Sampo Bredenberg

Uhkatietojen rikastus julkisten lähteiden tiedustelutiedoilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tutkinto-ohjelman nimi

Insinöörityö

31.12.2024

Tiivistelmä

Tekijä: Sampo Bredenberg

Otsikko: Uhkatietojen rikastus julkisten lähteiden tiedustelutiedoilla

Sivumäärä: xx sivua + x liitettä

Aika: 31.12.2024

Tutkinto: Insinööri (AMK)

Tutkinto-ohjelma: Tieto- ja viestintätekniikka

Ammatillinen pääaine: Ohjelmistotuotanto

Ohjaajat: Tehtävänimike Etunimi Sukunimi  
Tehtävänimike Etunimi Sukunimi

**Tiivistelmä**

Avainsanat: avainsana, avainsana

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tämän opinnäytetyön alkuperä on tarkastettu Turnitin Originality Check -ohjelmalla.

Abstract

Author: First Name Last Name

Title: Title of the Thesis

Number of Pages: xx pages + x appendices

Date: 1 March 2021

Degree: Bachelor of Engineering

Degree Programme: Name of the degree programme

Professional Major: Name of the professional major

Supervisors: First name Last name, Title (e.g., Project Manager)

First name Last name, Title (e.g., Principal Lecturer)

Tiivistelmän tekstiosuus kirjoitetaan niin, että se mahtuu sivulla käytössä olevaan tilaan. Tekstiosuudessa käytetään Leipäteksti ilman välistystä-tyyliä.

Keywords: Keyword, Keyword

Sisällys

Lyhenteet

[1 Johdanto 8](#_Toc187268938)

[2 Lähtökohdat 8](#_Toc187268939)

[2.1 Uhkatiedot 8](#_Toc187268940)

[2.1.1 Digitaaliset tunnistetiedot 9](#_Toc187268941)

[2.1.2 Uhkatietojen kerääminen 11](#_Toc187268942)

[2.1.3 Uhkatietojen käyttö 13](#_Toc187268943)

[2.2 Julkisiin lähteisiin perustuva tiedustelutieto 14](#_Toc187268944)

[2.2.1 Avoimet lähteet 15](#_Toc187268945)

[2.2.2 Kaupalliset toimijat 18](#_Toc187268946)

[2.2.3 Joukkoistettu tiedustelutieto 19](#_Toc187268947)

[2.3 Uhkatietojen rikastaminen 19](#_Toc187268948)

[3 Teknologiat 20](#_Toc187268949)

[3.1 Python 21](#_Toc187268950)

[3.1.1 Flask 21](#_Toc187268951)

[3.1.2 Gunicorn 22](#_Toc187268952)

[3.1.3 Celery 22](#_Toc187268953)

[3.2 Redis 23](#_Toc187268954)

[3.3 Docker 24](#_Toc187268955)

[4 Työn toteutus 25](#_Toc187268956)

[4.1 Flask API 26](#_Toc187268957)

[4.1.1 API kehitys 27](#_Toc187268958)

[4.1.2 API reitit 27](#_Toc187268959)

[4.1.3 Palautusmuoto 30](#_Toc187268960)

[4.2 Celery workers 31](#_Toc187268961)

[4.2.1 Celery tasks 32](#_Toc187268962)

[4.3 Uhkatietolähteet 34](#_Toc187268963)

[4.3.1 BaseSource 36](#_Toc187268964)

[4.3.2 Rikastuksen tulokset 37](#_Toc187268965)

[4.4 Tiedon varastointi 38](#_Toc187268966)

[4.4.1 Välimuisti 39](#_Toc187268967)

[4.4.2 Salaus 39](#_Toc187268968)

[4.5 Docker 40](#_Toc187268969)

[4.5.1 Containers 41](#_Toc187268970)

[4.5.2 Healthcheck 41](#_Toc187268971)

[4.6 Testaus 41](#_Toc187268972)

[4.6.1 Kehityksen aputoiminnot 42](#_Toc187268973)

[4.6.2 API endpoint testaus 42](#_Toc187268974)

[5 Tulokset 42](#_Toc187268975)

[5.1 Toiminnallisuus 43](#_Toc187268976)

[5.2 Käyttöönotto 43](#_Toc187268977)

[5.2.1 Yksinkertaisessa ympäristössä 43](#_Toc187268978)

[5.2.2 Kehittyneemmässä ympäristössä 43](#_Toc187268979)

[5.3 Käyttö 43](#_Toc187268980)

[5.4 Jatkokehitys 43](#_Toc187268981)

[6 Yhteenveto 44](#_Toc187268982)

Liitteet

Liite 1: Liitteen nimi

Liite 2: Liitteen nimi

Lyhenteet

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

Tietoturvaloukkaus:

Uhkatekijä:

Digitaalinen tunnistetieto:

SIEM:

Brute-force:

# Johdanto

Uhkatiedot ovat todisteita haitallisesta toiminnasta IT-ympäristöissä. Uhkatietoja keräämällä ja niitä tutkimalla pyritään paljastamaan aktiivinen väärinkäyttö, sekä väärinkäyttöyritykset.

