

# Turvalisest programmeerimisest

Meelis Roos

(Tartu Ülikool, Cybernetica)

mroos@ut.ee

Securefest

9. aprill 2004

# Kava

- Üldist
- Sisendi kontroll
- Võidujooksud
- Puhvri ületäitumised
- Veebirakenduste turvaprobleeme
- Shell, SQL
- (PHP)

## Üldist

- Turvalisus tuleb algusest peale sisse disainida, mitte hiljem paigata
- Turvalisust saab kontrollida mingil konkreetsel ajahetkel kehtiva seisundi kohta
- Ei saa teha nimekirja keelatud tegevustest
- Paranoia on programmeerimisel vooruseks
- Keerukus on vaenlane

## Miks programmid ei ole turvalised?

- Programmeerijaid ei õpetata piisavalt
- Ei mõelda mitmekasutajasüsteemidele
- Programmeerijad on laisad
- Programmeerijad pole turvaspetsialistid
- Turvalisuse tagamine võtab aega ja raha
- C/C++ on ebaturvalised keeled
- Kasutajad ei hooli
- Turvamudelid on kehvad
- Palju vana katkist tarkvara on kasutusel

## Enimohustatud programmiidid

- Mitteusaldatud allikast saadud andmete vaatamise programmid (brauser, editor, ...)
  - \* Igasuguste võõraste programmide käivitamine on ohtlik!
  - \* Makrod kui programmi eriliik
- Administraatori poolt käivitavad programmid
- Kohalikud serverid
- Võrguteenuseid pakuvad programmid
- Veebirakendused!!!
- Rakendid (*applet*)
- *Setuid/setgid* programmid

## Sisendi kontroll

- Programm peab edukalt toime tulema igasuguse sisendiga, mida talle võidakse anda, ka vigasega
- Aktsepteeritav sisend peab vastama soovitud mustrile
- Aktsepteeritav sisend peab vastama soovitud mahupiirangutele
- Sisendiks pole ainult sisseloetavad andmed — ka süsteemist saadu
- Testida ka valede läbilaskmiste ja valede filtreerimiste vastu (mitte ainult oodatud tulemuste kohta!)
- Sisendit kontrollida iga mooduli liidese juures, mitte ainult kasutajaliideses
- Lubatud mustrid vs keelatud mustrid?

## Sisendi valideerimine

- Programmi argumendid (käsurida)
- Keskkonnamuutujad
- Failipedemed
- Failide sisu
- Signaalid
- umask
- Jooksev kataloog
- ...

## Sisendi valideerimine — veeb

- URLi kodeering (+Unicode)
- Küpsised (*cookies*)
- Ei piisa kliendipoolsest andmete valideerimisest (JavaScript, ...)
- ...

## Võidujooksud

- Võidujooks — programmi töö korrektsus sõltub ajast (võistlusest välise teguriga)
- Tekib jagatud ressursside sünkroniseerimata kasutamisel
- Võidujookse võib tekkida nii usaldatavate kui mitteusaldatavate protsessidega
- Näide: ajutise faili varastamine
- Näide: nimeviidete kaudu muude failide üle kirjutamine
- Lahenduseks atomaarsete operatsioonide kasutamine
- Juhuslikud failinimed — kas ikka juhuslikud?
- Lukustamine — vahend sünkroniseerimiseks, enamasti sõbralike osapoolte vahel

## Head turvapraktikat

- Kontrollige kõigi funktsioonide poolt tagastatavat tulemust vigade suhtes
- Kasutage väliseid komponente ainult programmeerijale mõeldud liideste kaudu
- Ettevaatust andmetüüpidega (*int vs long, signed vs unsigned, ...*)
- Ärge kasutage varieeruva tähendusega ega ilmselt ebaturvalisi teegifunktsioone
- Ärge asjata tehke *setuid/setgid* programme; kui teete, siis pisi-kesi

## Avatud lähtekood ja turvalisus

- Kas avatud lähtekood annab turvalisust juurde?
- Rohkem ülevaatajaid  $\Rightarrow$  rohkem vigade leidjaid
- Võimalus veenduda koodi kvaliteedis
- Avatud kood iseenesest ei garanteeri kvaliteeti
- Ründajal on rohkem infot
- Ründajal on vähem eeliseid kaitsja ees
- Parandamise võimalus
- Kokkuvõtteks: avatud lähtekoodil on parem potentsiaal *saada* turvaliseks

