**Fake Architecture Orchestrator**

Immagine che contiene Elementi grafici, arte, bianco e nero, schermata

Descrizione generata automaticamente

**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 la reputazione di un'azienda.

Queste 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, perdita di fiducia dei clienti e danneggiando la reputazione aziendale.

Una delle pratiche più diffuse al giorno d’oggi nel panorama della sicurezza informatica è quella di utilizzare dati fittizi per proteggere dati reali e prende il nome di “honeytoken”. Si tratta di record falsi che vengono aggiunti a database legittimi e che forniscono agli amministratori di rete e agli esperti di sicurezza un modo proattivo di monitorare attività sospette o tentativi di violazione della sicurezza prima che 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 esposizione involontaria di dati sensibili.

L’obiettivo di questo progetto è, dunque, quello di creare facilmente un’infrastruttura “fake” da un diagramma architetturale e 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 tutto è stato realizzato in Python utilizzando PyCharm CE come IDE, ed inoltre sono stati necessari strumenti aggiuntivi come Docker, Terraform, YoloV5 e Roboflow (questi ultimi necessari per il modello di rete neurale per analizzare i file immagine).

Il codice è stato progettato per essere modulare ed è, quindi, 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;

Immagine che contiene schermata, testo, nero, Carattere

Descrizione generata automaticamente

Una volta eseguito lo script viene dapprima caricato da remoto il modello di rete neurale e successivamente è 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* contenuto nella directory del progetto.
* 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 con un albero. ET ha due classi per questo scopo: ***ElementTree*** rappresenta l’intero documento come un albero ed ***Element*** rappresenta un singolo nodo in questo albero.

Una volta definito il path dove andare a leggere il file .xml questo è elaborato dalla funzione di parser che va ad iterare lungo tutte le righe del documento cercando nei tag il valore “***Object***” o “***MxCell***” dal quale estrarre le informazioni sulle risorse a cui si è interessati.

***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.

Una volta identificato il tag, da questo sono estratte tutte le informazioni dell’oggetto tra cui “id” , “number”, “name” e, il più fondamentale “type” che identifica il tipo di componente rappresentato tra quelli riconosciuti. Tutte queste informazioni sono aggiunte ad un dizionario “*components”* che verrà successivamente fornito in input alla funzione responsabile di tradurre il tutto in codice HCL per la generazione del terraform plan.

Tutte le informazioni estratte dal documento XML sono, inoltre, stampate a schermo in modo tabulare, utilizzando l’apposito modulo *tabulate,* per una visione ordinata e più efficiente.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

* DATASET ED ADDESTRAMENTO DEL MODELLO DI OBJECT DETECTION

Per la realizzazione del dataset e l’addestramento del modello è stato utilizzato RoboFlow; si tratta di una piattaforma che fornisce ottimi strumenti per l’annotazione e l’addestramento di modelli di *object detection.* Esso offre strumenti per semplificare il processo di annotazione dei dati, che è un passo cruciale nell’addestramento di modelli di machine learning ed inoltre fornisce strumenti semplificati per l’addestramento e l’ottimizzazione del modello. 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 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

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

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”.

Immagine che contiene testo, Diagramma, schermata, linea

Descrizione generata automaticamente

Come è possibile valutare dal valore di ***mAP***, acronimo di “*mean average precision”*, metrica utilizzata comunemente per valutare le 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 alla dimensioni di 640x640 pixel, cosi da essere compatibile con il modello addestrato in precedenza, e ne viene salvata una copia nel file *resized.jpg.*

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

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.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

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 un dizionario chiamato *components* che verrà passato in input alla funzione responsabile di generare il terraform plan e, dunque, istanziare i vari container docker.