Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

UN SISTEMA DI SESSION TRACKING E THREAT EVALUATION PER APPLICAZIONI WEB BASATO SU HONEYTOKEN

Elaborato in:
Systems Integration

Relatore:
Prof. Vittorio Ghini

Presentata da:
Colotti Manuel Enrique

Correlatore:
Dott. Ciro Barbone

Tutor Aziendale:
Dott. Eugenio Cavina

Sessione I
Anno Accademico 2020/2021

Introduzione

La situazione di emergenza che stiamo vivendo in questi anni ha contribuito in maniera determinante a velocizzare il processo di digitalizzazione delle piccole e medie imprese, un esempio di ciò risulta essere l'adozione da parte di queste ultime di nuove soluzioni a base web per permettere l'acquisto e la successiva consegna di ogni genere di prodotto a domicilio. Parallelamente a questo processo di innovazione risulta essere sempre più rilevante il tema della sicurezza dei sistemi informatici, continuamente bersagliati da attacchi di giorno in giorno più efficaci volti al furto di informazioni sensibili.

In questo scenario la mia tesi proporrà una soluzione al problema della sicurezza di applicazioni web realizzando un sistema capace di etichettare, registrare e valutare le azioni che gli utenti svolgono su di una determinata piattaforma, sfruttando diverse tecniche di tracciamento a base web. Una delle componenti fondamentali di tale sistema risulta sicuramente essere WordPress: il CMS che mi ha permesso di realizzare il plugin impiegato per la gestione delle informazioni di ogni singolo utente. Risulta altrettanto importante l'utilizzo di Docker per il dispiegamento dei servizi che comporranno il sistema e dei container che verranno utilizzati in fase di testing.

La dissertazione sarà costituita da 4 diversi capitoli. Nel primo verrà descritto il contesto all'interno del quale il sistema è stato sviluppato trattando dello stato dell'arte in ambito di web tracking e dei concetti teorici relativi alla virtualizzazione tramite Docker. Nel secondo verranno discussi i requisiti del sistema e le diverse tecnologie utilizzate per realizzarlo. Nel terzo verrà ampiamente esposto il processo di implementazione di ogni parte del sistema insieme alla motivazioni che mi hanno portato ad effettuare ogni specifica scelta. Infi-

ne nell'ultimo capitolo verranno mostrati i test implementati per verificare le funzionalità del sistema ed i risultati ottenuti.

Indice

Capitolo 1

Contesto

In questo primo capitolo mi sono occupato di descrivere lo stato dell'arte, risalente al periodo in cui questa tesi è stata elaborata, per quanto riguarda due argomenti che costituiscono le fondamenta sopra le quali la trattazione si sviluppa:

- Tecniche di tracciamento utenti sul Web [2] [3] [6] [1]
- Virtualizzazione a livello sistema operativo e approfondimenti sulla piattaforma Docker

1.1 Stato dell'arte nel Web Tracking

Negli ultimi anni il tracciamento degli utenti all'interno di siti web è diventato un tema molto caldo anche a causa delle numerose nuove tecnologie e metodologie proposte in letteratura e implementate da numerose compagnie per poter analizzare il comportamento delle persone. Nelle seguenti sezioni saranno discusse: le macro categorie all'interno delle quali le tecniche di tracciamento possono essere suddivise e alcune metodologie utilizzate in scenari reali.

1.1.1 Session Based Web Tracking

Storicamente il tracciamento basato sulle sessioni è stato il primo meccanismo implementato dai siti web per la classificazione di utenti, di conseguenza le

tecniche che questi metodi sfruttano sono semplici e non rappresentano, se prese singolarmente, una vera e propria minaccia per la privacy degli utenti. Di seguito verranno brevemente descritte le più comuni utilizzate ancora oggi.

1.1.1.1 Metodologie

- **Autenticazione Esplicita:** Consiste sostanzialmente nell'obbligare l'utente a registrarsi e ad effettuare il login sul sito web per poterne navigare tutti i contenuti. Questo rende l'identificazione dell'utente molto precisa poiché indipendente dal browser o computer utilizzato e dalla posizione geografica; tuttavia il problema principale legato a questa tecnica è che potrebbe risultare scomodo e sospetto richiedere ad un utente delle credenziali ad ogni nuovo accesso alla piattaforma. Un'altra considerazione è invece legata più all'aspetto comportamentale: un utente sapendo di essere registrato eviterebbe sicuramente azioni che possano attirare l'attenzione su di lui.

- **ID di sessione nascosti nei form:** Prima dell'introduzione dei cookie nel 1994, le tecniche di tracciamento erano molto più deboli a causa della non persistenza delle informazioni. Una di queste consiste nel passare un identificatore univoco legato all'utente attraverso l'URL come parametro GET o tramite un form nascosto come parametro POST. Sebbene tale identificativo possa essere passato anche a siti terzi, la sua utilità sarebbe circoscritta solo ad una singola sessione di navigazione.

- **Proprietà window.name:** La window.name è una proprietà del Document Object Model (DOM) definito dal W3C, al cui interno è possibile salvare dati fino ad un massimo di 2MB. Le informazioni salvate lato client in questo attributo risultano persistenti anche dopo l'aggiornamento della pagina e sono accessibili anche a siti terzi.

1.1.2 Storage Based Web Tracking

Il tracciamento di tipo storage based risulta essere ancora oggi il più utilizzato e avanzato poiché permette il salvataggio diretto di dati all'interno dei computer

degli utenti nonché anche, a seconda della tecnica impiegata, di riconoscere determinate caratteristiche di un dispositivo tra cui sistema operativo o specifica istanza del browser.

1.1.2.1 Metodologie

- Cookies HTTP
- Flash Local Shared e LocalConnections Objects
- Silverlight Isolated Storage
- HTML5 Global, Local and Session Storage
- HTML5 IndexedDB

Cookies HTTP

I cookie possono essere descritti ad alto livello come dei contenitori di dati di dimensione ridotta che vengono salvati all'interno dei computer degli utenti, si suddividono in cookie di sessione e cookie persistenti: i primi vengono eliminati alla chiusura del browser, i secondi invece vengono eliminati solo dopo un certo periodo di tempo stabilito da chi li crea.

La proprietà di questa tecnologia di salvare informazioni trasparentemente all'utente, unita alla sua velocità, gli ha permesso di diventare la base per tutte le moderne tecniche di tracciamento nonché la più utilizzata nel World Wide Web odierno.

Flash Local Shared e LocalConnection Objects

I Local Shared Objects sono un ulteriore tecnica per salvare dati sul computer di un utente utilizzata da Adobe Flash. Questi oggetti possono memorizzare fino a 100kB di informazioni, rispetto ai cookie sono più difficili da cancellare e sono condivisi tra tutti i browser di un sistema che usano Adobe Flash.

I LocalConnection Objects invece sono un ulteriore meccanismo implementato da Adobe per la comunicazione di diverse istanze di Flash all'interno dello

stesso sistema. Questa tecnica unita a quella dei Local Shared Objects può essere implementata con il fine di scambiare informazioni in maniera trasparente tra sessioni di navigazione pubbliche e sessioni in incognito.

Sebbene queste due tecniche siano ampiamente state sfruttate in passato, oggi non sono più applicabili a causa della fine del supporto da parte di Adobe al plugin Flash e della rimozione di Flash da tutti i browser più famosi.

Silverlight Isolated Storage

Il plugin Silverlight di Microsoft permetteva il salvataggio di dati fino ad un massimo di 100kb per sito in uno storage locale all'utente. Tuttavia come per Adobe Flash, Microsoft ha annunciato la fine del supporto entro Ottobre 2021 per tutti i motori di ricerca.

HTML5 Local and Session Storage

L'avvento di HTML5 ha introdotto due nuovi tipi di storage per le informazioni salvate da parte dei siti web:

- Il **Local Storage** permette di salvare coppie di dati chiave-valore fino ad un massimo di 5MB che possono essere condivise tra diverse schede dello stesso browser; lo spazio occupato dalle variabili presenti nel Local Storage viene automaticamente liberato all'eliminazione dei cookie da parte dell'utente.

- Il **Session Storage** è molto simile al Local Storage, le uniche differenze riguardano il fatto che i dati presenti al suo interno non possono essere condivisi tra finestre dello stesso motore di ricerca e che vengono automaticamente eliminati alla chiusura del browser.

HTML5 IndexedDB

IndexedDB è un ulteriore novità introdotta da HTML5 in sostituzione a Web SQL Database; quest'ultima tecnologia permetteva il salvataggio di dati lato client in un database SQLite. IndexedDB opera in maniera molto simile al Local Storage discusso nel paragrafo precedente, pertanto gli impatti sulla privacy sono analoghi.

1.1.3 Cache Based Web Tracking

Anche le tecnologie comprese in questo gruppo sfruttano lo storage dei dispositivi, la differenza è che in questo caso non vi viene salvato alcun dato al suo interno. Lo scopo principale delle seguenti tecniche infatti è quello di sfruttare le varie cache presenti per identificare istanze diverse dei motori di ricerca e ricostruire un indice dei siti già visitati dall'utente.

1.1.3.1 Web Cache Tracking

Loading Performance Test
Attraverso l'impiego di Javascript è possibile misurare il tempo di caricamento di uno specifico oggetto all'interno di una pagina web come un immagine o un logo. Questa misura delle prestazioni da la possibilità di identificare quali pagine di un sito sono già state visionate dall'utente poiché richiederanno, a causa dei meccanismi di caching, un tempo di caricamento minore.

ID di oggetti in Cache
Questo metodo da la possibilità di tracciare un utente facendogli richiedere un documento HTML contenente un identificatore univoco, che verrà salvato all'interno della cache HTTP del browser. L'identificatore può essere facilmente inserito all'interno di un *div* nascosto e recuperato quando necessario attraverso la lettura del documento HTML già presente in cache. In aggiunta, dato che i file all'interno della cache possono essere inclusi da un qualunque sito, l'identificativo dell'utente può essere utilizzato per il tracciamento anche da parte di altri siti web.

ETags e HTTP Last-Modified headers
In letteratura sono stati anche proposti metodi per il tracking che sfruttano degli specifici header di richieste HTTP, questo è il caso dei due header Etag e Last-Modified. Entrambe vengono inviati da un web server quando l'utente scarica un oggetto di una pagina per la prima volta; l'Etag header può memorizzare fino ad un massimo di 81864 bit, Last-Modifed invece sebbene sia pensato per memorizzare un timestamp, può accettare una qualsiasi stringa. Quando

l'utente visita nuovamente il sito, gli header If-Modified-Since e If-None-Match vengono inviati come parte della richiesta HTTP, il loro contenuto sarà rispettivamente ciò che era presente negli header Last-Modified e Etag; la ricezione da parte del web server degli stessi identificativi impostati durante la prima visita dell'utente, costituisce il meccanismo di tracciamento.

1.1.3.2 Cache DNS

Un ulteriore cache che può essere sfruttata è quella DNS: Javascript infatti può indirettamente generare un DNS lookup per uno specifico sito e misurarne il tempo di risposta. Nel caso in cui non sia mai stato visitato e quindi non presente nella cache DNS, la query ci metterà molto più tempo ad essere risolta; al contrario se la query viene eseguita in tempo breve, si ha un indicazione che il sito era già stato visitato dall'utente.

1.1.3.3 Cache Operazionali

Le cache operazionali sono delle porzioni di memoria specifiche all'interno delle quali i browser memorizzano informazioni relative a diversi tipi di operazioni che effettuano trasparentemente all'utente: redirect permanenti, credenziali di autenticazione o liste di domini che devono essere acceduti mediante l'utilizzo del protocollo HTTP Strict Transport Security. Nei seguenti paragrafi verrà descritto come alcune di queste cache possono essere sfruttate con il fine di tracciare gli utenti.

HTTP 301 Redirect Cache

Il meccanismo di redirect 301 di HTTP è stato creato per segnalare ad un motore di ricerca che la risorsa che si sta cercando è disponibile ad un altro URL. Un servizio che al primo accesso di uno specifico utente risponde con un redirect allo stesso URL, con però un identificativo univoco appeso in fondo, sarebbe in grado di identificare lo specifico utente ad ogni nuova richiesta. Questo avviene poiché all'atto del redirect 301 il browser inserisce in cache la coppia di valori costituita dall'URL originale e da quello modificato; in questo modo anche se l'utente cercasse di accedere alla risorsa attraverso l'URL

originale, il motore di ricerca gli sostituirebbe quello modificato contenente l'id univoco.

Cache per l'autenticazione tramite HTTP

Il protocollo HTTP mette a disposizione due meccanismi di autenticazione: Basic Access Authentication e Digest Access Authentication. Quando un utente inserisce le sue credenziali all'interno di un sito, quest'ultimo le memorizza temporaneamente in modo da poterle inviare automaticamente nell'header di autenticazione delle successive richieste HTTP. In un articolo di Jeremiah Grossman [4] vengono presentati diversi metodi per poter forzare un browser ad autenticarsi con le credenziali già presenti in cache, il tutto senza mostrare alcun pop-up o notifica all'utente.

HTTP Strict Transport Security Cache

Il protocollo HTTP Strict Transport Security permette ad un webserver di richiedere che tutte le connessioni effettuate ad esso utilizzino il protocollo HTTPS; tale meccanismo, insieme ai certificati SSL Wildcard, può essere sfruttato per identificare univocamente un'istanza di un browser attraverso il seguente processo:

1. Il server mappa un identificativo univoco ad una lista di sottodomini del sito (ID: "ABC", sottodomini: 1-A.domain.com, 2-B.domain.com, 3-C.domain.com). Ognuno di questi sottodomini potrà disporre di un certificato SSL grazie ai Wildcard SSL Certificate che permettono di anteporre al vero dominio del sito un qualunque sottodominio (*.domain.com).

2. Durante un primo accesso l'utente viene reindirizzato ad ognuno di questi domini effettuando delle richieste HTTPS. Dal momento in cui anche il protocollo HTTP Strict Transport Security è attivo per il dominio, i sottodomini visitati vengono inseriti come entry nella cache di HSTS.

3. Ad un successivo accesso al sito da parte dell'utente, uno script JavaScript si occuperà, con una tecnica simile ad un attacco a forza bruta, di effettuare richieste HTTP alle varie combinazioni dei sottodomini (ex. 1-A.domain.com, 1-B.domain.com, 1-C.domain.com, ecc.). In questo modo

le richieste effettuate ai sottodomini 1-B.domain.com e 1-C.domain.com non verranno promosse da HTTP a HTTPS poiché non è presente una entry per questi sottodomini all'interno della tabella di HSTS; quando invece verrà interrogata la risorsa 1-A.domain.com, il server si accorgerà che la richiesta è stata promossa da protocollo HTTP a HTTPS, indicando che un record per quel sottodominio è presente nella cache di HSTS.

4. Iterando questo processo per ogni combinazione possibile dei sottodomini è possibile ricostruire l'identificativo originario generato durante il primo accesso al sito.

Cache e ID di sessioni TLS

TLS utilizza una cache per memorizzare gli ID di sessione inviati dal server al client durante la fase di handshake. Questi ID sono utilizzati per ripristinare le sessioni durante i successivi accessi senza dover effettuare nuovamente il computazionalmente costoso processo di handshake. Tale identificativo può essere potenzialmente usato per tracciare utenti, a causa di ciò l'implementazione del meccanismo di recupero delle sessioni è stata modificata all'interno di Tor Browser, noto soprattutto per la privacy che offre agli utenti che ne fanno uso.

1.1.4 Altri meccanismi di tracciamento

Le seguenti tecniche sebbene non impieghino tecnologie particolarmente diverse da quelle già trattate, risultano molto efficienti e persistenti per quanto riguarda il tracciamento di utenti all'interno del web poiché sono in grado di sfruttare contemporaneamente diversi meccanismi basati sul salvataggio di dati lato client.

1.1.4.1 Evercookies

Un **Evercookie** può essere descritto come un meccanismo che permette di salvare all'interno dello storage di un utente numerosi tipi di cookie, tutti

caratterizzati dallo stesso contenuto. Il motivo principale di questa implementazione è la resilienza delle informazioni: se un utente cercasse di eliminare i cookie impostati durante il primo accesso ma ne lasciasse in memoria anche solo uno, tutti quelli eliminati potrebbero essere immediatamente ricostruiti a partire da quelli ancora presenti. Perché ciò abbia successo è necessario che un Evercookie comprenda numerose tipologie di cookies salvati in diversi storage; Samy Kamkar nel 2010 rilasciò un applicazione Javascript [5] che implementava questo concetto sfruttando le seguenti tecnologie, molte di queste sono già state citate e descritte nelle precedenti sezioni:

- **HTTP Cookies**
- **Local Shared Objects (Flash Cookies)**
- **Silverlight Isolated Storage**
- **Cronologia delle ricerche**
- **Etags**
- **Web Cache**
- **Proprietà window.name del DOM**
- **Internet Explore userData storage**
- **HTML5 Session Storage**
- **HTML5 Local Storage**
- **HTML5 Global Storage**
- **HTML5 Database SQLite Storage**
- **HTML5 IndexedDB**
- **Java JNLP PersistenceService**
- **Java CVE-2013-0422 exploit**
- **Cookie all'interno di immagini PNG**

1.1.4.2 Favicon Supercookies

Una delle tecniche di tracciamento più discussa del momento è sicuramente quella che riguarda i **Favicon Supercookies**. Questo metodo che può essere classificato tra quelli cache based, permette ad un server di identificare univocamente un motore di ricerca attraverso le piccole immagini presenti nel menu che mostra le diverse schede del browser, affiancate al titolo di ogni singolo sito web aperto (Favicon images).

L'implementazione di questa tecnica ricorda da un certo punto di vista il tracciamento che viene effettuato tramite la cache del protocollo HSTS, tuttavia vi sono delle differenze che rendono quest'ultimo metodo molto più efficace ed applicabile ad uno scenario reale. Di seguito descriverò più nel dettaglio i passaggi che permettono di realizzare questo tipo di tracciamento nonché i motivi per cui risulta essere molto più precisa ed invasiva rispetto a qualunque altra descritta fino ad ora.

