Patrik Jokela

Web-sovelluksen haavoittuvuustestauksen automatisointi

Tietotekniikan kandidaatintutkielma

16. joulukuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Patrik Jokela

Yhteystiedot: patrik.a.jokela@student.jyu.fi

Työn nimi: Web-sovelluksen haavoittuvuustestauksen automatisointi

Title in English: Automating penetration testing for web-application

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 21+0

Tiivistelmä: Haavoittuvuustestauksessa käytetään samanlaista lähestymistapaa ja tekniikkaa miltei jokaisessa testauksessa. Toistettavien töiden automatisointi auttaa haavoittuvuustestaajia käyttämään testausaikansa hyödyllisemmin suorittaakseen syvällisempiä testejä. Tämän avulla saadaan parannettua järjestelmien tietoturvallisuutta. Tutkimuksessa havaittiin, että tiedustelu- ja skannausvaihe voidaan automatisoida miltei kokonaan sekä hyökkäysvaiheen osia on mahdollista suorittaa automaattisesti työkaluilla.

Avainsanat: automatisointi, haavoittuvuustestaus, hakkerointi, hakkeri, tietoturvallisuus, tietoturva

Abstract: Penetration testing contains several basic approaches and techniques that are similar in almost every test case. Automating repeatable tasks helps penetration testers to use their testing time more efficiently to carry out more deeper tests. This has an positive affect on the cyber security. In this research we found out that it is possible to almost fully automate the reconnaissance and scanning phases but also parts of the exploitation phase can be automated using automatic tools.

Keywords: automating, penetration testing, hacking, hacker, information security, cyber security

Kuviot

Sisältö

1 Johdanto

Haavoittuvuustestauksessa käytetään yleensä samoja lähestymistapoja ja työkaluja. Näiden toistuva kaavamainen suorittaminen vie kallisarvoista aikaa haavoittuvuustestaajalta. Monesti myös testausaika on rajallinen, jolloin testit pitää saada suoritettua nopeasti, mutta myös tarpeeksi syvällisesti. Tällaisten suoritusten automatisointi säästäisi sekä aikaa että rahaa, mutta antaisi myös testaajille mahdollisuuden suorittaa syvällisempiä testejä. Tämän seurauksena myös turvallisuus ohjelmistoissa parantuisi.

Haavoittuvuustestauksessa edetään yleensä samalla kaavalla: tiedustelu, skannaus, hyökkäys. Näiden vaiheiden sisältö vaihtelee riippuen testattavasta kohteesta. Kuitenkin osa näistä vaiheista on kaikille järjestelmille samoja, jolloin haavoittuvuustestaajat joutuvat toistamaan samoja toimenpiteitä jokaiselle kohteelle.

Tutkimusstrategiana käytetään systemaattista kirjallisuuskartoitusta. Tällä strategialla saadaan selkeä vastaus tutkimuskysymyksiin. Kyseiseen tutkimuskysymykseen ei löydy suoraan muita tutkimuksia, mutta itse haavoittuvuustestauksesta löytyy monia tutkimuksia ja tieteellisiä artikkeleita, joiden avulla voidaan suorittaa tutkimus. Tutkimuksessa käytetään myös apuna kirjoittajan omia kokemuksia haavoittuvuustestaajana.

2 Haavoittuvuustestaus

Haavoittuvuustestauksella (engl. *penetration testing*) tarkoitetaan yrityksen tai ohjelmiston tietoturvan testaamista (Northcutt, Madden ja Welti 2004). Se auttaa löytämään haavoittuvuuksia, joita mahdollinen hyökkääjä voisi käyttää hyödykseen esimerkiksi lamauttamaan yrityksen palveluita tai varastamaan liikesalaisuuksia (Baloch 2015, ss. 3).

2.1 Eettinen hakkerointi

Haavoittuvuustestaus sisällytetään yleisesti eettiseksi hakkeroinniksi (engl. *ethical hacking*). Eettinen hakkerointi tarkoittaa hyväntahtoista hakkerointia, jolla pyritään löytämään haavoittuvuuksia ja parantamaan tietoturvaa.

Eettinen hakkeri, valkohattuhakkeri (engl. *white hat hacker*) ja haavoittuvuustestaaja ovat kaikki synonyymejä toisilleen. Niillä tarkoitetaan hyväntahtoisia hakkereita, jotka suorittavat haavoittuvuustestausta parantaakseen järjestelmien tietoturvaa (Engebretson 2013, ss. 2–3). Heidän tulisi toimia samoin kuin oikeat pahaa tahtovat mustahattuhakkerit (engl. *black hat hacker*), jotta testin tulokset olisivat mahdollisimman todenmukaisia. Harmaahattuhakkeri luokitellaan mustahattuhakkerin ja valkohattuhakkerin väliin. Harmaahattuhakkeri voi esimerkiksi toimia valkohattuhakkerina töissä, mutta sivutöikseen varastaa ja myy eteenpäin asiakasyrityksen tietoja (Baloch 2015, ss. 1).

