Sampo Bredenberg

Uhkatietojen rikastus avointen lähteiden tiedustelutiedoilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

23.01.2025

Tiivistelmä

Tekijä:	Sampo Bredenberg
Otsikko:	Uhkatietojen rikastus avointen lähteiden tiedustelutiedoilla
Sivumäärä:	44 sivua
Aika:	23.1.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaaja:	Lehtori Jorma Räty

Työssä tutkittiin, kuinka avointen lähteiden tiedustelutietoja voidaan hyödyntää uhkatietojen rikastamisessa. Tavoitteena oli kehittää API-rajapintapohjainen alusta, joka mahdollistaa digitaalisten tunnisteiden automaattisen rikastamisen useilla uhkatietolähteillä. Rikastamisen avulla pyrittiin parantamaan tietoturvaloukkausten tunnistamista ja torjuntaa tarjoamalla monipuolista, ajantasaista ja luotettavaa tietoa digitaalisille tunnisteille.

Työn toteutus sisälsi alustan suunnittelun ja kehittämisen sekä käyttöönoton. Työssä analysoitiin uhkatietojen käyttöön liittyviä vaatimuksia, valittiin keskeiset uhkatietolähteet ja toteutettiin alusta, joka yhdistää nämä lähteet yhteen alustaan. Alustan toiminnallisuudet suunniteltiin skaalautuviksi ja yhteensopiviksi olemassa olevien järjestelmien kanssa.

Työn tuloksena syntyi alusta, joka yksinkertaistaa uhkatietojen käyttöä vähentämällä manuaalisia vaiheita ja yhdistämällä useita uhkatietolähteitä yhteen järjestelmään. Alusta mahdollistaa suurten tietomäärien käsittelyn ja tarjoaa joustavuutta sen käyttöönotossa. Kuitenkin toiminnallisuuksiin, kuten API-rajapintadokumentointiin, arkojen tietojen salaamiseen ja pyyntöjen autentikointiin, jäi jatkokehityksen varaan.

Työn johtopäätöksenä todettiin, että uhkatietojen rikastaminen ja niiden automatisoitu hyödyntäminen tukevat merkittävästi tietoturvaloukkausten torjuntaa ja haitallisen toiminnan tunnistamista. Alustan kehittäminen tarjoaa työkalun tietoturvatoiminnan vahvistamiseen, ja sen jatkokehityksellä voidaan lisätä alustan käytettävyyttä ja tehokkuutta entisestään.

Avainsanat: Uhkatiedot, avoimet lähteet, rikastaminen, tietoturva, kyberturvallisuus, uhkatietolähteet, tietoturvajärjestelmä, digitaaliset tunnisteet, tiedustelutieto

---

Abstract

Author:	Sampo Bredenberg
Title:	Enriching Indicators of Compromise Using Open-Source Intelligence
Number of Pages:	44 pages
Date:	23 January 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisor:	Jorma Räty, Senior Lecturer

The study explored how open-source intelligence (OSINT) can be utilized to enrich threat intelligence data. The objective was to develop an API-based platform that enables the automated enrichment of digital identifiers with multiple threat intelligence sources. The enrichment process aimed to improve the detection and prevention of cybersecurity incidents by providing diverse, up-to-date, and reliable information for digital identifiers.

The implementation involved designing, developing, and deploying the platform. The study analyzed the requirements for using threat intelligence, selected key threat intelligence sources, and built a platform that integrates these sources into a single system. The functionalities of the platform were designed to be scalable and compatible with existing systems.

The result of the study was a platform that simplifies the use of threat intelligence by reducing the number of manual steps and consolidating multiple threat intelligence sources into a single system. The platform enables the processing of large volumes of data and offers flexibility in deployment. However, certain functionalities, such as API documentation, encryption of sensitive data, and request authentication, were left for future development.

The study concludes that enriching and automating the utilization of threat intelligence significantly supports the prevention of cybersecurity incidents and the identification of malicious activities. The development of the platform provides a tool to enhance cybersecurity operations, and further development could improve the usability and efficiency of the platform even further.

Keywords: Threat intelligence, open sources, enrichment, information security, cybersecurity, threat intelligence sources, security system, digital identifiers

---

Sisällys

Lyhenteet

ORM: *Object-relational mapping*. Ohjelmiston oliomallin mukaisen esityksen kuvaaminen relaatiotietokantamallin mukaiseksi esitykseksi ja kääntäen.

SIEM: *Security Information and Event Management*. Tietoturvavalvonnassa käytettävä järjestelmä, joka yhdistää tietoturvatiedon hallinnan ja tietoturvatapahtumien hallinnan yhteen paikkaan.

API: *Application Programming Interface*. Ohjelmointirajapinta, jonka avulla sovelluksen toimintoja voidaan käyttää.

IP: *Internet Protocol*. Tietoverkkojen viestintäprotokolla, jonka avulla laitteet välittävät tietoja toisilleen tietoverkoissa.

URL: *Uniform Resource Locator*. Yhdenmukainen verkkoresurssin osoite, jota käytetään paikantamaan resursseja tietoverkoissa.

REST: *Representational State Transfer*. Verkkopalveluarkkitehtuurityyli, joka määrittelee, miten verkkopalveluiden toiminnallisuudet tulisi rakentaa.

JSON: *JavaScript Object Notation*. Tiedonvaihtoformaatti, jossa tiedot määritetään avainsana, arvopareina.

1 Johdanto

Tietoturvan tärkeyttä korostetaan ja tietoisuutta jaetaan laajemmin kuin koskaan. Tietoturvauhkien ja onnistuneiden tietoturvamurtojen kasvaessa uhkien aikainen tunnistaminen korostuu. Vuonna 2023 Red Canary-yritys, joka erikoistuu uhkien havaitsemiseen, havaitsi 37 miljoonaa mahdollisesti haitallista tapahtumaa Red Canaryn asiakkaiden ympäristöissä, joista 58 000 varmistettiin oikeiksi uhkiksi [1]. Uhkatiedot toimivat tärkeässä osassa uhkien tunnistamisessa. Niiden tehokkuus auttaa tunnistamaan uhkia, on riippuvainen siitä, kuinka paljon luotettavaa informaatiota niistä kerätään.

Tässä työssä tutkitaan, kuinka avointen lähteiden tiedustelutietoja voidaan hyödyntää uhkatietojen rikastamiseen. Opinnäytetyön lopputuloksena on useita uhkatietolähteitä yhdistävä API-rajapintapohjainen alusta, joka mahdollistaa digitaalisten tunnisteiden automaattisen rikastamisen luotettavilla uhkatiedoilla. Sovelluksen avulla tietoturva-ammattilaiset voivat tehostaa työtään keskittymällä olennaisiin uhkiin ja havaita haitallista toimintaa nopeammin.

2 Lähtökohdat

Tietoturva on tärkeä osa tietojärjestelmiä. Tietoturvallisilla toimintatavoilla turvataan järjestelmien luotettava toiminnallisuus ja sisältö. Tässä luvussa pohjustetaan tietoturvapoikkeamien ja haitallisten uhkien tutkimiselle olennaiset käsitteet, sekä sivutaan niiden tutkintaa.

2.1 Uhkatiedot

Uhkatiedot ovat ennestään tunnistettuja, uhkatoimijoiden käyttämiä tapoja ja tunnistettavia tietoja, joita on käytetty tunnistetuissa tietoturvaloukkauksissa ja väärinkäyttöyrityksissä. Niitä kerätään tutkimalla tietoturvaloukkauksien jälkiä ja tunnistamalla hyökkääjän käyttämiä toimintatapoja, työkaluja, haittatiedostoja, sekä verkkoliikenneosoitteita. Uhkatietojen avulla pyritään paljastamaan

aktiivisia tietoturvaloukkauksia ja torjumaan hyökkäyksiä ennakoivasti, kun uhkatekijöille ominaisia toimintatapoja valvotaan ennestään tunnistetuilla tiedoilla.

Tässä opinnäytetyössä rajataan uhkatiedot digitaalisiin tunnistetietoihin, joita ovat:

- IP-osoitteet (IPv4 ja IPv6)
- verkkotunnisteet
- URL:it
- tiedostojen hajautusarvot.

Digitaalisia tunnistetietoja löytyy jokaisesta tietojärjestelmästä verkkoliikenteestä, järjestelmän tapahtumia keräävistä lokitiedostoista sekä muista tiedostoista, kuten ohjelmatiedostoista.

Haittaohjelmia käytetään saavuttamaan jokin tavoite. Ennen lopullista tavoitetta haittaohjelma pyrkii usein tunkeutumaan syvemmälle ympäristöön, saamaan salasanoja haltuun ja korottamaan käyttöoikeuksia, kunnes tavoite on mahdollista toteuttaa. Tavoitteen edellytyksenä voi olla jokin tietty laite, verkkoympäristö tai tietyn tasoiset käyttöoikeudet. Tiedostoja voidaan tunnistaa hajautusarvojen perusteella.

Verkkotunnisteet isännöivät haitallisia tiedostoja tai muita resursseja kuten kalastelusisältöä, joiden kautta saadaan ensimmäinen pääsy ympäristöön. Ne toimivat myös kommunikaatio-osoitteina, joiden kautta voidaan komentaa haittaohjelmien toimintaa, sekä minne viedä kaapattuja tiedostoja.

2.1.1 Digitaaliset tunnistetiedot

IP-osoitteet toimivat tietoliikenteen perustana. Kun tietoa halutaan lähettää, vastaanottaa tai hakea toiselta laitteelta, laitteen on osoitettava, mistä liikenne kulkee ja mihin se päätyy. IP-osoitteet toimivat lähtö- ja kohdeosoitteina, joiden avulla tietojärjestelmät reitittävät, mikä laite lähettää tietoja ja minne. IP-osoitteet voidaan jakaa neljään tyyppiin:

- julkiset IP-osoitteet
- yksityiset IP-osoitteet
- dynaamiset IP-osoitteet
- staattiset IP-osoitteet. [2.]

Uhkatiedoille oleelliset IP-osoitetyypit ovat julkiset IP-osoitteet. Julkiset IP-osoitteet ovat yleensä pysyviä, aina ainutlaatuisia ja ne toimivat tunnisteina julkisessa internetissä. Uhkatietojen kontekstissa IPv4 ja IPv6 eivät juurikaan eroa. IP-osoitteet ovat yleensä aina osana tietoturvaloukkauksia, sillä kaikki verkkoliikenne tarvitsee aina IP-osoitteen.

Verkkotunniste on ihmisymmärrettävä merkkijono, joka tarvitsee aina IP-osoitteen, jonka takana jotain palvelua pidetään yllä. Verkkotunnisteet toimivat hierarkialla, jossa tasot erotetaan pisteellä. Ylätason verkkotunnusta hallitsee aina jokin organisaatio, joka jakaa toisen tason verkkotunnisteita organisaatioille sekä yksityishenkilöille. Uhkatiedot koskevat toisen tai jopa kolmannen tason verkkotunnisteita, joissa uhkatoimija ylläpitää haitallista toimintaa. Usein toisen tason verkkotunniste osallistuu kokonaisuudessaan haitalliseen toimintaan.

Verkko-osoitteilla tarkoitetaan tarkkaa osoitetta verkkotunnisteella, joka johtaa tiettyyn resurssiin. Verkko-osoitteessa voi olla mukana myös kyselyparametrejä.

Kaikkiin verkkoliikenteeseen liittyvien tunnistetietojen kautta voidaan suorittaa samoja haitallisia toimia, koska IP-osoite, verkkotunniste ja URL-osoite ovat kaikki jokseenkin riippuvaisia toisistaan. Näiden tunnistetietojen avulla voidaan isännöidä haitallisia tiedostoja, joita yritetään huijata laitteen käyttäjää lataamaan, tai mahdollistetaan aktiivisen haittaohjelman lataaminen jatkotoimenpiteiden suorittamiseksi. Verkossa isännöidään myös erilaisia huijauksia, joiden tavoitteena on yksinkertaisimmillaan seurata käyttäjän toimintaa, tai varastaa henkilökohtaisia tietoja.

IP-osoitteet ja verkkotunnisteet ovat hallittuja tunnisteita. Eri organisaatiot hallinnoivat verkkoliikenteeseen liittyviä tunnisteita ja jakavat osia alemmille organisaatioille käyttöön. Tämän hallinnollisen rakenteen takia IP-osoitteet sekä

verkkotunnisteet eivät ole pysyviä uhkatietoja. IP-osoitteiden ja verkkotunnisteiden käyttö voidaan mitätöidä, jos niistä tunnistetaan suoritettavan haitallista toimintaa. Näitä samoja tunnisteita voidaan jakaa myöhemmässä vaiheessa toiselle toimialle, joka käyttää niitä laillisiin toimiin.

