Fake Architecture Orchestrator

Orlando De Bernardis
Pierantonio Carrozzini

Nel panorama sempre più digitale ed interconnesso del mondo contemporaneo, la protezione dei dati è diventata una priorità critica per le aziende di ogni settore. La rapida evoluzione della tecnologia ha aperto nuove opportunità per l'innovazione e la crescita, ma ha anche introdotto nuove sfide legate alla sicurezza informatica. Al giorno d'oggi, la salvaguardia delle informazioni sensibili è diventata un elemento fondamentale per il successo e per la reputazione di un'azienda. Le minacce variano da attacchi di malware sofisticati a frodi informatiche, e le conseguenze di un'eventuale violazione della sicurezza possono essere devastanti, causando danni finanziari e perdita di fiducia dei clienti, danneggiando così la reputazione aziendale.

Una delle pratiche contemporanee più diffuse nel panorama della sicurezza informatica è quella dell'utilizzo di dati fittizi per proteggere dati reali, tecnica che prende il nome di "honeytoken". Questa comporta un trattamento di record falsi che vengono poi aggiunti a database legittimi, che forniscono così agli amministratori di rete e agli esperti in materia un modo proattivo di monitorare attività sospette o tentativi di violazione della sicurezza, prima che questi possano causare danni reali; eventuali aggressori, infatti, potrebbero dedicare tempo e risorse a cercare di compromettere questi elementi falsi invece che prendere di mira risorse di valore effettivo. Tuttavia, è importante gestire attentamente l'implementazione di tali honeytoken per evitare false allerte o un'eventuale esposizione involontaria di dati sensibili.

L'obiettivo di questo progetto è, dunque, quello di creare facilmente un'infrastruttura "fake" da un diagramma architetturale e quello di istanziare un container Docker per ogni tipo di risorsa individuata, tramite la creazione di un "terraform plan" adeguato.

La soluzione da noi proposta va a soddisfare tutte le specifiche di progetto richieste, permettendo all'utente di scegliere se fornire il diagramma architetturale tramite un opportuno file ".xml" o tramite una vera e propria immagine in formato ".png" o ".jpeg".

Il codice è stato progettato per essere modulare ed è perciò suddiviso in vari moduli, ognuno dei quali svolge funzioni ben precise.

- ***generate_infrastructure.py*** si tratta del main dell'intero progetto;
- ***img_parser.py*** contiene il codice per l'analisi tramite immagine;
- ***xml_parser.py*** contiene il codice per l'analisi e la traduzione del file xml;
- ***utils.py*** contiene la definizione di tutti i path e le funzioni di utility;
- ***terraform_plan_generator.py*** si occupa della creazione del terraform_code per generare il plan;

```
loading Roboflow workspace...
loading Roboflow project...
Scegli un'opzione (digita 'img', 'xml' o 'compose'):
```

Una volta eseguito lo script viene dapprima caricato da remoto il modello di rete neurale e successivamente viene chiesto all'utente come si intende procedere:

- Digitando **"xml"** si sceglierà di istanziare le varie componenti e, dunque, di generare il terraform plan fornendo in input un diagramma in formato xml.
- Digitando "**img"** si procederà tramite l'analisi e il riconoscimento delle risorse da un file immagine.

- Digitando **“compose”** verrà lanciato il comando <<docker compose up -d>> e verranno istanziati i container contenuti nel file di configurazione *compose.yaml* presente nella directory del progetto.

Lo script è stato realizzato quasi interamente in Python utilizzando:

- **IDE**: *PyCharm Community Edition 2023*.
- *Docker Desktop -Versione 4.25.0.*
- *Terraform -AMD64 -Versione 1.6.5.*
- YoloV5
- Roboflow