Haavoittuvuustestaus voidaan jakaa viiteen erilaiseen testiluokkaan: nollan tiedon testi (engl. *zero knowledge*), osittaisen tiedon testi (engl. *partial knowledge*), täyden tiedon testi (engl. *full knowledge*), sokkotesti (engl. *blind*) ja tuplasokkotesti (engl. *double blind*) (Henry 2012, ss. 39).

Nollan tiedon testissä testaajille ei kerrota kohdeyrityksestä mitään, jolloin testausvaiheet muistuttavat eniten oikeaa hyökkäystä (Henry 2012, ss. 40–41). Testaajien täytyy tiedustella kohteesta ensin kaikki tieto julkisista lähteistä, jonka jälkeen voidaan edetä tarkempaan skannaukseen. Tämä on haastavin sekä aikaa vievin luokka haavoittuvuustestaajille. Osittaisen tiedon testit ovat yleisimpiä, sillä ne ovat tehokkaimpia suorittaa. Niissä haavoittuvuus-

testaajille annetaan tietoa kohdejärjestelmästä ja esimerkiksi kohde IP-osoitteet joita testata (Henry 2012, ss. 41). Tällä tiedolla testaamisen voi aloittaa ilman syvempää tiedusteluvaihetta. Täyden tiedon testit suoritetaan yleensä yrityksen sisäisen tiimin toimesta ja silloin testaajilla on kaikki tieto järjestelmistä, niiden toiminnallisuuksista sekä asetuksista (Henry 2012, ss. 41).

Sokkotestejä voidaan suorittaa mikäli halutaan testata yrityksen toimintakykyä ja reagointia hyökkäyksiin. Tällöin yrityksen hallinnointiyksikkö ei tiedä haavoittuvuustestaajien hyökkäyksestä, jolloin voidaan tarkastella yksikön kykyä reagoida ja toimia hyökkäyksen sattuessa (Henry 2012, ss. 42). Tuplasokkotesti on käytännössä sama kuin sokkotesti, mutta nyt myöskään yrityksen turvallisuusyksikkö ei tiedä testihyökkäyksestä, jolloin saadaan tarkasteltua myös yrityksen sisäisten yksiköiden yhteistyön toimivuutta (Henry 2012, ss. 42).

2.2 Haavoittuvuustestauksen vaiheet

Tiedustelu (engl. *reconnaissance*) ja tiedonkeruu (engl. *information gathering*) ovat haavoittuvuustestauksen tärkeimpiä vaiheita (Weidman 2014, ss. 113; Wilhelm 2013, ss. 151) Tässä vaiheessa pyritään saamaan niin paljon tietoa kohdeympäristöstä kuin mahdollista. Vaikka yleensä asiakas kertoo omasta ympäristöstään haavoittuvuustestaajalle, voi tiedonkeruu vaiheessa löytyä kohteita asiakkaan infrastruktuurista, jotka asiakas on esimerkiksi unohtanut kertoa (Wilhelm 2013, ss. 152–153). Tiedustelun ja tiedonkeruun haavoittuvuustestauksessa voi jakaa kahteen osa-alueeseen: aktiiviseen sekä passiiviseen. (Baloch 2015, ss. 53; Wilhelm 2013, ss. 152).

Passiivisella tiedustelulla tarkoitetaan tiedonkeräämistä ja etsimistä avoimista lähteistä (OSINT, engl. *Open Source Intelligence*). Lähteinä voidaan käyttää sosiaalista mediaa, työnhaku sivustoja, foorumeita, uutisia, arkistoituja versioita nettisivuista sekä esimerkiksi työntekijöiden omia nettisivuja (Baloch 2015, ss. 54; Broad ja Bindner 2014, ss. 99–101; Weidman 2014, ss. 114)

Aktiivinen tiedustelu voidaan rinnastaa hyvin usein myös skannausvaiheeseen, josta kerrotaan seuraavassa alaluvussa. Aktiivisen tiedustelun aikana kohdeverkkoa skannataan työkaluilla, jotta voidaan löytää kaikki avoimet portit ja saada tietoa mitä järjestelmiä kohdever-

kossa on (Baloch 2015, ss. 53; Wilhelm 2013, ss. 172–181). Tämä vaihe on todella ”äänekäs” eli se voidaan havaita helposti mikäli kohdeverkossa on palomuuri tai IDS (Tunkeilijan havaitsemisjärjestelmä, engl. *Intrusion detection system*), jotka pitävät yllä lokia (Baloch 2015, ss. 53; Engebretson 2013, ss. 22).

