POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Email & Fraud Protection con tecnologie open source

Relatori
prof. Cataldo Basile
prof. Andrea Atzeni

Candidato
Simone Fasolis

Responsabile SOC
CTO SicuraNext
Gabriele Peiretti

Anno accademico 2022-2023

Sommario

Le email sono uno dei principali mezzi di comunicazione odierni.

Nonostante l'uso massivo della messaggistica istantanea e dei numerosi social media disponibili, le email continuano ad essere il canale più usato per ogni tipologia di conversazione. Grazie ai costi ridotti e alla velocità di comunicazione, queste continuano a trovare favore sia nelle realtà aziendali che personali.

Dato l'elevato numero di email inviate ogni giorno, è lecito immaginare che una email possa essere usata come vettore per azioni più o meno dannose. Ci sono multiple classificazioni di email malevole, in base allo scopo e a che tipo di attacco o danni possa creare, come scam, phishing o spam, e, di conseguenza, nasce la necessità di proteggersi da tali email utilizzando tecniche applicabili sia a lato utente che a lato mail server.

Esistono numerose soluzioni che operano in questo contesto, sia proprietarie che open-source, e cercano di prevenire, o quantomeno limitare, l'azione di attori malevoli attraverso i sistemi di posta elettronica. Queste azioni sono una delle cause principali di attacchi informatici in quanto la combinazione tra ingegneria sociale e phishing è ottima per accedere a dati privati e compromettere la confidenzialità e l'integrità di qualunque utenza.

La tesi è stata sviluppata all'interno della SicuraNext S.r.l., azienda specializzata in defensive cybersecurity. Questa ha deciso di sviluppare un progetto interno per l'analisi e classificazione di artefatti. Il risultato deve lavorare in modo complementare con le soluzioni già offerte dall'azienda, analizzando, per esempio, file o email sospette ricevute sui dispositivi dei clienti. Lo sviluppo di un sistema interamente analizzato e realizzato dall'azienda ha permesso una maggiore flessibilità sulle scelte tecnologiche impiegabili per la realizzazione, sull'inserimento del sistema nell'architettura attuale e sulla logica di utilizzo dell'applicativo.

Come punto di partenza sono state scelte le email, sia a causa della mole di letteratura disponibile sull'argomento, che nella reperibilità di materiale da utilizzare come base. Lo scopo di questa tesi è la definizione di un modello di classificazione che possa riconoscere se una email ricevuta sia potenzialmente pericolosa o meno. Gli obiettivi riguardano l'analisi di una metodologia dietro a un sistema di classificazione, lo studio degli aspetti da considerare per la sua realizzazione e lo sviluppo di un'implementazione adeguata. Questo sistema è stato studiato e applicato principalmente ai sistemi Microsoft, quindi le email di tali sistemi sono state usate come punto di partenza per l'analisi del problema.

Vista la loro rilevanza in questo contesto, è stato definito un modello dedicato all'analisi e classificazione degli url. I due modelli creati, per l'email e per l'url, sono stati verificati con alcune implementazioni, sia custom che tramite librerie apposite. Inoltre, queste implementazioni sono state comparate con alcune soluzioni proposte dalla letteratura disponibile. Infine, è stato definito un prototipo per la classificazione di email in una casella postale. Questo prototipo si interfaccia

con un account Microsoft, tramite Azure e Active Directory, e classifica le email ricevute, inviando dei report allo stesso utente nel caso in cui la classificazione sia risultata negativa.

La metodologia seguita è la seguente:

1. interfacciamento con i servizi utilizzati nel corso del progetto, in particolare Microsoft Azure e Active Directory, per ottenere i permessi necessari a prelevare le email da un generico account. Questa fase include la registrazione dell'applicazione sviluppata in Active Directory, la definizione dei permessi assegnati all'applicazione, la definizione delle chiavi API utilizzate sia per l'autenticazione dell'applicativo, che per l'autorizzazione delle richieste HTTP verso Microsoft, la definizione delle chiavi API necessarie per utilizzare le funzionalità di alcuni servizi diffusi nel campo della Cyber Threat Intelligence per ottenere informazioni aggiuntive riguardo alcuni elementi, come dominio o url.

2. analisi di una email prelevata da Outlook, analizzando quali campi sono potenzialmente rilevanti per classificare una email ricevuta come legittima o sospetta. Questi campi si trovano sia nell'header, sfruttando, per esempio, gli indirizzi, l'oggetto o protocolli di autenticazione, che nel body, sfruttando un'analisi testuale tramite keyword. In questa fase vengono impiegati alcuni servizi esterni, per esempio Urlscan.io nel caso di un URL, per ottenere informazioni non reperibili tramite la mail, o URLhaus, per verificare se un generico URL si trova in una blacklist.

3. definizione di un database in cui salvare ogni tipo di informazione considerata utile. L'email è divisa in categorie, come body, header o allegati, e salvata in una decina di schemi. In questo progetto, tutte le informazioni necessarie per ricostruire l'email sono salvate in un database relazionale PostgreSQL.

4. ricerca di alcuni dataset di partenza da usare per verificare il modello sviluppato e, nel caso, addestrarne di piu efficienti. I dataset utilizzati sono pubblici e presi da diverse sorgenti online. Lo scopo in questa fase è quella di raccogliere una discreata varietà di email per costruire un dataset il più vario possibile. L'idea alla base è quella di ricreare unn caso reale.

5. estrazione delle feature potenzialmente utili per una classificazione utilizzando uno script in Python. Questo programma simula la fase di estrazione delle informazioni rilevanti per una classificazione. Sono stati usati alcuni modelli più complessi che richiedono una fase di addestramento per poter essere utilizzati, quindi è stata introdotta una fase di salvataggio nel database, affinchè venga costruito un dataset da utilizzare sia per la fase di addestramento che per la fase di validazione del modello.

6. definizione un modello che possa classificare una email utilizzando le feature estratte. Sono stati definiti molteplici modelli che possano classificare una email, con diversi risultati. Molti di questi modelli necessitano di una fase di addestramento, utilizzando le feature estratte nelle fasi precedenti. Al termine della definizione, sono stati scelti tre modelli, uno basato sull'uso di regole e due utilizzando un approccio più modellistico.

7. definizione di un tool che lavori in background su un dispositivo e applichi il modello implementato ad un account Microsoft. Questo è uno script Python che sfrutta uno dei modelli implementati per classificare le email ricevute dal generico utente. Il tool agisce in background sul dispositivo utente e si attiva solo quando l'account Microsoft configurato riceve una email. Il tool possiede tutte le funzionalità complementari affichè lavori correttamente: estrazione delle feature, salvataggio dei dati, aggiornamento delle informazioni utente, selezione del modello e invio di report.

Indice

Capitolo 1

Introduzione

1.1 Le Email

Le email sono uno dei principali mezzi di comunicazione odierni.

Nonostante l'uso massivo della messaggistica istantanea e dei numerosi social media disponibili, le email continuano ad essere il canale più usato per comunicazioni di ogni tipo. Grazie ai costi ridotti e alla velocità di comunicazione, queste trovano favore sia nelle realtà aziendali che personali.

Secondo 99firms [1], esistono oltre 4 miliardi di utenti che fanno uso di email, con una crescita di circa 100 milioni annua, abbastanza distribuiti in età, al contrario di quello che si possa comunemente pensare. Il numero di account attivi è circa pari a 7 miliardi, con un numero di email giornaliere che si aggira attorno ai 330 miliardi, circa 3 milioni al secondo.

Secondo Statista [2], il numero di utenti crescerebbe fino a 4.6 miliardi entro la fine del 2025, con un numero di email giornaliere inviate pari a 376 miliardi.

1.2 Uso improprio delle email

1.2.1 Descrizione

Dato l'elevato numero di email inviate ogni giorno, è lecito immaginare che una email possa essere usata come vettore per azioni più o meno dannose.

Ci sono innumerevoli classificazioni di email malevole, in base allo scopo e a che tipo di attacco o danni possa creare, ma le più diffuse sono solo due: spam e phishing.

Lo spam, o unsolicited bulk email, è un'email inappropriata, non richiesta, disinformativa o irrilevante per l'utente, come recensioni di prodotti, pubblicità, segnalazioni di antivirus, risultati di ricerche, immagini o richieste di denaro per qualche causa. Le email di spam possono anche contenere url, potenzialmente per download di malware, backdoors o avere allegati con, potenzialmente, dei file eseguibili malevoli. In questo casi ci si muove verso la direzione del phishing.

Esistono entità che, a pagamento, si dedicano alla creazione di tali email, affinchè lo spammer abbia la massima distribuzione possibile, non curandosi della qualità del contenuto.

Il phishing ha un danno più diretto e tangibile. Le email di phishing cercano di estrarre informazioni sensibili, come le password, sfruttando delle vulnerabilità software o delle forme di ingegneria sociale.

Le modalità di estrazione delle informazioni hanno molte sfumature:

- impersonazione di un utente conosciuto, creando un indirizzo falso e una struttura lessicale simile all'utente impersonato. L'utente ignaro potrebbe non accorgersi dell'impersonazione e rivelare informazioni sensibili. Esempi di questo tipo sono gli spear phishing, ovvero tentativi di phishing mirati a uno specifico utente

- forme di ingegneria sociale, simulando un contesto vicino alla vittima per convincerlo a rivelare informazioni sensibili. Durante il vivo della pandemia circolavano molte email di promesse per una vaccinazione anticipata solo compilando alcuni form. Esempi di questo tipo sono gli scam

- uso di url malevoli, l'email potrebbe contenente uno o più url per redirigere l'utente su siti web fasulli in cui inserire informazioni sensibili. Un target comune è la pagina di accesso di un servizio bancario, ingannando la vittima ad inserire le proprie credenziali

- uso di malware, di ogni tipologia. Questo può essere un backdoor per accedere al dispositivo della vittima senza che essa se ne accorga, un keylogger per monitorare ogni evento o azione da tastiera (per esempio l'inserimento di credenziali), o un ransomware per bloccare totalmente il dispositivo colpito fino al pagamento di un riscatto.

La pandemia in corso, per esempio, è stata una causa indiretta di molti tentativi di phishing. Esempi comuni di phishing erano email risarcimento o form da compilare con i propri dati per ottenere prima la vaccinazione. Secondo Sophos [3] il settore medico/sanitario, finanziario e governativo ha registrato un aumento del 70% di attacchi phishing, di tutte le forme.

Secondo 99 Firms [4], nel 2020 il 47,3% di tutte le email erano email di spam, in ambito pubblicitario, catene, richieste di denaro, avvisi su malware circolanti e contenuti per adulti.

Secondo Statista [5], durante il Luglio del 2021, in una giornata sono state inviate 283 miliardi di email di spam su un totale di 336 miliardi di email inviate (84.22% del totale), su contenuti pubblicitari, per adulti e consigli finanziari. Il 2.5% delle email di spam appartengono alla categoria del phishing e questa rappresenta il 96% di tutto il phishing effettuato nel 2021.

Secondo Securelist by Kaspersky [7], nel 2021 il 45.56% di tutte le email inviate erano spam. Solo i loro servizi hanno bloccato quasi 150 milioni di allegati malevoli, soprattutto trojan e script per lo sfruttamento di vulnerabilità, e 250 milioni di link a malware di ogni tipo. Una nota importante è il fatto che il Top Level Domain più utilizzato per gli url di phishing è il *.com*, con il 31.55%, ovvero il TDL più utilizzato su Internet.

Secondo IT Governance [8], i settori più colpiti da attacchi di phishing sono il settore finanziario, senza sorprese, i webmail providers e i siti di shopping online, anche questo senza sorprese.

1.2.2 Motivazioni

Le ragioni per l'abuso di email sono molteplici:

- lo spam può essere usato per promuovere servizi o prodotti di ogni genere, inoltre potrebbe migliorare la sua posizione nei comuni motori di ricerca, aumentando il profitto

- le entità per la distribuzione di pubblicità già menzionate ricavano profitto dalla generazione di spam. Gli spammers usano servizi simili per ampliare il proprio bacino di utenza

- il phishing è usato per rubare informazioni personali e/o sensibili in diverse modalità tutte al fine di rubare o estorcere denaro. Il danno subito dalla vittima non si limita alla perdita economica, ma si traduce in perdita di produttività e danno alla propria reputazione. In questo caso il danno non è calcolabile

1.2.3 Soluzioni e mitigazioni

Come conseguenza nasce la necessità di proteggersi da tali email. Esistono soluzioni applicabili sia a lato utente che a lato mail server. Le tecniche lato utente sono implementate una volta che la email è arrivata nella casella postale, mentre le tecniche lato mail server sono implementate prima che la mail arrivi nella casella postale dell'utente.

Alcuni esempi sono:

- uso di whitelist, ovvero selezionare solo i contatti o domini da cui si vuole ricevere email
- uso di blacklist, opposto al whitelist, selezionando tutti quei contatti o domini da cui non si vuole ricevere email. Questa è, in linea teorica, la soluzione migliore, ma tenere traccia di tutti gli elementi da inserire in blacklist (considerando la facilità nell'ottenere un nuovo indirizzo email) è impossibile. Esempi sono i DNS blacklist o gli URI blacklist.
- greylisting, ovvero ogni email ricevuta da un utente nuovo è temporaneamente rifiutata. Questo utente dovrà inviare la email una seconda volta affinché arrivi al destinatario. Si punta sul fatto che gli attori malevoli, in particolare gli spammers, preferiscano inviare quante più email possibili e non sprecare tempo con un mail server che rifiuti le email, per qualunque motivazione
- uso di protocolli di autenticazione, come il DomainKeys Identified Mail, che permetterebbe passaggio alle email provenienti da sorgenti autenticate e bloccherebbe le email non autenticate, il Sender Policy Framework, che controllerebbe la relazione tra il domino della email l'IP di origine, o un sistema challenge/response durante il passaggio delle email da un server all'altro
- uso di server appositamente configurati come esche, come un honeypot o un tarpit. Un honeypot è un server mascherato da open proxy o da mail transfer agent e utilizzabile per inviare email malevole. L'honeypot, ovviamente, non invia email, facendo sprecare risorse all'utente, potenzialmente identificando la sorgente di origine e ottenendo un'email potenzialmente pericolosa da analizzare. Un tarpit è un server volutamente lento che gli attaccanti, in particolare gli spammers, possono usare come open relay. La lentezza limita la velocità di invio di email
- filtri di contenuto, ovvero un filtri che riconoscono determinati elementi della email e, in caso, la bloccano. Questi possono essere statici, filtrando con keyword, sulla nazione di origine, tramite la lingua usata o l'IP di provenienza, o dinamici, filtrando usando algoritmi più intelligenti in grado di selezionare automaticamente quali aspetti considerare
- attenzione e cautela, che è perfettamente considerabile come soluzione. Per esempio occorre prestare attenzione a quali link si usano, con chi si condivide la email e a controllare l'indirizzo del mittente

Bisogna notare come molte delle soluzioni proposte siano delle mitigazioni al problema, limitando la quantità di email dannose, ma non delle soluzioni. In questa categoria appartengono tecniche quali la disabilitazione dell'HTML a livello del browser, il greylisting, gli honeypots e i tarpits. Il loro scopo non è eliminare le email malevole ma ridurre il numero inviato, complicando l'azione dell'attaccante. Queste tecniche trovano molta utilità contro l'azione, per esempio, di uno spammer il cui scopo è inviare il maggior numero possibile di email alle vittime.

Un altro punto da discutere è il fatto che molte tecniche siano statiche e richiedano un aggiornamento costante, nell'ordine di multiple volte al giorno. In questa categoria appartengono tecniche quali i vari tipi di blacklist e diverse tipologie di filtri. Le tecniche appena menzionate sono utili nel bloccare ogni tipo di attaccante ma richiedono aggiornamento sia nella registrazione degli utenti malevoli, molto complicato a causa della facilità di creare nuovi indirizzi email, che nel metodo, regola o implementazione utilizzato per classificarli. Si parla anche di data drift, ovvero la modifica nella struttura e modalità di azione dell'email nel corso del tempo e che, potenzialmente, può rendere obsolete le implementazioni utilizzate dai software attuali.

Infine, resta il problema della creazione e distribuzione delle numerose chiavi utilizzate per i vari protocolli di autenticazione e della configurazione dei dispositivi affinché usino e gestiscano correttamente sia i protocolli che le chiavi associate.

Per questo motivo, c'è molto interesse verso la ricerca e sviluppo di filtri dinamici. Questi utilizzano algoritmi appartenenti alla branca del machine learning, ovvero modelli in grado di addestrarsi su insiemi di email al fine di estrarre pattern o informazioni rilevanti per una classificazione. I filtri realizzati con queste tecniche possono classificare le email e, allo stesso tempo, addestrarsi con le nuove email ricevute, restando perennemente aggiornati. Di contro, questi algoritmi, oltre ad essere molto complessi, richiedono tempo e dati estremamente precisi con cui addestrarsi. In caso contrario le loro prestazioni si riducono considerevolmente.

Esistono numerosi software per il riconoscimento e la classificazione di email malevole. Alcuni esempi di tool open-source possono essere:

- *Apache SpamAssassin*, tool per la pura classificazione di email di spam. Il tool esamina le email ricevute e assegna un indicatore alla email definendola come spam o legittima, sarà compito di un tool esterno decidere cosa fare: eliminarla, indirizzarla verso un cartella predefinita o lasciarla invariata.
  Per classificare le email il tool usa diverse tecniche, quali un filtro bayesiano, per identificare dei token tipicamente trovati nelle email di spam, euristiche, come uso di DNS e URI blacklist e whitelist, e match di regole pre-definite applicate all'header, al body e agli url, come controlli numerici sul from, presenza di termini tipici nella mail di spam o uso di una porta anomala.
  L'uso di diverse tecniche è stato scelto per avere diversi metodi su cui appoggiarsi, per non costringere l'utente a impiegare tempo per addestrare il modello bayesiano all'avvio del software, e, una volta che il filtro statistico è stato addestrato a sufficienza, per compensare i tentativi di attori malevoli nell'aggirare le regole definite

- *MailScanner*, email gateway per il filtraggio di spam, virus, phishing e malware. Il tool si appoggia su diversi software per la classificazione quali SpamAssassin per il riconoscimento di email di spam, ClamAV e BitDefender per il riconoscimento di malware di vario genere. Inoltre il tool si interfaccia con i database di PhishTank e Alexa per mantenere aggiornati i propri database sui domini associati ai tentativi di phishing

- *Proxmox Mail Gateway*, email gateway per il filtraggio di spam, virus, phishing e malware. Il sistema integra le funzionalità di SpamAssassin per riconoscere le email di spam e ClamAV per riconoscere i malware. Le tecniche usate per il filtraggio includono l'uso di blacklist (DNS e URI), whitelist (SMTP), protocolli di autenticazione (SPF), greylisting e filtraggio bayesiano. Inoltre permette all'utente di definire le proprie regole per il filtraggio (per esempio in base al mittente o al contenuto) e regole per l'azione post-classificazione (cosa farne della email)

Esistono anche molti tool e servizi di supporto, almeno parziale, al riconoscimento di email di phishing, analizzando url e file sospetti. Alcuni esempi possono essere:

- *Shodan*, servizio per l'analisi di indirizzi IP in grado di fornire geolocalizzazione, informazioni sulle richieste HTTP e sui certificati SSL utilizzati dal dominio utilizzato

- *VirusTotal*, servizio per la scansione di file e url in grado di fornire degli indicatori di bontà basandosi sul riscontro di numerosi vendors, come Fortinet, Google Safebrowsing, Kaspersky e URLhaus

- *Browserling*, sandbox per il test cross-browser di url sospetti. Ogni tipo di azione o evento generato dal sito visitato è circoscritto al sandbox

- *Whois*, servizio per l'analisi i indirizzi IP e domini di rete, fornisce ogni genere di informazione associata alla risorsa, come data di registrazione, geolocalizzazione, organizzazione e sottoreti

- *Phishhunt.io*, servizio per la raccolta e condivisione di siti web, compagnie, IP e certificati TLS associati in qualche modo al phishing
- *Abuse.ch*, insieme di piattaforme per la condivisione di materiale considerato pericoloso, tra malware, url, file e certificati SSL

Tutti i servizi elencati sono disponibili sia tramite applicazione web che tramite API.

Molti moduli coprono un sottoinsieme degli oggetti disponibili, tra url, file, IP e domini, e dei metodi e campi considerati per l'analisi. Per questo non è raro che un applicazione ne faccia uso di diversi, sia per coprire eventuali lacune che un singolo modulo potrebbe lasciare, che per avere più riscontri su una stessa risorsa. Sarà compito dell'applicativo gestire i diversi riscontri.

1.3 Archietettura di rete

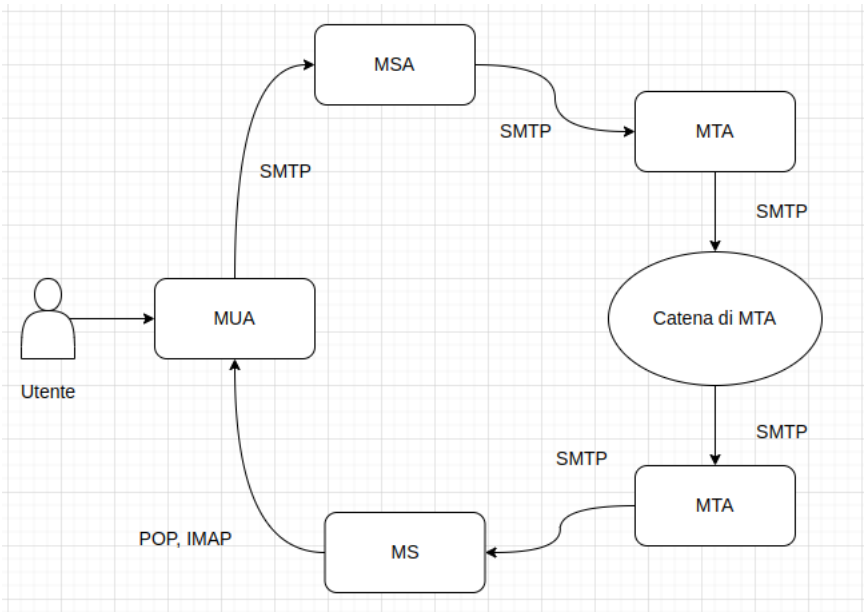
Questa sezione è dedicata ad una breve descrizione dell'architettura tipicamente usata per inviare le email e come inserire un classificatore in essa.

