Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Creazione di un sistema di Cyber Threat Hunting mediante algoritmi di Machine Learning e Deep Learning

Creation of a Cyber Threat Hunting system through Machine Learning and Deep Learning algorithms

Relatore

Prof. Domenico Ursino

Candidato

Paolo Grossetti

Correlatori

Ing. Paolo Russo

Ing. Rossella Oliva

Anno Accademico 2018-2019

Indice

Introduzione 1

1 Descrizione dello scenario di riferimento 5
1.1 Cyber Threat Hunting 5
1.2 Le fasi della Cyber Threat Hunting 7
1.2.1 "The Hunting Loop" di Sqrll 8
1.2.2 "A Practical Model for Conducting Cyber Threat Hunting" della SANS Institute 10
1.3 Le risorse della Cyber Threat Hunting 12
1.3.1 Hunters 12
1.3.2 Dati 13
1.3.3 Tools 13

2 Contesti scientifici di riferimento 15
2.1 Anomaly Detection 15
2.2 Log Analysis 17
2.2.1 Log di un Server Web 18
2.3 Machine Learning 20
2.3.1 Forme di Machine Learning 22
2.4 Reti Neurali e Deep Learning 24
2.5 OWASP Top 10 28

3 Strumenti utilizzati 33
3.1 Elastic Stack 33
3.1.1 Elasticsearch 34
3.1.2 Logstash 36
3.1.3 Kibana 38
3.1.4 Beats 43
3.2 Python e Librerie 43
3.2.1 Pandas 45
3.2.2 Scikit-Learn 45
3.2.3 Keras 48
3.3 MISP 50

4 Modello di Anomaly Detection in uno scenario controllato 53
4.1 Scenario 53
4.2 Dataset 54
4.3 Data Exploration 54
4.3.1 Pipeline Logstash 55
4.3.2 Kibana 58
4.4 Approccio Proposto 61
4.5 Algoritmi Utilizzati 62
4.5.1 SVM 62
4.5.2 Isolation Forest 65
4.5.3 One-Class SVM 67
4.6 Data Extraction 67
4.7 Session Extraction 69
4.8 Feature Extraction 69
4.8.1 IP Reputation e MISP 72
4.9 Training del Modello 75
4.10 Testing e Valutazione 77

5 Modello di Anomaly Detection in un caso reale 79
5.1 Scenario 79
5.2 Dataset 79
5.3 Data Exploration 80
5.4 Approccio Proposto 83
5.5 Algoritmi Utilizzati 83
5.5.1 Isolation Forest 84
5.5.2 One-Class SVM 84
5.5.3 Gaussian Mixture Models 84
5.5.4 Autoencoder 86
5.6 Data Extraction 88
5.7 Session Extraction e Data Filtering 90
5.7.1 Data Filtering 92
5.8 Feature Extraction 92
5.9 Training del Modello 93
5.10 Testing e Valutazione 96

6 Discussione in merito al lavoro svolto 99
6.1 SWOT Analysis 99
6.1.1 Strengths 99
6.1.2 Weaknesses 101
6.1.3 Opportunities 101
6.1.4 Threats 102
6.2 Lezioni apprese 103

7 Conclusioni e uno sguardo al futuro 105

Riferimenti bibliografici 107

Elenco delle figure

1.1 Threat Hunting: metodologia 7
1.2 Schematizzazione del "Threat Hunting Loop" proposto da Sqrrl 8
1.3 Il modello di Threat Hunting proposto dal SANS Institute 10

2.1 Esempio di *Anomaly Detection* 16
2.2 Esempio di log di accesso di un Server Web Nginx 20
2.3 Esempio di valutazione basata su un approccio tradizionale 21
2.4 Esempio di valutazione basata su un approccio *Machine Learning* . . . 22
2.5 Ambiti dell'Intelligenza Artificiale 24
2.6 Architettura di una *Artificial Neural Network* 25
2.7 Struttura di un neurone biologico 26
2.8 Struttura di un percettrone 26
2.9 Struttura di un neurone artificiale in una rete MLP 27
2.10 Differenze architetturali tra una *ANN* ed una *DANN* 28
2.11 *OWASP Top 10* del 2017 30

3.1 *Elastic Stack* 34
3.2 Processo di generazione di un *inverted index* 36
3.3 Flusso di una pipeline in *Logstash* 37
3.4 Home page di *Kibana* 7.5 39
3.5 Il pannello *Discover* di *Kibana* 7.5 40
3.6 Dashboard d'esempio in *Kibana* 7.5 41
3.7 La *Console* di *DevTools* in *Kibana* 7.5 42
3.8 Panoramica delle funzionalità offerte da *Scikit-Learn* 46
3.9 Architettura di Keras 49
3.10 Esempio di evento in MISP 51

4.1 Documenti associati all'indice `logstash-nginx` in *Discover* 58
4.2 Documenti associati all'indice `newlog` in *Discover* 59
4.3 Retta che separa gli elementi delle due classi in un problema bidimensionale 62
4.4 Iperpiano, margine e *support vector* in un problema bidimensionale . . 63
4.5 Separabilità in un problema bidimensionale 64

4.6 Esempio di *Soft-Margin* 64
4.7 Meccanismo di *Kernel Trick* 65
4.8 Meccanismo di isolamento dell'*Isolation Forest* 66
4.9 Parte dei *feed* disponibili in MISP 73

5.1 Documenti associati all'indice `production` in *Discover* 81
5.2 Esempio di *GMM* con 3 componenti 85
5.3 Esempio di architettura di un *Autoencoder* 87
5.4 Applicazione di un *Autoencoder* nell'ambito della *Computer Vision* . . 88
5.5 Porzione dei dati utilizzati nella fase di *training* 94

6.1 Matrice *SWOT* 100

Elenco delle tabelle

4.1 Feature di una sessione Web 71
4.2 Risultati ottenuti 77

5.1 Feature individuate per la presente sessione. In verde sono riportate le nuove feature individuate 93
5.2 Risultati ottenuti 97

Elenco dei listati

4.1 Pipeline *Logstash* utilizzata per caricare i dati grezzi in *Elasticsearch* . 55
4.2 Codice di una query di ricerca effettuata tramite la *Console* 59
4.3 Codice della funzione che estrae i dati da *Elasticsearch* 68
4.4 Codice della funzione che ricostruisce le sessioni degli utenti 69
4.5 Modulo MISP per l'IP Reputation 74
4.6 Codice del metodo che effettua una richiesta al modulo MISP per il calcolo della reputazione dell'IP 75
5.1 Pipeline *Logstash* utilizzata per caricare i dati grezzi in *Elasticsearch* . 80
5.2 Codice sorgente della query DSL utilizzata per individuare risorse statiche 82
5.3 Codice della funzione utilizzata per estrarre i dati da *Elasticsearch* . . 89
5.4 Codice della funzione utilizzata per filtrare le sessioni generate da noti Web crawler 91
5.5 Codice della funzione che ricostruisce le sessioni degli utenti 91
5.6 Codice della funzione che etichetta approssimativamente le sessioni di *testing* 96

Introduzione

La presente tesi si colloca nell'ambito della *Cybersecurity*, quel campo dell'informatica che si occupa di adottare tecniche preventive per proteggere computer, server, dispositivi mobili, sistemi elettronici, dati e reti da attacchi informatici [1].

Il tema della sicurezza informatica, attualmente, è di estrema importanza ed una delle principali ragioni sta nel fatto che la nostra società è diventata sempre più tecnologicamente dipendente, e questo trend non è destinato a rallentare. Secondo le ultime statistiche, al momento, nel mondo sono presenti più di 22 miliardi di dispositivi connessi ad Internet e si stima che si possa arrivare a 50 miliardi nel 2030; ciò significa che grandissime quantità di informazioni vengono scambiate nei modi più diversi, a partire dai tantissimi dispositivi *IoT*, fino ad arrivare alle recenti autovetture connesse [2].

La rapida evoluzione tecnologica ha stravolto il modo in cui svolgiamo le attività quotidiane, e questo ha fatto sì che nella rete viaggino dati davvero delicati: molto spesso referti sanitari vengono comunicati per e-mail, informazioni su carte di credito e conti correnti sono richieste per usufruire dei servizi di internet banking, senza contare la miriade di dati personali che noi stessi pubblichiamo nei social media.

La riservatezza di queste informazioni è messa a rischio da criminali informatici, detti *black hat hacker*, il cui scopo è di violare sistemi informatici per distruggere file, tenere computer in ostaggio o rubare password, numeri di carte di credito o altre informazioni personali.

La violazione dei sistemi informatici è un problema globale che riguarda tutti, sia singoli individui che piccole aziende o multinazionali: autovetture connesse, sistemi di sorveglianza smart, dispositivi medici, ad esempio pacemaker, applicazioni di mobile banking, sistemi aerei e infrastrutture critiche, ad esempio reti elettriche o dighe, sono esempi di sistemi tecnologici molto diversi fra loro, ma che hanno in comune la caratteristica di essere stati violati da pirati informatici [3].

L'impatto, in termini economici, dei crimini informatici per le imprese è significativo; tali crimini, infatti, causano perdite per miliardi di dollari ogni anno: secondo gli ultimi report, mediamente, l'impatto finanziario per un'azienda di livello enterprise che subisce un *data breach* è pari a 1.4 milioni di dollari; esso è, invece, pari a 162 mila dollari per una piccola-media impresa [4]. In Europa, con l'avvento del GDPR (Regolamento Europeo per Protezione dei Dati Personali), le aziende che trattano dati personali sono tenute a garantire determinati standard di sicurezza

per la conservazione e protezione; inoltre, le organizzazioni che subiscono un *data breach* sono tenute a denunciare la violazione, e le autorità potrebbero comminare sanzioni economiche qualora si dimostri che la fuga di dati sia dovuta alla mancata applicazione degli obblighi definiti nel regolamento.