Tiedostojen hajautusarvot ovat matemaattisia laskentatuloksia tiedoston sisällöstä. Tiedostojen nimet ovat mielivaltaisia ja niitä pystytään muuttamaan ilman, että tiedoston sisältöön tarvitsee tehdä muutoksia. Tiedostonimi ei tällöin toimi tunnisteena tiedostoille. Hajautusarvot ovat uniikkeja merkkijonoja, jotka tehdään erilaisten laskenta algoritmien avulla ja kuvaavat tiedoston sisältöä. [3.] Vertaamalla tiedoston hajautusarvoa voidaan varmistaa, onko tiedoston sisältö muuttunut, jolloin voidaan varmistaa, onko ennestään tunnettua tiedostoa muutettu esimerkiksi hyökkääjän toimesta. Tiedoston toiminta voidaan analysoida, sen haitallisuus arvioida ja tulokset julkaista tai tallentaa. Hajautusarvot pysyvät samana, jos tiedoston sisältö pysyy täysin samana, joten tallennettuja tuloksia voidaan käyttää arvioimaan tiedostoa, jos se huomataan uudestaan. Useat virustentorjuntaohjelmat käyttävät osana toimintaansa hajautusarvoja tunnistaakseen ennestään löydettyjä haittaohjelmia.

2.1.2 Uhkatietojen kerääminen

Uhkatiedot ovat reaktiivisia luonteeltaan. Niitä kerätään joko osana teknistä tutkimusta tapahtuneen tietoturvaloukkauksen jälkeen, tai todettujen väärinkäyttöyritysten perusteella tunnistamalla digitaaliset tunnistetiedot, jotka olivat osallisina yritykseen [4]. Nämä digitaaliset tunnistetiedot tallentuvat valvottavien ympäristöjen digitaalisiin lokeihin.

Lokit ovat sovellusten ja käyttöjärjestelmien tuottamia tietoja kyseisen tuotteen toiminnasta, jotka tallennetaan tiedostoon tai näytetään tulosteena laitteella. Niiden muoto ja sisältö vaihtelevat tuotteista toisiin. Lokien tarkoituksena on kuvata tuotteen toimintaa. Ne voivat sisältää merkintöjä esimerkiksi sovelluslogiikasta, muutoksista, virheistä sekä tapahtumiin johtaneista toiminnoista. Palomuurin tehtävänä on määritellä verkkoliikennettä ja suojella sisäverkkoa

haitallisilta toimijoilta. Palomuurit tallentavat tiedot verkkoliikenteestä, kuten lähdeosoitteen ja kohdeosoitteen, käytetyn portin sekä lähetettyjen ja vastaanotettujen tavujen määrän. [5.] Jos palomuuri estää yhteyden, se voi tallentaa ilmoituksen estosta ja sen syystä.

Jäsennellyillä lokeilta, joilla on jokin standardi tiedostomuoto, tunnistetiedot kuten IP-osoitteet ovat aina tietyssä kohtaa, jolloin tunnisteet voidaan erottaa helposti lokeilta ohjelmallisesti. Ohjelmointi- ja skriptauskielet tarjoavat usein valmiita ratkaisuita yleisimpien tiedostomuotojen, kuten JSON-, XML- ja CSV-tiedostojen jäsentelyyn.

Lokeilta, joilla ei ole struktuuria, voidaan digitaaliset tunnistetiedot erottaa käyttämällä säännöllisiä lausekkeita. Säännöllisillä lausekkeilla voidaan poimia tekstistä tietyn mallisia tekstijonoja. Kaikki digitaaliset tunnistetiedot noudattavat tietynlaista mallia, joka mahdollistaa niiden poimimisen.

Tekninen tutkimus seuraa tietoturvaloukkausta. Sen tarkoituksena on käydä systemaattisesti läpi digitaalista todistusaineistoa, kuten tietoliikennelokeja, järjestelmälokeja ja haittaohjelmia. Sen tavoitteena on usein jäljittää, mitä tietoturvaloukkauksessa on tapahtunut, mitä hyökkääjä on saanut aikaan ja miten laajalle toiminta on ulottunut. Tutkimuksen tavoitteena voi myös olla uhkatietojen kerääminen. Tuloksena on listaus digitaalisista tunnistetiedoista, mitä tietoturvaloukkauksessa on käytetty. Nämä toimivat uhkatietoina, joita voidaan käyttää muissa ympäristöissä uhkien löytämiseen ja torjumiseen.

Haittaohjelmille voidaan suorittaa samanlaista teknistä tutkimusta, jotta selvitetään niiden toimintatavat ja poimia uhkatiedot, joita ne saattavat käyttää toiminnassaan. Haittaohjelmien tutkiminen voi kuulua osaksi tietoturvaloukkaukseen liittyvää teknistä tutkimusta tai täysin itsenäinen toimenpide. Tietoturvatiimit voivat tutkia haittaohjelmia kartoittaakseen tietyn haitallisen toimijan toimintatapoja tai palveluita. Näitä tietoja voidaan käyttää ennaltaehkäisevästi.

Aktiivisen valvonnan ja tehokkaiden turvallisuusmenetelmien kautta voidaan tunnistaa väärinkäyttöyrityksiä. Väsytyshyökkäyksessä hyökkääjä pyrkii

pakottamaan kirjautumisen järjestelmään kokeilemalla jokaista mahdollista salasanaa. Tunnistamalla normaalista toiminnasta poikkeavaa verkkoliikennettä voidaan tietoturvaloukkaukset estää ennenaikaisesti. Väsytysvakaahyökkäykset ovat helposti tunnistettavia tapauksia. Muita selkeitä poikkeavia toimintoja ovat erilaisten haavoittuvuuksien hyväksikäyttöyritykset. Tapahtumien pohjalta voidaan myös kerätä digitaaliset tunnistetiedot ja koostaa ne uhkatiedoiksi.

Erityinen tapa kerätä uhkatietoja on jättää ympäristöön näkyvälle laite, joka houkuttelee hyökkääjiä yrittämään tunkeutua siihen. Näitä laitteita kutsutaan hunajapurkeiksi. Niiden tarkoituksena on esittäytyä helppoina kohteina, joiden annetaan tahallaan joutua väärinkäytön uhriksi. Nämä laitteet toimivat ansoina, ja ovat eristettynä muusta verkosta. Hyökkääjän toimintoja voidaan seurata ja saadaan tietoja aitojen uhkien käyttämistä tekniikoista ja tunnisteista. [6.]

2.1.3 Uhkatietojen käyttö

Ympäristöissä on normaalia toimintaa, johon liittyy paljon verkkoliikennettä ja tiedostojen toimintaa. Vertaamalla tunnettuja uhkatietoja ympäristön toiminnassa esiintyviin tunnisteisiin voidaan aktiiviset tietoturvaloukkaukset ja hyökkäysyritykset tunnistaa ajoissa. Kehittyneissä tietoturva-järjestelmissä voidaan esimerkiksi verkkoliikenteen kohdeosoitteita ja prosessien toimintaa verrata suoraan tunnettuihin uhkatietoihin. Virustorjuntaohjelmat perustuvat usein siihen, että ne vertaavat laitteen toimintaa tunnettuihin uhkatietoihin. Uhkatietoja käytetään myös manuaalisessa analysoinnissa.

Poikkeavan toiminnan ilmetessä tietoturvatiimi suorittaa usein manuaalista analysointia, jossa lokien pohjalta pyritään tunnistamaan haitallinen toiminta ja löytöjen perusteella tekemään vastatoimenpiteitä. Analysoinnissa käsitellään usein lokeja, jotka sisältävät useita erilaisia digitaalisia tunnisteita. Tunnisteiden rikastaminen uhkatiedoilla nopeuttaa mahdollisen haitallisen toiminnan löytämistä ja tehostaa toimintaa, kun tutkintaa voidaan keskittää pois normaalista toiminnasta. Uhkatietoja voidaan siis käyttää tutkinnan tukena, jolloin annetaan syvempää kontekstia haitalliselle toiminnalle. Niiden avulla voidaan tehdä

lähteisiin perustuvia päätöksiä tilanteissa, joissa pitää selvittää, onko jokin toiminta haitallista.

Kehittyneemmissä ympäristöissä lokeja viedään SIEM-alustoille, jotka toimivat keskitettyinä lokien hallinta-alustoina. SIEM-alusta mahdollistaa tietoturvatapahtumien keskitetyn monitoroinnin, analysoinnin ja hallinnan. Alustalle viedään tietoturvatapahtumien valvontaan tarvittavia lokeja, kuten palomuuri-, käyttäjienhallinta alusta- ja laitetapahtumalokia. Useat SIEM-alustat jäsentelevät lokit automaattisesti sellaiseen muotoon, joka mahdollistaa tehokkaan uhkatietojen keräämisen sekä niiden analysoinnin.

2.2 Julkisiin lähteisiin perustuva tiedustelutieto

Uhkatietojen kerääminen on reaktiivista. IP-osoitteet ja verkkotunnisteet ovat lyhytkestoisia, sillä niiden käyttö voidaan helposti mitätöidä havaittaessa niistä suoritettavan haitallista toimintaa. Yksittäisen tahon on hankala pitää ajantasaista uhkatietotietokantaa yksin pystyssä.

Kyberturvallisuusyhteisö on hyvin avoin jakamaan tietoja aktiivisista uhista. Uhkatietoja julkaisevat niin yksityiset tutkijat ja ammattilaiset kuin organisaatiot ja yrityksetkin, joko vapaasti tai kaupallisesti. Monet organisaatiot, yhteisöt ja yritykset erikoistuvat uhkatietojen keräykseen, analysointiin ja niiden julkaisuun. Uhkatietojen jakamisessa on usein tavoitteena lieventää tietoturvaloukkausten vaikutuksia, havaita aktiiviset hyökkäykset nopeammin ja ennaltaehkäistä niitä. Julkisten lähteiden uhkatietojen käyttäminen mahdollistaa ajantasaisen uhkien seurannan. Vertaamalla monia lähteitä keskenään saadaan tarkka kuva uhkatietojen todenmukaisuuksista ja nopeutetaan analysointia, kun haitallinen toiminta on jo löydetty.

Monet alustat tarjoavat sovellusrajapinnan alustojen ominaisuuksien, kuten uhkatietojen hakemisen käyttöön.

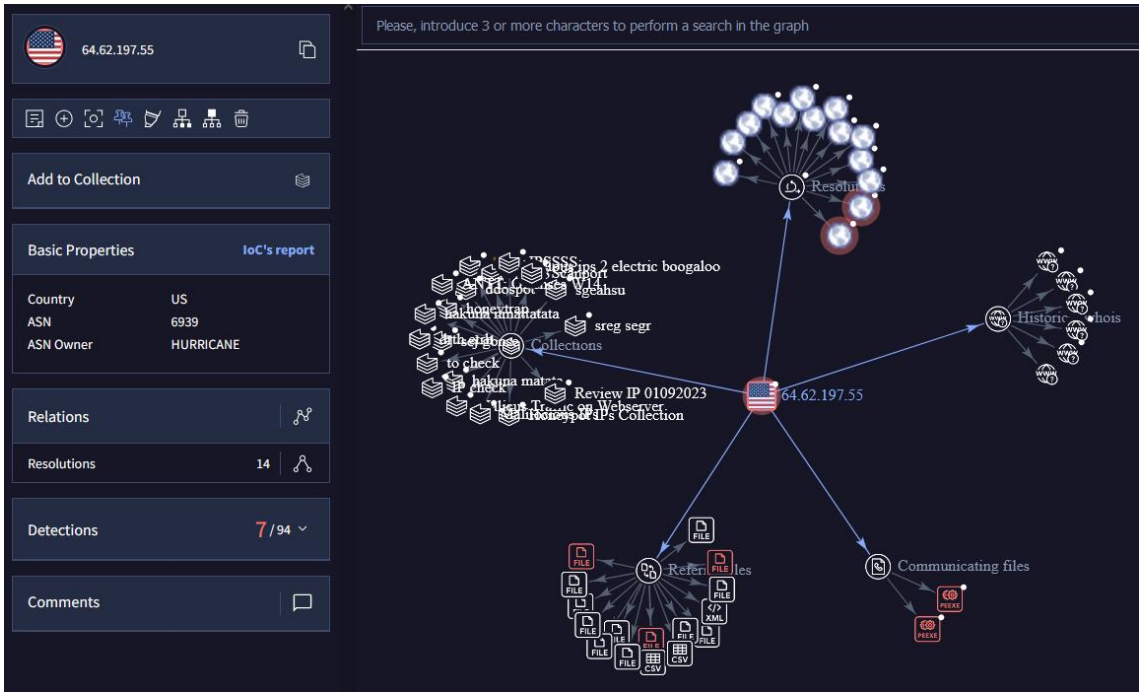
2.2.1 Avoimet lähteet

Avoimet toimijat julkaisevat sekä mahdollistavat uhkatietojen hakemisen alustoiltaan vapaasti. Moni avoin toimija perustuu joukkoistettuun tiedustelutiedon keräämiseen, eli yhteisöön kuuluvien henkilöiden tai organisaatioiden nostamiin havaintoihin.

VirusTotal on yksi suosituin ja tunnetuin julkisiin lähteisiin perustuvien tiedustelutietojen alustoista. Se tarjoaa paljon erilaisia ominaisuuksia, ja sen toiminnallisuus on integroitu moneen tietoturvatuotteeseen. VirusTotal kokoaa eri tietoturvatoimijoiden virustorjuntaohjelmamoottoreita, verkkosivuskannereita ja muita havaintoja tuottavia tuotteita alustalleen. Alusta mahdollistaa kenen tahansa syöttää tiedosto, verkkotunnus, URL- tai IP-osoite, joka arvioidaan näillä tuotteilla ja tulokset jaetaan vapaaseen käyttöön. Havainnot eri toimijoiden tuotteista antavat kattavan näkymän tunnisteiden toiminnasta ja onko niiden toiminnassa havaittu haitallisuutta. Alustalta löytyy esimerkiksi tiedostojen skannaamiseen tunnetuimmat virustentorjuntaohjelmatoimittajat, kuten McAfee, WithSecure, Microsoft ja CrowdStrike.

