46

Kasvata hännästä demoni

`tail`-ohjelmalla (tail = häntä) voidaan helpolla tehdä jatkuvasti toimiva taustaohjelma (demoni), joka käsittelee jotakin jatkuvasti kasvavaa tekstitiedostoa. Tälläisia kasvavia tekstitiedostoja ovat esimerkiksi järjestelmän lokitiedostot. Niihin kertyy järjestelmän toiminnasta ja tilasta kertovaa tietoa.

Ennen kuin otamme esimerkin käsittelyyn, tarkastellaan hieman joitakin `tail`-komennon parametrejä:

- **-f, --follow[={name|descriptor}]** `tail` seuraa tiedoston loppua ja tulostaa uudet rivit sitä mukaa, kun niitä kirjoitetaan tiedostoon. Oletus on seurata tiedostokahvaa (descriptor), jolloin jos tiedosto siirretään eli nimetään uudelleen (logrotate tekee tämän normaaliasetuksilla kerran viikossa kaikille lokitiedostoille), ei uutta tiedostoa enää seurata. Oletus on järkevä silloin, kun tiedetään, että tiedostoa ei siirretä. Jos käytetään muotoa `--follow=name`, `tail` tarkistaa ajoittain, että tiedostokahva vastaa tiedoston nimeä ja jos se on muuttunut, `tail` siirtyy seuraamaan uutta samannimistä tiedostoa.
- **-n, --lines=N** `tail` lukee `N` riviä tiedoston lopusta. Jos `N` on 0, `tail` hyppää suoraan tiedoston loppuun ja jos lisäksi on käytetty `-f`-parametriä, `tail` tulostaa vain käynnistyksensä jälkeen kirjoitetut rivit.

Lisäksi `tail`-ohjelmalle pitää antaa tiedoston nimi, jota käsitellään. Tiedostoja voi olla myös useita ja vaikka `tail`-ohjelman man-sivulla väitetään, että `tail` tulostaa tiedoston nimen,

kun tiedostoja on useita, ei se sitä tee, kun tulostus menee putkeen.

Useiden tiedostojen käsittely on kätevä ominaisuus esimerkiksi silloin, kun halutaan käsitellä useen palvelinohjelman lokitiedostoa. Käytännössä tämä tilanne voi tulla vastaan esimerkiksi, kun jokin palvelu on jaettu usealle koneelle vikasietoisuuden tai kapasiteetin takia.

46.1 Esimerkki – Nimda-madon esto Apachelle

Syyskuussa 2001 maailmalla levisi Microsoftin IIS-palvelimiin nimda-mato. Se yritti tarttua myös Linux-koneisiin, mutta ei tietenkään onnistunut. Yrityksistä oli kuitenkin se haitta, että nimda oli sitkeä ja yksi saastunut kone lähetti yhteensä 384 sivupyyntöä. Tämä paisutti Apachen lokia ja aiheutti turhaa liikennettä.

Ongelman ratkaisuun on monta vaihtoehtoa ja eräs aivan hyvä oli olla välittämättä ongelmasta, koska sen merkitys oli mitätön. Perusteellinen ratkaisu olisi tarkkailla IP-pakettien sisältöä palomuurissa tai tarkkailla Apachelle tulevia sivupyyntöjä Apachen moduulilla. Näillä tavoilla voitaisiin ongelma poistaa hyvin aikaisessa vaiheessa.

Tämän esimerkin ratkaisu perustuu siihen, että sivupyyntö pääsee Apachelle asti, siitä tulee lokimerkintä ja vasta siihen reagoidaan. Esimerkkinä ratkaisu on hyvä, sillä se esittelee hyvin yleispätevän tavan käyttää lokitiedostoja demonien luontiin.

```
#!/bin/sh

# Missä Apachen loki luuraa
cd /var/log/httpd
# Varmistetaan tiedostojen olemassaolo
touch nimda.txt
touch codered.txt
# Tapetaan vanha ajossa oleva versio
kill ‘ps -C "tail -n100" h | cut -f2 -d" "‘

# Muuta -n suuremmaksi, jos on tarvetta.
tail -n10000 --follow=name access_log | \
  awk ’
# Pyydystetään nimda-mato
  /scripts\/root.exe/ {
  vanha=0
  while(getline rivi < "nimda.txt"){if($1 ~ rivi) vanha=1}
  if(vanha==0){
# Matopesän nimi talteen
  print $1 >> "nimda.txt"
  system("/sbin/ipchains -A input -j DENY  -i eth0 -p tcp -s " $1 " --destination-port 80")
  }
  close("nimda.txt")
  }
# Pyydystetään Code Red -mato
  /default.ida/ {
```

```
  vanha=0
  while(getline rivi < "codered.txt"){if($1 ~ rivi) vanha=1}
  if(vanha==0){
# Matopesän nimi talteen
  print $1 >> "codered.txt"
  system("/sbin/ipchains -A input -j DENY  -i eth0 -p tcp -s " $1 " --destination-port 80")
  }
  close("codered.txt")
  }' - &
```