I professionisti della sicurezza, quindi, hanno l'obiettivo di proteggere i sistemi informatici da accessi non autorizzati e di definire delle strategie difensive che consentano di individuare e bloccare sul nascere possibili attacchi, prima che causino danni maggiori. Una risorsa molto preziosa, il più delle volte sottovalutata, che consente di capire cosa sta succedendo su una macchina, sono i *log*; essi sono dei file di testo che hanno il compito di tenere nota di tutto ciò che viene fatto su un sistema. Una corretta ed adeguata analisi dei log consente di individuare anomalie, attività di ricognizione e possibili attacchi.

Il lavoro della presente tesi è stato svolto presso l'azienda Negg S.r.l. che si occupa, tra le diverse attività, di sviluppo Web. Per un'organizzazione, il proprio sito Web rappresenta uno strumento di comunicazione fondamentale per promuovere e far conoscere le proprie attività; tuttavia, un sito Web è accessibile a chiunque, e, quindi, può essere facilmente preso di mira da malintenzionati. Monitorare le attività degli utenti tramite l'analisi dei log consente di definire una strategia per il rilevamento e la prevenzione di attacchi sferrati contro le applicazioni Web.

L'obiettivo della presente tesi, quindi, è quello di sviluppare un modello che, tramite l'utilizzo di algoritmi di Intelligenza Artificiale, definisca il comportamento, standard e corretto, di un utente dell'applicazione Web, al fine di individuare attività anomale. Lo sviluppo del modello si basa sull'analisi dei log grezzi di un Server Web, al fine di definire un comportamento standard in modo tale che, quando le attività di un utente differiscono dalla norma, verrà lanciato un alert. Il comportamento di un utente in una applicazione Web è definito dall'insieme di attività che egli compie in un determinato intervallo di tempo; si è, quindi, deciso di definire un profilo basato sulle sessioni utente; una sessione consiste nel traffico generato da un specifico utente in un intervallo di tempo.

L'idea, pertanto, è di sviluppare un modello basato sulle sessioni degli utenti giudicate "normali", in modo tale da rilevare come anomalie quelle che si discostano maggiormente dal modello. In questo contesto, un'anomalia potrebbe indicare un tentativo d'intrusione, oppure un'insieme di attività che, seppur non malevole, meritano un approfondimento. Analizzare la sessione, quindi, permette di avere una visione completa delle attività svolte da un utente. In questo ambito, i log di un Server Web corrispondono ad un insieme di righe, e ciascuna di essa corrisponde ad una richiesta HTTP che contiene una serie di informazioni riguardo l'utente. I più comuni Server Web, quotidianamente, sono in grado di gestire centinaia di migliaia di richieste e ciò si traduce in file di log molto corposi; l'analisi dei log richiede, quindi, l'utilizzo di strumenti adeguati e, per questa ragione, nelle fasi iniziali, si è deciso di utilizzare gli strumenti offerti da *Elastic*. Quest'ultimo propone uno stack software, in cui, nello specifico, si è utilizzato *Elasticsearch*, un motore di ricerca e un DBMS NoSQL, e *Kibana*, una piattaforma di *Data Visualization*, che ha consentito di effettuare ricerche e visualizzazioni dei dati. Il modello proposto si basa sulle sessioni degli utenti, ma i dati a disposizione corrispondono alle singole richieste; pertanto, si è reso necessario definire una metodologia che permettesse, a partire dai dati grezzi, di ricostruire le sessioni; di quest'ultime, poi, si è individuato un

insieme di caratteristiche in grado di descriverle. Successivamente, gli algoritmi di Intelligenza Artificiale, implementati grazie a *Scikit-learn* e *Keras*, sono stati utilizzati per definire un modello che apprenda le caratteristiche delle sessioni valutate come "normali".

La presente tesi consiste di sette capitoli strutturati come di seguito specificato:

- Nel Capitolo 1 si descriverà l'attività di Cyber Threat Hunting, illustrandone le fasi di cui si compone e le risorse fondamentali.
- Nel Capitolo 2 si fornirà una panoramica dei principali argomenti trattati durante il lavoro svolto.
- Nel Capitolo 3 si illustreranno i principali strumenti utilizzati. Nello specifico, si parlerà dello stack software *Elastic*, delle librerie Python, utilizzate per la manipolazione dei dati e l'implementazione degli algoritmi di Intelligenza Artificiale, e di MISP, una piattaforma software di condivisione delle informazioni di sicurezza.
- Nel Capitolo 4 si descriverà la metodologia adottata per lo sviluppo di un modello di *Anomaly Detection* sfruttando dei dati sperimentali.
- Nel Capitolo 5 si descriverà la metodologia adottata per lo sviluppo di un modello di *Anomaly Detection* sfruttando dei dati provenienti da uno scenario reale.
- Nel Capitolo 6 verrà valutato il lavoro svolto e si illustreranno alcune considerazioni in merito alle le lezioni apprese.
- Nel Capitolo 7 verranno tratte le conclusioni in merito al lavoro realizzato e verranno esaminati alcuni possibili sviluppi futuri.

1

Descrizione dello scenario di riferimento

In questo primo capitolo sarà definito il concetto di Cyber Threat Hunting, illustrandone le finalità, i benefici e le più importanti caratteristiche. Si passerà, poi, ad un'analisi dettagliata delle diverse fasi di cui si compone il processo di Threat Hunting. Si parlerà infine dei requisiti essenziali che svolgono un ruolo fondamentale nella ricerca alle minacce.

1.1 Cyber Threat Hunting

La Cyber Threat Hunting è un'attività di *Cybersecurity* difensiva che riguarda la ricerca proattiva, iterativa e *human-driven*, attraverso reti, endpoint o dataset, al fine di rilevare attività dannose, sospette o rischiose, che hanno eluso il rilevamento da parte di strumenti automatici di sicurezza.

La ricerca di minacce informatiche non rappresenta una nuova attività nell'ambito della sicurezza informatica; tuttavia, solo di recente è diventata di estremo interesse per i più moderni Security Operation Center o, in breve, SOC. Un SOC è una struttura che ospita un team che si occupa del monitoraggio e dell'analisi della sicurezza di una organizzazione in maniera continuativa. L'obiettivo del team di un SOC, quindi, è quello di rilevare, analizzare e rispondere agli incidenti di sicurezza che occorrono in un'organizzazione utilizzando una combinazione di soluzioni tecnologiche ed una solida serie di best practice.

Il processo di Threat Hunting può rivoluzionare lo sforzo di rilevamento delle minacce in un'organizzazione; infatti, molte organizzazioni hanno già riconosciuto che tale attività deve svolgere un ruolo di primo piano nelle loro pratiche di rilevamento. Secondo un recente sondaggio condotto dall'istituto SANS, il 91% delle organizzazioni riferisce di miglioramenti riguardo velocità ed accuratezza nella fase di response dovuto all'attività di Threat Hunting.

Altri benefici menzionati nel sondaggio sono:

- tempo intercorso tra infezione e rilevamento ridotto;
- prevenzione della diffusione dell'infezione o del movimento laterale attraverso la rete;
- numero di violazioni effettive ridotto;

- esposizione a minacce esterne ridotta;
- tempo e denaro spesi per l'attività di risposta ad incidenti ridotti;
- numero e frequenza di infezioni dovuti a malware ridotti.

Un aspetto che vale la pena sottolineare è che la ricerca di minacce informatiche non è un'operazione completamente automatica, anzi, in realtà, è tradizionalmente manuale: uno dei motivi è che la fase di Threat Hunting non è un'attività reattiva. Per un analista di sicurezza, se il principale dato di input è il risultato di qualcosa che un tool ha automaticamente trovato, allora vuol dire che si sta adottando un comportamento reattivo e non proattivo. Un approccio alla sicurezza di tipo reattivo potrebbe risolvere un incidente identificato in maniera puntuale, che è una pratica di fondamentale importanza in un SOC, ma non si tratta di Threat Hunting, cioè, letteralmente, di "andare a caccia" di minacce. Le operazioni di Cyber Threat Hunting richiedono il contributo di un analista umano e riguardano ricerche proattive basate su ipotesi; l'importanza del fattore umano risulta, quindi, fondamentale.

Lo scopo dell'attività di Threat Hunting, in particolare, è quello di trovare ciò che è sfuggito dai sistemi di allarme automatici. Un allarme di uno strumento automatico può essere certamente un buon punto di partenza per definire un'ipotesi; tuttavia, un analista dovrebbe esplorare a fondo il contesto di ciò che è stato trovato per ottenere il vero valore della ricerca.

L'attività di Threat Hunting, come si può ben capire, si compone, in primo luogo, di capitale umano di qualità, poi di un adeguato quantitativo di dati ed informazioni da cui estrapolare ipotesi, ed, infine, di strumenti che siano di supporto all'attività di ricerca delle minacce.