Background lato server

Perché la tecnica possa essere implementata è necessario che il server metta a disposizione una serie di favicon differenti, ognuna posizionata in sottocartelle o sottodomini diversi. Nei seguenti paragrafi verrà descritto: come avviene la loro distribuzione ai vari client, come da queste verrà generato un identificativo univoco e come una specifica cache dei browser denominata F-Cache sarà sfruttata per recuperare tale ID.

Scrittura dell'identificativo univoco

In questa fase viene innanzitutto generato un ID binario univoco relativo al singolo utente, questo a sua volta verrà mappato in una serie di sottocartelle nel seguente modo:

- N-esimo bit uguale a 1: La favicon image presente nella cartella n-esima verrà fatta scaricare all'utente attraverso un redirect.
- N-esimo bit uguale a 0: la favicon image presente nella cartella n-esima non verrà resa disponibile all'utente.

Iterando questo procedimento per ognuno dei bit costituenti l'identificativo sarà possibile far scaricare ad ognuno degli utenti, durante il loro primo accesso, una combinazione di favicon diverse che verranno poi utilizzate per la successiva fase di identificazione.

Lettura dell'identificativo univoco

Dopo l'iniziale fase di scrittura sarà necessario da parte del server sfruttare la F-Cache dei browser degli utenti per riconoscerli durante gli accessi successivi al primo.

Per raggiungere questo obbiettivo l'utente viene reindirizzato ad ognuna delle sottocartelle in modo che il browser trasparentemente richieda di scaricare le varie favicon al loro interno; i loghi non ancora scaricati verranno richiesti al webserver che risponderà con uno status 404 Not Found per preservare lo stato corrente della F-Cache, quelli già scaricati invece non verranno richiesti poiché già presenti in cache.

Terminato questo processo il server sarà in grado di ricostruire l'identificativo dell'utente tenendo conto di quali favicon sono state richieste e quali invece ignorate:

- Il browser effettua una richiesta per la favicon nella sottocartella n-esima: bit N-esimo impostato a 0
- Il browser ignora la favicon nella sottocartella N-esima: bit n-esimo impostato a 1

Rilevanza del tracciamento tramite Supercookie

I motivi per cui questa tecnica è molto più efficiente di altre deriva da una serie di fattori elencati qui di seguito:

- La F-Cache all'interno della quale vengono salvate automaticamente le favicon da parte dei browser è uno speciale tipo di cache che allo stato attuale può essere cancellata solo ripristinando il browser alle impostazioni di fabbrica o reinstallandolo. A causa di ciò l'identificativo mappato al suo interno da parte del webserver risulta essere estremamente persistente.

- Compatibilità con tutti i browser desktop e mobile più importanti.
- L'impiego di software anti-tracciamento, adblockers, VPN o ricerca in modalità incognito non protegge da questo tipo di tecnica.
- La tecnica scala proporzionalmente con il numero di bit impiegati per memorizzare l'identificativo binario; a N bit corrispondono 2^N utenti univoci, N redirect e N diverse sottocartelle o sottodomini. Per questi motivi, maggiore sarà la lunghezza dell'identificativo, maggiore sarà il tempo impiegato per effettuare gli N redirect.

1.2 Virtualizzazione

La virtualizzazione è una tecnica che permette di eseguire uno o più sistemi operativi all'interno di un unico PC, l'uno isolato dall'altro, all'interno di una macchina virtuale; in questo contesto il sistema operativo in cui viene eseguita la macchina virtuale è chiamato host, mentre le macchine virtuali vengono identificate come guest.

A differenza di ciò che avviene per l'emulazione, dove le istruzioni della macchina guest vengono tradotte in una sequenza di istruzioni della macchina host, nella virtualizzazione il codice della macchina virtuale viene direttamente eseguito dal sistema host; questa differenza garantisce delle prestazioni di gran lunga migliori per i sistemi virtualizzati piuttosto che per quelli emulati.

Normalmente quando si parla di virtualizzazione ci si riferisce implicitamente a quella del livello Hardware, tuttavia in questi ultimi anni grazie anche all'avvento di nuove tecnologie come Docker, sta diventando molto popolare il concetto di virtualizzazione del livello OS o container-level.

1.2.1 Virtualizzazione del livello OS

La virtualizzazione a livello OS è una tecnica che permette l'esecuzione di applicazioni in un ambiente logicamente isolato rispetto a quello dell'host, all'interno di un singolo sistema operativo. A differenza di ciò che avveniva per

la virtualizzazione hardware, in questo caso è presente solo il kernel del sistema ospitante e più istanze di user-space (figura 1.1).

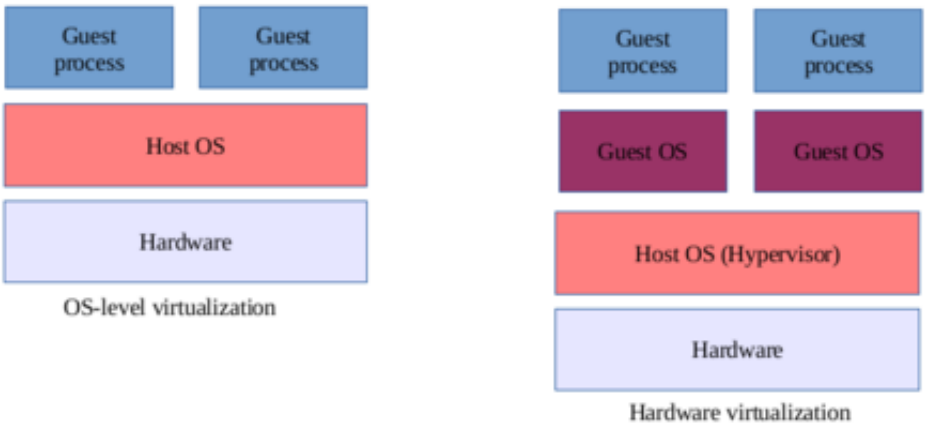

Figura 1.1: Confronto tra virtualizzazione a livello OS e Hardware

1.2.1.1 Isolamento di Processi

La virtualizzazione delle istanze nello user space sfrutta dei meccanismi nativi dei sistemi operativi Unix come **Chroot**; quest'ultimo, abbreviazione di *"change root"*, è un operazione che permette di cambiare la directory root di un processo. Teoricamente quando un processo viene trasferito in una directory root virtuale esso potrà accedere solo ai file all'interno di quest'ultima; chroot da solo garantisce solo una virtualizzazione di base che però, se integrata con altre funzioni kernel native come **Cgroups** e **Namespaces**, permette di avere processi in esecuzione in ambienti isolati con accesso limitato alle risorse hardware.

Cgroups

Rappresentano dei gruppi di processi associati ognuno ad un insieme di criteri che hanno lo scopo di limitare l'uso delle risorse del sistema da parte dei diversi

programmi in esecuzione. Le principali funzionalità di Cgroups possono essere riassunte in questi 4 punti:

1. **Limitazione delle risorse:** utilizzo della CPU, utilizzo della memoria, letture/scritture su disco.
2. **Gestione della priorità:** è possibile dare ad alcuni gruppi di processi la possibilità di utilizzare più risorse rispetto ad altri gruppi.
3. **Accounting:** funzionalità che permette di tenere conto delle risorse in uso da ciascun gruppo.
4. **Controllo:** permette di fermare e riavviare processi che stanno usando un quantitativo di risorse superiore rispetto a quanto concesso dai criteri del gruppo.

Namespaces

Un **namespace** permette di astrarre una o più risorse globali del sistema come mount points, utenti, processi, dispositivi di rete e altri, in un nuovo ambiente parzialmente o completamente isolato; i processi associati ad uno specifico namespace vedranno tali astrazioni delle risorse globali come fossero delle istanze isolate di esse, pertanto eventuali cambiamenti alle risorse globali di un namespace sono visibili soltanto ai processi all'interno dello stesso namespace. I namespaces in Linux sfruttano tre diverse system calls:

- **clone()**: Duplica un processo in esecuzione specificando cosa del contesto del sistema sia condiviso tra padre e figlio e cosa invece debba essere scollegato.
- **unshare()**: Permette di separare logicamente alcune risorse che attualmente sono condivise tra il processo padre e quello figlio; in sostanza realizza le stesse funzionalità della clone senza però la necessità di duplicare nuovamente il processo.
- **setns()**: Associa un processo ad un determinato namespace, le risorse del sistema vengono suddivise in 6 diverse categorie di namespace: processi,

inter-process communication, nome host e di dominio, punti di mount, dispositivi di rete e utenti.

1.2.2 Docker

Docker è una tecnologia sviluppata dalla Docker Inc. e rilasciata nel 2013 che comprende un insieme di prodotti che sfruttano la virtualizzazione a livello sistema operativo per eseguire delle applicazioni in ambienti logicamente separati e isolati dalla macchina host denominati container.

1.2.2.1 Container Docker

Un container come già introdotto è una componente standard che racchiude al suo interno il codice di un applicazione e tutte le dipendenze per eseguirlo correttamente anche su macchine host completamente diverse. A differenza delle macchine virtuali che sfruttano la virtualizzazione hardware, i container sfruttano la virtualizzazione del sistema operativo garantendo un isolamento analogo a quello delle virtual machine, ma una portabilità ed un'efficienza superiori.

Ogni container può essere inserito in una o più reti appositamente create e può essere dotato di uno o più volumi su cui salvare i propri dati; di default i container sono costruiti in modo da essere abbastanza isolati rispetto alla macchina host ma questo aspetto può essere modificato configurando in maniera specifica reti e volumi.

1.2.2.2 Immagini Docker

Un container docker viene costruito a partire da una o più immagini: file template che permettono di definire una serie di istruzioni tramite le quali il container verrà costruito e configurato.

Queste immagini possono essere create essenzialmente in due modi:

- Tramite un file di configurazione denominato Dockerfile che contiene le specifiche per costruire l'immagine.

- In maniera interattiva modificando un container in esecuzione e salvandone le modifiche in una nuova immagine.

1.2.2.3 Architettura di Docker

L'architettura client-server di Docker che permette la containerizzazione di applicazioni e la loro gestione può essere suddivisa in tre componenti fondamentali:

1. **Docker Client:** permette l'interazione tra utente e Docker daemon mettendo a disposizione una REST API per l'invio di comandi tramite socket UNIX o tramite un'interfaccia di rete. I messaggi inviati dal client, una volta ricevuti verranno interpretati ed eseguiti dal demone.

2. **Docker Daemon:** è la componente che espone un API tramite la quale può ricevere messaggi da Docker client o da altri demoni, il suo ruolo è quello di permettere la costruzione, la configurazione e l'esecuzione di immagini, reti, volumi e container. Il demone *dockerd* è anche in grado di comunicare con altri Docker daemon per la gestione di altri servizi Docker.

3. **Docker Registry:** immagazzina un insieme di immagini che possono essere utilizzate dal Docker Daemon per la containerizzazione di un'applicazione. Docker è di default configurato per utilizzare Docker Hub come registro pubblico, tuttavia viene fornita la possibilità di mantenere dei propri registri privati.

Capitolo 2

Analisi e Progettazione

In questo capitolo verranno discusse le caratteristiche e le funzionalità del sistema progettato, nonché le diverse tecnologie impiegate per la sua realizzazione.

2.1 Requisiti del sistema di Session Tracking e Threat Evaluation

Il sistema dovrà soddisfare i seguenti requisiti:

- Tracciamento di visitatori e utenti di una Web Application attraverso uno o più metodi di Web Tracking.
- Valutazione della pericolosità delle richieste effettuate.
- Persistenza dei dati relativi alle sessioni e alle richieste degli utenti.
- Log di richieste e sessioni a fini statistici.
- Meccanismo proattivo per limitare utenti considerati pericolosi in relazione al loro livello di minaccia.

2.2 Descrizione delle tecnologie utilizzate

I paragrafi seguenti saranno dedicati alla descrizione delle varie tecnologie utilizzate per l'implementazione del sistema a cominciare dal Content Manage-

ment System **WordPress** che risulta essere la componente fondamentale sopra cui il sistema è implementato, per poi continuare con tutte le altre che integrate tra loro permettono la realizzazione dei requisiti descritti nella sezione 2.1.

2.2.1 CMS WordPress

WordPress è un diffusissimo Content Management System (CMS) open source sviluppato in PHP che permette di creare Web Application e siti web dinamici mettendo a disposizione un DBMS MySQL integrato e numerosi strumenti amministrativi all'interno di una Dashboard dedicata. Il CMS nasce nel 2003 dopo che Matt Mullenweg e Mike Little decisero di continuare lo sviluppo del CMS b2/cafelog partendo da una sua copia (forking); a Maggio 2021 WordPress risulta essere utilizzato nel **64,8%** dei siti di cui è noto il sistema di gestione dei contenuti. I motivi della sua popolarità comprendono sicuramente la semplicità d'uso, la sicurezza, nonché la possibilità di integrazione con qualsiasi altro tipo di applicazione o servizio.

WordPress da anche la possibilità agli sviluppatori di estendere le sue funzionalità attraverso la creazione di **plugin** scritti in linguaggio PHP che possono essere pubblicati e venduti in un marketplace pubblico. In questo modo chiunque con conoscenze anche basiche di informatica avrà a disposizione un potente strumento per la creazione di applicazioni web basate su plugin sviluppati da terzi, senza la necessità di dover scrivere neanche una linea di codice.

Il sistema di **Session Tracking e Threat Evaluation** si basa su di un plugin WordPress da me realizzato, il quale permette di raccogliere e gestire le informazioni di ogni utente che sta utilizzando uno specifico sito su cui il plugin è installato ed attivo.

Plugin di WordPress

Per lo sviluppo di plugin, WordPress mette a disposizione un'API che permette di interfacciarsi ad ognuna delle componenti del CMS: database, dashboard admin o sito web. Nella sua forma più semplice un plugin è costituito da un singolo file .php che specifica in un header alcune informazioni relative al

plugin stesso, queste verranno interpretate dal CMS al fine di rendere attivabile o disattivabile il plugin; la figura 3.5 presente in un successivo paragrafo mostra la dashboard admin di WordPress da cui è possibile gestire i plugin.

Tra le funzionalità più importanti messe a disposizione dall'API di WordPress individuiamo gli Hooks: quest'ultimi permettono di cambiare il comportamento di default del CMS; in particolare gli Action Hooks danno la possibilità di aggiungere delle funzionalità personalizzate mettendo in esecuzione un task definito dallo sviluppatore, i Filter Hooks invece modificano il contenuto delle pagine di un sito per come sono mostrate all'utente, ad esempio tramite l'aggiunta di un foglio di stile CSS o uno script JavaScript quando determinate condizioni vengono soddisfatte.

Dashboard Amministrativa

Un sito WordPress è sempre affiancato da una Dashboard denominata **wp-admin** che ne permetterà la gestione completa da parte dell'amministratore tramite un interfaccia semplificato; tra le funzionalità più importanti messe a disposizione individuiamo:

- Modifica dello stile delle diverse pagine attraverso l'aggiunta di temi custom anche sviluppati da terze parti.
- Aggiunta di nuove pagine, post, commenti e contenuti multimediali.
- Gestione degli utenti del sito e dei loro permessi.
- Acquisto di plugin da un negozio virtuale.
- Gestione dei singoli plugin tramite delle specifiche dashboard fornite dai loro sviluppatori.

Database API

Dato che il CMS sfrutta un DBMS MySQL per la persistenza dei dati del sito (dashboard, contenuti, utenti, ecc...), WordPress fornisce anche la possibilità di interfacciarsi ad esso tramite un'API per il linguaggio PHP, la quale risulta molto comoda per gli sviluppatori di plugin che desiderano personalizzare la struttura delle tabelle o salvare nuovi dati su Database.

2.2.2 Linguaggio PHP

PHP è un linguaggio di scripting ampiamente utilizzato per lo sviluppo web lato server, creato da Rasmus Lerdorf nel 1994. All'interno del progetto verrà utilizzato per lo sviluppo di un plugin WordPress e per l'implementazione del sistema di valutazione delle minacce.

2.2.3 Linguaggio Java

Java è un linguaggio di programmazione ad alto livello orientato agli oggetti ufficialmente annunciato nel 1995 a SunWorld. Uno dei principi fondamentali del linguaggio è quello di essere eseguibile in un qualunque sistema in seguito ad una singola compilazione; ciò è possibile grazie all'impiego della Java Virtual Machine il cui compito principale è quello di interpretare il bytecode derivante dalla compilazione del codice.

All'interno del progetto Java è stato impiegato per implementare un client utilizzato in fase di test che si occuperà di controllare molteplici istanze di dei Remote Web Driver (2.2.9) sfruttando una libreria messa a disposizione da Selenium.

2.2.4 DBMS MySQL

MySQL è un Database Management System relazionale open-source sviluppato nel 1995 da Michael Widenius. Interfacciandosi ad esso con le API di WordPress, sarà usato per mantenere le informazioni relative alle sessioni e alle richieste effettuate da ogni utente all'interno dell'applicazione web.

2.2.5 Webserver Apache

Apache è un Webserver gratuito e cross-platform sviluppato dalla Apache Software Foundation ed utilizzato da circa il 40% dei siti web in tutto il mondo. All'interno del progetto verranno utilizzate due diverse istanze di Apache: una su cui si troverà il sito WordPress e una per il sistema di valutazione delle minacce.

2.2.6 Honeytoken

Gli **Honeytoken** sono uno strumento di deception utilizzato in ambito CyberSecurity per la rilevazione di intrusioni o di furto di dati all'interno di uno specifico ambiente o sistema.

Nello specifico gli honeytoken vengono descritti come dati che attirano l'attenzione di utenti malevoli ma che in realtà non hanno nessun valore informativo, il loro scopo infatti è proprio quello di agire come esca in modo da rivelare specifiche informazioni relative all'attaccante che sta tentando di accedervi.

Tipologie di Honeytoken

Il concetto di honeytoken per come appena descritto viene realizzato in numerosi modi anche in relazione all'ambiente in cui verranno dispiegati; di seguito mi occuperò di descriverne le principali tipologie e degli scenari in cui risultano estremamente importanti.