Il processo di trasporto di una email da un mittente alla casella postale del destinatario utilizza alcuni protocolli di rete e alcune componenti software.
I protocolli principali sono classificabili in due tipologie:

- protocolli per l'invio di email, come il SMTP (Simple Mail Transfer Protocol) e le sue varianti ESMTP (Extended SMTP), attualmente in uso, e SMTPS, applicazione del protocollo TLS a SMTP
- protocolli per il prelievo di email, come il POP (Post Office Protocol) e le sue varianti POP3 (POP versione 3) e APOP (Authenticated POP), o il IMAP (Internet Message Access Protocol)

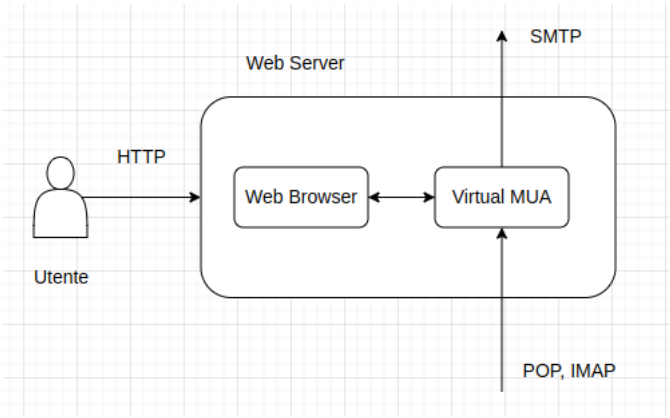
In seguito c'è una breve descrizione delle componenti utilizzate:

Figura 1.1. Architettura generale.



- **MUA** (Message User Agent), è una componente software che l'utente usa per gestire le proprie email. Questo può essere sostituito da un webmail, ovvero un MUA virtuale eseguito in un browser. In questo caso la comunicazione tra l'utente e il MUA utilizza HTTP/HTTPS

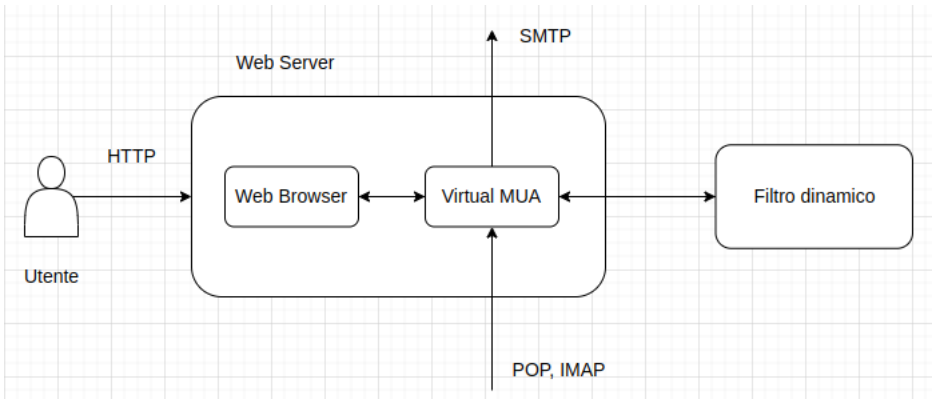
Figura 1.2. MUA virtuale.



- **MSA** (Message Submission Agent), è il mail server, ovvero una componente software responsabile per l'immissione della email dal MUA nella rete usando il SMTP o una sua variante
- **MTA** (Message Transfer Agent), componente responsabile del trasporto dell'email. Ogni email, quindi, attraversa un certo numero di MTA prima di arrivare alla destinazione finale. Ogni MTA utilizza SMTP o una sua variante per inviare l'email al prossimo MTA
- **MS** (Message Store) o Post Office, è la destinazione finale della email, passando per gli MTA. Il MUA del destinatario richiede le email dal MS usando POP, IMAP o una loro variante

Il compito di un classificatore è quello di filtrare (isolare o eliminare) tutte le email ritenute pericolose dall'algoritmo. Questi possono essere inseriti sia a livello di client nel dispositivo dell'utente come componente aggiuntiva, che a livello di mail server, bloccando le email ancora prima che possano raggiungere il destinatario.

Figura 1.3. Filtro su un client.

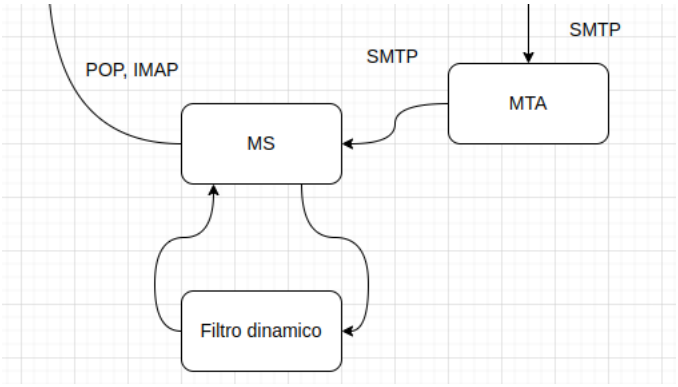


La maggior parte dei filtri sono a livello i server e inseriti di fronte ai MS, cosi che l'email possa essere indirizzata in determinate cartelle o messa in quarantena in modo trasparente all'utente.

Un esempio di architettura, vista la centralità del sistema nell'elaborato, è l'Exchange Online Protection di Microsoft [9]. Il sistema è composto da una serie di filtri applicati prima che la email possa raggiungere la casella postale dell'utente.
Ha quattro componenti principali:

- filtro a livello di rete, ovvero applicazione di IP blacklist e IP whitelist

Figura 1.4. Filtro su un server.



- filtro a livello di allegato, analizzando, per esempio, le estensioni dei file ricevuti
- filtro a livello di policy, ovvero usando delle regole definite dall'utente
- filtro a livello di contenuto, analizzando la mail ricevuta e potenzialmente classificarla come spam, phishing o spoofing. I risultati della classificazione sono inseriti nell'header della email e la mail viene inviata in quarantena, in una cartella specifica o eliminata

Un secondo esempio è estratto da *A multi-layer architecture for spam-detection system* [10], in cui viene presentata un'architettura anti-spam composta da quattro filtri:

- filtro applicativo di blacklist e whitelist
- filtro applicato al contenuto della mail
- filtro applicato alle immagini allegate
- selezione negativa, ovvero applicazione di un algoritmo appartenente ad una branca degli artificial immune systems per il riconoscimento di pattern associati alle email di spam

1.4 Scopo della tesi

La tesi è stata sviluppata all'interno della SicuraNext S.r.l., azienda specializzata in defensive cybersecurity. Questa ha deciso di sviluppare un progetto interno per l'analisi e classificazione di artefatti. Il risultato deve lavorare in modo complementare con le soluzioni già offerte dall'azienda, analizzando, per esempio, file o email sospette ricevute sui dispositivi dei clienti. Lo sviluppo di un sistema interamente analizzato e realizzato dall'azienda ha permesso una maggiore flessibilità sulle scelte tecnologiche impiegabili per la realizzazione, sull'inserimento del sistema nell'architettura attuale e sulla logica di utilizzo dell'applicativo.

Lo scopo di questa tesi è la definizione un modello di classificazione che possa riconoscere se una email ricevuta sia potenzialmente pericolosa o meno. Gli obiettivi riguardano l'analisi di una metodologia dietro a un sistema di classificazione, lo studio degli aspetti da considerare per la sua realizzazione e lo sviluppo di un'implementazione adeguata. Per questo motivo, una parte della ricerca è stata dedicata all'analisi delle diverse soluzioni proposte nel corso degli anni in ambito Email & Fraud Protection.

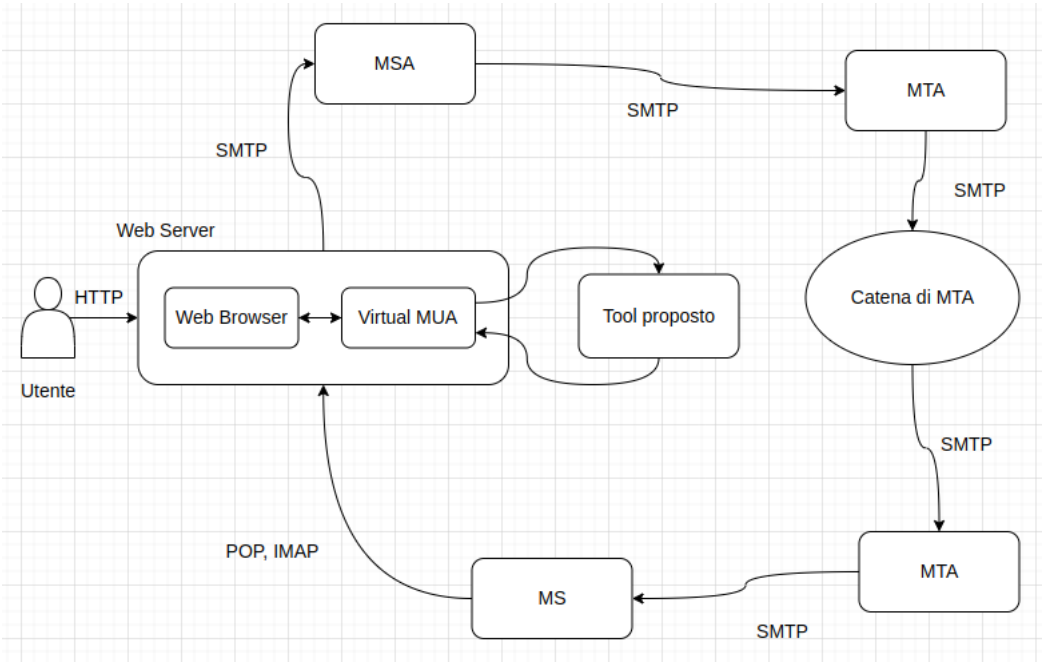
Vista la loro rilevanza in questo contesto, è stato definito un modello dedicato all'analisi e classificazione degli url. I due modelli creati, per l'email e per l'url, sono stati verificati con alcune implementazioni, sia custom che tramite librerie apposite. Inoltre, queste implementazioni sono state comparate con alcune soluzioni proposte dalla letteratura disponibile. Infine, è stato definito un prototipo per la classificazione di email in una casella postale. Questo prototipo si interfaccia

con un account Microsoft, tramite Azure e Active Directory, e classifica le email ricevute, inviando dei report allo stesso utente nel caso in cui la classificazione sia risultata negativa.

Il prototipo definito potrà essere rivisto e ottimizzato a sufficienza per l'inserimento in una realtà aziendale, possibilmente come elemento di supporto al sistema di gestione email utilizzato internamente.

Considerando la topologia di rete usata nel progetto, con un MUA virtuale, il classificatore proposto viene applicato a livello di client, ovvero in locale sul dispositivo utente.

Figura 1.5. Architettura proposta.



Capitolo 2

Ricerche sull'argomento

2.1 Introduzione

Sono stati effettuati molti studi sull'argomento dell'email and fraud protection, ovvero sull'analisi e classificazione di email e url considerati malevoli, sia per l'analisi dello spam che per l'analisi del phishing.

La metodologia più comune dell'ultimo decennio prevede l'uso di modelli dinamici, o machine learning, ovvero modelli in grado di estrarre pattern da un insieme di elementi e prevedere le caratteristiche di quelli che riceve in input.

Il maggior punto di forza di questo tipo di sviluppo è la flessibilità di applicazione. Un modello di machine learning è applicabile ad una sostanziale varietà di campi e problematiche esistenti. Se correttamente configurato, il sistema produrrà un risultato coerente con gli elementi usati per addestrarsi, traducendosi in classificazioni e predizioni di dati.

Di contro, la corretta configurazione di un modello è un processo complesso, in quanto la minima variazione parametrica è in grado di alterare l'output ricevuto e produrre, quindi, risultati errati. Un secondo vincolo sono gli elementi utilitzzati per addestrare il modello. Gli elementi di input devono avvicinarsi il più possibile alle perfezione. Elementi non necessari o, peggio, errati, compromettono la bontà del risultato tanto quanto un'errata parametrizzazione. Infine, questi modelli richiedono una notevole quantità di tempo per potersi addestrare, oltre che ad una notevole quantità di risrorse computazionali.

Nonostante la loro difficoltà, la maggior parte delle soluzioni attuali impiegano alcune forme di modellizzazione dinamica per analizzare elementi o prevedere pattern interessanti per il contesto in cui operano. Questi, spesso, lavorano in sintonia con elementi e modelli statici, sia per rafforzare l'output, che per compensare eventuali errori di classificazione. Alcune delle soluzioni anti-spam open-source più popolari, come *Apache SpamAssassin* operano con questa metodologia, cosi come la maggior parte delle soluzioni proprietarie nel panorama attuale.

Il trend attuale riguarda la coesione di questi due tipologie di modelli, almeno fino a quando la modellizzazione dinamica non riesca ad eseguire il salto di qualità per surclassare la controparte statica, evento non troppo distante.

La maggior parte degli studi segue una metodologia riassumibile in cinque fasi:

1. pre-processing, ovvero applicazione di tecniche per la conversione del messaggio in elementi significativi per future analisi. Questa parte è fondamentale per la classificazione testuale. Alcune tecniche comunemente usate sono:
    - tokenization, ovvero la conversione delle parole/frasi nel messaggio in elementi in sé privi di significato, detti token
    - stop word removal, ovvero la rimozione di tutte quelle parole irrilevanti da sole, come gli articoli e le congiunzioni

- stemming, ovvero la trasformazione delle parole in una forma base, eliminano plurale, suffissi e forme verbali

2. trasformazione dei token in una forma appropriata per essere processato da un classificatore. Elementi ricorrenti in questa fase possono essere:
   - bag of words, ovvero la rappresentazione di ogni parole/token con la sua frequenza
   - vector space model, ovvero tutte le feature esaminate con un vettore di elementi di vario tipo, come stringhe, numeri o booleani
3. feature selection, ovvero selezionare, tra tutte le feature considerate, un sottoinsieme ottimale affinché il modello risultante risulti più veloce e abbia performance maggiore. Queste tecniche possono essere interne o esterne al classificatore
4. applicazione di un algoritmo di classificazione, utilizzando uno tra le diverse tipologie di modelli disponibili, quali alberi decisionali, classificatori bayesiani, reti neurali o SVM. Questi modelli dovranno essere addestrati con un elevato numero di elementi rappresentanti l'oggetto che si vuole classificare. Gli elementi utilizzati per addestrare il modello dovranno essere il più rappresentativi possibili per tutte le tipologie di classi che si vogliono analizzare o c'è la possibilità di raggiungere un overfitting, ovvero la costruzione di un classificatore estremamente specifico solo per un piccolo sottoinsieme di elementi.
5. valutazione delle performance utilizzando diversi indicatori e metriche, ad esempio:
   - True Positive (TP), significa classificare un email legittima come legittima
   - True Negative (TN), significa classificare un email malevola come malevola
   - False Positive (FP), significa classificare un email legittima come malevola
   - False Negative (FN), significa classificare un email malevola come legittima
   - Accuracy, rappresenta la relazione tra le email correttamente classificate e la totalità delle email prese in considerazione

     $$Accuracy = (TP + TN)/(TP + TN + FP + FN)$$

   - Precision, rappresenta la relazione tra le email malevole correttamente classificate e la totalità delle email classificate come malevole

     $$Precision = TP/(TP + FP)$$

   - Recall, rappresenta la relazione tra le email malevole correttamente classificate e la totalità delle email correttamente classificate

     $$Recall = TP/(TP + FN)$$

   - Error, definito come il complementare dell'accuracy

     $$Error = 1 - Accuracy = (FP + FN)/(TP + TN + FP + FN)$$

   - F-measure, definito come media armonica di Precision e Recall

     $$F - measure = 2 * Precision * Recall/(Precision + Recall)$$

Le tecniche, metodologie e le considerazioni applicate sulla classificazione sono numerose. L'analisi effettuata considera tre tipologie di classificazione:

- classificazione testuale, ovvero tutte le feature analizzate sono state estratte esclusivamente dal corpo della email
- classificazione di url, ovvero il modello analizza solo feature relative all'url considerato
- classificazione ibrida, ovvero il modello considera, oltre al corpo, anche l'header ed eventuali url inseriti

Le sezioni successive riassumono alcuni degli articoli che hanno contribuito alla definizione della metodologia usata in questa tesi, separando gli argomenti per metodo di analisi.

2.2 Classificazione testuale

Questi degli articoli si concentrano solo sul corpo della email, affrontando il problema come pura classificazione testuale.

In *A comparative study for content-based dynamic spam classification using four machine learning algorithms* [13] viene analizzato il problema implementando la metodologia appena descritta con la rappresentazione bag of words e il Relevance Vector Machine (RVM) ottenendo performance tra il 94% e il 96% di accuratezza.

In *URL based Email Phishing Detection Application* [14] viene estesa la soluzione applicando tecniche più avanzate di feature selection, tra cui il Complete Gini-Index Text, per la categorizzazione testuale. Rispetto a altre tecniche di feature selection, come l'Information Gain o il Chi Square, il Complete Gini Index, con 80000 feature, raggiunge valori pari al 98% di accuratezza.

La pura classificazione testuale è ottima per la classificazione di email di spam.

2.3 Classificazione degli url

Il secondo argomento trattato è una classificazione basata esclusivamente sugli url.

In *Using Lexical Features for Malicious URL Detection - A Machine Learning Approach* [15] viene proposto un modello in grado di classificare gli url analizzando esclusivamente feature lessicali, ovvero dati e statistiche prelevate direttamente dall'url. Il classificatore migliore è stato il Random Forest con performance del 92%.

In *Machine learning based phishing detection from URLs* [16] vengono unite le feature lessicali alle feature estratte con un approccio Word Vectors. Le feature ibride sono testate con multipli classificatori. Si nota come l'approccio word vector non è efficiente per quanto riguarda la classificazione di url, mentre gli approcci MLP e ibridi toccano accuratezze tra il 95% e 97%.

In *Hybrid Rule-Based Model for Phishing URLs Detection* [17] vengono integrate alle feature lessicali delle feature ottenute analizzando il dominio, il traffico web e i record DNS associati all'url. Non è stato usato un modello di classificazione ma 55 regole ottenute analizzando le feature considerate. Analizzano due dataset, ottenendo accuratezze pari al 94% e al 99%.

In *Lightweight URL-based phishing detection using natural language processing transformers for mobile devices* [18] vengono integrate alle feature lessicali feature riguardanti la presenza di keyword e indici di similarità tra domini e sottodomini. Dei vari classificatori, il Random Forest si rivela il migliore con performance del 98%.

2.4 Classificazione ibrida

Questo tipo di classificazione prevede l'integrazione di feature estratte dall'header e dagli url alle feature estratte dal corpo della email. Non viene considerata una classificazione testuale completa, riducendo enormemente lo spazio dimensionale su cui il modello lavora, passando da migliaia di feature a diverse decine. Questo incrementa enormemente la velocità di classificazione.

In *Detecting Phishing Emails the Natural Language Way* [19] viene introdotta una classificazione ibrida che include, oltre all'analisi NLP, un'analisi sugli header e url. L'unione delle varie analisi, NLP, header e url,produce un classificatore con performance del 97%

In *A Multi-Classifier Based Prediction Model for Phishing Emails Detection Using Topic Modelling, Named Entity Recognition and Image Processing* [20] vengono include feature estratte dalle immagini, feature estratte da informazioni strutturali e da informazioni contestuali con l'uso di keyword. Il multi classificatore definito ha performance nell'ordine del 97%

In *Phishing Email Detection Based on Binary Search Feature Selection* [21] viene integrata nella metodologia una nuova tecnica di feature selection, il Binary Search Feature Selection,

paragonata con l'uso del Sequential Forward Feature Selection e l'uso senza feature selection. Le feature considerate sono estratte dall'oggetto, dal corpo e dagli url. I risultati mostrano come l'uso ottimale di una tecnica di feature selection porti a performance del 97%, rispetto al 95% senza feature selection. Il modello trae beneficio nella velocità di classificazione.

In *Email Fraud Attack Detection Using Hybrid Machine Learning Approach* [22] viene affrontata la classificazione considerando le informazioni estratte prevalentemente dall'header e dagli url all'interno del messaggio con il Knowledge Discovery. Definiscono un classificatore ibrido sviluppato su Adaboost con Majority Voting che, usando le feature estratte con il KD, ottiene performance attorno al 98%.

In *Phishing Email Detection Based on Hybrid Features* [23] vengono applicati dei controlli sulla presenza di Javascript all'interno della email, così come eventi pop up e OnClick, e feature psicologiche, alle feature estratte usando header e url, senza usare una classificazione testuale. L'aggiunta di queste feature, nonostante l'assenza di una classificazione testuale, porta a classificazioni con performance del 95%.

In *Classification of Phishing Email Using Random Forest Machine Learning Technique* [24] vengono inlcuse delle feature ottenute analizzando il dominio del mittente e gli hostname degli url presenti nella email. L'analisi dei domini applicata a un algoritmo di alberi decisionali (Random Forest) produce risultati attorno al 98%.

Si nota come, per le email di phishing, la classificazione testuale perda di importanza. Le informazioni rilevanti si concentrano più nell'header delle email e negli eventuali url che nel testo.

Capitolo 3

Azioni preliminari

3.1 Introduzione

In questo capitolo sono descritti una serie di azioni preliminari necessarie per l'inizio dell'analisi condotta.

Tutta l'analisi effettuata utilizza Microsoft come soggetto di analisi, motivo per cui molti dei passi preliminari riguardano proprio le piattaforme Microsoft.