Anton Chuvakin, un autorevole specialista di sicurezza, in una pubblicazione per conto di Gartner, ha definito quali debbano essere le caratteristiche chiave di una corretta attività di Threat Hunting. Chuvakin, nel suo documento, illustra 7 caratteristiche fondamentali [5]:

- *Proattiva*: è un requisito essenziale, già introdotto sopra. Gli analisti non devono attendere un allarme o un altro segnale da un tool automatico di sicurezza. Essere proattivi, in questo contesto, significa prendere l'iniziativa prima che scatti un allarme.
- *Basata su ipotesi ed indizi*: gli analisti devono seguire indizi ed intuizioni personali e non segnali ricevuti da strumenti o sistemi di rilevamento basati su regole. Tuttavia, i risultati di un'attività di Threat Hunting possono, poi, diventare regole.
- *Incentrata sugli analisti*: l'attività si basa fortemente sul fattore umano e non sugli strumenti. Gli analisti possono utilizzare diversi tools, ma svolgono un ruolo ausiliario e di supporto nell'individuazione di minacce nascoste.
- *Possibili violazioni*: la ricerca delle minacce si deve basare sul presupposto che le violazioni possono accadere e che gli attaccanti hanno lasciato delle tracce nell'ambiente IT dell'organizzazione.
- *Interattiva ed iterativa*: il processo di Threat Hunting richiede un'ipotesi iniziale da cui avviare la ricerca; tuttavia alcune scelte saranno ritrattate e quindi, molto probabilmente, saranno necessarie più iterazioni prima di arrivare a dei risultati corretti.

- *Metodologia ad-hoc e creativa*: secondo Chuvakin, la maggior parte degli esperti concorda sul fatto che l'attività di Threat Hunting non consiste nell'attenersi a regole, quanto, piuttosto, seguire un processo creativo ed una brillante metodologia che mira a individuare un'abile attaccante.
- *Basata su una profonda conoscenza delle minacce e dell'organizzazione*: la ricerca delle minacce si fonda su un'avanzata conoscenza degli attacchi informatici e dell'ambiente informatico dell'organizzazione. Gli analisti devono utilizzare le proprie conoscenze per aiutare l'organizzazione a conoscere meglio il suo dipartimento IT e capire in quale zona della rete gli attaccanti possono nascondersi;

Concludiamo questa trattazione con la Figura 1.1, che mostra uno schema che permette di comprendere la metodologia dell'attività di Threat Hunting.

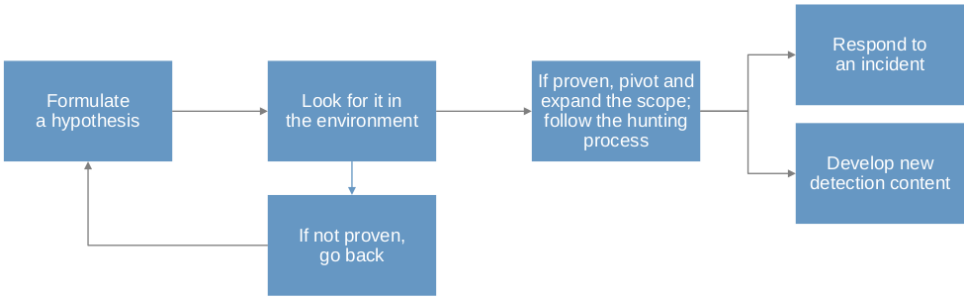


Figura 1.1: Threat Hunting: metodologia

1.2 Le fasi della Cyber Threat Hunting

Le odierne minacce informatiche sono diventate sempre più difficili da individuare a causa della loro continua mutevolezza, complessità ed organizzazione. Ci sono molti approcci per aumentare le difese di un'organizzazione contro cyberattacchi, e abbiamo già detto che l'attività di Threat Hunting, in questo senso, è una soluzione di fondamentale importanza.

Dopo aver fornito una contestualizzazione del Cyber Threat Hunting ed aver elencato i requisiti che questo tipo di attività deve possedere, rimane da definire un modello che stabilisce quali sono le fasi di cui si compone la Threat Hunting.

La Sqrrl e la SANS Institute, due autorevoli organizzazioni che si occupano di sicurezza, hanno tentato di fornire delle linee guida riguardo le fasi da seguire per implementare un modello formale di Cyber Hunting. La prima, nel suo "A Framework for Cyber Threat Hunting" [6], individua quattro fasi, mentre la seconda, con il suo "A Practical Model for Conducting Cyber Threat Hunting" [7], con un taglio più pratico, ne individua sei. È interessante analizzare entrambi gli approcci, il primo più teorico, il secondo più pratico, per avere un quadro completo della metodologia.

1.2.1 "The Hunting Loop" di Sqrll

Sqrrl è un'azienda americana, fondata nel 2012 e acquisita da Amazon nel Gennaio del 2018, che realizza software per attività di *Big Data Analytics* e *Cybersecurity*. Sqrrl, nel 2016, con il documento "A Framework for Cyber Threat Hunting", ha proposto il modello denominato "Threat Hunting Loop"; esso consiste in un processo iterativo composto da quattro fasi che definiscono un approccio efficace alla ricerca delle minacce.

In Figura 1.2 viene schematizzato il modello, completo di tutte le fasi, ed il relativo flusso. Secondo Sqrll, gli analisti di un team di Threat Hunting dovrebbero attraversare il ciclo nel modo più rapido ed efficace possibile.

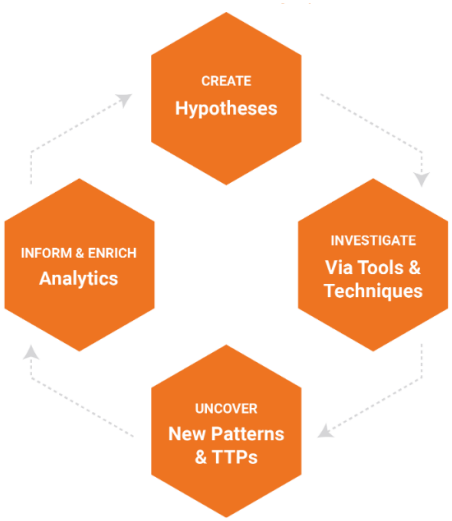


Figura 1.2: Schematizzazione del "Threat Hunting Loop" proposto da Sqrrl

Create: Hypothesis

L'attività di Threat Hunting inizia con la creazione di un'ipotesi riguardo qualche tipo di attività che potrebbe svolgersi nell'ambiente IT dell'organizzazione. Un'ipotesi, ad esempio, potrebbe essere che alcuni dipendenti dell'organizzazione hanno dovuto affrontare una trasferta di lavoro all'estero che li ha costretti a connettersi a reti di dubbia affidabilità; quindi, si potrebbe iniziare la ricerca pianificando di cercare tracce di compromissioni nei loro portatili, oppure supponendo che i loro account siano stati utilizzati in maniera impropria nella rete dell'organizzazione. Ciascuna di queste ipotesi dovrebbe essere testata individualmente. Gli analisti possono sviluppare manualmente ipotesi basate su tali attività di intelligence.

Nel framework proposto, Sqrrl, per quanto riguarda la generazione di ipotesi, fa riferimento a tre tipologie:

- *Analytics-Driven*: algoritmi di *Machine Learning* e *UEBA* (*User and Entity Behavior Analytics*) sono utilizzati per sviluppare score aggregati di rischio che possono essere utilizzati come ipotesi iniziali.
- *Situational-Awareness Driven*: rientrano in questa categoria ipotesi che si basano su analisi *Crown Jewels* oppure su valutazioni dei rischi aziendali.
- *Intelligence-Driven*: rientrano in questa categoria ipotesi che si basano su rapporti di intelligence sulle minacce, analisi di malware ed analisi delle vulnerabilità.

Investigate: Via Tools and Techniques

Nella seconda fase, le ipotesi iniziali vengono analizzate utilizzando diverse tecniche e strumenti, ad esempio tramite tool di *Data Visualization* o di *Data Linked Analysis*. In questa fase saranno i dati sono analizzati mediante analisi statistiche, analisi visive cioè visualizzando in maniera grafica le informazioni, ed, infine, tramite tecniche che sfruttano algoritmi di *Machine Learning*. Esistono molte altre tecniche complementari, tra cui il *clustering* dei dati o altre procedure che si basano sullo *stack counting*.

Gli analisti di sicurezza possono usare tutte le tecniche precedentemente citate per scoprire nuovi pattern sospetti nei dati e ricostruire le complesse attività di un'intrusione per rilevare tecniche, tattiche e procedure (TTP) di un attaccante.

Uncover: New Patterns and TTPs

I comportamenti sospetti ed i TTP degli attaccanti sono stati individuati utilizzando le tecniche di cui si è discusso nella sottosezione precedente. Questa fase è una parte critica del modello di Threat Hunting proposto. Un esempio di tale processo potrebbe essere il seguente: una precedente indagine ha rivelato che l'account di un utente si sta comportando in maniera anomala, in quanto che l'account sta generando una quantità insolita di traffico in uscita. Dopo degli accertamenti approfonditi si scopre che l'account dell'utente era stato inizialmente compromesso da un exploit, che ha violato un servizio di terze parti, esterno all'organizzazione. Questo TTP dovrebbe essere registrato, condiviso (sia internamente che esternamente) e tracciato nel contesto di una campagna di cyberattacchi più ampia. Tramite relazioni tra dati sarà possibile rivelare anche quali altri account sono stati associati al servizio di terze parti compromesso.

Automated Analytics

Nell'ultima fase, le attività di Threat Hunting che hanno avuto esito positivo, cioè sono riuscite ad identificare minacce concrete, costituiscono la base per arricchire gli strumenti automatici di analisi. Il team di Threat Hunting non deve perdere tempo sulla stessa minaccia più e più volte. Dopo aver trovato una tecnica che permette di individuare il problema in questione, si deve cercare di automatizzare il processo di rilevamento in modo tale che il team possa concertarsi su nuove minacce.

Ci sono molti modi in cui ciò può essere fatto, ad esempio sviluppando uno script che esegue, in maniera periodica, una ricerca salvata, sviluppando tool di analytics in Apache Spark, R, o Python, oppure fornendo un feedback ad un algoritmo di *Machine Learning* supervisionato, confermando che un determinato pattern è dannoso.

