

## Sisältö

<b>1 Johdanto</b>	<b>3</b>
<b>2 Yleisiä tietoturvaongelmia</b>	<b>3</b>
2.1 Muistiturvallisuus . . . . .	4
<b>3 Testausmenetelmät</b>	<b>5</b>
3.1 Luokittelu . . . . .	5
3.2 Testauksen kriteereitä . . . . .	6
<b>4 Staattinen analyysi</b>	<b>6</b>
<b>5 Fuzzaus</b>	<b>6</b>
5.1 Testitapausten generointi . . . . .	7
<b>6 Yhteenvetö</b>	<b>7</b>
Lähteet	8

## 1 Johdanto

Ohjelmistojen haavoittuvuuksia ovat verrattaen ikäviä: Tietoturva-aukoista aiheutuneesta ylimääriäisestä työstä tietojärjestelmien ylläpitäjille sekä menetystä työajasta voi seurata suuria rahallisista tappioita, puhumattakaan mahdollisista henkilötietojen tai yrityssalaisuuksien vuotamisesta. Esimerkiksi Microsoftin IIS-palvelimen haavoittuvuuden avulla levinneestä Code Red-madosta arvioidaan aiheutuneen yhteensä noin 2,6 miljardin tappiota [MSC02]. Siksi on toivottavaa, että ohjelmiston tietoturvasta voidaan varmistua ennen ohjelmiston käyttöönottoa.

Tietoturvaongelmia voi yrittää löytää käsitönä tutkimalla ohjelman lähdekoodia, mikä tietenkään on mahdollista vain ohjelmiston varsinaisille kehittäjille tai avoimen lähdekoodin ohjelmille. Lähdekoodin puuttuessa täytyy ensin ohjelmatiedosto takaisinkääntää disassembler-ohjelmalla symboliselle konekielelle ja tutkia ohjelmaa konekielitasolla. Kummassakin tapauksessa manuaalinen tutkiminen on työlästä ja aikaavievää, joten automatisoitu ratkaisu on paikallaan.

+paragrafi: Luossa 2 kerrotan sitä, 3 tarkemmin

## 2 Yleisiä tietoturvaongelmia

Tietoturva-ja haavoittuvuuden vakavuuteen vaikuttaa keskeisesti aiheutuuko siitä *luottamusajan* (trust boundary) ylitys. Esimerkiksi jos jokin palvelun kaatavan ohjelointivirheen voi laukaista pelkästään järjestelmän ylläpitäjä, ei kyseisellä virheellä ole suurta tietoturvamerkitystä. Lähes samaan loppulokseenhan päädytään jos ylläpitäjä sammuttaa palvelun tavallisia keinoja käyttämällä. Sen sijaan jos vaikkapa Javascript-sovelmat selaimessa pääsee lukemaan käyttäjän tiedostoja, on tilanne vakavampi, koska web-sivujen katselemisen tarkoitus on olla turvallista ja selaimen siten kuuluu estää sovelmiin pääsy tiedostojärjestelmään. Vastaavaan tapaan tavallisen käyttäjän pääsy suorittamaan ohjelmia ylläpitäjän oikeuksilla tai käyttöjärjestelmätilassa

hypoteesi  
3  
lähdevirheet

rikkoo luottamusrajan.

## 2.1 Muistiturvallisuus

Ohjelointikieli vaikuttaa ratkaisevasti siihen miten laajoja seurauksia ohjelointivirheillä voi olla tietoturvan kannalta. Suurin ohjelointikielestä riippuva tekijä on kielen *muistiturvallisuus*, eli tunnistaako ajonaikainen ympäristö kiellettyjen muistiviitteiden tekemisen. Nykypäivänä yleisimmät muistiturvallisuuden puutteesta kärsivät ohjelointikielet ovat C ja C++, joita tehokkuussystä edelleen käytetään laajasti.