- **Database Honeytoken**: Gli honeytoken all'interno di un Database sono implementati come dei record fittizi indistinguibili da quelli reali. L'idea è che questi record non dovrebbero mai essere utilizzati dalla logica dell'applicazione, pertanto la creazione di specifici trigger all'interno del Database può permettere ad un analista della sicurezza di rilevare accessi impropri o fughe di dati.

- **File Honeytoken**: Un honeytoken può anche essere realizzato nella forma di un file appositamente modificato per rivelare specifiche informazioni come l'indirizzo IP, di ogni utente che cercherà di aprirlo.

  Questa tipologia può risultare molto utile all'interno di un contesto aziendale per rilevare fenomeni di spionaggio industriale che conducono alla diffusione di documenti riservati.

- **Webapplication Honeytoken**: All'interno di un'applicazione web è possibile includere tantissimi diversi tipi di honeytoken, tra i più comuni individuiamo: parametri GET e POST, utenti fittizi, URI specifiche e cookies.

Ognuno di questi honeytoken da la possibilità di individuare richieste potenzialmente pericolose effettuate da un utente malevolo che sta tentando di sfruttare vulnerabilità come Cross Site Scripting, Session Hijacking, SQL Injection e tante altre.

Il sistema di valutazione della minaccia che verrà implementato all'interno del progetto sfrutta esattamente questa tipologia di honeytoken per cercare di riconoscere pattern comportamentali riconducibili a quelli di un attaccante.

2.2.7 Docker Compose

Docker Compose è uno strumento di sviluppo che permette di automatizzare il deployment di applicazioni anche complesse, mettendo in esecuzione più container Docker in parallelo.

Le varie componenti dell'applicazione vengono costruite automaticamente tramite un file **YAML** che definisce quali sono i vari servizi che comporranno l'applicazione e per ognuno di essi una serie di configurazioni specifiche; di seguito verranno elencate e spiegate alcune delle principali configurazioni che possono essere presenti all'interno di un file YAML di Docker Compose.

- **version**: definisce la versione di Docker Compose che si sta attualmente utilizzando.
- **build**: sezione che specifica come verrà costruito un servizio, al suo interno può contenere le seguenti tag:
  - **context**: definisce una directory o un repository git all'interno della quale verrà ricercato il Dockerfile utilizzato per la costruzione del container su cui eseguirà il servizio; inserendo anche il tag 'dockerfile' è possibile definire un Docker file caratterizzato da un nome specifico.
  - **image**: permette di costruire un container specificandone l'immagine Docker piuttosto che un Dockerfile.

  - **args**: specifica degli argomenti che potranno essere utilizzati nella fase di build all'interno di un Dockerfile.

- **ports**: definendo delle tuple HOST_PORT:CONTAINER_PORT sarà possibile esporre delle porte del container e mapparle su delle porte dell'host.

- **networks**: all'interno del tag networks è possibile definire una o più reti a cui il container sarà connesso; se il tag non è presente i container faranno parte di un network di default generato da docker.

- **volumes**: permette di mappare volumi della macchina host in volumi del container rendendone visibili i contenuti al servizio in esecuzione.

- **environment**: specifica una o più variabili d'ambiente che potranno essere utilizzate dal servizio nel container

All'interno del progetto, Docker Compose verrà utilizzato per il dispiegamento su container diversi dell'architettura del sistema di Session Tracking e Threat Evaluation.

2.2.8 Elasticsearch e Kibana

Elasticsearch

Elasticsearch è un motore di ricerca open source distribuito rilasciato nel Febbraio del 2010 da Shay Banon come successore del precedente Compass rilasciato nel 2004. All'interno dello Stack Elastic, che comprende un insieme di tool per la raccolta, l'elaborazione e la visualizzazione di qualsiasi tipo di dato, Elasticsearch risulta essere la componente principale poiché si fa carico dell'indicizzazione di dati provenienti da una moltitudine di sorgenti: file di log, applicazioni web e metriche varie di un sistema.

Indici di Elasticsearch

Un indice mantiene in formato **JSON** il contenuto di un insieme di documenti correlati tra loro. Elasticsearch sfrutta una struttura dati denominata *inverted*

index per permettere ricerche testuali estremamente veloci anche in presenza di enormi moli di dati.

Kibana
Una volta indicizzati dei dati su Elasticsearch è possibile sfruttare un altro strumento dello Stack per la loro visualizzazione: **Kibana**. Kibana permette di creare delle Dashboard contenenti grafici di ogni genere (istogrammi real-time, pie charts, line graphs, ecc...) attraverso un interfaccia grafico estremamente semplice da utilizzare.

Il tool da anche la possibilità di sfruttare funzionalità più avanzate per la creazione di grafici personalizzati che si adattano specificamente al tipo e al formato dei dati con cui si sta lavorando.

All'interno del progetto Elasticsearch e Kibana verranno utilizzate per la raccolta, l'indicizzazione e la visualizzazione dei dati relativi alle sessioni e alle richieste effettuate da ogni utente.

2.2.9 Selenium Remote Web Driver

E' un **framework** che permette di controllare da remoto una o più istanze di browser automatizzando e simulando l'interazione tra un utente e una pagina web. Come già descritto nel paragrafo 2.2.3, verranno create molteplici istanze di Remote Web Driver ognuna pilotata dal client Java.

2.3 Progetto Finale

Dopo aver parlato delle diverse tecnologie utilizzate e di come esse verranno utilizzate, mi occuperò ora di illustrare in breve quali sono le varie parti che comporranno il sistema e quale sarà il loro compito (figura 2.1).

Webserver WordPress
E' il Webserver con CMS WordPress che mette a disposizione la Web Application con cui gli utenti potranno interagire.

DBMS MySQL
DBMS utilizzato da WordPress e dai suoi plugin per la persistenza dei dati.

Threat Evaluation System
Sistema che si occupa della valutazione del livello di minaccia relativo alle richieste registrate dal Webserver WordPress. Il Threat Evaluator riceve tali richieste attraverso un'apposita API.

Elasticsearch e Kibana
Elasticsearch salva in degli appositi indici i dati di richieste e sessioni inviati da WordPress e li mette a disposizione di Kibana per la creazione di Dashboard.

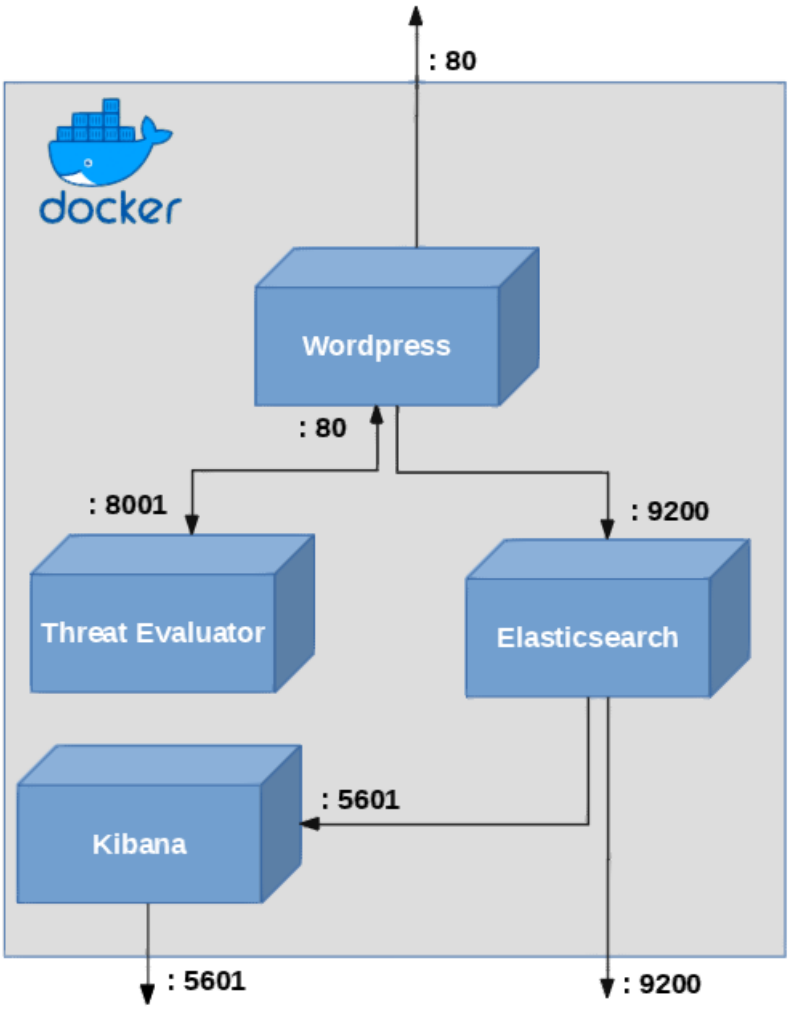

Figura 2.1: Architettura del Sistema

Capitolo 3

Realizzazione del Progetto

Terminata l'analisi e la progettazione del sistema di Session Tracking e Threat Evaluation, verrà ora descritta la parte di realizzazione progettuale che si è articolata nelle seguenti fasi:

1. Realizzazione dell'infrastruttura tramite Docker Compose.
2. Implementazione del Plugin WordPress in PHP.
3. Progettazione del Database MySQL per la persistenza di sessioni e richieste.
4. Implementazione del sistema di valutazione della minaccia basato su Honeytoken.
5. Configurazione del motore di ricerca Elasticsearch.
6. Creazione di Dashboard personalizzate su Kibana.

3.1 Progettazione e Deployment dell'infrastruttura

Nella fasi preliminari all'implementazione è stata progettata un'architettura a **microservizi** che permettesse di dispiegare i vari sistemi all'interno di contai-

ner Docker utilizzando Docker Compose, tecnologia già descritta nella sezione 2.2.7.

Di seguito è possibile visionare il contenuto del file **docker-compose.yml** insieme ad una breve descrizione di ogni sistema presente nella sezione *services*.

```
version: "3.0"

services:
  db:
    image: mysql:5.7
    volumes:
            - ./db_data:/var/lib/mysql:rw
    ports:
      - "3306:3306"
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
            - ./web_data:/var/www/html
    ports:
      - "80:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
```

```
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

  api:
        build: .
        ports:
            - "8001:80"
        volumes:
            - ./www_api:/var/www/html/
        links:
            - db

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.13.0
    container_name: elasticsearch
    environment:
      - node.name=elasticsearch
      - cluster.name=es-docker-cluster
      - cluster.initial_master_nodes=elasticsearch
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - ./elasticsearch:/usr/share/elasticsearch/data
    ports:
      - 9200:9200

  kibana:
    image: docker.elastic.co/kibana/kibana:7.13.0
```

```
container_name: kibana
ports:
  - 5601:5601
environment:
  ELASTICSEARCH_URL: http://elasticsearch:9200
  ELASTICSEARCH_HOSTS:
    '["http://elasticsearch:9200"]'
restart: always
```

- **Servizio WordPress, (wordpress)**: Container con Webserver Apache e CMS WordPress su cui verrà implementata ed eseguita la Web Application. Espone la porta HTTP 80 per essere accessibile dall'esterno e sfrutta alcune variabili d'ambiente con cui il CMS sarà configurato.

- **Servizio MySQL, (db)**: DBMS MySQL a cui la Web Application si interfaccerà per il salvataggio di informazioni relative a sessioni e richieste. Espone la porta 3306 e come il precedente servizio, viene configurato tramite una serie di variabili d'ambiente.

- **Servizio Threat Evaluator, (api)**: Webserver Apache su cui vengono eseguiti una serie di script PHP che tramite una semplice API ricevono delle richieste inoltrate dalla Web Application, valutano il loro livello di minaccia e lo restituiscono indietro come risposta. Il servizio mappa la porta 80 del Webserver sulla porta 8001 del container, quest'ultima sarà la porta che verrà utilizzata dagli altri servizi per instaurare una connessione con il container.

- **Servizio Elasticsearch, (elasticsearch)**: Motore di ricerca che effettua l'indicizzazione di sessioni e richieste; espone la porta 9200 in modo da essere raggiungibile dall'esterno e dai due servizi WordPress e Kibana.

Anch'esso viene configurato con delle variabili d'ambiente standard.

- **Servizio Kibana, (kibana)**: Kibana è il servizio che, sfruttando i dati indicizzati su Elasticsearch, permette di creare Dashboard altamente personalizzate per la visualizzazione di ogni tipo di log. Esso espone verso l'esterno la porta 5601 e sfrutta due variabili d'ambiente che specificano l'indirizzo dell'host elasticsearch di riferimento.

3.2 Progettazione del Database MySQL

WordPress per poter funzionare correttamente necessita anche di un **RDBMS MySQL** su cui appoggiarsi per salvare diversi tipi di dati tra cui utenti, metadati, post e sessioni.

Personalizzazione della struttura del Database
In aggiunta alle tabelle di default utilizzate dal CMS mi sono anche occupato di progettarne delle altre che permettessero al backend PHP di memorizzare dati su sessioni e richieste.

La motivazione di queste personalizzazioni alla struttura del Database deriva dal fatto che, in questo specifico scenario, il meccanismo di gestione delle sessioni utilizzato da WordPress non è sufficiente per implementare tutte le funzionalità che il sistema di Session Tracking e Threat Evaluation si pone di realizzare; pertanto come verrà anche descritto nella sezione 3.3 relativa allo sviluppo del plugin, al fine di raggiungere gli obbiettivi descritti tra i requisiti di sistema (2.1), ho optato per l'implementazione di una gestione di sessioni e richieste altamente personalizzata.

Di seguito vengono mostrati gli schemi concettuali e logici delle tabelle realizzate e le query SQL che permettono di generare tale architettura. In aggiunta a quelle presenti negli schemi, è stata anche realizzata un ulteriore tabella, **blacklist_ip**, il cui unico scopo è quello di mantenere gli indirizzi IP banditi dal sito e la data di inserimento per ciascuno di essi. I dati contenuti in

quest'ultima tabella verranno messi a disposizione del backend dell'applicazione per implementare un meccanismo proattivo di protezione (3.3.5).

Poiché su questa tabella non sarà necessario effettuare alcuna operazione di join, ho deciso di non includerla nelle progettazioni concettuale e logica, effettuate invece per tutte le altre tabelle relative a sessioni e richieste.