1.2.2 "A Practical Model for Conducting Cyber Threat Hunting" della SANS Institute

Il SANS Institute (SysAdmin, Audit, Networking, and Security) è un'organizzazione, fondata nel 1989, che si occupa di fornire educazione informatica e training in materia di sicurezza informatica. Nel Dicembre del 2018, il SANS Institute ha pubblicato l'articolo "A Practical Model for Conducting Cyber Threat Hunting", con l'obiettivo di fornire un modello formale su cui basare le operazioni di Threat Hunting e quantificare il successo di tali operazioni in maniera precisa e completa. Il documento fornisce un modello di Threat Hunting pratico e rigoroso utilizzando sei fasi: scopo, ambito, equipaggiamento, revisione del piano, esecuzione e feedback. Anche in questo caso, il modello proposto è di tipo iterativo.
In Figura 1.3 viene illustrato il modello, completo di tutte le fasi, ed il relativo flusso.

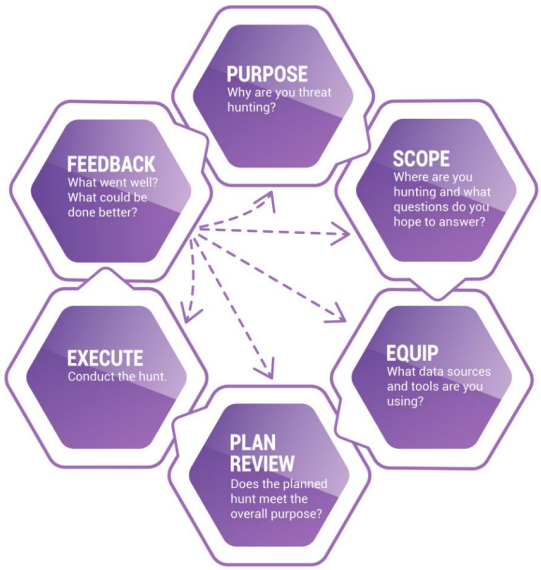


Figura 1.3: Il modello di Threat Hunting proposto dal SANS Institute

La bontà del modello è stato testato in campagne di Threat Hunting con dati reali. Secondo il SANS Institute, le attività di ricerca delle minacce condotte con e senza il modello hanno dimostrato l'efficacia e la praticità di tale metodologia.

Purpose

In questa fase ci si deve concentrare sugli obiettivi che l'organizzazione mira di raggiungere attraverso la campagna di Threat Hunting. Nello specifico, prima di avviare la campagna, è fondamentale definire in maniera chiara e rigorosa tre aspetti:

- *Scopo dell'attività di Threat Hunting:* lo scopo indica perchè si ritiene necessaria tale attività.

- *Dove avverrà la ricerca:* la definizione dello scopo richiede, anche, di determinare in quali ambienti avverrà la ricerca, identificando assunzioni e limiti della stessa.
- *Esito desiderato della ricerca:* il risultato desiderato dovrebbe essere in linea con gli obiettivi aziendali.

Scope

Il passo successivo consiste nella definizione dell'ambito, inclusa l'identificazione di sistemi e reti che devono essere analizzate. In questa fase, gli obiettivi principali sono l'individuazione dei sistemi da studiare e la generazione di ipotesi iniziali. Le due attività appena citate dovrebbero essere completate in maniera sequenziale, prima una e poi l'altra, per preservare l'integrità della ricerca.

Come già detto, in primo luogo, occorre individuare quali sono i sistemi sotto osservazione. Per fare ciò si potrebbe iniziare definendo quali sono le strutture coinvolte. Successivamente, si individua la sottorete, o le sottoreti, che devono essere studiate. Infine, al livello più basso di granularità, si selezionano i principali host d'interesse. Un errore molto comune che viene commesso è quello di rendere troppo ristretto l'ambiente di ricerca.

Dopo aver selezionato i sistemi da analizzare, gli analisi generano delle ipotesi. Quest'ultime servono per mantenere il focus sulla ricerca e per delineare la direzione che quest'ultima deve prendere. Così come i bersagli di un attacco possono essere scelti utilizzando degli algoritmi di targeting, allo stesso modo anche l'attività di Threat Hunting può sfruttare approcci di targeting ed algoritmi per la generazione di ipotesi iniziali. Il modello proposto dal SANS Institute è concorde con quello proposto dal Sqrrl riguardo le tipologie di ipotesi: esse, infatti, possono essere generate a partire da una conoscenza di dominio, da attività di intelligence o dalla consapevolezza della situazione.

Equip

La fase di "Equip" è incentrata sullo sviluppo di un accurato piano di raccolta ed analisi dei dati. In questa terza fase si richiede sia di selezionare le sorgenti dati sia le tecniche, le tattiche e le procedure che gli analisti impiegheranno per rispondere alle ipotesi definite nel passo precedente. È logico pensare, quindi, che, dopo aver sviluppato le ipotesi, identificare le fonti dei dati rappresenta lo step ulteriore per le validazione delle congetture iniziali. Le sorgenti dei dati, di conseguenza, saranno utilizzate per confermare o confutare le ipotesi iniziali.

Plan Review

La revisione del piano rappresenta un checkpoint per assicurarsi che la ricerca soddisfi gli obiettivi definiti nella prima fase. Fare una revisione del piano significa anche assegnare risorse aggiuntive che si rendono necessarie per eseguire l'attività di Threat Hunting in maniera ottimale. Tali risorse aggiuntive possono essere l'acquisizione di nuovi strumenti o l'assunzione di risorse esterne.

Execute

La fase di esecuzione avviene solo dopo che il piano della ricerca è stato approvato e consiste in iterazioni multiple di raccolta ed analisi dei dati. Gli analisti raccolgono le informazioni identificate nella fase di "Scope" ed eseguono delle analisi per verificare, o confutare, le ipotesi sviluppate. In questa fase, per migliorare la ricerca, si possono sfruttare altri dataset ed utilizzare tecniche di analisi aggiuntive. Al termine della fase di "Execute", ovvero quando tutte le analisi sono state completate, l'analista può iniziare a sviluppare un report. Quest'ultimo dovrebbe rispondere alle domande iniziali, illustrare i risultati ottenuti e segnalare tutti gli eventi degni di nota.

Feedback

La fase di "Feedback" fornisce un quadro completo di tutte le fasi precedenti e degli effetti che esse hanno avuto sui risultati ottenuti. Quest'ultima fase deve essere un momento costruttivo per l'organizzazione perchè consente di capire come gestire le future attività di Threat Hunting con maggior efficienza, sulla base dei punti di forza e di debolezza riscontrate nelle precedenti attività. Si devono quindi segnare carenze e fornire potenziali soluzioni in modo tale da evitare di commettere gli stessi errori in futuro.

1.3 Le risorse della Cyber Threat Hunting

La Cyber Threat Hunting è un'attività complessa che richiede significativi investimenti in personale, strumenti e tempo.
Prima che il processo di ricerca delle minacce abbia inizio, è bene definire in maniera chiara le risorse fondamentali richieste da un'attività di Threat Hunting; queste sono il capitale umano, i dati e gli strumenti.

1.3.1 Hunters

Gli analisti umani rappresentano un componente critico del processo. Abbiamo già sottolineato più volte l'importanza del "fattore umano" in questo tipo di attività ed è bene ribadirlo ulteriormente. Le attuali tecnologie di sicurezza sono sicuramente in grado di individuare un gran numero di minacce; tuttavia, il più efficace sistema di rilevamento rimane il cervello umano. La ricerca proattiva di attività sospette dipende dall'iterazione e dall'intervento umano; di conseguenza il successo di tale processo dipende da chi sta effettuando la ricerca. Tuttavia, ciò non è semplice: la Threat Hunting è una delle più difficili discipline di sicurezza da padroneggiare.

A coloro che desiderano entrare nel team di Threat Hunting non solo vengono richieste conoscenze tecniche avanzate in settori quali l'analisi della rete, gli IDS, la forensics e l'analisi di malware, ma anche skill non tecnici come, ad esempio, la comprensione del processo organizzativo di un'azienda.

1.3.2 Dati

Nessuna attività di Threat Hunting può essere compiuta senza dati a sufficienza. I dati svolgono un ruolo fondamentale per gli analisti: le informazioni, collegate tra loro e correlate con altri indicatori, possono rivelare minacce nascoste. La Threat Hunting richiede un ampio assortimento di strumenti e sensori per collezionare, aggregare e analizzare i dati per la ricerca di indicatori di compromissione. La raccolta dei dati dovrebbe essere ampia, in modo tale da poter includere tutte le fonti che si possono gestire, quali:

- server;
- dispositivi di rete (firewall, switch, router);
- database;
- endpoint;
- dispositivi basati sul cloud;
- altri dispositivi.

È chiaro, quindi, che il successo della ricerca delle minacce è proporzionale alla quantità di dati che gli analisti hanno a disposizione.

1.3.3 Tools

Ogni dispositivo monitorato genererà una notevole quantità di dati che, ovviamente, non potranno essere elaborati manualmente. In questo caso, soluzioni come la gestione degli incidenti di sicurezza e degli eventi (SIEM) saranno essenziali per automatizzare una parte del processo, compresa la raccolta, la correlazione e la normalizzazione dei dati dai dispositivi appena menzionati.