Muistivirheet ilmenevät useimmiten prosessin tappamisena tietyllä käytöjärjestelmäkohtaisella virhekoodilla. Unix-pohjisissa järjestelmissä tämä virhe on tunnettu 'Segmentation fault'. Koska muistivirheistä aiheutuu useimmiten ohelman kaatuminen, mahdollistaan muistivirheen olemassaolo palvelunestohyökkäyksen eli DoS, denial-of-service-hyökkäyksen. Lisäseurauksena prosessin pakotetusta kuolemasta on se, ettei prosessilla ole mahdollisuutta siivota jotakin säilyvä tilaa, esimerkiksi väliaikaistiedostoja. Hallitsematon kuolema voi myös aiheuttaa tiedon korruptoitumista jos ohjelma kautta esimerkiksi kesken levykirjoitusta. Kriittisimmät muistivirheet voivat mahdollistaa hyökkääjän suorittaa mielivaltaista koodia ohjelmaprosessin käyttöoikeuksilla (RCE, Remote Code Execution).

Muistivirheksi luokitellaan tyypillisesti seuraavat:

- Taulukon yli tai ali indeksointi: C-kielen taulukoissa ohjelmojan vastuulla on huolehtia, ettei taulukkoa indeksoida liian suurella tai negatiivisella indeksillä. Jos näin pääsee tapahtumaan, kielen spesifikaatio ei otta kantaa siihen mitä tapahtuu. Käytännössä usein tapahtuu joko muistiviittaus taulukkoa ympäröiviiin muuttujaan tai ajonaikaympäristön tietorakenteisiin. Suurin riski pinossa olevan taulukon ohi kirjoittamisessa onkin aktivaatiotietueessa sijaitsevan funktion paluuosoitteeseen ylikirjoittaminen. Tästä seuraa käytännössä suoraan mielivaltaisen

## Segmentointivirheet (Seg. fault.)



rikoo luottamusrajan.  
virheen 2:n tarkkuus nim suprast  
ett pohditaan  
pudotus vrti vittan - pohd 2:n  
ylivuotoa, onko  
siti hine mutta?  
(suhkeessa  
paineohjeen)

koodin suoritus.

- Puskurin ylivuoto: Vastaavasti kuin taulukoiden kanssa, C-kielessä täytyy merkkijonoja ja tavupuskureita kopioitaessa tai luettaessa ohjelmojan itse huolehtia, että puskurissa on riittävästi tilaa. Seuraukset ovat enimmäkseen samat kuin taulukon yli indeksoimisella. Puskureiden ylivuoto on C:ssä harmillisen helppo tehdä. Jo C:n standardikirjastosta löytyy funktoita, joille ei ole ollenkaan mahdollista antaa tulospuskurin kokoa ja siten aiheuttavat ylivuodon jos tulos ei mahdu puskuriin. Tällaisia funktoita ei koskaan pitäisi käyttää syötteen käsittelyn yhteydessä. Näistä syistä puskurin ylivuodot pinossa ovat ylivoimaisesti yleisin syy RCE-haavoittuvuksille.

## 3 Testausmenetelmät

Ohjelmistotekniikan menetelmistä tuttu laadunvarmistustekniikka on automaattiset testit [Som06]. Testausta voidaan soveltaa tietoturvaongelmien välttämiseen tietyn edellytyksin: sen sijaan, että testataan toivotun toiminnallisuuden olemassaoloa, testataankin *epätoivotun käytöksen* puuttetta [PHP<sup>+</sup>11]. Yleisempä epätoivottuja tapahtumia ovat kaatumiset ja jumiutumiset (esimerkiksi ikisilmukat).

pitkist vltta  
vys. spesif  
(ketti vengro pml)

### 3.1 Luokittelu

Testauskeinoja voidaan tavallisten funktionalistien tapaan luokitella karkeasti *blackbox*- ja *whitebox*-testeiksi sen mukaan kuinka paljon testaamisen kohdistuu ohjelmiston tavanomaisiin rajapointoihin ja kuinka paljon ohjelmiston sisäisen rakenteen toimintaan [Som06].

Blackbox-testauksessa ohjelmatiedostoa ajetaan aivan normaalisti, ja tutkitaan sen käytäytymistä ohjelmaan sopivilla syötteillä. Esimerkiksi palvelinohjelmiston ollessa kyseessä siihen avataan verkkoyhteys, tai komen-

esim. 3.3 Auton-testi-menetelmä +  
 → yksellisiä suuria  
 lyhyt lähdekuva  
 Fuzzaukselle (I paray) voi olla  
 erilaisia aiheita

toriviohjelmalle annetaan syöte normaalista komentoriviparametrien kautta.