Architettura Database per persistenza Sessioni

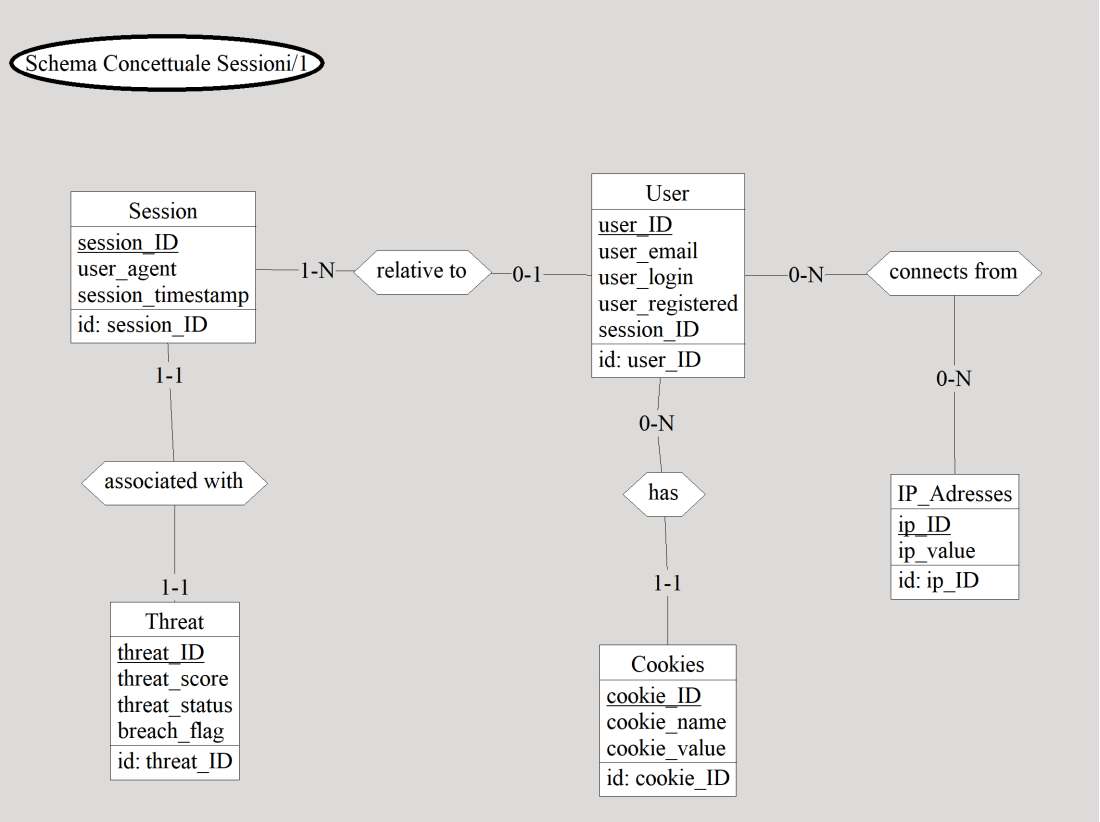

Figura 3.1: Schema Concettuale Sessioni

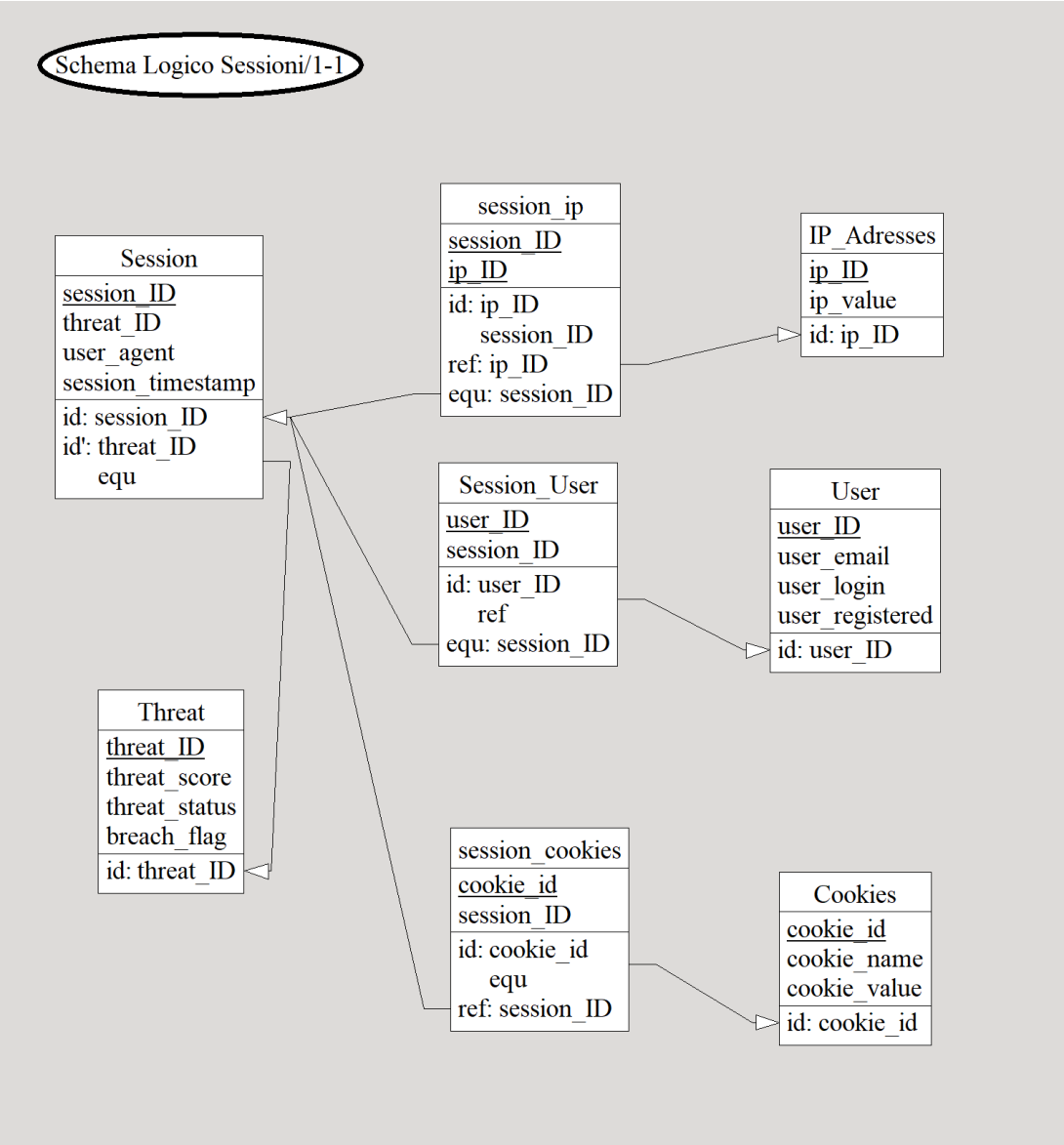

Figura 3.2: Schema Logico Sessioni

E' importante notare che la tabella in entrambi gli schemi denominata *User* fa riferimento alla tabella *wp_users* all'interno della quale WordPress mantiene i dati relativi agli utenti del sito web. In questo modo posso evitare di ridondare informazioni che il CMS si occupa già di gestire in maniera efficiente.

Query per la creazione delle tabelle relative alle sessioni

```
-- Database Section
-- ________________

use wordpress;

-- Tables Section
-- _____________

create table COOKIE (
    cookie_ID int(20) auto_increment not null,
    cookie_name varchar(100) not null,
    cookie_value varchar(200) not null,
    constraint IDCookies_ID primary key (cookie_id));

create table IP_ADDRESS (
    ip_ID int(20) auto_increment not null,
    ip_value varchar(100) not null,
    constraint IDIP_Adresses primary key (ip_ID));

create table SESSION (
    session_ID varchar(100) not null,
    threat_ID int(20) not null,
    user_agent varchar(200) not null,
    session_timestamp timestamp,
    constraint IDSession_ID primary key (session_ID));

create table SESSION_COOKIE (
    cookie_ID int(20) not null,
    session_ID varchar(100) not null,
    constraint FKses_Coo_ID primary key (cookie_ID));

create table SESSION_IP (
```

```
    session_ID varchar(100) not null,
    ip_ID int(20) not null,
    constraint IDsession_ip primary key (ip_ID, session_ID));

create table SESSION_USER (
    user_ID bigint(20) unsigned not null,
    session_ID varchar(100) not null,
    constraint FKrel_Use_ID primary key (session_ID));

create table THREAT (
    threat_ID int(20) auto_increment not null,
    threat_score int not null,
    threat_status varchar(100) not null,
    breach_flag bool not null,
    constraint IDThreat_ID primary key (threat_ID));



alter table SESSION add constraint FKassociated_with_FK
    foreign key (threat_ID)
    references THREAT (threat_ID);

alter table SESSION_COOKIE add constraint FKses_Coo_FK
    foreign key (cookie_ID)
    references COOKIE (cookie_ID);

alter table SESSION_COOKIE add constraint FKses_Ses_cookie
    foreign key (session_ID)
    references SESSION (session_ID);

alter table SESSION_IP add constraint FKses_IP_
    foreign key (ip_ID)
    references IP_ADDRESS (ip_ID);

alter table SESSION_IP add constraint FKses_Ses
```

```
    foreign key (session_ID)
    references SESSION (session_ID);

alter table SESSION_USER add constraint FKrel_Ses
    foreign key (session_ID)
    references SESSION (session_ID);

alter table SESSION_USER add constraint FKrel_Use_FK
    foreign key (user_ID)
    references wp_users (ID);
```

Architettura Database per persistenza Richieste

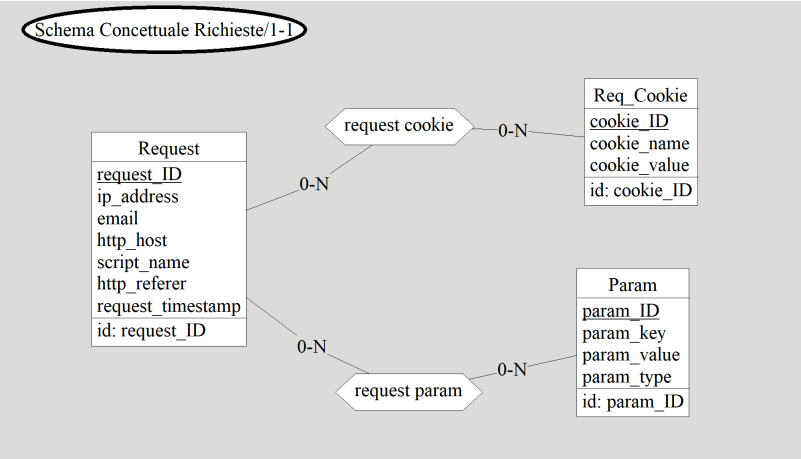

Figura 3.3: Schema Concettuale Richieste

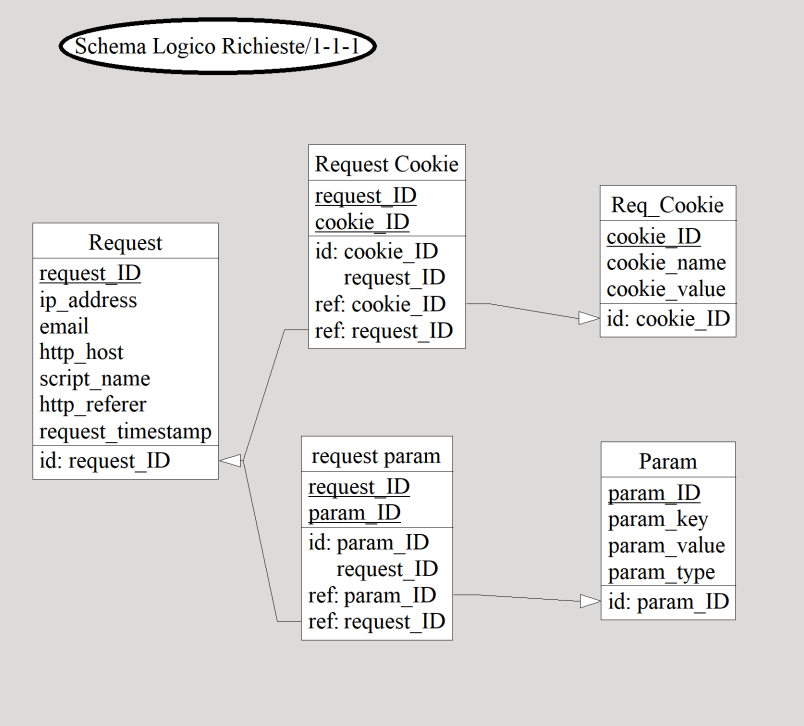

Figura 3.4: Schema Logico Richieste

Query per la creazione delle tabelle relative alle richieste

```
use wordpress;

create table request (
   request_ID int(20) auto_increment not null,
   ip_address varchar(100) not null,
   email varchar(100),
   http_host varchar(100) not null,
   script_name varchar(100) not null,
   http_referer varchar(100),
   request_timestamp datetime not null,
   constraint RequestID primary key (request_ID)
);

create table request_cookie (
   request_ID int(20) not null,
   cookie_ID int(20) not null,
   constraint FKrequest_cookie primary key (request_ID,
      cookie_ID)
);

create table req_cookie (
   cookie_ID int(20) auto_increment not null,
   cookie_name varchar(100) not null,
   cookie_value varchar(200) not null,
   constraint FKcookieID primary key (cookie_ID)
);

create table request_parameter (
   request_ID int(20) not null,
   param_ID int(20) not null,
   constraint FKrequest_cookie primary key (request_ID,
      param_ID)
```

```
);

create table param (
   param_ID int(20) auto_increment not null,
   param_key varchar(100),
   param_value varchar(200),
   param_type varchar(10) not null,
   constraint FKparamID primary key (param_ID)
);


alter table request_cookie add constraint
   FK_request_ID_cookie
   foreign key (request_ID)
   references request (request_ID);

alter table request_cookie add constraint FK_cookie_ID
   foreign key (cookie_ID)
   references req_cookie (cookie_ID);

alter table request_parameter add constraint
   FK_request_ID_param
   foreign key (request_ID)
   references request (request_ID);

alter table request_parameter add constraint FK_param_ID
   foreign key (param_ID)
   references param (param_ID);
```

3.3 Implementazione del Plugin WordPress

In questa parte mi sono dedicato all'implementazione del plugin WordPress che verrà attivato all'interno dell'applicazione web; quest'ultimo si occuperà in particolare di raccogliere e salvare tutti i dati utili al tracciamento delle azioni effettuate da ogni utente. Tali dati verranno poi inoltrati agli altri sotto-sistemi che li utilizzeranno a loro volta per raggiungere i propri scopi.

3.3.1 Plugin per la registrazione di utenti

In aggiunta al plugin sopracitato ho deciso di aggiungerne un altro di nome **User Registration**, installabile dal marketplace di WordPress, che permette agli utenti del sito di crearsi un proprio account ed accedervi. Questo tornerà molto utile per tenere traccia di utenti diversi che accedono con lo stesso account e per la creazione lato amministratore di account fittizi che non dovrebbero essere acceduti da nessun utente (*HoneyUser*)

3.3.2 Inizializzazione del Plugin di Session Tracking

Lo script PHP principale mostrato qui sotto verrà utilizzato per inizializzare il plugin e le varie componenti, come il client per interagire con il Database MySQL o la classe DataLogger che gestirà la raccolta dei dati.

```
<?php

/**
 *
 * @since        1.0.0
 * @package      SessionThreatPlugin
 *
 * @wordpress-plugin
 * Plugin Name:  WordPress Session Tracker
 * Description:  Session Tracker and Threat Evaluator.
```

```
 * Version:       1.0.0
 * Author:        Colotti Manuel Enrique
 * License:       GPL-2.0+
 * License URI:   http://www.gnu.org/licenses/gpl-2.0.txt
 * Text Domain:   session-tracker and threat evaluator
 * Domain Path:   /
 */


if ( ! defined( 'ABSPATH' ) ) {
  die;
}


define( 'SESSION_TRACKER_VERSION', '1.0.0' );


if (file_exists(dirname(__FILE__) .
   '/vendor/autoload.php')) {
  require_once dirname(__FILE__) . '/vendor/autoload.php';
}

define('PLUGIN_PATH', plugin_dir_path(__FILE__));
define('PLUGIN_URL', plugin_dir_url(__FILE__));
define('PLUGIN', plugin_basename(__FILE__));


use Inc\Base\SessionThreatPluginActivate;
use Inc\Base\SessionThreatPluginDeactivate;


function activate_session_threat_plugin(){
  SessionThreatPluginActivate::activate();
}
```

```
function deactivate_session_threat_plugin(){
  SessionThreatPluginDeactivate::deactivate();
}



register_activation_hook( __FILE__,
   'activate_session_threat_plugin' );
register_deactivation_hook( __FILE__,
   'deactivate_session_threat_plugin');

if ( class_exists('Inc\\Init')){
  Inc\Init::register_services();
}
?>
```

L'esecuzione dello script viene effettuata automaticamente dal CMS tramite i due hook *register_activation_hook* e *register_deactivation_hook* ogni qualvolta il plugin viene attivato o disattivato dall'apposito pannello di amministrazione (figura 3.5).

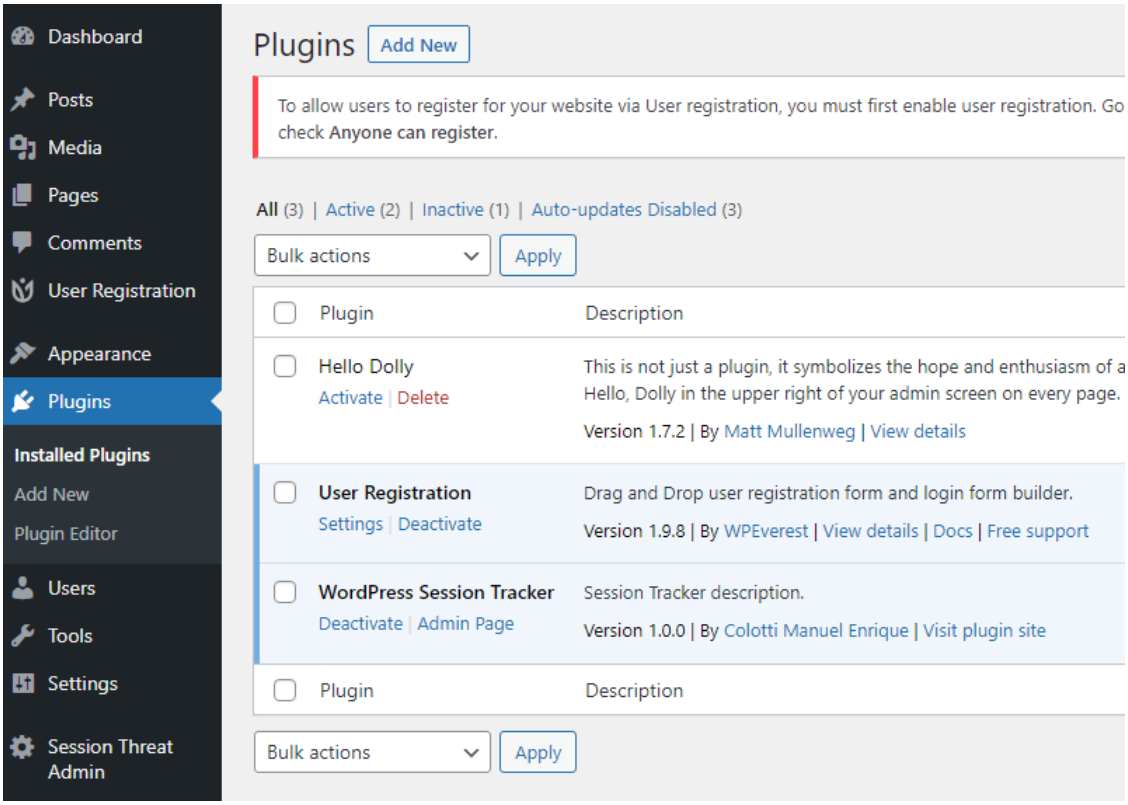

Figura 3.5: Dashboard di gestione dei Plugin WordPress

3.3.3 Raccolta dei dati di sessioni e richieste

All'interno del plugin è implementato uno script di nome **DataLogger** il cui compito è quello di raccogliere una serie di informazioni sulle richieste ricevute che verranno poi messe a disposizione delle altre componenti del sistema. I dati raccolti sull'utente elencati di seguito sono tutti ottenibili sfruttando le funzioni base di PHP e la Database API di WordPress.

- Visitor ID
- Indirizzo IP del mittente
- Email e User_ID dell'utente presenti all'interno del DB di WordPress
- User Agent
- Data e ora di ricezione della richiesta
- Durata della sessione (se l'utente ha effettuato l'accesso in un account)
- Nome dello script richiesto
- URL del Webserver che sta processando la richiesta *(HTTP host)*
- Parametri GET e POST presenti nella richiesta
- HTTP Referer
- Cookie salvati
- Indicatori di minaccia: threat_score, threat_status e breach_flag

Il modo in cui gli indicatori di minaccia sono computati verrà maggiormente approfondita nel paragrafo 3.4 relativo al Threat Evaluator; per adesso è sufficiente indicare che il threat score rappresenta un valore numerico direttamente proporzionale alla pericolosità di un utente.

