Luku III: Käyttöjärjestelmien turvallisuus

Tässä luvussa käsittelemme

- prosessien turvallisuutta,
- muistin ja tiedostojärjestelmien turvallisuutta ja
- sovellustason ohjelmien turvallisuutta.

Esitietoina oletetaan käyttöjärjestelmien perusteet eli prosessin käsite ja prosessien hallinta, virtuaalimuisti, tiedostojärjestelmä, kääntäminen ja linkitys.

Prosessien turvallisuus: Käynnistysjärjestys I

- Kun tietokone käynnistetään, KJ:ää aletaan ladata muistiin.
- Ensimmäiseksi suoritetaan BIOS-järjestelmässä (Basic Input/Output System) oleva koodi (firmware).
- BIOS lataa muistiin toisen vaiheen käynnistyslataajan (second-stage boot loader), jonka tehtäviin kuuluu muiden KJ:n osien lataaminen muistiin. Sen jälkeen ohjaus siirtyy käyttöjärjestelmälle.
- Hyökkääjä voisi yrittää kaapata koneen käynnistyksen monessa vaiheessa. Jotta hyökkääjä ei pääsisi edes aloittamaan käynnistystä, monissa järjestelmissä käytetään BIOS-salasanaa, jota ilman toista vaihetta ei pääse aloittamaan.

Käynnistysmedioiden hierarkia I

- Käyttäjä voi monissa järjestelmissä määritellä, miltä medialta toisen vaiheen käynnistys aloitetaan: kovalevy, DVD, USB.
- Käynnistys tulisi aina tehdä luotettavalta medialta.
- Voidaan määritellä hierarkia, jonka mukaan käynnistykseen sopiva media etsitään. Tämä helpottaa asennuksia ja virhetilanteiden selvittelyä, mutta antaa mahdollisuuksia hyökkääjälle.
- Monissa tietokoneissa on KJ:n toisen vaiheen lataaja, joka vaatii salasanan myös ulkoisilta medioilta ladattaessa.

Virransäästötila I

- Kun siirtyy virransäästötilaan (hibernation), KJ tallettaa koko keskusmuistin sisällön levylle, josta se voidaan ladata nopeasti keskusmuistiin.
- Myös kaikki salasanat yms sensitiivinen tieto viedään levylle.
- Tällöin hyökkääjä voi yrittää CD-hyökkäystä (live CD attack, kts. myöhemmin).
- Windows tallettaa keskusmuistin `C:\hiberfil.sys`-tiedostoon. On onnistuttu kehittämään (reverse engineering) algoritmi, jonka avulla tiedoston sisältämästä tiivistetystä tiedosta voidaan palauttaa muistin sisältö.
- On myös hyökkäyksiä, joissa `hiberfil.sys`-tiedostoa on muutettu.
- Lisäksi Windows ei poista kyseistä tiedostoa sen jälkeen, kun kone on taas normaalissa tilassa.
- Kovalevyn salaus estää nämä hyökkäykset.

Monitorointi, hallinta ja kirjanpito I

- On tärkeää olla tietoinen, mitä kone on tekemässä (military term: situational awaress):
  - mitkä prosessit ovat toiminnassa,
  - minkä koneiden kanssa kommunikoidaan,
  - epätavalliset tapahtumat (kuten jatkuvasti epäonnistuvat sisäänkirjautumisyritykset).

Tapahtumien kirjaus I

- Windowsissa on kolmenlaisia kirjauksia:
  - system log: KJ:n toiminta;
  - application log: sovellusten toiminta;
  - security log: esimerkiksi tiedot todennuksista.
- Ainoastaan `lsass.exe` eli Local Security Authority Subsystem Service voi kirjoittaa security-lokiin. Palvelun tavoitteena on pakottaa noudattamaan turvapolitiikkaa kuten pääsynvalvontaa ja todennusta.
- Käyttäjät voivat määritellä myös omia lokeja.
- Unix-pohjaisilla järjestelmillä on erilaisia kirjausmekanismeja riippuen versiosta.
- Tyypillisesti lokitiedostot sijaitsevat hakemistossa `/var/log` tai vastaavassa. Ne ovat tavallisesti tekstitiedostoja.
- Esimerkiksi `auth.log` sisältää käyttäjien todennukseen liittyviä tietoja, `kern.log` taas odottamattomia ytimen toimintoja. Näihin tiedostoihin voi kirjoittaa vain `syslog` demoni.

Tapahtumien kirjaus II

- Windowsin lokitiedostot eivät ole tekstitiedostoja ja niitä on helpompi käsitellä Microsoftin tapahtumakirjaus-välineillä (event logging tools). Unixin lokitiedostot ovat heti sellaisenaan katsottavissa.

Monitorointi I

- On useita tilanteita, joissa halutaan tietää, mitä koneessa oikein tapahtuu. Esim. kone hidastuu tai muistia käytetään runsaasti.
- Jokaisessa KJ:ssä on välineet tarkistaa käynnissä olevat prosessit: Task manager Windowsissa ja ps-, top-, pstree- ja kill-komento Linuxissa.

Process Explorer I

- Process Explorer on Windowsin ohjelmisto, joka on hyödyllinen monitoroitaessa prosesseja.
- Vasen sarake esittää prosessipuun eli prosessit ja niiden isä-lapsi -suhteen. Esim. `explorer.exe` on sekä Firefoxin että Thunderbirdin isä.
- Seuraavat sarakkeet ovat: prosessin ID, prosenttiosuus CPU:sta, osoiteavaruuden koko ja prosessin kuvaus.
- Jos CPU-aikaa kuluu paljon tai jos virtuaalinen osoiteavaruus on suuri, niin usein kysymyksessä on tilanne, jolloin prosessi täytyy lopettaa.
- Voidaan määritellä myös värejä, joiden avulla eri prosessiryhmiä voidaan paremmin erottaa toisistaan.
- On myös tärkeää kerätä tietoja prosessin alkuperästä (process image). Esimerkiksi ohjelman luoja ja sijainti levyllä.