Skannausvaiheessa (engl. *scanning phase*) käytetään tiedusteluvaihessa kerättyä dataa, jonka avulla pureudutaan syvemmälle järjestelmään. Tässä vaiheessa etsitään kohdeverkosta kaikki järjestelmät ja niiden käyttöjärjestelmät sekä versiot (Baloch 2015, ss. 97; Wilhelm 2013, ss. 186). Skannaustuloksien avulla voidaan aloittaa etsimään tunnettuja haavoittuvuuksia kohdeverkon järjestelmistä. Järjestelmää voidaan myös skannata automaattisilla työkaluilla, jotka ovat ohjelmoituja etsimään tiettyjä haavoittuvuuksia järjestelmästä ja raportoimaan niistä (Hutchens 2014, ss. 53). Näillä automaattisilla työkaluilla siirrytään kuitenkin hyvin nopeasti skannausvaiheen yli hyökkäysvaiheeseen, sillä hyvin usein löydettyään haavoittuvuuden, nämä työkalut myös kokeilevat hyödyntää tätä haavoittuvuutta (Baloch 2015, ss. 186). Automaattisten skannerien toiminta pohjautuu järjestelmästä lähetettyjen vastausten tulkitsemiseen. Järjestelmälle lähetetään skannerin toimesta tietty datapaketti, johon järjestelmä vastaa aina samalla sille tyypillisellä tavalla. Tämän avulla skannerin on mahdollista saada selville avoimet portit, käyttöjärjestelmä ja versio, sekä haavoittuvuudet (Baloch 2015, ss. 121).

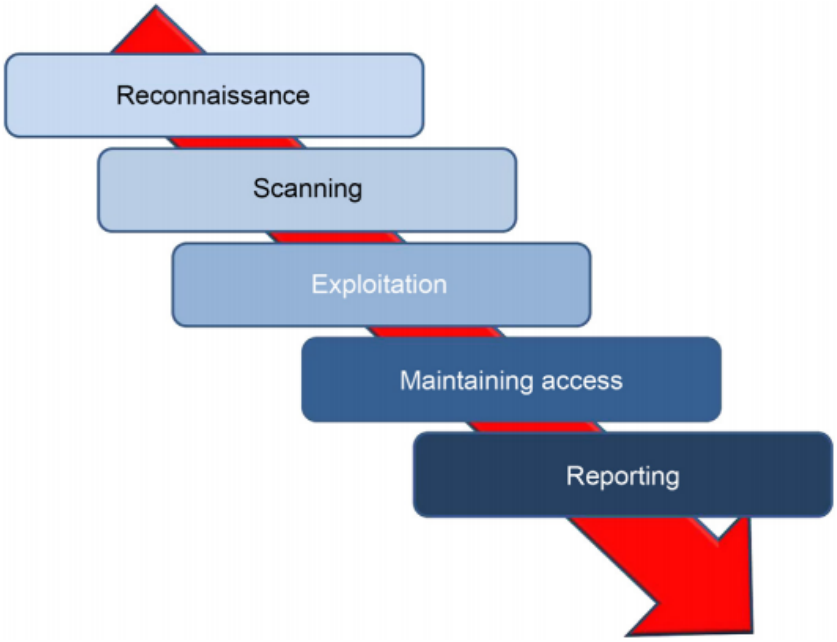
Hyökkäysvaiheessa (engl. *exploitation phase*) käydään läpi skannausvaiheessa löydetyt haavoittuvuudet ja yritetään niitä hyödyntämällä päästä järjestelmään sisään (Allen, Heriyanto ja Ali 2014, ss. 288; Weidman 2014, ss. 179). Kaikki haavoittuvuudet eivät kuitenkaan suoraan anna täysiä järjestelmänvalvoja oikeuksia, vaan voi olla, että yhden haavoittuvuuden luota avautuu ovi toiseen (Engebretson 2013, ss. 79). Hyökkäysvaihe voidaan jakaa paikallishyökkäyksiin (engl. *local exploits*) sekä etähyökkäyksiin (engl. *remote exploits*). Normaalisti hyökkäysvaiheessa aloitetaan etähyökkäyksenä, jolloin yritetään päästä sisään verkon toisessa päässä olevaan järjestelmään. Tämän jälkeen kun ollaan päästy järjestelmään sisään, aloitetaan eteneminen paikallishyökkäyksenä. Hyökkäysvaiheessa tulee muistaa testaajan oikeat eettiset ohjeet. Testaus tulee lopettaa välittömästi saavutettua asiakkaan kanssa sovittu tavoite, eikä järjestelmä tai data saa vaarantua (Mehan 2014, ss. 209–216). Haavoittuvuustestauksessa voi riittää todisteena jo järjestelmään sisään pääsyn, eikä ole tarvetta yrittää päästä

tästä eteenpäin.