La specifica piattaforma vanta un ampio bacino di utenza e una discreta documentazione, inoltre Microsoft Azure offre l'opportunità di lavorare su un sistema multiutente per la raccolta e l'uso dei dati.

La scelta di usare Microsoft, comunque, non è stato un fattore rilevante per lo sviluppo dell'elaborato. La scelta di utilizzare un'altra piattaforma, come Google, per esempio, sarebbe stata accettabile. Vista la diffusione di Microsoft come infrastruttura utilizzata dalle aziende a contatto con la SicuraNext S.r.l., si è scelto, alla fine, quest'ultima.

Tali passi includono:

- interfacciamento con Azure Active Directory per registrare l'applicazione sviluppata e ottenere i permessi di accesso all'account utente
- analisi della Microsoft Authentication Library (MSAL) [35] per il prelievo di token di autenticazione da usare per l'autenticazione delle API di Graph
- analisi di Microsoft Graph [36] per le API da usare
- definizione delle API key richieste dalle API dei vari servizi esterni usati per l'analisi di domini, hostname, url e file

Vengono anche descritti alcuni servizi utilizzati per l'analisi di domini, hostname e url nel corso dello studio:

- VirusTotal [31], servizio molto usato nel campo CTI per l'analisi di file e con un'ampia community che lo mantiene costantemente aggiornato
- AlienVault OTX [32], servizio molto usato in ambito CTI per la capacità di analizzare file, url, domini e hostname
- URL haus [33], è un servizio per la distribuzione di url malevoli
- Urlscan [34] , è un servizio usato per l'analisi url, domini, autonomous system e hashes

Questi servizi sono molto diffusi nel campo della Cyber Threat Intelligence. Le loro API e plugin sono elementi quasi costanti nei progetti in questo ambito. Considerato il contesto di sviluppo della tesi, la scelta di utilizzare queste funzionalità è stata quasi automatica.

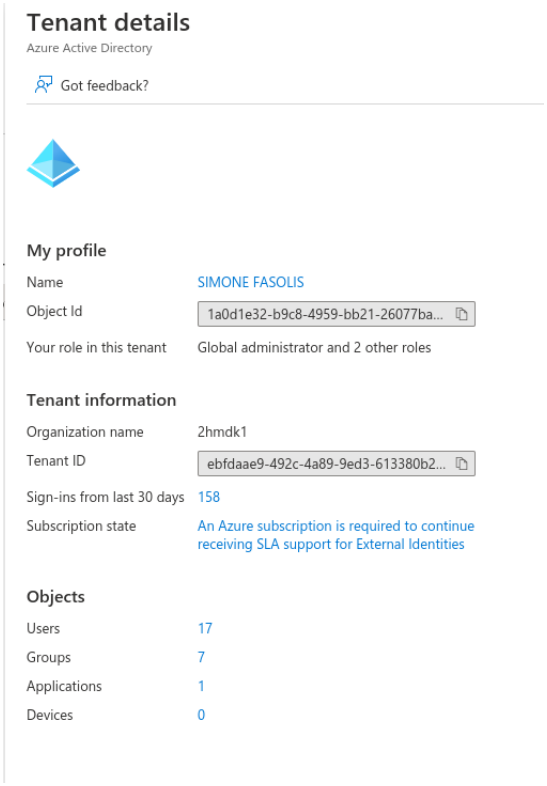
3.2 Servizi Microsoft

3.2.1 Azure e Active Directory

Il primo passo è stato la creazione di un account Microsoft Azure e di un tenant su Active Directory, affinché il sistema possa interfacciarsi con esso e gestire tutte le informazioni necessarie.

Per questo punto è stato di grande utilità il Microsoft 365 Developer Program. Il Developer Program ha permesso la creazione di un tenant AD su Microsoft Azure, chiamato *2hmdk1*, con permessi amministrativi

Figura 3.1. Dettagli del tenant.

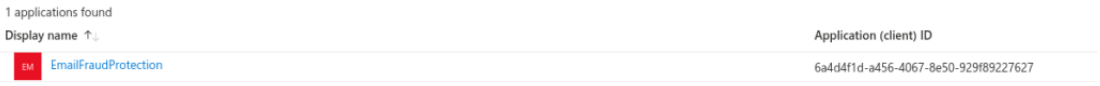


In questo tenant sono stati registrati 17 account utente (16 di default), di cui uno personale s287515@2hmdk1.onmicrosoft.com

3.2.2 Registrazione

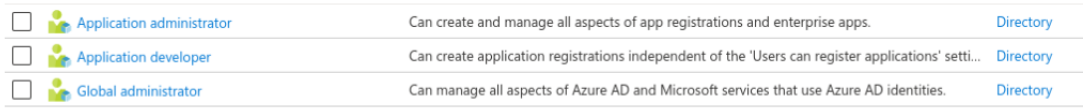
Per poter usare le funzionalità Active Directory attraverso un programma, è stato necessario usare Microsoft Authentication Library (MSAL), una libreria con tutte le API necessarie per comunicare con la piattaforma Microsoft Identity.
Microsoft Identity è responsabile dell'autenticazione e della generazione dei token di autenticazione usati nelle richieste verso la Directory. Il primo requisito per usare MSAL è la registrazione della propria applicazione nel tenant appena creato.

Figura 3.2. Registrazione dell'applicativo.



All'account utente sono stati associati dei permessi da amministratore per poter creare e gestire la registrazione di applicazioni, e per poter gestire le identità Azure AD.

Figura 3.3. Ruoli dell'account.



Si nota l'*Application ID*, usato per ottenere un token di accesso per le API di Microsoft 365.

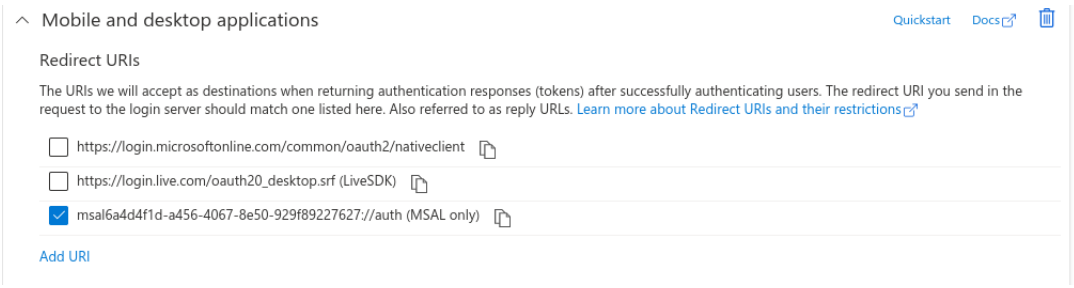
Figura 3.4. Dettagli dell'applicativo.

3.2.3 Configurazione

Alla registrazione seguono alcune fasi:

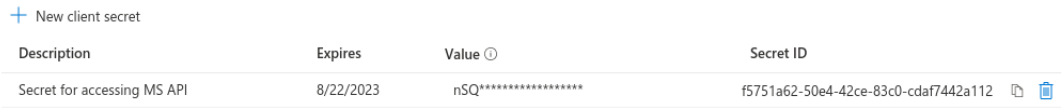
- specifica dell'URI che l'applicazione userà per prelevare i token di autenticazione; questo punto è gestito dall' SDK MSAL in Python

Figura 3.5. URI di autenticazione.



- creazione di una chiave segreta usata durante l'autenticazione iniziale dell'applicazione con Microsoft Identity, attraverso le API di MSAL

Figura 3.6. Segreti dell'applicativo.



- configurazione dei permessi da inserire nel token di autenticazione per l'applicazione. Questi definiscono quali API l'applicazione potrà chiamare tramite Microsoft Graph. I permessi scelti riguardano la lettura e scrittura di email e la lettura di gruppi, utenti e file

Figura 3.7. Permessi dell'applicativo.

API / Permissions name	Type	Description	Admin consent requ...	Status
EmailFraudProtection (1)				
Email.Read	Application	Email Reader	Yes	Granted for 2hmdk1
Microsoft Graph (9)				
Files.Read.All	Application	Read files in all site collections	Yes	Granted for 2hmdk1
Group.Read.All	Application	Read all groups	Yes	Granted for 2hmdk1
Mail.Read	Application	Read mail in all mailboxes	Yes	Granted for 2hmdk1
Mail.ReadBasic	Application	Read basic mail in all mailboxes	Yes	Granted for 2hmdk1
Mail.ReadBasic.All	Application	Read basic mail in all mailboxes	Yes	Granted for 2hmdk1
Mail.ReadWrite	Application	Read and write mail in all mailboxes	Yes	Granted for 2hmdk1
Mail.Send	Application	Send mail as any user	Yes	Granted for 2hmdk1

A questo punto l'applicazione ha tutti i requisiti necessari per poter usufruire delle API messe a disposizione da Microsoft Graph.

3.2.4 Graph

Microsoft Graph è un insieme di REST API in grado di manipolare, prelevare e inserire ogni tipo di informazione processabile da Microsoft 365, per esempio:

- Advanced Threat Analytics
- Advanced Threat Protection
- Azure Active Directory
- Bookings
- Calendar
- Excel
- Identity Manager
- Microsoft Search
- OneDrive
- OneNote
- Outlook/Exchange
- Outlook contacts
- SharePoint
- Teams

Il servizio offre un unico endpoint di partenza: *https://graph.microsoft.com*. Usandolo è possibile accedere a tutte le risorse disponibili variando il path dell'url, a patto che ci siano i permessi necessari per interagire con la risorsa richiesta.

Per chiamare le API di Graph è necessario, quindi, aver ricevuto un token di accesso da Identity, che verrà inserito nell'header di ogni richiesta.

Nell'applicazione sono state usate le seguenti API (l'hostname è sempre lo stesso, in seguito viene indicato solo l'URI):

- **GET**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages*, uri usato per prelevare i messaggi da un determinato folder per un determinato utente

```
emails = requests.get(
     base_url,
    headers={'Authorization': 'Bearer ' + token}
).json()
```

- **GET**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages/{email_id} ?$select=internetMessageHeaders*, uri usato per prelevare numerosi metadati relativi ad una particolare email

```
header = requests.get(
    f"{base_url}/{email['id']}?$select=internetMessageHeaders",
    headers={'Authorization': 'Bearer ' + token}
).json()
```

- **GET**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages/{email_id}/attachments*, uri usato per prelevare gli allegati associati ad una email

```
attachment = requests.get(
    f"{base_url}/{email['id']}/attachments",
    headers={'Authorization': 'Bearer ' + token}
).json()
```

- **GET**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages/delta?${delta_token}*, uri usato per prelevare tutte le email che hanno subito modifiche successivamente ad un determinato istante temporale, indicato dal delta token, da un determinato folder di un determinato utente

```
emails = requests.get(
    f"{base_url}/delta?${delta_token}",
    headers={'Authorization': 'Bearer ' + token}
).json()
```

- **POST**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages*, uri usato per inviare una mail in un determinato folder ad un determinato utente. La richiesta richiede di inserire il messaggio che si vuole inviare

```
requests.post(
    f"{base_url}/{client_email}/mailFolders/{report_id}/messages",
    headers={'Authorization': 'Bearer ' + token},
    json=message
)
```

- **GET**: */v1.0/users/{client_mail}/mailFolders*, uri usato per ricevere tutti i folder disponibili nella casella postale di un determinato utente

```
inboxes = requests.get(
    f"{base_url}/{user['mail']}/mailFolders",
    headers={'Authorization': 'Bearer ' + token}
).json()
```

- **POST**: */v1.0/users/{client_mail}/mailFolders*, uri usato per aggiungere un folder alla casella postale di un determinato utente. La richiesta richiede di specificare il nome del nuovo folder, la risposta ritorna l'id del nuovo folder

```
requests.post(
    f"{base_url}/{user['mail']}/mailFolders",
    headers={'Authorization': 'Bearer ' + token},
    json={"displayName": "Reports"}
)
```

- **GET**: */v1.0/users/*, uri usato per ottenere la lista degli utenti attualmente registrati nell'Active Directory dell'applicazione

```
requests.get(
    base_url,
    headers={'Authorization': 'Bearer ' + token}
).json()
```

3.3 Servizi open-source

3.3.1 VirusTotal

VirusTotal è un servizio open-source per l'analisi di file, domini, IP e url. VirusTotal utilizza antivirus e web scanner per rilevare malware o contenuti sospetti. Per ogni elemento analizzato, VirusTotal offre un indicatore di bontà basato sul riscontro di vari vendors, come Fortinet, Google Safebrowsing, Kaspersky e URLhaus.

Il servizio è disponibile tramite applicazione web o, come usato nel progetto, attraverso delle chiamate API [37]. L'uso delle API richiede la creazione di un account sulla piattaforma per la creazione di una API Key da usare in tutte le richieste verso il servizio.

Le API usate nel progetto sono due:

- **POST**: *https://www.virustotal.com/api/v3/files*, per passare al servizio l'allegato che si vuole analizzare. Nella richiesta è necessario inserire l'API key nell'header e il file che si vuole analizzare. Questa richiesta ritorna, con uno status code pari a 200, un id da usare in una futura GET per ottenere il report sul file analizzato.

  ```
  requests.post(
      "https://www.virustotal.com/api/v3/files",
      headers={'x-apikey': API_KEY_VIRUS_TOTAL},
      files={'file': attachment["contentBytes"]}
  ).json()
  ```

- **GET**: *https://www.virustotal.com/api/v3/analyses/*{*response_id*}, per ottenere il report del file analizzato. Il risultato è un JSON con una serie di indicatori assegnati da vari vendors (Avira, Fortinet, AlienVault, Kaspersky, URLhaus) sulla bontà del file

  ```
  requests.get(
      f"https://www.virustotal.com/api/v3/analyses/{response['data']['id']}",
      headers={'x-apikey': API_KEY_VIRUS_TOTAL}
  ).json()
  ```

3.3.2 AlienVault Open Threat Exchange

AlienVault OTX è una piattaforma per la condivisione di ogni genere di materiale riguardo la Cyber Threat Intelligence, quali minacce emergenti, metodologie di attacchi, attori e risorse malevoli

Il servizio ha diverse funzionalità; le funzionalità usate nel progetto riguardano l'analisi di file ricevuti come allegati, l'analisi di indirizzi IP e l'analisi di hostnames [38]. L'uso di OTX richiede una API key valida, ottenuta con la creazione un account AlienVault, e di un Software Development Kit (SDK, per Python), per l'accesso API del servizio.

```
OTX_SERVER = 'https://otx.alienvault.com/'
otx = OTXv2(API_KEY_OTX, server=OTX_SERVER)
```

OTX è stato utilizzato per l'integrazione che offre con gli applicativi e per la possibilità di analizzare molteplici fonti, come file, url, domini, e hostname.

Gli indicatori usati nel progetto sono di quattro tipi:

- IndicatorTypes.IPv4, per gli indirizzi IP degli url
- IndicatorTypes.HOSTNAME, per gli hostnames degli url
- IndicatorsType.DOMAIN, per i domini degli indirizzi email

- IndicatorTypes.FILE_HASH_SHA256, per gli allegati

Per gli indirizzi IP, gli hostnames e i domini vengono richieste tre metriche associate al dato in questione:

```
otx.get_indicator_details_by_section(ty, data, "malware")
otx.get_indicator_details_by_section(ty, data, "url_list")
otx.get_indicator_details_by_section(ty, data, "passive_dns")
```

Per gli allegati, invece, viene richiesto il report sull'analisi del file in questione, elencandone i dettagli negativi.

```
otx.get_indicator_details_by_section(IndicatorTypes.FILE_HASH_SHA256,
content_hash, "analysis")
```

3.3.3 URLhaus

URLhaus è un progetto originato da abuse.ch [39] per l'analisi e condivisione degli url malevoli usati per la distribuzione di malwares.

Nel progetto non vengono usate delle chiamate ad API, ma è stato pre-caricato un database con tutti gli url raccolti negli ultimi anni [40]. Questa scelta è stata fatta per ridurre il tempo di attesa causato dalle richieste API. In caso di email con numerosi url nel body, infatti, il tempo di attesa può essere piuttosto elevato.

Lo script dedicato al pre-caricamento degli url nel database utilizza anche una chiamata API a URLhaus:

- **POST**: *https://urlhaus-api.abuse.ch/v1/url/*, per verificare se il determinato url si trova in una blacklist.

La descrizione dello schema nel database è lasciata al capitolo dedicato al database design.

3.3.4 Urlscan

Urlscan è un servizio open-source per l'analisi di url. Questo fornisce una serie di statistiche:

- numero e descrizione (metodo, status code, path, latenza) delle richieste effettuate
- uso di TLS
- percentuale di indirizzi ipv6
- numero di domini e sottodomini
- numero di cookies

Questi valori vengono usati per generare un indice di bontà variabile da -100 (legittimo) a +100 (malevolo).

Il progetto fa uso di chiamate API verso il servizio [41], autenticate da un API Key ottenuta con la creazione di un account, per verificare se un determinato url trovato nella mail, classificato come malevolo, dovrà essere salvato in un database relativo a soli url malevoli.

Le API sono due:

- **POST**: *https://urlscan.io/api/v1/scan/*, per comunicare al servizio l'url da analizzare. Alla richiesta va integrata un'API key e l'url da analizzare. L'API ritorna, nel caso di status code pari a 200, un id da usare in una futura GET per prelevare il report sull'url analizzato

```
requests.post(
    'https://urlscan.io/api/v1/scan/',
    headers = {
            'API-Key': URL_SCAN_API_KEY,
            'Content-Type': 'application/json'
         },
    data=json.dumps({"url": url, "visibility": "public"})
)
```

- **GET**: *https://urlscan.io/api/v1/result/*{*response_id*}, per prelevare il report generato dal servizio sull'url analizzato

```
requests.get(
    f"https://urlscan.io/api/v1/result/{response.json()['uuid']}",
    headers = {'API-Key': URL_SCAN_API_KEY,
               'Content-Type': 'application/json'}
)
```

Capitolo 4

Analisi della email

4.1 Introduzione

Questo capitolo è dedicato alla descrizione della struttura di una generica email prelevata tramite le API di Microsoft Graph. Alla email possono essere associati degli allegati e una serie di headers inseriti dagli exchange server di Microsoft durante il trasferimento della email nella casella postale dell'utente.

Le principali API usate per estrarre queste informazioni sono (viene elencato solo l'URI per la risorsa):

- **GET**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages*, uri usato per prelevare i messaggi da un determinato folder per un determinato utente
- **GET**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages/{email_id} ?$select=internetMessageHeaders*, uri usato per prelevare numerosi metadati relativi ad una particolare email
- **GET**: */v1.0/users/{client_mail}/mailFolders/{folder_id}/messages/{email_id}/attachments*, uri usato per prelevare gli allegati associati ad una email

Ogni richiesta è integrata con una API key per l'autenticazione della richiesta.