* Aidon datan kerääminen rajoitettua, yleensä vain oikeista ympäristöistä dataa
* Työlästä tunnistaa suuria määriä haitallista toimintaa
* Hyötyä muiden tehdystä ja automatisoidusta työstä

# Lähtökohdat

Tietoturva on tärkeä osa tietojärjestelmiä, jolla turvataan järjestelmien luotettava toiminnallisuus. Tietoturvan tärkeyttä korostetaan ja tietoisuutta jaetaan laajemmin kuin koskaan. Tässä luvussa pohjustetaan tietoturvapoikkeamien ja haitallisten uhkien tutkimiselle olennaisia käsitteitä, jotka ovat digitaalisia tunnistetietoja. Opinnäytetyön lopputuloksena on sovellus ratkaisu, joka mahdollistaa uhkatietojen keskitetyn rikastamisen. Sovelluksen avulla tietoturva-ammattilaiset voivat tehostaa työtään oleellisiin kohteisiin ja havaita tietoturvapoikkeamat nopeammin.

## Uhkatiedot

Uhkatiedot ovat tapoja ja tunnistettavia tietoja, joita on käytetty tietoturvaloukkauksissa ja väärinkäyttöyrityksissä. Niitä kerätään prosessoimalla tutkimalla tietoturvaloukkauksista ja tunnistamalla mitä hyökkääjä on tehnyt. Tässä opinnäytetyössä rajataan uhkatiedot digitaalisiin tunnistetietoihin, joita ovat:

* IPv4- ja IPv6-osoitteet
* Verkkotunnisteet (domain)
* Verkko-osoitteet (URL) **Vaihda verkko-osoitteet -> URL:it**
* Tiedostojen hajautusarvot (hash) **tarkistelaskenta?**

Digitaalisia tunnistetietoja löytyy palomuuri- ja kirjautumislokeilta lähtö- tai kohdeosoitteina, sekä laitteiden toiminnasta verkkoliikenteenä sekä tiedostoina. Uhkatiedot ovat ennestään tunnistettuja uhkatoimijoiden käyttämiä haitallisia tiedostoja, sekä tietoverkko liikennettä uhkatoimijoiden hallitsemiin verkko-osoitteisiin. Uhkatietojen avulla pyritään paljastamaan aktiivisia tietoturvaloukkauksia tai jopa torjumaan hyökkäyksiä ennakoivasti, kun uhkatekijöille ominaisia toimintatapoja valvotaan ennestään tunnistetuilla digitaalisilla tunnistetiedoilla.

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 saamaan tietyn tasoiset käyttöoikeudet. Tiedostoja voidaan tunnistaa hajautusarvojen perusteella.

Verkkotunnisteet isännöivät haitallisia tiedostoja tai muita resursseja kuten kalastelu sisältöä, joiden kautta saadaan ensimmäinen pääsy ympäristöön. Ne toimivat myös osoitteina, joihin viedään luvattomasti tietoaineistoa ympäristöstä, ja komentokeskuskina, joiden kautta voidaan komentaa haittaohjelmia toimimaan halutulla tavalla.

### Digitaaliset tunnistetiedot

IP-osoitteet toimivat tietoliikenteen perustana. Kun tietoa halutaan lähettää, vastaanottaa tai hakea toiselta laitteelta, täytyy laitteen osoittaa mistä liikenne liikkuu mihin. IP-osoitteet toimivat lähtö- ja kohdeosoitteina, joiden avulla voidaan reitittää mikä laite lähettää tietoja, minne. IP-osoitteet voidaan jakaa neljään tyyppiin:

* Julkiset IP-osoitteet
* Yksityiset IP-osoitteet
* Dynaamiset IP-osoitteet
* Staattiset IP-osoitteet

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.

[IP-osoite tyypit] <https://nordvpn.com/fi/blog/types-of-ip-addresses/> 17.11

Verkkotunniste on usein ihmisymmärrettävä merkkijono, joka tarvitsee aina IP-osoitteen, jonka takana jotain palvelua pidetään yllä. Verkkotunnisteet toimivat hierarkialla, jossa tasot erotetaan pisteellä. Ylätason verkkotunnusta (top-level domain) 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 koko toisen tason verkkotunniste osallistuu haitalliseen toimintaan.