➔ XML_PARSER.PY

Per l’analisi e la traduzione di un file formato xml è necessario utilizzare il modulo ***xml.etree.ElementTree*** , il quale implementa API semplici ed efficienti per la creazione e l’analisi di dati xml.
XML è un formato dati intrinsecamente gerarchico e il modo più naturale e semplice per rappresentarlo è attraverso una struttura ad albero. ET ha due classi per questo scopo: ***ElementTree*** rappresenta l’intero documento come un albero ed ***Element*** rappresenta un singolo nodo dell’albero in questione.
Questo script è stato implementato con l’obbiettivo di analizzare informazioni provenienti da un file XML creato con Draw.io (o applicazioni simili) e raccogliere i risultati come una lista di dizionari. Ogni elemento della lista è un dizionario che contiene diverse coppie chiave-valore, rappresentanti le informazioni di ciascun componente. In questo modo, i dati vengono resi disponibili allo script che andrà così a generare il piano Terraform, oltre ad essere stampati in forma tabulare in console e in un foglio di lavoro Excel.
Il ciclo principale scorre tutti gli elementi XML, identificando i componenti con i tag **<object>** ed **<mxCell>**. Questo espediente è stato utilizzato al fine di estendere il caso d’uso presentato nella traccia di progetto e permettere al programma di riconoscere anche eventuali componenti “non standard”.
Object è un termine generico utilizzato per rappresentare qualsiasi oggetto o elemento all’interno di un documento XML, definito da uno sviluppatore o da uno standard per rappresentare dati o particolari informazioni all’interno del documento.
MxCell, al contrario, è spesso associato al framework JS “mxGraph”, utilizzato per la visualizzazione e la manipolazione di grafici e diagrammi interattivi. In questo contesto, esso rappresenta una cella all’interno di un grafico, che può rappresentare nodi, connessioni o relazioni tra i vari componenti. In altre parole, tutti quei componenti che presentano delle configurazioni aggiuntive implementate direttamente dal creatore del diagramma restano comunque invisibili ad occhio nudo.
Una volta identificati i componenti standard e non, lo script in questione ne estrae le informazioni di base come ID, nome, valore, tipo, stile, posizione e dimensioni; successivamente conta il numero di elementi riconosciuti e non riconosciuti, tenendo traccia del conteggio per ciascun tipo.
Per dimostrare quanto detto in precedenza, qualora fosse possibile, vengono anche estratte e visualizzate le informazioni aggiuntive relative a porte interne ed esterne.
Infine, lo script utilizza la libreria **tabulate** per stampare le informazioni di ciascun componente in formato tabulare nella console ed aggiunge i componenti ad una struttura dati tabulare (**table_data**) per la successiva creazione di una tabella Excel.

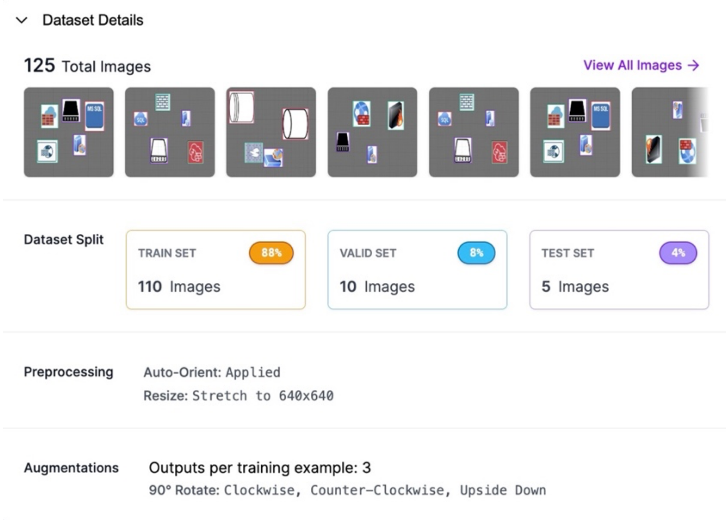
```
Scegli un'opzione (digita 'img', 'xml' o 'compose'): xml
Hai scelto l'opzione 'xml'. Proseguo con il codice relativo a 'xml'.
=== Starting Detection ===
---------------------------------------------
---------------------------------------------
---------------------------------------------
---------------------------------------------
---------------------------------------------
| Chiave        | Valore                                                                                                                 |
|---------------+------------------------------------------------------------------------------------------------------------------------|
| number        | 1                                                                                                                      |
| id            | bVUwOe1Q3J5sBHsqbzTg-1                                                                                                 |
| name          | bVUwOe1Q3J5sBHsqbzTg-1                                                                                                 |
| value         | web_server                                                                                                             |
| type          | web_server                                                                                                             |
| Internal_port |                                                                                                                        |
| External_port |                                                                                                                        |
| style         | image;aspect=fixed;perimeter=ellipsePerimeter;html=1;align=center;shadow=0;dashed=0;spacingTop=3;image=img/lib/active_directory/web_server.svg; |
| x             | 380                                                                                                                    |
| y             | 230                                                                                                                    |
| width         | 40                                                                                                                     |
| height        | 50                                                                                                                     |
```