Tutti i JSON ottenuti dalle API tornano, oltre alle risorse richieste, alcuni campi particolari:

- *@odata.context*, è il path completo associato alla richiesta API
- *@odata.nextLink*, contiene il path per il prossimo blocco di email in quanto sono ricevute a blocchi di dieci messaggi alla volta
- *@odata.deltaLink*, presente solo con la variante incrementale delle API. Contiene un url con un delta token da usare per la prossima richiesta incrementale

4.2 Descrizione dell'email

La email ottenuta dalle API è ricevuta in formato JSON. In seguito c'è la descrizione dei vari campi:

- *@odata.etag*, campo usato per la concorrenza dei dati ed evitare la scrittura multipla, simile ad un Lock
- *id*, identificativo della mail nel dominio MS

- *createdDateTime*, timestamp della creazione della mail nella casella postale dell'utente
- *changeKey*, versione della mail
- *categories,* lista con le categorie associate al messaggio
- *receivedDateTime*, timestamp del tempo di ricezione
- *sentDateTime*, timestamp di invio dal from
- *hasAttachment*, indica se nell mail è presente almeno un allegato
- *internetMessageId*, id del messaggio secondo il formato RFC2822
- *subject*, oggetto della mail
- *bodyPreview*, primi 255 caratteri della corpo della mail
- *importance*, priorità del messaggio. Può essere low, normal o high
- *parentFolderId*, id del folder in cui la mail è stata recapitata
- *conversationId*, id della conversazione a cui appartiene la mail
- *conversationIndex*, indica la posizione del massaggio nella conversazione, simile ad un offset
- *isDeliveryReceiptRequested*, se presente, la ricezione della email invierà una notifica al mittente
- *isReadReceiptRequested*, se presente, l'apertura della mail invierà una notifica al mittente
- *isRead*, indica se il messaggio è stato aperto o meno
- *isDraft*, indica se il messaggio è considerato come bozza
- *webLink*, url per aprire il messaggio nell'applicazione web
- *inferenceClassification*, classificazione del messaggio per importanza. Può essere focused o other
- *body*, corpo della mail. Questo campo è un JSON
  - *contentType*, tipo MIME del messaggio
  - *content*, contenuto della mail
- *sender*, account usato per generare il messaggio, in genere è lo stesso del from. Questo campo è un JSON
  - *emailAddress*, JSON che rappresenta le informazioni di un contatto
    - *name*, nome associato all'indirizzo email usato
    - *address*, indirizzo mail
- *from*, proprietario della casella postale da cui il messaggio è stato inviato. Questo campo è un JSON
  - *emailAddress*, JSON che rappresenta le informazioni di un contatto
    - *name*, nome associato all'indirizzo email usato
    - *address*, indirizzo mail
- *toRecipients*, campo contenente una lista di JSON con i dati dei contatti a cui la mail è stata inviata
- *ccRecipients*, campo contenente una lista di JSON con i dati dei contatti in copia
- *bccRecipients*, campo contenente una lista di JSON con i dati dei contatti in copia nascosta
- *flag*, flag usato per indicare lo status o date di inizio, completamento previsto o reale. I campo possono essere completedDateTime, dueDateTime, flagStatus, o startDateTime

4.3 Descrizione dei custom headers

Gli headers ottenuti dalle API sono ritornati in formato JSON. In seguito c'è la descrizione dei campi principali:

- *@odata.etag*, campo usato per la concorrenza dei dati ed evitare la scrittura multipla, simile ad un Lock
- *id*, identificativo univoco della mail
- *internetMessageHeaders*, lista con tutti gli headers
  - *Received*, questi campi possono essere multipli, contengono il percorso eseguito dalla email per attivare alla casella postale dell'utente
  - *Received-SPF*, risultato del Sender Policy Framework
  - *DKIM-Signature*, risultato del Domain Key Identified Mail
  - *MIME-Version*, versione del protocollo MIME usata
  - *From*, indirizzo da cui il messaggio è stato inviato
  - *Date*, timestamp con la data di ricevimento
  - *Message-ID*, identificativo della mail in formato RFC-2822
  - *Subject*, oggetto della mail
  - *To*, indirizzo del destinatario
  - *Content-Type*, content type della mail
  - *Return-Path*, campo con l'indirizzo del mittente, usato in caso di errore nell'invio della mail per inviare la mail all'origine
  - *X-Received*, campo aggiunto da un user-agent o da un server SMTP
  - *X-Google-DKIM-Signature*, campo contenente informazione riguardo il sender, il messaggio e la locazione della chiave pubblica usata per la verifica del messaggio
  - *X-Gm-Message-State*, questo campo determina lo stato del messaggio. Può essere successo o ritornato indietro
  - *X-Google-Smtp-Source*, campo usato per identificare il percorso SMTP della mail
  - *Authentication-Results*, campo contenente i risultati di alcuni protocolli di autenticazione, come SPF, DKIM, DMARC o COMPAUTH generato da operazioni degli Exchange Online Protection di Microsoft
  - *X-Microsoft-Antispam*, contiene informazioni sullo classificazione della mail come spam o phishing; generato da operazioni degli Exchange Online Protection di Microsoft
  - *X-Forefront-Antispam-Report*, contiene informazioni su come il messaggio è stato elaborato; generato da operazioni degli Exchange Online Protection di Microsoft
    - *CIP*, è l'indirizzo IP usato dal mittente
    - *CTRY*, è la nazione di origine
    - *LANG*, è il linguaggio del messaggio
    - *SCL*, è lo Spam Confidence Level del messaggio. Un valore alto significa che il messaggio è probabilmente spam
    - *SRV*, indica se il messaggio è stato considerato come spam
    - *IPV*, indica se l'IP è stato trovato in qualche reputation list
    - *SFV*, indica se il messaggio è stato marchiato come non spam e inviato all'utente
    - *H*, è la stringa HELO usata durante la connessione con l'email server tramite SMTP
    - *PTR*, è il PTR record dell'IP di origine
    - *CAT*, è la categoria della policy di protezione
    - *SFS* , indica quali regole di spam sono state attivate

4.4 Descrizione degli allegati

Gli allegati ottenuti dalle API sono ritornati in formato JSON. In seguito c'è la descrizione dei campi:

- *@odata.type*, indica il tipo di allegato. Può essere fileAttachment (un file), itemAttachment (un evento, contatto o messaggio) o referenceAttachment (un link ad un file)
- *@odata.mediaContentType*, tipo MIME dell'allegato
- *id*, identificativo dell'allegato nel dominio Microsoft
- *lastModifiedDateTime*, timestamp di ultima modifica
- *name*, nome dell'allegato
- *contentType*, tipo MIME dell'allegato
- *size*, dimensione dell'allegato
- *isInline*, indica se è possibile vedere il file direttamente dal corpo della mail
- *contentId*, identificativo del conenuto
- *contentLocation*, path dell'allegato, può essere nullo se il contenuto è inserito direttamente in formato JSON
- *contentBytes*, contenuto codificato

Capitolo 5

Feature analizzate

5.1 Introduzione

Questo capitolo è dedicato alla descrizione di tutte le feature usate nel progetto.

Le feature sono divise in quattro categorie, in base ai campi da cui sono state estratte:

- oggetto
- corpo
- contatti
- url

Non tutte le feature estratte sono state usate per la classificazione delle email nel modello finale. La parte di selezione delle feature migliori è descritta nel capitolo dedicato alla modellistica.

Non ci sono feature estratte degli allegati a causa dell'impossibilità nel trovare un dataset con abbastanza elementi. L'analisi e la classificazione degli allegati segue un percorso differente rispetto agli altri elementi. Questo verrà discusso nel capitolo dedicato alla classificazione.

5.2 Feature dell'oggetto

In seguito sono descritte le feature estratte dall'oggetto della mail:

- *subj_exist*, è 1 se l'oggetto esiste, 0 altrimenti. Si presume che ogni email abbia un oggetto e che la mancanza di esso sia un indicatore di pericolosità
- *subj_length*, è la lunghezza dell'oggetto. Potrebbero esserci relazioni tra un oggetto troppo lungo/corto e una mail potenzialmente pericolosa
- *subj_nr_words*, è il numero di parole dell'oggetto. Potrebbero esserci relazioni tra un oggetto con troppo/poche parole e una mail potenzialmente pericolosa
- *subj_ratio_words*, è il rapporto tra il numero di parole nell'oggetto e la sua lunghezza. Potrebbero esserci relazioni tra un oggetto con troppo/poche parole e una mail potenzialmente pericolosa
- *subj_nr_non_alpha*, è il numero di caratteri non alfanumerici nell'oggetto. La presenza di troppi caratteri non alfanumerici nell'oggetto potrebbe essere un indicatore di pericolosità
- *subj_ratio_non_alpha*, è il rapporto tra i caratteri non alfanumerici e la lunghezza dell'oggetto. La presenza di troppi caratteri non alfanumerici nell'oggetto potrebbe essere un indicatore di pericolosità

- *subj_nr_digits*, è il numero di caratteri numerici nell'oggetto. La presenza di troppi numeri nell'oggetto potrebbe essere un indicatore di pericolosità
- *subj_ratio_digits*, è il rapporto tra i caratteri numerici e la lunghezza dell'oggetto. La presenza di troppi numeri nell'oggetto potrebbe essere un indicatore di pericolosità
- *subj_nr_sus_words*, è il numero di parole sospette, ovvero parole trovate in un database di parole comunemente usate nelle email di phishing e di spam. Troppe parole potrebbe essere un indicatore di pericolosità
- *subj_tz*, è 1 se il timezone del timestamp è in un certo intervallo, nello specifico tra -0100 e +0200, 0 altrimenti. Una email ricevuta da un luogo con un fuso orario anomalo potrebbe essere un indicatore di pericolosità. Questa è anche una delle feature introdotte nell'articolo [22]

5.3 Feature del corpo

In seguito sono descritte le feature estratte dal corpo della mail:

- *body_length*, è la lunghezza del corpo. Corpi troppo lunghi/corti potrebbero essere indicatori di pericolosità
- *body_nr_words*, è il numero di parole del corpo. Corpi con troppe/poche parole potrebbero essere indicatori di pericolosità
- *body_nr_url*, è il numero di url nel corpo. Email con troppi url potrebbero essere pericolosi
- *body_ratio_url*, è il rapporto tra il numero di url e la lunghezza del corpo. Un rapporto troppo elevato potrebbe essere un indice di pericolosità
- *body_nr_atch*, è il numero di allegati nel corpo. Email con troppi allegati potrebbero essere pericolose
- *body_ratio_atch*, è il rapporto tra il numero di allegati e la lunghezza del corpo. Un rapporto troppo elevato potrebbe essere un indicatore di pericolosità
- *body_subj_ratio*, è il rapporto tra la lunghezza dell'oggetto e la lunghezza del corpo. Un rapporto troppo elevato o ridotto potrebbe essere un indicatore di pericolosità
- *body_nr_digits*, è il numero di caratteri numerici nel corpo. La presenza di troppi numeri potrebbe essere un indicatore di pericolosità
- *body_ratio_digits*, è il rapporto tra i caratteri numerici e la lunghezza del corpo. La presenza di troppi numeri potrebbe essere un indicatore di pericolosità
- *body_nr_non_alpha*, è il numero di caratteri non alfanumerici nel corpo. La presenza di troppi caratteri non alfanumerici potrebbe essere in indice di pericolosità
- *body_ratio_non_alpha*, è il rapporto tra i caratteri non alfanumerici e la lunghezza del corpo. La presenza di troppi caratteri non alfanumerici potrebbe essere in indice di pericolosità
- *body_nr_img_tags*, è il numero di tag *img* nel corpo. Questi potrebbero essere vettori per codici malevoli. Delle varianti di questa feature sono presenti negli articoli [20] e [23]
- *body_nr_a_tags*, è il numero di tag *a* nel corpo. Questa feature è utilizzata anche nell'articolo [24]
- *body_nr_href*, è il numero di *href* nel corpo
- *body_nr_form*, è il numero di tag *form* nel corpo. Potrebbero essere form da compilare con informazioni sensibili

- *body_nr_iframe*, è il numero di tag *iframe* nel corpo. Questi potrebbero essere vettori per script dannosi
- *body_nr_pwd_input*, è il numero di tag *input* con type= password nel corpo. L'inserimento di una password potrebbe essere pericoloso
- *body_nr_script_tags*, è il numero di tag *script* nel corpo. Potrebbe essere uno script malevolo. Questa feature è ricorrente nell'articolo [23]
- *body_nr_onclick*, è il numero di eventi OnClick nel corpo. L'evento potrebbe eseguire script dannosi. Questa feature è ricorrente nell'articolo [23]
- *body_nr_sus_words*, è il numero di parole sospette nel corpo. Troppe parole sospette potrebbe essere un indicatore di pericolosità
- *body_ratio_sus_words*, è il rapporto tra le parole sospette e il numero di parole nel corpo. Troppe parole sospette potrebbe essere un indicatore di pericolosità
- *body_content_type_exist*, è 1 se il contentType esiste, 0 altrimenti. Deve esserci un content type
- *body_content_type_is_text_plain*, è 1 se il contentType è text/plain, 0 altrimenti
- *body_charset_exist*, è 1 se il contentType contiene un charset, 0 altrimenti. Dovrebbe esserci un charset

5.4 Feature dei contatti

In seguito sono descritte le feature estratte dai contatti:

- *sender_addr_ret_path*, è 1 se il l'indirizzo email del sender è uguale al return path, 0 altrimenti
- *sender_addr_reply*, è 1 se l'indirizzo email del sender si trova anche nel replyTo, 0 altrimenti
- *sender_found_in_db*, è 1 se il nome del sender, con un certo indirizzo, si trova nella lista di contatti nel database, 0 altrimenti. Un utente sconosciuto è molto più probabile che sia uno scammer rispetto ad un contatto salvato
- *from_addr_ret_path*, è 1 se il l'indirizzo email del from è uguale al return path, 0 altrimenti
- *from_addr_reply*, è 1 se il l'indirizzo email del from è inserito nel replyTo, 0 altrimenti
- *from_found_in_db*, è 1 se il nome del from, con un certo indirizzo, si trova nella lista di contatti nel database, 0 altrimenti. Un utente sconosciuto è molto più probabile che sia uno scammer rispetto ad un contatto salvato
- *recipients_found_in_db*, è il numero di associazioni nome e indirizzo email nei campi replyTo/cc/bcc trovati nella contact list
- *from_nr_digits*, è il numero di caratteri numerici trovati all'indirizzo email del from. Un indirizzo con dei caratteri numerici è anomalo e potrebbe essere un indicatore di pericolosità
- *from_nr_non_alpha*, è il numero di caratteri non alfanumerici trovati all'indirizzo email del from. Un indirizzo con caratteri non alfanumerici è anomalo e potrebbe essere un indicatore di pericolosità
- *sender_nr_digits*, è il numero di caratteri numerici trovati all'indirizzo email del sender. Un indirizzo con dei caratteri numerici è anomalo e potrebbe essere un indicatore di pericolosità
- *sender_nr_non_alpha*, è il numero di caratteri non alfanumerici trovati all'indirizzo email del sender. Un indirizzo con caratteri non alfanumerici è anomalo e potrebbe essere un indicatore di pericolosità

- *from_virustotal_nr_harmless*, è il numero di risultati harmless del report associato al dominio del from, ottenuto da una richiesta API di VirusTotal. Un dominio con con pochi giudizi positivi potrebbe essere un indicatore di pericolosità
- *from_virusTotal_nr_suspicious*, è il numero di risultati suspicious del report associato al dominio del from, ottenuto da una richiesta API di VirusTotal. Un dominio con molti giudizi sospetti potrebbe essere un indicatore di pericolosità
- *from_virusTotal_nr_dangerous*, è il numero di risultati dangerous del report associato al dominio del from, ottenuto da una richiesta API di VirusTotal. Un dominio con dei giudizi negativi potrebbe essere un indicatore di pericolosità
- *from_virusTotal_ratio*, è il rapporto tra il numero di malicious e il numero di harmless del report associato al dominio del from. Un rapporto troppo alto potrebbe essere un indicatore di pericolosità
- *from_domain_malware_otx*, è il numero di malware associati al dominio del from, ottenuto da una richiesta API di AlienVault OTX. Troppi malware associati al dominio potrebbero essere un indicatore di pericolosità
- *from_domain_url_otx*, è il numero di url associati al dominio del from, ottenuto da una richiesta API di AlienVault OTX. Non è usato come feature diretta
- *from_domain_dns_otx*, è il numero di record DNS associati al dominio del from, ottenuto da una richiesta API di AlienVault OTX. Non è usato come feature diretta
- *from_domain_ratio_url_otx*, è il rapporto tra il numero di malware e il numero di url associati al dominio del from. Un indice troppo alto potrebbe essere un indicatore di pericolosità
- *from_domain_ratio_dns_otx*, è il rapporto tra il numero di malware e il numero di url associati al dominio del from. Un indice troppo alto potrebbe essere un indicatore di pericolosità

5.5 Feature degli url

In seguito sono descritte le feature estratte dal singolo url:

- *url_length*, è la lunghezza dell'url. Un url troppo lungo/corto potrebbe essere un indicatore di pericolosità
- *url_netloc_length*, è la lunghezza del network location. Netloc troppo lunghe o corte potrebbero essere indicatori di pericolosità
- *url_ratio_netloc*, è la rapporto tra la lunghezza del network location e la lunghezza dell'url. Rapporti troppo lunghi o corti potrebbero essere pericolosi
- *url_path_length*, è la lunghezza del path. Path troppo lunghe potrebbero essere indicatori di pericolosità
- *url_ratio_path*, è il rapporto tra la lunghezza del path e la lunghezza dell'url. Path troppo lunghe potrebbero essere indicatori di pericolosità
- *url_queries_length*, è la lunghezza della query. Query troppo lunghe potrebbero essere vettori di attacco
- *url_nr_netloc*, è il numero di domini nel network location. Url con troppi domini potrebbero essere indicatori di pericolosità
- *url_nr_sub*, è il numero di sottodomini del network location. Troppi sottodomini potrebbero essere indicatori di pericolosità
- *url_nr_path_digits*, è il numero di caratteri numerici nel path. Un path con dei numeri è anomalo e potrebbe essere pericoloso

- *url_ratio_digits*, è il rapporto tra il numero di caratteri numerici e la lunghezza dell'url. Un path con dei simboli è anomalo e potrebbe essere pericoloso
- *url_nr_path_non_alpha*, è il numero di caratteri non alfanumerici nel path. Un rapporto elevato potrebbe essere un indicatore di pericolosità
- *url_ratio_non_alpha*, è il rapporto tra il numero di caratteri non alfanumerici e la lunghezza dell'url. Un rapporto elevato potrebbe essere un indice di pericolosità
- *url_nr_queries*, è il numero di argomenti nella query string. Troppi argomenti potrebbero essere indicatori di pericolosità
- *url_path_level*, è il numero di livelli del path
- *url_ssl*, è 1 se usa TLS, 0 altrimenti. Un url che non usa TLS è potenzialmente dannoso. Questa feature è utilizzata anche nell'articolo [25]
- *url_port*, è 1 se la porta è 80 o 443, 0 altrimenti. Un url con una porta non standard potrebbe essere dannoso. Questa feature è anche usata nell'articolo [17]
- *url_short_not_exist*, è 1 se l'url non è short, 0 altrimenti. Un url short potrebbe essere dannoso. Questa feature è utilizzata anche nell'articolo [14]
- *url_not_clear_ip*, è 1 se non compare un indirizzo IP in chiaro nell'url, 0 altrimenti. Un url con un indirizzo IP in chiaro è anomalo ed è potenzialmente dannoso. Questa feature è utilizzata anche nell'articolo [14]
- *url_not_path_upper*, è 1 se nel path non compaiono almeno due caratteri maiuscoli di fila, 0 altrimenti. Un path con multipli caratteri maiuscoli è anomalo ed è potenzialmente dannoso
- *url_not_path_single_char*, è 1 se non compare un livello del path con un singolo carattere, 0 altrimenti. Un path con un livello a singolo carattere è anomalo ed è potenzialmente dannoso
- *url_nr_percent*, è il numero di % nell'url. Gli % sono indicatori di url encoding e potrebbero essere indicatori di pericolosità
- *url_nr_dot*, è il numero di . nell'url. Un url con tanti . è anomalo e potrebbe essere un indicatore di pericolosità
- *url_nr_at*, è il numero di @ nell'url. La presenza di @ è anomala ed potrebbe essere un indicatore di pericolosità
- *url_nr_hyphen*, è il numero di - nell'url. Gli - sono anomali e potrebbero essere indicatori di pericolosità
- *url_nr_underscore*, è il numero di _ nell'url. Gli _ in un url sono anomali e potrebbero essere degli indici di pericolosità
- *url_nr_double_slash*, è il numero di // nell'url. Un // indica la presenza di un url. Due // sono anomali e potrebbe essere un indicatore di pericolosità
- *url_not_bl*, è 1 se l'url non è in blacklist, 0 altrimenti. Un url in blacklist è pericoloso
- *url_tld_not susp*, è 1 se il Top Level Domain non è in blacklist, 0 altrimenti. Un Top Level Domain in blacklist è pericoloso
- *url_not_found*, è 1 se l'url non si trova nel database di url malevoli, 0 altrimenti. Un url trovato nel database di url malevoli è pericoloso
- *url_host_not_found*, è 1 se l'hostname dell'url non si trova del database di url malevoli, 0 altrimenti. Un url con un hostname appartenente al database di url pericolosi è un indice di potenziale pericolosità

- *url_path_not_found*, è 1 se il path non si trova nel database di url malevoli, 0 altrimenti. Un url con un path appartenente al database di url malevoli è un indicatore di potenziale pericolosità
- *url_path_query_not_found*, è 1 se il path con una certa query non si trova nel database di url malevoli, 0 altrimenti. Un url con un path e una query appartenenti al database di url malevoli è un indicatore di potenziale pericolosità
- *url_host_malware_otx*, è il numero di malware associati all'hostname, ottenuto da una richiesta API di AlienVault OTX. Un hostname con troppi malware associati è un potenziale indicatore di pericolosità
- *url_host_url_otx*, è il numero di url associati all'hostname, ottenuto da una richiesta API di AlienVault OTX. Non è usato con feature diretta
- *url_host_dns_otx*, è il numero di record DNS associati all'hostname, ottenuto da una richiesta API di AlienVault OTX. Non è usato con feature diretta
- *url_host_ratio_url_otx*, è il rapporto tra il numero di malware e il numero di url trovati nella query verso AlienVault riguardo l'hostname dell'url. Un rapporto elevato è un potenziale indicatore di pericolosità
- *url_host_ratio_dns_otx*, è il rapporto tra il numero di malware e il numero di dns passivi trovati nella query verso AlienVault riguardo l'hostname dell'url. Un rapporto elevato è un potenziale indicatore di pericolosità

Capitolo 6

Database design

6.1 Introduzione

Per il corretto funzionamento dell'applicazione è opportuno tenere traccia delle informazioni ricevute in un qualche tipo di database.

Per questo progetto si è scelto di usare un database relazionale, nello specifico PostgreSQL, per poter sfruttare le operazioni di JOIN.

La scelta di quasto tipo di database è originata della maggior familiarità con questa tipologia di schemi. In questo progetto, la rigidità degli schemi fissi non è un problema, in quanto tutte le email trattate hanno le stesse informazioni e da esse vengono estratte le stesse tipologie di feature. Non esiste, quindi, la problematica di chiavi mancanti, trovata spesso nei database non relazionali.

Non è stato necessario rivolgersi verso soluzioni non relazionali, come MongoDB, o a grafo, come GraphQL, a causa della mancanza di benefici effettivi che queste tipologie di database avrebbero potuto offrire.

Un'ulteriore morivazione riguardo l'uso di postgreSQL è data dalla presenza di alcune funzionalità offerte dal sistema, che trovano grande utilità in questo contesto. PostgreSQL fornisce alcune funzioni specifiche, come l'uso di *tsvector* e *tsquery*, tipologie di dati adatte alla ricerca testuale. Queste due strutture verranno utilizzate per sostituire totalmente l'analisi testuale. Il *tsvector* è una struttura dati che permette la conversione di una generica stringa in token, ovvero frammenti di stringa, come parole o frasi, e associate alla loro frequenza. Il *tsvector* è una query focalizzata sui token del *tsvector*. La ricerca di parole o frasi utilizzando queste funzionalità è molto più efficiente.

Vengono utilizzati un paio dischemi esclusivamente per la persistenza degli id. A questi sono associati uno più schemi con le informazioni effettive. Per esempio sono state utilizzate delle tabelle eslusive per gli id associati ai messaggi e aglli indirizzi. L'associazione tra questi due è create con una terza tabella. Nel caso specifico ne esiste anche una quarta, in quanto viene anche separata l'associazione tra il destinatario del messaggio e il suo mittente.

L'id univoco di ogni record è ottenuto dall'estensione *uuid-ossp*; inoltre ogni record ha un campo *creation_time* con il timestamp di creazione.

6.2 Database design per le email

6.2.1 *contacts*

La tabella *contacts* salva i contatti dell'utente.

