jamk

Tietoturvavaatimusten ja uhkien automatisoidun testauksen suunnittelu ja toteutus

Esko Vuohensilta

Opinnäytetyö, AMK
Toukokuu 2022
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Vuohensilta, Esko

Tietoturvavaatimusten ja uhkien automatisoidun testauksen suunnittelu ja toteutus

Jyväskylä: Jyväskylän ammattikorkeakoulu. Elokuu 2021, 32 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Tavoitteena työssä oli keskittyä tietoturvallisen tuotekehityksen elinkaaren standardiin IEC 62443-4-1 ja luoda kyseisen standardin vaatimusten mukaisia tietoturvatestejä. Testattavia kohteita olisivat pääosin Valmet DCS -ohjelmiston aiemmin korjatut tietoturvahaavoittuvuudet, joista tulisi saada korjauksen varmistus. Testit luodaan myös automatisoitaviksi, joten uusien sovellusversioiden tietoturvaa voidaan jatkuvasti testata.

Avainsanat (asiasanat)

Robot Framework, SDL, Jenkins, automaatiotestaus

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Vuohensilta, Esko

Design and implementation of automated testing of security requirements and threats

Jyväskylä: JAMK University of Applied Sciences, August 2021, 32 pages.

Information and Communications. Degree programme in Information and Communications Technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

The objective of this thesis was to focus on the license on Secure Developement Lifecycle license IEC 62443-4-1 (SDL) and create security focused testing according to the license. The testing areas are mainly previously fixed security vulnerablities in Valmet DCS software that needs validation of the fix. The tests are created in automated fashion to continously test the newest software versions.

Keywords/tags (subjects)

Robot Framework, SDL, Jenkins, Automated testing

Miscellaneous (Confidential information)

Sisältö

Kuviot

Lyhenteet

SDL	Secure Developement Lifecycle
DCS	Distributed Control System
IEC	International Electrotechnical Commission
OWASP	Open Web Application Security Project
CI / CD	Continuous Integration / Continuous Development
IDE	Integrated Development Environment

1 Johdanto

1.1 Aihe ja motiivit

Opinnäytetyön tarkoituksena on suunnitella automatisoitujen tietoturvatestien elinkaari suunnittelusta aina toteutukseen asti. Testit kohdentuvat täyttämään tietoturvavaatimuksia sekä myös testaamaan mahdollisien haavoittuvuuksien olemassaoloa. Testien toteutukseen käytetään pääosin Robot Frameworkia. Työ toteutetaan Valmet Oy:lle.

Työn motiivina minulle on kiinnostus erilaisille tietoturvaa kannustaville testeille. Tietoturvatestien automatisointi voi mielestäni luoda erittäin tehokkaita ja tietoturvallisia ratkaisuja. Testien luominen todella laajaan ympäristöön on varmasti työlästä, mutta myös palkitsevaa.

1.2 Tavoitteet

Työn tavoitteena on tutkia kuinka eri tietoturvavaatimuksia ja haavoittuvuuksia voisi testata automatisoiduilla ratkaisuilla. Tutkimusta varten luodaan suunnitelma tietoturvatestauksen toteutusta varten tietoturvavaatimusten ja standardien määritelmien mukaisesti. Työssä seurattu tietoturvastandardi on IEC-62443-4-1 (SDL), joka koskee turvallisen tuotekehityksen elinkaarta ja sen parantamista. Työn aikana tutkitaan myös Robot Frameworkin toimintaa ja sen toimivuutta tietoturvatesteihin.

Osana työn tavoitetta on myös luoda yksinkertainen mutta laadullisesti hyvä suoritus automatisoidusta testistä. Testi suunnitellaan standardien mukaisesti, ja toteutetaan Robot Frameworkillä ja automatisoidaan Jenkins – automaatioserverillä. Samalla myös tutkitaan, kuinka Robot Framework luonnistuu tietoturva-aiheiseen testaukseen.

1.3 toimeksiantaja

Työn toimeksiantajana toimii Valmet Automation Oy. Valmetin tuotekehityspuolella kehitetään erilaisten automaatioiden ohjausta varten hajautettua hallintajärjestelmää eli DCS:ää (Distributed Control system). Valmet DNA soveltuu prosessinohjaukseen, laiteohjaukseen, käyttölaitteisiin ja laadunvalvontaan. (Valmet Distributed control system n.d.)

Tietoturvallisuuteen panostus on aikaisempaa tärkeämpää yrityksissä. Tämän myötä yritysten asiakkaat vaativat ostamiltaan tuotteilta entistä parempaa turvallisuutta ja laatua. Tämän takia tietoturvatestien suunnitelu ja toteuts on tarpeellista, koska se tuo kilpailuetua Valmetille. (Traficom 2018.)

2 Tietoturvallinen tuotekehitys

Tietotekniikan jatkuva kehitys on johtanut eri palveluiden, tuotannon ja myös automaatiojärjestelmien verkostoitumiseen. Monimuotoisuuden kasvaessa myös eri palveluiden hyökkäyspinta-ala kasvaa. Erilaiset kyberhyökkäykset ovat myös jatkuvassa kasvussa, joka lisää riskejä tuotekehityksen sisällä. Tietoturvallisuuteen on siis kiinnitettävä paljon huomiota nykyajan tuotekehityksessä. (Lehto, Limnéll, Kokkomäki, Pöyhönen, Salminen 2018.)

Tietoturvallisuuden huomioimistavalla on myös merkitystä tuotekehityksessä. Yksi tapa on sisäänrakennettu turvallisuus, jolla tarkoitetaan tietoturvallisuuden huomioimista koko tuotekehityksen aikana ja tietoturvallisuutta testataan ja ylläpidetään koko tuotekehityksen ajan. Täten saamme luotua tietoturvallisuuden kiinteänä ominaisuutena. Toinen vaihtoehto on korjata tietoturvapuutteet vasta tuotekehityksen loppupuolella, jossa tietoturvapuutteet korjataan vasta tuotekehityksen lopussa liittämällä uusia tietoturvaa lisäävillä lisäominaisuuksilla ja komponenteilla. (Traficom 2018.)

Tietoturvallisuuden luominen sisäänrakennetulla tavalla on paljon parempi vaihtoehto. Uusien tietoturvaa korjaavien ominaisuuksien ja komponenttien lisääminen kehityksen loppupuolella tuo lisäkustannuksia tuotekehitysvaiheeseen. Jälkeenpäin liitetyissä komponenteissa ja ominaisuuksissa ilmenee myös usein integrointiongelmia. Näiltä voidaan kuitenkin välttyä luomalla sovellus sisäänrakennetulla tietoturvallisuudella, jossa turvallisuus on valmiiksi jo olemassa. Kun tietoturvallisuus otetaan huomioon jo projektin alussa ja tuotekehittäjät käyvät tarpeelliset tietoturvakoulutukset, kehitetty sovellus pysyy turvallisena projektin loppuun asti. (Traficom 2018.)

2.1 Tietoturvavaatimukset ja uhkamallinnus

Ohjelmistovaatimuksilla tarkoitetaan toiminnallisuuksia, joita ohjelmistolta odotetaan. Sama pätee myös tietoturvavaatimuksiin. Niissä määritellään vaadittuja toiminnallisuuksia tietoturvan kannalta. Ohjelmistovaatimusten tapaisesti tietoturvavaatimukset määritellään ennen projektin kehitysvaihetta, jotta vaadittujen turvatoimien täyttäminen alkaa jo projektin alussa. Tämän myötä pääsemme luomaan sisäänrakennettua tietoturvallisuutta ohjelmistoon. toimintotapa vaatii kattavan tietoturvavaatimuksen laatimisen etukäteen. (Traficom 2018.)

Tietoturvavaatimukset voivat olla toiminnallisia tai ei-toiminnallisia. Toiminnalliset vaatimukset voivat olla esimerkiksi autentikaation vaatimista tiettyjä operaatioita varten. Toiminnalliset vaatimukset ovat yleensä nähtävillä käyttäjälle. Ei-toiminnalliset vaatimukset ovat hieman huomaamattomia, tietoturvallisten käytäntöjen noudattamista. Esimerkiksi käyttäjän syötteen validointi on ei-toiminnallinen vaatimus. Syötteen validoinnilla estetään mahdollisia koodi-injektio haavoittuvuuksia. (Traficom 2018.)

Turvallisuusvaatimusten määrittelyn yhteydessä tarkastellaan yleensä mahdollisia uhkia ja niiden korjaamista. Kun tuotteen käyttötarkoitus on tiedossa, mahdollisten uhkien miettiminen on helpompaa. Tätä prosessia kutsutaan uhkamallinnukseksi. Uhkamallinnuksen aikana pohditaan tuotteen mahdollista väärinkäyttöä, onko se mahdollista ja jos on, miten sitä voisi estää. Uhkamallinnuksessa tutkitaan myös, onko mahdollisen hyökkääjän mahdollista saada hänelle kuulumatonta tietoa palvelusta, kuten eri komponenttien tietoja ja niitten versioita. (Traficom 2018.)

Tietoturvavaatimusten vaikein osuus on niiden täyttymisen todentaminen. Tämä vaihe vaatii ekstensiivistä ja jatkuvaa tietoturvatestausta. Testeillä koitetaan todentaa, että tietoturvavaatimuksissa listatut uhat ovat täytetty. (Traficom 2018.)