VirusTotal sisältää myös yhteisöominaisuuksia. Rekisteröityneet käyttäjät pystyvät äänestämään tunnistetietoja positiivisesti tai negatiivisesti sekä jättämään vapaan kommentin tunnisteille. Yhteisön määrittämä yhteispistemäärä sekä kommentit näytetään tunnisteen yhteydessä. Käyttäjät voivat luoda omia uhkatieto kokoelmia ja luoda miellekartan, jossa voi yhdistellä erilaisia tunnisteita ja niiden ominaisuuksia (kuva 1). Näitä kaavioita voi jakaa muulle yhteisölle. Alusta mahdollistaa kaikkien tietojen hakemisen API-rajapinnan avulla. Ilmainen API-rajapinta on rajoitettu 500 kyselyyn päivässä, 4 kyselyä minuutissa. Hinnoitellussa API-rajapinnassa ei ole kyselyrajoituksia, eikä sen hinnasta ole julkista

tietoa.



Kuva 1. VirusTotal-miellekartta.

AbuseIPDB tarjoaa alustan, jossa rekisteröitynyt ja varmistettu käyttäjä pystyy ilmoittamaan IP-osoitteiden suorittaneen haitallista toimintaa. IPv4- tai IPv6hteenyosoitteen lisäksi täytyy väärinkäyttö kategorisoida ainakin yhteen valmiista kategorioista ja vapaamuotoinen kommentti liittää ilmoitukseen. Alustalta voi etsia IP-osoitetta, jonka sivulla mahdolliset ilmoitukset ja niiden tiedot näkyvät. Ilmoitusten perusteella alusta laskee "Confidence of Abuse" -arvon IP:lle, joka kuvaa, kuinka varmoja alustan ylläpitäjä organisaatio on siitä, että IP-osoite suorittaa väärinkäyttöä asteikolla 0–100. Alusta tarjoaa API-rajapinnan, jonka kautta voi tehdä ilmoituksia sekä hakea IP-osoitteiden tietoja ilmoituksista. Ilmainen API-rajapinta mahdollistaa 1 000 kyselyä päivässä. Varmistetut käyttäjät rajoitetaan 5 000 kyselyyn päivässä. Maksullisilla sopimuksilla kyselyiden rajoituksia voi kasvattaa 50 000 kyselyyn asti. [7.]

Open Threat Exchange on yksi suurimmista avoimista uhkatieto yhteisöistä. Alusta mahdollistaa uhkatietojen jakamisen ”Pulse” -raporttien muodossa. ”Pulset” ovat yhteisön jäsenten postauksia, joissa on lähde ja listaus tähän

lähteeseen liittyvistä uhkatiedoista. Yhteisön jäsenet ovat yksittäisiä tietoturva-ammattilaisia tai organisaatiota, jotka keräävät ja julkaisevat uhkatietoja esimerkiksi tutkintojen tai hunajapurkkien tulosten perusteella. Alustalla voi seurata tiettyjä jäseniä ja saada heidän julkaisemiaan pulseja suoraan. Tämän avulla voi luoda syötteen itselleen oleellisista toimijoista tai toimialoista. Pulsejen lisäksi yksittäisiä tunnisteita voi etsiä ja alusta antaa näkymän, jossa siihen liittyviä pulseja ja muita liittyviä tunnisteita voi seurata. Open Threat Exchange tarjoaa rajattoman API-rajapinnan, jonka avulla alustan toimintoja voi käyttää.

Tranco on listaus, joka mittaa verkkotunnisteiden suosiota. Se yhdistää viiden eri toimijan kokoamat verkkotunnisteen suosiosijoitukset, jotka luovat listauksen miljoonasta suosituimmasta verkkotunnisteesta viimeiseltä 30 päivältä. Yhdistämällä eri toimijoiden sijoituksia tavoitteena on luoda luotettava listaus suosituimmista verkkotunnuksista, jota on vaikea manipuloida keinotekoisesti. Tranco-listaus antaa kontekstia verkkotunnusten toiminnasta. Aidot verkkotunnisteet ovat yleensä vakiintuneita palveluita, joita käytetään jatkuvasti. Uudet verkkotunnisteet tai sellaiset, joille ei ole paljoa liikennettä, voi muiden todisteiden avulla kertoa mahdollisesti haitallisesta toiminnasta. Tranco tarjoaa rajattoman API-rajapinnan, jonka avulla voi saada tiedot verkkotunnisteen sijoituksista viimeisen 30 päivän ajalta.

Stop Forum Spam -alusta kokoaa raportoituja roskapostitapauksia, jotka ovat kohdistuneet foorumeihin sekä blogeihin. Alustan tavoitteena on joukkoistaa roskapostiraportointi ja koota ne yhteen paikkaan, jotta roskapostien lähteitä voidaan estää ennaltaehkäisevästi. Stop Forum Spam listaa yli 1 000 erilaista lähdettä raporteilleen. Nämä lähteet ovat erilaisia foorumeita sekä muita verkkosivuja. Alustan käyttö on vapaata. Kyselyiden tekeminen sekä tilastojen selaaminen alustan verkkosivuilta ei vaadi rekisteröitymistä. API-rajapinnan käyttöön sekä osoitteiden raportoiminen vaatii rekisteröitymistä. API-rajapinta palauttaa tiedon siitä, onko kyseisestä IP-osoitteesta raportteja, kuinka monta sekä milloin raportti on viimeksi tehty.

Abuse.ch ylläpitää kolmea alustaa, jotka erikoistuvat erilaisiin osa-alueisiin uhkatietojen rikastamisessa. Jokainen alusta on vapaasti käytettävissä ja sisältävät käyttäjien syöttämiä uhkatietoja. API-rajapinnan käyttö vaatii ilmaisen rekisteröitymisen. URLHaus keskittyy URL-osoitteisiin, joita käytetään aktiivisesti haittaohjelmien jakamiseen. Palvelu sisältää käyttäjien syöttämiä URL-osoitteita, jotka hostaavat haittaohjelmia, sekä mikä hostattava tiedosto on. MalwareBazaar keskittyy vahvistettuihin, tuoreisiin haittaohjelmahavaintoihin. Käyttäjät toimittavat alustalle haittaohjelmia ja kontekstia siitä, mistä se on tullut. Jos alkuperä on URL-osoite, linkittyvät MalwareBazaarin ja URLHausen merkinnät. ThreatFox keskittyy tunnisteisiin, jotka ovat indikaattoreita haittaohjelmien toiminnasta. Nämä ovat verkko-osoitteita tai tiedostoja, jotka voivat olla merkki tunnistamattoman haittaohjelman toiminnasta ympäristössä. Yhdessä nämä kolme alustaa mahdollistavat tunnisteiden seuraamisen niiden alkuperästä siihen, miten niiden toiminta voidaan tunnistaa. Rekisteröitymällä jokaista alustaa voi käyttää API-rajapinnan avulla ja automatisoida tunnisteiden rikastamista.

2.2.2 Kaupalliset toimijat

Kaupalliset toimijat keräävät uhkatietoja kaupalliseen käyttöön. Kaupalliset toimijat suorittavat uhkatietojen keräämisen monilla tavoilla. Monet kaupalliset toimijat suorittavat omia tutkimuksia sekä tiedonkeräystoimenpiteitä, jotka kohdistuvat eri kohteisiin. Tarkkoja menetelmiä ei usein kerrota avoimesti, joten kuvaukset rajoittuvat yleensä siihen, millaista tietoa alustat sisältävät ja mihin niitä voidaan käyttää. Useat kaupalliset toimijat tarjoavat ilmaisversion tuotteestaan, jossa on yleensä rajoitettu määrä kyselyitä sekä vähemmän tietoa saatavilla. Maksullisten versioiden hinnoittelu perustuu usein käyttäjien tai käytön mukaan, eikä hinnoittelu ole julkista tietoa.

GreyNoise kerää dataa internettiä skannaavista IP-osoitteista ja analysoi niitä mahdollisten haitallisten toimijoiden löytämiseksi. Internettiä skannaa moni eri taho, kuten hakukoneet, tutkimuskeskukset sekä yksittäiset henkilöt. Myös haitalliset toimijat skannaavat internetiä löytääkseen hyväksikäytön mahdollisuuksia tai kartoittaakseen tavoitteena olevan kohteen ympäristöä. GreyNoise pyrkii

erottelemaan kaikesta verkkoliikenteestä ne, joiden toiminta on mahdollisesti haitallista. [8.] Alustaa käyttämällä voidaan havaittuun verkkoliikenteeseen saada kontekstia sen alkuperästä ja tarkoituksista. Jos IP-osoitteen on havaittu skannaavan koko internetiä, kyseinen toiminta ei ole kohdistettua mihinkään tiettyyn ympäristöön. Jos osoitetta ei ole havaittu, voi se olla merkki mahdollisesta hyökkäyksestä, tai toimijasta, joka pyrkii löytämään tavan hyökätä. GreyNoise tarjoaa ilmaisen yhteisö API-rajapinnan, jolla voi tehdä 50 kyselyä viikossa.

Maltiverse on alusta, joka yhdistää eri toimijoiden uhkatietoja yhteen paikkaan. Alusta tallentaa ulkoisten toimijoiden julkaisemia uhkatietoja ja luo niistä aikajanan, josta näkee uhkatiedon kategorisoinnin historian. Alusta tarjoaa 20 API-rajapintakyselyä päivässä.

2.2.3 Joukkoistettu tiedustelutieto

Moni alusta mahdollistaa julkaisun tai jopa perustuu yksittäisten käyttäjien tekemiin havaintoihin. Itsenäiset tietoturva-ammattilaiset voivat kerätä ja suorittaa omia tutkimuksia ja julkaista löydökset vapaasti muiden käyttöön. Yksittäisen henkilön johtopäätökset jonkin tunnisteen haitallisesta toiminnasta ei usein ole luotettava lähde, mutta jos kymmenet käyttäjät tuovat esiin samanlaisia nostoja, voi näitä verrata omiin huomioihin luotettavammin.

Joukkoistetut alustat voivat yksinkertaisimmillaan antaa käyttäjien tehdä ilmoituksia haitallisesta toiminnasta, kuten Abuse.ch -alustoilla, tai vain äänestää tunnisteen luotettavuudesta. Syvempää kontekstia löytyy alustoilta, kuten Open Threat Exchange. Käyttäjät julkaisevat nostoja haitallisesta toiminnasta, joissa mukana tunnisteita ja lähteitä tai raportteja.

2.3 Uhkatietojen rikastaminen

Uhkatietojen rikastamisessa tavoitteena on tuoda lisäkontekstia digitaalisen tunnisteen toiminnasta tutkintaan.

Järjestelmien tunnistettua epänormaali toiminta, tietoturvatiimi tutkii, mitä tämän toiminnan takana on ja pyrkii selvittämään, onko se asiallista vai onko sen takana haitallinen toimija. Toiminnassa on melkein aina jokin digitaalinen tunniste mukana, joka voi johdattaa tutkintaa oikeaan suuntaan. Edistyneissä järjestelmissä voi olla integraatioita alustoille, joista haetaan automaattisesti lisätietoja tunnisteista, jotka liittyvät tunnistettuun toimintaan.

Tutkinnan edetessä tunnisteiden määrä usein lisääntyy, mitä enemmän toimintaa selvitetään. Näiden tunnisteiden rikastaminen uhkatiedoilla tehostaa tutkintaa, kun normaali toiminta voidaan erotella ja löytää aikaisempien tutkintojen perusteella todettu haitallinen toiminta. Jotta tutkija saisi mahdollisimman laajan kontekstin ja moninaisen näkemyksen tunnisteista, tarvittaisiin rikastukseen useita lähteitä. Alustat, kuten VirusTotal, yhdistävät jo valmiiksi monien toimijoiden analyysejä. Yksinkertaisessa käytössä tämä usein riittää lisäkontekstin rakentamisessa. Toimijoiden analysointi moottorit antavat ilmoille vain tiedon siitä, pitävätkö ne tunnistetta haitallisena, eivätkä miksi se on haitallinen. Moninaisen näkemyksen rakentaminen edellyttää useamman lähteen käyttöä, joiden erilainen toiminnallisuus ja näkökulmat antavat paremman kokonaiskuvan, kun arvioidaan mahdollisesti haitallista toimintaa.

Opinnäytetyön tuloksena on alusta, joka yhdistää eri uhkatietolähteet yhteen paikkaan, ja mahdollistaa digitaalisten tunnisteiden moninaisen rikastamisen yhdestä paikasta. Tämä poistaa tarpeen käyttää eri palveluiden verkkosivuja, kun niitä voidaan käyttää yhdestä paikkaa. Alusta käyttää esiteltyjen uhkatietolähteiden API-rajapintojen ominaisuuksia, ja tuottaa yhdenmukaista dataa niiden antamista tiedoista.