Un contatto è ogni tipo di utente rappresentato nella email: mittente, destinatari, cc, bcc. Di questi salvo solo l'indirizzo email.

```
email_address text not null
```

6.2.2 *contact_names*

La tabella *contact_names* salva tutti i nomi associati ad un indirizzo email. Bisogna notare come ad un unico indirizzo email possano essere associati diversi nomi, per questo motivo i nomi sono separati dagli indirizzi. Un'operazione di JOIN permetterà di collegare i due.

Oltre ai nomi, quindi, viene salvato il relativo *contact_id*, chiave esterna associata alla chiave primaria della tabella

```
name text not null
contact_id, uuid not null
```

6.2.3 *messages*

La tabella *messages* salva l'*external_id* associato all'email service e l'*internetMessageId* unico dell' email. Il corpo della email è salvato in un'altra tabella descritta in seguito.

```
external_id text not null
internet_message_id text not null
```

6.2.4 *messages_bodies*

La tabella *messages_bodies* salva il *message_id* (chiave esterna associata alla chiave primaria della tabella *messages*), il corpo della email e i token, elementi contenuti in una struttura dati unica di PostgreSQL chiamata *tsvector*, adatta alla ricerca testuale. Questi token rappresentano le singole parole e frasi estratte dalla email.

```
message_id uuid not null
body_content_type text not  null
body text not null
tokens tsvector not null
```

6.2.5 *messages_senders*

La tabella *messages_senders* salva la relazione tra il mittente e il messaggio che ha inviato. La separazione tra mittenti e destinatari è stata fatta per alleggerire il carico di lavoro del database nel mantenere tutte le relazioni tra i contatti. Una singola tabella, infatti, avrebbe avuto dimensioni molto maggiori rispetto alle due create.

Per questo, gli unici dati inseriti sono il *message_id*, chiave esterna associata alla chiave primaria della tabella *messages*, e il *sender_name_id* associato al nome dell'email del mittente, chiave esterna associata alla chiave primaria della tabella *contact_names*.

La combinazione dei due elementi forma la chiave primaria di questa tabella.

```
message_id uuid not null
sender_name_id uuid not null
```

6.2.6 *messages_recipients*

La tabella *messages_recipients* salva la relazione tra il destinatario e il messaggio che ha ricevuto. La separazione tra mittenti e destinatari è stata fatta per alleggerire il carico di lavoro del database nel mantenere tutte le relazioni tra i contatti. Una singola tabella, infatti, avrebbe avuto dimensioni molto maggiori rispetto alle due create.

Per questo gli unici dati inseriti sono il *message_id*, chiave esterna associata alla chiave primaria della tabella messages, e *recipient_name_id* associato al nome della email del destinatario, chiave esterna associata alla chiave primaria della tabella *contact_names*.

Si nota che possono esserci diversi tipologie di destinatari, quali to, cc e bcc. Al fine di differenziare possibili utenti che si trovano in due o più di queste categorie per lo stesso messaggio, ho creato una struttura enumerated con tre valori (to, cc e bcc).

```
create type recipient_types as enum ('to', 'cc', 'bcc')
```

La combinazione dei due id e del valore dell'enumerated forma la chiave primaria.

```
message_id uuid not null
recipient_name_id uuid not null
recipient_type recipient_types not null
```

6.2.7 *attachments*

La tabella *attachments* salva tutte le informazioni relative agli allegati, quali l'id nel dominio dell'email server, l'id del messaggio associato all'allegato (chiave esterna associata alla chiave primaria della tabella *messages*), il nome, i tokens relativi al nome, il formato MIME, la dimensione e l'hash SHA256 del contenuto.

```
attachment_id text not null
message_id uuid not null
name text not null
tokens tsvector not null
content_type text not null
size text not null
content_hash bytea not null
```

6.2.8 *headers*

La tabella *headers* salva l'oggetto dell'email, i token relativi all'oggetto, l'id del messaggio associato all'header (chiave esterna associata alla chiave primaria della tabella *messages*) e il return-path dell'email.

```
subject text
tokens tsvector
return_path text not null
```

6.3 Database design per le feature

6.3.1 *features*

La tabella *features* salva tutte le feature relative all'oggetto. Viene anche salvato l'identificativo relativo alla email da cui sono state estratte queste feature (chiave esterna relativa alla chiave primaria nella tabella *messages*).

```
subj_exist int
subj_length int
subj_nr_words int
subj_ratio_words float
subj_nr_non_alpha int
subj_ratio_non_alpha float
subj_nr_digits int
subj_ratio_digits float
subj_nr_sus_words int
```

6.3.2 *contact_features*

La tabella *contact_features* salva tutte le feature relative ai contatti (sender, from, to, cc, bcc). Viene anche salvato il *feature_id*, chiave esterna associata alla chiave primaria della tabella *features*. Nella tabella è possibile trovare feature relative a:

- equivalenze fra from, sender e return-path, replyTo, internetMessageId

  ```
  sender_addr_ret_path int not null
  from_addr_ret_path int not null
  from_addr_reply int not null
  from_addr_domain_eq int not null
  ```

- controllo della presenza dei contatti nel database

  ```
  from_found_in_db int not  null
  sender_found_in_db int not null
  recipients_found_in_db int not null
  ```

- controllo di lettere insolite o numeri

  ```
  from_nr_digits int not null
  from_nr_non_alpha int not null
  sender_nr_digits int not null
  sender_nr_non_alpha int not null
  ```

- uso di VirusTotal

  ```
  from_virusTotal_nr_malicious int not null
  from_virusTotal_nr_harmless int not null
  from_virusTotal_ratio float not  null
  ```

- uso di AlienVault OTX

  ```
  from_domain_malware_otx int not  null
  from_domain_url_otx int not null
  from_domain_dns_otx int not null
  from_domain_ratio_url_otx float not null
  from_domain_ratio_dns_otx float not null
  ```

6.3.3 *body_features*

La tabella *body_features* salva tutte le feature relative al corpo della email. Viene anche salvato il *feature_id*, chiave esterna associata alla chiave primaria della tabella *features*.

Nella tabella è possibile trovare feature relative a:

- statistiche varie sulla presenza di numeri, lettere insolite e parole sospette

```
body_length int not null
body_nr_words int not null
body_nr_digits int not null
body_ratio_digits float not null
body_nr_non_alpha int not null
body_ratio_non_alpha float not null
body_nr_sus_words int not null
body_ratio_sus_words float not null
body_subj_ratio float not null
```

- statistiche su url e allegati

```
body_nr_url int not null
body_ratio_url float not null
body_nr_atch int not null
body_ratio_atch float not null
```

- presenza di tag specifici, eventi o parole chiave

```
body_content_type_exist int not null
body_nr_img_tags int not null
body_nr_a_tags int not null
body_nr_href_links int not null
body_nr_form_tags int not null
body_nr_pwd_input int not null
body_nr_script_tags int not null
body_nr_iframe_tags int not null
body_nr_onclick int not null
```

6.3.4 *url_features*

La tabella features salva tutte le feature relative agli url. Viene anche salvato il *feature_id*, chiave esterna associata alla chiave primaria della tabella *features*.

Nella tabella è possibile trovare feature relative a:

- statistiche sulle lunghezze dell'url, path, netloc e query

```
url_length int not null
url_netloc_length int not null
url_ratio_netloc float not null
url_path_legnth int not null
url_ratio_path float not null
url_queries_length int not null
```

- statistiche sul numero di domini e livelli nel path

```
url_nr_netloc int not null
url_nr_sub int not null
url_nr_queries int not null
url_path_level int not null
```

- statistiche sulla presenza di caratteri insoliti e numeri

```
url_nr_path_digits int not null
url_ratio_digits flaot not null
url_nr_path_not_alpha int not null
url_ratio_non_alpha float not null
```

- presenza di caratteristiche insolite

```
url_ssl int not null
url_port int not null
url_short_notexist int not null
url_not_clear_ip int not null
url_not_path_upper int not null
url_not_path_single_char int not null
```

- presenza di specifici simboli

```
url_nr_percent int not null
url_nr_dot int not null
url_nr_at int not null
url_nr_underscore int not null
url_nr_hyphen int not null
url_nr_double_slash int not null
```

- presenza dell'url, path, hostname o query in un database di url malevoli

```
url_not_bl int not null
url_tld_not_susp int not null
url_not_found int not null
url_host_not_found int not null
url_path_not_found int not null
url_path_query_not_found int not null
```

- uso di VirusTotal

```
url_host_malware_otx int not null
url_host_url_otx int not null
url_host_dns_otx int non null
url_host_ratio_url_otx float not null
url_host_ratio_dns_otx float not null
```

6.4 Database design per le tabelle di supporto

6.4.1 Introduzione

Queste tabelle sono usate come supporto al progetto per l'estrazioni di alcune feature. Bisogna notare che tutte queste tabelle dovrebbero essere costantemente aggiornate nel tempo. In questo progetto questa problematica non viene considerata per semplicità e per la poca rilevanza con gli scopi del progetto.

6.4.2 *suspect_words*

La tabella *suspect_words* salva tutte le parole comunemente trovate nelle email di phishing o di spam. Questa tabella è usata in combinazione con una tsquery per una ricerca testuale ottimizzata.

```
word text not null
```

6.4.3 *urls_malicious*

La tabella *urls_malicious* salva tutti gli url malevoli trovati nel corso della classficazione. Di essi viene salvato lo schema, l'hostname, la porta, il path, i parametri, la query string, i fragment e se è stato trovato in una blacklist.

```
url text not null
schema text not null
hostname text not null
port int not null
path text not null
parameters text not null
queries text not null
fragments text not null
not_bl int not null
```

6.4.4 *domains*

La tabella *domains* salva il dominio di un indirizzo email e gli indicatori ritornati dai report di VirusTotal riguardo quel dominio.

```
domain text not null
nr_malicious int not null
nr_suspicious int not null
nr_harmless int not null
```

La scelta di salvare questi indicatori è stata fatta per non dover interrogare VirusTotal multiple volte per lo stesso dominio.

6.4.5 *domains_otx*

La tabella *domains_otx* salva gli indicatori ritornati dalle API di AlienVault per un determinato dominio email.

```
domain text not null
malware_nr int not null
url_nr int not null
dns_nr int not null
```

La scelta di salvare questi indicatori è stata fatta per non dover interrogare AlienVault multiple volte per lo stesso dominio.

6.4.6 *hostname_otx*

La tabella *domains_otx* salva gli indicatori ritornati dalle API di AlienVault per un determinato hostname.

```
hostname text not null
malware_nr int not null
url_nr int not null
dns_nr int not null
```

La scelta di salvare questi indicatori è stata fatta per non dover interrogare AlienVault multiple volte per lo stesso hostname.

6.5 Indici

Ad alcune tabelle sono associate degli indici per velocizzare le operazioni di ricerca e di JOIN tra tabelle, soprattutto nelle tabelle di grosse dimensioni.

Gli indici sono applicati:

- alle tabelle principali

```
create index i_messages_external_id_1 on messages_1( external_id );
create index i_messages_recipients_r2m_1
    on messages_recipients_1 ( recipient_name_id );
create index i_headers_id_1 on headers_1 ( message_id );
create index i_attachment_id_1 on attachments_1 ( message_id );
```

- alle tabelle delle feature

```
create index i_features_id_1 on features_1 ( message_id );
```

- alle tabelle di supporto

```
create index if not exists i_urls_malicious_hostname
    on urls_malicious ( url );
create index if not exists i_domain on domains ( domain );
create index if not exists i_domain_otx on domains_otx_1 ( domain );
create index if not exists i_host_otx on hostname_otx_1 ( hostname );
```

Capitolo 7

Dataset analizzati

7.1 Ricerca dei dataset

Per addestrare e verificare il funzionamento dei modelli sviluppati per la classificazione è stato necessario raccogliere del materiale. Lo scopo di questa raccolta è quello di lavorare con un'ampia varietà di tipologie di email, sia per classificazione (legittima, spam o phishing) che per numero effettivo di risorse.

Questo metodo è stato applicato sia alle email che ai soli url, argomento a cui è stato dedicato un modello e, quindi, dei dataset per l'addestramento.

In seguito c'è la descrizione dei vari dataset utilizzati in ambito email:

- *Fraudulent E-mail Corpus* [42], collezione di quasi 3000 email di phishing originarie dalla Nigeria, in genere riguardo richiesta di denaro con urgenza in cambio di somme più cospicue
- *The Enron Email Dataset*, collezione rielaborata del 2015 di email associate alla Enron Corporation. Contiene oltre mezzo milione di email di numerosi impiegati e dirigenti della compagnia
- *Fraud Email Dataset* [43], collezione di quasi 12000 corpi di email di phishing
- *Hillary Clinton's Emails* [44], collezione di email usate in una campagna diffamatoria. Il dataset contiene poco più di 7000 email
- *PublicCorpus* [45], collezione di oltre 9000 email estratte dai dataset di SpamAssassin, contenenti 6400 email legittime e 2600 email di spam
- *LingSpam Dataset* [46], collezione di 2600 email generiche, di cui 2200 legittime e 400 di spam
- *Spam Email* [47], collezione 5500 email generiche, di cui 4800 email legittime e 700 di spam
- *Spam Emails Dataset* [48], collezione di 5000 email prelevate da Spambase, contenente 3500 email legittime e 1500 email di spam

In seguito c'è la descrizione dei vari dataset utilizzati in ambito url:

- *Malicious And Benign URLs* [50], collezione di 450000 url prelevati prevalentemente da PhishTank, di cui 100000 legittimi e 350000 di phishing
- *Phishing Site URLS* [51], collezione di 550000 url, di cui 400000 legittimi e 150000 di phishing
- *URLS Classification Dataset* [52], collezione di 150000 url di The Pudding, di cui 100000 legittimi e 50000 malevoli
- *URLhaus Dataset* [53], collezione di 65000 url malevoli appartenenti ai dataset di URLhaus

7.2 Script per l'analisi

Questa sezione è dedicata allo script usato per prelevare le informazioni utili dai dataset selezionati.

La struttura degli script è simile, se non per alcune differenze nella struttura delle email tra i vari dataset.

In seguito sono descritti i vari passi:

- la lettura del file *txt* o *csv* è fatta tramite la libreria *Pandas* [54] con la funzione *pandas.read_csv()*.

```
df = pd.read_csv(f'{directory}/{file}')
```

  L'unica eccezione è stata fatta per il Public Corpus, con una ricerca ricorsiva nelle cartelle

```
for (dirpath, dirnames, filenames) in walk(directory):
    for direc in dirnames:
        for (dp, dn, fn) in walk(f"{directory}/{direc}"):
            for file in fn:
                try:
                    email = build_email(dp, file)
```

- la struttura usata per contenere le informazioni ricalca esattamente quella delle email Outlook descritte nel capitolo 4. Questa parte è diversa per ogni dataset, in quanto ognuno ha una struttura diversa

```
"internetMessageId": parsed["Message-Id"][1:-1]
    if "Message-Id" in keys else ""

"subject": parsed["Subject"] if "Subject" in keys else ""

"importance": parsed["Precendence"] if "Preference" in keys else ""

"bodyContentType": parsed["Content-Type"]
    if "Content-Type" in keys else ""

"body": parsed.get_payload()
```

- in caso di allegati, la struttura dell'allegato ricalca esattamente quella degli allegati Outlook descritti nel capitolo 4. Questa parte è diversa per ogni dataset, in quanto ognuno ha una struttura diversa

7.3 Script per il caricamento in database

In seguito c'è la descrizione dello script che, una volta ricevuti i dati descritti nel paragrafo precedente, li elabora se li salva nel database. In questo script avviene sia il caricamento delle email nel database, che l'estrazione e persistenza delle feature.

Lo script esegue:

- elaborazione del messaggio

```
check_message(conn, cur, email)
```

- elaborazione dell'header

```
check_header(conn, cur, email)
```

- elaborazione del corpo

```
check_body(conn, cur, email)
```

- se esistono, elaborazione degli allegati

```
if email["hasAttachments"] is True:
    for att in email["attachments"]:
        check_attachments(conn, cur, att, email["emailId"])
```

- per ogni contatto nei campi *from*, *sender*, *toRecipents*, *ccRecipients* e *bccRecipients*, elaborazione del contatto (indirizzo e nome)

```
if email["senderName"] and email["senderAddress"]:
       check_contact(conn, email["emailId"], email["senderName"],
                     email["senderAddress"], cur)

if email["fromName"] and email["fromAddress"]:
    check_contact(conn, email["emailId"], email["fromName"],
                  email["fromAddress"], cur)
```

Per ogni sezione descritta sono definite due funzioni, una dedicata al caricamento delle informazioni nel database e la seconda dedicata all'estrazione delle feature.

```
body = email["body"]
text = BeautifulSoup(body, "html.parser").getText().lower()
cur.execute(insert_messages_bodies_sql,
            (mid, email["bodyContentType"], body, text)
)
conn.commit()
if body != "":
    insert_body_features(conn, cur, email, mid, body, text)
```

Capitolo 8

Modello per la classificazione

8.1 Introduzione

Questo capitolo è dedicato al modello realizzato per classificare le email.

Il modello è stato realizzato analizzando la distribuzione delle feature estratte e il loro impatto sia sulle email legittime che malevole. La stessa analisi è stata fatta anche per gli url, mentre per gli allegati è stato scelto un approccio differente, basato su richieste API a servizi open source esterni.

In seguito c'è la descrizione più nel dettaglio della scelta delle feature usate dal modello.

8.2 Prelievo dal database

Il primo punto da analizzare è il prelievo delle feature dal database, estratte in una fase precedente.

Lo script fa uso di *Pandas*, libreria Python adatta alla visualizzazione dei dati e *Psycopg2* [55], libreria Python per interfacciarsi con il database PostgreSQL

Il flusso è molto semplice:

1. apertura di una connessione con PostgreSQL usando Psycopg2

```
conn = psycopg2.connect(
        host=db_config["host"],
        port=db_config["port"],
        database=db_config["database"],
        user=db_config["user"],
        password=db_config["password"]
    )
```

2. esecuzione di una query SELECT, per ogni tabella usata

```
get_features_sql = """ SELECT * FROM features """

cur.execute(get_features_sql)
```

3. costruzione di un dataframe Pandas con le feature prelevate

```
df_subject_features = pd.DataFrame(
    cur.fetchall(),
    columns=columns["subject_features"]
).set_index("id").drop(columns="message_id")
```

8.3 Analisi dei dataframe

Una volta che i dataframe Pandas sono stati caricati, il secondo punto è stato l'analisi della distribuzione delle feature estratte. L'obiettivo è stato quello di individuare, per ogni feature estratta, un valore soglia come discriminante per la classificazione della risorsa come legittima o malevola.

Per le feature continue sono state analizzate alcune statistiche al fine di scegliere la/le soglie migliori. Queste statistiche includono il valore massimo, minimo e la media.

```
print(df_legit.subj_nr_words.min())
print(df_legit.subj_nr_words.max())
print(df_legit.subj_nr_words.mean())
```

Viene analizzata anche la distribuzione grafica delle feature al fine di avere un migliore visione d'insieme su di esse tramite l'uso delle librerie *Seaborn* [56] e *Matplotlib* [57].

```
seaborn.displot(df_fraud.subj_nr_words)
plt.show()
```

Per le feature booleane è stato sufficiente verificare il numero di elementi pari a 0/False.

```
df_legit[df_legit.from_found_in_db == 0].info()
df_fraud[df_fraud.from_found_in_db == 0].info()
```

La fase di analisi ha permesso la creazione di un elenco di feature efficienti, ovvero in grado di selezionare più email malevole che legittime. Nel corso del progetto, con l'aumentare delle feature e con l'aggiunta di dataset nuovi, questo elenco ha subito numerose variazioni.

8.3.1 Feature delle email

Feature eliminate

Nel corso del progetto molte delle feature considerate e descritte nel capitolo 5 sono state scartate in quanto poco efficaci.

Questo significa che l'inclusione della feature nella classificazione porta due risultati:

- la classificazione non subisce modifiche nel caso in cui la feature sia presente. La feature non è rilevante in alcun modo
- la classificazione peggiora nel caso in cui la feature sia presente. La feature favorisce le email malevole rispetto alle email legittime

Queste feature includono:

- la presenza dell'oggetto
- la lunghezza dell'oggetto
- la presenza di numeri o simboli nell'oggetto
- statistiche varie sul sender
- presenza del sender nel database
- lunghezza o numero di parole nel corpo
- numero di url nel corpo
- numero di allegati
- presenza di un charset

Feature selezionate

La tabella 8.1 elenca tutte le feature selezionate e applicate al modello finale per la classificazione delle email, con la relativa regola e la quantità di email malevole classificate come malevole per ogni email legittima classificata come malevola.

8.3.2 Feature degli url

Feature eliminate

Nel corso del progetto molte delle feature considerate e descritte nel capitolo 5 sono state scartate in quanto poco efficaci.

Questo significa che l'inclusione della feature nella classificazione porta due risultati:

- la classificazione non subisce modifiche nel caso in cui la feature sia presente. La feature non è rilevante in alcun modo
- la classificazione peggiora nel caso in cui la feature sia presente. La feature favorisce le email malevole rispetto alle email legittime

Queste feature includono:

- lunghezza dell'url
- lunghezza del path
- uso di url shortening
- uso di redirect
- uso di alcuni simboli, quali - e _

Feature selezionate

La tabella 8.2 elenca tutte le feature selezionate e applicate al modello finale per la classificazione degli url, con la relativa regola e la quantità di url malevoli classificate come malevole per ogni url legittimo classificato come malevolo.

8.4 Pesi delle feature

8.4.1 Introduzione

Il modello creato, nonostante funzioni anche con pesi unitari, richiede l'applicazione di pesi proporzionali all'importanza della feature nella classificazione. Una feature rilevante nella classificazione di una risorsa malevola non può avere la stessa importanza di una feature che non risulti influente sulla decisione.