Di seguito viene riportata la porzione di codice della classe DataLogger incaricata di raccogliere gli altri dati qui sopra elencati:

```
function collect_data(){
   if($this->drop_favicon_request($_SERVER["REQUEST_URI"])){
     return;
   }

   //session data
```

```
        $email = $this->retrieve_email();
        $user_ID = $this->get_user_ID();
        $ip = $_SERVER['REMOTE_ADDR'];
        $user_agent = $_SERVER['HTTP_USER_AGENT'];
        $last_request_datetime = date("c");
        $session_duration = $this->get_session_duration();


        $this->set_session_cookie(rand());
        $this->setup_visitor_cookie($user_agent.$ip);

        $this->check_blacklist_ip($ip);

        //request data
        $script_name = explode('?', $_SERVER['REQUEST_URI'],
           2)[0];
        $http_host = $_SERVER['HTTP_HOST'];
        $request_params = $this->collect_request_params();

        $cookies = $request_params["cookies"];
        $get_params = $request_params["get_params"];
        $post_params = $request_params["post_params"];
        $http_referer = $request_params["http_referer"];
    }
```

Tutti i dati raccolti sono suddivisi in due macro gruppi non disgiunti: campi relativi alle **richieste** e campi relativi alle **sessioni**

- **Richieste**: indirizzo IP, email, cookies, HTTP host, nome dello script, parametri GET, parametri POST, HTTP referer.
- **Sessioni**: visitor ID, indirizzo IP, user agent, data e ora di ricezione della richiesta, email, durata della sessione, cookies di sessione, indicatori di minaccia

Il cookie **Visitor ID** è sicuramente uno dei parametri più importanti tra quelli salvati. Esso è generato come hash della concatenazione di User Agent e Indirizzo IP ed il suo scopo è quello di permettere di tracciare univocamente, con sufficiente precisione, istanze di browser diversi; in questo modo non sarà possibile che due utenti in reti diverse ottengano il medesimo identificativo.

La scelta di questo metodo di web tracking *storage based* deriva dal fatto che nel contesto che stiamo analizzando non sarà necessario implementare un tracciamento univoco di utenti all'interno della stessa rete. Il requisito fondamentale da soddisfare è infatti quello di essere in grado di tracciare con sufficiente precisione utenti che effettuano richieste da posizioni geografiche differenti e di limitare eventualmente quelle pericolose inserendo l'IP da cui provengono all'interno di una lista nera. (3.3.5)

3.3.4 Procedure per il salvataggio dei dati su Database

Una volta raccolti i dati necessari alla registrazione di sessioni e richieste è necessario che questi vengano salvati seguendo specifici criteri per mantenerne la consistenza all'interno del database MySQL integrato in WordPress.

Il plugin da me sviluppato dispone di due diverse classi PHP che si occupano di scegliere quale siano le operazioni più adatte da svolgere sul Database e quindi di realizzarle attraverso delle apposite query eseguibili grazie alla Database API messa a disposizione da WordPress.

Nei due seguenti paragrafi verranno discusse più nel dettaglio le operazioni realizzate da ognuna delle due classi che permettono un salvataggio dei dati di sessioni e richieste in maniera non ridondante.

DBClient: Operazioni su Database

DBClient è la classe PHP del plugin che implementa tutte le query che verranno sfruttate per la creazione, la modifica e la cancellazione di record sul Database; essa sfrutta l'oggetto globale *$wpdb* che espone dei metodi per effettuare in maniera semplice e efficace una qualsiasi delle sopracitate operazioni.

Nella seguente porzione di codice sono mostrate alcune delle query principali implementate in DBClient:

```
class DBClient {
   private static $wpdb;

   public static function register(){
      self::$wpdb = $GLOBALS['wpdb'];
   }

   public static function search_session_id($session_id){
      $sql = "SELECT s.session_ID FROM `session` s WHERE
         s.session_ID = %s";

      $query = self::$wpdb->prepare($sql, $session_id);
      $result = self::$wpdb->get_results($query, ARRAY_A);

      return self::check_search_result($result);
   }

   public static function
      get_session_id_by_user_id($user_id){
      $sql = "SELECT su.session_ID AS session_ID
            FROM `session_user` su
            WHERE su.user_ID = %d";

      $query = self::$wpdb->prepare($sql, $user_id);
      $result = self::$wpdb->get_results($query, ARRAY_A);

      return self::check_select_result($result);
   }

   public static function
      get_all_cookies_by_session($session_ID){
```

```
    $sql = "SELECT sc.cookie_ID AS cookie_ID,
       c.cookie_name AS cookie_name, c.cookie_value AS
       cookie_value
          FROM session_cookie sc, cookie c
          WHERE sc.session_ID = %s AND sc.cookie_ID =
             c.cookie_ID";

    $query = self::$wpdb->prepare($sql, $session_ID);
    $result = self::$wpdb->get_results($query, ARRAY_A);

    return self::check_select_result($result);
}

public static function update_session($session_ID,
    $user_agent, $session_timestamp){
     $sql = "UPDATE session s
           SET s.user_agent = %s, s.session_timestamp = %s
           WHERE s.session_ID = %s";

    $query = self::$wpdb->prepare($sql,
       array($user_agent, $session_timestamp,
       $session_ID));
    $result = self::$wpdb->get_results($query, ARRAY_A);

    return self::check_update_result($result);
}

public static function insert_session($session_ID,
    $threat_ID, $user_agent, $session_timestamp){
    $result = self::$wpdb->insert('session',
       array('session_ID' => $session_ID, 'threat_ID' =>
       $threat_ID, 'user_agent' => $user_agent,
       'session_timestamp' => $session_timestamp));

    return self::check_insert_result($result);
```

```
    }
}
```

DBProcedures: Procedure di gestione dei Dati

DBProcedures usando le operazioni messe a disposizione da DBClient è in grado di inserire nuove informazioni su Database o aggiornare quelle già presenti. Questa classe racchiude al suo interno l'intera logica che, basandosi sui nuovi dati e su quelli già salvati in dei record, permette di determinare come una specifica sessione verrà salvata.

La logica implementata da DBProcedures segue una serie di criteri e regole da me definite per una gestione consistente e non ridondante dei dati relativi alle sessioni; ognuno di questi verrà analizzato per stabilire il ruolo che gioca al fine di ottenere un efficace tracciamento delle sessioni e del livello di minaccia.

- Le sessioni identificate dallo stesso cookie *visitor_id* vengono inserite nel DB solo una volta, questo permette di associare un utente con solo una sessione.

- Quando un utente effettua il login non viene creata una nuova entry per la sessione ma vengono aggiornati i dati già presenti (upgrade di sessione); così facendo si evita ad ogni accesso di generare un nuovo record per ogni singolo utente.

- Allo stesso modo, anche quando un utente effettua il logout non verrà creato un nuovo record ma verranno solamente aggiornate le sue informazioni (downgrade di sessione).

- Per dare la possibilità a più utenti di accedere allo stesso account contemporaneamente è stato necessario decidere come gestire l'integrazione delle diverse sessioni: a questo scopo ho scelto di mantenerle separate ma allo stesso tempo di unificare il loro livello di minaccia.

  Dal momento in cui due utenti accedono allo stesso account il punteggio di minaccia non sarà più valutato a livello di singola sessione ma piuttosto

a livello di account; ciò significa che le azioni potenzialmente pericolose effettuate da uno si rispecchieranno anche nel punteggio dell'altro.

- Per il criterio al punto precedente, quando un utente effettua il login su un account già in uso da qualcun' altro, il punteggio di minaccia dei due viene sommato e il loro *threat_id* diventerà lo stesso (figura 3.6). In questo modo si garantisce che:

  1. Un utente non diminuisca illegalmente il proprio livello di minaccia accedendo ad un account con punteggio minore.
  2. Il livello di minaccia sia gestito a livello account come precedentemente descritto.

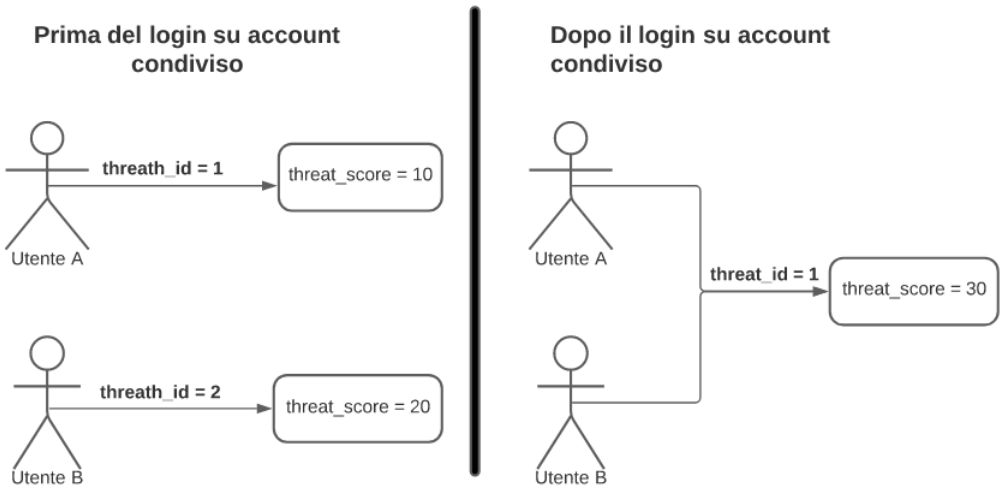

Figura 3.6: Login di due utenti su account condiviso

- Simmetricamente al criterio precedente, se un utente effettua il logout da un account condiviso sarà nuovamente necessario aggiornare il suo *threat_id* in modo che il suo livello di minaccia e quello degli altri utilizzatori dell'account non sia più in comune.

Pertanto nell'esatto momento in cui viene effettuato il logout da tale profilo, all' utente verrà associato un nuovo record che identifica la minaccia; quest'ultimo verrà popolato con lo stesso threat score rilevato per quell'utente prima del logout(figura 3.7).

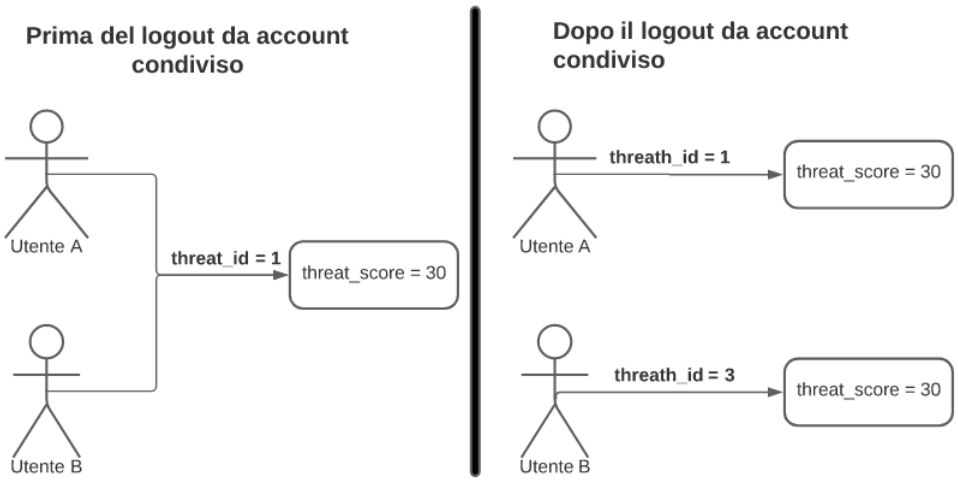

Figura 3.7: Logout di due utenti da account condiviso

Agendo secondo questa logica si impedisce la perdita di consistenza degli indicatori di minaccia; senza questo meccanismo sarebbe banale per un utente effettuare il logout da un account solo per poter azzerare il proprio threat score.

- Un altro criterio non legato alla gestione della minaccia riguarda il modo in cui viene affrontato l'inserimento e la rimozione di record relativi ai cookie all'interno del database: per limitare al minimo le ridondanze ho fatto in modo che i cookie relativi ad una sessione vengano aggiornati dinamicamente a seconda se sono ancora presenti o meno nel browser dell'utente; in particolare:

  – Se sul DB sono presenti dei record relativi a cookie scaduti, questi verranno rimossi.

– Se il valore di dei cookie viene per qualche motivo cambiato, tale aggiornamento verrà replicato anche su DB in modo da non creare record duplicati per lo stesso cookie.

Dopo aver esposto tutti i criteri seguiti da DBProcedures per la persistenza dei dati su Database, mostrerò ora il codice relativo ad alcune delle procedure più importanti implementate; un particolare rilevante è l'esecuzione di ognuna delle procedure all'interno di una **transazione**, questo permetterà un ritorno allo stato originale del database in caso una o più query SQL falliscano.

- **choose_action**: è l'unico metodo pubblico della classe, esso viene invocato dalla classe DataLogger e ha il compito di decidere quale sia l'operazione più opportuna da effettuare sui dati ricevuti.

```
public static function choose_action($data){
      if(!DBClient::search_session_id($data["session_id"])){

         if(is_null($data["user_id"])){
            //create new session
            self::create_session($data);

         }
         else{
            //recover session_id associated with user_id
            if(DBClient::compare_session_id_by_user_id(
               $data["session_id"], $data["user_id"]
              )){
               if(self::check_session_consistency($data)){
                  self::update_session($data);
               }
            }
            else{
               //user_id not associated with any
                  session, create new session.
```

```
                self::create_session($data);
                self::update_threat($data);
            }
        }
    }
    else{
        //update existent session
        self::update_session($data);
        $result = DBClient::get_threat_data_by_id(58);
    }

    return
        self::compute_updated_session($data['session_id']);
}
```

- **create_session**: è un metodo che permette di inserire su DB tutte le informazioni relative ad una nuova sessione e viene invocato ogni qualvolta il cookie identificativo di un utente non sia già presente all'interno del database.

```
private static function create_session($data){
    DBClient::start_transaction();
    DBClient::disable_foreign_checks();

    $threat_ID =
        DBClient::insert_threat($data["threat_score"],
        $data["threat_status"], $data["breach_flag"]);
    if(is_null($threat_ID)){
        DBClient::rollback_transaction();
        return;
    }

    $session_index =
```

```
        DBClient::insert_session($data["session_id"],
        $threat_ID, $data["user_agent"],
        $data["session_duration"],
        $data["last_request_datetime"]);
    if(is_null($session_index)){
        DBClient::rollback_transaction();
        return;
    }

    $cookie_insertion_result =
        self::multiple_cookie_insertion($data["cookie"],
        $data["session_id"]);
    if(is_null($cookie_insertion_result)){
        DBClient::rollback_transaction();
        return;
    }

    $bind_session_user_result =
        self::bind_session_and_user($data["session_id"],
        $data["user_id"]);
    if(is_null($bind_session_user_result)){
        DBClient::rollback_transaction();
        return;
    }


    $ip_ID =
        DBClient::insert_ip_address($data["ip_address"]);
    if(is_null($ip_ID)){
        DBClient::rollback_transaction();
        return;
    }

    $session_ip_result =
        DBCLient::insert_session_ip($data["session_id"],
```

```
            $ip_ID);
        if(is_null($session_ip_result)){
            DBClient::rollback_transaction();
            return;
        }

        DBClient::commit_transaction();
        DBClient::enable_foreign_checks();
    }
```

- **update_session**: nel caso l'identificativo dell'utente corrisponda con uno già salvato su DB, questo metodo verrà invocato da choose_action e si occuperà di aggiornare come descritto nei vari criteri, il record relativo all'utente.

```
private static function update_session($data){
        DBClient::start_transaction();
        DBClient::disable_foreign_checks();

        $page_loads =
            DBClient::get_page_loads($data["session_id"]);
        $session_update_result =
            DBClient::update_session($data["session_id"],
            $data["user_agent"],
            $data["session_duration"],
            $data["last_request_datetime"], $page_loads +
            1);

        if(is_null($session_update_result)){
            DBClient::rollback_transaction();
            return;
        }
```

```
$threat_ID =
   DBClient::get_threat_by_session($data["session_id"]);
if(is_null($threat_ID)){
   DBClient::rollback_transaction();
   return;
}

$threat_update_result =
   self::compute_score($threat_ID,
   $data["threat_score"], $data["breach_flag"]);
if(is_null($threat_update_result)){
   DBClient::rollback_transaction();
   return;
}

if(is_null(self::add_new_ip_address($data["session_id"],
   $data["ip_address"]))){
   DBClient::rollback_transaction();
   return;
}

if(is_null(self::add_new_cookies($data["cookie"],
   $data["session_id"]))){
   DBClient::rollback_transaction();
   return;
}

if(is_null(self::update_session_user($data))){
   DBClient::rollback_transaction();
   return;
}

DBClient::commit_transaction();
DBClient::enable_foreign_checks();
```

```
    }
```

Diversamente da ciò che avviene per le sessioni, le richieste non necessitano di alcuna modifica prima di essere salvate; una volta aggregati i dati da cui sono costituite esse vengono direttamente inserite all'interno del Database attraverso la seguente procedura:

```
public static function create_request($request_data){
      DBClient::start_transaction();

      $request_ID = DBClient::insert_request($request_data);
      if(is_null($request_ID)){
         DBClient::rollback_transaction();
         return;
      }

      //insert cookies
      if(!is_null($request_data["cookies"]) > 0){
         foreach($request_data["cookies"] as $cookie_name
            => $cookie_value){
            $result =
               DBClient::insert_request_cookie($request_ID,
               $cookie_name, $cookie_value);

            if(is_null($result)){
               DBClient::rollback_transaction();
               return;
            }
      }
      }

      //insert params
      if(!is_null($request_data["get_params"]) > 0){
```

```
            foreach($request_data["get_params"] as $param_key
                => $param_value){
                $result =
                    DBClient::insert_request_params($request_ID,
                    $param_key, $param_value, "GET");

                if(is_null($result)){
                    DBClient::rollback_transaction();
                    return;
                }
            }
        }

        if(!is_null($request_data["post_params"]) > 0){
            foreach($request_data["post_params"] as $param_key
                => $param_value){
                $result =
                    DBClient::insert_request_params($request_ID,
                    $param_key, $param_value, "POST");

                if(is_null($result)){
                    DBClient::rollback_transaction();
                    return;
                }
            }
        }

        DBClient::commit_transaction();
    }
```

3.3.5 Limitazione di Utenti Malevoli

Al fine di rendere effettivamente utile il punteggio di minaccia registrato per ogni utente, ho sviluppato un meccanismo **proattivo** di protezione che permette di limitare le azioni effettuate sulla Web Application da utenti con threat score molto alto.

Per raggiungere tale scopo ho implementato lo script **BlacklistController.php** che realizza le seguenti funzionalità:

- **Autorizzazione degli utenti:** viene ricercato l'IP dell'utente in una blacklist salvata su DB; se l'indirizzo non è tra quelli in blacklist l'utente viene autorizzato e può navigare normalmente, se viene invece trovata una corrispondenza, il BlacklistController decide se autorizzare o meno l'utente in relazione alla data in cui l'IP è stato inserito in lista nera. Una volta passate 24 ore dall'inserimento di un indirizzo in blacklist, esso viene rimosso e gli indicatori di minaccia degli utenti associati a tale indirizzo resettati.