Etähyökkäykset ovat hyökkäyksiä verkon yli toiseen laitteeseen tai järjestelmään. Kaikki laitteet ja järjestelmät, jotka ovat yhteydessä internettiin ovat alttiita etähyökkäyksille (Broad ja Bindner 2014, ss. 134). Paikallishyökkäyksessä testaajan täytyy olla samassa verkossa testattavan laitteen tai järjestelmän kanssa jollain tavalla (Baloch 2015, ss. 197; Broad ja Bindner 2014, ss. 133). Testattava kohde saattaa olla palomuurien takana tai suljettu avoimelta internetiltä kokonaan, tällöin testaajan täytyy saada kohteeseen yhteys esimerkiksi VPN (engl. *Virtual Personal Network*) tai SSH (engl. *Secure Shell*) tunnelin avulla (Broad ja Bindner 2014, ss. 133). Paikallishyökkäyksen toteutuksessa testaajan ei välttämättä tarvitse olla edes yhteydessä kohteeseen, sillä sosiaalisella manipuloinnilla (engl. *social engineering*) voidaan saada sama lopputulos aikaan (Baloch 2015, ss. 197–198; Broad ja Bindner 2014, ss. 133). Esimerkkinä kohdeyrityksen työntekijä, joka avattuaan haitallisen sähköpostin, suorittaa ohjelman joka antaa järjestelmänvalvoja oikeudet testaajalle.

3 Haavoittuvuustestauskaava ja tekniikat

Haavoittuvuustestauksessa pyritään etenemään tietyllä kaavalla ja käymään läpi kaikkien kohteiden mahdolliset haavoittuvuudet (Broad ja Bindner 2014, ss. 85). Broad ja Bindner (2014) esittelevät tekstissään hyvän yleisen haavoittuvuustestausvaiheet sisältävän kaavan. Tätä kaavaa kuvataan Kuviossa 1. Tutkielmassa seurataan Broadin ja Bindnerin versiota kaavasta ilman ylläpitovaihetta (engl. *maintaining access*) tai raportointivaihetta (engl. *reporting*), sillä haavoittuvuustestauksessa hyvin harvoin onnistuneen hyökkäysvaiheen jälkeen on tarvetta asentaa takaportteja järjestelmään.



Kuvio 1. Haavoittuvuustestauksen kaava. Tiedustelu, Skannaus, Hyökkäys, Ylläpito ja Raportointi (Broad ja Bindner 2014)

Haavoittuvuustestaukseen on luotu myös vakioituja kaavoja, joista on tehty standardeja. Näiden standardien avulla on mahdollista tarkastaa kohde läpi, jonka jälkeen voidaan antaa kohteelle standardin läpäissyt hyväksyntä. Yhtenä standardina voidaan pitää esimerkiksi OWASP:in ASVS (engl. *Application Security Verification Standard*). Standardien avulla voidaan asettaa suuntaviivoja haavoittuvuustestaukseen ja sovellusten kehittämiseen, jotta tes-

taukset ja kehitys suoritetaan kaikkialla samalla tavalla (OWASP-ASVS 2020, ss. 10). Yhtenä tärkeänä standardina nykypäivänä voidaan pitää PCI DSS (engl. *Payment Card Industry Data Security Standard*), joka on määritelty korttimaksamista käsittelevissä järjestelmissä tietoturvan minimitasoksi (Henry 2012, ss. 23; PCI-Security-Standards-Council 2018).

3.1 Tiedustelutekniikat

Aloittaakseen haavoittuvuustestauksen, täytyy testaajan keskustella kohdeyrityksen edustajien kanssa ja rakentaa kuva itselleen yrityksen järjestelmistä. Tämän lisäksi testaajan olisi hyvä etsiä yrityksen työntekijöiden julkisia sosiaalisen median profiileita, blogipostauksia ja nettisivuja, sillä monesti sosiaaliseen mediaan jaetaan työhön liittyviä asioita (Henry 2012, ss. 56–58). Työkaluna tähän voidaan käyttää The Harvester -nimistä ohjelmaa (Baloch 2015, ss. 69–71). Tällä voidaan löytää kohdeyrityksen sähköpostiosoitteet, nimet, aliverkkojen tunnukset, IP-osoitteet, sekä URL osoitteet erinäisistä julkisista lähteistä. Passiivista tiedustelua tehdessä voidaan käyttää myös apuna Shodan-nimistä palvelua, joka on hakukone internettiin yhdistetyille laitteille (Baloch 2015, ss. 93–95). Shodanista voidaan esimerkiksi löytää salaamattomia laitteita, käyttäjätunnuksia ja salasanoja.