3 Teknologiat

Opinnäytetyön tuloksena rakennettuun alustaan käytettiin monia yleisiä ohjelmiston kehitykseen tarvittavia teknologioita. Teknologiat valittiin halutun käyttötarkoituksen puitteissa ja niitä lisättiin sitä mukaan, kun tarve toiminnallisuuteen tuli. Perusteena oli helposti käyttöönotettava, REST API -rajapinta pohjainen

sovellus. Tämän perusteella työ toteutettiin Pythonin Flask-ohjelmistokehyksellä ja rakennetaan Docker-sovellukseksi. Tässä luvussa käydään läpi alustan rakentamiseen käytetyt teknologiat.

3.1 Python

Työn ohjelmointikielenä käytetään Python versiota 3.11. Python tarjoaa laajan valikoiman kehittyneitä kirjastoja, joiden käyttäminen nopeuttaa kehittämistä. Työ perustuu erilaisten API-rajapinta-alustojen käyttöön ja niiden tuomista yhteen. Uusien API-rajapinta-alustojen lisääminen haluttiin tehdä helpoksi ja modulaariseksi. Pythonin yksinkertaisuus mahdollistaa ylläpitämisen ja uusien yhteyksien luonnin minimaalisella perehtymisellä alustan syvempään toimintaan.

Kirjastojen käyttö on yksinkertaista pythonilla. Toiminnallisuuden rakentaminen ja toisten teknologioiden tuominen sovellukseen on helppoa, kun integroimiseen ja perusteiden rakentamiseen ei tarvitse käyttää aikaa.

Kehityksen aikana käytettiin Pythonin virtuaalista ympäristöä. Virtuaalinen ympäristö sisältää vain määritetyt kirjastot ja tietyn Python version, jonka avulla kehitys on yhtenäistä eri alustoilla ja käyttöönottoprosessi helppoa, kun kehitys tehdään aina identtisessä kehitysympäristössä.

3.1.1 Flask

Flask on verkkosovelluskehys, joka mahdollistaa yksinkertaisten http-pohjaisten sovellusten rakentamisen [9]. Flaskia käytetään työssä määrittämään API-rajapinnat, joita kutsumalla sovelluksen toiminnallisuuksia käynnistetään ja määritetään.

Flaskin ominaisuuksista työssä käytetään kaavioita nimeltä blueprint, yhtenäisten toiminnallisuuksien kuten virheilmoitusten määrittämiseen, ”route” dekoraattoreita rekisteröimään URL-osoitteet, jotka käynnistävät sovelluksen eri

toimintoja, sekä valmiita tietokantaliittimiä ja apufunktioita. Tietokantaliittimiä käyttämällä Redis- ja SQLite-yhteyksien rakentamisen hoitaa Flask automaattisesti.

Tekemällä kutsuja määritettyihin URL-osoitteisiin, sovellus käynnistää taustaprosesseja ja palauttaa JSON-formatoituja vastauksia (esimerkkikoodi 1).

```
@main.route("/health", methods=["GET"])
def health_check():
     return jsonify({"status": "running"}), 200
```

Esimerkkikoodi 1. API-rajapintareitti, joka palauttaa JSON-vastauksen palvelun tilasta.

3.1.2 Gunicorn

Gunicorn on Web app Gateway Interface (WSGI) http-palvelin. Sen tehtävänä on välittää http-pyynnöt verkkosovellukselle. Vaikka Flask tarjoaa http-palvelimen kehitystarpeisiin, eivät sen ominaisuudet ja tehokkuus ole niin kehittyneitä kuin projektin, joka keskittyy toimimaan täysin http-palvelimena. Gunicorn toimii monisäikeisenä, pystyy prosessoimaan useita pyyntöjä yhtäaikasesti ja mahdollistaa erilaisten asetusten asettamisen, joilla voidaan tehostaa http-palvelimen nopeutta sekä turvallisuutta.

Gunicorn mahdollistaa alustan joustavan konfiguraation sekä käyttöönoton. Alustaa voidaan käyttää sellaisenaan Gunicorn toimien web-palvelimena tai yhdistää käänteiseen välityspalvelimeen. Kehittyneemmissä ympäristöissä Gunicornin toiminnan voi helposti yhdistää välityspalvelimeen, kuten Nginxiin tai Apacheen.

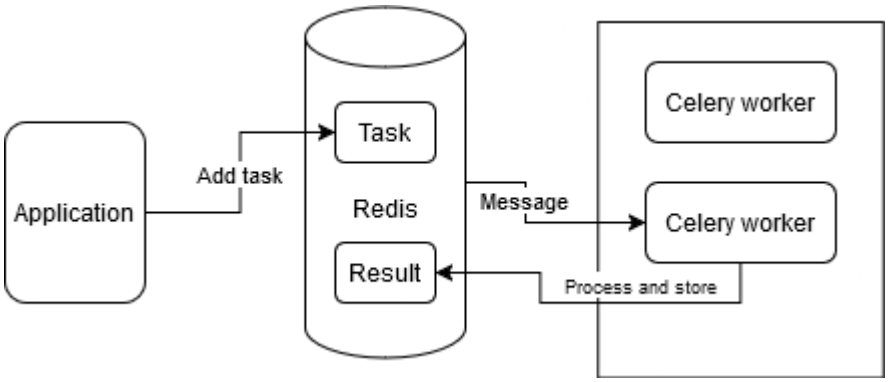
3.1.3 Celery

Celery on asynkroninen, hajautettu tehtäväjono. Tehtäväjonoon pusketaan tehtäviä, joita Celery-työntekijät suorittavat. Jonoksi ja tehtävien välittämiseksi tarvitaan viestien välittäjä, joka on tässä työssä Redis. Rediksen ylläpitämään jonoon lisätään dynaamisesti tehtäviä, jotka Redis välittää työntekijöille. Celery-

työntekijät ovat erillisiä prosesseja, joita voidaan määrittää useita toimimaan samanaikaisesti, ja ne suorittavat funktion toiminnallisuuden.

Sovelluksessa API-rajapinnan kautta tehdyt kutsut lisäävät tehtäviä Redis-tehtäväjonoon, jonka kautta Celery-työntekijä saa tarkat parametrit tarvittaville toimille. Työntekijät ovat erotettuna muusta sovelluksesta, ja ne hoitavat raskaammat tehtävät. Lisäämällä työntekijöiden määrää voidaan sovellusta skaalata helposti ottamaan vastaan suuriakin määriä pyyntöjä ilman toiminnan hidastumista.

Kehittäjälle Celeryn käyttö on yksinkertaista. Työntekijöiden suorittavat toiminnallisuudet ovat Python-koodia, joka määritetään Celery-tehtävän suoritettavaksi. Tämän lisäksi Celery-instanssi määritetään koodiin, jossa asetukset sekä kaikki sille halutut tehtävät asetetaan. Celery-työntekijä käynnistetään erillisenä taustaprosessina odottamaan sille määritettäviä tehtäviä. Jonoon pusketaan uusi tehtävä, ja Celery huolehtii sen suorittamisesta (kuva 2).



Kuva 2. Sovelluksen, Rediksen ja Celeryn toiminnan käsitemalli.

3.2 Redis

Redis on keskusmuistipohjainen varastointiin käytettävä palvelinalusta. Redis on erityisen tehokas varastoimaan ja lukemaan tallennettua dataa, koska se säilyttää kaiken keskusmuistissa. Se soveltuu toimimaan tietokantana, viestien kääntäjä- ja välittäjäpalveluna sekä välimuistina. Tässä työssä Redis toimii samaan aikaan viestien välittäjänä Celerylle sekä välimuistina tuloksille.

Välimuistiin voidaan tallentaa tiettyyn avainsanaan arvoja, joiden hakeminen vie aikaa tai on muuten raskasta. Redis vähentää näiden hitaiden ja raskaiden toimintojen määrää, kun niiden tuloksia voidaan tallentaa ja myöhemmin hakea tehokkaasti Rediksen välimuistista. Arvoille voidaan määrittää vanhentumisaika, jonka tultua täyteen arvo poistuu välimuistista ja näin pidetään se ajantasaisena. Välimuisti tehostaa sovelluksen toimintaa, vähentää vastausaikoja ja mahdollistaa skaalautumisen.

Redis toimii myös Celeryn tehtäväjonona. Celery-kirjasto sisältää valmiin toiminnallisuuden Rediksen käyttöön. Celery määritettyjä tehtäviä voidaan ohjelmallisesti puskea Rediksen tehtäväjonoon helposti, josta työntekijät saavat ne käsittelyyn. Tehtäväjono toimii ensimmäisenä sisään, ensimmäisenä ulos periaatteella.

3.3 Docker

Docker on virtualisointialusta, joka tekee kehittämisestä, testauksesta ja lopulta käyttöönotosta yksinkertaista ja yhtenäistä. Dockeriin määritetään infrastruktuuri, jonka päällä kehitetty sovellus pyörii. Docker rakentaa määritysten perusteella ”kontin”, joka sisältää sovelluksen, sen tarvitsemat kirjastot, konfiguraatiot ja alustan, joka käynnistetään eristetysti laitteen käyttöjärjestelmästä. Dockerin toimintaa voidaan verrata virtuaalikoneeseen, mutta Docker virtualisoi vain sovelluksen toiminnan kannalta tärkeät palvelut, jonka takia Docker käyttää yleensä vähemmän laitteen resursseja. Dockeria käyttämällä voidaan käynnistää ja hallinnoida useita erilaisia palveluita ja sovelluksia helposti.

Docker Compose on Dockerin työkalu, joka soveltuu määrittämään suurempia, useiden Docker-konttien kokonaisuuksia, joiden monitorointi sekä hallinnointi voidaan yhtenäistää. Docker Composen avulla voidaan rakentaa laajempi sovellus, joka sisältää kehitetyn sovelluksen, sille oman tietokannan sekä verkkopalvelimen. Jokainen palvelu pystytetään yhden konfiguraatiotiedoston avulla, docker-compose.yml. Konfiguraatioon määritetään jokainen haluttu, yksittäinen Docker-palvelu ja niiden erityisominaisuudet. Konfiguraatiotiedoston avulla

yksittäisen palvelun, kuten tietokannan vaihtaminen ei vaadi muuta kuin konfiguraatiomuutosta ja uudelleenkäynnistämistä. Docker Compose tukee myös skaalausta. Yksittäisiä Docker-palveluita voidaan käynnistää useita käynnistysvaiheessa, jolloin esimerkiksi Celery-työntekijöitä voidaan määrittää useita.

Työssä Docker Composella erillisiä Docker-kontteja hallinnoidaan yhtenäisesti. Sovelluksen eri osat eritellään yksittäisiin Docker-kontteihin, joita hallinnoidaan tarvittavien palveluiden kanssa Docker Composella. Tuloksena on alusta, joka on itsenäinen kokonaisuus.

4 Työn toteutus

Työn tavoitteena oli luoda yksittäinen sovellus kokonaisuus, joka rikastaa sille syötettyjä digitaalisia tunnistetietoja tekemällä useita sovellusrajapintahakuja uhkatietolähteisiin. Sovelluksen tarkoitus on yhdistää uhkatietojen rikastus yhteen paikkaan, näin poistaa tarve käydä useissa eri paikoissa hakemassa uhkatietoja. Sovelluksen täytyi täyttää seuraavat ominaisuudet:

- sovellusrajapinta avainten tallentaminen
- digitaalisten tunnisteiden tunnistaminen ja kategorisointi
- useiden ulkoisten sovellusrajapintojen käyttäminen.

Sovelluksen haluttiin myös toimia niin, että uusien uhkatietolähteiden lisääminen olisi mahdollisimman yksinkertaista. Toteutus pyrittiin pitämään yksinkertaisena ja monipuolisena, jotta alustan sisällyttäminen valmiisiin toteutuksiin sekä yhdistäminen muihin komponentteihin olisi helppoa ja samalla toimisi myös itsenäisenä.

Rakenteeltaan työssä päädyttiin ratkaisuun, jossa toiminnallisuutta ohjataan Flaskilla toteutetun API-rajapinnan kautta, joka käynnistää Celery-työntekijöille tehtäviä. Näin erotellaan rajapinnan toiminta raskaammista tehtävistä, joissa käydään hakemassa uhkatietoja ulkoisista API-rajapinnoista eikä Flask API -rajapinnan käyttö hidastu.

Sovellusta kehittäessä, ominaisuuksia tarkennettiin ja lisättiin sitä mukaan, kun tarve niille tuli. Kehitykseen käytettiin Visual Studio Code -koodieditoria Python-laajennuksilla, kehityksen tukena Docker Composea ja versionhallintana GitHub-alustaa. API-rajapinnan toiminnallisuuden testaamiseen käytettiin Postman-sovellusta. Docker mahdollisti uusien palveluiden helpon lisäämisen ja jatkuvan testauksen. Kehittäminen tapahtui useilla laitteilla ja käyttöjärjestelmillä riippuen siitä, mikä oli sillä hetkellä kätevintä. Docker mahdollisti yhtenäisen, alustariippumattoman kehityksen ja jatkuvan testauksen.