2.2 Verkkosovellusten tietoturvatestaus ja OWASP

Verkkosovellusten tietoturvatestaus viittaa usein kehitetyn sovelluksen turvallisuuden testaamiseen, joten tietoturvatestaus kohdistuu vain sovellukseen itseensä. Verkkosovellusten tietoturvatestauksessa harvoin testataan verkkosovellusten ympäristöä, laitteistoa, käyttöjärjestelmiä, verkkoa tai muita näkökohtia. Verkkosovellusten tietoturvatestaus keskittyy itse sovelluksen turvallisuuteen. Tavoitteena on varmistaa, että sovellus täyttää määritetyt turvallisuusvaatimukset. (Meucci & Muller 2014. 27)

OWASP eli Open Web Application Security Project on avoimen lähdekoodin organisaatio, joka on keskittynyt verkkosovellusten tietoturvan kehittämiseen. OWASP:n yhteisöprojektit ovat tuottaneet paljon erilaisia työkaluja verkkosovellusten tietoturvan edistämistä varten, kuten artikkeleita, työkaluja, kirjoja ja muuta teknologiaa. Merkittävin ja eniten viitattu projekti on kuitenkin OWASP top 10 -lista. (OWASP 2021.)

Nykyajan verkkosovellusten kehityksessä OWASP top 10:stä on tullut standardi verkkosovellusten suojauksessa. OWASP top 10 listaa 10 yleisintä verkkohaavoittuvuutta verkkosovelluksissa. OWASP top 10 on kehitetty tietoisuuden lisäämiseksi tietoturvahaavoittuvuuksista ohjelmistoteollisuuden kasvaessa. OWASP top 10 listan tavoitteena oli alun perin lisätä tietoisuutta yleisimmistä verkkosovellusten haavoittuvuuksista, mutta ajan myötä siitä on tullut standardi organisaatioille verkkosovellusten tieoturvatestausta varten. Vuoden 2021 päivitetyn listan top 10 haavoittuvuutta:

rikkoutunut autentikointi- ja istuntahaavoittuvuudet (Broken Access Control)

Rikkoutuneessa autentikointi- ja istuntohaavoittuvuuksissa todennettujen käyttäjien sallittuja rajoituksia ei usein aseteta asianmukaisesti käytäntöön. Hyökkääjät voivat hyväksikäyttää näitä puutteita päästäkseen käsiksi arkaluonteisiin tiedostoihin tai käyttääkseen luvattomia toimintoja. 94% testatuista web-sovelluksista sisälsi jossain määrin rikkoutuneen kirjautumis- tai pääsynhallinnan, tekien rikkoutuneen autentikoinnin ja istuntohaavoittuvuuksen suurimmaksi verkkosovellusten haavoittuvuudeksi. (OWASP 2021.)

Salausvirheet (Cryptographic Failures)

Jos verkkosovelluksessa käsitellään arkaluonteisia tietoja kuten talous- ja henkilötietoja, tulisi tiedot suojata asianmukaisesti sekä tallennetussa muodossa että tiedonsiirron aikana. Asianmukainen arkaluonteisten tietojen käsittely vaatii tarpeeksi hyvää salausta tietojen käsittelyn aikana ja niitten tallennukseen. (OWASP 2021.)

injektio (Injection)

injektio on haavoittuvuus, missä palvelin kysyy käyttäjältä syötettä, kuten vaikka kirjautumistietoja tai hakuparametria. Oletetun vastauksen sijasta käyttäjä antaakin palvelulle käskyn, jonka palvelu suorittaa tietämättään. Yleisin injektioille haavoittuvainen palvelu on SQL -palvelin.

2021 päivitetyssä OWASP top 10 -versiossa injektio-kategoriaan sisältyy myös SQL injektioiden lisäksi cross-site scripting. Cross-site scripting tarkoittaa tapahtumaa, missä hyökkääjä pääsee sisällyttämään haitallista koodia verkkosovelluksen sivulle itseensä. Tämän jälkeen haitallinen koodi pystyy ajamaan uhrien koneilla niitten avatessa verkkosovelluksen. (OWASP 2021.)

Epäturvallinen suunnittelu (Insecure Design)

Epäturvallinen suunnittelu on uusi kategoria vuoden 2021 listalle, ja se keskittyy suunnitteluvirheisiin järjestelmän ytimessä. Nykyaikana turvallisten verkkosovellusten kehitys tulee aloittaa ajoissa ja jatkaa koko kehityksen aikana. Kattava uhkamallinnus, turvalliset suunnittelumallit- ja periaatteet sekä muut hyvät käytänteet ovat tärkeitä verkkosovellusten kehitysprosessissa. (OWASP 2021.)

Turvattomat konfiguraatiot (Security misconfiguration)

Verkkosovellusten turvattomat konfiguraatiot ovat yleisimmin löydetty haavoittuvuus. Se johtuu yleensä turvattomista konfiguraatioista, oletusasetusten jättämisestä, avoimesta pilvitallennuksesta, väärin määritellyista HTTP -otsikoista ja arkaluonteisia tietoja sisältävistä virheilmoituksista. Turvallisten konfiguraatioiden määrittely tulee kattaa kaikki järjestelmät, kehykset, kirjastot ja sovellukset sekä niitä tulee päivittää jatkuvasti tarpeen tullessa. (OWASP 2021.)

Haavoittuvat ja vanhentuneet komponentit (Vulnerable and outdated components)

Kirjastot ja muut ohjelmistomoduulit toimivat usein samoilla oikeuksilla kuin itse sovellus. Jos haavoittuvaa komponenttia tai ohjelmistomoduulia hyväksikäytetään, hyökkäyksessä voidaan menettää arkaluonteisia tietoja tai jopa koko palvelin voidaan kaapata. Ohjelmistot, jotka käyttävät haavoittuvaisia tai vanhentuneita osia altistuvat isommalle tietoturvariskille laajentamalla hyökkäyspinta-alaansa eli mahdollistavat erilaisia hyökkäyksiä ja vaikutuksia. (OWASP 2021.)

Tunnistus- ja todennusvirheet (Identification and authentication failures)

Istunnonhallintaan ja käyttäjien todentamiseen liittyvät sovelluksen toiminnot toteutetaan usein väärin, jolloin hyökkäjät voivat vaarantaa salasanoja tai istuntotunnuksia tai hyödyntää muita toteutusvirheitä olettaakseen muiden käyttäjein henkilöllisyyden tilapäisesti tai pysyvästi. (OWASP 2021.)

Ohjelmistojen- ja tietojen eheysvirheet (Software and Data Integrity Failures)

ohjelmistojen- ja tietojen eheysvirheet on uusi kategoria vuoden 2021 listalle, ja se käsittelee eri ohjelmistojen yhteisestä toiminnasta. Eheysvirheitä tapahtuu, kun eri komponentit olettavat tärkeitä tietoja, kuten järjestelmäversioita, eikä tietojen eheyttä voida varmistaa. Tietojen eheysvirheet voivat mahdollistaa virheitä tuotekehityksen automatisoiduissa palveluissa laajentaen hyökkäyspinta-alaa. (OWASP 2021.)

Tapahtumien kirjaamis- ja seurantahäiriöt (Security logging and Monitoring Failures)

Tapahtumien kirjaamis- ja seurantahäiriöt usein viittaavat puutteelliseen tapahtumien seurantaan tai tehottomaan seurannan konfigurointiin. Tämä mahdollistaa mahdollisten hyökkäysten huomioimisen virheitä, jos vaikka tiedostoja poistetaan tai muokataan, sitä ei huomata tai hyökkäys huomataan liian myöhään. Useimmat rikkomustutkimukset osoittavat, että tietoturvahyökkäysten havaitsemiseen kuluva aika on yli 200 päivää. (OWASP 2021.)

Palvelinpuolen pyyntöjen väärentäminen(Server-side Request Forgery)

Palvelinpuolen pyyntöjen väärentämisessä hyökkääjä pystyy lähettämään muokattuja HTTP-pyyntöjä palvelimen sisäverkossa oleville komponenteille. Tämän myötä ulkoverkon kautta voi päästä yhdistämään suojatun sisäverkon sisällä komponentteihin. Tämän myötä voi hyökkääjä saada arkaluonteista tietoa palvelusta ja aiheuttaa vahinkoa. (OWASP 2021.)

2.3 SDL standardi

Valmetin tuotekehityksessä on keskitytty suuresti tietoturvaan. Valmetin tuotekehitys on ulkoisesti sertifioitu standardien ISO27001 ja IEC 62443-4-1 vaatimusten mukaisesti. Tässä työssä keskitytään ylläpitämään ja parantamaan IEC 62443-4-1 standardin mukaisia vaatimuksia ja niiden todentamista. (Valmet Distributed control system n.d.)

IEC eli International Electrotechnical Commission on kansainvälinen sähköalan standardiorganisaatio. IEC on kehittänyt yli 10 000 eri standaria eri sähköalojen toimintatapoihin. Heillä on monia eri standaredja liittyen tietoturvaan, IEC 62443 standardi on yksi niistä. Tässä työssä keskitytään vain IEC 62443 standardin osaan 62443-4-1.

IEC 62443 standardi käsittelee teollisuuden automaatio, ja ohjausjärjestelmien (IACS tai Industrial Automation and Control Systems) tietoturvallisuutta. 62443-4-1 standardin osassa määritellään prosessivaatimukset teollisessa automaatiojärjestelmässä käytettävien sovellusten ja ohjausjärjestelmien turvalliselle tutkimukselle ja kehitykselle. (IEC 62443-4-1:2018, 11.)

