifieras unikt:
  - IP-adress, port, protokoll

UNIX DOMAIN SOCKETS

- Domänen AF_UNIX
- Sockets lokalt på en dator.
- Använder sig av filsystemet för adresser

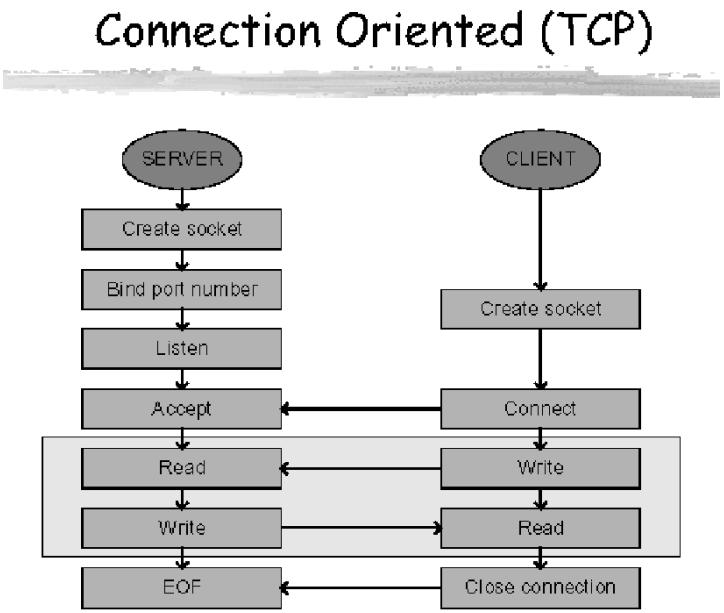
- `socketpair(int domain, int type, int protocol, int socket_vector[2])`
  - Genererar två hopkopplade sockets som man kan kommunicera mellan på ett liknande sätt som om vi använt pipe. Båda socketarna går dock att både läsa och skriva från

OPERATIONER FÖR EN SERVER

- socket
  - Skapar en namnlös socket
- bind
  - Ger socketen en adress (ett namn)
- listen
  - Meddelar att processen är villig att ta emot anslutningar till socketen
- accept
  - Acceptera ett anslutningsförsök. Ger tillbaka en ny socket med samma egenskaper som kan användas för den faktiska kommunikationen med klienten.

OPERATIONER FÖR EN KLIENT

- socket
  - Skapar en namnlös socket
- connect
  - Anslut socketen till en annan socket



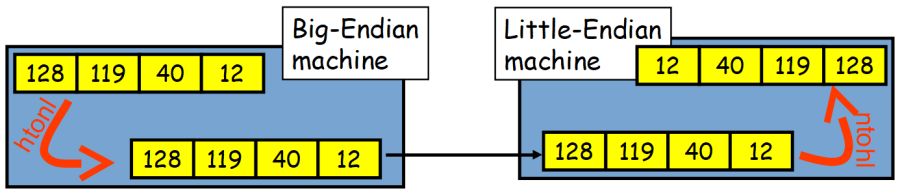
BYTE-ORDNING

- Data kan representeras olika på olika arkitekturer
  - Little-endian: minst signifikanta byte först
    - Kallas ibland *Intel order*
  - Big-endian: mest signifikanta byte först
    - Kallas ibland *Motorola order*
- Exempel: 32-bitarstalet 23798

Little-endian:	0	0	92	246
Big-endian:	246	92	0	0

BYTE-ORDNING

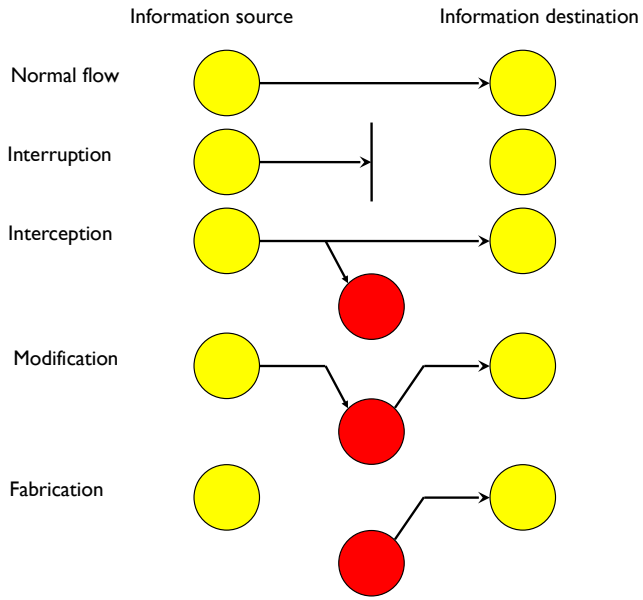
- Network byte order (Big-endian)
  - För utbyte av multibyte-data på nätet
  - `htonl(),htons(),ntohl(),ntohs()`
    - Konverterar till/från nätverk/lokal byte-ordning