Process Explorer II

- Esim. sijainti saattaa paljastaa viruksen, jonka tiedostonimi on sama kuin sallitun ohjelman, mutta joka sijaitsee epätavallisessa paikassa.
- Hyökääjä voi myös yrittää korvata ohjelmatiedoston modifioidulla versiolla, joka tekee jotain haitallista tai epätoivottavaa. Tällaisten muutosten estämiseksi ohjelmistokehittäjä voi allekirjoittaa digitaalisesti ohjelmatiedoston. Process Explorer voi tällöin verifioida allekirjoituksen ja näyttää osapuolen, joka on tehnyt allekirjoituksen.

Muistin ja tiedostojärjestelmän turvallisuus I

- Windowsissa virtuaalimuistin sivut sijaitsevat levyllä paikassa `C:\pagefile.sys`.
- Linux vaatii sensijaan, että levylle alustetaan erityinen osio, swap partition, jonne virtuaalimuistin sivut talletetaan. Linuxissa voi olla myös vaihtoehtoisesti ns. swap file, joka toimii samoin kuin Windowsissa.
- Kumpikin KJ estää virtuaalimuistin sivujen katselun ajoaikana. Voidaan myös konfiguroida KJ siten, että se poistaa sivut levyltä, kun kone sammutetaan.
- Jos kuitenkin hyökkääjä sulkee koneen nopeasti ja käynnistää toisen KJ:n ulkoiselta medialta, niin silloin on mahdollista katsella näitä tiedostoja ja ainakin osittain konstruoida muistin sisältö.
- Tällaisten hyökkäysten estämiseksi levy tulisi salakirjoittaa.

Salasanatiedosto ja suolaus I

- On aina mahdollista, että kaksi käyttäjää valitsee saman salasanan. Jos esimerkiksi Aapo ja Bertil molemmat valitsevat salasanan "aprilli" ja jos Bertil pääsee katselemaan salasanatiedostoa, hän huomaa, että Aapolla on sama salasana.
- Unix ratkaisee tämän ongelman laajentamalla salasanaa ns. suolalla.
- Alunalkaen suola oli 12-bittinen luku, joka muodostettiin systeemin ajasta ja prosessin tunnuksesta. Siten suola on todennäköisesti yksikäsitteinen jokaisella käyttäjällä.
- Suola liitetään käyttäjän salasanaan, kun salasana valitaan. Jos Bertilin salasana on p, niin salasanatiedostoon viedään $Hash(p||salt_B)$, eli tiiviste alkuperäisestä salasanasta lisättynä suolalla.
- Lisäksi suola talletetaan Bertilin tunnuksen ja muutetetun salasanan yhteyteen.

Salasanatiedosto ja suolaus II

- Vanhoissa Unix-järjestelmissä suola oli 12-bittinen. Tällaiset salasanatiedostot voidaan nykyään murtaa sateenkaaritaulujen avulla.
- Linuxissa suola on nykyään 48 ja Solariksessa 128 bittiä. Tällaiset pituudet estävät ennalta lasketut hyökkäykset pitkälle tulevaisuuteen.
- Windows NT/2000:teen kuuluvat LAN Manager NT LAN Manager eivät käyttäneet suolaa, mikä tekikin niistä suosittuja hyökkäyskohteita näille menetelmille.

Salasanoihin perustuva todennus Windowssissa ja Linuxissa I

- Windowsin salasanojen tiivisteet ovat SAM-tiedostossa (Security Accounts Manager). Tiedosto ei ole tavallisten käyttäjien käytössä KJ:n ollessa aktiivinen.
- Vanhemmat Windowsin versiot käyttivät DES:iin perustuvaa tiivistetekniikkaa, jota kutsutaan nimellä LAN Manager hash tai LM hash. Se etenee seuraavasti:
  1. Salasanan pienet kirjaimet muutetaan isoiksi.
  2. Salasana kasvatetaan 14 merkkiin lisäämällä null-merkkejä.
  3. Salasana pilkotaan kahteen 7 tavun osaan.
  4. Näistä puoliskoista muodostetaan kaksi DES-avainta á 64 bittiä lisäämällä nollabitti jokaisen 7 bitin jonon jälkeen. Näin saadaan 64 bitin avaimet.

Salasanoihin perustuva todennus Windowssissa ja Linuxissa II

5. Merkkijono "KGS!#$%" salataan erikseen näillä avaimilla. Tuloksena on kaksi 8 tavun salatekstiä. DES:iä käytetään ECB-moodissa ilman täytettä (padding).
6. Kaksi salatekstiä yhdistetään yhdeksi 16 tavun jonoksi, LM-tiivisteeksi.

- Heikkouksia: Käytetään pelkkiä isoja kirjaimia. Suolaa ei käytetä. Salasana jaetaan kahteen lyhyempään osaan, mikä helpottaa kokeiluja. Pass the hash -hyökkäys on mahdollinen (kts. myöhemmin).
- Seuraavaksi Windowsissa otettiin käyttöön NTLM-algoritmi. Siitä on kaksi versiota, NTLMv1 ja NTLMv2. Kumpikin on haaste-vastaus -protokolla (challenge-response). Palvelin lähettää 8-tavuisen satunnaisen bittijonon, haaasteen, asiakkaalle, joka laskee siitä kaksi vastausta käyttämällä tiivistefunktioita ja salasanaansa. Laskut ovat monimutkaisempia kuin LM:ssä. Menetelmä on kuvattu dokumentissa RFC1320.

Salasanoihin perustuva todennus Windowssissa ja Linuxissa III

- Menetelmässä on samoja puutteita kuin LM:ssä: suolaa ei käytetä, ei nykyaikaisia vahvoja salauksia AES:ää tms, tiivistefunktiot MD4 ja MD5 eivät ole vahvimpia.
- Microsoft ei enää suosittele NTLM:n käyttöä sovelluksissa. Dominique Brezinski löysi kaksi hyökkäystä 1997. Vuonna 2010 Amplia Security löysi lukuisia aukkoja NTLM:n toteutuksessa Windowsissa. Nämä löydökset mursivat NTLM:n täydellisesti. Yksi aukko liittyi satunnaislukujen ennustettavaan generointiin.Nämä aukot olivat olleet toteutuksissa 17 vuotta. MS10-012 korjasi virheet helmikuussa 2010.
- Kerberos on korvannut NTLM:n todennusmekanismina. Kerberos on tyypillinen, tosin melko monimutkainen, todennusprotokolla, jota käsitellään tietoturvan jatkokurssilla. NTLM jatkaa kuitenkin olemassaoloaan monissa tilanteissa.