Per questo motivo sono state definite due metodologie per l'assegnazione dei pesi alle feature:

- selezione manuale, assegnando dei pesi in base all'analisi effettuata sui dataframe di feature, basandosi su come la feature impatti sui dataframe
- selezione combinatoria, assegnando dei pesi utilizzando un metodo brute force, con uno script che analizzi le possibili permutazioni di un certo numero di pesi. Questa selezione è stata eseguita in due modalità:
  - parziale, usando Python, in quanto un'esplorazione completa di vari pesi applicati a tutte le feature avrebbe richiesto costi computazionali e temporali eccessivi
  - completa (eseguita in un secondo momento), usando Go, sfruttando le performance del linguaggio per eseguire un brute force completo delle possibili permutazioni dei pesi. Questi pesi sono quelli usati durante l'implementazione finale del modello

8.4.2 Selezione manuale

Pesi sulle feature delle email

Il tentativo di aggiunta dei pesi manualmente è stato fatto considerando l'impatto della feature sul dataframe, in particolare:

- se il rapporto 1:X è compreso tra 1:5 e 1:9.9, assegno un peso pari a 2
- se il rapporto 1:X è superiore a 1:10, assegno un peso pari a 3

Con queste regole sono stati aggiunti dei pesi a otto feature:

- subj_ratio_words, con peso 3
- from_nr_digits, con peso 3
- from_nr_non_alpha, con peso 3
- from_virusTotal_nr_harmless, con peso 3
- from_domain_ratio_url_otx, con peso 2
- recipients_found_in_db, con peso 2
- body_nr_href_links, con peso 2
- body_content_type_exist, con peso 2

Pesi sulle feature degli url

Il tentativo di aggiunta dei pesi manualmente è stato fatto considerando l'impatto della feature sul dataframe, in particolare

- se il rapporto 1:X è maggiore di 1:9, assegno un peso pari a 2
- per due feature specifiche ho assegnato un peso di 0.5, in quanto la presenza di una implica, quasi sempre, la presenza dell'altra

Con queste regole sono stati aggiunti dei pesi a cinque feature:

- url_queries_length, con peso 2
- url_nr_queries, con peso 2
- url_port, con peso 2
- url_path_not_found, con peso 0.5
- url_path_query_not_found, con peso 0.5

8.4.3 Selezione combinatoria parziale

Pesi sulle feature delle email

La selezione combinatoria si basa sul cercare tutte le possibili permutazioni di un dato vettore dei pesi.

Vista la complessità computazionale dell'operazione si è deciso di modificare due pesi alla volta e analizzare il risultato. Una volta trovata la posizione ottimale, questi vengono fissati, e viene iniziato un nuovo ciclo di ricerca.

Sono stati aggiunti dei pesi a quattro feature:

- subj_ratio_words, con peso 2
- from_nr_digits, con peso 3
- from_virusTotal_nr_harmless, con peso 3
- body_nr_sus_words, con peso 3

Pesi sulle feature degli url

La selezione combinatoria si basa sul cercare tutte le possibili permutazioni di un dato vettore dei pesi.

Vista la complessità computazionale dell'operazione si è deciso di modificare due pesi alla volta e analizzare il risultato. Una volta trovata la posizione ottimale, questi vengono fissati, e viene iniziato un nuovo ciclo di ricerca.

Sono stati aggiunti dei pesi a cinque feature:

- url_nr_netloc, con peso 2
- url_ssl, con peso 2
- url_not_found, con peso 2
- url_path_not_found, con peso 0.5
- url_path_query_not_found, con peso 0.5

8.4.4 Selezione combinatoria completa

Pesi sulle feature delle email

La selezione combinatoria si basa sul cercare tutte le possibili permutazioni di un dato vettore dei pesi.

Per questa selezione si è usato un programma scritto in Go in quanto lo stesso scritto in Python avrebbe richiesto un tempo eccessivo.

I pesi assegnati alle feature sono:

- subj_nr_words, con peso 0.5
- subj_ratio_words, con peso 1
- sender_addr_ret_path, con peso 1
- from_nr_digits, con peso 1
- from_nr_non_alpha, con peso 0.5
- from_virusTotal_nr_harmless, con peso 3
- from_domain_ratio_url_otx, con peso 0.5
- from_domain_ratio_dns_otx, con peso 0.5
- from_found_in_db, con peso 0.5
- recipients_found_in_db, con peso 1
- body_ratio_non_alpha, con peso 2
- body_subj_ratio, con peso 0.5

- body_nr_img_tags, con peso 0.5
- body_nr_a_tags, con peso 0.5
- body_nr_href_links, con peso 1
- body_nr_form_tags, con peso 0.5
- body_nr_sus_words , con peso 1
- body_ratio_sus_words, con peso 0.5
- body_content_type_exist, con peso 0.5

Pesi sulle feature degli url

La selezione combinatoria si basa sul cercare tutte le possibili permutazioni di un dato vettore dei pesi.

Per questa selezione si è usato un programma scritto in Go in quanto lo stesso scritto in Python avrebbe richiesto un tempo eccessivo.

I pesi assegnati alle feature sono:

- url_ratio_netloc, con peso 1
- url_queries_length, con peso 2
- url_nr_netloc, con peso 3
- url_nr_path_digits, con peso 1
- url_nr_queries, con peso 1
- url_path_level, con peso 2
- url_ssl, con peso 3
- url_port, con peso 2
- url_not_path_upper, con peso 1
- url_nr_percent, con peso 1
- url_nr_dot, con peso 1
- url_nr_at, con peso 1
- url_nr_double_slash, con peso 2
- url_not_bl, con peso 1
- url_not_found, con peso 2
- url_host_not_found, con peso 1
- url_path_not_found, con peso 1.5
- url_path_query_not_found, con peso 1.5
- url_host_ratio_url_otx, con peso 1

8.5 Tabelle

Tabella 8.1. Feature delle email.

Descrizione	*Regola*	*Rapporto*
numero parole dell'oggetto	if subj_nr_words > 15	1:3.35
rapporto numero parole e lunghezza oggetto	if subj_ratio_words > 0.35	1:13.64
equivalenza indirizzo sender e return path	if sender_addr_ret_path == 0	1:3.01
numero caratteri numerici nel from	if from_nr_digits > 0	1:11.08
numero caratteri non alfanumerici nel from	if from_nr_non_alpha < 2	1:519
numero di harmless associato al dominio del from, richiesta API di VirusTotal	if from_virusTotal_nr_harmless < 81	1:18.1
rapporto numero malware e url associati al dominio del from	if from_domain_ratio_url_otx > 1	1:5.49
rapporto numero malware e record dns associati al dominio del from	if from_domain_ratio_dns_otx > 0.2	1:3.29
associazione nome-indirizzo del from non si trova tra i contatti	if from_found_in_db == 0	1:3.25
numero associazioni nome-indirizzo in replyTo/cc/bcc trovati nei contatti	if recipients_found_in_db == 0	1:9.76
rapporto numero caratteri non alfanumerici e lunghezza corpo	if 0.05 > body_ratio_non_alpha > 0.3	1:4.45
rapporto lunghezza oggetto e corpo	if body_subj_ratio > 0.2	1:3.53
numero di tag *img* nel corpo	if body_nr_img_tags > 0	1:2.51
numero di tag *a* nel corpo	if body_nr_a_tags > 1	1:4.25
numero di href nel corpo	if body_nr_href_links > 1	1:5.20
numero di tag *form* nel corpo	if body_nr_form_tags > 0	1:1.91
numero parole sospette nel corpo	if body_nr_sus_words > 17	1:2.55
rapporto tra numero di parole sospette e numero di parole nel corpo	if body_ratio_sus_words > 0.1 o body_ratio_sus_words == -1	1:2.43
il contentType non esiste	if body_content_type_exist == 0	1:9.64

Tabella 8.2. Feature degli url.

Descrizione	*Regola*	*Rapporto*
rapporto lunghezza del network location e url	if url_ratio_netloc > 0.5	1:2.39
lunghezza della query	if url_queries_length > 30	1:9.68
numero di domini nel network location	if url_nr_netloc < 3	1:2.24
numero di caratteri numerici nel path	if url_nr_path_digits > 20	1:1.89
numero di argomenti nella query string	if url_nr_queries > 1	1:13.97
numero di livelli del path	if url_path_level > 6	1:1.33
non usa SSL/TLS	if url_ssl == 0	1:2.49
la porta non è 80 o 443	if url_port == 0	1:322
nel path compaiono almeno 2 caratteri maiuscoli di fila	if url_not_path_upper == 0	1:1.18
numero di % nell'url	if url_nr_percent > 0	1:1.27
numero di . nell'url	if url_nr_dot > 3	1:3.29
numero di @ nell'url	if url_nr_at > 0	1:5.15
numero di // nell'url	if url_nr_double_slash != 1 1:2.78	
url trovato in una blacklist	if url_not_bl == 0	solo malevoli
url trovato nel database di url malevoli	if url_not_found == 0	solo malevoli
hostname trovato del database di url malevoli	if url_host_not_found == 0	1:3.43
path trovato nel database di url malevoli	if url_path_not_found == 0	1:5.35
associazione path-query trovata nel database di url malevoli	if url_path_query_not_found == 0	1:5.82
rapporto numero malware e url associati all'hostname usano AlienVault	if url_host_ratio_url_otx > 0.3	1:3.87

Capitolo 9

Implementazione del modello

9.1 Introduzione

Questo capitolo è dedicato all'implementazione del modello sviluppato in questo progetto in due modalità:

- Uso di un metodo procedurale utilizzando una serie di costrutti if sulle varie feature scelte. Il sistema tiene traccia di un contatore *cnt*; ogni volta che una generica feature di una risorsa (email, url o allegato) risulta anomala, il contatore viene aumentato. Una feature è considerata anomala se risulta pari a 0/False, in caso di feature booleana, o esce da un certo range di valori, in caso di feature continua.
  Al termine della cascata di if, se il contatore è superiore ad un certo limite, la risorsa viene classificata come negativa.
- Uso di un metodo modellistico utilizzando un algoritmo della libreria *Scikit-Learn* [58]. Il modello riceve le feature in input, viene addestrato su queste e ritorna un classificatore utilizzabile.
  Non avendo un'esperienza a priori sull'argomento, ho scelto di considerare otto modelli della libreria.

9.2 Metodo procedurale

9.2.1 Classificazione delle email

La classificazione della email utilizza una variabile contatore *cnt*. La email viene esaminata da una serie di costrutti if, i quali, nel caso in cui la feature risulti anomala per qualche limite, aumentano il contatore.

Il modello aumenta il contatore secondo alcuni pesi assegnati alle feature, descritti nel capitolo 8, e segna l'errore in un dizionario che verrà utilizzato per generare un report da inviare all'utente su cui il tool è in esecuzione.

```
if email["subj_nr_words"] > 15:
    cnt += w[0]
    errors["subj_nr_words"] =
        f"Found a subject with {email['subj_nr_words']} words"
```

Al termine dell'analisi della email, il contatore viene controllato: se supera una certa soglia, 4 in questo caso, il sistema classifica la email come negativa.

```
if cnt > 4:
return 0, errors
```

9.2.2 Classificazione degli url

La classificazione degli url avviene utilizzando una variabile contatore *cnt*. L' url viene esaminato da una serie di costrutti if, i quali, nel caso in cui la feature risulti anomala per qualche limite, incrementano il contatore.

Il modello aumenta il contatore secondo alcuni pesi assegnati alle feature, descritti nel capitolo 8, e segna l'errore in un dizionario che verrà utilizzato per generare un report da inviare all'utente su cui il progetto è in esecuzione.

```
if url["url_ratio_netloc"] > 0.5:
    cnt += w[0]
    errors["url_ratio_netloc"] =
        f"The ratio len network location /
        len url is {url['url_ratio_netloc']}"
```

Al termine dell'analisi della email, il contatore viene controllato: se supera una certa soglia, 4 in questo caso, il sistema classifica l'url come negativo.

```
if cnt > 4:
return 0, errors
```

9.2.3 Classificazione dagli allegati

La classificazione degli allegati avviene utilizzando una variabile contatore *cnt*. L'allegato è analizzato da due servizi open source, VirusTotal e AlienVault OTX, tramite richieste API. Le richieste API sono descritte nel capitolo 3.

Se il sistema trova una certa condizione in uno dei report ricevuti dalle API, il sistema classifica l'allegato come dannoso e segna l'errore in un dizionario che verrà utilizzato per generare un report da inviare all'utente su cui il progetto è in esecuzione.

```
if 'data' in report.keys() and
report['data']['attributes']['stats']['malicious'] > 0:
    result = 0
    errors["malicious"] =
        f"{report['data']['attributes']['stats']['malicious']}
        vendors reported this file as malicious"
```

9.3 Metodi di Scikit-Learn

9.3.1 Metodi scelti

I modelli scelti sono i classici per la classificazione supervisionata:

- Support Vector Machines, insieme di algoritmi di classificazione basata sulla costruzione di un o più piani in uno spazio dimensionale infinito. Se consideriamo ogni email come rappresentabile in uno spazio geometrico, i piani sono costruiti affinché la distanza tra

gli input più vicini di classi diverse sia massimizzata. Questi algoritmi possono lavorare con kernel differenti, ovvero è possibile scegliere la funzione con la quale costruire i piani di separazione delle classi, come funzioni lineari, polinomiali, sigmoidali o gaussiane. Le implementazioni considerate sono:

  - C-Support Vector Classification, implementazione del Support Vector Machine basata sulla libreria *libsvm*
  - Linear Support Vector Classification, versione del SVC con un kernel lineare, adatto quando le classi per la classificazione sono due

- Decision Tree, algoritmo basato sulla costruzione di un albero i cui nodi sono rappresentati da regole condizionali ricavate dalle feature. Ogni nodo, quindi, è una scelta basata sul valore di una certa feature mentre ogni foglia è il risultato della classificazione condizionata ai nodi precedenti. Il punto caratteristico di questo classificatore è la semplicità nell'interpretazione del classificatore

- Metodi Ensemble, insieme di algoritmi basati sulla combinazione di multipli classificatori "deboli", in genere decision trees, per la costruzione di un classificatore "forte". Le implementazioni considerate sono:

  - Random Forest, basato su multipli Decision Tree, ognuno con un sottoinsieme casuale dei dati originali, per poi creare una media dei valori ottenuti
  - Gradient Boosting Classification Tree, basato sulla costruzione di modelli di classificazione sequenziali. L'output di un modello diventa l'input del prossimo, e così via fino all' ultimo
  - Histogram-based Gradient Boosting Classification Tree, variazione del Gradient Tree Boosting e bastato sulla discretizzazione delle variabili continue in input utilizzando solo un centinaio di valori. Adatto quando il dataset considerato è dell'ordine di decine di migliaia di elementi

- K Nearest Neighbors, algoritmo che sfrutta la distanza tra i punti in input per individuare delle zone in uno spazio dimensionale con alta concentrazione di essi. Le aree ricavate in questo modo avranno punti con caratteristiche simili e verranno definite come le classi in cui valutare le email ricevute

- Percettrone, algoritmo appartenente alla categoria delle Reti Neurali, basato sulla costruzione di reti complesse composte da collezioni di nodi, detti neuroni. Questi neuroni sono funzioni non lineari che lavorano sull'output dei neuroni precedenti, utilizzando dei pesi e un certo bias

9.3.2 Pre-processing

Prima di utilizzare un modello e cercarne i parametri migliori è opportuno trasformare i dati in input in una forma più adatta al modello.

La fase di preprocessing si basa su tre fasi:

1. eliminazione delle feature nulle dai dataframe, per questo pandas mette a disposizione la funzione *pandas.dropna(axis=1)*

2. separazione della variabile target dal dataframe e dividere il dataset che si vuole usare in dataset di addestramento e dataset di valutazione. Scikit-learn mette a disposizione una funzione apposita.

```
X_train, X_test, y_train, y_test =
    train_test_split(x, y, test_size=0.2, random_state=None)
```

3. standardizzazione delle feature di addestramento, eliminando la media e dividendo per la varianza. Scikit-Learn mette a disposizione una classe apposita.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Si nota come la funzione *fit_transform* sia stata applicata solo al dataset di addestramento per evitare data leakage

9.3.3 Selezione delle feature

In questo ambito potrebbe essere utile ridurre il numero di feature per aumentare la velocità del modello e, possibilmente, per aumentare le prestazioni su dataset di alte dimensioni.

Nel progetto sono stati esaminati tre casi:

- modelli senza feature selection, ovvero i modelli considerano tutte le feature estratte
- modelli con feature selection usando Scikit-Learn, ovvero eliminando tutte le feature con varianza pari a zero e selezionando solo un certo numero utilizzando un criterio di selezione.

  ```
  threshold = VarianceThreshold()
  X_train = threshold.fit_transform(X_train)
  X_test = threshold.transform(X_test)
  ```

  Nell'implementazione ho scelto di selezionare le venti feature migliori con la classe *SelectKBest*. La scelta delle feature migliori è stata fatta con il criterio *mutual_info_classif*

  ```
  kbest = SelectKBest(mutual_info_classif, k=20)
  X_train = kbest.fit_transform(X_train, y_train)
  X_test = kbest.transform(X_test)
  ```

- modelli con feature selection selezionando manualmente le feature desiderate. In questo caso le feature scelte per la classificazione sono le stesse delle feature usate nel modello custom

9.3.4 Scelta dei parametri

Modalità di scelta

Una parte fondamentale per l'uso corretto di un algoritmo di classificazione è la scelta dei parametri adeguati. Ogni categoria di algoritmi ha la propria selezione di parametri.

Per scegliere i parametri è stato fatto uso della classe *GridSearchCV* della libreria Scikit-Learn, in grado di selezionare un classificatore, una collezione di parametri con la lista di possibili valori e una strategia per la validazione. Questo fa ritornare un JSON con la migliore configurazione possibile chiave-valore per i parametri selezionati.

```
GridSearchCV(
    GradientBoostingClassifier(random_state=42),
    [
        {
            "loss": ["log_loss", "exponential"],
            "n_estimators": [100, 150],
            "criterion": ["friedman_mse", "squared_error"]
        }
    ],
    cv=5,
)
```

In quanto iterazione brute force la velocità della predizione con i modelli più lenti, quali SVC o reti neurali, è piuttosto lenta.

Parametri per le email

In seguito c'è la descrizione dei modelli per le email con i parametri generati da GridSearchCV. Bisogna notare che quasi tutti i modelli hanno un parametro *random_state=None*, in grado, dove è necessario, di generare o selezionare casualmente eventuali elementi interni al modello.

I modelli utilizzati sono:

- **SVC**(random_state=None, C=10, gamma="scale", kernel="rbf", shrinking=True)
  - C indica l'approssimazione del piano che divide le classi, valori piccoli significano un piano con maggiore distanza di separazione
  - kernel è il tipo di funzione a cui apparterrà il piano di separazione (lineare, polinomiale, esponenziale o sigmoide), in questo caso è esponenziale
  - gamma è un coefficiente per il kernel utilizzato
  - shrinking è un ottimizzazione del modello basato sull'esclusione di certi elementi. Riducendo gli elementi il modello è più veloce
- **LinearSVC**(random_state=None, dual=False, C=1, loss="squared_hinge", max_iter=1500, penalty="l1")
  - C indica l'approssimazione del piano che divide le classi, valori piccoli significano un piano con maggiore distanza di separazione
  - dual indica il tipo di problema di ottimizzazione da risolvere, un valore False è da preferire quando il numero di elementi da classificare è superiore al numero di feature
  - loss indica il tipo di funzione di penalizzazione
  - max_iter è il numero massimo di iterazione dell'algoritmo
  - penalty è la norma usata nella penalizzazione
- **DecisionTreeClassifier**(random_state=None, criterion="gini", max_depth=7, max_features=None, min_samples_leaf=2, min_samples_split=2, splitter="best")
  - criterion è la funzione per misurare la qualità della divisione con cui creare il nodo
  - max_depth è la massima profondità dell'albero costruito
  - max_features è il numero massimo di feature considerate, nel caso None tutte le feature sono considerate
  - min_samples_leaf è il numero di elementi da considerare in ogni nodo
  - min_samples_split è il numero di elementi da necessari per dividere un nodo
  - splitter è la strategia usata per la divisione del nodo
- **RandomForestClassifier**(random_state=None, criterion="gini", n_estimators=250, max_depth=15, max_features="sqrt", min_samples_leaf=1, min_samples_split=2)
  - criterion è la funzione per misurare la qualità della divisione con cui creare il nodo
  - max_depth è la profondità massima dell'albero costruito
  - max_features è il numero massimo di feature considerate, nel caso sqrt considera la radice quadrata del numero di feature
  - min_samples_leaf è il numero di elementi da considerare in ogni nodo
  - min_samples_split è il numero di elementi da necessari per dividere un nodo
  - n_estimators è il numero di alberi nell'algoritmo

- **GradientBoostingClassifier**(random_state=None, criterion="friedman_mse", loss="log_loss", min_samples_split=2, min_samples_leaf=2, max_features="sqrt", max_depth=6, n_estimators=250)
  - criterion è la funzione per misurare la qualità della divisione con cui creare il nodo
  - loss è la funzione usata per l'ottimizzazione
  - min_samples_leaf è il numero di elementi da considerare in ogni nodo
  - min_samples_split è il numero di elementi da necessari per dividere un nodo
  - max_depth è la profondità massima dell'albero costruito
  - max_features è il numero massimo di feature considerate, nel caso sqrt considera la radice delle feature
  - n_estimators è il numero di alberi nell'algoritmo
- **HistGradientBoostingClassifier**(random_state=None, loss="log_loss", max_leaf_nodes=31, max_iter=3000, max_depth=5, min_samples_leaf=30)
  - loss è la funzione usata per l'ottimizzazione
  - max_leaf_nodes è il numero massimo di foglie per ogni albero
  - max_iter è il numero massimo di alberi dell'algoritmo
  - max_depth è la profondità massima dell'albero costruito
  - min_samples_leaf è il numero minimo di elementi per foglia
- **KNeighborsClassifier**(algorithm="ball_tree", weights="distance", n_neighbors=4, leaf_size=10, p=1) algorithm è l'algoritmo usato per creare le zone
  - weights è la funzione per la determinazione del peso usata nella predizione, distance assegna il peso in base all'inverso della distanza
  - n_neighbors è il numero di zone da usare nella classificazione
  - leaf_size è la dimensione delle foglie dell'albero costruito nell'algoritmo
  - p è un parametro per la distanza, 1 significa distanza di Manhattan
- **MLPClassifier**(random_state=None, solver="adam", activation="tanh", max_iter=800, hidden_layer_sizes=(110,))
  - solver è l'ottimizzatore usato nell'algoritmo, adam è un ottimizzatore stocastico con ottime prestazioni in grandi dataset
  - activation è la funzione di attivazione per i layer nascosti, tanh è la tangente
  - max_iter indica il numero massimo di iterazioni dell'algoritmo
  - hidden_layer_sizes indica il numero di layer e di neuroni usati

La descrizione dei vari parametri non verrà ripetuta nel prossimo paragrafo.