SOCKETS – MER INFO

- För mer info om nätverkskommunikation, sockets och de olika protokollen man kan använda och deras egenskaper rekommenderar jag att ni läser en datakommunikationskurs

SÄKERHET



FAROR

- Sabotage (virus, trojaner, spionprogram)
- Intrång (”inbrott” eller slarv)
- Olyckor (diskfel, fummel vid tgb)
- Handhavande (studenter som hittar tentor)
- Fysisk säkerhet (kommer man åt servern)
- Identitet (verifiering av maskiner och personer)
- Programfel (ex buffer overflow, lita på “skitig” information)
- Systemfel (dåliga policybeslut

SÄKERHET

- Viktigt
- Komplext
  - datakommunikation
  - matematik
  - programutveckling
  - organisation
  - planering
  - Policy
  - …

PROGRAMFEL

- Ett alldeles för vanligt fel
- Slarviga eller ovetande programmerare
  - Fel algoritmer
  - Tankefel
  - Testar inte alla fall

GENERELL SÄKERHET

- Här finns det inga direkta rätt eller fel.
- Snarare saker man bör tänka på när man implementerar ett program.
- Kort sagt: Efter att du kommit på hur du tänkt göra, sov på saken och tänk en gång till...

LOGGNING - HUR MYCKET SKA MAN LOGGA?

- Inloggningsrutinerna i Unix
- loggar alla inloggningsförsök. Om en användare av misstag skriver in sitt lösenord istället för användar-namn kommer det att stå i klartext i logfilerna. Om det sedan loggar in en användare 3 sekunder senare är det inte svårt att lista ut och matcha användarnamn och lösenord. Problemet är att man vill kunna logga ev. intrångsförsök. Vid användarnamn som inte finns, ska man logga det automatiskt?

LOGGNING

- Webserver
  - loggar ibland komplett Url, även det som står efter ?-tecknet. Det som står efter ?-tecknet (query-strängen) brukar innehålla användarnamn och ev. lösenord för gästböcker och liknande.
  - Behövs det verkligen dyka upp i log-filerna?
    - Problemet är att man vill kunna se vem som gjort ett inlägg i en gästbok t.ex..
    - En lösning: använda metoden post istället för get när det gäller känslig data. Då måste man själv göra nödvändiga loggningar för att kunna kontrollera vem som gjort vad på sin hemsida.

LOGGNING

- Tänk på vad som ska loggas när ni skriver ett program.
- Ska allt loggas, även om det riskerar att innehålla känsliga uppgifter?
- Vem ska kunna läsa logfilerna?

LÖSENORD

- Ett lösenord på 8 tecken kan vara ett av 256^8 kombinationer. Men i realiteten kan man lista ut ganska mycket om ett lösenord.
- Ofta brukar man inte tillåta tecken som man inte kan skriva på ett normalt tangentbord. Redan då är man nere i ca 127^8 kombinationer.
- Därefter kan man anta att ett lösenord inte innehåller fler siffror än tecken, inte fler specialtecken än vanliga tecken, inte fler versaler än gemener osv...
  - Helt plötsligt är man nere i ännu färre kombinationer.

LÖSENORD

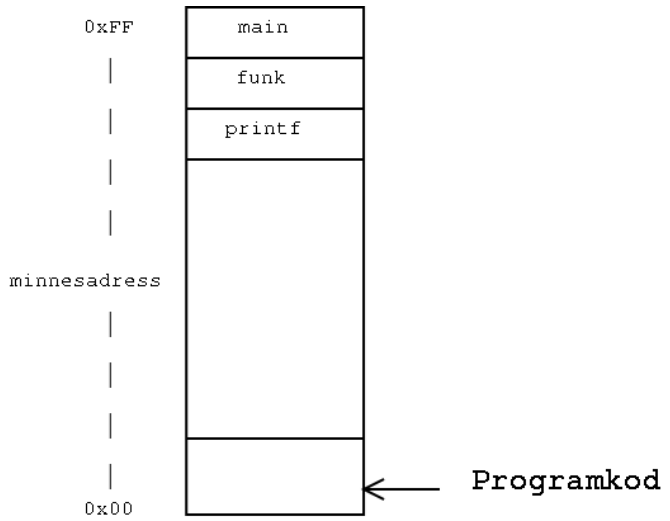
- Dessutom använder många användare samma lösenord på flera ställen.
  - Vad finns det för garantier att ett lösenord lagras på ett säkert sätt på ett annat system?
- Vid ett nytt system, kan man tvinga användaren att använda ett nytt lösenord?
  - Hur ska man lagra lösenordet så att det inte går att komma åt även om man hackar datorn som det ligger lagrat på?
- Mer läsning finns på http://www.psychpage.com/tech/passwords.html

PROGRAMINFO – VEM ÄR DU?