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

Kaikkiin verkkoliikenteeseen liittyvien tunnistetietojen kautta voidaan suorittaa samoja haitallisia toimia, koska IP-osoite, verkkotunniste ja osoite ovat kaikki jokseenkin riippuvaisia toisistaan. Näiden tunnistetietojen takana voidaan isännöidä haitallisia tiedostoja, joita huijataan laitteen käyttäjä lataamaan, tai mahdollistetaan aktiivisen haittaohjelman käydä lataamassa 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. 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 saman, joten tallennettuja tuloksia voidaan käyttää arvioimaan tiedostoa, jos se huomataan uudestaan. Useat virustentorjuntaohjelmat käyttävät osana toimintaansa hajautusarvoja tunnistaakseen ennestään löydettyjä haittaohjelmia.

[Microsoft PowerShell Get-FileHash, hyvää dokumentaatiota mihin hasheja käytetään] <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash?view=powershell-7.4> 16.11

### Uhkatietojen kerääminen

**Tarkoituksena kertoa mistä uhkatietoja nostetaan ensimmäisen kerran.**

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

Lokit ovat sovellusten ja käyttöjärjestelmien tuottamia tietoja kyseisen tuotteen toiminnasta, jotka tallennetaan tai tulostetaan laitteelle. Niiden muoto ja sisältö vaihtelevat tuotteista toisiin. Lokien tarkoituksena on kertoa tuotteen toiminnasta ja ne voivat sisältää merkintöjä esimerkiksi sovelluslogiikasta, muutoksista, virheistä ja mitkä toiminnot ovat johtaneet kyseisiin tapahtumiin. Palomuurin tehtävänä on määritellä verkkoliikennettä ja suojella sisäverkkoa haitallisilta toimijoilta. Palomuurit yleensä tallentavat tiedot verkkoliikenteestä, kuten mikä lähdeosoite otti yhteyttä mihin kohdeosoitteeseen, millä portilla, kuinka monta tavua lähetettiin ja otettiin vastaan. Jos palomuuri estää yhteyden, se saattaa ilmoittaa tämän ja syyn, miksi yhteys estettiin.

[Palomuuri on] <https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-firewall.html> 28.11

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 (regex). 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, selvittääkseen niiden toimintatavat ja poimia uhkatiedot, joita ne saattavat käyttää toiminnassaan. Haittaohjelmien tutkiminen voi kuulua osaksi tietoturvaloukkaukseen liittyvää teknistä tutkimusta tai täysin itsenäinen toimenpide. Tietoturva tiimit voivat tutkia haittaohjelmia, kartoittaakseen tietyn haitallisen toimijan toimintatapoja tai palveluita. Näitä tietoja voidaan käyttää ennaltaehkäisevästi.

Aktiivisen valvonnan ja tehokkaiden turvallisuusmenetelmien kautta voidaan tunnistaa väärinkäyttöyrityksiä. Brute-force hyökkäyksessä hyökkääjä pyrkii pakottamaan itsensä sisään, esimerkiksi kokeilemalla jokaista salasanaa, kunnes jokin onnistuu. Tunnistamalla normaalista toiminnasta poikkeavaa verkkoliikennettä, voidaan tietoturvaloukkaukset estää ennenaikaisesti. Brute-force hyökkäykset 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 (honeypot). 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.

[Threat intelligence] <https://www.ibm.com/topics/threat-intelligence> 21.11

[What is a honeypot] <https://www.fortinet.com/resources/cyberglossary/what-is-honeypot> 19.11

### Uhkatietojen käyttö

Ympäristöissä on normaalia toimintaa, johon liittyy paljon verkkoliikennettä ja tiedostojen toimintaa. Vertaamalla tunnettuja uhkatietoja ympäristön toiminnassa esiintyviin tunnisteisiin, voidaan aktiiviset tietoturvaloukkaukset ja hyökkäys yritykset, 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, tietoturva-tiimi 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, antaen 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 laite tapahtumalokia. Useat SIEM-alustat jäsentelevät lokit automaattisesti sellaiseen muotoon, joka mahdollistaa tehokkaan uhkatietojen keräämisen sekä niiden analysoinnin.

## Julkisiin lähteisiin perustuva tiedustelutieto

Koska uhkatietojen kerääminen on reaktiivista, IP-osoitteet ja verkkotunnisteet usein lyhytkestoisia ja uhka tekijöitä on useita, yksittäisen tahon on hankala pitää ajantasaista uhkatieto tietokantaa yksin pystyssä.