3.2 Skannaustekniikat

Skannauksen ensimmäisessä vaiheessa tarkistetaan mitkä löydetyistä IP-osoitteista ovat avoinna internetille, eli saadaanko niihin yhteys. Tätä voidaan kokeilla esimerkiksi seuraavanlaisella komennolla:

```
ping target_ip
```

Ping-komento lähettää ICMP (engl. *Internet Control Message Protocol*) paketin kohteelle, mihin kohde vastaa mikäli se ei ole erikseen estetty ja kohdejärjestelmä on päällä (Engebretson 2013, ss. 57). Mikäli ICMP paketti blokataan järjestelmän tai palomuurin toimesta, voidaan myös joko TCP (engl. *Transmission Control Protocol*) tai UDP (engl. *User Datagram Protocol*) paketeilla selvittää onko kohdejärjestelmä päällä (Baloch 2015, ss. 98). Seuraavaksi selvitetään mitä portteja järjestelmässä on avoinna, esimerkiksi Netcat- tai Nmap-työkalulla (Weidman 2014, ss. 124–132). Tämän avulla saadaan selville mitä palveluita jär-

jestelmässä on käytössä, sillä tietyt palvelut käyttävät yleensä tiettyjä portteja (Henry 2012, ss. 67). Muutamia tunnettuja eri palveluiden käyttämiä portteja kuvataan Kuviossa 2.

Port Number	Service
20	FTP data transfer
21	FTP control
22	SSH
23	Telnet
25	SMTP (e-mail)
53	DNS
80	HTTP
137–139	NetBIOS
443	HTTPS
445	SMB
1433	MSSQL
3306	MySQL
3389	RDP
5800	VNC over HTTP
5900	VNC

Kuvio 2. Yleisimpiä palveluiden käyttämiä portteja (Engebretson 2013)

Tietokoneessa on 65 536 porttia ja ne jakautuvat TCP ja UDP portteihin palvelun tyypin ja käytettävän kommunikaatio metodin mukaan (Engebretson 2013, ss. 59). Näiden jokaisen portin läpikäynti olisi aikaa vievää ja myöskin erittäin äänekästä. Tämän takia moneen työkaluun on määritelty suosituimmat ja tärkeimmät portit, joita käydä läpi. Porttiskannaus voi myös pahimmillaan aiheuttaa järjestelmälle DoS-tilan (engl. *Denial of Service*), jolloin järjestelmä ei pysty enää vastaamaan oikeisiin kutsuihin, eikä täten ole käytettävissä sen oikeaan tarkoitukseen (Engebretson 2013). Tämän takia porttiskannauksessa täytyy aina olla varovainen.

3.3 Hyökkäystekniikat

Jokaiselle tietyn tyyliselle järjestelmälle on erilaiset hyökkäyslähestymistavat. Web-sovelluksia varten löydettyään käynnissä olevan web-serverin, haavoittuvuustestaaja aloittaa etsimään web-sovelluksen versioita sekä aloittaa luomaan puurakennetta nettisivuston eri alisivuista.

Puurakenteen avulla testaaja voi nähdä kaikki sivustot joita web-sovellus sisältää, jolloin testaaja voi nähdä mahdollisesti mielenkiintoisina pidettävät sivustot (esim. kirjautumissivut). Tämän jälkeen testaaja etsii mahdollisia kirjautumisikkunoita ja yrittää esimerkiksi SQL-injektiota päästäkseen sisään järjestelmään.

Tutkimuksen laajuuden vuoksi tutkimuksessa keskitytään vain Web-pohjaisiin haavoittuvuuksiin, joten Broadin ja Bindnerin (2014) Kuviossa 3 esittelemistä hyökkäysvektoreista keskitymme XSS-, CSRF- ja SQL -injektioihin.

Attack Vectors	Attack Types
Code Injection	Buffer Overflow
	Buffer Underrun
	Viruses
	Malware
Web Based	Defacement
	Cross-Site Scripting (XSS)
	Cross-Site Request Forgery (CSRF)
	SQL Injection
Network Based	Denial of Service (DoS)
	Distributed Denial of Service (DoS)
	Password and Sensitive Data Interception
	Stealing or Counterfeiting Credentials
Social Engineering	Impersonation
	Phishing
	Spear Phishing
	Intelligence Gathering

Kuvio 3. Yleisimmät hyökkäysvektorit (Broad ja Bindner 2014)