Sovellusta lähdettiin rakentamaan tekemällä perusta Flask API -rajapinnan ja Celery-työntekijöiden toiminnallisuudelle, jossa kutsu API-rajapintaan käynnistää Celery-työntekijän taustalle. Flask huolehtii toiminnallisuuksien orkestroinnista ja syötteiden validoinnista. Yksinkertaisilla säännöllisillä lausekkeilla syötetyt digitaaliset tunnistetiedot kategorisoidaan, ja luodaan uusi tehtävä Celeryn suoritettavaksi. Flask palauttaa tehtävän tunnisteen, jota kyselemällä saadaan tehtävän tila ja lopulta sen tulokset. Celery-työntekijä suorittaa validoidun tehtävän ja hoitaa tunnistetiedon rikastamisen useilla API-rajapintakutsuilla. Celery sekä Flask käyttävät Redistä välimuistina. Celery tallentaa välimuistiin rikastetut tunnistetiedot ja Flask tietokantaan määritetyt API-rajapinta avaimet, mikä vähentää raskaampien toimintojen käyttöä. Flask API -rajapintaa käyttämällä pystytään myös määrittämään uhkatietolähteiden API-rajapintayhteyksissä käytettäviä API-rajapinta-avaimia. Useat lähteet edellyttävät rekisteröitymisen, jonka kautta API-rajapintakutsuja pystytään tekemään.

Lähdekoodin ylläpitämiseen käytettiin git-pohjaista GitHub-palvelua, joka säilyttää muutokset sekä lopullisen koodin.

4.1 Flask API

Flask toimii välittäjänä, jonka kautta kaikkia sovelluksen toiminnallisuuksia käynnistetään ja määritetään. Flask-kirjasto määritetään ja käynnistetään Flaskin dokumentaation mukaisesti. Määrittelyssä käytetään apuluokkaa ”Config”, joka sisältää koko sovellukselle tärkeitä arvoja, kuten Redis-tietokannan osoitteen ja

kuunneltavat portit. Config-luokan muuttujat voidaan määrittää ympäristömuuttujina esimerkiksi Dockerilla, joka mahdollistaa toiminnallisuuksien helpon mukautumisen.

Flask määritetään ja luodaan __init__.py-moduulissa Flask-sovellustehdasfunktiolla, jota kutsutaan ensimmäisenä koko sovelluksen käynnistyessä. Flask varmistaa, alustaa ja käynnistää kaikki tietokantayhteydet, joita käytetään koko sovelluksen toiminnassa. Kaikki tietokantaan tallennettavat tiedot ovat Python-luokkia, joiden siirtäminen tietokantaan hoidetaan kokonaan Flaskin toimesta. Flask luo ja omistaa tietokantayhteyden. Kaikki tietokantaan tehtävät toiminnot tulisi näin tehdä Flaskin toimesta.

Pääosin Flask toimii täysin välittäjänä. Raskaat tehtävät kirjataan Redikseen ja annetaan Celeryn hoidettavaksi, jonka tuloksia viedään käyttäjälle. Koska Flask ylläpitää tietokantayhteyksiä, päätettiin Flaskin hoitaa alustan konfigurointi. Kun käyttäjä tekee muutoksia uhkatietolähteiden konfiguraatioihin, hoitaa Flask muutokset tietokantaan. Tietokantaan tehtävät muutokset ovat yksinkertaisia, ja tietokanta on paikallinen. Sen muokkaaminen on nopeaa, joten se selkeyttää alustan toiminnallisuutta, kun tuloksia saadaan suoraan http-pyyntöön.

4.1.1 API-rajapintakehitys

API-rajapinnan kehityksessä pyrittiin noudattamaan RESTful-periaatteita. Flaskin blueprint-moduulia käytettiin yhtenäistämään API-rajapinnan toiminnallisuuksia, ja varmistettiin yksinkertainen skaalautuminen, jos uusia toiminnallisuuksia täytyy lisätä.

4.1.2 API-rajapintareitit

API-rajapinnan reitit määritetään routes.py moduulissa Blueprint-dekoraattorilla, joka yhdistetään Flaskiin. Reitin rekisteröiminen ja toiminnallisuuden liittäminen siihen toimii kuin normaaleiden funktioiden määrittäminen. Esimerkkikoodi 2 rekisteröi Flaskiin reitin ”/search”, joka ottaa vastaan ”GET”-pyyntöjä, ja suorittaa

funktion ”search” toiminnallisuuden. Pyynnön body parametri ”indicator” asetetaan muuttujaan indicator ottamalla se JSON-muotoisesta pyynnöstä.

```
@main.route("/search", methods=["GET"])
def search():
     indicator = request.json.get("indicator")
     print(“Performing search...”)
```

Esimerkkikoodi 2. Pelkistetty API-rajapintareitti /search.

Sovelluksen koko toiminnallisuus haluttiin pystyä hallitsemaan Flask API -rajapinnan kautta, joten jokaiselle toiminnalle määritettiin API-rajapintaan reitti, jonka kautta niitä pystytään käynnistämään. Seuraavaksi kuvataan alustan API-rajapintareitit sekä niiden hyväksyvät http-metodit ja body-parametrit.

GET /health

/health palauttaa staattisen vastauksen ”status”: ”running”. Sen avulla sovelluksen tilaa voidaan pyytää. Docker compose käyttää kyseistä reittiä määrittääkseen, toimiiko Flask-sovellus.

DELETE /purge

/purge tyhjentää Redis-välimuistin. Reittiä käytettiin kehitysvaiheessa välimuistin toiminnallisuuden testaamisessa. Tyhjentämällä välimuisti poistetaan kaikki tehtävät sekä tallennetut tulokset. Tätä voidaan käyttää ongelmatilanteissa, sekä kun halutaan sovelluksen hakevan tuoreimmat uhkatiedot.

POST /search body: {”indicator”: “<string>”}

/search käynnistää taustaprosessin, joka rikastaa body-parametrinä annetun tunnisteen. Flask varmistaa parametrin olevan validi tunniste, lisää tehtävän tehtäväjonoon ja palauttaa tehtävän tunnisteen sekä URL:in, jolla tehtävää voidaan seurata.

GET /search/status/<task_id>

Tehtävän suorittamista ja sen tuloksia tarkastellaan tehtävän tunnisteen avulla. Mahdolliset statukset tehtävälle ovat:

- PENDING – tehtävän suorittaminen on käynnissä.
- FAILURE – tehtävän suorittaminen epäonnistui.
- SUCCESS – tehtävän suorittaminen onnistui.

Palautuksen mukana on statukseen oleellisia tietoja. FAILURE sisältää virheilmoituksen. SUCCESS palauttaa rikastuksen tulokset.

GET /sources

/sources palauttaa kaikki sovelluksen uhkatietolähteet sekä niiden konfigurointitilanteen. Osa lähteistä tarvitsee API-rajapinta-avaimen. Jos API-rajapinta-avainta ei ole asetettu, ei lähteestä haeta rikastustietoja.

GET /sources/configured

/sources/configured alauttaa listauksen uhkatietolähteistä, joille on määritetty API-rajapinta-avain. Tarjoaa tavan seurata mitä lähteitä sovellukseen on määritetty.

POST /sources/<source_id> {“api_key”: “<string>”}

Määrittää tietylle uhkatietolähteelle API-rajapinta-avaimen. Parametrina annettu API-rajapinta-avain tallennetaan sovelluksessa paikalliseen SQLite tietokantaan salattuna ja asetetaan myös välimuistiin. Sovelluksessa ei ole mahdollisuutta hakea määritettyä API-rajapinta-avainta uudestaan. Vanha API-rajapinta-avain yli kirjoitetaan, jos se on jo määritetty. Välimuistiin tallennetut tulokset tyhjennetään sen jälkeen, kun API-rajapinta-avain on määritetty. Tämä mahdollistaa uhkatietojen hakemisen välittömästi uudesta määritetystä uhkatietolähteestä.

DELETE /sources/<source_id>

Poistaa määritetyn API-rajapinta-avaimen tietokannasta ja välimuistista. Uhka-tietolähdettä, joka tarvitsee API-rajapinta-avaimen ja jolla ei sellaista ole määritettynä, ei käytetä.

GET /docs

Suunniteltu OpenAPI määritelty API-rajapintadokumentaatio. Dokumentaatiota ei ehditty viimeistelemään, joten se jätettiin pois alustalta.

RESTful-periaatetta ”Uniform interface” (yhdenmukainen rajapinta) käytettiin reittejä suunnitellessa.

4.1.3 Palautusmuoto

Sovelluksen Flask-API-rajapinta palauttaa tuloksen aina JSON-muodossa ja käyttäen standardisoituja http-vastaustiloja [10]. Sovelluksen toiminnasta palautetaan vain oleellinen tieto, jossa mukana on tietoa kyseisen toiminnallisuuden tilanteesta sekä miten sen suorittaminen onnistui. Virhetilanteista palautetaan kuvaavaa tietoa siitä, mikä aiheutti virheen.

Tunnisteiden rikastamisessa tehtävän käynnistämisestä palautetaan tehtävän tunniste ja URL, jota seuraamalla voidaan seurata tehtävän suorittamista. Suorittamisesta palautetaan senhetkinen tila. Kun rikastus on valmis, palautetaan esimerkkikoodi 3 mallinen tieto rikastetusta tunnisteesta.

```
{
  "indicator": "<indicator>",
  "type": "IPV4 | IPV6 | DOMAIN | URL | HASH | Unknown",
  "sources": [
    {
      "summary": "<summary of the threat intel source results>",
      "verdict": "ERROR | NONE | BENIGN | SUSPICIOUS | MALICIOUS",
      "url": "<url to the threat intel source>",
      "data": {<raw data in json format>}
    }, {...}
  ]
}
```

Esimerkkikoodi 3. Palautus rikastetusta tunnisteesta.

http-virhenumerot yhtenäistettiin Flaskin virheiden käsittelijä dekoraattorilla, joka yhdistetään kaavaan (esimerkkikoodi 4) Senior Lecturer. Flask-reitin sisällä voidaan kutsua ”abort”-funktiota, jolle annetaan parametrina http-vastausstatus.

```
@main.errorhandler(400)
def bad_request_error(error: Exception | str = ""):
     error_str = str(error)
     return jsonify({
          "error": "Bad Request",
          "message": error_str,
          "status_code": 400,
          "path": request.path,
          "timestamp": str(datetime.now(timezone.utc)),
     }), 400
```

Esimerkkikoodi 4. 400 Bad Request virheen palautus.

Virheviesti sisältää virheen korjaamiseen oleellisia tietoja. ”message” voidaan määrittää muualla koodissa kuvaamaan tarkemmin, mikä sai virheen aikaan. Koodiesimerkissä 4 se kuvaisi esimerkiksi, mitä parametreja kyseinen reitti vaatii.

4.1.4 OpenAPI

API-rajapinta suunniteltiin dokumentoitavaksi OpenAPI:n määritelmällä. JSON-tiedosto, joka on rakennettu kuvaamaan API-rajapinnan jokaista toiminnallisuutta, muutetaan flask-swagger-ui-moduulilla luettavaan mutoon, joka toimii API-rajapinnan dokumentointina. Flask-sovellus tarjoaisi dokumentaation reitissä /docs. Dokumentointia tehtiin, mutta sitä ei ehditty viimeistelemään, joten sitä ei sisällytetty lopulliseen alustaan.

4.2 Celery workers

Työssä tarvittiin tapa, jolla käynnistää useita http-kutsuja, prosessoi tulokset ja palauttaa ne käyttäjälle. Varhaisessa vaiheessa tunnistettiin ongelmaksi http-kutsujen tekemisessä se, että vastauksien saaminen ei ole aina johdonmukainen. Jokaista palvelua ylläpidetään omilla tavoillaan, ja vastausajoissa on paljon vaihtelevuutta. Yhden kyselyn tekemisessä voi mennä millisekunteja, kun taas

toisessa useita sekunteja. Odotellessa vastauksia alustan käyttö pysähtyi, kunnes jokainen vastaus saapui. Oli selkeää, että alusta täytyi erotella rajapinnaksi, jonka kanssa käyttäjä on vuorovaikutuksessa, sekä osaksi, joka suorittaa toimenpiteet käyttäjän tarpeiden mukaan.

Celery sopi tähän käyttötarkoitukseen. Toiminnallisuus annetaan Celeryn hoidettavaksi, joka tarpeen mukaan käynnistää taustaprosesseja. Celeryn avulla alusta pystyy käsittelemään useita pyyntöjä ilman, että sen käytössä tulee huomattavia pysäytyksiä. Uhkatiedon rikastamispyynnön saadessaan luo rajapinta uuden Celery-tehtävän jonoon ja palauttaa tehtävälle uniikin tunnisteen, jota käytetään tehtävän seuraamiseen. Celery suorittaa uhkatiedon rikastamisen, ja formatoidun tuloksen Redis-tietokantaan, samaan tunnisteeseen, josta se voidaan hakea. Celery-työntekijöiden käytön heikkous on tulosten käsittely. Koska tulokset tallennetaan välittäjäpalveluun, kuten Redikseen, jos niitä tarvitaan muualla koodissa, täytyy tuloksia odottaa ja hakea niitä jatkuvasti. Tämä haku voidaan suorittaa helposti ohjelmallisesti, mutta lisää kehitysrasitetta. Työssä docker composella käynnistetään yksi tai useampi Celery-työntekijä taustalle. Toimintojen tehokkuutta edistetään myös rinnakkaisprosessoinnilla, jossa tavoitellaan prosessin käyttävän mahdollisimman vähän aikaa vastausten odottamiseen.