Standardissa esitetään tietoturvallisen tuotekehityksen elinkaaren vaatimukset tuotteille, jotka on tarkoitettu käytettäviksi teollisuusautomaatio- ja ohjausjärjestelmien ympäristössä. Tietoturvallisen tuotekehityksen elinkaarta viitataan SDL-prosessina (Secure Development Lifecycle). Standardin mukaisessa tietoturvallisessa tuotekehityksessä tulee huomioda kyberuhkia jo tuotekehityksen alusta asti. Valmetin tuotekehityksessä on käytössä turvallista tuotekehitysprosessia varten defence-in-depth -menetelmä, jonka mukaan mahdollisiin kyberuhkiin kiinnitetään huomiota jo tuotekehitysprosessin alussa ja koko prosessin aikana. (Valmet Distributed control system n.d.)

Tämä tietoturvallinen tuotekehitysmenetelmä vaatii paljon testien rakentamista monesta eri syystä. Mahdollisten kyberuhkien olemassaoloa voidaan todentaa testaamalla, ja haavoittuvuuksten korjauksia tarvitsee myös todentaa, jotta voidaan olla varmoja, että haavoittuvuus on korjattu. Tuotteen päivittyessä testausta tulee jatkuvasti toistaa, jotta uusimpien sovellusversioiden tietoturvallisuus voidaan myös todentaa, koska pienetkin muutokset olemassa olevaan sovellukseen saattaa muuttaa sen tietoturvallisuutta. (Koskinen 2019, 10.)

3 Työkalut

Tässä kappaleessa listataan työkalut, joita käytetään työn aikana. Nämä työkalut tulevat olemaan käytössä tulevaisuudessa tehtävissä tietoturvatesteissä, jotka luodaan suunnitellun ohjeen mukaisesti. Nämä työkalut ovat myös käytössä esimerkkitoteutuksessa. Työkalujen asennusta ja käyttöönottoa ei tässä työssä tutkittu.

3.1 Robot Framework

Robot Framework on työssä käytetty testauskehys. Robot Framework on kirjoitettu Python -ohjelmointikielellä vuonna 2005 diplomityönä Pekka Klärchin toimesta. Klärch kehitti Robot Frameworkia pidemmälle Nokia Siemensillä. Robot Framework tuli myöhemmin saataville avoimena lähdekoodina vuonna 2008 Apache License 2.0 -lisenssillä. (robotframework.org)

Robot Framework käyttää avainsana-pohjaista (keyword) lähestymistä ohjelmoinnissa luotettavuuden ja helpomman luomisen auttamiseksi. Avainsanapohjaisen ohjelmointikielen kokonaisuuden voi jakaa 4 eri vaiheeseen: Test step, Test object, action ja test data.

Test step kuvaa eri toimintojen nimiä, joiden toiminta muistuttaa yleisimpien ohjelmointikielien funktioita. Test steppien luomisessa annetaan nimi, dokumentoidaan toiminta kommenttina ja pyydetään tarvittaessa argumentteja toimintoja varten. Test objectin voi määrittää esim. regexillä, hakea objektin xpath tai määritellä halutun elementin css-elementti. Test object voi olla esimerkiksi Käyttäjänimi -kenttä kirjautumissivulla tai painettava näppäin. (robotframework.org)

Toinen vaihe on Test Object. Test objectilla viitataan Robot Frameworkin toimimiseen testattavan kohteen kanssa. Test objectissa määritellään web-sovelluksessa olevan objektin tai elementin nimi. Robot Framework etsii halutun objektin käyttäjän antaman argumentin avulla. (robotframework.org)

Kolmas vaihe on action, jolla viitataan suoritettavaan toimintoon. Robot Framework voi suorittaa samoja toimintoja kuin normaali verkkosovelluksen käyttäjä, kuten avata seilaimen, syöttää dataa kenttään tai klikata näppäintä.

Viimeinen vaihe on test data. Test datalla viitataan toimintoihin vaadittua dataa. Esimerkiksi käyttäjänimi -kenttään syötettävä string on test dataa. (robotframework.org)

Robot Frameworkillä on monia testikirjastoja ja muita työkaluja käytettävissä. Tobot Frameworkia voidaan käyttää verkkosovellusten testaukseen, mutta sillä voi myös testata esimerkiksi ftp-, mongodb-, android- ja muita sovelluksia. Robot Frameworkiin voi myös luoda omia kirjastoja tarvittaessa. (robotframework.org)

3.2 Jenkins

Työssä käytetty automaatiopalvelin on Jenkins. Jenkinsin kehitys alkoi alun perin Hudson -projektina. Hudsonin luominen alkoi 2004 Sun Microsystemsissä. Myöhemmin vuonna 2010 Sun Microsystems ostettiin osaksi Oraclea, jonka myötä Hudson -projektin tavaramerkki siirtyi Oraclen omaisuudeksi. vuonna 2011 alkuperäinen Hudson -projekti nimettiin käyttäjien äänestyksen jälkeen Jenkins -projektiksi. (Proffitt 2011.)

Jenkins on Java -ohjelmointikielellä kirjoitettu avoimen lähdekoodin jatkuvan integraation ja jatkuvan toimituksen (CI / CD) palvelin. Jenkinsiä käytetään ohjelmistoprojektien jatkuvaan rakentamiseen ja testaamiseen, jolloin kehittäjien on helpompi integroida muutoksia projektiin ja käyttäjien on helpompi saada projektin uusimpia versioita. (What is Jenkins 2022.)

Jenkins mahdollistaa jatkuvan integroinnin laajennusten tai pluginien avulla. Laajennukset mahdollistavat erilaisten vaiheiden automatisointia rakennusprosessin tai testauksen yhteydessä. Jenkinsiin on olemassa yli 1800 Jenkins-yhteisön luomaa laajennusta, jotka ovat suoraa asennettavissa jenkins -palvelimelle. Lisäosat mahdollistavat suosittujen työkalujen integroinnin rakentamisen yhteyteen, esim. Git versionhallinta, Maven 2-projekti tai Amazon Pilvipalvelut. (What is Jenkins 2022.)

Jenkins -jobien automatisointiin on eri tapoja, joista tulee valita eri automatisointeihin sille sopiva ajoehto. Yksi tapa automatisoida Jenkins jobeja on luoda ajoehto, joka ajaa Jenkins jobin siihen linkatun jobin suorittuessa, esimerkiksi viimeisimmän ohjelmistoversion rakentamisen jälkeen. Toinen tapa olisi ajastaa testien ajo esimerkiksi joka yölle tai viikonlopuille. (What is Jenkins 2022.)

3.3 Jira / Atlassian

Jira on työssä käytetty tehtävienhallintaohjelmisto. Jira on alun perin kehitetty vuonna 2002 bugi- ja issueiden hallintaan. Nykyään Jira ja sen kattava valikoima erilaisia lisäosia toimii tehokkaana ohjelmistona testien suunnittelussa, hallinnassa ja automatisoinnissa. Jiralla voi suunnitella kattavia elinkaaria ohjelmistokehitykselle ja testaukselle. (Morpus 2022.)

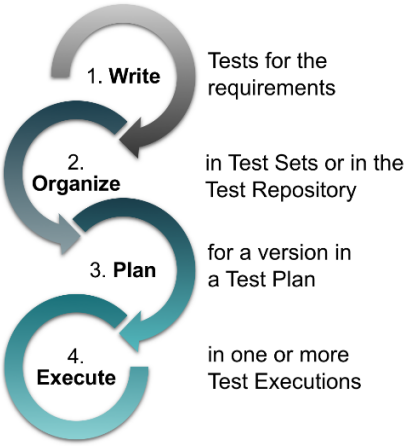
Tietoturvatestejä varten luodaan Jirassa tikettejä, jotka sisältävät tarvittavat tiedot testin rakentamista varten. Testien tulosten perusteella voidaan luoda jälkitoimenpiteitä varten taskeja ja linkittää siihen alkuperäisen testin tiketti ja testin tulokset. Nämä tiketit osoitetaan suoraan tuotekehittäjille, jolloin testien tulosten perusteella voidaan korjata mahdolliset haavoittuvuudet tai bugit nopeasti.

3.4 Xray

Testauksen dokumentointiosuuteen käytetään avuksi Jiraan liitettyä Xray – laajennusta. Xray tukee tämän työn mukaisen tietoturvatestin dokumentoinnin koko elinkaarta: testin määrittelyvaiheesta suunnitteluun, suorittamiseen ja raportointiin asti. (About Xray n.d.)

Xray hyödyntää Jiran eri issue – tyyppejä erityyppisten dokumentointien kategorisoimiseen. Xray tuo uusia issue- tyyppejä Jiraan (kuvio 1), joista käytetään tässä työssä pääosin kolmea eri issue-tyyppiä: test plan, test design ja test execution. Xrayn eri Jira issue- tyyppejä voidaan luomisen yhteydessä linkata toisiinsa testin eri osien yhdistämistä varten. (About Xray n.d.)