Parametri per gli url

In seguito c'è la descrizione dei modelli per le email con i parametri generati da GridSearchCV. Bisogna notare che quasi tutti i modelli hanno un parametro random_state=None, in grado, dove è necessario, di generare o selezionare casualmente eventuali elementi interni al modello.

I modelli utilizzati sono:

- **SVC**(random_state=None, shrinking=True, kernel="rbf", C=100, gamma="scale")
- **LinearSVC**(random_state=None, dual=False, loss="squared_hinge", C=10, max_iter=400, penalty="l2")
- **DecisionTreeClassifier**(random_state=None, criterion="gini", max_depth=13, max_features=None, min_samples_leaf=1, min_samples_split=3, splitter="random")

- **RandomForestClassifier**(random_state=None, min_samples_split=2, max_features=None, criterion="entropy", n_estimators=200, max_depth=10, min_samples_leaf=2)
- **GradientBoostingClassifier**(random_state=None, loss="log_loss", min_samples_leaf=1, min_samples_split=2, criterion="squared_error", max_depth=5, max_features="sqrt")
- **HistGradientBoostingClassifier**(loss="log_loss", random_state=None, max_leaf_nodes=31, max_depth=5, max_iter=150, min_samples_leaf=25)
- **KNeighborsClassifier**(algorithm="auto", weights="distance", leaf_size=25, n_neighbors=6, p=1)
- **MLPClassifier**(random_state=None, activation="relu", solver="adam", hidden_layer_sizes=(110,), max_iter=1500)

9.4 Valutazione delle performance

9.4.1 Modalità

La valutazione delle performance dei vari modelli è stata effettuata utilizzando tre metriche:

- Accuracy, rappresenta la relazione tra le email correttamente classificate e la totalità delle email prese in considerazione

$$Accuracy = (TP + TN)/(TP + TN + FP + FN)$$

- Precision, rappresenta la relazione tra le email malevole correttamente classificate e la totalità delle email classificate come malevole

$$Precision = TP/(TP + FP)$$

- Recall, rappresenta la relazione tra le email malevole correttamente classificate e la totalità delle email correttamente classificate

$$Recall = TP/(TP + FN)$$

- F-measure, definito come media armonica di Precision e Recall

$$F - measure = 2 * Precision * Recall/(Precision + Recall)$$

I modelli di scikit sono stati generati considerando tutte le feature (NFS), con una selezione eseguita da scikit-learn (WFS) e con una selezione delle feature manuale usando quelle del modello custom (CFS).

Considerando i modelli di scikit-Learn, l'addestramento e verifica dei modelli di email è stato usato un dataset di quasi 28000 email (selezionando un limitato numero di email da ogni dataset per avere maggiore varietà), mentre per l'addestramento e verifica dei modelli degli url è stato usato un dataset di 100000 url (selezionando un limitato numero di url da ogni dataset per avere maggiore varietà)

9.4.2 Modello procedurale

Classificazione delle email

La classificazione delle email usando il modello procedurale ha fatto uso di un dataset con 27845 email, di cui 15954 legittime e 11891 malevole:

- Accuracy score: 94.28%
- Precision score: 93.43%
- Recall score: 93.15%
- F-Measure: 93.29%

Classificazione degli url

La classificazione degli url usando il modello procedurale ha fatto uso di un dataset con 426451 url, di cui 183199 legittimi e 243252 malevoli:

- Accuracy score: 90.46%
- Precision score: 92.60%
- Recall score: 90.50%
- F-Measure: 91.54%

9.4.3 Modelli di Scikit-Learn

Classificazione delle email

Per ogni ogni modello e tecnica di feature selection sono state ricavate accuracy, precision e recall. (Tab: 9.1)

Si può notare come gli algoritmi gli algoritmi ensemble abbiano ottenuto i risultati migliori:

- RandomForest senza feature selection:
  - Accuracy score: 99.44%
  - Precision score: 99.39%
  - Recall score: 99.63%
  - F-Measure: 99.51%
- HistGradientBoosting con feature selection "manuale":
  - Accuracy score: 99.52%
  - Precision score: 99.56%
  - Recall score: 99.59%
  - F-Measure: 99.57%

Classificazione degli url

Per ogni ogni modello e tecnica di feature selection sono state ricavate accuracy, precision e recall. (Tab: 9.2)

Si può notare come, anche per la classificazione degli url, gli algoritmi ensemble abbiano ottenuto i risultati migliori:

- RandomForest senza feature selection:
  - Accuracy score: 95.13%
  - Precision score: 93.92%
  - Recall score: 96.46%
  - F-Measure: 95.17%
  -
- HistGradientBoosting senza feature selection:
  - Accuracy score: 96.00%
  - Precision score: 94.97%
  - Recall score: 97.16%
  - F-Measure: 96.05%

Modelli dei related works

Questa sezione descrive brevemente i risultati di alcuni articoli con una metodologia simile a quella usata in questo progetto:

- [19] introduce una classificazione ibrida includendo, oltre ad un'analisi lessicale, un'analisi sugli header e url. Utilizzano multipli dataset, per un totale di 7000 email di phishing e 100 legittime. L'*accuracy* dell'implementazione proposta non scende al di sotto del 97%
- [20] include feature estratte dalle immagini e da informazioni contestuali alle feature estratte con un'analisi lessicale. Utilizzano un dataset di 5260 email, tra legittime e malevole, ottenendo *accuracy* del 96%, *precision* del 97% e *recall* del 95%
- [21] integra nella metodologia una nuova tecnica di feature selection, il Binary Search Feature Selection, e lo paragona con il Sequential Forward Feature Selection e l'uso senza feature selection. Utilizzano un dataset di 3428 email, di cui 1824 malevole e 1604 legittime, ottenendo *accuracy* pari a 97.41%, *precision* pari a 96.24%, *recall* pari a 99.67% e *F-Measure* del 97.41%
- [22] affrontano la classificazione considerano informazioni estratte dall'header e dagli url all'interno del messaggio con il Knowledge Discovery e Majority Voting. Le 10 configurazioni selezionate, utilizzando un dataset di 9298 email, di cui 4632 malevole e 4666 legittime, ottengono performance variabili con *precision* tra il 98% e 99% e con *error rate* pari a 1%
- [23] applica controlli sulla presenza di Javascript all'interno della email, così come eventi pop up e OnClick, e feature psicologiche, alle feature estratte usando header e url, senza usare una classificazione testuale. Utilizzando un dataset di 1000 email, 500 malevole 500 legittime, ottenendo *accuracy* del 95% e *precision* del 91.7%
- [24] includono delle feature ottenute analizzando il dominio del mittente, i domini degli url presenti nella email e presenza di keyword. Utilizzando un dataset di 2000 email, di cui 200 malevole 1800 legittime, ottengono *accuracy* del 99.7%, *precision* del 99.47%, *recall* del 97.5% e *F-Measure* del 98.45%.

Il modello analizzato in questo elaborato è un classificatore ibrido con analisi testuale, analisi dell'header, analisi degli url e analisi parziale degli allegati. Le analisi sono state fatte principalmente con NLP, analisi di dominio, controlli di equivalenza e analisi degli url.

9.5 Tabelle

Tabella 9.1. Performance dei modelli per le email.

Modello	*FS*	*Accuracy*	*Precision*	*Recall*
SVC	NFS	99.052	99.091	99.195
SVC	WFS	98.665	98.613	98.990
SVC	CFS	98.626	98.349	99.202
LinearSVC	NFS	95.512	95.961	96.028
LinearSVC	WFS	94.177	94.597	95.022
LinearSVC	CFS	94.660	94.532	95.918
DecisionTree	NFS	97.930	98.380	97.941
DecisionTree	WFS	97.620	98.057	97.684
DecisionTree	CFS	97.562	97.754	97.954
RandomForest	NFS	99.439	99.390	99.626
RandomForest	WFS	99.246	99.136	99.514
RandomForest	CFS	99.168	99.074	99.449
GradientBoosting	NFS	99.497	99.303	99.790
GradientBoosting	WFS	99.265	99.229	99.496
GradientBoosting	CFS	99.420	99.523	99.455
HistGradientBoosting	NFS	99.497	99.472	99.612
HistGradientBoosting	WFS	99.439	99.488	99.522
HistGradientBoosting	CFS	99.516	99.558	99.592
KNeighborsClassifier	NFS	99.052	98.792	99.513
KNeighborsClassifier	WFS	98.955	98.948	99.218
KNeighborsClassifier	CFS	98.414	97.981	99.203
MLPClassifier	NFS	98.975	98.829	99.342
MLPClassifier	WFS	98.897	98.757	99.271
MLPClassifier	CFS	98.975	98.893	99.271

Tabella 9.2. Performance dei modelli per gli url.

Modello	*FS*	*Accuracy*	*Precision*	*Recall*
SVC	NFS	95.035	94.083	96.201
SVC	WFS	94.230	92.167	96.749
SVC	CFS	93.925	92.652	95.406
LinearSVC	NFS	91.860	90.080	94.095
LinearSVC	WFS	89.845	86.126	95.202
LinearSVC	CFS	90.090	86.671	94.646
DecisionTree	NFS	94.150	93.813	94.544
DecisionTree	WFS	92.780	90.804	95.055
DecisionTree	CFS	92.970	90.814	95.526
RandomForest	NFS	95.130	93.916	96.464
RandomForest	WFS	93.585	91.235	96.535
RandomForest	CFS	93.640	91.450	96.128
GradientBoosting	NFS	95.815	94.660	97.066
GradientBoosting	WFS	93.870	91.861	96.131
GradientBoosting	CFS	93.345	92.488	94.477
HistGradientBoosting	NFS	96.005	94.969	97.161
HistGradientBoosting	WFS	94.235	92.334	96.382
HistGradientBoosting	CFS	93.880	92.872	95.173
KNeighborsClassifier	NFS	94.865	93.685	96.176
KNeighborsClassifier	WFS	93.860	92.820	95.004
KNeighborsClassifier	CFS	93.790	93.503	94.066
MLPClassifier	NFS	95.295	94.267	96.523
MLPClassifier	WFS	94.150	92.276	96.245
MLPClassifier	CFS	94.115	92.318	96.227

Capitolo 10

Implementazione del tool

10.1 Introduzione

Questo prototipo raccoglie e implementa tutte le funzionalità sviluppate e discusse nell'elaborato.

Il compito di questo prototipo scritto in Python è quello di collegarsi con un account Azure Active Directory, interfacciarsi con i vari utenti e classificare le email che arrivano nella loro casella postale. Se l'email è classificata negativamente, viene generato un report ed inviato tramite email all'utente.

Il modello utilizzato per la classificazione è scelto in fase di istanziazione della classe principale. La scelta ricade tra il modello custom o un modello addestrato della libreria Scikit-Learn.

Il prototipo considera due casi "reali", ovvero la multiutenza dell'account Active Directory e l'esecuzione del prototipo in background, risultando trasparente all'utente che lo usa.

I pattern di programmazione più rilevanti per lo sviluppo sono il classico Object Oriented Programming e la programmazione parallela tramite l'uso di threads e meccanismi di sincronizzazione. Questi permettono una maggiore flessibilità di scrittura e la possibilità di eseguire azioni multiple nello stesso arco temporale.

10.2 Descrizione

Il tool esegue un setup iniziale e resta in attesa dei comandi utente.

Il setup è eseguito dalla funzione *__init__()* della classe EMProtection. A questa deve essere associata un file di configurazione chiamato *general_config.json* o una serie di parametri da linea di comando.

Le descrizioni dettagliate del file *general_config.json* e della funzione *__init__()* sono lasciate ai paragrafi successivi.

L'utente può eseguire quattro comandi:

- **start**, comando per iniziare l'esecuzione del tool di classificazione. La funzione associata genera un thread per ogni utente nell'account Active Directory. Ogni thread preleva l'ultimo delta token disponibile ed esegue la funzione *__get_delta_emails()*, funzione principale del tool e descritta nei paragrafi successivi. Periodicamente viene eseguito un controllo su un flag, variabile di terminazione del thread.

- **stop**, comando per interrompere l'esecuzione dei vari thread lanciati con il comando *start*. La funzione associata utilizza una variabile condivisa tra tutti i thread per segnalare la loro terminazione. Il tool resta in esecuzione e in attesa di nuovi comandi.

Figura 10.1. Setup del tool.

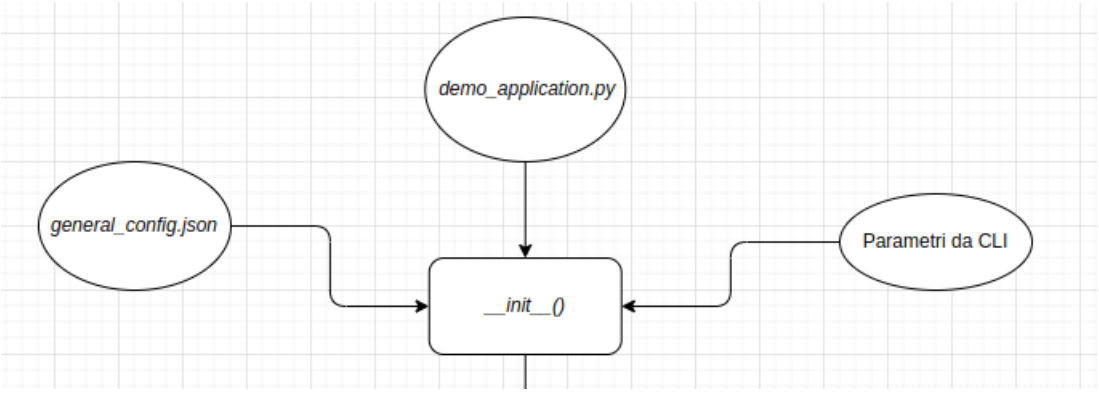


Figura 10.2. Esecuzione live.

- **users**, comando per l'esecuzione della funzione *check_users()* per aggiornare il file di configurazione *ms_users.json*. Le descrizioni dettagliate del file *ms_users.json* e della funzione *check_users()* sono lasciate ai paragrafi successivi.

- **exit**, comando per terminare l'esecuzione del tool. La funzione associata utilizza un flag di uscita per segnalare la terminazione del processo principale. Nel caso in cui il sistema abbia ancora dei thread in esecuzione, questi vengono terminati agendo sulla variabile condivisa del comando *stop*.

10.3 File di configurazione

Il progetto usa quattro file di configurazione con tutti i dati necessari per collegarsi con Microsoft Identity, con il database PostgreSQL, per fornire i path dei file di log e dei modelli, e per fornire la lista degli utenti da seguire.

10.3.1 *ms_config.json*

Questo file JSON contiene tutti i dati per collegarsi con Microsoft Identity e ricevere un token per l'autenticazione delle richieste API.

```
{
    "client_id": "6a4d4f1d-a456-4067-8e50-929f89227627",
    "secret": "secret",
    "scope": [
        "https://graph.microsoft.com/.default"
    ],
    "authority": "https://login.microsoftonline.com/
                  ebfdaae9-492c-4a89-9ed3-613380b24d25",
    "api_url": "https://graph.microsoft.com/v1.0/users"
}
```

Nel file ci sono cinque campi:

- *client_id*, id dell'applicativo registrata su Microsoft Azure
- *secret*, stringa segreta associata all'applicazione da usare durante la fase di autenticazione del prototipo
- *scope*, indica il set di API utilizzabili dall'utente
- *authority*, url usato da MSAL per prelevare il token di autenticazione
- *email_url*, url di base per accedere alle API

10.3.2 *ms_users.json*

Questo file JSON contiene i dati di tutti gli utenti a cui verrà applicato il prototipo.

Il file contiene tante coppie chiave - valore quanti sono gli utenti da seguire. La chiave è l'indirizzo email dell'utente mentre il valore è un JSON contenente informazioni necessarie per il corretto funzionamento del prototipo.

```
"S287515@2hmdk1.onmicrosoft.com": {

    "name": "SIMONE FASOLIS",

    "inbox_id": "AAMkADk5MzAwY2I4LTQ2YzktNDYzMC1hM2JkLTlmMj
                k2MTNiYjFjMQAuAAAAAABIzsoi0gqhQ6Ysvzgws5Nt
                AQDHAwgx8AUdSKUG8NkDbyPcAAAAAAEMAAA=",

    "report_id": "AAMkADk5MzAwY2I4LTQ2YzktNDYzMC1hM2JkLTlm
                Mjk2MTNiYjFjMQAuAAAAAABIzsoi0gqhQ6Ysvzgw
                s5NtAQDHAwgx8AUdSKUG8NkDbyPcAAAmQW5VAAA=",

    "delta_token": "deltatoken=LztZwWjo5IivWBhyxw5rAKku
```

```
                    tS5z0Mt_rHjlTXGKCPDXb3owzSsfXwP1xvPukpf
                    tXQQGUKcX_JwXD6CQmQWisEYCcVQ8ZaLkxnbJT
                    VS1CtY.PtOLU2kMuw1dBW-BZoNNBxDLNuwvF3RT
                    QkBHO2T20f4"
}
```

I dati contenuti in ogni JSON interno (associati a ogni indirizzo email) sono:

- *name*, è il nome associato all'indirizzo email considerato
- *inbox_id*, è l'identificativo della casella postale in cui verranno analizzate le email
- *report_id*, è l'identificativo di una cartella Reports in cui verranno inviati gli eventuali report di classificazione negativa
- *delta_token*, è il token utilizzato del prototipo per tenere traccia dell'ultima email analizzata

10.3.3 *db_configuration.json*

Questo file JSON contiene i dati necessari per la connessione con un'istanza PostgreSQL da utilizzare per salvare le email nel database. Per il progetto l'istanza in questione si trova in un container Docker in locale.

```
{
    "host": "127.0.0.1",
    "port": 5432,
    "database": "postgres",
    "user": "user",
    "password": "password"
}
```

Nel file ci sono cinque campi:

- *host*, hostname dell'istanza
- *port*, porta dell'hostname a cui interfacciarsi
- *database*, nome del database contenente tutti gli schemi utilizzati
- *user*, username dell'account associato all'istanza
- *password*, password dell'account associata all'istanza

10.3.4 *general_config.json*

Questo file JSON contiene tutti i path a vari file utilizzati da prototipo.

Nel file ci sono sei campi:

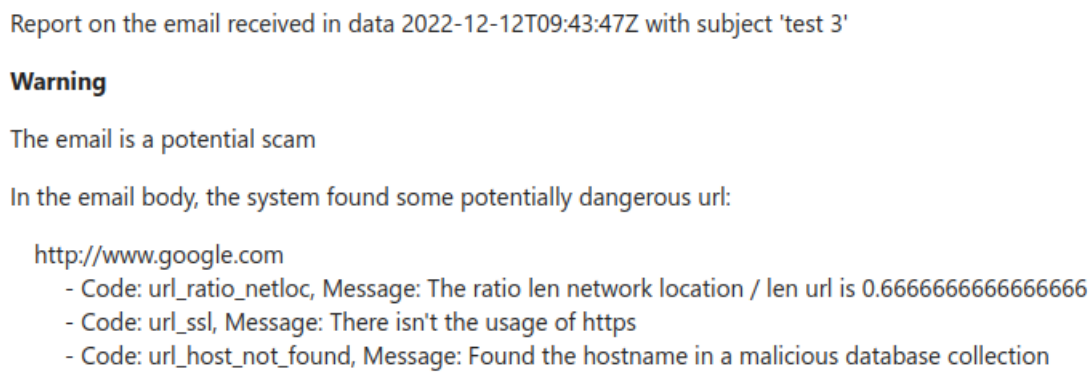
- *log_file*, path al file di log usato dal prototipo per registrare tutte le operazioni eseguite
- *ms_config_file*, path al file di configurazione per la connessione con Microsoft Identity
- *ms_users_file*, path per il file contenente tutti gli utenti a cui applicare il prototipo
- *db_config_file*, path per il file di configurazione utilizzato per la connessione con il l'istanza PostgreSQL
- *mod_email_file*, path al modello scikit usato per la classificazione di email. Nota che stringa vuota indica che verrà utilizzato il modello custom per la classificazione di email
- *mod_url_file*, path al modello scikit usato per la classificazione di url. Nota che stringa vuota indica che verrà utilizzato il modello custom per la classificazione di url

10.4 Report di classificazione

Come già accennato più volte, il prototipo invia un report all'utente via email in caso di classificazione negativa, nella cartella Report della casella postale dell'utente.

Il report contiene la lista, per ogni risorsa classificata, di tutte le feature che non hanno superato la relativa regola e fatto aumentatare il contatore.

Figura 10.3. esempio report.



Il formato del file cambia in base all' oggetto che il sistema ha classificato come negativo: il corpo e/o l'header della email, eventuali url o eventuali allegati.

10.5 Classe EPException

Questa è una semplice classe wrapper per ogni tipo di eccezione incontrata durante il processo.

Il lancio di un'eccezione implica la realizzazione di un istanza EPException e la scrittura dell'errore nel file di log.

