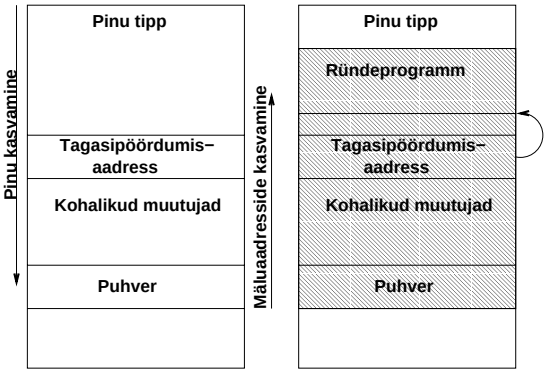
Buffer Overflow

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

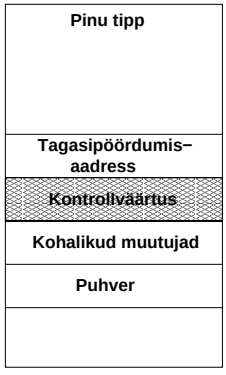
Pinu ja puhvrid



Puhvri ületäitumiste kaitse

- Päris kaitse
  - Kontrollime kõigi puhvrite pikkusi
  - Ei kasuta ebaturvalisi funktsioone, mis lasevad üle puhvri otsa kirjutada
- Mälukaitse (mittetäidetav pinu, . . .)
- Pinusse kontrollväärtuste lisamine
  - Randomiseerimine
  - Nullbaidi sisalduvus
- Teegid pinuviidaga arvestavate variantidega ohtlikest funktsioonidest (*libsafe, libverify, . . .*)
- Ükski neist hädavahenditest ei anna täielikku kaitset

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`

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
- ⇒ saame kirjutada mälus peaaegu suvalisele aadressile etteantud värtusi