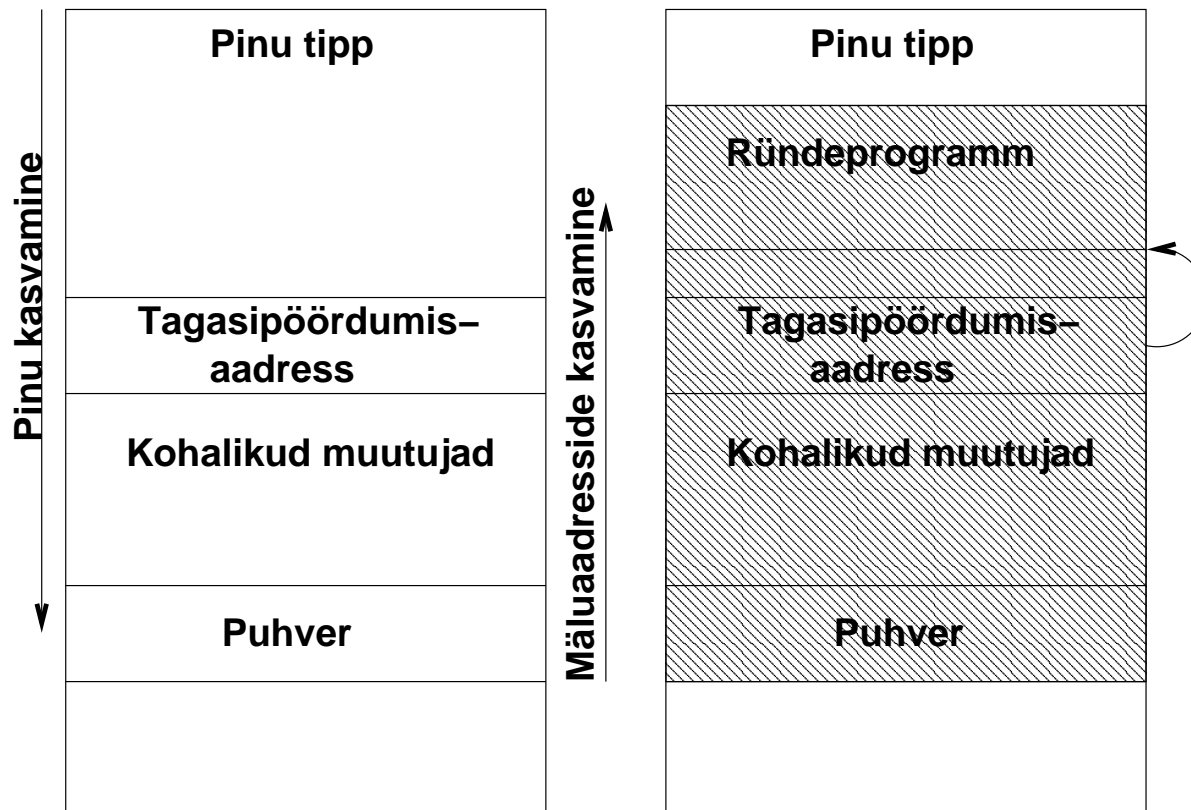
## Buffer Overflow

- Puhvri ületäitumine — üks levinumaid turvaprobleeme programmides
- Juba Morris Interneti-uss kasutas sellist auku aastal 1988
- Von Neumanni arhitektuur: programm ja andmed on samamoodi samasuguses mälus
- Lohakas programmeerija ei kontrolli, kas andmed reaalselt puhvrisesse mahuvad
- Kirjutatakse üle mälu puhvri järelt

## Puhvri ületäitmine koodi täitmiseks

- Pinus asuvate puhvrite puhul soditakse ära pinu ja muudetakse tagasipöördumisaadress
- Ka 1-baidist puhvri ületäitumist on turvaauguna ära kasutatud!
- Peale pinu võib ka muid mälualasid rünnata (*heap*, teiste muutujate üle kirjutamine)