Skriptin alussa tehdään alustavia toimenpiteitä, joista mielenkiintoisin on `kill`-rivi. Se tappaa skriptin mahdollisen edellisen käynnistyskerran jäljiltä toiminnassa olevan `tail`-ohjelman. Edelliseltä käynnistyskerralta olisi toiminnassa myös `awk`, mutta se kuolee, kun sitä putken kautta syöttävä `tail` kuolee.

Varsinaisen työn tekevä osuus voidaan yksinkertaistaa: "`tail ... | awk '{skripti}' - &`". `tail` siis syöttää putkeen jatkuvasti uusia rivejä tiedostosta sitä mukaa, kun niitä tulee ja putken toisessa päässä `awk` käsittelee niitä. Merkintä – tarkoittaa, että putken syöte luetaan siinä kohtaa. Koko komeus laitetaan tausta-ajoon `&`-merkillä.

Skripti voidaan käynnistää sellaisenaan konsolilta, mutta jos ollaan `ssh`:lla etäyhteydessä WWW-palvelinkoneeseen, skripti on käynnistettävä komennolla "`nohup ./skripti`". `nohup` ohjaa parametrinään olleen ohjelman tulosteen tiedostoon `nohup.out` ja ohjelma jatkaa toimintaansa vaikka ssh-yhteys katkeaisi. Kolmas vaihtoehto on laittaa käynnistys koneen käynnistysskripteihin, esimerkiksi tiedostoon `/etc/rc.d/rc.local`.

Näin helppoa on oman demonin luominen. `tail` huolehtii useasta hankalasta yksityiskohdasta, kuten "nukkumisesta" työn teon välillä ja tiedostojen käsittelystä.

46.2 Vastaava ongelma ja sen erilaisia ratkaisuja

Vuoden 2005 aikana alkoi ilmestyä Internetissä olevien palvelimien lokeihin ilmoituksia useista epäonnistuneista kirjautumisyrityksistä `ssh`:lla. Niitä saattoi olla satoja, jopa tuhansia vuorokaudessa. Jos järjestelmään on asennettu paketti `logwatch`, tulee pääkäyttäjälle seuraavan tapaisia tiivistelmiä lokitiedostojen epätavallisista tapahtumista:

```
 --------------------- pam_unix Begin ------------------------

sshd:
   Invalid Users:
      Unknown Account: 1020 Time(s)
   Authentication Failures:
      test (85.234.136.215 ): 1 Time(s)
      mysql (85.234.136.215 ): 1 Time(s)
      unknown (221.8.13.200 ): 2 Time(s)
```

```
      mail (85.234.136.215 ): 1 Time(s)
      unknown (roliveira.por.ulusiada.pt ): 562 Time(s)
      postgres (srv1.ipqtecnologia.com.br ): 1 Time(s)
      games (85.234.136.215 ): 1 Time(s)
      postgres (85.234.136.215 ): 1 Time(s)
      bin (srv1.ipqtecnologia.com.br ): 1 Time(s)
      root (85.234.136.215 ): 1 Time(s)
      unknown (85.234.136.215 ): 103 Time(s)
.....
 ---------------------- SSHD Begin ------------------------


Failed logins from these:
   Victor/password from 193.136.185.21: 1 Time(s)
   a/password from 200.223.129.101: 1 Time(s)
   a/password from 221.8.13.200: 1 Time(s)
   abba/password from 85.234.136.215: 1 Time(s)
   abby/password from 85.234.136.215: 1 Time(s)
   abcd/password from 200.223.129.101: 1 Time(s)
   accept/password from 85.234.136.215: 1 Time(s)
   acer/password from 85.234.136.215: 1 Time(s)
   ad/password from 200.223.129.101: 1 Time(s)
   adm/password from 200.223.129.101: 1 Time(s)
   administracion/password from 200.223.129.101: 2 Time(s)
   administration/password from 200.223.129.101: 2 Time(s)
```