Salasanoihin perustuva todennus Windowssissa ja Linuxissa IV

- Unix-pohjaiset järjestelmät käyttävät samatyyppistä menetelmää kuin NTLM. Todennustieto tallennetaan tiedostoon `at/etc/passwd`. Unix käyttää kuitenkin suolaa eikä pakota käyttämään tiettyä tiivistefunktiota. Usein käytetään MD5:tä suolan kanssa tai DES:n muunnelmia.

Pääsynvalvonta ja kehittyneet tiedosto-oikeudet I

- Pääsyoikeuksia voidaan määritellä yksittäiselle käyttäjälle tai ryhmälle.
- Ryhmä voi olla eksplisiittisesti määritelty yksittäisten käyttäjien ryhmä tai se voi olla KJ:n etukäteen määrittelemä. Esim. Unixissa on *owning group*, joka oletusarvoisesti liitetään tiedostoon. Ryhmä *all* käsittää kaikki, ryhmä *other* kaikki paitsi tiedoston omistajan.
- Tiedoston pääsynvalvontatietue on kolmikko *(käyttäjä, tyyppi, oikeus)*, missä käyttäjä on yksittäinen käyttäjä tai ryhmä ja tyyppi on joko "salli" tai "kiellä".
- *Pääsynvalvontalista* on lista pääsynvalvontatietueita.
- On monia yksityiskohtia, kun suunnitellaan pääsynvalvonnan toteutusta KJ:ssä. Esimerkiksi:
  - Periytyvätkö tiedot hakemistosta tiedostoon?
  - Mitä tapahtuu, jos käyttäjällä on oikeus kirjoittaa tiedostoon, muttei hakemistoon?

Pääsynvalvonta ja kehittyneet tiedosto-oikeudet II

- Luku-, kirjoitus- ja suoritusoikeudet vaikuttavat selviltä, kun kysymys on tiedostoista, mutta entä hakemistojen kohdalla?
- Jos oikeuksia ei anneta eikä kielletä, miten ne pitää asettaa oletusarvoisesti?

- Esim. Linux ja Windows eroavat suuresti toisistaan ratkaisuissaan.

Oikeudet Linuxissa I

- Käytössä on matriisit, joiden avulla esitetään käyttäjät ja heidän oikeutensa.
- Kaikki, mikä ei ole erikseen sallittu, on kielletty. Siten ei tarvita kieltokäskyjä.
- Jotta voisi lukea tiedoston, hakemistopolun jokaiseen hakemistoon täytyy olla suoritusoikeus ja tiedostoon lukuoikeus.
- Tiedoston omistajat voivat muuttaa tiedoston oikeuksia. Kysymyksessä on siis yksilöpohjainen pääsynvalvonta (discretionary access control).
- Perusoikeuksien lisäksi tiedostolla voi olla laajennettuja attribuutteja. Esim. tiedostosta voidaan tehdä "append only-tyyppinen, jolloin kirjooittaa voi vain tiedoston loppuun. Toinen esim. on "immutable", jolloin edes pääkäyttäjä eli juuri ei voi poistaa tiedostoa ellei poista ensin attribuuttia.

Oikeudet Linuxissa II

- Attribuutteja voidaan katsoa ja asettaa lsattr- ja chattr-käskyillä.
- Linux on alkanut tukea myös vaihtoehtoista pääsynvalvontalistamekanismia (ACL).
- Pääsylistoja voidaan katsella getfacl-käskyllä ja asettaa setfacl-käskyllä.
- Jokaisella tiedostolla on pääsylista käyttäjille owner, group ja other.
- On mahdollista luoda myös lisälistoja erityisiä käyttäjiä tai ryhmiä varten (named users, groups).
- Käytössä on myös ns. *valvontatietuemaski* (mask ACE), joka määrittelee maksimaaliset oikeudet, jotka nimetyt käyttäjät voivat saada.
- Joissakin Linux-versioissa on rajoittavampia mekanismeja. Esim. NSA:n kehittämä SELinux (Security Enhanced Linux) soveltaa sääntöpohjaista (rule-based tai mandatory) pääsynvalvontaa, joka määrittelee kaikki sallitut toimenpiteet. Omistajallakaan ei ole mahdollisuuksia muuttaa tiedostonsa oikeuksia.

Oikeudet Windowsissa I

- Käytössä on pääsynvalvontalistat.
- Käyttäjät voivat luoda sääntöjoukkoja kullekin käyttäjälle tai ryhmälle.
- Säännöt joko sallivat tai kieltävät. Jos ei ole sääntöä, oletusarvoisesti kielletään.
- Perusoikeudet eli Windowsin termilogialla standardit oikeudet tiedostoon ovat *modify, reand and execute, read, write, full control*. Viimeksi mainittu antaa kaikki oikeudet.
- On myös edistyneitä (advanced) oikeuksia, joista tavalliset oikeudet kootaan. Esim. tavallinen read-oikeus käsittää seuraavat edistyneet oikeudet: *read data, read attributes, read extended attributes, read permissions*.
- Hakemistoilla on myös oikeudet. Read sallii listata hakemiston sisällön, write sallii luoda uusia tiedostoja.

Oikeudet Windowsissa II

- Linux tarkistaa kaikissa hakupolun solmuissa, onko käyttäjällä oikeus suorittaa hakemisto. Windows ei tarkista tätä. Siten on mahdollista käsitellä tiedostoa, vaikkei ole mitään oikeuksia hakemistoon. Tämä ei onnistu Linuxissa.
- Hakemiston oikeudet voidaan asettaa periytymään alihakemistoille. Periytyminen voidaan keskeyttää joissain alihakemistossa niin, etteivät oikeudet enää periydy alaspäin.
- Periytymisen takia tarvitaan säännöt, missä järjestyksessä oikeuksia sovelletaan. deny on enne allow:ta, eksplisiittinen oikeus ennen perittyä. Perittyjä oikeuksia sovelletaan etäisyyden mukaan: vanhempi peritään ennen isovanhempaa.
- Windowsin ratkaisu on joustava, mutta se altistaa monimutkaisuutensa takia myös turvallisuutta vaarantaville virheille.