XSS on yksi eniten löydetyistä haavoittuvuuksista joka löytyy web-sovelluksista. Sen avulla voidaan injektoida koodia nettisivulle ja tämän avulla voidaan esimerkiksi manipuloida taustalla löytyvää serveriä tai nettisivun käyttäjän selainta (Muniz ja Lakhani 2013, ss. 204–205). Kaikista yksinkertaisin testi XSS-haavoittuvuuden löytämiseen on syöttää esimerkiksi hakupalkkiin seuraava teksti:

```
<script> alert("XSS") </script>
```

Mikäli paikka johon teksti sijoitettiin on haavoittuvainen, ilmestyy testaajan selaimeen ikkuna jossa lukee ”XSS”. Tämä haavoittuvaisuus perustuu käyttäjän antaman syötteen sanitoinnin ja filtteröinnin puuteeseen. Käyttäjän syöte pitää sanitoida aina ennenkuin sitä aletaan

käsittelemään, sillä muutoin web-sovellus voi luulla sitä osaksi suoritettavaa koodia (Grossman ym. 2007). Heikko sanitointi ja syötteen filtteröinti voidaan kuitenkin kiertää, jonka vuoksi testaajan on kokeiltava monia eri XSS vaihtoehtoja samaan paikkaan, esimerkiksi seuraavia (Grossman ym. 2007):

```
'><script> alert("XSS") </script>
"><script> alert("XSS") </script>
/><script> alert("XSS") </script>
<BODY onload="alert('XSS')">
<img src="" onerror="alert('XSS')">
```

CSRF-haavoittuvuus toimii käytännössä siten, että hyökkääjä pakottaa kohdekäyttäjän selaimen tekemään hyökkääjän haluaman pyynnön haavoittuvalle web-sovellukselle, esimerkiksi pyyntö vaihtaa salasana hyökkääjän haluamaksi (Baloch 2015, ss. 413). CSRF-Haavoittuvuutta testataan valitsemalla sovelluksesta esimerkiksi salasanan vaihto funktio. Tästä otetaan talteen sovelluksen käyttämä GET- tai POST-pyyntö, mihin modifioidaan haluttu lopputulos, kuten esimerkiksi haluttu uusi salasana. Tämä pyyntö liitetään sisään esimerkiksi .html tiedostoon ja kirjautuneena käyttäjänä avataan tiedosto. Mikäli sovellus on haavoittuvainen, pyyntö suoritetaan kuin normaalisti ja salasana vaihtuu. Esimerkki CSRF-haavoittuvuutta testaavasta koodista (Baloch 2015, ss. 414):

```
<form action="http://target.com/password.php" onload="this.form.submit()">
    <input name="newpass" value="12345">
    <input name="confpass" value="12345">
    <input type="submit" value="submit">
</form>
```

SQL-injektiot ovat hyökkäyksiä nettisivulta sen taustalla olevaan tietokantaan ja niissäkin korostuu käyttäjän antaman syötteen sanitointi ja filtteröinti. SQL-injektioissa annetaan SQL-syntaksilla oleva arvo tietokannalle, jolloin se ei osaa erottaa käyttäjän syötettä alkuperäisestä taustalla olevasta SQL koodista (Henry 2012, ss. 111–112). Esimerkiksi kirjautumisikkunan SQL koodi saattaisi olla seuraavanlainen:

```
SELECT name, phone FROM data WHERE name = ' ' AND password = ' ';
```

Tässä tapauksessa käyttäjän syöte tulee muuttumattomana kohtiin *name* ja *password*. Tällöin SQL-injektion voi aiheuttaa syöttämällä *name* kohtaan: *' OR 1=1;–*. SQL koodi näyttää tällöin tältä:

```
SELECT name, phone FROM data WHERE name = ' ' OR 1=1;--' AND password = ' ';
```

Tässä esimerkissä järjestelmä kirjaisi sisään ensimmäisenä käyttäjänä joka löytyy järjestelmän tietokannasta (Henry 2012, ss. 112). SQL-injektion avulla hyökkääjät voivat myös lukea tai kirjoittaa dataa tietokantaan, joten se on erittäin vaarallinen haavoittuvuus (Muniz ja Lakhani 2013, ss. 200).

4 Tekniikoiden automatisointi

Haavoittuvuustestauksen vaiheista on löydettävissä kaavamaisia ja toistettavina pysyviä kohtia. Näiden kaavamaisten töiden automatisoinnin mahdollisuuksia käsitellään seuraavissa alaluvuissa. Nykyisin automaattiset työkalut ovat yhä parempia ja hyödynnetään enenevissä määrin. Kuitenkin monien eri työkalujen käyttäminen ja niiden tulosten tarkastelu erikseen vie paljon ylimääräistä aikaa, joten yhtenä vaihtoehtona on koodata itse ohjelma joka käyttää kaikkia näitä työkaluja ja näyttää tulokset yhdessä paikassa. Eri vaiheiden automatisointi ei silti poista haavoittuvuustestaajan työtä, vaan se antaa testaajalle enemmän aikaa tutkia järjestelmää syvällisemmin ja etsiä vaikeammin löydettävissä olevia haavoittuvuuksia. Motivoituneella hyökkääjällä on aina aikaa etsiä haavoittuvuuksia, joten on tärkeää, että haavoittuvuustestaajilla on aikaa syvälliseen testaukseen.