Un ulteriore strumento, che non dovrebbe mancare in un'attività di Threat Hunting è l'EDR (Endpoint Detection and Response). Secondo Chuvakin nel suo "How to Hunt For Security Threats", se si dovesse iniziare al più presto una nuova attività volta a ricercare delle minacce, il primo strumento su cui investire è proprio un EDR perché "gli attaccanti trovano si trovano negli endpoint". Un EDR è uno strumento che permette di monitorare gli eventi generati da dispositivi hardware, chiamati endpoint, per ricercare attività sospette e gli allarmi generati da un EDR sono di aiuto per gli analisti di sicurezza al fine identificare, investigare e risolvere i problemi. Nell'ambito della ricerca alle minacce, un EDR consente di esaminare le tracce inequivocabili di un aggressore: esecuzioni, azioni su file, download, azioni di sistema, etc.

Un altro strumento cruciale è la threat intelligence: poiché gli attaccanti aggiornano costantemente le loro tecniche, avere informazioni aggiornate sugli IOC (Indicatori di Compromissione) può essere di immenso valore. Per threat intelligence si intende la conoscenza necessaria per prevenire o mitigare un cyberattacco. Gli indicatori di compromissione sono degli artefatti osservati su una rete o in un sistema operativo che, con un'elevata probabilità, indicano un'intrusione. IOC comuni sono: signature di virus, hash di malware, URL o nomi di domini di server di comando e controllo.

2

Contesti scientifici di riferimento

Nel capitolo corrente si fornirà una panoramica dei principali argomenti che vengono trattati in questa tesi. In primo luogo, si illustrerà il problema dell'Anomaly Detection, ponendo l'attenzione sul concetto di anomalia e di Novelty Detection. Seguirà, poi, una sezione che tratta il concetto di log, fornendo un resoconto del processo di Log Analysis, per poi porre l'attenzione, nello specifico, sui log di un Server Web. Nella sezione successiva si fornisce una panoramica degli algoritmi di Intelligenza Artificiale, a partire dai principi su cui si basa il Machine Learning, fino ad arrivare alle più recenti reti neurali e Deep Learning. Nell'ultima sezione viene riportato l'OWASP Top 10, un documento che definisce quali sono i principali rischi che coinvolgono le applicazioni Web.

2.1 Anomaly Detection

Per *Anomaly Detection* si intende il problema di individuare pattern nei dati che differiscono dalla norma [8]. Gli elementi imprevisti, a seconda dei diversi campi d'applicazione, sono spesso indicati come anomalie, *outlier*, osservazioni discordanti, eccezioni, aberrazioni, sorprese, peculiarità o contaminazioni. L'*Anomaly Detection* trova ampio impiego in una grande varietà di applicazioni o domini come, ad esempio, il rilevamento di frodi nelle transazioni con carte di credito, in compagnie assicurative o di assistenza sanitaria, nel rilevamento di intrusioni per la sicurezza informatica e nel rilevamento di guasti in sistemi critici per la sicurezza.

L'importanza del rilevamento delle anomalie è dovuta al fatto che queste ultime si traducono in informazioni significative (e spesso critiche), utilizzabili in un'ampia varietà di settori applicativi.

La Figura 2.1 mostra le anomalie in un dataset bidimensionale.

Un approccio superficiale all'*Anomaly Detection* consiste nel definire una regione che rappresenta il comportamento normale e definire come anomalia qualsiasi osservazione che non appartiene a tale regione. Le cose, in realtà, non sono così semplici, e diverse ragioni rendono quest'attività una sfida davvero interessante:

- Definire una singola regione che comprenda ogni comportamento normale è molto difficile. Inoltre, il confine tra comportamento normale ed anomalo non è pre-

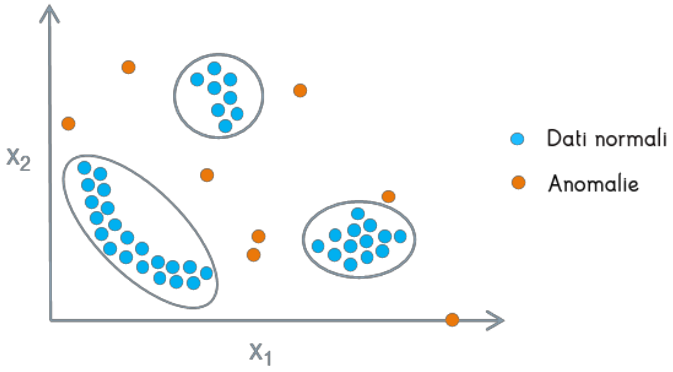


Figura 2.1: Esempio di *Anomaly Detection*

ciso; un *outlier* che si trova vicino al bordo decisionale può essere, in realtà, normale e viceversa.
- Quando le anomalie rappresentano comportamenti sospetti, i malintenzionati, spesso, adattano le loro azioni per far apparire le osservazioni anomale come normali, rendendo così più difficile definire il comportamento normale.
- In molti campi applicativi il concetto di normalità è in continua evoluzione e, quindi, l'attuale definizione potrebbe non esser sufficientemente rappresentativa in futuro.
- La mancata disponibilità di dati etichettati, utilizzati per le fasi di training e validation nella generazione dei modelli, è, di solito, un problema importante.
- Spesso i dati contengono rumore, simili alle anomalie reali, che, quindi, sono difficili da distinguere.

A causa delle motivazioni citate in precedenza (e di tante altre), come si può ben capire, il problema dell'*Anomaly Detection*, nella sua forma generale, non è di semplice risoluzione.

Un aspetto importante di una tecnica di rilevamento delle anomalie è la natura dell'anomalia. Le anomalie possono essere classificate nelle seguenti tre categorie:

- *Punto anomalo*: se un singolo dato può essere considerato anomalo rispetto al resto dei dati, allora il singolo elemento è definito come punto anomalo. Esempi di anomalie possono essere trovati nel rilevamento di frodi nelle carte di credito: una transazione per la quale l'importo speso è molto elevato rispetto al normale range di spesa per una certa persona sarà considerata come punto anomalo.
- *Anomalia contestuale*: se un singolo dato è anomalo in uno specifico contesto, ma non in altri, allora il dato è trattato come anomalia contestuale. La nozione di contesto è indotta dalla struttura dell'insieme di dati e deve essere specificata come parte della formulazione del problema. Facendo sempre riferimento al dominio delle frodi nelle transazioni di carte di credito, un esempio potrebbe essere il seguente: supponiamo che un individuo ha un tetto spesa settimanale di €100, ad eccezione della settimana di Natale, quando può arrivare a €1000 per spese extra e regali. Un nuovo acquisto di €1000 in una settimana di Febbraio sarà

considerata come anomalia contestuale, poichè la transazione non è conforme al normale comportamento dell'individuo in quel contesto di tempo.

- *Anomalia collettiva*: se un insieme di dati correlati è anomalo rispetto al resto del dataset, esso viene definita come anomalia collettiva. I singoli dati, presi singolarmente, di per sè, possono non essere anomali, ma, considerati nel loro complesso, possono definirsi *outlier*. Considerando la sequenza di azioni che vengono svolte in un sito web, un massiccio quantitativo di log che riporta il fallito tentativo di autenticazione di un utente corrisponde ad un tipico attacco che tenta di scoprire le credenziali tramite dei metodi *bruteforce*.

Un caso particolare di *Anomaly Detection*, molto utilizzata in certi campi applicativi, è la *Novelty Detection*.

La *Novelty Detection* si pone l'obiettivo di riconoscere i dati di test che differiscono da quelli utilizzati nella fase di training. Questo tipo di attività è, anche, comunemente riconosciuta come classificazione "one-class", dal momento che il modello è costruito per riconoscere i dati "normali" di allenamento.

La *Novelty Detection* è tipicamente utilizzata quando la quantità di dati anomali è insufficiente per costruire un modello che esplicitamente individui i dati "anormali". Tipici campi applicativi sono: rilevamento di masse in mammografie, sistemi di rilevamento guasti, intrusioni in sistemi elettronici di sicurezza, e molti altri.

Molto spesso i termini *outlier detection* e *Novelty Detection* sono utilizzati in maniera interscambiabile; tuttavia in questo contesto, è bene chiarirne la differenza: nella prima, il dataset contiene *outlier* che sono definiti come elementi che si discostano dagli altri; nella seconda, il dataset non contiene valori anomali e quindi l'obiettivo di questa tecnica di rilevamento è capire se una nuova osservazione è un'anomalia. In questo contesto, un punto anomalo è chiamato "novità".

2.2 Log Analysis

La *Log Analysis* è il termine utilizzato per indicare il processo di analisi di record, chiamati log, generati da sistemi elettronici.

Un log è un file che contiene, in ordine cronologico, le attività che sono state compiute su un sistema. I log sono generati da una grande maggioranza di sistemi tecnologici, inclusi dispositivi di rete, sistemi operativi, applicazioni e molti altri. L'analisi dei log, quindi, fornisce utili metriche che consentono di avere un quadro chiaro di tutto ciò che è accaduto in un'infrastruttura.

La maggior parte delle aziende, per essere conformi a regolamenti e norme, è tenuta ad archiviare ed analizzare i log. I record dei sistemi devono essere monitorati ed analizzati al fine di individuare errori, anomalie, attività sospette o non autorizzate: ciò permette di ricreare la catena di eventi che ha generato il problema e risolverlo in modo efficace.

Come già detto, quindi, dalla gestione dei problemi di sicurezza al rilevamento di anomalie o alla conformità delle normative, esistono molte situazioni in cui l'analisi dei log fornisce indicazioni preziose.