Kyberturvallisuus yhteisö on hyvin avoin jakamaan tietoja aktiivisista uhista. Uhkatietoja julkaistaan yksityisten tutkijoiden ja ammattilaisten, sekä organisaatioiden ja yritysten toimesta niin vapaasti, kuin 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 API ratkaisun alustojen ominaisuuksien, kuten uhkatietojen hakemisen käyttöön.

### 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 tietoturva-tuotteeseen. VirusTotal kokoaa eri tietoturva-toimijoiden virustorjuntaohjelma moottoreita, verkkosivu skannereita 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 virustentorjuntaohjelma toimittajat, kuten McAfee, WithSecure, Microsoft ja CrowdStrike.

VirusTotal sisältää myös yhteisö ominaisuuksia. Rekisteröityneet käyttäjät pystyvät äänestämään (-1 tai +1) tunnistetietoja sekä jättämään vapaan kommentin tunnisteille. Yhteisön määrittämä yhteispistemäärä sekä kommentit näytetään tunnisteen yhteydessä. Käyttäjät voivat luoda omia uhkatieto kokoelmia, ominaisuus mahdollistaa mind-map tyylisen kanvaasin, jossa voi luoda yhteyksiä tunnisteiden ja uhkatietojen välille. Näitä kaavioita voi jakaa muulle yhteisölle. Alusta mahdollistaa kaikkien tietojen hakemisen API:n avulla. Ilmainen API on rajoitettu 500 kyselyyn päivässä, 4 kyselyä minuutissa. Hinnoitellussa API:ssa ei ole kyselyrajoituksia, eikä sen hinnasta ole julkista tietoa. A screenshot of a computer

Description automatically generated

Kuva X

AbuseIPDB tarjoaa alustan, jossa rekisteröitynyt ja varmistettu käyttäjä pystyy ilmoittamaan IP-osoitteiden suorittaneen haitallista toimintaa. IPv4- tai IPv6-osoitteen lisäksi, täytyy väärinkäyttö kategorisoida ainakin yhteen valmiista kategorioista ja vapaa muotoinen 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:n, jonka kautta voi tehdä ilmoituksia, sekä hakea IP-osoitteiden tietoja ilmoituksista. Ilmainen API mahdollistaa 1000 kyselyä päivässä. Varmistetut käyttäjät rajoitetaan 5000 kyselyyn päivässä. Maksullisilla sopimuksilla kyselyiden rajoituksia voi kasvattaa 50000 kyselyyn asti.

[AbuseIPDB FAQ] <http://www.abuseipdb.com/faq.html> 17.11

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 julkaisemia 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:n jonka avulla alustan toimintoja voi käyttää.

Tranco on listaus, joka mittaa verkkotunnisteiden suosiota. Se yhdistää viiden eri toimijan kokoamat verkkotunniste suosio sijoitukset, luoden listauksen miljoonasta suosituimmasta verkkotunnisteesta viimeiseltä 30 päivältä. Yhdistämällä eri toimijoiden sijoituksia, tavoitteena on luoda vakaan listaus, jota 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:n, jonka avulla voi saada tiedot verkkotunnisteen sijoituksista viimeisen 30 päivän ajalta.

Stop Forum Spam alusta kokoaa raportoituja roskaposti tapauksia, jotka ovat kohdistuneet foorumeihin sekä blogeihin. Alustan tavoitteena on joukkoistaa roskaposti raportointi ja koota ne yhteen paikkaan, jotta roskapostien lähteitä voidaan estää ennaltaehkäisevästi. Stop Forum Spam listaa yli 1000 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:n käyttöön sekä osoitteiden raportoiminen vaatii rekisteröitymistä. API 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:n 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 haittaohjelma havaintoihin. 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:n avulla ja automatisoida tunnisteiden rikastamista.

### Kaupalliset toimijat

Kaupalliset toimijat keräävät uhkatietoja kaupalliseen käyttöön. Kaupalliset toimijat suorittavat uhkatietojen keräämisen monilla tavoilla. Monet kaupalliset toimijat suorittavat omia tutkimuksia sekä tiedonkeräys toimenpiteitä, jotka kohdistuvat eri kohteisiin. Tarkkoja menetelmiä ei usein kerrota avoimesti, joten nämä kuvailut eivät sisällä kuin 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 internettiä, löytääkseen hyväksikäytön mahdollisuuksia tai kartoittaakseen tavoitteena olevan kohteen ympäristöä. GreyNoise pyrkii erottelemaan kaikesta verkkoliikenteestä ne, joiden toiminta on mahdollisesti haitallista. Alustaa käyttämällä, voidaan havaittuun verkkoliikenteeseen saada kontekstia sen alkuperästä ja tarkoituksista. Jos IP-osoitteen on havaittu skannaavan koko internettiä, 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:n, jolla voi tehdä 50 kyselyä viikossa.