- **Limitazione degli utenti:** il BlacklistController recupera il threat score di un utente e determina se inserire il suo indirizzo IP in lista nera. La soglia attuale impostata prevede la limitazione di utenti con threat score superiore ai **2000 punti**.

3.3.6 Log dei dati su Elasticsearch

All'interno del plugin WordPress viene anche implementato lo script **Logging.php** esclusivamente dedicato al log di richieste e sessioni su Elasticsearch; esso sfrutta una libreria messa a disposizione dal motore di ricerca che, tramite un client, permette l'indicizzazione di questi dati.

3.3.7 Log dei dati su file JSON

In aggiunta alla persistenza dei dati su Database e su Elasticsearch mi sono anche occupato di implementare il salvataggio delle richieste ricevute dal Webserver in un file **JSON** memorizzato lato server. Inizialmente anche le

sessioni venivano salvate in un file JSON dopo essere state modificate al fine di mantenerle conformi agli stessi criteri di consistenza usati nel DB; tuttavia ho deciso di rimuovere questa funzionalità a causa delle numerose letture e scritture che avrebbero inevitabilmente rallentato il backend dell'applicazione e reso non scalabile l'intero sistema.

Il salvataggio delle richieste invece, non richiedendo la lettura dei dati già scritti su file JSON ma solo di delle scritture sequenziali, risulta essere computazionalmente molto più semplice.

Sebbene il file di log delle richieste (**requests.json**) non venga utilizzato attivamente nel progetto, potrebbe comunque risultare estremamente interessante per sviluppi futuri come:

- Creazione di uno o più dataset per addestrare modelli di **Machine Learning** volti a classificare richieste in sicure ed insicure.
- Visualizzazione delle richieste in un'apposita **Dashboard WordPress** creata ad-hoc.

3.4 Implementazione del sistema di Threat Evaluation

Un altro sottosistema fondamentale del progetto è sicuramente il **Threat Evaluator**: esso è in esecuzione su un Webserver diverso da quello utilizzato dall'applicazione web e si occupa di esporre un'interfaccia su cui ricevere i dati delle singole richieste, valutarne il livello di minaccia ed infine restituire il risultato della valutazione al Webserver WordPress.

Lo script PHP mostrato di seguito rappresenta l'interfaccia del Threat Evaluator che permette all'applicazione WordPress di inoltrare le richieste di connessione ricevute dagli utenti.

```
$request_json_data = file_get_contents('php://input');
$request_data = json_decode($request_json_data, true);
```

```
$wordlist_paths = array("wordlists/blacklist_get_keys.txt",
                  "wordlists/blacklist_get_values.txt",
                  "wordlists/blacklist_post_keys.txt",
                  "wordlists/blacklist_post_values.txt",
                  "wordlists/blacklist_user.txt",
                  "wordlists/file_extensions.txt",
                  "wordlists/page_names.txt",
                  "wordlists/whitelist_cookie_keys.txt");

$evaluator = new Evaluator($wordlist_paths);
$threat_score = $evaluator->analyze_request($request_data);
$breach_flag = $evaluator->get_breach_flag();


echo json_encode(compute_evaluation_result($threat_score,
   $request_data, $breach_flag));


function compute_evaluation_result($threat_score,
   $request_data, $breach_flag){
   $location = geolocate_ip($request_data["ip_address"]);

   if(is_null($location)){
      return array("threat_score" => $threat_score,
         "breach_flag" => $breach_flag, "location" => null);
   }
   else{
      return array("threat_score" => $threat_score,
         "breach_flag" => $breach_flag, "location" =>
         array("lat" => $location["lat"], "lon" =>
         $location["lon"]));
   }

}
```

3.4.1 Valutazione del livello di minaccia tramite Honeytoken

Ho deciso di rappresentare il **livello di minaccia** di un utente con dei valori nel continuo che vanno da 0 a salire; per aumentare l'interpretabilità di tale punteggio ho anche introdotto il parametro *threat_status*: esso indica a quale categoria di pericolosità l'utente appartiene tra le seguenti:

1. **safe_user**: punteggio compreso tra 0 e 499 punti
2. **warning_user**: punteggio compreso tra 500 e 1999 punti
3. **dangerous_user**: punteggio uguale o superiore ai 2000 punti
4. **evil_user**: punteggio qualsiasi ma *breach_flag*, descritta in seguito, settata a *true*

La computazione del punteggio di minaccia avviene attraverso l'analisi dei diversi parametri presenti nelle richieste HTTP, in particolare quelli utilizzati dal valutatore sono:

- **Parametri** GET e POST.
- **Nome** della pagina richiesta.
- **Cookie** del sito presenti nel browser.
- **Utente** con cui è stato effettuato il login.

Per ognuno di questi parametri sono state create lato server delle liste di termini o Honeytoken che non dovrebbero comparire nelle richieste effettuate da un utente poiché potenzialmente dannosi; Un esempio di ciò sono parametri GET o POST inseriti appositamente per individuare delle vulnerabilità come **Cross Site Scripting** e **SQL Injection**, oppure nomi di pagine riservate, cookie inseriti manualmente dall'utente ed infine utenti che sono stati esplicitamente banditi da un amministratore.

La presenza di uno o più termini proibiti tra i parametri della richiesta si riflette sul punteggio calcolato dal valutatore tramite la seguente politica:

- Per ogni parametro GET o POST presenti in blacklist viene aggiunto al threat score **+10**, chiavi e valori sono valutati tramite due file diversi pertanto ad un parametro come "*admin=admin*" sarà attribuito **+20**, mentre ad uno come "*admin=1*" solo +10 (parola *"admin"* in lista nera).

- Ogni volta che si cerca di accedere ad una pagina in lista nera vengono sommati **20 punti** al threat score; in aggiunta al nome dello script viene tenuto anche conto dell'estensione del file richiesto, la presenza di tale in blacklist aggiunge ulteriori **10 punti**.

- I termini nel file **whitelist_cookie_keys** rappresentano una whitelist delle chiavi relative ai cookie che dovrebbero essere presenti; se la somma totale dei cookie inviati dall'utente è superiore o inferiore del numero di quelli in lista bianca, vengono aggiunti **+25 punti** per ognuno in eccesso o difetto. Per rendere più facile l'aggiunta delle chiavi in whitelist ho anche implementato la possibilità di usare la **wildcard "*"**, in questo modo soltanto i caratteri prima della wildcard verranno considerati per determinare se il cookie è ammissibile o meno. **Esempio**: il termine *wordpress** in whitelist indica che tutti i cookie la cui chiave inizia per wordpress saranno ammessi.

- Il file **blacklist_user** specifica un insieme di email di account fittizi che essendo tali non dovrebbero mai essere acceduti da un utente dell'applicazione web. Il login su anche solo uno di questi account indicherebbe inevitabilmente una fuga di dati e pertanto, oltre ad essere aggiunti **+500 punti** al threat score, viene anche settato un parametro *breach_flag* che etichetta l'utente come estremamente pericoloso.

Di seguito viene mostrato il codice dell'Evaluator che rende possibile il calcolo del threat score:

```
public function analyze_request($request){
      $get_score = 0;
```

```
$post_score = 0;
$cookies_score = 0;
$script_score = 0;
$honeyuser_score = 0;

if($this->wordlist_sizes["blacklist_get_keys"] &&
   $this->wordlist_sizes["blacklist_get_values"] &&
   !empty($request["get_params"]) &&
   count($request["get_params"]) > 0){
  $get_score =
     $this->analyze_get_params($request["get_params"]);
}

if($this->wordlist_sizes["blacklist_post_keys"] &&
   $this->wordlist_sizes["blacklist_post_values"] &&
   !empty($request["post_params"]) &&
   count($request["post_params"]) > 0){
   $post_score =
      $this->analyze_post_params($request["post_params"]);
}

if($this->wordlist_sizes["whitelist_cookie_keys"] &&
   !empty($request["cookies"]) &&
   count($request["cookies"]) > 0){
   $cookies_score =
      $this->analyze_cookies($request["cookies"]);
}

if($this->wordlist_sizes["blacklist_user"] &&
   !empty($request["email"])){
   $honeyuser_score =
      $this->analyze_honeyuser($request["email"]);

   if($honeyuser_score > 0){
      $this->breach_flag = true;
```

```
        }
    }

    if($this->wordlist_sizes["file_extensions"] &&
        $this->wordlist_sizes["page_names"]){
        $script_score =
            $this->analyze_script($request["script_name"]);
    }

    return $get_score + $post_score + $cookies_score +
        $script_score + $honeyuser_score;
}

public function get_breach_flag(){
    return $this->breach_flag;
}

private function analyze_get_params($get_params){
    $key_score =
        count(array_intersect(array_keys($get_params),
        $this->wordlists["blacklist_get_keys"])) * 10;
    $value_score = count(array_intersect($get_params,
        $this->wordlists["blacklist_get_values"])) * 10;

    return $key_score + $value_score;

}

private function analyze_post_params($post_params){
    $key_score =
        count(array_intersect(array_keys($post_params),
        $this->wordlists["blacklist_post_keys"])) * 10;
    $value_score = count(array_intersect($post_params,
        $this->wordlists["blacklist_post_values"])) * 10;
```

```
        return $key_score + $value_score;
    }

    private function analyze_cookies($cookies){
        $over_cookies = array_diff(array_keys($cookies),
            $this->wordlists["whitelist_cookie_keys"]);
        $score = count($over_cookies);

        $score -=
            $this->parse_cookies_wildcards($over_cookies);

        return $score <= 0 ? 0 : $score * 25;
    }

    private function analyze_script($script_name){
        $path_array = explode("/", $script_name);
        $script_filename = $path_array[count($path_array) -
            1];

        foreach($this->wordlists["page_names"] as $page){
            if($script_filename == $page){
                return 20;
            }

            foreach($this->wordlists["file_extensions"] as
                $ext){
                if($script_filename == $page.$ext){
                    return 10;
                }
            }
        }

        return 0;
    }
```

```
private function analyze_honeyuser($user){
   foreach($this->wordlists["blacklist_user"] as
      $blacklisted_account){
      if($user == $blacklisted_account){
         return 500;
      }
   }

   return 0;
}

private function parse_cookies_wildcards($over_cookies){
   $wildcards_found = 0;

   foreach($this->wordlists["whitelist_cookie_keys"] as
      $cookie_key){
      if (substr($cookie_key, -1) == '*') {
         $cookie_length = strlen($cookie_key) - 2;
         foreach($over_cookies as $over_cookie){
    $over_cookie_substr = substr($over_cookie,
       $cookie_length);
            if($over_cookie_substr &&
               strcmp($over_cookie_substr, $cookie_key)){
    $wildcards_found += 1;
            }
         }
      }
   }

   return $wildcards_found;
}
```

3.4.2 Geolocalizzazione tramite indirizzo IP

In aggiunta alla valutazione della minaccia il Treath Evaluator si occupa anche di individuare e restituire le coordinate geografiche approssimative degli utenti in relazione al loro indirizzo IP. Per fare ciò viene utilizzato il Database **MaxMind** *GeoLite2-City.mmdb* strutturato come un albero binario che permette di ricavare tali coordinate attraverso delle ricerche estremamente veloci.

Il codice PHP che permette di utilizzare tale Database memorizzato su file è il seguente:

```
use GeoIp2\Database\Reader;

function geolocate_ip($ip_address){
   $reader = new Reader('geoip/GeoLite2-City.mmdb');

   try {
      $record = $reader->city($ip_address);

   } catch (GeoIp2\Exception\AddressNotFoundException $e) {
      return null;
   }

   $latitude = $record->location->latitude;
   $longitude = $record->location->longitude;

   $location = array("lat" => $latitude, "lon" =>
      $longitude);

   return $location;
}
```

3.4.3 Considerazioni sul Threat Evaluator

La scelta di dispiegare il Treath Evaluator in un Webserver dedicato diverso da quello della Web Application deriva dalla volontà di voler testare l'integrazione di più sistemi tra loro attraverso anche l'implementazione di una semplice API per il Threat Evaluator.

In uno scenario reale sarebbe sicuramente stato necessario o dispiegare il Threat Evaluator in cloud (**Multitenancy**) o decentralizzarlo e replicarlo per ogni istanza di WordPress piuttosto che mantenerlo su di un singolo server; entrambe le opzioni permetterebbero agli utenti che utilizzano il plugin di personalizzare le metriche di valutazione per ognuno dei parametri descritti nella sezione 3.4.1.

3.5 Elasticsearch

Come già introdotto nella sezione 3.3.6, ho anche optato per l'indicizzazione di sessioni e richieste all'interno del motore di ricerca **Elasticsearch**; quest'ultimo mi ha permesso di creare due indici separati su cui definire altrettanti mapping in grado di descrivere il formato dei dati che verranno memorizzati.

Di seguito verranno mostrate le query Elasticsearch utilizzate per la creazione di ciascuno dei due indici insieme ai loro rispettivi mapping.

Indice delle sessioni

PUT /sessions

PUT /sessions/_mapping

```
{
  "properties":{
    "breach_flag":{
      "type":"boolean"
    },
```

```
"session_ID":{
  "type":"keyword"
},

"email":{
  "type":"keyword"
},

"ip_addresses":{
  "type":"ip"
},

"last_request_datetime":{
  "type":"date"
},

"page_loads":{
  "type":"integer"
},

"wp_session_cookie":{
  "type":"keyword"
},

"session_duration":{
  "type":"date",
  "format":"strict_hour_minute_second"
},

"threat_score":{
  "type":"integer"
},
```

```
    "threat_status":{
      "type":"keyword"
    },

    "timestamp":{
      "type":"date"
    },

    "user_agent":{
      "type":"keyword"
    }
  }
}
```

Indice delle richieste

PUT /requests

PUT /requests/_mapping

```
{
  "properties":{
    "get_params":{
      "type":"flattened"
    },

    "cookies":{
      "type":"flattened"
    },
```

```
    "http_host":{
      "type":"keyword"
    },

    "email":{
      "type":"keyword"
    },

    "ip_address":{
      "type":"ip"
    },

    "http_referer":{
      "type":"keyword"
    },

    "location":{
      "type":"geo_point"
    },

    "post_params":{
      "type":"flattened"
    },

    "script_name":{
      "type":"keyword"
    },

    "timestamp":{
      "type":"date"
    }
  }
}
```

3.6 Creazione di Dashboard su Kibana

La sola indicizzazione dei dati su Elasticsearch risulta essere fine a se stessa se non si prevede di impiegare delle tecnologie per una loro corretta visualizzazione. Questo risulta essere lo scenario d'integrazione adatto per un software come **Kibana**; esso è in grado di lavorare in simbiosi con Elasticsearch dando la possibilità di creare delle Dashboard ad-hoc per la rappresentazione dei dati in un linguaggio più interpretabile rispetto al formato JSON utilizzato di default dal motore di ricerca.

Il contenuto informativo dei dati rappresentati graficamente potrà poi essere impiegato per diversi scopi:

- **Analisi Statistiche:** pagine più visitate, trend delle richieste effettuate in un intervallo di tempo arbitrario, geolocalizzazione degli utenti, ecc...
- **Analisi Comportamentale**: Presta particolare attenzione agli utenti che sono stati etichettati come potenzialmente minacciosi in relazione al loro threat score e permette di seguirne i movimenti all'interno del sito web per determinare se rappresentano effettivamente un rischio per la sicurezza.