Viene così utilizzato **pandas** per creare un DataFrame contenente le informazioni sui componenti, da cui viene scritto il DataFrame in un foglio di lavoro Excel, utilizzando la libreria **openpyxl**.
Viene stampato il numero totale di componenti trovati e il numero di componenti per tipo e viene mostrata la tabella dei componenti nella console.

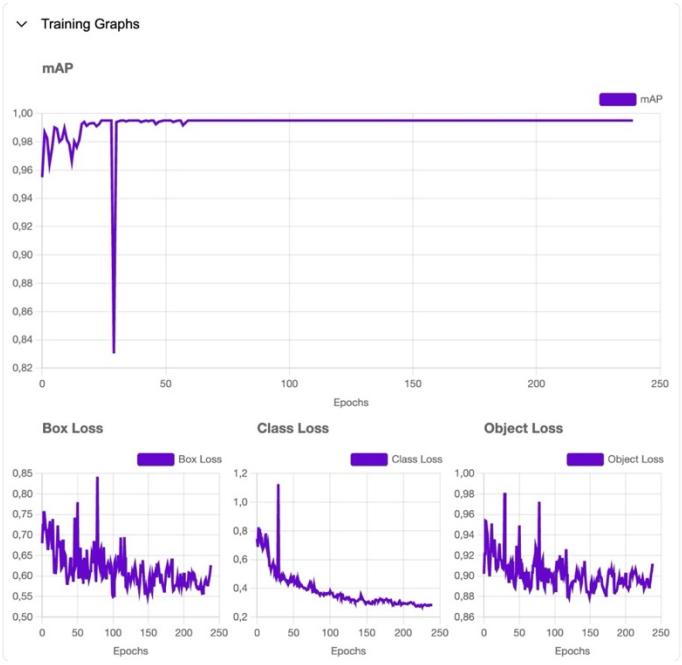
➔ DATASET ED ADDESTRAMENTO DEL MODELLO DI OBJECT DETECTION

Per la realizzazione del dataset e l'addestramento del modello è stato utilizzato RoboFlow: una piattaforma che fornisce ottimi strumenti per l'annotazione e l'addestramento di modelli di *object detection.* Questa offre strumenti per semplificare il processo di annotazione dei dati, un passo cruciale nell'addestramento di modelli di machine learning, e fornisce inoltre strumenti semplificati per l'addestramento e l'ottimizzazione del modello. Inoltre, si integra facilmente con i framework più popolari come TensorFlow o PyTorch, rendendo più veloce ed immediato il processo di sviluppo e implementazione.
Il dataset è stato creato realizzando su Draw.io all'incirca un centinaio di diagrammi architetturali, includendo le icone di AWS17, AWS 18 e AWS19, e sono anche state effettuate operazioni di *pre-processing* e *data augmentation* quali il resize, portando ogni campione alla dimensione di 640x640 pixel, e la rotation di 90° in ogni direzione delle immagini.
Si è scelto di suddividere il dataset in tre diverse cartelle:

- ***train_set***: contenente 110 immagini ed utilizzato per il training dei dati;
- ***validation_set:*** composto da 10 immagini ed utilizzato per la validation;
- ***test_set:*** contenente sole 5 immagini da utilizzare per il test finale del modello;



Il modello è stato in un primo momento addestrato e testato in locale tramite il modello pre-addestrato YoloV5, ma successivamente si è optato per addestrarlo e valutarne le prestazioni direttamente tramite la piattaforma di RoboFlow per via dei tempi ridotti ed, infine, integrato nel progetto in questione tramite l'apposito package "roboflow".