Un elenco, non esaustivo, dei campi applicativi in cui l'analisi dei log trova comunemente impiego è il seguente:

- *Risoluzione di problemi in sistemi elettronici migliorata*: uno dei casi d'uso più ovvi per l'analisi dei log è probabilmente nella risoluzione dei problemi dei server, delle reti o dei sistemi, dai crash delle applicazioni ai problemi di configurazione e ai guasti hardware. Una rapida risoluzione dei problemi aiuta ad evitare i tempi di inattività.
- *Response efficaci a Data Breach ed altri incidenti di sicurezza*: nell'ambito della sicurezza informatica, i log rappresentano una risorsa preziosissima in quanto forniscono numerose informazioni su possibili attaccanti come, ad esempio, indirizzi IP, richieste client/server, codici di stato HTTP, ed altro ancora. Molte aziende sottovalutano il valore dei log e non capiscono come utilizzarli per migliorare la sicurezza della propria organizzazione. I log agiscono come delle bandiere rosse e, con l'analisi dei log, è possibile rintracciare attività sospette ed impostare regole e soglie che consentono di proteggere il sistema da attacchi futuri. Riprendendo l'ambito di questa tesi, un'adeguata analisi di log consente di rilevare anomalie che, quindi, possono essere subito individuate ed eliminate. In tali casi, è possibile utilizzare algoritmi di Intelligenza Artificiale che rilevano comportamenti sospetti che, altrimenti, sarebbero passati inosservati.
- *Garantire la conformità con le politiche di sicurezza, i regolamenti e gli audit*: la maggior parte delle organizzazioni sono soggette a leggi governative e requisiti di settore che devono rispettare per garantire la sicurezza e l'integrità dei propri sistemi. In alcuni casi, le aziende sono tenute a registrare i dati e ad analizzarli quotidianamente. In questo modo, tale attività non solo aiuta a difendersi dalle minacce interne ed esterne, ma anche a dimostrare la volontà di rispettare l'ISO, il General Data Protection Regulation (GDPR), l'Health Insurance Portability and Accountability Act (HIPAA), e molti altri regolamenti. Inoltre, l'analisi può anche essere d'aiuto per garantire i requisiti di audit, le richieste di citazione in giudizio e le indagini forensi.
- *Comprendere il comportamento degli utenti online*: l'analisi dei log è uno dei migliori modi per capire il comportamento degli utenti su una piattaforma online. Non solo consente di capire il numero di visitatori sulla piattaforma, ma permette anche di ripercorrere il loro esatto percorso e di capire su quali pagine hanno trascorso più tempo, cosa stavano facendo, perché ci sono cambiamenti nel numero di visitatori ed altre informazioni utili; inoltre, l'analisi di log degli utenti permette di migliorare sensibilmente campagne di marketing. Infine, i log contengono informazioni riguardo il carico del traffico nonchè informazioni significative su come ottimizzare le performance della piattaforma.

Come detto nell'introduzione, la presente tesi tratta l'analisi dei log nell'ambito della sicurezza per il rilevamento di anomalie.

2.2.1 Log di un Server Web

Nella sezione precedente è stato accennato al fatto che quasi la totalità dei sistemi elettronici generi dei log e, chiaramente, il formato del log può variare a seconda del sistema preso in considerazione.

In questo lavoro di tesi sono stati analizzati i log di accesso di un Server Web Nginx e quindi è bene capire qual è il loro formato e quali informazioni contengono.

I log di accesso di un Server Web contengono tutte le richieste processate dal server. La raccolta dei dati relativi alle richieste degli utenti in un Server Web è essenziale: infatti, analizzando tali informazioni, diventa possibile non soltanto scoprire le prove di eventuali penetrazioni e determinare la portata delle azioni compiute dall'attaccante, ma, anche, di individuare tutte quelle operazioni di ricognizione attiva che, pur non rappresentando dei veri e propri attacchi, consentono di individuare eventuali debolezze e vulnerabilità del sistema. Come detto sopra, il Server Web preso in considerazione è Nginx; esso è un software open source che può essere utilizzato per Server Web, reverse proxy, caching, load balancer, media streaming, e molto altro. Nginx è nato come Server Web, progettato per le massime performance e stabilità; inoltre, può anche funzionare come server proxy per servizi di posta elettronica (IMAP, POP3, and SMTP) e come reverse proxy e load balancer per server HTTP, TCP e UDP [9].
In Nginx esistono due tipologie di log:

- *Error Log*: sono i log che contengono messaggi d'errore generali. Se si verifica un errore nella propria applicazione Web, gli *Error Log* rappresentano il punto di partenza per la risoluzione dei problemi in quanto forniscono informazioni aggiuntive sul perchè si è verificato l' errore;
- *Access Log*: ogni volta che viene elaborata una richiesta da parte del Server si genera un nuovo record negli *Access Log*. Ogni record contiene un timestamp ed include diverse informazioni riguardo il client e la risorsa richiesta. Gli *Access Log* possono mostrare l'indirizzo IP degli utenti, la pagina che hanno visitato, quanto tempo hanno passato su una determinata pagina e molte altre informazioni;

I record presenti negli *Error Log* sono, sicuramente, molto importanti perchè permettono di individuare in maniera semplice eventuali problemi nell'applicazione, tuttavia gli *Access Log* contengono tutte le richieste effettuate da utenti ed elaborate dal server, e, quindi, se analizzate in maniera corretta, rappresentano una vera e propria miniera d'oro per individuare attacchi, anomalie e attività sospette di malintenzionati. Il lavoro di tale tesi si colloca esattamente nel contesto appena citato.

Per default, nel file system del Server Web Nginx, i log si trovano in `logs/access.log` e le informazioni sono scritte secondo un formato predefinito, detto combinato. Il formato combinato definisce quali informazioni, e in che ordine, sono registrate in una richiesta HTTP. La Figura 2.2 mostra un esempio di log contenuto in un Access Log di un Server Web Nginx.

I diversi campi che costituiscono il log sono i seguenti:

- *indirizzo IP*: è l'indirizzo IP del client che ha inviato la richiesta al server;
- *identd*: indica l'identità del client; in Nginx, molto spesso, questo dato non è fornito e l'assenza di informazione è indicata con il simbolo "-";
- *userid*: indica lo userid dell'utente che ha richiesto il documento; in Nginx, quando questo dato è mancante, viene sostituito con il simbolo "-".
- *timestamp*: indica la data, l'ora ed il fuso orario in cui la richiesta è stata ricevuta dal server;
- *richiesta*: indica la richiesta effettuata dal client; tale campo è composto da:
  - *metodo*: indica il metodo HTTP utilizzato;

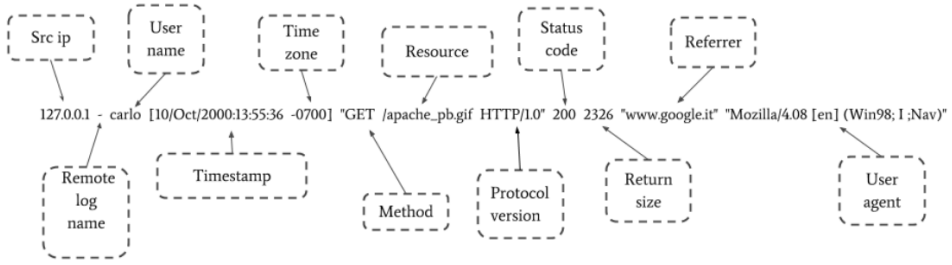


Figura 2.2: Esempio di log di accesso di un Server Web Nginx

  - *risorsa*: indica la risorsa richiesta dall'utente;
  - *protocollo*: indica il protocollo utilizzato con la relativa versione;
- *response code*: indica il codice di stato HTTP inviato dal server al client che fornisce informazioni riguardo l'esito della richiesta;
- *content length*: indica il peso dell'oggetto ritornato al client misurato in byte;
- *referer*: indica l'URL originale che porta al server corrente;
- *user agent*: fornisce informazioni sullo *user agent*, in particolare sul browser utilizzato dal client per il collegamento web, sul sistema operativo e sul modello del dispositivo utilizzato;

Come possiamo notare, i log di accesso di un Server Web Nginx contengono informazioni standard sulle richieste HTTP, che possiamo ritrovare in altri Server Web. Seppur con un ordine diverso, il formato dei log adottato da Nginx è molto simile al *Common Log Format*, conosciuto come *NCSA Common log format*, un formato testuale standard per la generazione di log da parte di Server Web. Essendo tale formato uno standard, esso è facilmente malleabile da parte di strumenti di analisi automatici; quindi, di conseguenza, questa importante proprietà è condivisa anche dai log generati da Nginx.

2.3 Machine Learning

Il *Machine Learning* è un campo dell'Intelligenza Artificiale che fornisce ai sistemi la capacità di apprendere autonomamente dai dati e migliorare dall'esperienza, senza che essi siano esplicitamente programmati per farlo. Per esempio, i filtri spam, oramai utilizzati dalla totalità dei client di posta elettronica, utilizzano dei programmi che si basano su algoritmi di Intelligenza Artificiale che sono in grado di imparare a segnalare spam con esempi di e-mail di spam (ad esempio, segnalate dagli utenti) ed esempi di e-mail regolari. L'insieme di esempi che il sistema utilizza per apprendere conoscenza è chiamato *training set*. Il *Machine Learning* si basa su un insieme di concetti matematici e statistici che, implementati su sistemi informatici, consente di estrarre informazioni, scoprire dei pattern e fare inferenze sui dati.