Le immagini che vengono mostrate di seguito rappresentano alcune delle Dashboard realizzate su Kibana per l'aggregazione dei dati di sessioni e richieste salvati su Elasticsearch:

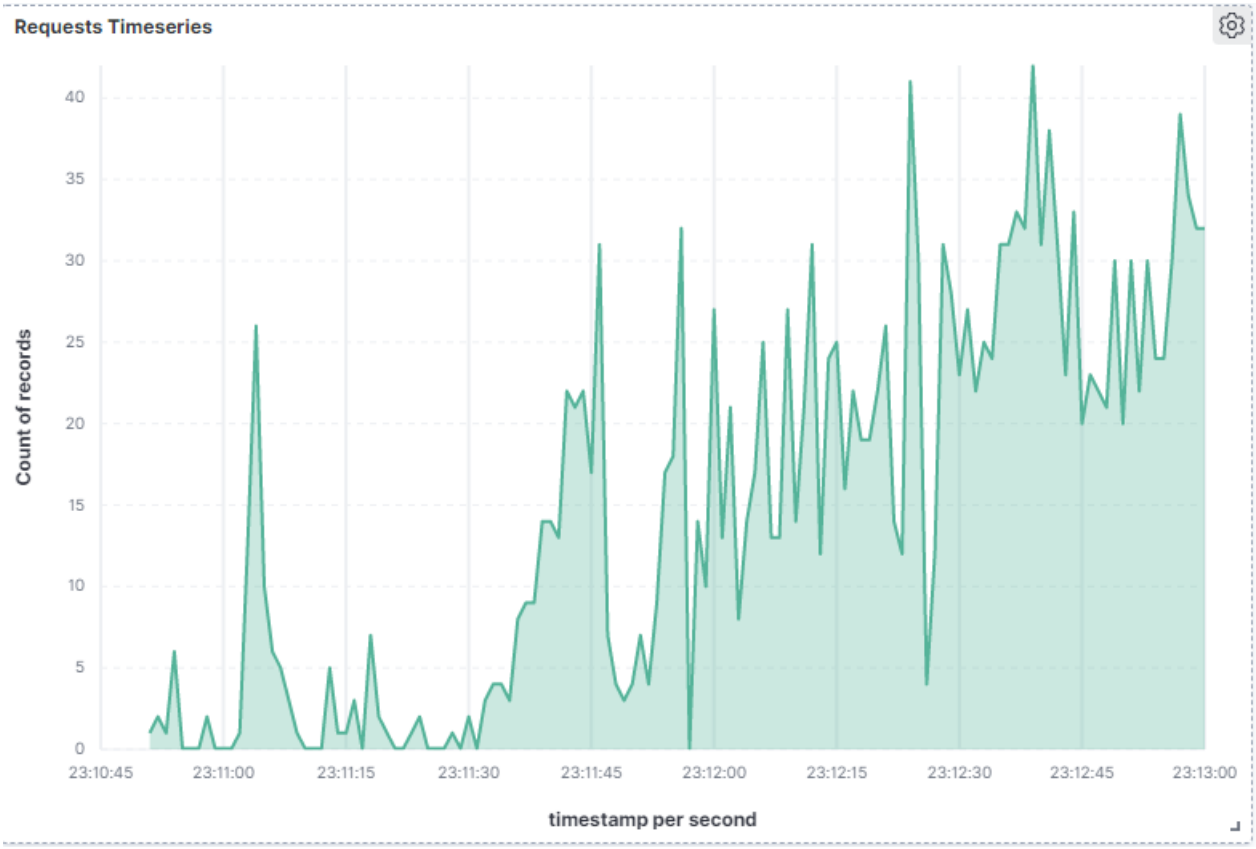

Figura 3.8: Serie temporale delle richieste effettuate al sito

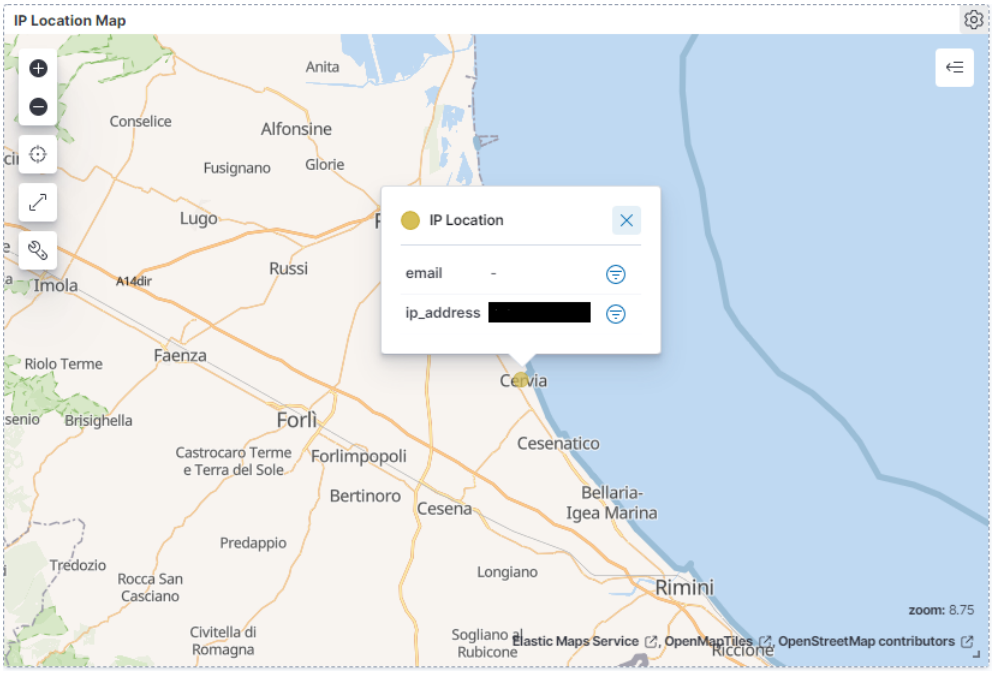

Figura 3.9: Geolocalizzazione degli IP registrati

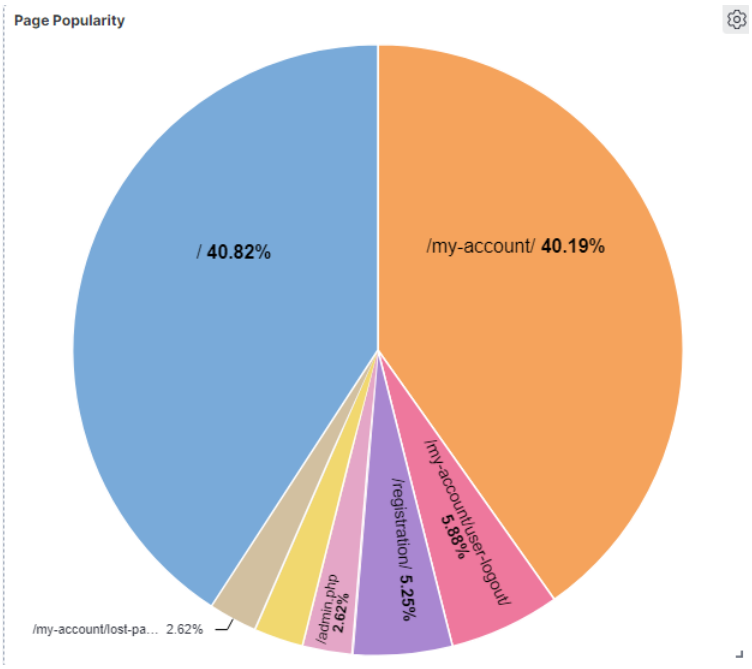

Figura 3.10: Pagine del sito più richieste

Session ID	IP Address	Threat Score	Threat Status	Breach Flag	Email	Session Duration
43567581a715836...	172.29.0.90	9,910	evil_user	true	-	00:00:00
4ac03e1faa855f43...	172.29.0.40	9,910	evil_user	true	-	00:00:00
4ca6575ed09a76e...	172.29.0.84	9,885	evil_user	true	-	00:00:00
698269bba32c10b...	172.29.0.14	9,885	evil_user	true	-	00:00:00
67f5037f0956427a...	172.29.0.54	9,860	evil_user	true	executor_bad47@t...	00:00:32
b26e80b294a29f5...	172.29.0.20	9,860	evil_user	true	executor_bad13@t...	00:00:35
d58d8c523988dfa...	172.29.0.92	9,860	evil_user	true	executor_bad85@t...	00:00:34
0462a11a20ff204e...	172.29.0.106	9,335	evil_user	true	executor_bad99@t...	00:00:31
ec145d76cb1ecd5...	172.29.0.30	9,335	evil_user	true	executor_bad23@t...	00:00:32
0d754f18692b839...	172.29.0.80	8,810	evil_user	true	executor_bad73@t...	00:00:29
212d3b86985f8c3...	172.29.0.44	8,810	evil_user	true	executor_bad37@t...	00:00:29

Figura 3.11: Dati di sessione in relazione al session ID degli utenti

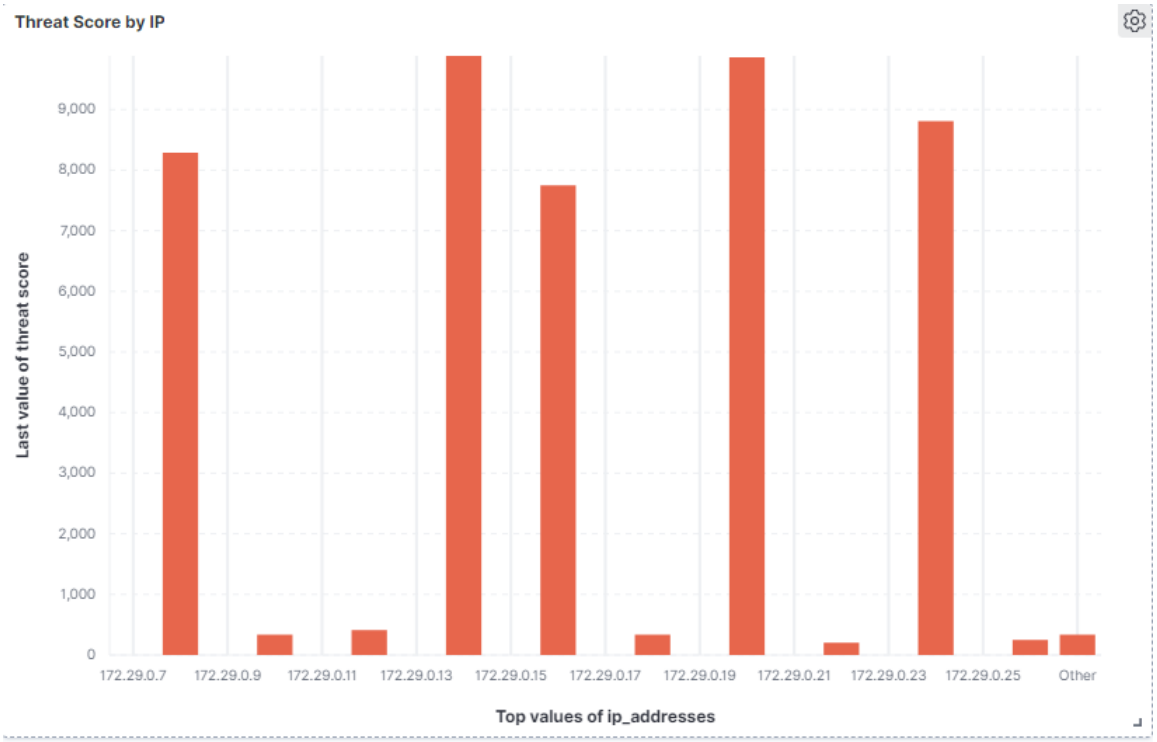

Figura 3.12: Threat Score in relazione all'indirizzo IP

Capitolo 4

Testing del Sistema Finale

In questo capitolo conclusivo mi sono occupato di descrivere il processo che mi ha portato all'implementazione di un **test** per il sistema e di analizzare i risultati ottenuti in seguito alla sua esecuzione.

Lo scopo principale di questo test era quello di verificare il comportamento del sistema in seguito al suo utilizzo da parte di molteplici utenti che effettuano ricerche di vario genere.

4.1 Progettazione e Sviluppo del Test

Lo sviluppo del test può essere suddiviso in due macro fasi che verranno approfondite nei seguenti paragrafi:

1. **Deployment dei Remote Web Drivers**: esecutore il cui compito è quello di navigare all'interno della Web Application.
2. **Implementazione del Controller**: Client Java che pilota i Remote Web Drivers specificando che azione devono effettuare.

4.1.1 Deployment dei Remote Web Drivers

Come già introdotto, i Remote Web Drivers in questo scenario rappresentano degli utenti che navigano tramite browser Chrome all'interno del sito; ciò è possibile grazie alle API messe a disposizione da Selenium che permettono di

controllare ogni loro azione da remoto sfruttando linguaggi come Java, Python, Javascript, Kotlin, ecc...

Per poter simulare più di un utente ho deciso di dispiegare sullo stesso server su cui è in esecuzione il sistema, diversi container docker che sfruttano l'immagine standard *selenium/standalone-chrome-debug* messa a disposizione da Selenium su Docker Hub.

I seguenti script bash sono stati rispettivamente utilizzati per realizzare il deployment dei container e la loro successiva distruzione:

```
#!/bin/bash
#Dispiegamento dei RemoteWebDrivers

for X in $(seq 100)
do
      docker run -d -p$((50000+$X)):4444 -p
         $((50200+$X)):5900 --shm-size=2g
         --network=webapplication_default
         --name=seleniumTest$X
         selenium/standalone-chrome-debug
done
```

```
#!/bin/bash
#Rimozione dei RemoteWebDrivers

for X in $(seq 100)
do

docker stop seleniumTest$X
docker rm seleniumTest$X

done
```

4.1.2 Implementazione del Controller

Suddivisione in Thread

Per poter gestire le azioni di più Remote Web Drivers contemporaneamente, ho implementato una classe **RemoteExecutor** la quale estende la classe Thread e incapsula una singola istanza di Web Driver.

Ognuno di questi **Thread** è dotato di una coda che gli permetterà di stabilire la prossima azione che la propria istanza di Web Driver dovrà eseguire sul Browser.

La seguente porzione di codice rappresenta una parte della classe RemoteExecutor:

```
public class RemoteExecutor extends Thread {
   private static final int SLEEP_TIME = 1000;
   private final String ip_address;
   private final Integer port;
   private final ChromeOptions options;
   private final Integer executorID;
   private URL url;
   private RemoteWebDriver wd;
   private boolean isStopped = false;
   private Queue<Pair<String, List<Pair<String, String>>>>
      actionsQueue;
   private String executorEmail = "";
   private String executorPassword = "";


   public RemoteExecutor(String ip_address, Integer port,
      ChromeOptions options, Integer executorID) throws
      InterruptedException {
      this.ip_address = ip_address;
      this.port = port;
      this.options = options;
      this.executorID = executorID;
```

```
        this.actionsQueue = new LinkedList<>();
    }

    @Override
    public void run() {
        this.connectToSeleniumInstance();
        try {
            this.registerNewAccount();
        } catch (InterruptedException e1) {
            System.out.println("Couldn't sign up " +
                this.executorID);
        }

        while (!this.isStopped) {
            try {
                if(!this.actionsQueue.isEmpty()) {
                    Pair<String, List<Pair<String, String>>>
                        currentAction =
                        this.actionsQueue.remove();
                    this.executeAction(currentAction);
                }

                sleep(SLEEP_TIME);
            } catch (InterruptedException e) {
            }
        }
    }
```

Azioni eseguite dai Remote Web Drivers

Per stabilire quali azioni dovrebbe compiere ogni Web Driver ho creato un'ulteriore classe, **RoutineActions**, all'interno della quale sono stati definiti i seguenti tipi di azione:

- **Search:** richiesta da parte del Web Driver di uno specifico URL.
- **Login:** accesso del Web Driver ad un account del sito.
- **Logout:** uscita da un account a cui si è acceduto tramite Login.
- **Cookie:** aggiunta o rimozione di un cookie del sito.

Una volta definiti questi tipi di azione ho suddiviso gli esecutori in due categorie: **standard** e **malevoli**; ognuno di essi a seconda della tipologia riceverà istruzioni diverse da far eseguire alla propria istanza di Web Driver.

Il metodo tramite il quale ho distribuito ad ogni esecutore le azioni della propria categoria sfrutta il raggruppamento di queste ultime in due diverse liste: *goodActionsList*, utilizzata dagli esecutori con *executor_id* pari e *badActionsList*, utilizzata da quelli con *executor_id* dispari.

L'idea è che gli esecutori standard effettueranno azioni innocue per la sicurezza dell'applicazione che non innalzeranno il threat score dell'utente; quelli malevoli invece cercando pagine riservate, utilizzando parametri proibiti, modificando i cookie ed accedendo ad account fittizi, metteranno in allerta il sistema che si comporterà, in caso di threat score eccessivamente alto, come definito nel paragrafo 3.3.5.

4.2 Esecuzione del Test

L'esecuzione del test implementato prevede come primo passo il dispiegamento di **100 Remote Web Drivers**: 50 standard e 50 malevoli che simuleranno lo scenario in cui 100 diversi utenti navigano contemporaneamente all'interno della Web Application effettuando diversi tipi di richieste.

In seguito al deployment delle istanze di Selenium è possibile avviare il test eseguendo il client Java che, come descritto nel paragrafo 4.1.2, inizializzerà dei Thread RemoteExecutor che si interfacceranno direttamente ai Remote Web Drivers.

4.3 Analisi dei risultati

Per analizzare i risultati ottenuti dal test verranno mostrate e discusse le Dashboard Kibana che permettono la visualizzazione dei dati attraverso grafici e tabelle. Dai grafici sarà particolarmente evidente la suddivisione degli utenti in tre diverse classi che rappresentano le seguenti categorie:

- **Utenti standard**: Sono distinguibili in relazione al loro indirizzo IP, il byte meno significativo sarà un numero dispari (es. 172.29.0.11, 172.29..0.13, ecc...)
- **Utenti malevoli con account in blacklist**: Rappresentano quegli utenti che stanno utilizzando per il login un ipotetico account fittizio (*honeyuser* 3.4.1) inserito in blacklist dagli amministratori del sistema. Tra i 50 esecutori malevoli 25 di questi utilizzeranno degli honeyuser presenti in una lista nera del Threat Evaluator, anch'essi sono distinguibili dal byte meno significativo del loro indirizzo IP che sarà pari.
- **Utenti malevoli con account non in blacklist**: Rappresentano degli utenti malevoli il cui account non è fittizio poiché non presente in lista nera. Similmente agli altri utenti malevoli anch'essi avranno il byte meno significativo dell'indirizzo pari, pertanto sfruttando solamente l'IP gli utenti malevoli non saranno distinguibili tra loro.

Queste prime due tabelle rappresentano rispettivamente i dati di utenti malevoli e standard in relazione al loro session ID; nella figura 4.1 è possibile notare che:

- Tutti gli indirizzi IP hanno come byte meno significativo un numero pari.
- Il loro punteggio di minaccia ha raggiunto il limite superiore stabilito nel paragrafo 3.3.5 di 2000 punti, quest'ultimo ha permesso al sistema di attuare il meccanismo di difesa che impedisce all'utente di continuare la navigazione per 24 ore.
- La loro *breach_flag* è impostata a true poiché tutti gli utenti in questa tabella hanno effettuato l'accesso con un account fittizio in lista nera.

Session Status by ID

Session ID	IP Address	↓ Threat Sc...	Threat Status	Breach Flag	Email	Session Dur...	User Agent
d7f7461178...	192.168.176.36	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
d593944a20...	192.168.176.56	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
d36d26a33f...	192.168.176.54	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
c3e7ea1069...	192.168.176.32	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
c1a93b49ad...	192.168.176.46	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
c06a025833...	192.168.176.8	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
bc5801c570...	192.168.176.64	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
b219214f32...	192.168.176.14	2,020	evil_user	true	executor_ba...	00:00:11	Mozilla/5.0 (...
9b0e8c62be...	192.168.176.30	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
980fafb36ce...	192.168.176.58	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
97fe1510b6f...	192.168.176.80	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
94dcaecbc2...	192.168.176.84	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
8e9198efad...	192.168.176.40	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
8d269b8044...	192.168.176.52	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
84d6079a6b...	192.168.176.92	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...
7bc6fd5077f...	192.168.176.20	2,020	evil_user	true	-	00:00:00	Mozilla/5.0 (...