.



kuvio 1 Eri xray issue tyyppejä

3.5 Versionhallinta

Git on työssä käytetty versionhallintajärjestelmä. Git on erittäin laajasti käytetty avoimen lähdekoodin projekti, joka kehitettiin alun perin vuonna 2005 Linus Torvaldsin toimesta, joka on kehittänyt myös esimerkiksi Linux -käyttöjärjestelmän ytimen. Git on loistava versionhallinta moniin eri projekteihin, ja se toimii hyvin monilla käyttöjärjestelmillä ja tekstieditoreilla tai IDE:illä (Integrated Development Enviornments). (What is git n.d.)

Valmetin Robot Framework kirjastot ovat tallennettu gitissä sijaitsevaan koodikantaan, jonka voi ladata omalle tietokoneelle ja tehdä olemassa olevia testejä, tai luoda uusia testejä ja tallentaa muutokset tietokantaan vetopyynnön tai pull requestin kautta. Tämä vaihe takaa testin eheyden tarkastamalla koodin muiden henkilöiden kautta ennen koodikantaan tallentamista. (What is git n.d.)

3.6 muut käytetyt työkalut / kirjastot

Python

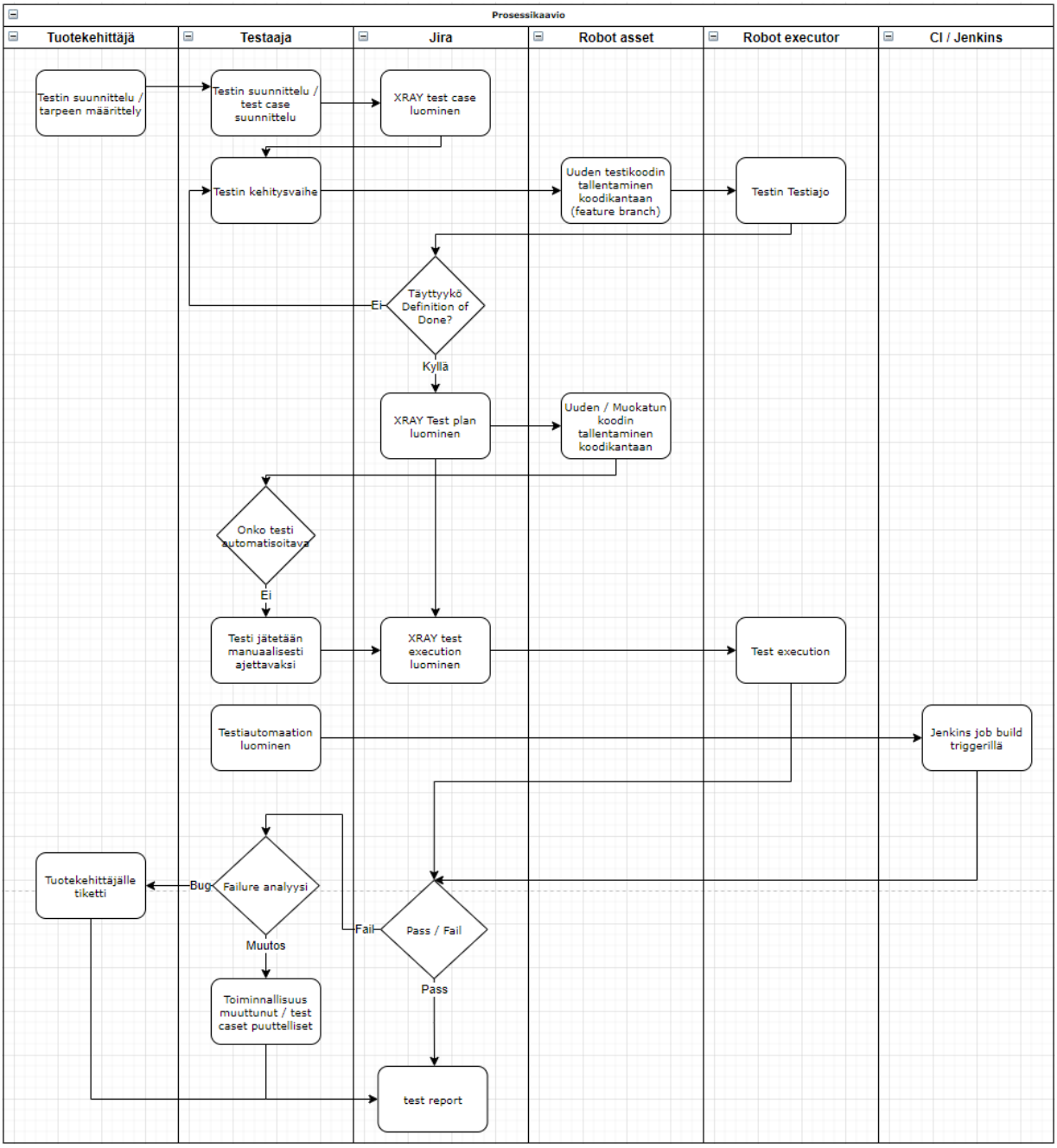
Python on suosittu ylemmän tason koodauskieli, joka on erittäin tehokas esimerkiksi tuotekehitykseen tai scriptaukseen. Python sisältää monia eri kirjastoja eri käyttötarkoituksiin, joka tekee siitä monipuolisen ja loistavan työkalun scriptaukseen. Pythonin käyttö on pääosin tarkoitettu vaiheille, jotka ovat vaikeita Robot Frameworkille, Esimerkiksi satunnaisen sisällön tai payloadien luomiseen. (What is python n.d.)

Chromedriver

Chromedriver on avoimen lähdekoodin työkalu, jolla voi automatisoida web-sovellusten testausta käyttäen eri selaimia. Chromedriverilla voi navigoida web-sovelluksen layouttia, syöttää tekstiä, ajaa erilaisia komentoja ja paljon muuta. Robot Framework käyttää Chromedriveria avaakseen ja käyttääkseen selainta testien yhteydessä. (Chromedriver n.d.)

4 Tietoturvatestauksen suunnittelu

Tässä kappaleessa käydään läpi tietoturvatestauksen koko elinkaari suunnittelusta ja testin kehitysvaiheesta testin käyttöönottoon ja raportin luomiseen. Testauksen suunnitteluun käytetään apuna kuviossa x olevaa prosessikaaviota, jonka vaiheet kuvaillaan tässä kappaleessa. Erilaisten testien prosessi ei välttämättä seuraa tarkasti tätä elinkaarta, mutta tämä kaavio on hyvä suuntaa antava esimerkki.

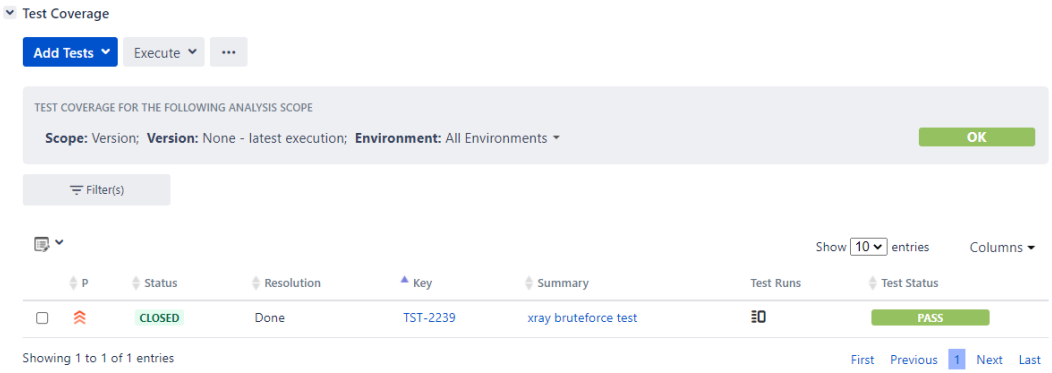


Kuvio 2 Testin suunnittelun prosessikaavio

4.1 suunnitelman teko

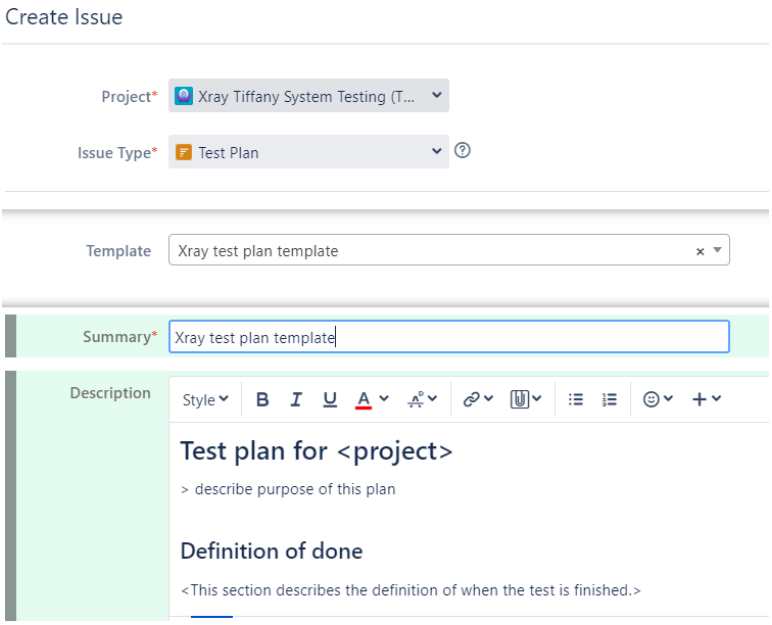
Testien toteutus alkaa suunnitteluvaiheesta. Testin tarpeen määrittelee ohjelmistosuunnittelija ja/tai testaaja. Testiä varten luodaan Jirassa Xray test design- issue tyyppi suoraan testin suunnittelu- issuesta Xrayn tarjoaman test coverage- osuudesta (kuvio 3). Test designiin liitetään tarvittava määrä test caseja. Test case on osa testin kokonaisuutta, joka pitää sisällään yhden vaiheen

tai osan testistä. Test casessa määritellään testin kohde, tarkemmat määritelmät, oletettu lopputulos ja Definition of Done. Definition of Done tässä tapauksessa määrittelee vaatimukset valmiille test caselle. Test casen tulee sisältää tarpeeksi informaatiota, jotta Definition of done voidaan täyttää.



kuvio 3 Test coverage

Testin suunnittelun jälkeen voidaan alkaa kehittämään testiä. Kehitysvaiheessa määritellään testin työympäristö ja käytetyt ohjelmistot. Testin suunnittelun aikana luodaan tarvittava määrä test caseja testiä varten. XRAY test planeja varten on luotu Jirassa templaatti (kuvio 4), joka sisältää formaatin, jota noudatetaan test planeja suunitellessa.