[GreyNoise mitkä skannaavat] <https://docs.greynoise.io/docs/understanding-greynoise-data-sets>

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 kyselyä päivässä.

### 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.

## Uhkatietojen rikastaminen

Uhkatietojen rikastamisessa tavoitteena on tuoda lisäkontekstia digitaalisen tunnisteen toiminnasta tutkintaan.

Järjestelmien tunnistettua epänormaali toiminta, tietoturva-tiimi 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. VirusTotal antaa kuitenkin **hyvin binäärin vastauksen**. 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 mahdollisesta haitallisesta toiminnasta.

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 alustojen API ominaisuuksia, ja tuottaa yhdenmukaista dataa niiden antamista tiedoista.

# 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 pohjainen sovellus. Tämän perusteella työ toteutettiin Pythonin Flask frameworkillä ja käyttöönotossa sovellus muutetaan kontiksi Docker työkalulla.

## Python

Työn ohjelmointi kielenä käytetään python 3.11. Python tarjoaa laajan valikoiman kehittyneitä kirjastoja, joiden käyttäminen nopeuttaa kehittämistä. Työ perustuu erilaisten API alustojen käyttöön ja niiden tuomista yhteen, uusien API 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 (Python virtual environment). Virtuaalinen ympäristö sisältää vain määritetyt kirjastot ja tietyn Python version, jonka avulla kehitys on yhtenäistä eri alustoilla ja käyttöönotto prosessi helppoa, kun kehitys tehdään aina identtisessä kehitysympäristössä.

### Flask

Flask on web-applikaatio framework kirjasto, joka mahdollistaa yksinkertaisten http-pohjaisten applikaatioiden rakentamisen. Flaskia käytetään työssä määrittämään API-rajapinnat, joita kutsumalla sovelluksen toiminnallisuuksia käynnistetään ja määritetään.

Flaskin ominaisuuksista työssä käytetään Blueprinttejä yhtenäisten toiminnallisuuksien kuten virheilmoitusten määrittämiseen, route dekoraattoreita rekisteröimään URL osoitteet, jotka käynnistävät sovelluksen eri toimintoja, sekä valmiita tietokanta liittimiä ja apu funktioita. Tietokanta liittimiä käyttämällä Redis ja SQLite yhteyksien rakentamisen hoitaa Flask automaattisesti.

Tekemällä kutsuja määritettyihin URL osoitteisiin, sovellus käynnistää tausta prosesseja ja palauttaa JSON formatoituja vastauksia.

@main.route("/health", methods=["GET"])

def health\_check():

    return jsonify({"status": "running"}), 200

Esimerkki /health URL osoitteesta, joka palauttaa JSON vastauksen palvelun statuksesta.

<https://flask.palletsprojects.com/en/stable/>

### Gunicorn

Gunicorn on Web Server Gateway Interface (WSGI) http-palvelin. Sen tehtävänä on välittää http-pyynnöt web-applikaatiolle. Vaikka Flask tarjoaa http-palvelimen kehitys tarpeisiin, 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 palvelimena, tai yhdistää käänteiseen välityspalvelimeen (reverse proxy). Kehittyneemmissä ympäristöissä Gunicornin toiminnan voi helposti yhdistää välityspalvelimeen, kuten Nginx tai Apache.

### Celery

Celery on asynkroninen, hajautettu tehtäväjono. Tehtäväjonoon pusketaan tehtäviä, joita Celery työntekijät (workers) suorittavat. Jonoksi ja tehtävien välittämiseksi tarvitaan message broker, 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 tehtävä jonoon, 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.

A diagram of a task

Description automatically generated

Celery tehtävä prosessi.

## Redis

Redis on keskusmuistipohjainen varastointi palvelin alusta. Redis on erityisen tehokas varastoimaan ja lukemaan tallennettua dataa, koska se säilyttää kaiken keskusmuistissa (RAM). Se soveltuu toimimaan tietokantana, message brokerina sekä välimuistina. Tässä työssä Redis toimii samaan aikaan message brokerina 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 tehtävä jonona. Celery kirjasto sisältää valmiin toiminnallisuuden Rediksen käyttöön. Celery määritettyjä tehtäviä voidaan ohjelmallisesti puskea Rediksen tehtävä jonoon helposti, josta työntekijät saavat ne käsittelyyn. Tehtävä jono toimii first in, first out periaatteella.