Figura 4.1: Dati di sessione in relazione al session ID degli utenti malevoli

Nella figura 4.2 invece tutti gli utenti presentano un livello di minaccia nullo poiché facenti parte della categoria di quelli standard (byte meno significativo dell'indirizzo dispari) che non hanno effettuato richieste potenzialmente pericolose.

Session Status by ID

Session ID	IP Address	↓ Threat Sc...	Threat Status	Breach Flag	Email	Session Dur...	User Agent
ff0007ba961...	192.168.176.19	0	safe_user	false	executor_go...	00:00:12	Mozilla/5.0 (...
fc435632ed...	192.168.176.101	0	safe_user	false	executor_go...	00:00:11	Mozilla/5.0 (...
f656416691...	192.168.176.25	0	safe_user	false	executor_go...	00:00:15	Mozilla/5.0 (...
efc1b86e67...	192.168.176.63	0	safe_user	false	executor_go...	00:00:11	Mozilla/5.0 (...
ef1020f7097...	192.168.176.97	0	safe_user	false	executor_go...	00:00:15	Mozilla/5.0 (...
eed52f00f1e...	192.168.176.13	0	safe_user	false	executor_go...	00:00:12	Mozilla/5.0 (...
e7dafc3414...	192.168.176.77	0	safe_user	false	executor_go...	00:00:04	Mozilla/5.0 (...
e7c72ab146...	192.168.176.39	0	safe_user	false	executor_go...	00:00:08	Mozilla/5.0 (...
e5cb1ba94a...	192.168.176.29	0	safe_user	false	executor_go...	00:00:10	Mozilla/5.0 (...
d9c4d9ca44...	192.168.176.11	0	safe_user	false	executor_go...	00:00:10	Mozilla/5.0 (...
d878ac1977...	192.168.176.51	0	safe_user	false	executor_go...	00:00:08	Mozilla/5.0 (...
d3ece3d9b4...	192.168.176.93	0	safe_user	false	executor_go...	00:00:08	Mozilla/5.0 (...
ce39ae1585...	192.168.176.41	0	safe_user	false	executor_go...	00:00:05	Mozilla/5.0 (...

Figura 4.2: Dati di sessione in relazione al session ID degli utenti standard

Con i due seguenti grafici a barre verticali possiamo visualizzare graficamente un riassunto dei livelli di minaccia suddivisi per indirizzo IP. Nella figura 4.3, analogamente a quanto riportato nelle tabelle in figura 4.1 e 4.2, i threat score risultano nulli per gli utenti standard e di valore più elevato per quelli malevoli.

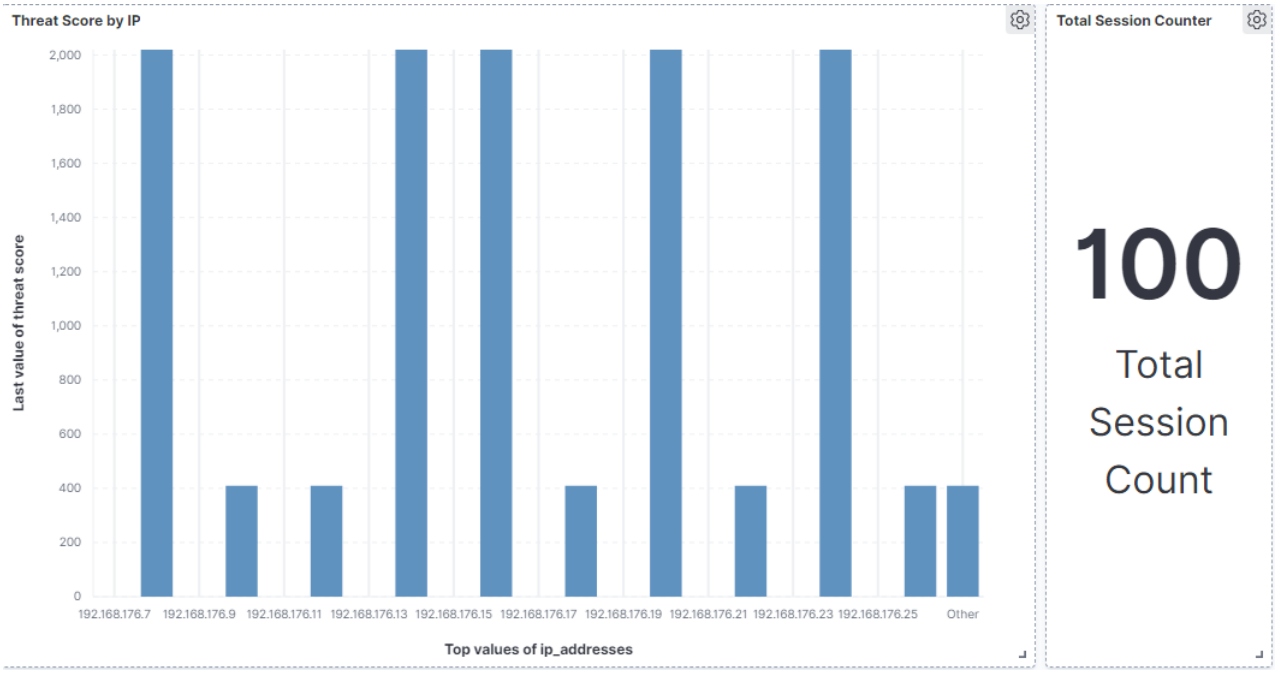

Figura 4.3: Grafico rappresentante il threat score degli utenti in relazione al loro indirizzo IP

Nella figura 4.4 è stato applicato un filtro ai dati per ottenere tutte le sessioni con livello di minaccia superiore a 2000 punti; come da aspettative il numero totale di sessioni che soddisfano questo criterio sono 25, corrispondenti a quegli utenti che stanno utilizzando un account in lista nera.

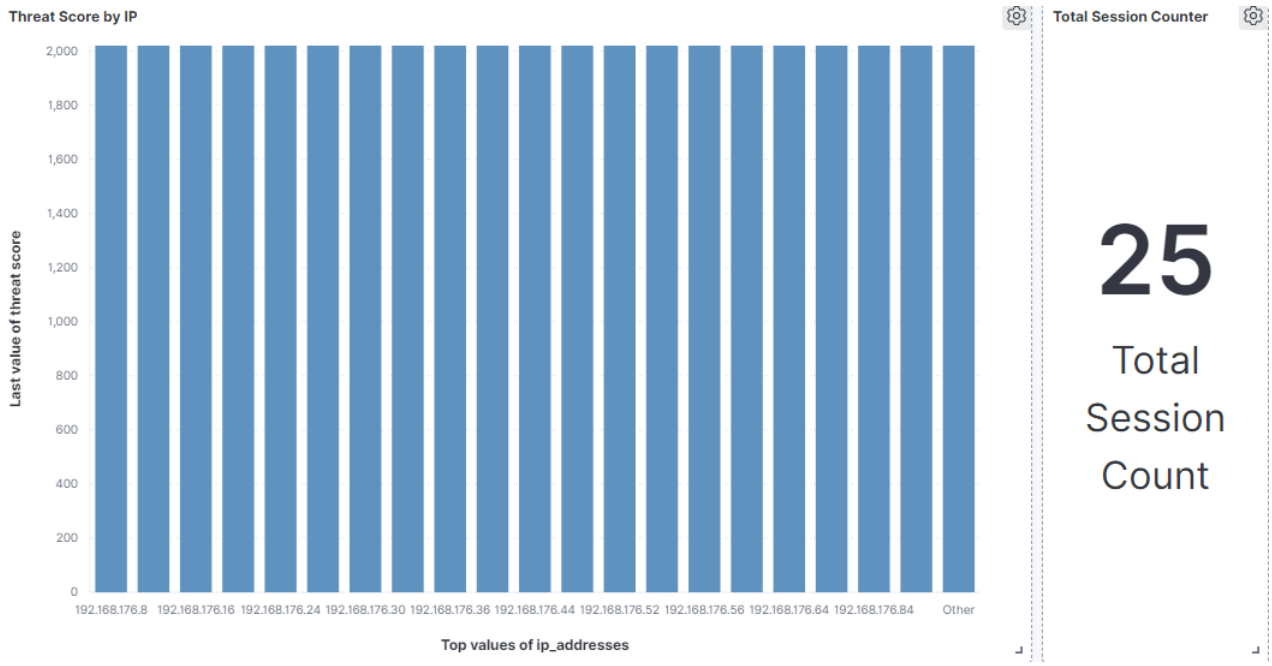

Figura 4.4: Grafico rappresentante il threat score degli utenti malevoli con account in lista nera

Gli ultimi grafici che prenderemo in considerazione sono quelli relativi alle richieste.

In questa prima figura (4.5) è possibile osservare l'impennata del numero di richieste effettuate al Webserver WordPress nel momento del lancio del test. Al termine dell'esecuzione delle azioni sui Remote Web Browsers, Elasticsearch ha registrato 1650 diverse richieste, numero che non tiene conto di tutte quelle scartate in seguito alla limitazione di utenti con threat score eccessivamente elevato.

Figura 4.5: Serie temporale delle richieste effettuate alla Web Application

Quest'ultimo grafico rappresentante il numero di richieste effettuate per indirizzo IP, evidenzia nuovamente la suddivisione nelle tre classi di utenti sopra discusse.

In particolare notiamo come gli utenti standard e quelli malevoli con account non in blacklist siano riusciti ad effettuare la totalità delle ricerche previste, rispettivamente 13 e 29; gli utenti malevoli con account in blacklist invece dopo solo 11 richieste sono stati limitati dal meccanismo proattivo di difesa che gli ha impedito di svolgere le 29 azioni previste.

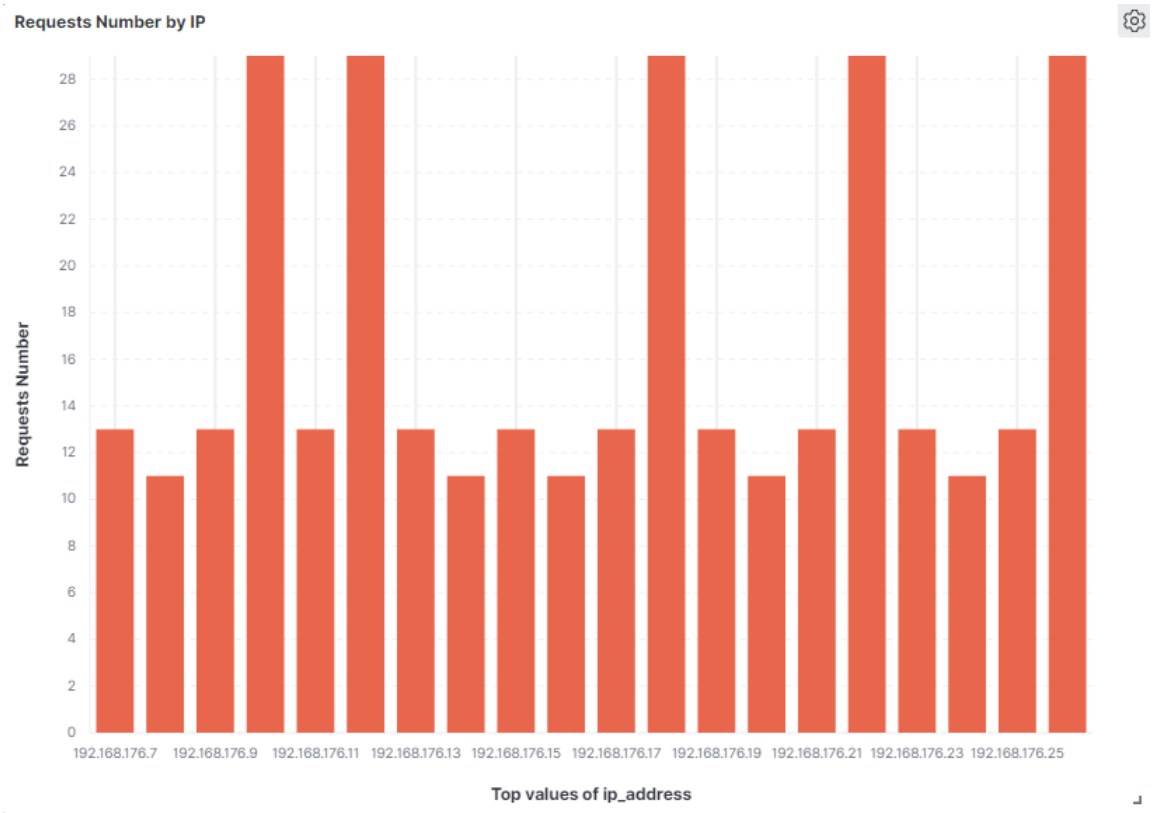

Figura 4.6: Numero di richieste effettuate dagli utenti in relazione all'indirizzo IP

4.4 Conclusioni del Test

Il test realizzato ha permesso di verificare l'efficacia del sistema di Session Tracking e Threat Evaluation a regime, mostrando in particolare come il tracking delle sessioni, la valutazione del livello di minaccia e l'attuazione di contromisure siano in grado di funzionare in maniera coerente alle specifiche di progetto.

In questo scenario è stato sufficiente dispiegare una singola istanza di ogni sottosistema; Ciò non esclude che, nel caso in cui il carico di richieste aumenti criticamente, sia possibile replicare le istanze dei Web Servers ed aumentare il numero di nodi del cluster Elasticsearch attraverso tecnologie come **Docker Swarm** o **Kubernetes**.

Scenari di utilizzo del sistema

Come abbiamo visto, il sistema di Session Tracking e Threat Evaluation è utilizzabile anche all'interno di siti non particolarmente complessi dal punto di vista delle funzionalità messe a disposizione, questo risulta fondamentale per dare la possibilità anche a privati o piccole aziende di integrare il plugin all'interno del proprio sito web.

Allo stato attuale per poter utilizzare l'intero sistema è necessario che ogni singolo utilizzatore dispieghi la propria infrastruttura in locale o in cloud attraverso il file docker-compose descritto nel paragrafo 3.1.

Per rendere l'intero sistema facilmente integrabile da un utente all'interno della propria infrastruttura, si potrebbe pensare di creare una versione più leggera che: aggrega tra loro il plugin WordPress ed il sistema di Threat Evaluation e sostituisce Elasticsearch e Kibana con delle dashboard consultabili dall'area amministrativa di WordPress; in questo modo non essendo più necessario dispiegare dei container per il Threat Evaluator, per Elasticsearch e per Kibana, un privato o un'azienda potrebbe iniziare ad utilizzare il sistema semplicemente installando il plugin dal marketplace di WordPress.

Conclusioni

Posso affermare con soddisfazione che il progetto è stato realizzato con successo e che soddisfa tutti i requisiti citati in una prima fase di analisi. Mi ha inoltre fatto molto piacere constatare come nella fase di test il sistema si sia comportato egregiamente, raggiungendo gli obbiettivi prefissati e confutando la correttezza del metodo di sviluppo progettuale impiegato.

Ringraziamenti

Vorrei iniziare ringraziando il Dott. Ciro Barbone, correlatore della tesi, che è sempre stato estremamente disponibile a chiarire qualsiasi mio dubbio offrendomi anche talvolta spunti interessantissimi che mi hanno permesso di accrescere le mie conoscenze nell'ambito della Cyber Security. Dei ringraziamenti speciali vanno anche al Prof. Vittorio Ghini, relatore della tesi, che mi ha permesso di affacciarmi al mondo dell'integrazione di sistemi attraverso l'omonimo corso tenuto dal Professore in maniera eccepibile sotto ogni punto di vista, ed alla azienda CyberLoop di Cesena presso la quale ho svolto le mie ore di tirocinio. Relativamente a quest'ultima vorrei citare in particolare il Dott. Eugenio Cavina che come mio tutor aziendale è sempre stato disponibile ad aiutarmi qualunque fosse il problema e ad offrirmi spunti che mi hanno guidato verso l'apprendimento di diverse nuove tecnologie di cui prima ignoravo l'esistenza. Sarebbe impossibile non citare nei ringraziamenti anche la mia famiglia, in particolare mia madre e mio padre che con i loro sacrifici mi hanno permesso di proseguire gli studi e di acquisire l'indipendenza necessaria per andare avanti nel mio percorso di crescita personale. Un enorme grazie va anche a tutti i miei amici che da diversi anni mi sono vicini supportandomi e sopportandomi, soprattutto durante i momenti più bui come i mesi di lockdown. Sebbene il periodo di pandemia mi abbia impedito di approfondire la conoscenza con i miei compagni di corso, ci tengo a ringraziare anche loro che nel momento del bisogno sono sempre stati disponibili per chiarire ogni mia perplessità. Tra loro vorrei ringraziare specialmente Andrea che è stato per me di riferimento sotto ogni punto di vista: dandomi la motivazione per proseguire questo percorso, credendo nelle mie capacità e spronandomi ogni giorno a dare il 110% per raggiungere gli obbiettivi a cui ambisco.

Bibliografia

[1] Gunes Acar, Christian Eubank, S. Englehardt, Marc Juárez, A. Narayanan, and Claudia Díaz. The web never forgets: Persistent tracking mechanisms in the wild. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[2] Tomasz Bujlow, Valentín Carela-Español, Josep Solé-Pareta, and Pere Barlet-Ros. A survey on web tracking: Mechanisms, implications, and defenses. *Proceedings of the IEEE*, 105(8):1476–1510, 2017.

[3] Tatiana Ermakova, Benjamin Fabian, Benedict Bender, and Kerstin Klimek. Web tracking - a literature review on the state of research. In *HICSS*, 2018.

[4] Jeremiah Grossman. Tracking users without cookies. https://blog.jeremiahgrossman.com/2007/04/tracking-users-without-cookies.html.

[5] Samy Kamkar. Webtracking with evercookies. https://samy.pl/evercookie/.

[6] G. Pugliese. Web tracking: Overview and applicability in digital investigations. *it - Information Technology*, 57:366 – 375, 2015.

[7] smashingmagazine.com. How to interact with the wordpress database. https://www.smashingmagazine.com/2011/09/interacting-with-the-wordpress-database/.

[8] WordPress.org. Wordpress php coding standards. https://make.wordpress.org/core/handbook/best-practices/coding-standards/php/.

[9] Wordpress.org. Wordpress plugin development handbook. https://developer.wordpress.org/plugins/.