Käyttäjien tunnukset I

- Tarkastellaan UNIX-järjestelmää. Käyttäjän identiteettiä esittää kokonaisluku, joka on 0:n ja jonkin suuren luvun (esim. 65 535) välissä. Tätä lukua kutsutaan UID:ksi (user identification number). Lisäksi käyttäjällä voi olla login-nimi. Jokaista login-nimeä vastaa tasan yksi UID, mutta yhdellä UID:lla voi olla monta login-nimeä.
- Kun käyttöjärjestelmän ydin käsittelee käyttäjän identiteettiä, se käyttää UID:ta. Kun taas käyttäjä kirjautuu sisään koneeseen, hän käyttää login-nimeään. Yhdellä käyttäjällä voi olla monta eri identiteettiä. Tyypillisesti identiteetti vastaa jotakin toiminnallisuutta.
- UNIX-versiot käyttävät usean tyyppisiä käyttäjän identiteettejä. Koska käyttäjät käynnistävät prosesseja, nämä eri identiteetit liittyvät prosesseihin.
- Todellinen UID (real UID) on käyttäjän alkuperäinen identiteetti, kun hän kirjautuu koneelle.

Käyttäjien tunnukset II

- Efektiivinen UID (effective UID) on identiteetti, jota käytetään pääsynvalvonnassa. Esimerkiksi jos vain UID 22 voi lukea tiettyä tiedostoa ja prosessin todellinen UID on 22 ja efektiivinen UID 35, niin prosessi ei voi lukea tiedostoa. Jos taas prosessin todellinen UID olisi 35, mutta efektiivinen 22, prosessi voi lukea tiedostoa.
- Systeemiohjelmat *setuid* luovat prosesseja, joiden efektiivinen UID on sama kuin ohjelman omistajan eikä sama kuin ohjelman suorittajan. Tällöin prosessin pääsyoikeudet ovat samat kuin ohjelman omistajan eikä ohjelman suorittajan.
- Monet UNIX-versiot tarjoavat myös talletetun UID:n (saved UID). Aina kun efektiivinen UID vaihtuu, talletetun UID:n arvoksi viedään ennen vaihtoa voimassa ollut efektiivinen UID. Käyttäjä voi käyttää kaikkia kolmea UID:ta. Tämä sallii, että käyttäjälle annetaan juurioikeudet joksikin lyhyeksi ajaksi, jonka jälkeen niistä luovutaan, mutta joihin voidaan palata myöhemmin (tallennetun UID:n avulla).

Käyttäjien tunnukset III

- Traditionaalisesti todellista UID:ta käytettiin prosessin alkuperäisen UID:n jäljittämiseen. Kuitenkin juurioikeuksien haltija voi muuttaa sitä. Jotta turvattaisiin alkuperäisen todellisen UID:n seuranta, monet UNIX-järjestelmät tarjoavat vielä neljättä UID-versiota, audit- tai login-UID. Tämä UID annetaan kirjautumisen yhteydessä, eikä sitä voi vaihtaa. (Kuitenkin jotkut systeemit sallivat juuren muuttaa audit-UID:ta!)

Esimerkki. I

- Salasanatiedosto kuuluu juurioikeuksien haltijalle, joten vain hän voi periaatteessa muokata tiedostoa.
- Kuitenkin tavallinen käyttäjä voi muuttaa salasanaansa. Hän siis muokkaa salasanatiedostoa.
- Käyttäjä käynnistää prosessin, joka muuttaa salasanatiedostoa. Jos salasanatiedoston setuid-bitti on 1, niin tiedoston omistaja (siis juuri) tulee tuon prosessin efektiiviseksi omistajaksi.
- Eli salasanatiedostoa muuttavan prosessin todellinen UID on käyttäjän, mutta efektiivinen UID on juurioikeuksien haltijan. Täten käyttäjä onnistuu muuttamaan salasanatiedostoa.
- Yleensä on tietenkin riskialtista asettaa setuid-bitti juurioikeuksien haltijan mukaan, mutta tässä tapauksessa se on välttämätöntä. Salasanatiedostoa pitää sen vuoksi suojella erityisen hyvin. □

Tiedostot I

- Tarkastelemme vielä tiedostojen ja käyttäjätunnusten käsittelyä pääsynvalvonnan kannalta. Näissä tarkasteluissa keskitytään UNIX-käyttöjärjestelmään, koska se on jo melko yleinen palvelimissa.
- Paikalliset systeemit identifioivat objekteja antamalla niille nimen. Nimi voi olla tarkoitettu ihmisen, prosessin tai käyttöjärjestelmän ytimen käyttöön. Jokaisella nimellä voi olla eri semantiikka.
- UNIX tarjoaa neljää erilaista tiedostonimityyppiä. Inode identifioi tiedoston yksikäsitteisesti. Se sisältää informaatiota tiedostosta kuten pääsynvalvonta-asetukset ja omistajan sekä määrittelee muistilohkot, josta tiedosto löytyy.
- Prosessit lukevat tiedostoja tiedostokuvaajien (file descriptors) avulla. Kuvaajat sisältävät samaa tietoa kuin inode, mutta sellaisessa muodossa, että prosessit voivat helposti hakea sen, kirjoittaa siihen jne. Kun kuvaaja on kerran luotu, sitä ei voi enää sitoa toiseen tiedostoon.

Tiedostot II

- Prosessit (ja käyttäjät) voivat myös käyttää tiedostonimiä, jotka identifioivat tiedoston kuvaamalla sen aseman tiedostohierarkiassa. UNIXin tiedostonimet voivat olla absoluuttisia polkunimiä, jotka kuvaavat tiedoston aseman juurihakemiston suhteen. Ne voivat olla myös suhteellisia polkunimiä, jotka kuvaavat aseman työhakemiston suhteen.
- Nimien semantiikka eroaa oleellisesti. Kun prosessi tai käyttäjä käsittelee tiedostoa, käyttöjärjestelmän ydin kuvaa tiedostonimen inodeksi käyttäen iteratiivista menetelmää. Se avaa ensin ensimmäisen hakemiston inoden hakupolulla ja etsii sieltä seuraavan alihakemiston inoden. Tämä jatkuu, kunnes halutun tiedoston inode on löytynyt. Kaksi viittausta samaan tiedostoon voivatkin viitata eri tiedostoihin, jos ensimmäisen viittauksen jälkeen tiedosto on poistettu ja uusi, saman niminen tiedosto on luotu tilalle ennen toista viittausta. Tämä voi luoda ongelmia ohjelmien yhteydessä.