- Många datorer och program sprider gladeligen ut information om vad och vilken version de kör.
  - Om nu den versionen har en säkerhetsbrist är det väldigt enkelt för en hackare att skanna runt i ett nätverk för att få reda på vilka datorer som är känsliga för en attack.
  - Å andra sidan, om den inte på ett enkelt sätt visar versionsnummer är det svårare för sysadmins att hålla koll på att man har de senaste patcharna.
- Ska vem som helt kunna få ut information om ett system? Behöver alla användare få reda på vilken tid och med vilka flaggor ett visst program kompilerades?
- Om man portscannar en dator kan man få en hel del info om möjliga mål för en attack.

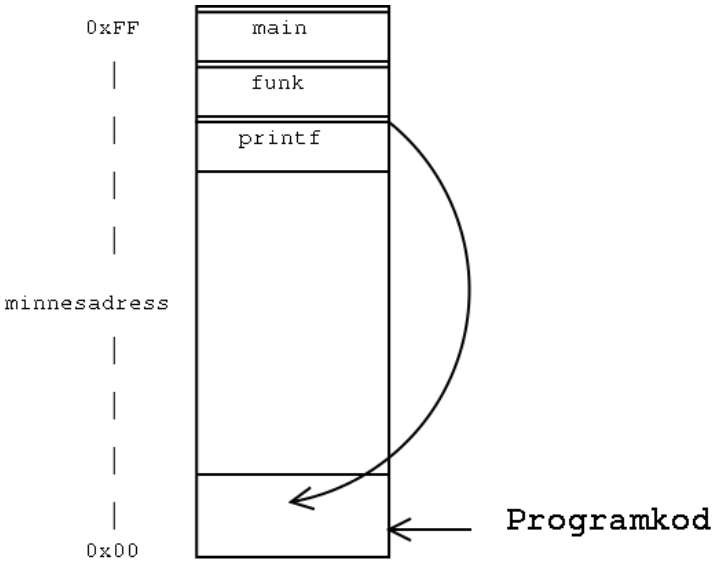
PROGRAMMERING

- Varje program får ett eget minnesutrymme som det körs i.
- Själva programmet läses in och hamnar på ett ställe och variablerna hamnar på ett annat.
- För att man ska kunna ha lokala variabler i varje funktion reserveras nytt minne varje gång en funktion anropas.



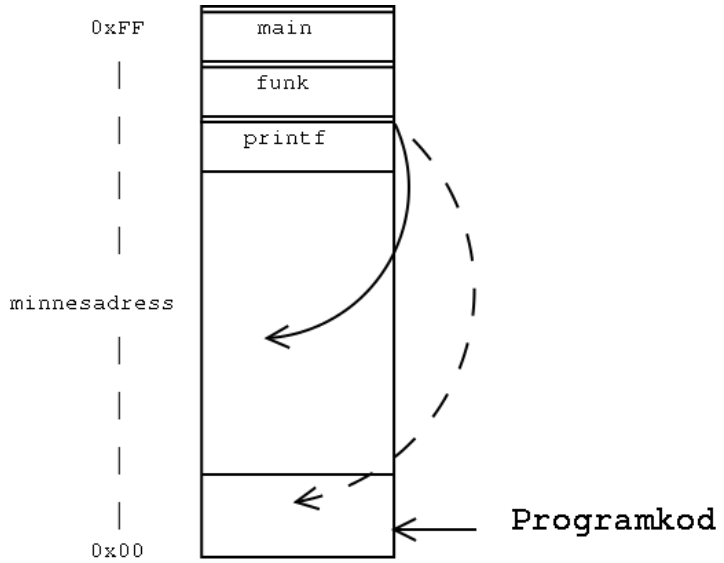
BUFFER OVERFLOW

- Varje instans av en funktion har en egen minnesplats för sina variabler.
  - Men t.ex. printf kan ju anropas från flera ställen i koden, hur vet den då vart den var innan anropet?
  - Genom att just innan det "lokala" minnet lägga upp den plats den var på, kan den sen hittat tillbaka till vart den var innan funktionsanropet.
- På så vis kan man anropa samma funktion från flera ställen i koden.
  - När printf avslutar kollar den den sk återhoppsadressen och hoppar tillbaka dit i koden.



