+ tyyli enimmäkseen hallussa, ilmaisu ja englanninvälttely vielä kuntoon mutta tulos erinomaista
+ lähdeluettelo sisällöllisesti tiiviisti

\bibliographystyle{alpha}

Ohjelmistojen haavoittuvuustestaus

Tuomas Tynkkynen

Essee

Helsingin Yliopisto
Tietojenkäsittelytieteen laitos

Helsinki, 24. tammikuuta 2013

1 Haavoittuvaisuustestaus

1.1 Johdanto

Ohjelmistojen haavoittuvuukset ovat verrattaen ikäviä. Tietoturva-aukoista aiheutunut ylimääräinen työ voi jo itsessään maksaa miljoonia dollareita [2], puhumattakaan mahdollisista henkilötietojen tai yrityssalaisuuksien vuotamisesta. Siksi on toivottavaa, että mahdolliset tietoturva-aukot löydetään mielellään jo ennen tuotantoon asentamista.

Tietoturvaongelmia voi toki ohjelmoija löytää tutkimalla lähdekoodia, tai koodin puuttuessa disassembloimalla ohjelman binääriä. Tämä kuitenkin on työlästä ja aikaavievää, joten automatisoitu ratkaisu on paikallaan.

2 Automaattiset menetelmät

Ohjelmistotekniikan menetelmistä tuttu laadunvarmistustekniikka on automaattiset testit [4]. Testausta voidaankin soveltaa tietoturvaongelmien välttämiseen tietyin erotuksin: sen sijaan, että testataan toivotun toiminnallisuuden olemassaoloa, testataankin *epätoivotun käytöksen* puutetta [1]. Selkeitä epätoivottuja tapahtumia ovat kaatumiset (esimerkiksi luku/kirjoitus muistialueen ulkopuolelta) ja jumiutumiset (esimerkiksi ikisilmukat). Näissä molemmissa on palvelunestohyökkäyksen (DoS, Denial of Service) uhka, sekä edeltävässä mahdollisesti myös ulkopuolisen koodin suorituksen mahdollisuus (RCE, Remote Code Execution) [?].

Testauskeinoja voidaan tavallisten funktionaalisten testien tapaan luokitella karkeasti *black box*- ja *white box*-testeiksi sen mukaan kuinka paljon testaaminen kohdistuu ohjelmiston tavanomaisiin rajapintoihin vai ohjelmiston sisäisen rakenteen toimintaan [4].

Black box-testauksessa ohjelmabinääriä ajetaan aivan normaalisti, ja

tutkitaan sen käyttäytymistä ohjelmaan sopivilla syötteillä. Esimerkiksi palvelinohjelmiston ollessa kyseessä siihen avataan verkkoyhteys, tai komentoriviohjelmalle annetaan erinäisiä tiedostoparametrejä.

White box-testauskeinoissa kajotaan ohjelman sisäiseen rakenteeseen. Tähän tapoja on lukuisia. Esimerkiksi jotkut keinot voivat vaatia pääsyä ohjelman lähdekoodiin, tai sitten ohjelman suoritusta voidaan analysoida tai muuttaa konekielitasolla.

2.1 Staattinen analyysi

Staattiseen analyysiin perustuvat keinot, kuten esimerkiksi Coverity-ohjelmisto [5] tai monien kääntäjien varoitukset yrittävät paikantaa ohjelmiston lähdekoodista tyypillisiä ohjelmointivirheitä. Monista tällaisista ongelmista saattaa olla seuraamuksia tietoturvan kannalta.

2.2 Fuzzing

Fuzzing on brute force-keino automaattiseen tietoturvatestaukseen. Fuzzauksen tarkoitus on generoida satunnaisia syötteitä testattavalle ohjelmalle siinä toivossa, että ohjelma ei käsittele niitä oikein [3]. Fuzzaukselle hyvin sopivia kohteita ovat muun muassa erinäisten tiedostoformaattien jäsentäjät [2, 1]: esimerkiksi web-selaimet joutuvat käsittelemään muun muassa HTML-, CSS-, PNG- ja JavaScript-muotoisia tiedostoja, tarjoten laajasti hyökkäyspintaalaa [1].

2.2.1 Syötteiden generointi

Fuzzaukseen sopivien syötteiden luontiin on olemassa runsaasti tekniikoita aina yksinkertaisista black box-menetelmistä monimutkaisempiin keinoihin:

- Yksinkertaisimmillaan syöte voi olla pelkkä jono satunnaisgeneroituja tavuja [3].
- Ennalta olemassa olevia valideja syötteitä voidaan *mutatoida* lisäämällä, poistamalla, muokkaamalla jne. satunnaisesti.
- Rakenteellisesti (enimmäkseen) valideja, mutta normaalisti harvoin esiintyviä syötteitä voidaan luoda jonkun kieliopin perusteella.
- Ohjelman suoritusta voidaan analysoida symbolisesti välttämään syötteitä, jolla ei ole vaikutusta ohjelmaan [2].

3 Lähteet

[1] Security Testing of Web Browsers. http://www.cloudsw.org/current-issue/201112226146

[2] SAGE: Whitebox Fuzzing for Security Testing. http://research.microsoft.com/en-us/um/people/pg/public_psfiles/cacm2012.pdf

[3] An Empirical Study of the Reliability of UNIX Utilities. ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf

[4] Ian Somerville: Software Engineering

[5] http://coverity.com