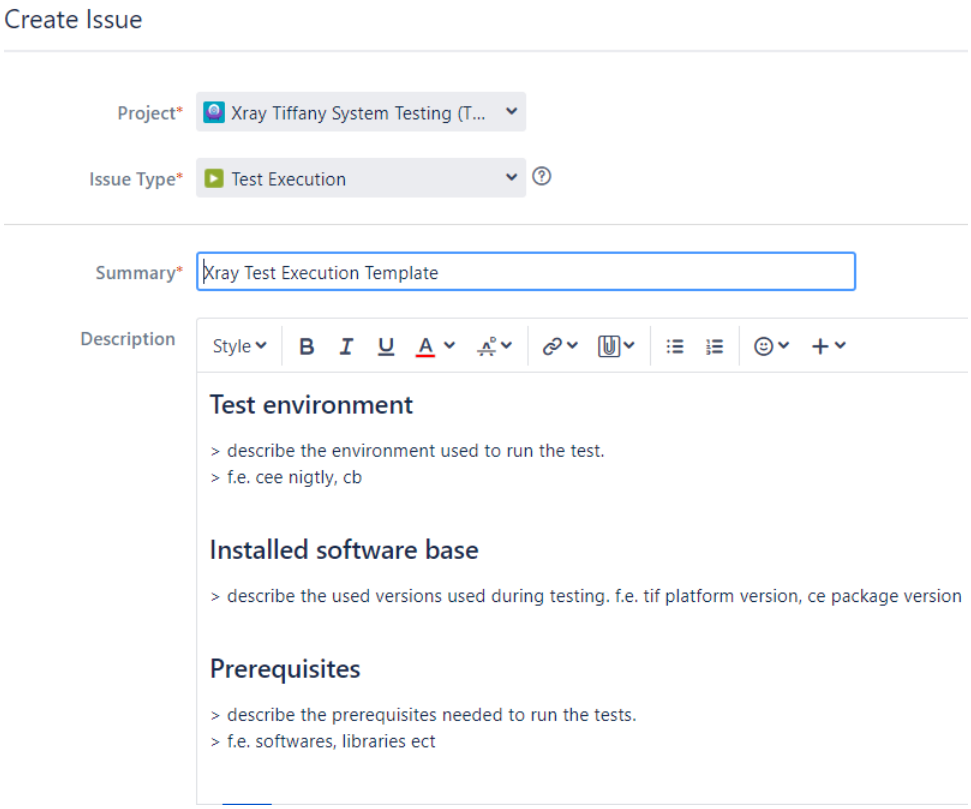


kuvio 4 Xray test plan template

suunnittelun aikana voidaan myös suunnitella ajoissa ominaisuusarviointeja testille, joissa voidaan pohtia mahdollisia lisäominaisuuksia testille tai eri kohteita testille. Ennen testin kehitysvaiheesta poistumista on vielä hyvä käytäntö suorittaa alkeellista testausta suljetussa ympäristössä ennen testikoodin tallentamista koodikantaan. Testin testiajosta saadaan selville, täyttyykö eri testien osien definition of done. Jos definiton of done ei käyty, test caseja tai itse testiä tulee päivittää.

4.2 Testin käyttöönotto

Test planin luonnin jälkeen päätetään, onko testille tarvetta automatisoida. Jotkut testit jätetään ajettavaksi manuaalisesti, joka voidaan asettaa ajettavaksi suoraa Jirasta XRAY test executionin avulla. XRAY test execution luominen tapahtuu templatesta (kuvio 5), jossa kuvaillaan testausympäristö, testissä käytetyt ohjelmistoversiot ja testin vaatimat prekvisiitat eli vaaditut ohjelmistot. XRAY test executen luomisen jälkeen tulee linkata test executioniin test planit.



kuvio 5 XRAY test execute template

Jos testillä on tarvetta automatisoinnille, sitä varten luodaan Jenkinsissä testiä varten projekti. Projektin ajoa varten määritellään joitain Jenkins -nodeja, joilla testit suoritetaan. Testin Automatisointia varten testille luodaan build trigger eli ajoehto. Yleinen tapa luoda Build trigger on putkittaa testi ajettavaksi esimerkiksi uuden ohjelmistoversion rakentamisen jälkeen.

4.3 Raportointi

Testin ajon jälkeen päästään tutkimaan testien tuloksia. Suurin osa testin tuloksista tulee selville tutkimalla Robot Frameworkin tuottamia lokeja ja raporttia. Ne pitää sisällään yhteenvedon testistä sekä yksityiskohtaiset vaiheet eri test caseista.

Jos testin tuloksista on selvää, että testi onnistui eikä mitään poikkeuksia löytynyt, luodaan testistä raportti. Raportti pitää sisällään testatun ohjelmiston version ja testin olennaiset osat testin tuloksista. Jos samalla testillä myöhemmin löydetään poikkeamia tai haavoittuvuuksia, voidaan ohjelmiston eri versioita verrata toisiinsa ja löytää ongelmat.

Jos testi epäonnistui, testaaja suorittaa vika -analyysin testin tulosten perusteella. Vika-analyysin tuloksena on yleensä joko poikkeaman tai haavoittuvuuden löytäminen, josta luodaan bug – tiketti kehittäjälle. Toinen mahdollisuus testin epäonnistumiselle on toiminnallisuuden muuttuminen. Toiminnallisuuden muuttumiselle on 3 mahdollisuutta:

1. Testitapaukset ovat vanhentuneita, ne sisältävät vanhoja tietoja tai tulokset ovat vääriä.

Testitapausten vanhentumisella viitataan siihen, että test caset ovat vanhentuneita tai sisältävät vanhaa tai väärää tietoa.

2. testausympäristö on muuttunut

Testausympäristön muuttumisella voi olla vaikutusta testin tuloksiin. Jos olemassa olevaa testiympäristöä on muutettu vaikuttavasti, testitulokset väärentyvät tai testi epäonnistuu täysin. Näitä muutoksia voivat olla esimerkiksi yhteysongelmat testiä suorittavan koneen ja palvelinten välillä tai nimipalvelimen muutokset. Testiympäristö on myös saatettu vaihtaa täysin uuteen.

3. epäselviä / vajaita testitapauksia

Jos testitapaukset ovat vajaita tai epäselviä, rakennettu testi on myös vajaa, kaikkia testille tarkoitettuja kohteita ei ole mainittu, tai testimenetelmiä ei ole kuvailtu tarpeeksi hyvin. Tämän seurauksena testin tuloksista ei saa mitään hyödyllistä palautetta.

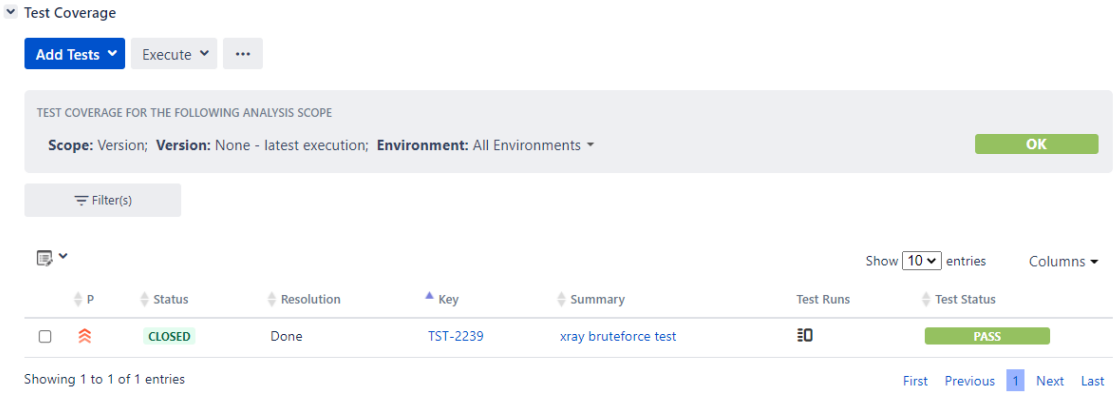
Vika – analyysin jälkeen luodaan testiraportti, joka sisältää poikkeamien kuvailun ja jälkitoimenpiteiden suunnittelua. Raportti sisältää testin eri test caset ja niiden statuksen, josta selviää mitkä test caset aiheuttivat ongelmia. Vika – analyysin jälkeen luotuihin tiketteihin tai storyihin tulee myös linkittää testiraportti.