Tiedostot III

- Joka tapauksessa kun tietty tiedostokuvaaja on luotu, se viittaa erityiseen objektiin. Riippumatta siitä, miten tiedostoa manipuloidaan, kuvaajaan liittyvä inode pysyy järjestelmässä siihen asti, kunnes kuvaaja on suljettu.

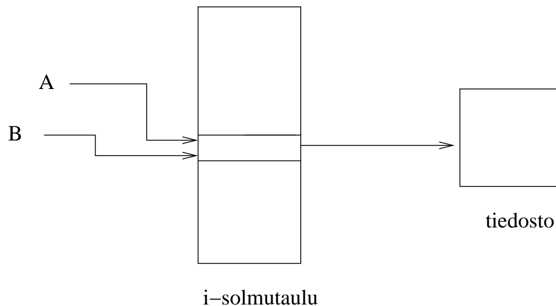
Symboliset linkit I

- Symbolinen linkki on erityinen tiedostotyyppi, jossa tiedosto sisältää viitteen toiseen tiedostoon tai hakemistoon absoluuttisen tai suhteellisen polkunimen muodossa. Symboliset linkit esiintyivät ensimmäisen kerran Berkeleyn Unixin versiossa 4.2 BSD. Nykyään niitä tukevat POSIX-standardi, useimmat Unixin kaltaiset KJ:t, Windows Vista ja Windows 7.
- Symbolinen linkki sisältää merkkijonon, jonka KJ tulkitsee poluksi toiseen tiedostoon tai hakemistoon. Jos symbolinen linkki tuhotaan, kohde jää ennalleen. Jos sen sijaan kohde siirretään (move), nimetään uudelleen tai tuhotaan, symbolinen linkki jää ennalleen, mutta viittaa nyt siis tyhjään.
- **Esimerkki.** POSIX:ssa ja Unixin tyyppisissä käyttöjärjestelmissä symbolinen linkki luodaan komennolla

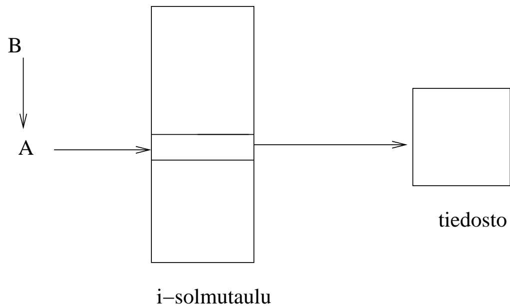
```
ln -s target link\_name
```

Symboliset linkit II

- Symbolisen linkin luomisen jälkeen se toimii kohteen aliaksena eli vaihtoehtoisena nimenä. Komennot, jotka kirjoittavat tai lukevat tiedostoja, kohdistavat toimenpiteensä kohteeseen, kun niille annetaan parametriksi symbolinen linkki. Sen sijaan rm-komento tuhoaa symbolisen linkin, ei kohdetta.
- Unixissa suoraa viittausta tiedostoon kutsutaan kovaksi linkiksi (hard link). Kuvassa 1 on kaksi nimeä, A ja B, jotka viittavat samaan tiedostoon. Nämä viittaukset ovat kovia. Symbolisen linkin rakenteen näyttää sen sijaan kuva 2.



Kuva: Kova linkki



Kuva: Symbolinen linkki

Kovia linkkejä luodaan Unixissa ln-komennolla:

```
$ ln A B
```

Voidaan tarkistaa, että molemmat viittaavat samaan i-solmuun:

```
$ ls -i A B
1321 A
1321 B
```

Symbolisten linkkien tapauksessa i-solmujen tulisi olla eri:

```
$ ln -s A B
$ ln -i A B
1321 A
1467 B
```

Symbolisten linkkien vaarat I

- Tarkastellaan esimerkkinä ohjelmaa, joka avaa ja lukee käyttäjän antamia tiedostoja. Ohjelma on suunniteltu siten, ettei se avaa salasanatiedostoa */home/admin/passwords*.
- Jos ohjelma tarkistaa vain, ettei käyttäjän antama tiedoston nimi ole */home/admin/passwords*, jää ohjelmaan vielä kuitenkin tietoturva-aukko.
- Hyökkääjä voi luoda symbolisen linkin salasanatiedostoon ja antaa sen ohjelmalle päästen näin lukemaan salasanatiedoston.
- Ohjelman pitäisi tarkistaa, ettei käyttäjän antama tiedostonimi ole symbolinen linkki. Toinen mahdollisuus on käyttää systeemikutsua *stat*, joka näyttää tiedostoon liittyvät tiedot.

Symbolisten linkkien vaarat II

- Windowsissa on myös symbolisia linkkejä, mutta tavallisempaa on käyttää oikoteitä (shortcuts). Oikotie on samanlainen kuin symbolinen linkki siinä mielessä, että se on osoitin toiseen tiedostoon. Oikotiet ovat kuitenkin tavallisia tiedostoja eikä Windows tulkitse niitä automaattisesti viittauksiksi toisiin tiedostoihin. Vain ohjelmat, jotka eksplisiittisesti tulkitsevat ne oikoteiksi, voivat seurata osoitinta oikotien viittaamaan tiedostoon.
- Tämä estää Unix-systeemeissä mahdolliset hyökkäykset, mutta ratkaisu tietenkin samalla rajoittaa oikoteiden käyttömahdollisuuksia.

Sovellusten turvallisuus: linkitys I

- Linkitys voi olla *staattista* tai *dynaamista*. Staattisessa linkityksessä kaikki tarvittavat kirjastot ja KJ:n KJ:n systeemifunktiot kopioidaan käännettyyn ohjelmaan levyltä.
- Staattinen linkitys on turvallista, mutta epäkäytännöllistä, koska samaa koodia täytyy kopioida moneen paikkaan.
- Dynaamisessa linkityksessä kirjastot ladataan ohjelman ajoaikana.
- Windowsissa nämä ulkoiset kirjastot tunnetaan nimellä "dynamic linking library"eli DLL. Unixissa näitä kutsutaan jaetuiksi objekteiksi (shared objects).
- Dynaaminen linkitys säästää tilaa ja helpottaa modularisointia. On nimittäin mahdollista muuttaa yhtä kirjastoa ja kääntää vain tämä uudestaan sen sijaan että käännettäisiin koko ohjelmisto.
- Mielivaltaista koodia voidaan soluttaa ohjelmaan jaettujen objektien kautta. Tämä DLL-injektio voi olla hyödyllistä testausvaiheessa, mutta se muodostaa myös tietoturvauhan.