Työssä Celery-työntekijät määritetään celery_worker.py-moduulissa, Celery-tehdasfunktiolla. Koska Flask alustaa ja huolehtii tietokantayhteyksistä, täytyy Celery-työntekijälle antaa oikeus käyttää Flaskin toimintoja. Celery luodaan esimerkkikoodin 5 mukaisesti niin, että sillä on mukana Flask-sovellus, jolloin saavutetaan haluttu toiminnallisuus. [11.]

```
def create_app(celery=False) -> Flask:
     app = Flask(__name__)
     if not celery:
          ... # Create API routes and seed database
     return app

def make_celery(app) -> Celery:
     celery = Celery(Config)
     return celery

flask_app = create_app(celery=True) # Create Flask without set up
celery = make_celery(flask_app) # Create Celery with Flask app context
```

Esimerkkikoodi 5. Celeryn luonti Flask-sovelluksen yhteyksillä.

Kehitysvaiheessa pyrittiin ensin poistamaan Celeryn tarve käyttää tietokantaa kokonaan ja Flask-sovellus olisi toimittanut kaikki tiedot Redis-välimuistiin, johon Celeryllä on oikeus. Tämä olisi mennyt alkuperäistä suunnitelmaa vastaan siitä, että Flask toimisi vain tehtävän välittäjänä ja Celery hoitaisi kaikki raskaammat operaatiot. Lopulta päädyttiin välimalliratkaisuun, jossa Flask lisää välimuistiin tietoja aina, kun se on ajankohtaista, ja muuten Celery käy hakemassa tiedot tietokannasta.

Työssä Celeryllä on vain yksi tehtävä, jota se pystyy suorittamaan. Flaskin /search-reitin käynnistämä toiminnallisuus lisää tehtävän jonoon Celeryn suoritettavaksi. Tehtävässä on mukana uhkatunniste, joka halutaan rikastaa.

Celery-tehtävä hakee jokaisen määritetyn uhkatietolähteen luokan lähderekisteristä, ja kutsuu jokaisen lähteen tietojen haku funktiota. Tehtävä käyttää Pythonin asynkronista kirjastoa asyncio saavuttaakseen rinnakkain suoritettavia toimintoja. Kehityksen alussa huomatut rajoitteet http-vastausten saamisessa voidaan minimoida. Tehtävä hakee kaikki määritetyt uhkatietolähteet ja lähettää jokaiseen http-kyselyn. Yhtä kyselyä odottaessa pystyy prosessi lähettämään toisia kyselyitä, sekä prosessoimaan vastauksia ja palata saatuaan vastauksen. Rinnakkaisprosessointi nopeuttaa tulosten saamista ja niiden prosessointia.

Moniprosessointia harkittiin ratkaisemaan http-kyselyiden teko. Celeryn ollessa itsessään ratkaisu, joka vastaa moniprosessointia monella tapaa, päätettiin, ettei sitä käytetä.

Tilanteissa, jossa alustalle tehdään paljon kyselyitä, ongelmaksi voi tulla uhkatietolähteiden API-rajapinnan kyselyrajoitukset. Liikenne voi myös näyttää haitalliselta, kun yhdestä kohteesta lähetetään jatkuvasti suuria määriä http-kyselyitä. Tähän ratkaisuksi valittiin rajoitin yhtäaikaisille kyselyille. Jokainen Celery-tehtävä suoritetaan kohtuullisessa ajassa, mutta yksittäisen tehtävän tekemiä yhtäaikaisia kyselyitä rajoitetaan. asyncio:n Semaphorea käytetään esimerkkikoodin 6 mukaisesti, jonka avulla voidaan rajoittaa suoritettavien rinnakkaisprosessien määrää. Jos asetettu yläraja ylitetään, odottaa prosessi, kunnes sille vapautuu tilaa suorittaa.

Tulokset formatoidaan lähteen määrityksien mukaisesti ja kerätään avainsana, arvopareihin uhkatietolähdekohtaisesti.

```
async def query_source(source: BaseSource) -> dict | None:
     semaphore = asyncio.Semaphore(Config.MAX_CONCURRENT_REQUESTS)
     async with semaphore:
          try:
               response = await source.fetch_intel(indicator, indica-
tor_type)
               return response
          except Exception as e:
               logger.error(f"Error fetching data: {e}")
               return None

tasks = [query_source(source) for source in sources.values()]
results = await asyncio.gather(*tasks)

results = {source.get_name(): result for source, result in
zip(sources.values(), results) if result}
```

Esimerkkikoodi 6. Celery-tehtävä asynkroninen toiminta.

Lopulta tulos tallennetaan välimuistiin indikaattorin nimellä, jotta tulokset voidaan hakea tulevaisuudessa ilman uusien http-kyselyiden tekoa.

4.3 Uhkatietolähteet

Työn perusta on uhkatietolähteet, joiden API-rajapinta ratkaisuista haetaan rikastusdataa. Tavoitteena oli tehdä http-hakuratkaisu, joka olisi modulaarinen, helposti muokattava ja uhkatietolähteet olisivat listattavia.

Ensimmäinen lähestymistapa oli toteutus, jossa käyttäjä määrittää itse jokaisen ulkoisen API-rajapintayhteyden. Koska jokainen ulkoinen API-rajapintaratkaisu toimii hieman eri tavalla käyttäjä olisi määrittänyt API-rajapintareitin, mitä indikaattorityyppejä API-rajapinta tukee sekä ylätunniste- ja runkomuuttuja. Tämän lisäksi olisi käyttäjän täytynyt määrittää, mitä palautusarvoille tehdään, jotta siitä saataisiin helposti luettavaa. Tämä ei sopinut haluttuihin tuloksiin, mutta selkeytti haluttua lopputulosta.

Koska ulkoset API-rajapinta ratkaisut ovat erilaisia, täytyi jokaisen uhkatietolähteen API-rajapinnan käsittely tehdä erillisesti. Jokaisesta uhkatietolähteestä päätettiin tehdä oma Python-moduuli. Moduulit sisältävät uhkatietolähdeluokan, joka toteuttaa yhtä abstraktia kanta luokkaa ”BaseSource”. BaseSource sisältää yhtenäisiä toimintoja, joilla eri lähteitä yhtenäistetään.

Käyttämällä yhtenäistä luokkaa pystytään eri uhkatietolähteitä käsittelemään samanlailla. Jokainen lähde voidaan lisätä listaan, ja käymällä lista läpi, kutsua jokaisen yhteistä tietojenhakufunktiota.

Modulaarisuus oli tärkeä osa tavoitteita. Uusi uhkatietolähde pitäisi pystyä lisäämään kirjoittamalla uusi moduuli lisäämällä se lähdekoodin lähteille tarkoitettuun pakettiin ja se ladattaisiin automaattisesti. Ensin toiminnallisuutta lähdettiin kokeilemaan ylläpitämällä tekstitiedostoa, joka jäsenneltiin läpi ja jokaisen rivin perusteella ladattiin tietystä Python-paketista moduuli.

Ajonaikainen moduulien lataus toimii käyttäen importlib-kirjastoa. Kirjastolla voidaan tuoda moduuleja käyttöön samalla tavalla kuin niitä määritetään koodista. Työssä päädyttiin luomaan lähde rekisteri yksittäisolio ”SourceRegistry”, joka lataa uhkatietolähteitä sisältävän paketin kaikki moduulit, luo ne ja täyttää hakurakenteen luoduilla luokilla (esimerkkikoodi 7).

```
class SourceRegistry:
    _instances: dict[str, BaseSource] = {}
    @classmethod
    def get_instance(cls) -> dict[str, BaseSource]:
        if not cls._instances:
            cls.load_sources()
        return cls._instances

    @classmethod
    def load_sources(cls):
        for filename in os.listdir("app/sources"):
            if filename.endswith(".py") and filename !=
"base_source.py" and filename != "__init__.py":
                module_name = filename[:-3] # Strip the .py extension
                module = importlib.import_module(f"app.sources.{mod-
ule_name}")

                for name, obj in inspect.getmembers(module):
                    if inspect.isclass(obj) and issubclass(obj, Base-
Source) and obj is not BaseSource:
                        instance = obj()
                        cls._instances[obj.__name__] = instance
```

Esimerkkikoodi 7. Uhkatietolähderekisteri.

Jokainen uhkatietolähde tehtiin erityisesti vastaamaan kyseisen API-rajapinnan toiminnallisuuksia. http-kyselyiden erityisyydet, vastausten analysointi ja formatointi käsitellään eri tavalla jokaisessa lähteessä. Kategorisointi, lopullinen formatointi sekä määritykset hoidetaan luokan BaseSource-funktioilla.

4.3.1 BaseSource

Jokainen uhkatietolähde ei tue jokaista digitaalista tunnistetta. Osa soveltuu esimerkiksi vain IP-osoitteiden rikastamiseen. Työssä digitaaliset tunnisteet määriteltiin luettelona (esimerkkikoodi 8). Luettelo yhtenäistää tunnisteiden käsittelyn ohjelman eri osissa.

```
class IndicatorType(Enum):
    IPv4 = "IPV4"
    IPv6 = "IPV6"
    DOMAIN = "DOMAIN"
    URL = "URL"
    HASH = "HASH"
    UNKNOWN = "Unknown"
```

Esimerkkikoodi 8. Tunnisteluettelo.

Jokainen tunniste omaa tietynlaisen rakenteen, josta ne voi tunnistaa. Tunnisteet syötetään ohjelmaan tekstinä. IP-osoitteet tunnistetaan ipaddress-kirjastoa käyttämällä, muut tunnisteet käyttämällä säännöllisiä lausekkeita. Tunnistettua mikä tunnistetyyppi on kyseessä, kutsutaan sitä vastaavaa funktiota ja käynnistetään oikeanlainen http-pyyntö uhkatietolähteeseen (esimerkkikoodi 9).

```
if indicator_type == IndicatorType.IPv4:
     data = await self.fetch_ipv4_intel(indicator)
     elif indicator_type == IndicatorType.IPv6:
          data = await self.fetch_ipv6_intel(indicator)
     elif indicator_type == IndicatorType.DOMAIN:
          data = await self.fetch_domain_intel(indicator)
     elif indicator_type == IndicatorType.URL:
          data = await self.fetch_url_intel(indicator)
     elif indicator_type == IndicatorType.HASH:
          data = await self.fetch_hash_intel(indicator)
     else:
          data = None
```

Esimerkkikoodi 9. Digitaalista tunnistetta vastaavan funktion kutsuminen.

Jokainen eri uhkatietolähdemoduuli toteuttaa rikastusfunktiot omalla tavallaan. Jos kyseinen uhkatietolähde ei tue tunnistetta, palauttaa se tyhjän vastauksen, jota ei oteta huomioon lopullisissa tuloksissa.

BaseSource sisältää metodit http-kutsujen tekemiselle, onnistuneiden sekä virheellisten vastausten käsittelemiselle ja tietokannasta API-rajapinta-avainten hakemiselle.

4.3.2 Rikastuksen tulokset

Loppukäyttäjälle haluttiin antaa monia vaihtoehtoja, miten tuloksia käsitellä. Uhkatietolähteet tarjoavat tuloksia eri tavoin, mutta yhtenäistä suurimmalle osalle on jokin arviointi, kuten AbuseIPDB:n ”Confidence score”, sekä tuomio tunnistetiedon haitallisuudesta. Tärkeänä työssä pidettiin myös, että yksittäisen lähteen tuloksiin pääsee helposti käsiksi sekä että rikastustiedot annetaan sellaisenaan, kuin ne lähteestä tulevat.

Tulokset annetaan avainsana, arvopareina. Päädyttiin neljään arvoon, jotka ovat:

- yhteenveto
- tuomio haitallisuudesta
- URL
- raaka data.

Yhteenvedon ja tuomion tarkoituksena on antaa loppukäyttäjälle silmäyksellä tieto siitä, mitä uhkatietolähde on mieltä tunnisteesta. Niiden avulla voidaan nopeasti arvioida, mihin suuntaan tutkintaa kannattaa lähteä viemään.

Yhteenveto koottiin uhkatietolähteen tuloksista poimien siitä tärkeitä tietoja. VirusTotalin kohdalla yhteenveto kertoo, kuinka monen virustorjuntaohjelman mielestä tunniste on haitallinen tai epäilyttävä, sekä mikä on yhteisön antama arvo sille. Tuomio määriteltiin jokaisen uhkatietolähteen arvioiden mukaan. Osa lähteistä antaa selkeän haitallinen tai epäilyttävä tuomion, mutta toisissa, kuten Tranco-listauksen kohdalla täytyi arvioida, miten tuloksen tietoja pitäisi tulkita ja määrittää tuomio niiden perusteella. Trancon kohdalla päädyttiin ratkaisuun, jossa jos tunnisteen alhaisin sijoitus on enemmän kuin 750,000 tai sillä ei ole sijoitusta, merkataan se epäilyttäväksi.