Come è possibile valutare dal valore di ***mAP***, acronimo di "*mean average precision*", metrica utilizzata comunemente per la valutazione delle prestazioni di modelli di object detection, il modello ha ottenuto prestazioni molto buone nonostante la mole ridotta di dati del nostro dataset.

➔ IMG_PARSER.PY

Nel modulo python ***img_parser.py***, inoltre, è stata utilizzata la libreria di *OpenCV* poiché l'immagine fornita in input deve essere opportunamente elaborata prima di essere sottoposta al modello per l'inferenza. Una volta caricata l'immagine viene effettuato un resize di questa, portandola alle dimensioni di 640x640 pixel, così da essere compatibile con il modello addestrato in precedenza, e ne viene salvata una copia nel file *resized.jpg.*

```
loading Roboflow workspace...
loading Roboflow project...
Scegli un'opzione (digita 'img', 'xml' o 'compose'): img
Hai scelto l'opzione 'img_parser'. Prosegui con il codice relativo a 'img'.
----- IMG_PARSER STARTING -----

 Fornire il path dell'immagine da analizzare o default per l'immagine predefinita:
```

Una volta digitata la sigla "***img***" l'utente si troverà dinanzi a questa schermata, nella quale gli verrà chiesto di inserire da tastiera il *path* dell'immagine da analizzare. Digitando "default", invece, verrà utilizzata l'immagine predefinita di test prevista nella directory di progetto.

Una volta scelta l'immagine, questa verrà analizzata e verranno stampati a schermo i risultati della predizione, indicando la "Class" e la relativa "Confidence" come mostrato nella seguente immagine.

```
Fornire il path dell'immagine da analizzare o default per l'immagine predefinita: default
Class: firewall, Confidence: 0.9400

Class: database, Confidence: 0.9262

Class: database, Confidence: 0.9107

Class: web_server, Confidence: 0.8327

Class: web_server, Confidence: 0.8191
```

Inoltre, tramite l'opportuna funzione model.predict.save, viene salvata una copia dell'immagine con le relative predizioni nella directory di progetto con il nome di prediction.jpg.
A questo punto, si procede nello stesso modo per il parsing da file XML, infatti, tali valori verranno inseriti in una lista chiamata *components* che verrà poi passata in input alla funzione responsabile di generare il terraform plan e, dunque, istanziare i vari container docker.

➔ TERRAFORM_PLAN_GENERATOR.PY

Questo script genera automaticamente il codice Terraform necessario al fine di istanziare dei container su Docker, è basato sulla lista di componenti ricevuta dal Parser e su un file di tipo "json" utile alla configurazione del Docker.
Il codice itera su ogni componente nella lista **components** e per ogni elemento vengono create risorse Terraform specifiche, inoltre, le configurazioni aggiuntive basate sul tipo di componente vengono gestite tramite blocchi *if-elif-else*. Ciascuno di questi blocchi gestisce un tipo specifico di componente, al momento il numero di elementi riconosciuti è di 15:
web_server, database, firewall, firewall-alpine, cms, cache, message_queque, proxy, prometheus, grafana, ci_cd, e_commerce, machine_learning, version_control

Variabili dinamiche: Alcune variabili sono incrementate dinamicamente per evitare conflitti di porte o nomi dei container; questo è il caso delle porte web o database. Tramite queste variabili si gestisce infatti un ulteriore caso d'uso, relativo alla presenza di più elementi con la stessa tipologia all'interno di un singolo diagramma.
Configurazioni per Vari Tipi di Componenti: Ogni risorsa ha le proprie configurazioni specifiche, e tra le più interessanti troviamo sicuramente il server web per l'enorme estendibilità permessa da Nginx.
Il database, vista la presenza sull'Hub di Docker di ogni immagine di database esistente e la grande quantità di documentazione disponibile grazie alla community, risulta di notevole interesse.

Questo script, infine, genera il terraform plan che tramite l'utilizzo della libreria **subprocess** verrà prima inizializzato e dopodiché applicato per portare all'effettiva creazione dei vari container sul Docker.