È importante comprendere quali sono i concetti essenziali dell'apprendimento automatico; tuttavia, allo stesso modo, è importante capire quali attività non rientrano in tale disciplina: ad esempio, scrivere regole statiche non rientra tra i metodi

di *Machine Learning*. Per chiarire questa differenza, riprendendo l'esempio dei servizi di posta elettronica, possiamo dire che esistono due modi per realizzare un applicativo che implementi un filtro per le e-mail di spam [10]:

- *Tramite un approccio tradizionale*: analizzando le e-mail di spam si potrebbe notare che, nell'oggetto della e-mail, molto spesso compaiono dei termini specifici, come ad esempio, "4U", "gratis", "fantastico", "carta di credito". Altri termini ricorrenti possono trovarsi nel nome del mittente e nel corpo della e-mail. Con tale approccio si dovrebbe scrivere un programma che rilevi ogni pattern che è stato individuato e generi un allarme qualora ci sia un matching con una delle regole definite. Dato che il problema non è banale, il programma diventerà, probabilmente una lunga lista di regole complesse, difficili da mantenere. La Figura 2.3 mostra uno schema riguardo l'approccio appena descritto.

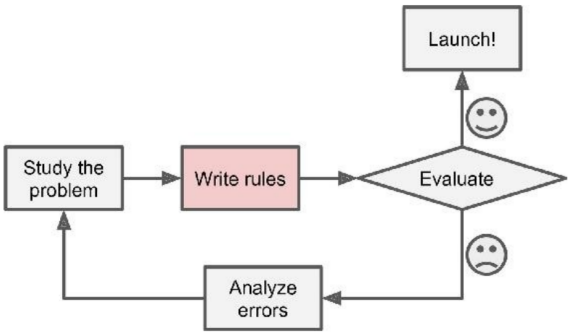


Figura 2.3: Esempio di valutazione basata su un approccio tradizionale

- *Tramite un approccio di tipo Machine Learning*: un filtro antispam costruito utilizzando tecniche di *Machine Learning* apprende automaticamente quali parole e frasi sono buoni predittori di spam, confrontando email di spam con quelle regolari e rilevando modelli di parole insolitamente frequenti. Il programma sarà più facile da mantenere e probabilmente più preciso. La Figura 2.4 illustra la metodologia descritta.

Confrontando, dal punto di vista pratico, le due soluzioni si può dire che, se gli *spammer* si accorgono che le e-mail contenenti una determinata parola sono bloccate, potrebbero iniziare a scrivere delle parole diverse al fine di eludere i controlli. Un filtro antispam che utilizza la prima soluzione dovrebbe essere, quindi, continuamente aggiornato. Ciò significa che, se gli *spammer* continuano a trovare ogni volta nuove parole, allora si dovrà continuare a scrivere nuove regole per sempre. Al contrario, un filtro antispam basato su tecniche di *Machine Learning* individua automaticamente che delle nuove parole stanno diventando insolitamente frequenti nelle e-mail di spam segnalate dagli utenti, quindi apprenderà dall'esperienza ed inizierà a segnalare anche i nuovi casi. Per riassumere, quindi, il *Machine Learning* è ottimo per:

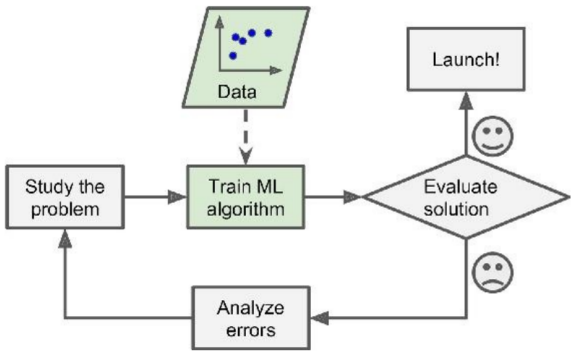


Figura 2.4: Esempio di valutazione basata su un approccio *Machine Learning*

- Problemi per i quali le soluzioni esistenti richiedono molte regolazioni manuali o lunghe liste di regole: un algoritmo di apprendimento automatico può semplificare il codice e funzionare meglio.
- Problemi complessi per i quali non esiste una buona soluzione utilizzando un approccio tradizionale: utilizzando adeguate tecniche di *Machine Learning* si può trovare una soluzione.
- Ambienti variabili: un sistema di *Machine Learning* può adattarsi ai nuovi dati.
- Penetrazione di problemi complessi e grandi quantità di dati.

2.3.1 Forme di Machine Learning

I sistemi che utilizzato algoritmi di Intelligenza Artificiale possono essere classificati in base alla quantità ed al tipo di supervisione che ricevono durante la fase di *training*. Una seconda categorizzazione possibile si basa, invece, sul tipo di output che ci aspettiamo di ricevere dal sistema.

Categorizzazione rispetto alla supervisione

Rispetto al grado di supervisione, possiamo individuare quattro grandi famiglie: apprendimento supervisionato, apprendimento non supervisionato, apprendimento semi-supervisionato ed apprendimento con rinforzo. Vediamo, ora, in dettaglio:

- *Apprendimento supervisionato*: con questo tipo di approccio l'algoritmo apprende dal passato. I dati di *training* contengono già la conoscenza, e gli algoritmi devono imparare a riconoscerli, al fine di predire correttamente gli eventi futuri. In questo caso si parla di dati etichettati; ciò vuol dire che i dati vengono forniti con una spiegazione. Ad esempio, se si sta progettando un filtro antispam, i dati utilizzati nella fase di addestramento del modello sono già etichettati, per esempio, come "e-mail spam" e "e-mail normale".
- *Apprendimento non supervisionato*: in questo caso, i dati non contengono informazioni aggiuntive riguardo il loro significato e, quindi, l'obiettivo è di estrarre dei pattern dai dati non etichettati. Con questo approccio, il sistema tenta di apprendere un modello senza nessun aiuto esterno. Riprendendo l'esempio del filtro

antispam, i dati non conterranno informazioni aggiuntive e, quindi, spetta all'algoritmo individuare pattern e schemi ricorrenti che consentiranno di distinguere lo spam dalle e-mail regolari.

- *Apprendimento semi-supervisionato*: questo tipo di approccio si colloca a metà strada tra le precedenti. In questo caso i dati sono parzialmente etichettati, ovvero alcuni di essi sono etichettati, mentre altri no. In genere, si usa una piccola quantità di dati etichettati per migliorare l'accuratezza del modello. Questo tipo di soluzione è adottata quando acquisire dati etichettati non è un'operazione agevole.
- *Apprendimento con rinforzo*: questo approccio è molto diverso dagli altri. Il sistema di apprendimento, chiamato agente in questo contesto, può osservare l'ambiente, selezionare ed eseguire azioni ed in cambio ottiene premi o penalità, a seconda dell'esito dell'azione compiuta. Questo tipo di approccio è utilizzato, ad esempio, dai robot che devono capire come devono muoversi in un ambiente sconosciuto.

Categorizzazione rispetto al risultato atteso

Un'altra categorizzazione dei compiti dell'apprendimento automatico si rileva quando si considera l'output desiderato. In particolare, possiamo individuare tre tipologie di task:

- *Classificazione*: la classificazione è una tecnica di apprendimento supervisionato tramite la quale i dati vengono classificati in categorie rilevanti, già conosciute a priori. Come per altre tecniche di apprendimento supervisionato, la classificazione è un processo a due passi:
  - Nella fase di *training*, al sistema vengono forniti dei dati già etichettati, in modo tale che esso possa sviluppare una comprensione delle diverse categorie.
  - Durante la fase di *testing* del modello, al sistema vengono forniti dati sconosciuti, ma simili. Il sistema, basandosi sulla comprensione che esso ha precedentemente sviluppato a partire dai dati di *training*, classificherà i nuovi dati.

  È bene sottolineare che la classificazione può essere binaria o multicategoria. Un'applicazione comune di questa tecnica riguarda il filtraggio dello spam delle e-mail, esempio già citato precedentemente.
- *Regressione*: la regressione è, anch'essa, una tecnica di apprendimento supervisionato, in cui il valore da predire è continuo. La regressione modella la relazione tra una o più variabili indipendenti e una variabile dipendente, dove i valori delle variabili indipendenti sono noti. Molti problemi pratici possono essere risolti mediante la regressione lineare oppure convertendo un problema non lineare in uno lineare. Nella regressione, i dati sono modellati a seconda del tipo di regressione: essa può essere lineare (i dati sono modellati con una linea retta) o polinomiale (i dati sono modellati con una funzione di ordine polinomiale). La regressione e la classificazione hanno delle proprietà in comune; tuttavia, la differenza tra i due task è che, nel primo caso, l'obiettivo è predire un valore numerico, mentre, nel secondo, un valore categorico. Un tipico esempio di regressione può essere la predizione del costo di una casa.

- *Clustering*: il *clustering* è una tecnica di apprendimento non supervisionato in cui gli oggetti devono essere raggruppati in insiemi, di cui non si conosce a priori la classe. Un *cluster* è un insieme di oggetti che sono simili l'un l'altro e sono dissimili dagli oggetti di altri cluster. Analogamente alla classificazione, il *clustering* non è uno specifico algoritmo, ma un task che può essere realizzato tramite tecniche differenti che dipendono dal modello scelto. I principali metodi di clustering possono essere suddivisi in:
  - *Metodi di partizionamento*: un algoritmo di partizionamento organizza gli N oggetti in K partizioni, al fine di massimizzare la distanza tra cluster differenti e minimizzare la distanza tra elementi che appartengono allo stesso *cluster*. Un algoritmo di partizionamento molto famoso è *K-means*.
  - *metodi gerarchici*: un metodo di *clustering* gerarchico lavora raggruppando gli oggetti in alberi di *cluster*, in cui il numero di quest'ultimi è scelto dall'utente.
  - *Metodi basati sulla densità*: i metodi di *cluster* basati sulla densità consentono di individuare *cluster* di forma arbitraria. Si considerano i *cluster* come regioni dense di oggetti nello spazio dei dati separate da regioni a bassa densità. L'algoritmo di *cluster* basato sulla densità più famoso è *DBSCAN*.
  - *Metodi basati sulla griglia*: i metodi di *cluster* basati sul griglia partono dal presupposto che i dati siano generati da una composizione di distribuzioni di probabilità.
  - *Metodi basati su vincoli*: il *clustering* basato sui vincoli ha lo scopo di costruire *cluster* che soddisfano preferenze o vincoli specificati dall'utente.