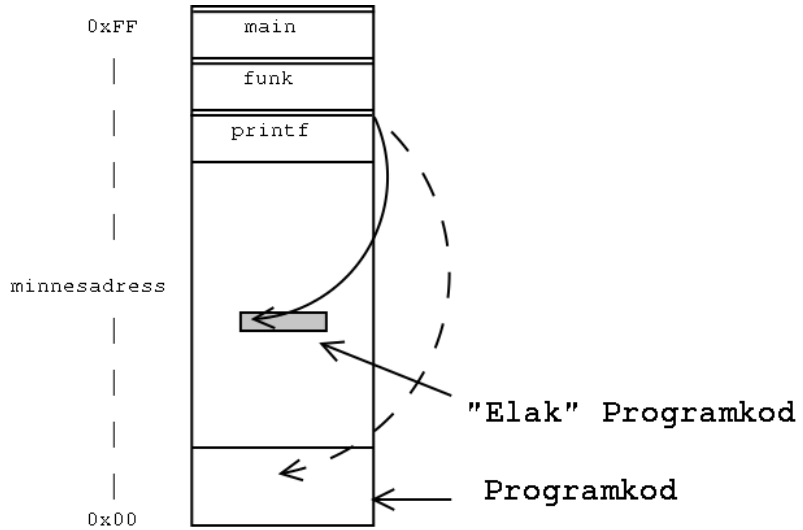
BUFFER OVERFLOW

- Det som händer vid en buffer overflow är att man t.ex. skriver utanför en array.
- Eftersom allt ligger i närheten av varandra i minnet riskerar man då att skriva över adressen som den använder för att hitta tillbaka i koden.
  - Vanligen kraschar programmet eftersom det inte blir en godkänd adress.



UTNYTTJA BUFFER OVERFLOW

- Vid en exploit så gör man en kontrollerad buffer overflow.
- Istället för att skriva sönder återhopps-adressen skriver man dit ett värde som gör att programmet hoppar till den kod som man själv vill köra.
- Den programkoden får man oftast in via stdin, inläsning från en fil, en miljövariabel, via en nätverksuppkoppling eller liknande. Huvudsaken att den finns någonstans i minnet.



KONTROLLERA ALL INDATA!

- Speciellt om ni skriver ett nätverksprogram eller ett program som ska köras av root/med root-rättigheter. Annars kan programmet hackas på ett väldigt enkelt sätt...
- Har man en buffert på 10 byte ska man inte undra några omständigheter kunna försöka fylla den med mer än 10 byte.

HUR SKYDDAR SIG OS MOT DETTA?

- Några exempel
  - Lagrar en checksumma på vissa ställen i stacken och kan då märka om något som inte borde ha ändrats har gjort det
  - Spärra stacken från att kunna exekveras
  - Byta plats på stacken mellan körningar
- Alla dessa går att ta sig runt (även om det gör det hela betydligt bökigare)

OLIKA TYPER AV FEL

- Segmentation fault
  - Får man om man försöker komma åt en minnesadress man inte har tillgång till eller försöker skriva till en adress som är read-only
  - Vanligen har man tappat kontrollen över en pekare

OLIKA TYPER AV FEL

- Illegal instruction
  - Får man om man försöker köra kod som inte är giltig maskinkod
  - Trolig orsak: kvaddad återhoppsadress

OLIKA TYPER AV FEL

- Bus error
  - Beror på att processorn försöker optimera läsning till och från minnet.
  - På en 32-bitarsmaskin läser den in 4 byte (32-bitar) åt gången.
  - För att det ska bli snabbt måste adressen som den läser/skriver vara jämt delbar med 4. Är den inte det blir det ett bus error.
  - Trolig orsak: man har tappat kontrollen över en pekare eller, återhoppsadressen är trasig.

VIDARE LÄSNING

- På webben finns mycket information om man letar.
- *Secure-Programs-HOWTO* är en ganska komplett genomgång på olika fel man kan göra. Inte enbart i C-programmering.
  - Kan rekommenderas att skumma igenom om man vill få en överblick över vilka fallgropar som finns när man skriver kod. Finns länk till den under ”Diverse länkar” på Cambro.
  - `https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html`