5 Esimerkkitoteutus Testauksesta

Työn automatisoidun testin suunnittelun ja testauksen esimerkkitoteutukseen valitaan kirjautumispalvelussa olleeseen haavoittuvuuteen, joka mahdollisti bruteforce- hyökkäyksen palvelimen kirjautumislomakkeessa. Kyseinen haavoittuvuus osuu OWASP -listan mukaan yleisimpään verkkosovellushaavoittuvuuteen, rikkoutuneeseen autentikointi -ja istuntohaavoittuvuuteen. Kyseessä on siis aiemmin korjatun haavoittuvuuden mitigoinnin testaus. Testin tarkoituksena oli todeta, että aiemmin havaittu ja korjattu kirjautumispalvelun hyväksikäyttö bruteforce -hyökkäykseltä on korjattu.

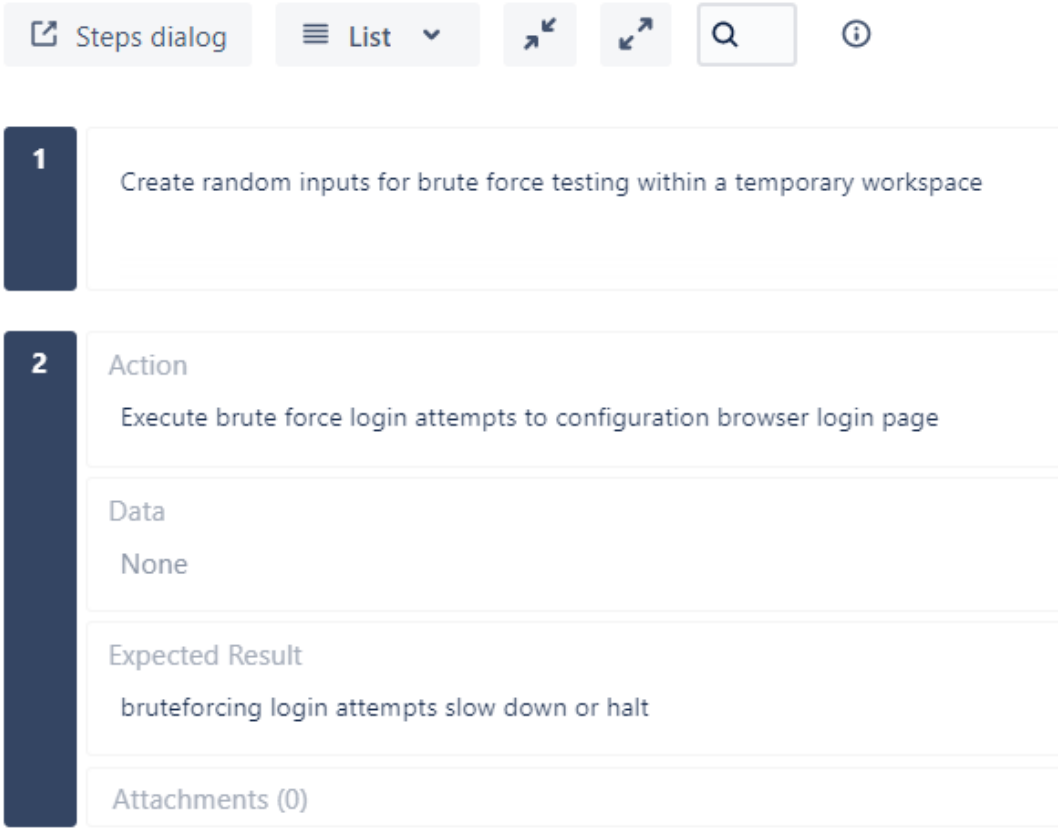
5.1 Testin suunnittelu

Testien suunnittelu aloitetaan Jirassa haavoittuvuuden alkuperäisestä määrittelyn sisältävästä tiketistä. Jira Xray -lisäosan avulla luodaan tiketille test coverage (kuvio 6), jonka kautta luodaan alkuperäiselle tiketille Xray test design. Kyseinen test design on myös täten linkattuna alkuperäiseen tikettiin. Tässä tilanteessa test planina pidetään haavoittuvuuden alkuperäistä Jira- tikettiä, jossa testin tarve on määritelty.

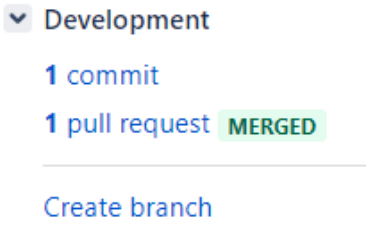


kuvio 6 Test coverage

Test designia varten luodaan joukko test caseja, jotka kuvailevat prosessin ja oletetun tuloksen (kuvio 7). Test designissa voidaan myös kätevästi luoda Git- versionhallintaan feature- haara testin koodia varten (kuvio 8). Tällöin Jiraan jää pysyvä linkki kyseiseen haaraan.



kuvio 7 Test caset



kuvio 8 Feature- haaran luominen Jirassa

5.2 Testin kehitysvaihe

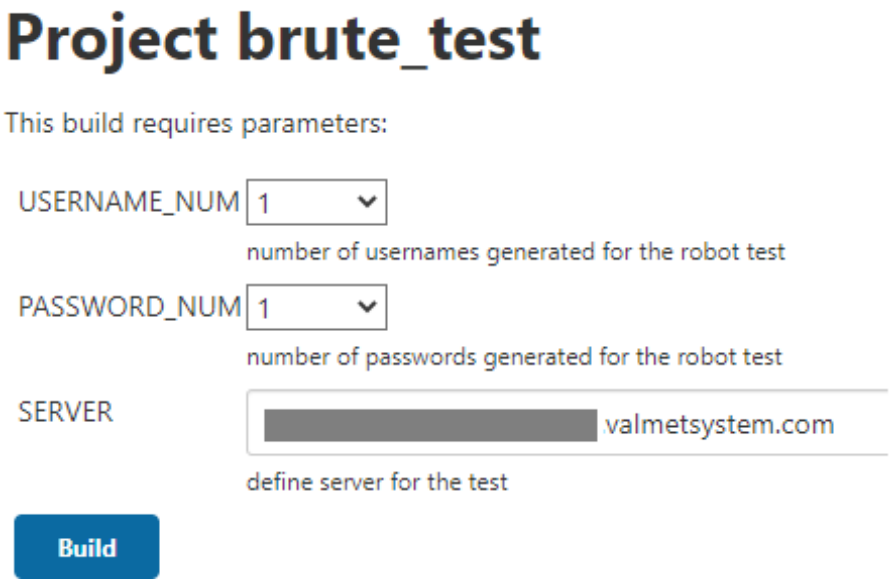
Testin kehitysvaiheessa siirrytään koodikannassa testiä varten tehdylle feature- haaralle, jonne tallennetaan testin eri osat. Kun kehitysvaiheen lopussa tulokseen ollaan tyytyväisiä, voidaan uusi testikoodi tallentaa koodikantaan pysyvästi. Tätä prosessia varten tulee hyödyntää Gitin pull request- työkalua, jonka avulla luodaan tallennuspyyntö tietokantaan. Tallennuspyyntöön lisätään muita testaajia, jotka tarkastavat kyseisen tallennuspyynnön sisällön, ja voivat joko hyväksyä pyynnön tai pyytää muutoksia tallennettavaan koodiin. Tässä työssä koodikantaan tallennetaan kaksi osaa: Robot Framework- koodi sekä pieni avustava Python- scripti koodia varten.

Ennen pull requestia on kuitenkin tarpeellista päivittää työn feature- haara ajan tasalle virallisen koodikannan kanssa. Tämänkaltaisessa koodikannassa, jossa on monia eri sisällöntuottajia, tapahtuu usein muutoksia. Jos työn feature- haaraa koitettaisiin tallentaa suoraan koodikantaan ilman haaran päivittämistä, tapahtuisi todennäköisesti konflikteja, koska vanha feature- haara ei sisältäisi muutoksia, joita muut testaajat ovat tallentaneet koodikantaan feature- haaran ottamisen jälkeen.

5.3 Testin suorittaminen

Tämän työn Robot Framework- koodin ajoa varten testaaja syöttää testattavan kohteen ip-osoitteen sekä bruteforcessa käytettävien käyttäjänimien ja salasanojen määrän. Oletuksella koodi luo 10 käyttäjänimeä ja salasanaa, mutta tarvittaessa voidaan kumpaakin luoda niin monta kuin tarpeellista. Robot- koodi käy läpi jokaisen mahdollisen yhdistelmän käyttäjänimiä ja salasanoja, ja pyrkii kirjautumaan tällä datalla kohteeseen käyttäen login- lomaketta.