4.1 Tiedustelun automatisointi

Tiedustelutekniikat luvussa 3.1 läpikäydyistä asioista huomaa kaavamaisen suorittamisen olevan tyypillistä haavoittuvuustestauksen toteutukselle. Esimerkiksi yrityksen sähköpostiosoitteet, sekä IP- ja URL-osoitteet ovat Harvester-työkalulla löydettävissä ilman, että testaajan täytyy syvällisemmin näitä yrittää etsiä. Maltego-työkalua voidaan käyttää myös keräämään suuret määrät OSINT-tietoja kohdeyrityksestä, jolloin testaajalle jää jäljelle vain tiedon analysointi (Weidman 2014, ss. 119–123). Shodan-palveluun on saatavilla myös API (engl. *Application programming interface*) rajapinta, jonka avulla pyyntöjä palveluun voi tehdä suoraan omasta koodista. Rajapinnan avulla on mahdollista luoda omia ohjelmia, jotka käyttävät esimerkiksi Shodanin palvelua, mutta myös käynnistävät Maltegon ja Harvesterin tietojen haun. Täten haavoittuvuustestaajan tarvitsee vain käynnistää yksi oma ohjelma, jolloin kaikkien kolmen työkalun/palvelun tietojen keräys alkaa.

4.2 Skannauksen automatisointi

Skannausvaihe on yhtälailla automatisoitavissa kuin tiedusteluvaihe. Nmap-työkalu sisältää monia valmiiksi ohjelmoituja skannausvaihtoehtoja, joiden avulla voidaan skannata esi-

merkiksi kaikki portit, mutta myös skannata ja etsiä esimerkiksi ainoastaan http-palveluita (Weidman 2014, ss. 125–132). Skannausvaiheessa voidaan ottaa tiedusteluvaiheesta löydetyt kohteena olevat IP osoitteet ja käydä näiden yleisimmät portit läpi, jotta voidaan selvittää taustalla olevat järjestelmät. Omaan ohjelmaan voidaan lisätä nmap, joka aloittaa oman skannauksensa heti kun Harvesterin ja Maltegon tulokset ovat valmistuneet. Tämä lyhentää viivettä, joka tulisi kun testaaja joutuisi manuaalisesti käymään löydetyt IP osoitteet läpi Nmapilla.

4.3 Hyökkäyksien automatisointi

Hyökkäysvaihe on kriittisimpiä vaiheita haavoittuvuustestauksessa, ja se on hyvin yksilöllinen jokaiselle järjestelmälle. Automaatio on mahdollista tiettyyn pisteeseen saakka, sillä haastavimpia haavoittuvuuksia ei voi ohjelmoida, koska ne vaativat ihmisen ongelmanratkaisutaitoja, joita ei voi koneelle opettaa. Aikaisempien vaiheiden automatisointi jättää kuitenkin aikaa paljon tarkemmalle ja syvällisemmälle hyökkäysvaiheelle.

Automaattiset skannerit kuten Nessus tai OpenVAS, voivat suorittaa yleisimpien haavoittuvuuksien etsimisen (Baloch 2015, ss. 121–122; Wilhelm 2013, ss. 211–215). Näiden työkalujen heikkoutena on kuitenkin väärät positiiviset löydöt, joten haavoittuvuustestaaja joutuu käyttämään aikaansa tarkastaakseen työkalun löytämät haavoittuvuudet (Baloch 2015, ss. 122).

Omaan ohjelmaan on mahdollista automatisoida Nmapista löytyneiden porttien perusteella hyökkäysvaiheen työkaluja. Esimerkiksi mikäli Nmap löytää portin 80 (http) avoimena, käynnistää ohjelma http-palvelulle tyypillisen etenemisen haavoittuvuustestauksessa. Ohjelma voi käynnistää esimerkiksi DirBuster-työkalun, jolla löydetään sivuston alisivut ja voidaan luoda puurakenne sivustosta. Tämän lisäksi Burp Suite- tai Owasp ZAP-työkalu voidaan käynnistää. Niillä etsitään automaattisesti nettisivun alisivuja sekä skannataan näiden sivujen mahdolliset yleiset haavoittuvuudet (Muniz ja Lakhani 2013, ss. 89–94).