Yllä on vain katkelmia murtoyritysten jättämistä jäljistä. Murtautujat käyttävät listaa tavanomaisista käyttäjätunnuksista ja salasanatkin arvataan. Koska jo olemattomaan käyttäjätunnukseen kirjautuminen on mahdotonta, ei murron onnistuminen ole kuin etäisesti mahdollista. Silti yrityksetkin ovat kiusallisia ja niihin varautuminen antaa kuvan tiukasta tietoturvasta.

Vaivaan on kehitetty monia ratkaisuja, joita kuvataan esimerkiksi artikkelissa Protecting Linux against automated attackers (`http://security.linux.com/article.pl?sid=05/09/15/1655234&tid=35`). Artikkelissa kuvataan peräti kolme ohjelmaa, jotka käyttävät hyvin samanlaista perustekniikkaa kuin edellä tekemämme demoni. Niissä on paljon enemmän säätöjä, mutta äskeinen esimerkki onkin enemmänkin esimerkki siitä, miten pienellä vaivalla voi itsekin tehdä demonin omaan hätäiseen tarpeeseen.

46.2.1 Palomuurin asetusten muuttaminen

Parempi tekniikka olisi käyttää suoraan palomuurin sääntöjä. Stephen Frost on tehnyt `iptables`:lle laajennoksen nimeltä IPTables/Netfilter Recent Module, joka löytyy osoitteesta

http://snowman.net/projects/ipt_recent/. Sen ideana on se, että ipt_recent muistaa edelliset yhteydet ja tiheästi toistuvat yritykset johtavat pakettien tiputtamiseen. Esimerkiksi palomuurisääntö:

```
iptables -A INPUT -p tcp --dport ssh -m recent --update --seconds 60 -j DROP
```

sallii vain yhden uuden ssh-yhteyden 60 sekunnin välein. Jos murtautuja yrittää tiuhempaan, kaikki yhteydet ensimmäisen jälkeen estetään eli yrityskertojen välillä pitää olla 60 sekunnin tauko.

46.2.2 Autentikoinnin asetusten muuttaminen

Toinen parempi tekniikka on käyttää autentikoinnin laajennosta nimeltä pam_abl (Auto Black-List Module), jonka kotisivu on osoitteessa http://www.hexten.net/pam_abl/. Valmiin rpm-paketin voi ladata osoitteesta http://www.lineox.net/3.0/i386/extras/pam_abl-0.2.2-2.i386.rpm. Sen käyttöönotto vaatii tiedoston /etc/pam.d/system-auth muokkaamista. Alla olevassa listauksessa lisätty rivi on toisena:

```
auth        required      /lib/security/pam_env.so
auth required /lib/security/pam_abl.so config=/etc/security/pam_abl.conf

auth        sufficient    /lib/security/pam_unix.so likeauth nullok
```

Tämän lisäksi haluat ehkä muokata myös tiedostoa /etc/security/pam_abl.conf. Oletuksena sen sisältö on:

```
# /etc/security/pam_abl.conf
# debug
host_db=/var/lib/abl/hosts.db
host_purge=2d
host_rule=*:10/1h,30/1d
user_db=/var/lib/abl/users.db
user_purge=2d
user_rule=!root:10/1h,30/1d
```

Ensimmäinen rivi määrittää tietokantatiedoston sijainnin, toinen säilyttää tiedot koneista, joista on yritetty sisään kaksi vuorokautta, kolmas, että kustakin koneesta voidaan yrittää korkeintaan 10 kertaa tunnissa ja 30 kertaa vuorokaudessa. Onnistuneet sisäänkirjautumiset eivät kuitenkaan "syö" yrityskertoja. Kolme viimeistä riviä asettaa vastaavat rajat käyttäjätunnuskohtaisesti, mutta ei pääkäyttäjän tunnukselle.

Aivan vastaavaa tekniikkaa käytetään verkkopankeissa. Esimerkiksi Nordean verkkopankissa yrityskertoja on sallittu viisi ja seuraavana päivänä voi yrittää uudelleen.