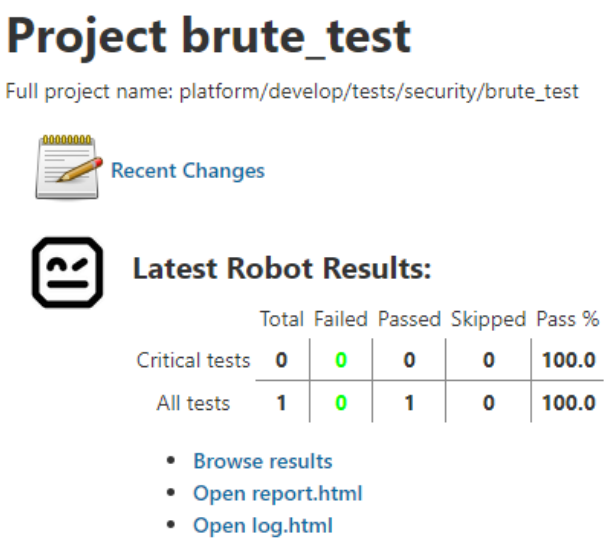
Testin suorittamista varten luodaan Jenkinsiin freestyle- job. Jenkins- job konfiguroidaan käyttämään koodikantaa, johon työn koodi on tallennettu. Koodille luodaan myös säädettäviä parametrejä, jotka voidaan ajon yhteydessä määritellä. Testissä käytettävät parametrit ovat käyttäjänimien ja salasanojen lukumäärä, sekä testauksen kohteena oleva palvelin (kuvio 9). Viimeiseksi Jenkins konfiguroidaan tallentamaan Robot Frameworkin generoimat dokumentit testistä sekä näyttämään testin tuloksen suoraan Jenkinsin käyttöliittymässä.



kuvio 9 Testin parametrien määrittely

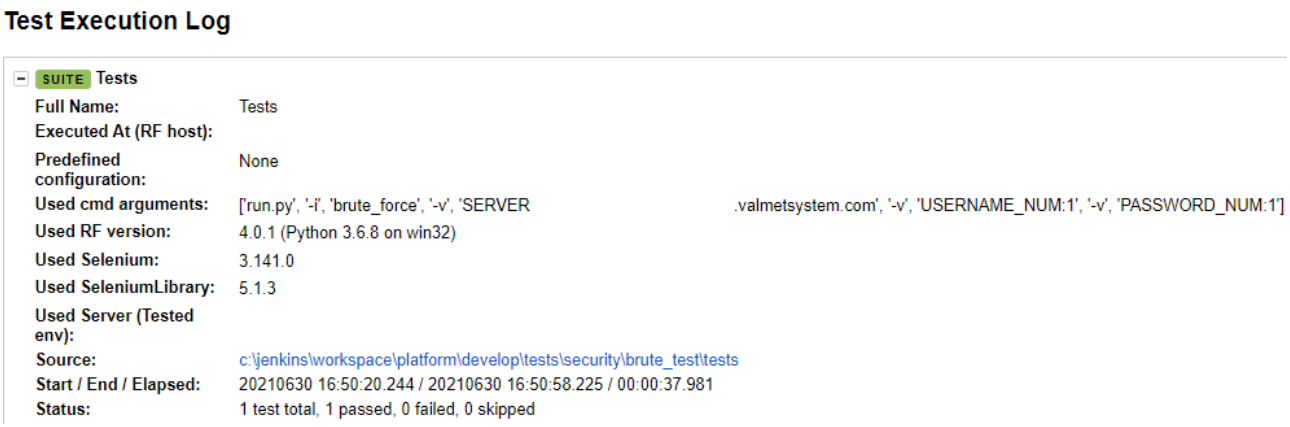
5.4 Testien raportointi

Testin ajamisen jälkeen tulokset ovat nähtävissä suoraan Jenkinsin käyttöliittymässä (kuvio 10) sekä tarkemmat raportoinnit ovat myös saatavilla. Robot Frameworkin generoimia raportteja ei yleensä tarvitse lukea, jos testi onnistuu.

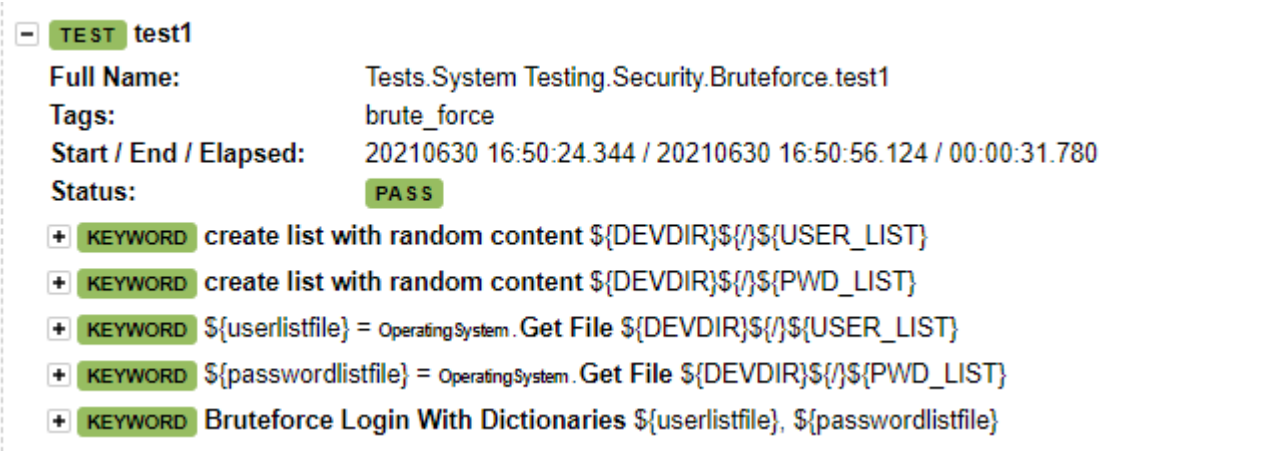


kuvio 10 Robot Frameworkin generoimat dokumentoinnit

Jos testi epäonnistuu, Robot Frameworkin generoimista dokumentoinneista löytyy helposti vian alkuperä. Testin logissa (log.html) näkyy testin ajokomento sekä argumentit (kuvio 11). Testin laajemmassa raportissa (report.html) on näkyvillä Robot- koodin ajon jokainen vaihe ja tietoa niiden suoriutumisesta (kuvio 12). Tämän raportin avulla selviää tarkalleen missä kohtaa testi epäonnistui.

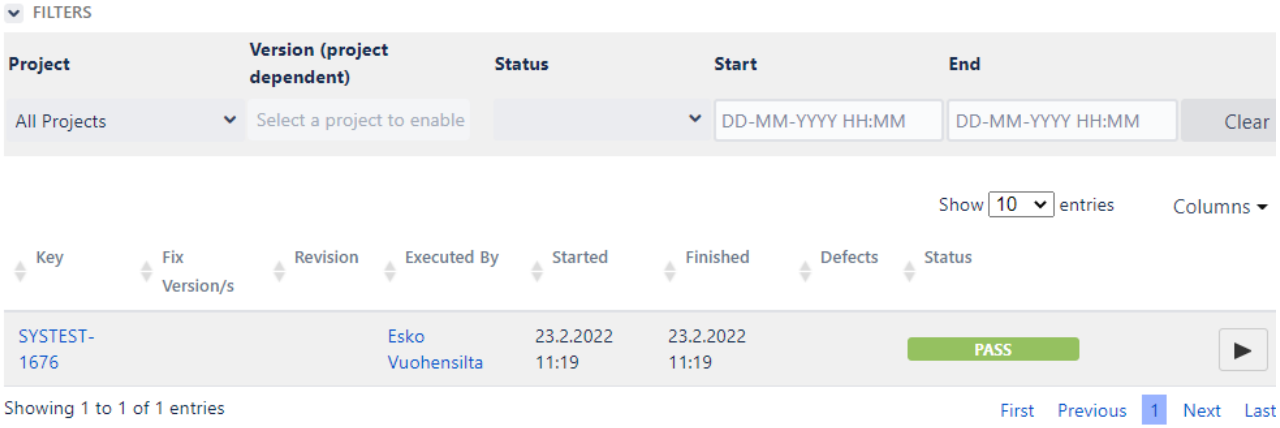


kuvio 11 Robot Frameworkin logi



kuvio 12 Robot Frameworkin raportointi

Testin ajon jälkeen luodaan Jiraan Xray test execution- issue testiä varten. Test executionin saa luotua suoraan Xray test design- issuesta (kuvio x). Testin tilaksi päivitetään PASS/FAIL testin ajon perusteella.



5.5 Testien jäljitettävyys ja automatisointi

Testin automatisoinnin toteuttaminen luodaan Jenkins- jobin konfiguraatioissa. Testin ajoa varten voidaan asettaa build trigger, jonka perusteella testi ajetaan. Build triggerillä määritellään, miten testiä ajetaan automaattisesti. Testi voi olla esimerkiksi osa isompaa testiryhmää, jolloin testi suoriutuu jonkun toisen Jenkins- jobin jälkeen. Testi voidaan myös ajoittaa ajettavaksi tietyin väliajoin, esimerkiksi joka viikonloppu.

6 Palaute ja jatkokehitys

Testin elinkaari on muiden testaajien antaman palautteen perusteella varsin pätevä. Aiemmin tehdyn haavoittuvuuden määrittelyn ja korjauksen dokumentoinnin pohjalta pääsi ketterästi luomaan haavoittuvuuden mitigointia testaavan kokonaisuuden. Xray- lisäosan tarjoamat eri issue- tyypit olivat hyvä tapa dokumentoida testin eri vaiheet. Xrayn avulla testin eri vaiheiden linkitys toisiinsa on myös selvä tapa toteuttaa testauksen jäljitettävyyttä.