# Pinu ja puhvrid



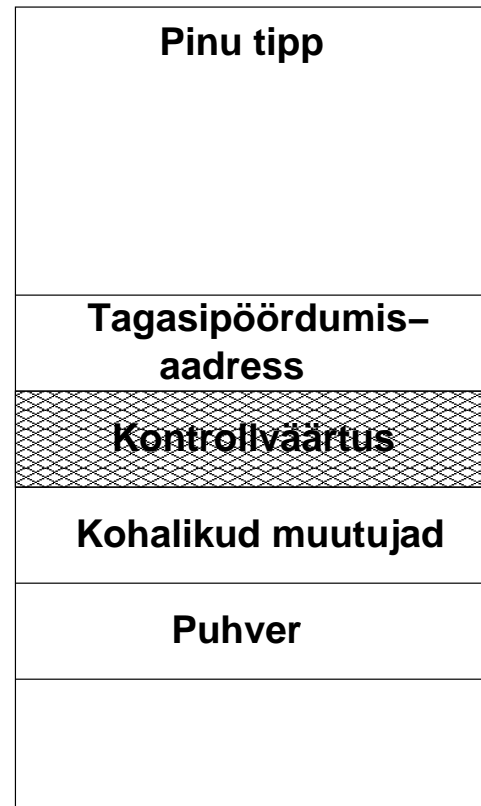
## Puhvri ületäitumiste kaitse

- Päriskaitse
  - \* Kontrollime kõigi puhvrite pikkusi
  - \* Ei kasuta ebaturvalisi funktsioone, mis lasevad üle puhvri otsa kirjutada
- Kõik muud on mittetäielikud hädalahendused

## Puhvri ületäitumiste kaitse abivahendid

- Mälukaitse (mittetäidetav pinu, W<sup>X</sup>, ...)
- Pinusse kontrollväärtuste lisamine (StackGuard, ProPolice, ...)
  - \* Randomiseerimine
  - \* Nullbaidi sisalduvus
- Teegid pinuviidaga arvestavate variantidega ohtlikest funktsioonidest (*libsafe*, *libverify*, ...)

# Pinu kaitse



## Ohtlikke funktsioone

`strcpy(char *dest, const char *src)`

dest ületäitumine → `strncpy`

`strcat(char *dest, const char *src)`

dest ületäitumine → `strncat`

`getwd(char *buf)`

buf ületäitumine → `getcwd`

`gets(char *s)`

s ületäitumine → `fgets`

`[vf]scanf(const char *format, ...)`

argumentide ületäitumine

→ argumentide max pikkused mustrisse

`[v]sprintf(char *str,`

`const char *format, ...)`

str ületäitumine → `[v]snprintf`

## Aukliku C koodi näide

```
#include <string.h>

int main(int argc, char * argv [])
{
    char buffer [1024];

    if (argc > 1)
        strcpy(buffer, argv[1]);
    return 0;
}
```

## Muid ohtlikke konstruktsioone

- `system(string);`
- `popen(string);`
- `printf(str);` või `printf("%s", str);` ?
- See viimane on formaadistringi rünnak: `%n` käsib printf-l kirjutada muutujasse seni väljastada kästud sümbolite arvu
- Edasised argumendid loetakse pinust järjest, sinna satub enamasti just meie formaadistring
- Oma formaadistringis saame anda mälupesa aadressi, mida hilisem `%n` täidab meie poolt ette antud väärtusega
- $\Rightarrow$  Ründaja saab kirjutada mälus peaaegu suvalisele aadressile soovitud väärtusi

## Veebirakenduste turvaprobleeme

- Olemuselt magus rünnakukoht
- Sisend täiesti ebausaldusväärne
- Ei saa eeldada, et sisend oleks tulnud meie poolt ette nähtud HTML vormist
- Ei saa eeldada, et peidetud muutujad veebivormil on ainult meie teada
- HTTP\_REFERER pole usaldusväärne
- POST vs GET
- Kelle õigustes veebirakenduste koodi täidetakse?
- Failide *upload*

## Veebi turvapraktikat

- Veebi kaudu kättesaadavaks teha AINULT need failid, mis tõesti on otse kasutaja poolt kättesaadavaks mõeldud. Siia ei kuulu:
  - \* Täiendavad programmifailid
  - \* Andmefailid
  - \* Failid autentimisinfoaga
  - \* Ajutised failid
  - \* Lähtekood (ka varukoopiaatena!)
  - \* Sessioonifailid, uploaditud failid, . . .