Tuomioissa käytettiin tuomioluetteloa, jossa jokaisella tuomiolla on numero arvo 0-2 välillä. Numeroarvo mahdollisti tuomion määrittämisen helposti niin, että siinä otetaan huomioon monia seikkoja. Uhkatietolähteen tuloksia voidaan arvioida yksi kerrallaan, ja jos niiden tulokset vaikuttavat epäilyttäviltä, voidaan tuomion arvoa nostaa yhdellä ylöspäin.

4.4 Tiedon varastointi

Kehityksen aikana tunnistettiin kaksi tarvetta tiedon varastoimiselle: API-rajapinta-avainten pysyvä tallentaminen, sekä välimuistin hyödyntäminen.

Useat uhkatietolähteet vaativat API-rajapinta-avaimen käyttöä. API-rajapinta-avain sisällytetään http-pyyntöön todentamaan pyynnön lähettäjä. API-rajapinta-avaimet tallennetaan alustalle, jotta niitä voidaan käyttää jatkuvasti tekemään todennettuja pyyntöjä uhkatietolähteisiin. Tarve tietojen varastoinnille oli kehityksen alussa hyvin yksinkertainen. Tarvittiin tapa yhdistää uhkatietolähde ja sille API-rajapinta-avain. Avainarvoparien käyttö soveltui tähän hyvin. Tekstitiedosto, joka oli JSON-muodossa, soveltui ylläpitämään API-rajapinta-avaimia. Tiedostosta avaimen hakeminen uhkatietolähteen kohdalta onnistui, sekä avainten lisääminen ja muokkaaminen. Kehityksen edetessä tiedoston ylläpitäminen osoittautui haasteelliseksi. Uhkatietolähdemoduulien lisääminen tai niiden poistaminen vaati sitä, että tiedostoa muokattiin vastaamaan senhetkisiä tilannetta. Toiminnallisuudessa huomattiin myös puutteelliseksi se, että saatavilla olevia sekä valmiita uhkatietolähteitä oli vaikea listata.

SQLAlchemy on Python-kirjasto, joka muuntaa koodin luokkia tietokantaan ja tietokannasta takaisin luokiksi. Flaskin flask_sqlalchemy-kirjasto sisältää toiminnallisuudet pystyttää tietokantayhteys, sekä muuntaa luokkia tietokannan ja koodin välillä. Tietokannaksi valittiin SQLite, sen yksinkertaisuuden takia. SQLite ei vaadi erillistä tietokanta ohjelmistoa, vaan tietokantaa luetaan ja sinne kirjoitetaan SQLite-kirjastojen avulla. Tietokanta on yksittäinen tiedosto, jota ohjelmat käsittelevät SQLite-kirjastojen avulla. Tietokannassa on kaksi tietokanta pöytää. Ensimmäiseen tallennetaan merkintä jokaisesta uhkatietolähde moduulista. Toinen on listaus API-rajapinta-avaimista, joista jokaiseen on liitetty viiteavaimena sen uhkatietolähde. Tietokanta alustetaan Flaskin käynnistyessä lukemalla uhkatietolähde rekisteri ja tallentamalla jokainen luokka instanssitietokantaan jättäen väliin ne, jotka tietokannasta jo löytyvät.

Tietokantaa käyttämällä pysytään yhdenmukaisena uhkatietolähde moduulien kanssa. API-rajapinta-avainten haku suoritetaan tekemällä hakuja tietokantaan. Uhkatietolähdemoduuli pyrkii ensin hakemaan API-rajapinta-avaimen välimuistista ja jos sitä ei löydy, käydään se hakemassa tietokannasta, sekä tallennettaan se välimuistiin. Kaikkien uhkatietolähde moduulien listaus suoriutuu myös tekemällä tietokantahakuja.

4.4.1 Välimuisti

Redis toimii työssä välimuistina. Se otettiin käyttöön, koska Celery tarvitsee viestin välittäjää tehtävien saamiseen ja tulosten varastoimiseen. Välimuistia haluttiin hyödyntää myös vähentämään tarvittavien operaatioiden määrää. Koska Redis pyörii täysin välimuistissa, on sinne tiedon varastoiminen ja tiedon hakeminen tehokasta. Työssä välimuistiin tallennetaan rikastetun tunnisteen tiedot määräajaksi. Jos sama tunniste halutaan rikastaa, ei alustan tarvitse käydä kysymässä jokaisesta uhkatietolähteestä tietoja uudestaan, vaan välimuistista palautetaan aikaisempi tieto. Tämä vähentää käyttöä ja kutsujen määrää ulkoisiin API-rajapintapalveluihin, jotka voivat rajoittaa API-rajapintojen käyttöä tiettyihin puitearvoihin.

4.4.2 Salaus

API-rajapinta-avaimet ovat arkaluontaisia tietoja. Ne toimivat salasanan tavoin. API-rajapinta-avaimen myöntäjä pystyy varmistamaan, kuka palvelua käyttää ja myöntää pääsyn käyttämään sen ominaisuuksia. Alusta ei ikinä palauta käyttäjälle sinne tallennettuja API-rajapinta-avaimia. Se tukee vain uuden lisäämistä tai kokonaan poistamista. Tämä oli tiedostettu toiminnallisuus, vähentääkseen riskiä siihen, että API-rajapinta-avaimiin voitaisiin päästä luvattomasti. API-rajapinta-avaimia säilytetään silti SQLite-tietokantatiedostossa ja parhaiden käytäntöjen mukaan arkaluontoisia tietoja ei pidä säilyttää salaamattomana [12]. Alustalle tallennetut API-rajapinta-avaimet salataan symmetrisellä salauksella. Alustalle määritetään salausavain sen pystytyksen yhteydessä, jota alusta käyttää salaamaan tietokantaan tallennettavat API-rajapinta-avaimet. Kun tietokannasta haetaan API-rajapinta-avain, salaus puretaan ja selkokielinen avain palautetaan.

Työssä käytetään Fernet-moduulia. Fernet-salaus salaa halutun viestin annetulla avaimella niin, että salaus voidaan purkaa vain samalla avaimella. [13.]

4.5 Docker

Dockerin käyttö työssä oli selkää alusta alkaen. Sitä hyödyntämällä nopeutettiin alustan testaamista ja pystyttämistä. Docker Composella konttien pystytys ja niiden määrittäminen onnistuu konfiguraatiotiedoston avulla. Konfiguraatiotiedosto auttaa pysymään järjestelmällisenä Dockerin käytössä.

Dockerin eri ominaisuuksia pyrittiin hyödyntämään työssä aina, kun se oli sopivaa Docker Compose -määrityksissä. Näissä määrityksissä pyrittiin huomioimaan ympäristö, jossa on jo käytössä useita Docker-kontteja. Alustan jokainen osa asetettiin samaan virtuaaliseen verkkoon ja ainoa osa, joka on avoin ulkopuolelle, on Flask API -rajapinta. Jokainen kontti tuottaa lokeja toiminnoistaan sekä ongelmista Dockerin lokeille. Lokien siivouksesta pidetään huolta määrittelemällä yläraja lokien määrälle.

Kaikki dynaamiset asetukset, joilla alustan toimintaa määritetään, asetettiin Docker-konteille ympäristömuuttujiksi. Tavoitteena oli, että alustan pystyy pystyttämään nopeasti ilman suuria määritysten muutoksia.

4.5.1 Containers

Työssä luotiin kaksi Docker-konttia, yksi Flask API -rajapinnalle ja toinen Celery-työntekijöille. Tämän lisäksi käytettiin Rediksen virallista konttia.

Flask- ja Celery-kontit molemmat pohjautuvat python:3.12-slim-konttiin ja molemmat sisältävät koko lähdekoodin. Flask-konttiin lisätään curl-työkalu, jota käytetään API-rajapinnan tarkasteluun, sekä gunicorn Python -kirjasto, joka tarjoaa Flask-sovelluksen. Celery-konttiin lisätään celeryworker-käyttäjä, jolla Celery-taustaprosessi käynnistetään. Tällä varmistetaan, ettei Celeryllä ole korkeampia oikeuksia, kuin mille on tarve.

4.5.2 Healthcheck

Alustan vakauden ylläpitämiseksi Docker Composeen määritettiin jokaiselle kontille healthcheck. Healthcheck on määritettävä testi, jonka Docker suorittaa ja arvioi testin tuloksilla kontin palveluiden saatavuuden. Testi kohdistetaan jotakin toiminnallisuutta kohti ja sillä arvioidaan, toimiiko kontti oikein. Jos toiminnallisuudessa huomataan ongelmia, käynnistää Docker Compose kontin uudestaan käyntiin, joka yleensä korjaa ongelmat. Kontit tarvitsevat toisien konttien toiminnallisuuksia toimiakseen oikein. Valvomalla toiminnallisuutta varmistutaan siitä, että alusta pysyy saatavilla oikein.

4.6 Testaus

Kehityksen aikana toiminnallisuuksien säännöllinen testaus on hyvä käytäntö, jolla varmistetaan kehityksen tuottavan oikeita tuloksia. Työn kehityksessä käytettiin manuaalista testausta koodin läpikäymiseen ja API-rajapinnan testausta sekä kuormatestausta Postman-työkalulla.

Dockerin avulla koodiin tehdyt muutokset voitiin nopeasti testata oikealla alustalla pystyttämällä alusta oikeasti pystyyn. Yksittäiset toiminnallisuudet testattiin ensin manuaalisesti, jonka jälkeen ne yhdistettiin kokonaiskuvaan ja testattiin kokonaisuudessaan.

4.6.1 Kehityksen aputoiminnot

Koodin toiminnallisuudet jaettiin kehityksen aikana moduuleihin niin, että niiden kehittäminen ja hallinnointi ei olisi raskasta. Yksittäisistä toiminnallisuuksista tehtiin funktioita, jotka nopeuttivat virheiden löytämistä ja korjaamista. Kehittämisen tukena luokat ja funktiot kommentoitiin docstringeillä, sekä koodin osiin lisättiin huomiot syötteisiin ja tulosteisiin. Kommentit sekä huomiot eivät vaikuta koodin toimivuuteen. Kehitysalustat käyttävät näitä toimintoja näyttääkseen hyödyllisiä apumerkintöjä, kuten millaista tietotyyppiä funktio palauttaa.

4.6.2 API-rajapinnan testaus

Postman-työkalulla voidaan rakentaa muokattavia, helposti toistettavia http-kutsuja. Työkalu sisältää myös ominaisuuksia, joilla voi automatisoida http-kutsujen lähetyksen. Työkalulla testattiin jokainen API-rajapinnan reitti ja varmistettiin, että ne hyväksyvät oikeita muuttujia ja palauttavat yhtenäisiä vastauksia sekä virheilmoituksia.

Alustan kuormasietoisuus testattiin Postman-työkalulla kehitysvaiheen lopussa. Postmanin ominaisuuksilla tehtiin kuormatesti lähettämällä satunnaisia rikastuspyyntöjä API-rajapinnan /search-reittiin, käyttäen 100 IP-osoitteen listaa. Testin aikana tarkkailtiin Flask-sovelluksen toimintaa, sekä miten Celery-työntekijä selviytyy suuresta määrästä tehtäviä. Testin tuloksista selvisi, että Flask suorittaa suuren määrän pyyntöjä ilman ongelmia. Celery toimi odotetulla tavalla, eikä sen toiminta estynyt. Rikastuksien valmistuttua, Celery vei tuloksia Redikseen talteen, ja sen toiminta nopeutui, kun jo valmiille rikastuksille alkoi tulemaan pyyntöjä.

5 Tulokset

Työn tuloksena saavutettiin joustava API-rajapinta, joka yhdistää useita uhkatietolähteitä yhteen paikkaan ja johon on helppo kehittää uusia uhkatietolähteitä moduulien muodossa. Alusta on täysin API-rajapinta pohjainen, joka tarjoaa rikastustietoja monessa muodossa. Alustaa voidaan käyttää antamaan yhteenvetoja, sekä tuomaan eri uhkatietolähteiden rikastustiedot kokonaisuudessaan. API-rajapintamallin kautta, alustan yhdistäminen olemassa oleviin järjestelmiin tai menettelytapoihin rikastamaan niissä käsiteltäviä digitaalisia tunnisteita.

Lopullisen työn lähdekoodin löytää työn tekijän henkilökohtaisesta GitHub-kuvauskannasta.

5.1 Käyttöönotto

Alusta rakennetaan Docker Composella ja sisältää usean Docker-kontin. Laitella, jolle työ sijoitetaan, täytyy olla asennettuna Docker sekä Docker Compose -liitännäinen. Työ otetaan käyttöön kloonaamalla sen lähdekoodi GitHub-projektista asettamalla docker-compose.yml-tiedostoon turvallinen salainen avain ja kutsumalla Docker Composen komentoja rakentamaan sekä käynnistämään sovellus esimerkkikoodi 10 mukaisesti.

```
root@server:/opt/docker/threat-lense# docker compose build
...
root@server:/opt/docker/threat-lense# docker compose up –detach
...
root@server:/opt/docker/threat-lense# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS      PORTS      NAMES
...
```

Esimerkkikoodi 10. Docker Composella sovelluksen rakentaminen, käynnistäminen ja käynnistyksen varmistaminen.

Riippuen siitä, miten alustaa halutaan käyttää sekä millaiseen ympäristöön se asennetaan, voidaan alustan ominaisuuksia määritellä ympäristöön sopiviksi.