Yhtenä hyökkäyskeinona web-sovelluksille voidaan myös käyttää niin kutsuttua fuzzausta, joka on käytännössä satunnaisen syötteen syöttämistä eri parametreille automatisoidusti (Weidman 2014, ss. 422–428; Wilhelm 2013, ss. 221–223). Fuzzausta voidaan käyttää myös

esimerkiksi SQL-injektioiden löytämiseen syöttämällä fuzzerilla kirjautumisikkunalle monia eri SQL-injektio vaihtoehtoja, joista mahdollisesti joku pääsee filtteröinnin ohi.

5 Yhteenveto

Tutkimuksessa havaittiin, että tiedustelu- ja skannausvaihe voidaan automatisoida miltei kokonaan sekä hyökkäysvaiheen osia on mahdollista suorittaa automaattisesti työkaluilla ja automaattisilla skannereilla. Haavoittuvuustestaajan työt eivät kuitenkaan lopu automatisoinnin johdosta, vaan automatisointi antaa testaajille aikaa tarkastella kohdetta pintaa syvemmältä. Tutkittavan aiheen kuitenkin ollessa erittäin laaja, ei tutkimuksessa päästy käsittelemään kaikkia haavoittuvuustestauksessa tarkasteltavia asioita ja web-sovelluksen haavoittuvuustestauksesta päästiin vain käsittelemään muutamaa osa-aluetta. Tutkimusstrategiana systemaattinen kirjallisuuskartoitus toimii hyvin, mutta tutkittava aihe vaatisi myös konkreettisia testejä saadakseen luotettavia tuloksia. Tästä syystä jatkotutkimuskohteeksi jää testien toteuttaminen, joiden avulla voidaan tarkastella haavoittuvuustestauksessa toteutettavien toimintatapojen toisteisuutta.

Tutkimusta voidaan jatkaa tutkimalla jo tehtyjä tai tekemällä itse ohjelmia, jotka yhdistävät monia yleisesti käytettyjä työkaluja yhdeksi työkaluksi, jolloin haavoittuvuustestaajan tarvitsee käyttää vain yhtä työkalua monen sijaan. Haavoittuvuustestauksen kaava etenee täten myös automatisoidusti eteenpäin ohjelman löydettyjen tulosten perusteella.

Lähteet

Allen, Lee, Tedi Heriyanto ja Shakeel Ali. 2014. *Kali Linux–Assuring security by penetration testing.* Packt Publishing Ltd.

Baloch, Rafay. 2015. *Ethical hacking and penetration testing guide.* Auerbach Publications.

Broad, James, ja Andrew Bindner. 2014. *Hacking with Kali: practical penetration testing techniques.* Newnes.

Engebretson, Pat. 2013. *The Basics of Hacking and Penetration Testing : Ethical Hacking and Penetration Testing Made Easy.* Nide 2nd ed. Syngress. ISBN: 9780124116443. `http://search.ebscohost.com.ezproxy.jyu.fi/login.aspx?direct=true&db=nlebk&AN=550423&site=ehost-live`.

Grossman, Jeremiah, Seth Fogie, Robert Hansen, Anton Rager ja Petko D Petkov. 2007. *XSS attacks: cross site scripting exploits and defense.* Syngress.

Henry, Kevin. 2012. *Penetration testing: protecting networks and systems.* IT Governance Publishing.

Hutchens, Justin. 2014. *Kali Linux network scanning cookbook.* Packt Publishing Ltd.

Mehan, Julie. 2014. *CyberWar, CyberTerror, CyberCrime and CyberActivism: An i-depth guide to the role of standards in the cybersecurity environment.* IT Governance Publishing.

Muniz, Joseph, ja Aamir Lakhani. 2013. *Web Penetration Testing with Kali Linux : A Practical Guide to Implementing Penetration Testing Strategies on Websites, Web Applications, and Standard Web Protocols with Kali Linux.* Packt Publishing Ltd.

Northcutt, Stephen, Cynthia Madden ja Cynthia Welti. 2004. *IT Ethics Handbook:: Right and Wrong for IT Professionals.* Elsevier.

OWASP-ASVS. 2020. *OWASP Application Security Verification Standard 4.0.2.* OWASP Foundation. `https://owasp.org/www-project-application-security-verification-standard/`.

PCI-Security-Standards-Council. 2018. *Payment Card Industry Data Security Standard, v3.2.1*. PCI Security Standards Council. https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf.

Weidman, Georgia. 2014. *Penetration testing: a hands-on introduction to hacking*. No Starch Press.

Wilhelm, Thomas. 2013. *Professional penetration testing: Creating and learning in a hacking lab*. Newnes.