## Docker

Docker on virtualisointi alusta, 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 web-palvelimen. Jokainen palvelu pystytetään yhden konfiguraatio tiedoston avulla, docker-compose.yml. Konfiguraatioon määritetään jokainen haluttu, yksittäinen Docker palvelu ja niiden erityisominaisuudet. Konfiguraatio tiedoston avulla yksittäisen palvelun, kuten tietokannan vaihtaminen ei vaadi muuta, kuin konfiguraatio muutosta ja uudelleen käynnistämistä. 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 alusta, joka on itsenäinen kokonaisuus.

# Työn toteutus

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

* API avainten tallentaminen
* Digitaalisten tunnisteiden tunnistaminen ja kategorisointi
* Useiden ulkoisten API rajapintojen käyttäminen

Sovelluksen haluttiin myös toimia niin, että uusien uhkatietolähteiden lisääminen olisi mahdollisimman yksinkertaista. Toteutus pyrittiin pitämään yksinkertaisena ja monipuolisena, että 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 ja Flask API rajapinnan käyttö ei hidastu.

Sovellusta kehittäessä, ominaisuuksia tarkennettiin ja lisättiin sitä mukaan, kun tarve niille tuli. Kehitykseen käytettiin Visual Studio Code koodi editoria 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, Windows 10, Windows 11 ja Ubuntu, riippuen siitä mikä oli sillä hetkellä kätevintä. Docker mahdollisti yhtenäisen, alusta riippumattoman 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 regex lausekkeilla syötetyt digitaaliset tunnistetiedot kategorisoidaan, ja luodaan uusi tehtävä Celeryn suoritettavaksi. Flask palauttaa tehtävän ID:n, jota kyselemällä saadaan tehtävän tila ja lopulta sen tulokset. Celery työntekijä suorittaa validoidun tehtävän ja hoitaa tunnistetiedon rikastamisen useilla API kutsuilla. Celery sekä Flask käyttävät Redistä välimuistina. Celery tallentaa välimuistiin rikastetut tunnistetiedot ja Flask tietokantaan määritetyt API avaimet, joka vähentää raskaampien toimintojen käyttöä. Flask API rajapintaa käyttämällä pystytään myös määrittämään uhkatietolähteiden API yhteyksissä käytettäviä API avaimia. Useat lähteet edellyttävät rekisteröitymisen, jonka kautta API kutsuja pystytään tekemään.

Lähdekoodin löytää työn tekijän henkilökohtaisesta GitHub reposta: <https://github.com/Sampob/threat-lense>

## 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 applikaatio tehdas funktiolla, 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 selkeyttää se alustan toiminnallisuutta, kun tuloksia saadaan suoraan http-pyytöön.

### API kehitys

API:n kehityksessä pyrittiin noudattamaan RESTful-periaatteita. Flaskin Blueprint moduulia käytettiin yhtenäistämään API:n toiminnallisuuksia, ja varmistettiin simppeli skaalautuminen, jos uusia toiminnallisuuksia täytyy lisätä.

### API reitit

API rajapinnan reitit määritetään routes.py moduulissa Blueprint dekoraattorilla, joka yhdistetään Flaskiin. Reitin rekisteröiminen ja toiminnallisuuden liittäminen siihen toimii kuin normaaleiden funktioiden määrittäminen. Esimerkiksi seuraava koodi 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...”)

Sovelluksen koko toiminnallisuus haluttiin pystyä hallitsemaan Flask API:n kautta, joten jokaiselle toiminnalle määritettiin API rajapintaan reitti, jonka kautta niitä pystytään käynnistämään. **Kuvaukset reiteistä, niiden hyväksymistä http pyyntö metodeista ja body parametreista:**

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.

GET /search {”indicator”: “<string>”}

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

GET /search/status/<task\_id>

Tehtävän suorittamista ja sen tuloksia tarkastellaan tehtävän ID:n 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

Palauttaa kaikki sovelluksen uhkatietolähteet sekä niiden konfigurointi tilanteen. Osa lähteistä tarvitsee API avaimen. Jos API avainta ei ole asetettu, ei lähteestä haeta rikastustietoja.

GET /sources/configured

Palauttaa listauksen uhkatietolähteistä, joille on määritetty API avain. Tarjoaa tavan seurata mitä lähteitä sovellukseen on määritetty.

POST /sources/<source\_id> {“api\_key”: “<string>”}

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

DELETE /sources/<source\_id>

Poistaa määritetyn API avaimen tietokannasta ja välimuistista. Uhkatietolähdettä, joka tarvitsee API avaimen ja jolla ei sellaista ole määritettynä, ei käytetä.

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

### Palautusmuoto

Sovelluksen Flask API palauttaa tuloksia aina JSON muodossa ja käyttäen standardisoituja http vastaus statuksia. Sovelluksen toiminnasta palautetaan vain oleellinen tieto, jossa mukana tietoa kyseisen toiminnallisuuden tilanteesta sekä miten sen suorittaminen onnistui. Virhetilanteista palautetaan kuvaavaa tietoa siitä, mikä aiheutti virheen.