Whitebox-testauskeinoissa kajotaan ohjelman sisäiseen rakenteeseen. Tämä minkälaiseen testaukseen on tapoja huomattavasti enemmän. Esimerkiksi jotkut keinot voivat vaatia pääsyä ohjelman lähdekoodiin, tai sitten ohjelman suoritusta voidaan analysoida tai muuttaa konekielitasolla.

### 3.2 Testauksen kriteereitä

Tarkastellaan seuraavaksi muutamia automaattista testausmenetelmää: staattista analyysiä ja fuzzusta. (=mutta eksakti :)

taasohjelto

### 4 Staattinen analyysi

Staattiseen analyysiin perustuvat keinot ovat lähdekoodin analyysiä tekeviä whitebox-menetelmiä. Tällaiset menetelmät yrittävät paikantaa lähdekoodista tyypillisiä ohjelmointivirheitä. Monista tällaisista ongelmista saattaa olla seuraamuksia tietoturvan kannalta [BPCL09]. Yleisin tällainen keino on käänäjien tarjoamat varoitukset ohjelmiston käänösaihiana. Usein käytettyjä staattisen analyysin työkaluja ohjelmointivirheiden etsintään ovat lint-tyyliiset [Joh78] ohjelmistot eri ohjelmointikielille sekä esimerkiksi Coverity [BBC<sup>+</sup>10].

### 5 Fuzzaus

Fuzzaus on raa'an voiman keinot automaattiseen tietoturvatestaukseen. Fuzzauksen tarkoitus on generoida satunnaisia syötteitä testattavalle ohjelmalle siinä toivossa, että ohjelma ei käsitle niitä oikein [MFS90]. Muun muassa aiemmin mainittuja muistivirheitä löytyy usein fuzzauksella: normaalista reilusti pidemmät merkkijonot voivat yliuotaa kiinteäkokoisia puskureita. Samaten erikoisia merkkejä sisältävät merkkijonot sekä erittäin suuret tai negatiiviset lukuarvot voivat olla ongelmallisia [Oeh05] esimerkiksi yliuotamalla laskutoimituksia.

Fuzzaukselle hyvin sopivia kohteita ovat muun muassa erinäisten tiedostoformaattien jäsentäjät [GLM12, PHP<sup>+</sup>11]: esimerkiksi web-selaimet joutuvat käsittämään muun muassa HTML-, CSS-, PNG- ja JavaScript-muotoisia tiedostoja [PHP<sup>+</sup>11]. Tämä on varsin suuri määriä koodia, joka ottaa syötteekseen tiedostoja suoraan lähtökohtaisesti epäturvallisesta Internetistä, jolloin hyökkäyspinta-alaakin on runsaasti.

Fuzzaukseen sopivien syötteen luontiin on olemassa runsaasti tekniikoita aina yksinkertaisista blackbox-menetelmistä monimutkaisempia keinoihin:

- Yksinkertaisimmillaan syöte voi olla lähes pelkkää satunnaista dataa. Esimerkiksi monien Unixin komentorivityökalujen syöte koostuu pelkistä tekstiriveistä ilman sen kummempaa rakennetta, jolloin jo no rivinvaihdoilla eroteltua satunnaisgeneroituja tavuja [MFS90] on riittävä syöte ohjelmalle.
- Ennalta olemassa olevia kelvollisia syötteitä voidaan mutataida lisäämällä, poistamalla, muokkaamalla jne. satunnaisesti.
- Rakenteellisesti (enimmäkseen) kelvollisia, mutta normaalista harvoin esiintyviä syötteitä voidaan luoda esimerkiksi kontekstittoman kielioin tai jonkin muun formaalin syöteen määrittelyn perusteella.
- Ohjelman suoritusta voidaan analysoida symbolisesti välttämään syötteitä, jolla ei ole vaikutusta ohjelmaan [GLM12].

#### 5.1 Testitapausten generointi

### 6 Yhteenveto

55 myt  
+5 sivua