## Veel veebi turvapraktikat

- Väliste programmide ja andmefailide nimed olgu võimalusel absoluutse kataloogiteega
- Veebiserveril olgu õigus lugeda ainult minimaalset vajalikku hulka failidest
- Ainult minimaalselt hulgal kasutajatel olgu õigus veebi sisu muuta
- Karda eval()'it jms koodi täitvaid funktsioone

## Käsuinterpretaatorid

- sh, bash, ksh, zsh, csh, tcsh, ...
- Võimalusel vältida turvakriitilistes rakendustes
- *Setuid* skriptid – põhimõtteline turvaprobleem (missugust interpretaatorit kasutada?)
- *csh programming considered harmful*
- Ettevaatust keskkonnamuutujatega! PATH, IFS, CDPATH, ENV, BASH\_ENV, ...

## Veel käsinterpreteritest

- Metamärgid: Shellis eritähendusega!  
& ; ' ' \ " | \* ? ~ ! ^ # < > ( ) [] {} \$ \n \r
- Ajutistest failidest on parem hoiduda, enamasti aitavad torud (*pipe*)
- Kõik signaalid ise kinni püüda ja adekvaatselt reageerida (keh-tib ka mujal) – `trap`
- `--` argumentide eraldamiseks
- `cd` kindlasse kohta
- Piiratud võimalustega käsinterpreteraator (*restricted shell*) – ai-nult lisameetmeks

# SQL

- Andmebaasipäringute keel, tihti kasutusel ka veebirakendustes
- Oma metamärgid, mis tuleb välja filtreerida
- Näide:  

```
"SELECT password FROM people  
WHERE name= ' "+$name+" ' "
```

Mis saab, kui  

```
$name="a' ; DROP TABLE people;" ?
```
- ```
"SELECT name FROM people where ID="+$ID  
$ID="3 OR 1=1"
```
- Ohtlikke sümboleid: `' " / \ * # % & ( ) , : ; |`  
...ja mitte ainult – lubage ainult tähti ja numbreid, kui võimalik
- Tabelitele minimaalsed õigused

# PHP

- Võimas, laialdaste sisseehitatud vahenditega
- Lihtne skriptikeel
- Lihtne programmeerima hakata

## AGA:

- Turvalisusele on (vähemalt algselt) vähe mõeldud
- Keeles endas palju võimalusi „jalga tulistada“
- Mitmed võimalused pole turvaliselt kasutatavad
- Jätab petliku mulje, et süsteemi tuntakse
- Selle tulemusena kasutab PHP-d palju oskamatuid programmeerijaid

## PHP globaalsed muutujad

- Suur üldine globaalne nimeruum:
  - \* Globaalsed muutjad
  - \* Keskkonnamuutujad
  - \* Kliendilt tagastatud vormielemendid!!!
  - \* Funktsioonides õnneks vaikimisi näha pole
- Muutujate automaatne algväärtustamine
- Lõtv tüüpimine

⇒ `register_globals = off`,  
modulaarne programmeerimine

## Aukliku PHP koodi näide

```
<?php
    if ($pass == "hello")
        $auth = 1;
    ...
    if ($auth == 1)
        echo "Eriti salajane info";
?>
```

## Veel PHP probleeme

- `include`, `fopen` jms failifunktsioonid tunnistavad failinimedeks URL'e – ka suvalisest masinast HTTP-ga
- $\Rightarrow$  `allow_url_fopen = off`,  
`open_basedir = ...`
- Kogu väliste programmide käivitamine käib shelli kaudu:  
`system()`, `exec()`, `popen()`, `passthru()`, ``...`` –  
seega on shelli erisümbolid ohtlikud:  
`& ; ` \ " | * ? ~ ^ <> () [] {} $ \n \r`
- `magic_quotes_*`, `escapeshellarg()` ja  
`escapeshellcmd()` – selle vastu, aga enne kasutamist  
endale täpselt selgeks teha
- `addslashes()` ja `addcslashes()` – andmebaasi minevate  
andmete ohutuks tegemiseks

## PHP: veel

- Failide upload
  - \* Kõigepealt salvestatakse failid serverisse ja alles hiljem saab PHP skript neid valideerima hakata
  - \* Varem anti failinimi edasi 4 muutujaga, see on ohtlik
  - \* Uuemal ajal spetsiaalne massiiv `HTTP_POST_FILES` – kasutage seda koos lisakontrollidega
  - \* Kanal failide masinasse sokutamiseks, et neid hiljem muu nõksuga täitma saada
- Mitte jätta include-faile (\*.inc tihti) kättesaadavale!
- Teegifailide kasutamisel ärge tekitage täiendavaid sisenemispunkte (\*.php)
- Igasugused teegifailid on kasulik üldse veebipuu alt välja tõsta