Tunnisteiden rikastamisessa tehtävän käynnistämisestä palautetaan tehtävän ID ja URL, jota seuraamalla voidaan seurata tehtävän suorittamista. Suorittamisesta palautetaan sen hetkinen tila. Kun rikastus on valmis, palautetaan status ja formatoitu tieto rikastetusta tunnisteesta. Tämä on seuraavanlaisessa muodossa:

{

"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>}

}, {...}

]

}

http virhe statukset yhtenäistettiin Flaskin virheiden käsittelijä dekoraattorilla, joka yhdistetään blueprinttiin. Flask reitin sisällä voidaan kutsua abort funktiota, jolle annetaan parametrina http vastaus status.

@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

400 Bad Request virheen palautus.

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

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status> [18.12.2024 HTTP vastaus statukset]

## Celery workers

Työssä tarvittiin tapa, jolla käynnistää useita HTTP-kutsuja, prosessoida 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 ID:n jota käytetään tehtävän seuraamiseen. Celery suorittaa uhkatiedon rikastamisen, ja formatoidun tuloksen Redis tietokantaan, samaan ID:seen, josta se voidaan hakea. Celery työntekijöiden käytön heikkous on tulosten käsittely. Koska tulokset tallennetaan välittäjä palveluun, kuten Redikseen, jos niitä tarvitaan muualla koodissa, täytyy tuloksia odottaa ja hakea niitä jatkuvasti. Tämä haku voidaan suorittaa helposti ohjelmallisesti, mutta lisää kehitys rasitetta. Työssä, docker composella käynnistetään yksi tai useampi Celery worker taustalle. Toimintojen tehokkuutta edistetään myös rinnakkaisprosessoinnilla, jossa tavoitellaan prosessin käyttävän mahdollisimman vähän aikaa vastausten odottamiseen.

Työssä Celery työntekijät määritetään celery\_worker.py moduulissa, Celery tehdas funktiolla. Koska Flask alustaa ja huolehtii tietokantayhteyksistä, täytyy Celery työntekijälle antaa oikeus käyttää Flaskin toimintoja. Celery luodaan niin, että sillä on mukana Flask applikaatio, jolloin saavutetaan haluttu toiminnallisuus.

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

Koodi X, Celeryn luonti Flask applikaation yhteyksillä

Kehitysvaiheessa, pyrittiin ensin poistamaan Celeryn tarve käyttää tietokantaa kokonaan ja Flask applikaatio 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 välimalli ratkaisuun, jossa Flask lisää välimuistiin tietoja aina kun se on ajankohtaista, ja muuten Celery käy hakemassa tiedot tietokannasta.

[Flask / Celery yhteys] https://flask.palletsprojects.com/en/stable/patterns/celery/

### Celery tasks

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 lähde rekisteristä, ja kutsuu jokaisen lähteen tietojen haku funktiota. **Lisää** **4.3.** 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 ratkaisemaa 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:n 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 käyttämällä, voidaan rajoittaa rinnakkaisprosessien määrää. Jos asetettu yläraja ylitetään, odottaa prosessi, kunnes sille vapautuu tilaa suorittaa.

Tulokset formatoidaan lähteen määrityksien mukaisesti ja kerätään avainsana, arvo pareihin 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, indicator\_type)

                return response

            except Exception as e:

                logger.error(f"Error fetching data from {source.get\_name()}: {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}

Koodi X, Celery tehtävä asynkroninen toiminta

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

## Uhkatietolähteet

Työn perusta on uhkatietolähteet, joiden API ratkaisuista haetaan rikastus dataa. Tavoitteena oli tehdä http-haku ratkaisu, joka olisi modulaarinen, helposti muokattava ja uhkatietolähteet olisivat listattavia.

Ensimmäinen lähestymistapa oli toteutus, jossa käyttäjä määrittää itse jokaisen ulkoisen API yhteyden. Koska jokainen ulkoinen API ratkaisu toimii hieman eri tavalla käyttäjä olisi määrittänyt API reitin, mitä indikaattorityyppejä API tukee, header sekä body muuttuja ja niihin liittyvät arvot. 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 ratkaisut ovat erilaisia, täytyi jokaisen uhkatietolähteen API:n käsittely tehdä erillisesti. Jokaisesta uhkatietolähteestä päätettiin tehdä oma Python moduuli. Moduulit sisältävät uhkatietolähde luokan, joka toteuttavat yhtä abstraktia kanta luokkaa ”BaseSource”. BaseSource sisältää yhtenäisiä toimintoja, joilla eri lähteitä yhtenäistetään. **Toiminnasta 4.3.1**

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

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 populoi hakurakenteen luoduilla luokilla.