Yksinkertaiset puskurin ylivuotohyökkäykset I

- Aina kun varataan kiinteän kokoinen tila muistista tietojen tallettamista varten, täytyy huolehtia, ettei tietoa tule liikaa, jolloin se voi vuotaa puskurin ulkopuolelle.
- Toisen tyyppinen ylivuoto on aritmeettinen ylivuoto. Useimmissa 32-bitin arkkitehtuureissa etumerkilliset kokonaisluvut esitetään kahden komplementtimuodossa. Tällöin luvut 0x00000000:sta (heksadesimaaliesitys) lukuun 0x7ffffff ($2^{31} - 1$) ovat positiivisia ja luvut 0x80000000-0xffffffff negatiivisia. Jos nyt positivista lukua kasvatetaan ja se vuotaa yli, se muuttuukin negatiiviseksi. Tällä voi olla haitallisia seurauksia, jos yli- ja alivuotoja ei tarkisteta.
- Esimerkki:

Yksinkertaiset puskurin ylivuotohyökkäykset II

```
include <stdio.h>

int main (int argc, char * argv[])
{
   unsigned int connections s = 0;
   // network code
   // ...........
   // ...........
   connections++;
   if (connections < 5)
      grant_access();
   else
      deny_access();

   return 1;
}
```

Yksinkertaiset puskurin ylivuotohyökkäykset III

Nyt hyökkääjä voi yrittää suurta määrää yhteyksiä, jolloin connections vuotaa yli ja muuttuu negatiiviseksi. Näin if-lauseen ehto onkin tosi ja hyökkääjä saa pääsyn resurssiin. Koodin tulisikin olla muotoa:

```
include <stdio.h>
   int main (int argc, char * argv[])
   {
         unsigned int connections s = 0;
        // network code
        // ..........
        // ..........
        if (connections < 5) connections++;
        if (connections < 5)
            grant_access();
        else
            deny_access();
        return 1;
```

Yksinkertaiset puskurin ylivuotohyökkäykset IV

```
}
```

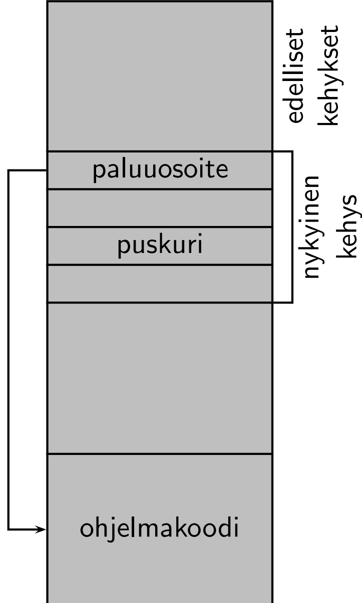
Seuraava esimerkki näyttää, että aritmeettinen ylivuoto voi johtaa myös liian pienen tilan varaamiseen, joka puolestaan johtaa sitten puskurin ylivuototilanteeseen.

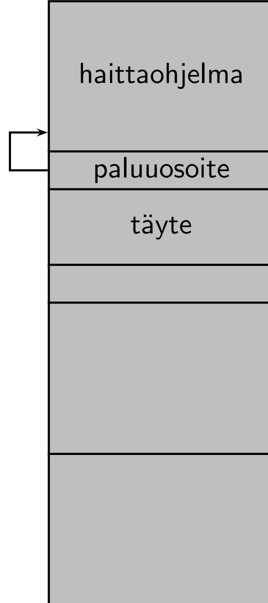
```
int main(int argc, char **argv) {
  unsigned short int total;
  total = strlen(argv[1])+strlen(argv[2])+1;
  char *buff = (char *) malloc(total);
  strcpy(buf, argv[1]);
  strcat(buf, argv[2]);
}
```

Tyyppi `short int` on 16-bittinen. Valitaan `argv[1]` ja `argv[2]` siten, että `total` on 65537. Tyypistä johtuen `total` on itse asiassa 1. Puskurille `buff` varataan siten 1 muistipaikka ja puskurin ylivuoto seuraa. □

Puskurin ylivuoto pinossa I

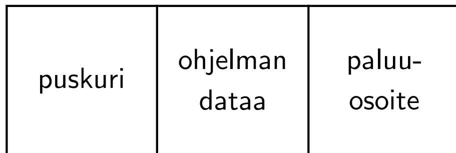
- Tavallisesti puskurin ylivuoto tapahtuu pinossa. Jos tiedolle on varattu puskuri eli esimerkiksi taulukko, joka on lokaali muuttuja, niin se sijaitsee prosessin pinossa. Mikäli tietoa tulee puskuriin enemmän kuin sinne mahtuu, se voi vuotaa yli tuhoten tai muuttaen pinon sisältöä.
- Erityisesti tämä on ongelma C-ohjelmissa. Vältettäviä proseduureja ovat kopiokäsky *strcpy()* ja lukukäsky *gets()*, jotka eivät tarkista, että merkkijono todella mahtuu sille varattuun tilaan.
- Puskurin ylivuoto pinossa voi muuttaa funktion paluuosoitetta, jolloin hyökkääjä voi siirtyä suorittamaan omaa koodiaan (stack smashing attack).