10.6 File *demo_utilities.py*

Questo file contiene tutte le funzioni di supporto alla classe principale. In particolare contiene:

- tutte le API key usate nelle varie richieste API verso servizi di terze parti
- tutte le query SQL usate con il database PostgreSQL
- le funzioni per il filtraggio della email da Outlook e la costruzione di una struttura più semplice usata per la classificazione effettiva

  ```
  def define_ms_date_item(email)
  ```

- le funzioni che estraggono le informazioni dalla email e le salvano nel database

  ```
  def check_contacts(conn, cur, external_id, name, address, recipient_type=None)
  def check_malicious_url(conn, cur, url, lock)
  ```

- le funzioni che estraggono le feature dalle email / url

  ```
  def extract_headers_features(cur, email, message_id)
  def extract_body_features(cur,email, body, text, subj_length, subj_words)
  def extract_url_features(conn, cur, url)
  def extract_contact_features(conn, cur, email. address, name)
  ```

10.7 Classe EMProtection

10.7.1 *__init__()*

__init__ è la funzione iniziale della classe. Questa è responsabile del setup di tutte le connessioni e servizi usati nel corso dell'esecuzione, oltre a definire molte delle variabili usate, tra cui i Lock sulle variabili condivise dai threads.

Vengono definiti:

1. il file di log per registrare tutte le azione effettuate

```
logging.basicConfig(
    filename=log_file, filemode='a', level=logging.INFO,
    format='%(asctime)s - %(name) - %(levelname) - %(message)s',
    datefmt='%m/%d/%Y %I:%M:%S %p')
```

2. la connessione con PostgreSQL

```
self.db_config = json.load(open(db_config_file))
self.conn = psycopg2.connect(
    host=self.db_config["host"],
    port=self.db_config["port"],
    database=self.db_config["database"],
    user=self.db_config["user"],
    password=self.db_config["password"]
)
```

3. i modelli da usare per la classificazione (email e url)

```
if mod_email_file != "":
    logging.info(f"Using email model from: {mod_email_file}")
    self.email_model = joblib.load(mod_email_file)
```

4. la connessione con MSAL, usando una *msal.ConfidentialClientApplication*

```
self.app = msal.ConfidentialClientApplication(
        client_id,
        authority=tenant_url,
        client_credential=secret
    )
```

5. il prelievo di un token di autenticazione dalla cache o da Microsoft Identity

```
self.result = self.app.acquire_token_silent(scopes, account=None)
if not self.result:
    self.result = self.__refresh_ms_token()


scopes = self.ms_config["scope"]
return self.app.acquire_token_for_client(scopes)
```

10.7.2 *__get_delta_emails()*

Questa è la funzione principale del prototipo.

La funzione è responsabile per:

1. acquisizione di un delta token dal file di configurazione o prelievo dell'ultimo token disponibile in caso di assenza dal file

```
if delta_token == "":
    self.__get_last_delta_token(client_email, inbox_id)
    delta_token = self.ms_users[client_email]["delta_token"]
```

2. prelievo delle email, header e allegati, se presenti, usando le API già discusse
3. classificazione della risorsa

```
 email_result, urls_result,
 atch_result = self.__classify_email(item, client_email)
```

4. inserimento degli url malevoli in database usando un daemon thread

```
x = threading.Thread(
    target=check_malicious_url,
    args=(self.conn, self.conn.cursor(), url, self.lock_db),
    daemon=True
)
x.start()
```

5. invio del report all'utente

```
self.__report_email(
    email, client_email, report_id,
    email_errors=email_result["errors"],
    urls_errors=negative_urls,
    atch_errors=negative_atch
)
```

6. aggiornamento del delta token

```
delta_link_url = emails["@odata.deltaLink"]
data = requests.get(
    delta_link_url,
    headers={'Authorization': 'Bearer ' + token}
).json()
delta_link_url = data["@odata.deltaLink"]
```

10.7.3 *__classify_email()*

La funzione è responsabile per l'estrazione delle feature della email, il salvataggio della email in database e la classificazione.

Nella funzione è possibile trovare:

1. l'estrazione delle varie feature (contact, headers, body and url)

```
df = extract_headers_features(cur, email, mid)
email_to_classify = pd.concat([df, email_to_classify], axis=1)
```

2. la persistenza dei dati in database

```
try:
    logging.info(f"Save a new email with id  {email['emailId']}")
    with self.lock_db:
        self.__save_email_in_db(cur, email)
```

3. la classificazione dell'email o dell'url, usando uno dei vari modelli

```
if self.email_model is None:
    email_res, email_errors = email_classification(
        email_to_classify.reset_index().to_dict('records')[0]
    )
else:
    email_res = self.email_model.predict(
        email_to_classify.to_numpy()
    )[0]
    email_errors = {}
```

4. la classificazione degli allegati

```
digest = hashes.Hash(hashes.SHA256())
digest.update(str.encode(attachment["contentBytes"]))
content_hash = digest.finalize().hex()
atch_result, atch_errors = atch_classification(
                                attachment, content_hash
                            )
```

10.7.4 *__save_email_in_db()*

Questa funzione è responsabile della persistenza della email nel database.

Vengono salvati:

- il messaggio

```
cur.execute(
    insert_messages_sql,
    (email["emailId"], email["internetMessageId"],
    email["subject"])
)
```

- l'header

```
cur.execute(
    insert_headers_sql,
    (mid, email["headerFromName"], email["headerFromAddress"],
    email["headerMessageId"], email["headerSubject"],
```

```
            email["headerSubject"], email["headerContentType"],
            email["headerDate"], email["SPF"], email["DKIM"],
            email["DMARC"], email["COMPAUTH"], email["CIP"],
            email["LANG"], email["IPV"], email["headerReturnPath"])
        )
```

- il corpo

```
        cur.execute(
            insert_messages_bodies_sql,
            (mid, email["bodyContentType"], body, text)
        )
```

- gli allegati

```
        cur.execute(
            insert_attachments_sql,
            (attachment["id"], mid, attachment["name"],
            attachment["name"], attachment["contentType"],
            attachment["size"], content_hash.hex())
        )
```

- i contatti da *sender, from, toRecipients, ccRecipients, bccRecipients*

```
        cur.execute(insert_contacts_sql, (address,))
        cur.execute(insert_contacts_names_sql, (name, cid))
        cur.execute(insert_messages_senders_sql, (mid, nid))
        cur.execute(
            insert_messages_recipients_sql,
            (mid, nid, recipient_type)
        )
```

10.7.5 *__report_email()*

Questa funzione è responsabile dell'invio del report all'utente sui risultati della classificazione.

La funzione raccogli i messaggi di errore generati durante la classificazione e li inserisce in un messaggio testuale.

```
em_str += f"<strong>Warning</strong><br>"
em_str += f"<p>The email is a potential scam"

if email_errors is not None:
    for key, value in email_errors.items():
        em_str += f"<br> - Code: {key}, Message: {value}"
em_str += "</p>"
```

Il messaggio generato diventa il *content* del corpo di una email e viene inviato all'utente nella cartella *Reports*.

```
requests.post(
    f"{base_url}/{client_email}/mailFolders/{report_id}/messages",
    headers={'Authorization': 'Bearer ' + token},
    json=message
)
```

10.7.6 *__get_live_emails()*

Questa funzione genera un thread per ogni utente nel file di configurazione.

La funzione utilizzata dal thread è la *__get_delta_emails*. La funzione *__add_user* inserisce l'utente nella lista contatti del database, nel caso in cui non fosse ancora stato inserito.

```
for client_email in self.ms_users:
    self.__add_user(client_email, self.ms_users[client_email]["name"])
    x = threading.Thread(
        target=func,
        args=(client_email, self.ms_users[client_email]["inbox_id"])
    )
    self.threads.append(x)
    x.start()
```

10.7.7 *check_users*

La *check_user* è una funzione di utilità per popolare automaticamente il file JSON contenente la lista di utenti dell'account Active Directory.

La funzione esegue, per ogni utente dell'account:

1. persistenza del nome

```
user_json["name"] = user["displayName"]
```

2. persistenza dell'id della cartella *Inbox*

```
inboxes = requests.get(
            f"{email_url}/{user['mail']}/mailFolders",
            headers={'Authorization': 'Bearer ' + token}
        ).json()
try:
    user_json['inbox_id']
        = list(
            filter(
                lambda x: x["displayName"] == "Inbox",
                inboxes["value"]
                )
            )[0]["id"]
```

3. persistenza dell'id della cartella *Reports*

```
x = requests.post(
            f"{email_url}/{user['mail']}/mailFolders",
            headers={'Authorization': 'Bearer ' + token},
            json={"displayName": "Reports"}
)
if x.status_code == 201:
    response = json.loads(x.content)
    user_json["report_id"] = response['id']
```

4. salvataggio dei dati nel file di configurazione

```
with self.lock_config:
   self.ms_users[user["mail"]] = user_json
   with open(self.ms_users_file, 'w') as fout:
       fout.write(json.dumps(self.ms_users, indent=4))
```

10.8 Prestazioni

L'ultima sezione del capitolo è dedicata all'influenza del tool sulle prestazioni del dispositivo su cui è utilizzato.

Il sistema è stato testato su un dispositivo Manjaro, basato su Arch Linux, versione 6.0.11-1 x86_64, con un processore Intel Core i7-1165G7 a 2.80 GHz e 8 GB di memoria DRAM DDR4 a 3200 MT/s. Per registrare le prestazioni del tool è stato fatto uso del tool *htop* e dei file in */proc/*, in particolare */proc/loadavg*.

Il tool è stato eseguito in background sul sistema simulando l'esecuzione su diciassette utenti, ovvero su tutti gli utenti nella Active Directory.
Considerando un singolo thread in esecuzione in background, è stato registrato un aumento nel consumo medio della CPU pari al 1% (il consumo massimo registrato è pari al 2.1%), considerando il Load Average dell'ultimo, ultimi cinque e ultimi quindici minuti. Inoltre, è stato registrato un aumento nel consumo di memoria, da parte del tool, pari al 1.5%, circa 120 MB

Capitolo 11

Conclusioni e lavori futuri

L'obiettivo principale del progetto presentato in questo elaborato è l'analisi di un modello per la classificazione di email potenzialmente pericolose, come scam, phishing o spam.

L'analisi è stata effettuata considerando numerosi aspetti di una mail, quali il messaggio ricevuto, il soggetto, il mittente, i possibili url e i possibili allegati.
Il modello sviluppato ha prestazioni paragonabili a modelli molto più complessi grazie a quattro caratteristiche inedite o poco comuni tra i modelli di classificazione simili (sia per metodologia che per campi analizzati):

- uso del database PostgreSQL sottostante per effettuare un'analisi testuale, rilevante solo nel caso di email di spam, al fine di ridurre il numero di feature analizzate (aumentando la velocità di classificazione) senza ridurre l'accuratezza della classificazione. Per questo è stato fatto uso delle strutture *tsvector* e *tsquery* del database.
- correlazione tra mittente, destinatari e return path, analizzando la loro relazione e presenza nei database. L'analisi dei contatti trovati nell'header della email è una delle metodologie più efficaci per l'analisi di email di phishing
- analisi dei domini degli indirizzi email nell'header e degli url nel messaggio utilizzando alcuni servizi esterni. L'analisi dei domini è una delle metodologie più efficaci per la classificazione di email di phishing
- analisi degli url trovati nel messaggio indipendente dall'analisi dalla email. Il modello definito è una combinazione di due modelli separati per la classificazione rispettiva di email e url

Sono state definite due implementazioni principali del modello realizzato:

- implementazione procedurale analizzando ogni feature considerata in modo indipendente. Considerando un dataset di oltre 27000 email e 420000 url, il modello ha classificato correttamente il 94.3% delle email e il 90.5% degli url.
- implementazione modellizzata con un algoritmi più avanzato della libreria scikit-learn. Considerando un dataset di addestramento e validazione di oltre 27000 email e 100000 url, i modelli Random Forest e Histogram Gradient Boosting hanno classificato correttamente il 99.4% / 99.5 delle email e il 95.1% / 96.0% degli url

Inoltre, è stato definito un tool scritto in Python con l'implementazione del modello. Questo si interfaccia con un account Active Directory, classifica le email associate agli utenti dell'account e genera dei report che invia agli utenti associati in caso di classificazione negativa.

Per quanto riguarda il tool, il progetto presentato è un prototipo e, in quanto tale, non è ancora applicabile a casi reali. Allo stato attuale, il tool utilizza, su un dispositivo Linux, una quantità di RAM pari al 1.5% e di CPU mediamente pari al 1%.

Restano diversi punti da ottimizzare, le direzioni percorribili possono riassumersi in:

- ottimizzazione del modello con l'implementazione di nuove features, come un'analisi approfondita delle immagini
- ottimizzazione delle feature considerate per minimizzare il data drift, ovvero l'evoluzione della struttura delle mail nel corso del tempo
- ottimizzazione dell'implementazione del modello, usando un algoritmo più complesso, come una rete neurale correttamente parametrizzata

Una volta che il sistema sia stato reso più efficiente andrebbe inserito in una realtà aziendale e questo potrebbe tradursi in:

- implementazione relativa all'aggiornamento delle tabelle di supporto, così che tutti i valori utilizzati per i domini, hostname, file e url siano costantemente ottimali
- inserzione del sistema a livello di mail server, affinchè l'email venga classificata prima che arrivi nella casella postale dell'utente
- rielaborazione del tool in un servizio cloud esponendo le proprie funzionalità tramite API

Per concludere, i risultati ottenuti con la libreria *scikit-learn* sono in linea con gli studi più recenti sullo stesso argomento e con simili feature considerate, come l'analisi di [23] o di [24]. Il progetto ha, quindi, i requisiti necessari per proseguire con lo sviluppo.

Bibliografia

[1] How Many Email Users Are There? https://99firms.com/blog/how-many-email-users-are-there/#gref

[2] Number of e-mail users worldwide from 2017 to 2025 https://www.statista.com/statistics/255080/number-of-e-mail-users-worldwide/

[3] Phishing Insights 2021 https://news.sophos.com/en-us/2021/08/26/phishing-insights-2021/

[4] Spam Statistics https://99firms.com/blog/spam-statistics/#gref

[5] Average daily spam volume worldwide from October 2020 to September 2021 https://www.statista.com/statistics/1270424/daily-spam-volume-global/

[6] Phishing Activity Trend Reports https://apwg.org/trendsreports/

[7] Spam and phishing in 2021 https://securelist.com/spam-and-phishing-in-2021/105713/

[8] Reported Phishing Attacks Reach an All-Time High https://www.itgovernance.eu/blog/en/reported-phishing-attacks-reach-an-all-time-high

[9] Microsoft Exchange Online Protection https://learn.microsoft.com/en-us/microsoft-365/security/office-365-security/exchange-online-protection-overview?view=o365-worldwide

[10] A Multi-Layer Architecture for Spam-Detection System https://www.researchgate.net/publication/269231005_A_Multi-Layer_Architecture_for_Spam-Detection_System

[11] Z. S. Torabi, M. H. Nadimi-Shahraki, A. Nabiollahi, "Efficient Support Vector Machines for Spam Detection: A Survey", International Journal of Computer Science and Information Security, Vol. 13 , No. 1, January 2015 , pp. 11-28

[12] A. Bhowmick, S. H. Hazarika, "Machine Learning for E-mail Spam Filtering: Review,Techniques and Trends", June 2016, DOI 10.48550/arXiv.1606.01042

[13] B. Yu, Z. Xu, "A comparative study for content-based dynamic spam classification using four machine learning algorithms", Knowledge-Based Systems, Vol. 21, No. 4, May 2008, pp. 355-362, DOI 10.1016/j.knosys.2008.01.001

[14] R. Ravi, A. A. Shillare, P. P. Bhoir, K. S. Charumathi, "URL based Email Phishing Detection Application", International Research Journal of Engineering and Technology, Vol. 8, No. 4, April 2021, pp. 355-360

[15] A. Joshi, L. Lloyd, P. Westin, S. Seethapathy, "Using Lexical Features for Malicious URL Detection - A Machine Learning Approach", October 2019, DOI 10.48550/arXiv.1910.06277

[16] O. K. Sahingoz, E. Buber, O. Demir, B. Diri, "Machine learning based phishing detection from URLs", Expert Systems with Applications, Vol. 117, January 2019 pp. 345-357

[17] K. S. Adewole, A. Akintola, S. A. Salihu, N. Faruk, "Hybrid Rule-Based Model for Phishing URLs Detection", Emerging Technologies in Computing, pp. 119-135 July 2019 DOI 10.1007/978-3-030-23943-5_9

[18] K. Haynes, H. Shirazi, I. Ray, "Lightweight URL-based phishing detection using natural language processing transformers for mobile devices", Procedia Computer Science, Vol. 191 May 2021, DOI 10.1016/j.procs.2021.07.040

[19] R. Verna, N. Shashidhar, N. Hossain, "Detecting Phishing Emails the Natural Language Way", ESORICS 2012: Computer Security, September 2012, pp. 824-841, DOI 10.1007/978-3-642-33167-1_47

[20] E. Shyni, S. Sarju, S. Swamynathan, "A Multi-Classifier Based Prediction Model for Phishing Emails Detection Using Topic Modelling, Named Entity Recognition and Image Processing", Circuits and Systems, Vol. 7, No. 9 January 2016, pp. 3507-2520, DOI 10.4236/cs.2016.79217

[21] G. Sonowal, "Phishing Email Detection Based on Binary Search Feature Selection", SN Computer Science, Vol. 1, No. 191, June, 2020, DOI 10.1007/s42979-020-00194-z
[22] Y. A. Yaseen, M. Qasaimeh, "Email Fraud Attack Detection Using Hybrid Machine Learning Approach", Recent Patents on Computer Science, Vol. 12, No. 5, June 2019, DOI 10.2174/2213275912666190617162707
[23] Z. Yang, C. Qiao, W. Kan, J. Qiu, "Phishing Email Detection Based on Hybrid Features", IOP Conference Series: Earth and Environmental Science, Vol. 252, No. 4, July 2019, DOI 10.1088/1755-1315/252/4/042051
[24] A. Akinyelu, A. Adewumi "Classification of Phishing Email Using Random Forest Machine Learning Technique" Journal of Applied Mathematics, April 2014, DOI 10.1155/2014/425731
[25] F. Salahdine, Z. El Mrabet, N. Kaabouch, "Phishing Attacks Detection – A Machine Learning-Based Approach", 2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, December 2021, DOI 10.48550/arXiv.2201.10752
[26] M. H. Alshira'H, M Al-Fawa'reh, "Detecting Phishing URLs using Machine Learning & Lexical Feature-based Analysis", International Journal of Advanced Trends in Computer Science and Engineering, Vol. 9, No. 4, September 2020, pp. 5828-5837, DOI 10.30534/ijatcse/2020/242942020
[27] R. Basnet, A. H. Sung, Q. Liu, "Rule-Based Phishing Attack Detection", April 2012,
[28] S. Salloum, T. Gaber, S. Vadera, K. Shaalan "Phishing Email Detection Using Natural Language Processing Techniques: A Literature Survey", Procedia Computer Science, Vol. 189, 2021, pp. 19-28, DOI 10.1016/j.procs.2021.05.077
[29] F. R. Imadudin, D. T. Murdiansyah, Adiwijaya "Implementation of Naïve Bayes and Gini Index for Spam Email Classification", Computational and Simulation, Vol. 6, No. 1, April 2021,
[30] H. Zuhair, A. Selemat, M. Salleh, "Feature selection for phishing detection: A review of research", International Journal of Intelligent Systems Technologies and Applications, Vol. 15, No. 2, DOI 10.1504/IJISTA.2016.076495
[31] VirusTotal https://www.virustotal.com/gui/home/search
[32] AlienVault OTX https://otx.alienvault.com/dashboard/new
[33] URLhaus https://urlhaus.abuse.ch/
[34] urlscan.io https://urlscan.io/
[35] Microsoft Authentication Library (MSAL) for Python https://github.com/AzureAD/microsoft-authentication-library-for-python
[36] Microsoft Graph https://learn.microsoft.com/en-us/graph/overview
[37] VirusTotal API v3 Overview https://developers.virustotal.com/reference/overview
[38] DirectConnect API https://otx.alienvault.com/apiI
[39] Abuse https://abuse.ch/
[40] URLhaus API https://urlhaus.abuse.ch/api/
[41] urlscan.io API v1 https://urlscan.io/docs/api/
[42] Fraudulent E-mail Corpus https://www.kaggle.com/datasets/rtatman/fraudulent-email-corpus.
[43] Fraud Email Dataset https://www.kaggle.com/datasets/pramodgupta92/fraud-email-datasets
[44] Hillary Clinton's Emails https://www.kaggle.com/datasets/kaggle/hillary-clinton-emails
[45] publicCorpus https://spamassassin.apache.org/old/publiccorpus/
[46] Email Spam Dataset https://www.kaggle.com/datasets/nitishabharathi/email-spam-dataset
[47] Spam Email https://www.kaggle.com/datasets/mfaisalqureshi/spam-email
[48] Spam Email Dataset https://www.kaggle.com/datasets/venky73/spam-mails-dataset
[49] Spambase Data Set https://archive.ics.uci.edu/ml/datasets/spambase
[50] Malicious And Benign URLs https://www.kaggle.com/datasets/siddharthkumar25/malicious-and-benign-urls
[51] Phishing Site URLs https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls
[52] URLS Classification Dataset https://www.kaggle.com/datasets/shivamb/spam-url-prediction

[53] URLhaus phishing https://urlhaus.abuse.ch/api/#csv
[54] The pandas development team "pandas-dev/pandas: Pandas", 2020, DOI 10.5281/zenodo.3509134
[55] Psycopg2 https://www.psycopg.org/
[56] Michael L. Waskom "seaborn: statistical data visualization", Journal of Open Source Software, Vol. 6, 2021, pp. 3021, DOI 10.21105/joss.03021
[57] Hunter, J. D. "Matplotlib: A 2D graphics environment", Computing in Science & Engineering, Vol. 9, No. 3 2007, pp. 90-95, DOI 10.1109/MCSE.2007.55
[58] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. "Scikit-learn: Machine Learning in {P}ython", Journal of Machine Learning Research, Vol. 12, 2011, pp. 2825-2830, DOI 10.21105/joss.03021