Tietokannan salaukseen käytetään salaista avainta, joka annetaan Docker ympäristömuuttujana. Salainen avain täytyy vaihtaa oletusarvosta toiseen, jotta tietokannan eheys varmistetaan. Salainen avain määritellään docker-compose.yml tiedostosta, muuttamalla app palvelun ”environment”-muuttujista SECRET_KEY-arvoa.

Yksinkertaisessa ympäristössä, kuten henkilökohtaisessa kotiverkossa, jossa pyörii yksittäinen palvelin ja alustaan käytetään vain henkilökohtaiseen käyttöön, alusta toimii ilman eritysmäärityksiä.

Kehittyneemmässä ympäristössä voidaan alustaa käyttää yhdessä toisen järjestelmän kanssa, tai muiden Docker-konttien kanssa. Alustan voidaan yhdistää toiseen Redis-tietokantaan asettamalla Flask- sekä Celery-palveluiden ympäristömuuttujat haluttuun osoitteeseen. Flaskin kuuntelema portti voidaan myös

määrittää asettamalla FLASK_PORT-ympäristömuuttuja, jos oletusportti 5000 on jo käytössä.

Flaskia tarjoama Python.verkkopalvelin Gunicorn toimii tehokkaasti käänteisten välityspalvelinten kanssa. Jos ympäristössä on käänteinen välityspalvelin kuten ngixn käytössä, suositellaan määrittämään se tekemään yhteydet alustaan.

5.2 Toiminnallisuus

Alustalle tulee asettaa API-rajapinta-avaimet osalle sen tukemista uhkatietolähteistä. Kaikki alustan tukemat uhkatietolähteet voidaan listata lähettämällä GET-pyyntö reittiin /sources. VirusTotal vaatii API-rajapinta-avaimen, joka voidaan luoda ilmaiseksi rekisteröitymällä. API-rajapinta-avaimen hankkimisen jälkeen voidaan VirusTotal-lähde konfiguroida sillä lähettämällä POST-pyyntö /sources-reittiin, body-parametrina ”api_key”: <API-RAJAPINTA-AVAIN>. Tämän jälkeen alustalle tehdyt rikastuspyynnöt pystyvät hakemaan tietoja myös VirusTotalista.

Kun tietty digitaalinen tunniste halutaan rikastaa, tehdään POST-pyyntö /search-reittiin, body-parametrina ”indicator”: <INDICATOR>. Palautuksena on URL, jonka kautta rikastuksen tulokset saadaan. Pyytämällä tätä URL-osoitetta, joka on mallia /search/status/<ID>, saadaan rikastuksen tulokset JSON-muodossa.

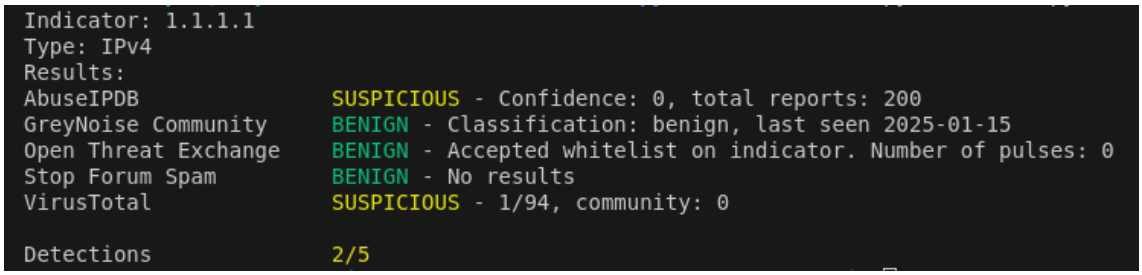
5.3 Käyttö

Alusta nopeuttaa digitaalisten tunnisteiden tunnistamista ja auttaa analysoimaan niitä tehokkaammin. Alusta on suunniteltu toimivan osana valmiita järjestelmiä, jotka käsittelevät tietoturvatapahtumia ja niihin liittyviä digitaalisia tunnisteita.

Järjestelmän havaitessa digitaalinen tunniste osaksi tietoturvatapahtumaa voidaan se automaattisesti käyttää työn alustan läpi ja saada tuloksena uhkatietoja liittyen tunnisteeseen. Alustan määrittelemiä tuomioita voidaan käyttää

priorisoimaan tapahtumia, mitkä vaativat mahdollisesti ensimmäisenä huomiota, tai antamaan järjestelmän käyttäjille lisätietoja tunnisteista.

Alustan toinen käyttötapaus on tietoturvatapahtumien analysoinnin tukena. Tietoturvatapahtumia tutkiessa voidaan nopeasti saada analysoitua tietoa digitaalisten tunnisteiden laadusta. Alustan toimintojen ympärille voidaan kehittää verkkokäyttöliittymä, johon syöttää tunnisteita ja saada tiedot jokaisen yksittäisen uhkatietolähteen huomioista. Toimintojen esittämiseksi kehitettiin komentorivityökalu, johon käyttäjä voi syöttää yhden tai useamman digitaalisen tunnisteen. Työkalu tekee API-rajapintakyselyt käyttäjän puolesta automaattisesti ja näyttää kaikki tulokset yhteenvetona (kuva 3). Se toimii analyysien apuna aikaisemmin mainitulla tavalla ja poistaa tarpeen käyttää jokainen yksittäinen löydös monien alustojen kautta. Eri digitaalisten tunnisteiden määrän noustessa satoihin, ei niiden manuaalinen rikastaminen ole järkevää ilman työkalua, joka automatisoi toiminnan.



Kuva 3. Komentorivityökalun tulokset

Tämä komentorivityökalu ottaa yhden tai useamman digitaalisen tunnisteen, tekee jokaiselle rikastamispyynnön API-rajapinta-alustalle ja odottaa rikastamisen valmistumista (esimerkkikoodi 11). Rikastamisen tulokset näytetään käyttäjälle kompaktissa muodossa, jossa ilmoitetaan, mikä jokaisen uhkatietolähteen tulos on.

```
def poll_status(status_url: str) -> dict:
     while True:
          status_response = requests.get(status_url)
          status_response.raise_for_status()
          status_data = status_response.json()
          state = status_data.get(“state”, “”)
          if state == “SUCCESS”:
               return status_data.get(“result”, {})
          elif state == “FAILED”:
               print(f“Task failed: {status_data.get(“status”)}”)
               return None
          else:
               time.sleep(1)
```

Esimerkkikoodi 11. Rikastamisen valmistumisen varmistaminen.

5.4 Jatkokehitys

Työssä päästiin tavoitteisiin ja saatiin tulokseksi alusta, jolla tarvittavat ominaisuudet. Ominaisuuksiin jäi puutteita sekä yksinkertaisia toteutuksia, joita voisi kehittää pidemmälle.

Todentaminen API-rajapinta kutsuille. Työn API-rajapinnan reitit eivät tue todentamista ja Flask-sovellus prosessoi jokaisen saamansa pyynnön. Jotta alustan käyttö ja ylläpitäminen olisi mahdollisimman yksinkertaista, ei työhön lähdetty kehittämään todentamista. Jatkokehityksenä todentamisen tukeminen ja käyttäjien sekä käyttäjäryhmien tekeminen olisi etusijalla.

OpenAPI-dokumentoinnin viimeistely. Dokumentointi nopeuttaisi API-rajapinnan käyttöä, kun jokainen toiminnallisuus löytyy OpenAPI-dokumentoinnista.

API-rajapintamalli mahdollistaa alustan käyttäjän rakentaa alustan ympärille tarpeilleen sopivan käyttöliittymän. Kehityskohteena olisi rakentaa alustalle sen ominaisuuksia käyttävä verkkosovellus, jonka kautta sitä voitaisiin konfiguroida sekä käyttää rikastamaan tunnisteita.

Tietokannan salaus toteutettiin yksinkertaisesti ja jälkeenpäin sen jälkeen, kun tietokannan toiminnallisuus oli jo pystyssä. Turvallisempi ratkaisu olisi salata koko tietokanta, eikä vain tiettyjä arvoja. Minimaalisin toimenpitein SQLite-

tietokanta voitaisiin vaihtaa SQLCipher, vaihtoehtoiseen SQLite-haaraan, joka tukee tietokannan salausta.

Testausten laajentaminen. Alustan kuormansietoisuutta testattiin ja sitä käytettiin lokaalisti yksityiskäyttöön. Alusta tulisi testata osana isompaa järjestelmää, joka käsittelee suuria määriä uhkatietoja ja sen pystytystä sekä mahdollisesti toimintaa, sopeuttaa tulosten mukaan.

Uhkatietolähteitä on yksinkertaista lisätä eikä niiden yhdistäminen vaadi lisätoimenpiteitä. Jatkokehityksessä uusia uhkatietolähteitä tulisi lisätä sitä mukaan, kun löytyy luotettavia ja lisäarvoa antavia lähteitä.

6 Yhteenveto

Tässä työssä tutkittiin, kuinka avointen lähteiden tiedustelutietoja voidaan käyttää uhkatietojen rikastamiseen. Uhkatiedot ovat tärkeä osa tietoturvaloukkausten torjuntaa ja tunnistamista. Niiden rikastaminen monipuolisilla, luotettavilla tiedoilla toimii tukena tutkinnoissa, antaa syvempää kontekstia tapahtumista, sen mahdollisista vaikutuksista ja ohjaa keskittymään olennaisiin tunnisteisiin.

Työn tuloksena on API-rajapintapohjainen alusta, joka kehitettiin tueksi digitaalisten tunnisteiden rikastamisessa. Alusta mahdollistaa digitaalisten tunnisteiden automaattisen rikastamisen useilla uhkatietolähteillä, antaa monipuolista, ajantasaista lisätietoa tunnisteille ja vähentää manuaalisesti uhkatietolähteiden käyttöä. Se yksinkertaistaa useiden uhkatietolähteiden käytön yhdistämällä ne yhteen alustaan, joka voidaan joustavasti ottaa käyttöön valmiiden järjestelmien tai työtapojen kanssa. Alusta on rakennettu käsittelemään suuria määriä uhkatietoja, ja se on suunniteltu skaalautumaan tarpeiden mukaan.

Työssä saavutettiin tavoite alustan kehityksessä. Alustan päätoiminnallisuus saatiin kehitettyä, mutta useita tärkeitä toiminnallisuuksia jäi uupumaan. Jatkokehityksessä pitäisi keskittyä dokumentoinnin viimeistelyyn, uhkatietolähteiden

lisäämiseen, API-rajapinta-avainten salaukseen sekä pyyntöjen autentikoimiseen.

Tämä työ korostaa uhkatietojen käyttämisen sekä niiden rikastamisen tietoturvatoiminnassa. Tietoturvaloukkausten määrien kasvaessa uhkatietojen automatisoitu käyttö ja haitallisten tunnisteiden tunnistaminen vahvistavat tietoturvaloukkausten torjumista sekä helpottavat niiden löytämistä.

Lähteet

1. 2024. 2024 Threat Detection Report. Verkkoaineisto. Red Canary. <https://resource.redcanary.com/rs/003-YRU-314/images/2024ThreatDetectionReport_RedCanary.pdf>. Luettu 17.11.2024.

2. Aurelija Einorytė. 2023. Types of IP addresses: All you need to know. Verkkoaineisto. NordVPN. <https://nordvpn.com/fi/blog/types-of-ip-addresses/> Päivitetty 18.7.2023. Luettu 17.11.2024.

3. Get-FileHash. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash?view=powershell-7.4> Luettu 16.11.2024.

4. What is a Firewall? Verkkoaineisto. Cisco Systems. <https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-firewall.html> Luettu 28.11.2024.

5. What is Threat Intelligence? 2022. Verkkoaineisto. International Business Machines IBM. <https://www.ibm.com/topics/threat-intelligence> 2.11.2022. Luettu 21.11.2024.

6. What Is a Honeypot? Verkkoaineisto. Fortinet. <https://www.fortinet.com/resources/cyberglossary/what-is-honeypot> Luettu 19.11.2024.

7. Frequently Asked Questions – AbuseIPDB. Verkkoaineisto. AbuseIPDB. <http://www.abuseipdb.com/faq.html> Luettu 17.11.2024.

8. Understanding GreyNoise Datasets. Verkkoaineisto. GreyNoise. <https://docs.greynoise.io/docs/understanding-greynoise-data-sets> Päivitetty 3.12.2024. Luettu 17.11.2024.

9. Flask User’s Guide. Verkkoaineisto. Pallets. <https://flask.palletsprojects.com/en/stable/> Luettu 18.12.2024.

10. HTTP response status codes. Verkkoaineisto. Mozilla. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status> Luettu 18.12.2024.

11. Background Tasks with Celery. Verkkoaineisto. Pallets. <https://flask.palletsprojects.com/en/stable/patterns/celery/> Luettu 18.12.2024.

12. Narendran Vaideeswaran. 2023. Data Encryption Explained. Verkkoaineisto. CrowdStrike. <https://www.crowdstrike.com/en-us/cybersecurity-101/data-protection/data-encryption/>. Luettu 4.1.2025.

13. Tom Maher. 2013. Fernet Spec. Verkkoaineisto. GitHub. <https://github.com/fernet/spec/blob/master/Spec.md>. Päivitetty 4.9.2014. Luettu 8.1.2025.