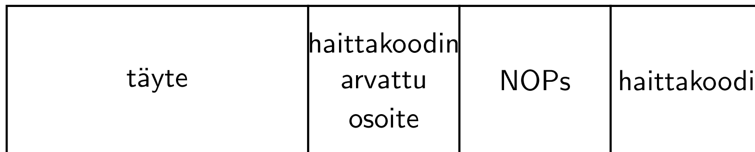

edelliset kehykset
paluuosoite
puskuri
nykyinen kehys
ohjelmakoodi


haittaohjelma
paluuosoite
täyte

- Hyökkääjän ensimmäinen tehtävä on arvata paluuosoitteen paikka. Sen lisäksi hänen täytyy selvittää, mikä osoite pitää kirjoittaa paluuosoitteen paikalle niin, että kone siirtyy suorittamaan haittakoodia. Näiden asioiden selvittäminen on haasteellista kahdesta syystä.
- Ensinnäkin prosessit eivät voi käyttää toisten prosessien muistialueita, joten haittakoodin täytyy sijaita haavoittuvan prosessin muistialueella.
- Toiseksi osoiteavaruus on ennustamaton ja se voi vaihdella koneesta toiseen. Annetussa arkkitehtuurissa kaikkien prosessien pinot alkavat samasta suhteellisesta osoitteesta, joten pinon alkamiskohta on helppo selvittää. Sen sijaan ei ole selvää, missä kohdassa pinoa puskuri sijaitsee. Sen selvittämiseksi tarvitaan paljon kokeita ja arvauksia.
- Jotta näistä haasteista selvittäisiin vähemmällä vaivalla, hyökkääjät ovat kehittäneet monia tekniikoita kuten
  - NOP-liuku (NOP sledding),
  - paluu c-kirjastoon (return-to-libc),
  - hyppy rekisteriin (jump-to register tai trampolining).

NOP-liuku





- Muutettu osoite viittaa nyt NOP-käskyihin, jotka edeltävät haittakoodia. NOP-käsky ei tee mitään, mutta aiheuttaa sen, että seuraava käsky haetaan seuraavasta muistipaikasta. Siten NOP-käskyt suoritetaan peräkkäin kunnes siirrytään haittakoodiin.

Hyppy rekisteriin I

- Vaikka NOP-liuku helpottaa puskurin ylivuotohyökkäyksen onnistumista, se vaatii silti arvauksia, eikä ole kovin luotettava. Hyppy rekisteriin -menetelmä on tarkempi.
- Prosesseja alustettaessa useimmat prosessit lataavat ulkoisten kirjastojen sisällön muistiavaruuteensa tiettyyn paikkaan, joka on ennustettavissa.
- Esim. hyökkääjä voi tietää erityisen konekäskyn Windowsin ytimen DLL:ssä. Oletetaan, että tämä käsky on hyppykäsky osoitteeseen, joka löytyy eräästä rekisteristä, vaikkapa ESP-rekisteristä (ESP-rekisteri on erityinen rekisteri prosessorissa. Se osoittaa aina pinon huipulle.) Jos hyökkääjä sijoittaa haittakoodin ESP:n osoittamaan paikkaan ja muuttaa sen jälkeen nykyisen funktion paluuosoitteen DLL:n hyppykäskyyn, niin palattaessa funktiosta sovellus suorittaa *jmp esp* -käskyn, joka johtaa haittakoodiin.

Hyppy rekisteriin II

- Tämä menetelmä onnistuu paremmin kuin NOP-liuku, sillä se ei riipu koneesta, jos koneet käyttävät samaa KJ:n versiota.

Paluu kirjastoon -hyökkäys I

- Jos hyökkääjä onnistuu määrittämään C:n kirjastofunktion osoitteen haavoittuvan prosessin muistiavaruudessa, prosessin voi pakottaa kutsumaan tätä funktiota, esim. *system()* tai *execve()*. (Edellinen suorittaa komennon, jälkimmäinen käynnistää ohjelman.)
- Puskurin ylivuodolla hyökkääjä saa paluuosoitteen muutettua siten, että osoite osoittaakin kirjastofunktioon. Paluuosoitteen jälkeen pinoon täytyy viedä kirjastofunktion paluuosoite ja parametrit.
- Näin hyökkääjä saa prosessin hallintaansa päästen suorittamaan haittakoodia. Tällä menetelmällä on se etu, ettei mitään koodia suoriteta pinossa. Pinossa on vain argumentit, joten hyökkäystä voi käyttää, vaikka pino olisi asetettu ei-suorittavaan tilaan.

Komentotulkkikoodi I

- Hyökkääjät valitsevat usein haittakoodin, joka käynnistää komentotulkin ja antaa näin hyökkäjälle mahdollisuuksia suorittaa lisäkomentoja.
- Koska haittakoodia suoritetaan pinossa, sen täytyy olla puhdasta konekieltä. Tällaisen kirjoittaminen voi olla vaikeaa.
- Esim. konekieli sisältää usein tyhjän merkin 0x00. Tätä ei voi kuitenkaan käyttää puskurin ylivuotohyökkäysten yhteydessä, sillä se on myös merkkijonon loppumerkki (erityisesti C:ssä). Jos se esiintyisi haittakoodissa, koodin lataaminen puskuriin ja ylivuotoalueelle loppuisi kesken kaiken. Siten hyökkääjien on keksittävä, miten tyhjät merkit korvataan konekielessä muilla merkeillä ilman, että haittakoodin tavoite muuttuu.

Komentotulkkikoodi II

- Jos ohjelmassa käytetään *setuid*-komentoja ja ohjelma on haavoittuva ylivuotohyökkäykselle, niin hyökkääjä voi haittakoodin alussa suorittaa *setuid*-käskyn ja vasta sitten käynnistää komentotulkin. Näin hyökkääjä pääsee suorittamaan käskyjä vaikkapa juurioikeuksin.

Ylivuotohyökkäysten estäminen I

- Suurin syy ylivuotoihin on huolimaton ohjelmointi. Syötteistä tulisi aina tarkistaa, että ne eivät ylitä sallittua pituutta eivätkä sisällä muita kuin sallittuja merkkejä.
- Erityisesti em. asiat on otettava huomioon C:ssä ja $\mathrm{C}++$:ssa.
- KJ:t sisältävät myös estomekanismeja. Yksi tällainen on kanarialinnun käyttö.

Ylivuotohyökkäysten estäminen II

	Ylivuotanut data	Muut–tunut osoite	Haitta–koodi

- Kun ylivuoto tapahtuu, se tuhoaa satunnaisluvun (kanarialinnun) ja KJ huomaa tämän lopettaen samalla prosessin toiminnan.