Yksi kehitetty idea on luoda manuaalisille testeille Jenkins -job, ja niputtaa samantyylisiä testejä yhteen build triggerien avulla. Kun testiryhmän ensimmäinen testi ajetaan, se triggeraa seuraavan testiryhmän testin ja sama periaate toistuu testiryhmän viimeiseen testiin asti. Testeistä olisi myös mahdollista viedä automaattisesti palautetta Jiran Xray Test Execution – issueen. Jenkins job konfiguraatioissa pystyy analysoimaan Robot Framework – testin tuloksia ja niiden perusteella lähettämään pass / fail ilmoituksen Jiraan, joka muuttaisi testien tilan Jirassa automaattisesti.

7 Pohdinta

Tämän opinnäytetyön tarve määräytyy IEC 62443-4-1 standardin antamista vaatimuksista, jota Valmet Automation Oy:llä noudatetaan. Standardi määrittelee tarpeen luoda mitigointitestausta korjatuille haavoittuvuuksille, jotta voidaan todeta mitigoinnin olevan kattava. Tismalleen vastaavaa testausta Valmetilla ei ole tehty, ja testejä varten haluttiin luoda selvä elinkaari. Tämän elinkaaren haastavin osuus on hahmottaa aiemmin haavoittuvuusten mitigointien perusteella kattava testi. Joskus aiempi dokumentointi voi olla epäselvää tai puutteellista, joten yhteistyö eri tiimien ja eri kolleegoiden kanssa on erittäin tärkeää.

Robot Framework on yllättävän hyvä alusta testien luomiseen. Kun testataan Valmet DCS järjestelmän kaltaista käyttöjärjestelmää, haavoittuvuudet liittyvät usein graafiseen käyttöliittymään. Robot Frameworkillä käyttöliittymän testaaminen selaimen kautta osoittautuu tehokkaaksi tavaksi. Robot Frameworkin generoimat testilogit ovat kattavia, joita vähemmän teknisetkin ihmiset voivat helposti lukea.

Robot Frameworkin avainsana-pohjainen toimintaperiaate on erittäin tehokas periaate varsinkin Valmetin kaltaisessa isommassa työympäristössä. Koodikannassa on todella paljon eri avainsanoja Valmetin DCS- käyttöjärjestelmän käyttämistä varten, joten olemassa olevia avainsanoja kuten kirjautumislomakkeen täyttämistä ja lähettämistä voidaan käyttää uudestaan.

Robot Framework ei ole kuitenkaan täydellinen, esimerkiksi satunnaisen sisällön luomista varten. Onneksi Robot frameworkin kautta voidaan kuitenkin suorittaa muita ohjelmia, kuten vaikka Python- scriptejä täydentämään toimintaa.

-

Lähteet

About Chromedriver. Chromedriverin kotisivut. Viitattu 20.4.2022. https://chromedriver.chromium.org/

About Xray. Xrayn kotisivut. Viitattu 20.4.2022. https://docs.getxray.app/display/XRAY/About+Xray

Distributed control systems. Valmetin kotisivut. Viitattu 10.12.2021. https://www.valmet.com/automation/distributed-control-system/

IEC 62443-4-1 International standard. 2018. Viitattu 25.2.2022. https://webstore.iec.ch/preview/info_iec62443-4-1%7Bed1.0%7Den.pdf

Koskinen, S. Web-sovellusten tietoturvastandardin testaaminen. Opinnäytetyö, AMK. Tieto- ja viestintätekniikan opintoala. Viitattu 15.9.2021 https://www.theseus.fi/bitstream/handle/10024/208444/Koskinen_Sami.pdf?sequence=2&isAllowed=y

Meucci & Muller. 2014. OWASP Testing Guide 4.0. Viitattu 15.9.2021. https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf

Morpus, N. 2022. What is Jira. Viitattu 20.4.2022. https://www.fool.com/the-ascent/small-business/project-management/articles/what-is-jira

OWASP top 10. 2021. Artikkeli owasp.org verkkosivustolla. Viitattu 25.11.2021. https://owasp.org/www-project-top-ten/

Proffitt, B. 2011. Hudson devs vote for name change. Viitattu 10.12.2021. https://www.computerworld.com/article/2746627/hudson-devs-vote-for-name-change--oracle-declares-fork.html

Robot Framework. Robot Frameworkin kotisivu. Viitattu 15.9.2021. robotframework.org

Traficom. 2018. Turvallinen Tuotekehitys: Kohti Hyväksyntää. Verkkojulkaisu kyberturvallisuuskeskuksen sivustolla. Viitattu 15.12.2021. https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/publication/Turvallinen_tuotekehitys_Suomi_J003_2018.pdf

What is Git. Atlassianin kotisivut. Viitattu 20.4.2022 https://www.atlassian.com/git/tutorials/what-is-git

What is Jenkins? 2022. Viitattu 20.4.2022. https://www.edureka.co/blog/what-is-jenkins/

What is python. Pythonin kotisivut Viitattu 20.4.2022. https://www.python.org/doc/essays/blurb/

Liitteet

Liite 1. Bruteforce.robot

```
*** Settings ***
Documentation     Test cases for brute force testing
library   Process

Re-
source            ${PROJECTROOT}${/}resources${/}system_testing${/}common.resource

Suite Setup         Suite Setup
Suite Teardown      Suite Teardown
Force Tags          brute_force

*** Variables ***
# Define username and password list file names (in /tests/system_testing/security/)
${DEVDIR} =       ${PROJECTROOT}${/}tests${/}system_testing${/}security
${SCRIPTPATH} =   ${PROJECTROOT}${/}test_data${/}scripts
${USER_LIST}    username_list.txt
${PWD_LIST}     password_list.txt
${USERNAME_NUM} =   10
${PASSWORD_NUM} =   10

*** Test cases ***
Randomized Dictionary Bruteforce Login Test
    [Documentation]    Create random data for login at-
tempts, then bruteforce the specified login-page
    Create List With Random Con-
tent      ${DEVDIR}${/}tempdir${/}${USER_LIST}   ${USERNAME_NUM}
    Create List With Random Con-
tent      ${DEVDIR}${/}tempdir${/}${PWD_LIST}    ${PASSWORD_NUM}
    ${userlistfile}        Get File      ${DEVDIR}${/}tempdir${/}${USER_LIST}
    ${passwordlistfile}    Get File      ${DEVDIR}${/}tempdir${/}${PWD_LIST}

    Bruteforce Login With Dictionaries   ${userlistfile}   ${passwordlistfile}


*** Keywords ***

Create List With Random Content
    [Documentation]    Calls listgen.py script with handles for num-
ber of strings (-i) and the destination file (-o)
    [Arguments]    ${filepath}    ${amount_of_inputs}
    Run Process    python   ${SCRIPTPATH}${/}listgen.py   -
i   ${amount_of_inputs}   -o   ${filepath}


Bruteforce Login With Dictionaries
```

```
    [Documentation]    Iterate through usrenames and pass-
words, and try to log in with each combination.
    [Arguments]    ${userlistfile}    ${passwordlistfile}
    @{userlist}=    Split to lines   ${userlistfile}
    @{passwordlist}=    Split to lines   ${passwordlistfile}
    FOR   ${username}     IN   @{userlist}
        FOR   ${password}   IN   @{passwordlist}
            Input Username     ${username}
            Insert Password    ${password}
            Submit Credentials
            Page Should Be Loaded
            Page Should contain    Incorrect username or password
        END
    END


Suite Setup
    Open Browser To Login Page  HMI
    Wait Until Page Contains Element    login-button


Suite Teardown
    Close All Browsers
    Remove Directory   ${DEVDIR}${/}tempdir    recurcive = True
```

Liite 2. listgen.py

```
import argparse,os,string,random


#-------------------------------------------------------------------------------
# this script creates a text file with a given number of random strings
#
# the purpose of this script is to be used from bruteforce.robot directly
# with the number of strings (-i) and path (-o) for the file

def parse_args(args=None):
    """
    Parse command-line arguments. Returns arguments objects with keys that corre-
spond
    to argument names
    """
    parser = argparse.ArgumentParser(
        description="""run a list generator for robot framework tests.
        """)

    parser.add_argument('-i',
                        '--iterator',
                        help='Specify a desired number of random strings',
                        type=int)

    parser.add_argument('-o',
                        '--output',
                        help='create a .txt file for robot-frame-
work to use (for example username_list.txt)',
                        type=str)

    return parser.parse_args(args)



#-------------------------------------------------------------------------------
# create a folder and files inside the folder.
def create_list_file(output):
    filedirectory = os.path.dirname(output)
    if not os.path.isdir(filedirectory):
        os.mkdir(filedirectory)
    if os.path.isfile(output):
        os.remove(output)
    with open(output, 'w'): pass

#-------------------------------------------------------------------------------
# create n number of random strings, and write then to the speci-
fied text file. string lenght is defined by S
def create_list(iterator,output):
```

```
    textfile = open(output, "w")
    for i in range(0,iterator):
        S = 6
        randomstring = ''.join(random.choices(string.ascii_uppercase + string.dig-
its, k = S))
        textfile. write("{}\n".format(randomstring))
    textfile.close()

#---------------------------------------------------------------------------------
# running main
def main(args):
    #args.iterator = how many random strings we add to file
    #args.output   = output file name
    create_list_file(args.output)
    create_list(args.iterator,args.output)


#---------------------------------------------------------------------------------
if __name__ == "__main__":
    main(parse_args())
```