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.{module\_name}")

for name, obj in inspect.getmembers(module):

if inspect.isclass(obj) and issubclass(obj, BaseSource) and obj is not BaseSource:

instance = obj()

cls.\_instances[obj.\_\_name\_\_] = instance

Koodi X, lähde rekisteri

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

### BaseSource

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

class IndicatorType(Enum):

    IPv4 = "IPV4"

    IPv6 = "IPV6"

    DOMAIN = "DOMAIN"

    URL = "URL"

    HASH = "HASH"

    UNKNOWN = "Unknown"

Koodi X, tunniste luettelo

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.

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

Koodi X, tunnistetta vastaavien funktioden kutsuminen

Jokainen eri uhkatietolähde moduuli 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-avainten hakemiselle.

### 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 tunnisteen alhaisin sijoitus on enemmäin kuin 750,000 tai sillä ei ole sijoitusta, merkataan se epäilyttäväksi.

Tuomioissa käytettiin tuomio luetteloa, 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.

## Tiedon varastointi

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

Useat uhkatietolähteet vaativat API avaimen käyttöä. API avain sisällytetään http-pyyntöön todentamaan pyynnön lähettäjä. API 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 avain. Avain, arvoparien käyttö soveltui tähän hyvin. Tekstitiedosto, joka oli JSON-muodossa, soveltui ylläpitämään API avaimia. Tiedostosta avaimen hakeminen uhkatietolähteen kohdalta onnistui, sekä avainten lisääminen ja muokkaaminen. Kehityksen edetessä, tiedoston ylläpitäminen osoittautui haasteelliseksi. Uhkatietolähde moduulien lisääminen tai niiden poistaminen, vaati sitä, että tiedostoa muokattiin vastaamaan sen hetkisiä 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 avaimista, joista jokaiseen on liitetty viiteavaimena sen uhkatietolähde. Tietokanta alustetaan Flaskin käynnistyessä, lukemalla uhkatietolähde rekisteri ja tallentamalla jokainen luokka instanssi tietokantaan, jättäen väliin ne, jotka tietokannasta jo löytyvät.

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

### 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 palveluihin, jotka voivat rajoittaa API:en käyttöä tiettyihin puitearvoihin.

### Salaus

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

[Fernet specs, 8.1] https://github.com/fernet/spec/blob/master/Spec.md

[Encryption basics, 8.1] https://www.crowdstrike.com/en-us/cybersecurity-101/data-protection/data-encryption/

## Docker

Dockerin käyttö työssä oli selkää alusta alkaen. Sitä hyödyntämällä, nopeutettiin alustan testaamista ja pystyttämistä. Docker Composella konttien pystytys ja niiden määrittäminen onnistuu konfiguraatio tiedoston avulla. Konfiguraatio tiedosto 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.

### 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:n tarkasteluun, sekä gunicorn Python kirjasto, joka tarjoaa Flask applikaation. Celery konttiin lisätään celeryworker käyttäjä, jolla Celery taustaprosessi käynnistetään. Tällä varmistetaan, ettei Celery omaa korkeampia oikeuksia, kuin mille on tarve.

### 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.

## 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.

### Kehityksen aputoiminnot

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

### API endpoint testaus

Postman työkalulla voidaan rakentaa muokattavia, helposti toistettavia http-kutsuja. Työkalu sisältää myös ominaisuuksia, joilla voi automatisoida http-kutsujen lähetyksen. Työkalulla testattiin jokainen API 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:n /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.

# Tulokset

Työn tuloksena on 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-mallin kautta, alustan yhdistäminen olemassa oleviin järjestelmiin tai menettelytapoihin rikastamaan niissä käsiteltäviä digitaalisia tunnistetietoja.

## Toiminnallisuus

## Käyttöönotto

### Yksinkertaisessa ympäristössä

### Kehittyneemmässä ympäristössä

## Käyttö

## Jatkokehitys

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

Todentaminen API kutsuille. Flask applikaatio ei tue minkäänlaista todentamista ja 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.

API-reittien dokumentointi yleisesti käytetyllä tavalla, kuten Swagger työkalulla auttaisi alustan käytössä. Dokumentointi löytyisi tällöin itse alustalta.

API-malli mahdollistaa alustan käyttäjän rakentaa alustan ympärille tarpeilleen sopivan käyttöliittymän. Kehityskohteena olisi rakentaa alustalle sen ominaisuuksia käyttävä web-applikaatio, 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.

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

# Yhteenveto