Ylivuotohyökkäysten estäminen III

- Toinen menetelmä on Microsoftin kääntäjäoptio *Point Guard*. Jos sitä käytetään, jokainen osoitinviittaus paluuosoitteet mukaanlukien salataan XOR-operaatiolla ennen ja jälkeen käyttöä. Hyökkääjä ei saa näin paluuosoitetta muutettua mielekkäällä tavalla.
- Pino voidaan tehdä myös sellaiseksi, ettei siellä olevaa tietoa voi tulkita käskyksi.
- Monet KJ:t käyttävät nykyään muistiavaruuden satunnaistamista (address space layout randomization, ASLR), joka järjestää prosessin datan uudestaan satunnaisella tavalla. Tämä vaikeuttaa oikean hyppykohdan valitsemista.
- Hyökkääjät ovat onnistuneet kuitenkin murtamaan näitäkin estokeinoja. Esim. ASLR-toteutukset 32-bittisissä Windows- ja Linux-systeemeissä sisältävät liian vähän satunnaisuutta niin, että raa'an voiman hyökkäykset onnistuvat.

Ylivuotohyökkäysten estäminen IV

- Sanoma onkin: Huolehdi koodista sovellustasolla, KJ-tasolla se voi olla liian myöhäistä.

Kekopohjaiset ylivuotohyökkäykset I

- Pinolle varataan tilaa staattisesti käännösaikana tai sitä varataan ja vapautetaan automaattisesti funktioioden kutsun yhteydessä.
- On kuitenkin hyödyllistä, että ohjelmoijat voivat varata muistia dynaamisesti siten, että varaus säilyy funktiokutsusta toiseen. Tällaista muistia varataan keosta.
- Ohjelmoijien pitäisi myös muistaa vapauttaa muisti, kun sitä ei enää tarvita. Muuten muisti fragmentotuu.
- Ylivuodot keossa aiheuttavat samanlaisia ongelmia kuin ylivuodot pinossa. Niiden hyödyntäminen on kuitenkin hankalampaa ja edellyttää syvällistä perehtymistä roskien keruuseen ja keon toteutukseen.
- Tarkastellaan esimerkkinä GNU-kääntäjän vanhempaa malloc-funktion toteutusta. Tässä versiossa keon varattuja muistilohkoja pidetään linkitetyssä listassa. Jokaisessa lohkossa on linkki edelliseen ja seuraavaan lohkoon.

Kekopohjaiset ylivuotohyökkäykset II

- Kun lohko merkitään jälleen vapaaksi, edellisen ja seuraavan lohkon osoittimia päivitetään osoittamaan toisiinsa, jolloin vapaa lohko poistuu listasta. Tämän operaation tekee *unlink*-makro.
- Eräs ylivuototekniikka hyödyntää tätä ratkaisua. Jos ohjelma tallettaa syötteen lohkoon tarkistamatta pituutta, hyökkääjä voi antaa syötteen, joka vuotaa lohkon yli seuraavaan lohkoon. Jos syöte on huolellisesti suunniteltu, voi käydä niin, että ylivuoto tuhoaa seuraavan lohkon linkit ja merkitsee lohkon vapaaksi sillä tavalla, että *unlink* huijataan kirjoittamaan dataa mielivaltaiseen muistipaikkaan. Erityisesti hyökkääjä voi huijata *unlink*-funktiota siten, että haittakoodin osoite kirjoitetaan paikkaan, joka johdattaa kontrollin jossain vaiheessa haittakoodiin.
- Eräs sellainen paikka on *.dtors*. GCC:llä käännetyt ohjelmat voivat sisältää konstruktori- ja destruktorifunktioita. Nämä suoritetaan ennenkuin ohjelma alkaa ja sen jälkeen kun ohjelman suoritus on päättynyt.

Kekopohjaiset ylivuotohyökkäykset III

- Toinen hyökkäyksille altis paikka on GOT (global offset table). Tämä taulukko näyttää funktioiden absoluuttisen osoitteen. Jos hyökkääjä onnistuu muuttamaan osoitetta GOT:ssa haittakoodin osoitteeksi, hän pääsee suorittamaan haittakoodia kun funktiota kutsutaan.

Kekoon perustuvan ylivuodon estäminen I

Monia keinoja kuten pinojenkin yhteydessä:

- Osoiteavaruuden satunnaistaminen.
- Keko voidaan tehdä sellaiseksi, ettei sen sisällä olevia käskyjä voi suorittaa.
- Uusissa järjestelmissä keon metadata (esim. lohkojen osoittimet) ovat eri paikassa kuin varsinainen data, mikä torjuu unlink-tyyppiset hyökkäykset.
- Sovellusohjelmien turvallisuus on kuitenkin tärkein keino suojautua ylivuotohyökkäyksiä vastaan.

NISTin yleisiä suosituksia KJ:n turvallisuuden takaamiseksi I

Ensimmäiset ohjeet koskevat suunnittelua. Nämä ohjeet sopivat yhtä hyvin sovellusohjelmien kuin palvelinohjelmistojen yhteyteen:

- S1 Määrittele systeemin tarkoitus, talletettujen tietojen laatu, sovellukset ja palvelut sekä niiden turvallisuusvaatimukset.
- S2 Luokittele käyttäjät ja heidän oikeutensa sekä tiedot.
- S3 Määrittele, kuinka käyttäjät todennetaan.
- S4 Määrittele, kuinka hallinnoidaan tiedostoihin pääsyä.
- S5 Selvitä, mihin, muissa koneissa oleviin tietoihin systeemi pääsee käsiksi.
- S6 Päätä, ketkä hallinnoivat systeemiä ja miten.
- 7 Analysoi, mitä muita turvamekanismeja systeemi tarvitsee kuten esimerkiksi palomuureja, virustentorjuntaa ja kirjanpitoa.

Tämän jälkeen seuraa yksityiskohtaisempia ohjeita KJ:hin liittyen:

NISTin yleisiä suosituksia KJ:n turvallisuuden takaamiseksi II

KJ1 Asenna ja päivitä käyttöjärjestelmä.

KJ2 Vahvista ja konfiguroi KJ:ää siten, että se vastaa sovelluksen vaatimuksia:

- poista tarpeettomat palvelut, sovellukset ja protokollat;
- konfiguroi käyttäjät, ryhmät ja oikeudet;
- konfiguroi resurssien hallinta.

KJ3 Asenna ja konfiguroi muita turvamekanismeja.

KJ4 Testaa järjestelmän turvallisuus varmistuaksesi, että toimenpiteet ovat olleet riittäviä.