2.4 Reti Neurali e Deep Learning

Le reti neurali ed il *Deep Learning* sono ambiti specifici dell'Intelligenza Artificiale i cui modelli computazioni si ispirano alla struttura del cervello umano. La Figura 2.5 aiuta a comprendere la gerarchia degli algoritmi di Intelligenza Artificiale: il *Deep Learning* è una specializzazione delle *Artificial Neural Networks*, ed entrambe sono un caso particolare del *Machine Learning*.

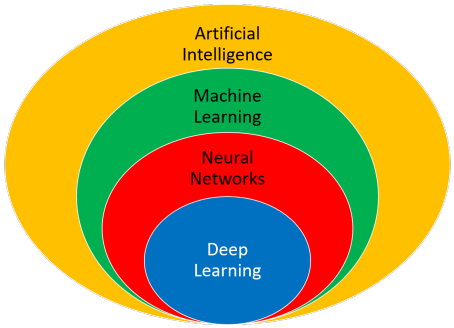


Figura 2.5: Ambiti dell'Intelligenza Artificiale

Artificial Neural Networks

Le *Artificial Neural Networks* rappresentano una classe di modelli di apprendimento automatico, ispirati da studi sul sistema nervoso centrale dei mammiferi. Ogni rete è composta da diversi neuroni interconnessi, organizzati in strati, che si attivano quando si verificano determinate condizioni. Il comportamento di una rete neurale è strettamente legato al tipo di architettura. Quest'ultima dipende da tre fattori:

- numero di neuroni.
- numero di strati.
- tipo di connessione tra gli strati.

La Figura 2.6 mostra l'architettura di una semplice *Artificial Neural Network*.

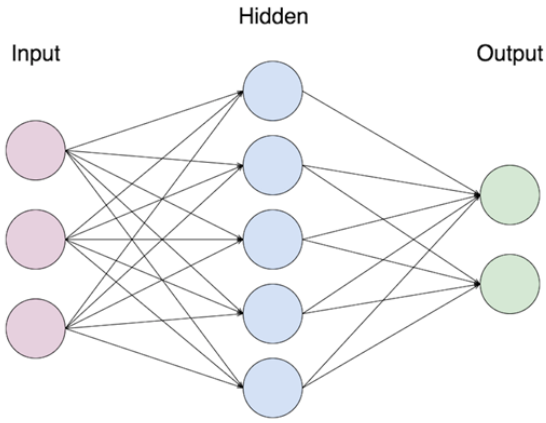


Figura 2.6: Architettura di una *Artificial Neural Network*

Il concetto di neurone di una rete neurale è, come già detto, ispirato al neurone biologico. In ambito biologico, un neurone è una cellula nervosa che fornisce l'unità funzionale fondamentale per i sistemi nervosi di tutti gli animali. I neuroni comunicano tra loro e trasmettono impulsi elettrochimici attraverso le sinapsi, da una cellula all'altra, purché l'impulso sia abbastanza forte da attivare il rilascio di sostanze chimiche attraverso la sinaptica. La forza dell'impulso deve superare una soglia minima o i prodotti chimici non verranno rilasciati. I principali componenti di cui è costituita una cellula nervosa sono i seguenti:

- *soma*: rappresenta il corpo della cellula;
- *dentriti*: consentono alla cellula di ricevere segnali dai neuroni vicini collegati;
- *sinapsi*: fungono da collegamento tra dentriti ed assone;
- *assone*: sono singole fibre che si estendono dal soma;

La Figura 2.7 mostra la struttura, appena descritta, di un neurone biologico.

Nelle *Artificial Neural Network*, un neurone è modellato con un percettrone. Percettrone è, anche, il nome di un modello lineare utilizzato per la classificazione binaria. Il percettrone è un classificatore binario a modello lineare con una semplice

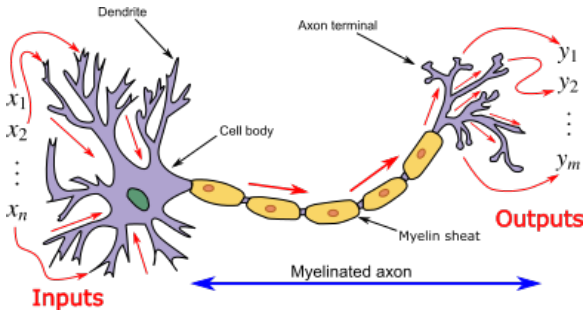


Figura 2.7: Struttura di un neurone biologico

relazione input-output. Come si può notare dalla Figura 2.8, N valori di input, moltiplicati per i relativi pesi sinaptici, sono sommati, ed il risultato viene inviato ad una funzione a gradino con un determinato *threshold*. In genere, con i percettroni, la funzione a gradino è la funzione di *Heaviside*, in cui il valore del *threshold* è pari a 0.5. Tale funzione genererà un valore binario (0 o 1), a seconda dell'ingresso.

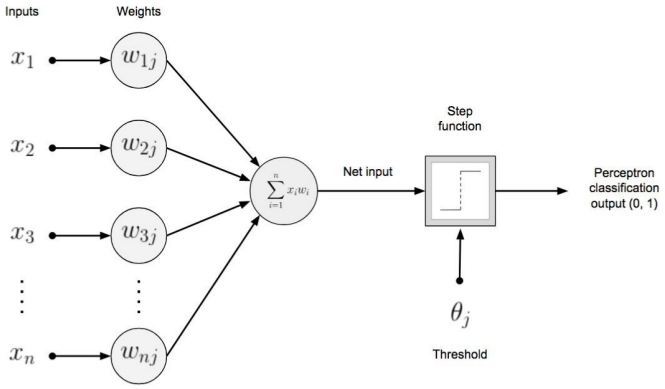


Figura 2.8: Struttura di un percettrone

La funzione di *Heaviside*, in questo caso, viene anche detta, nell'ambito delle *ANN*, funzione di attivazione e riceve, come detto in precedenza, il prodotto scalare degli input con i relativi pesi. L'output della funzione di attivazione è l'output del percettrone e ci fornisce una classificazione dei valori di input.

L'algoritmo di apprendimento modificherà i pesi del modello fino a quando tutti i record di input vengono correttamente classificati. I pesi tra le unità sono i principali mezzi di memorizzazione delle informazioni nelle reti neurali, e l'aggiornamento dei pesi è il modo principale in cui la rete neurale apprende nuove informazioni.

Il modello di percettrone, come già detto, si rifà al funzionamento del neurone biolologico, ed è possibile, quindi, trovare delle analogie tra i due sistemi: i dentriti di una cellula nervosa equivalgono ai nodi di input del percettrone, le sinapsi ai pesi

del modello, il soma alla funzione di attivazione e l'assone al valore di output del percettrone.

Storicamente, percettrone era anche il nome dato ad un modello composto da un singolo strato; di conseguenza, se la rete ha più di uno strato, viene chiamata rete MLP, o *Multi Layer Perceptron*. Una rete MLP è composta da uno strato di input, uno o più strati nascosti ed uno strato di output. Ogni strato ha uno o più neuroni artificiali. In questo tipo di rete, i neuroni artificiali sono simili ai percettroni citati precedentemente, con la differenza che, nelle reti MLP, si possono avere diverse funzioni di attivazione, a secondo dello specifico scopo di uno strato della rete. Il neurone di una rete MLP, quindi, è simile al suo predecessore, il percettrone, ma aggiunge flessibilità riguardo alla funzione di attivazione che possiamo usare. La Figura 2.9 mostra la struttura di un neurone in una rete MLP.

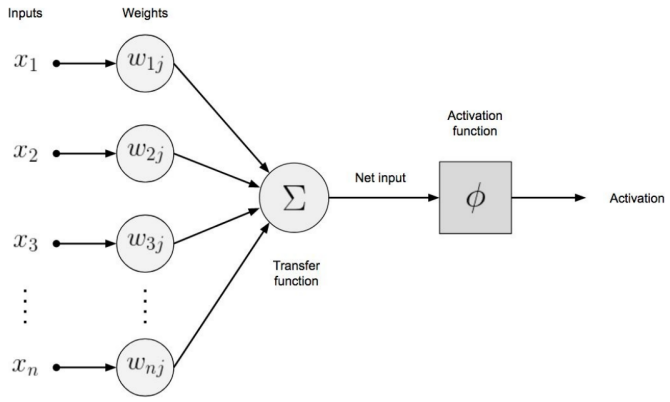


Figura 2.9: Struttura di un neurone artificiale in una rete MLP

Deep Learning

Il *Deep Learning* è una branca dell'Intelligenza Artificiale ed è una specializzazione delle reti neurali artificiali. In particolare, una rete neurale artificiale viene detta *deep*, quando ha due o più strati nascosti.

In linea generale, possiamo dire che le sfaccettature per cui una *Deep Neural Network* differisce da una rete MLP di tipo *Artificial Neural Network* sono le seguenti:

- numero di neuroni maggiore;
- complessità dei modi per collegare gli strati maggiore;
- complessità della fase di training maggiore;
- possibilità di estrarre automaticamente le feature;

Nel corso degli anni, la complessità delle reti neurali è aumentata in maniera esponenziale. L'aumento del numero di neuroni fa sì che sia possibile esprimere modelli sempre più complessi. Anche gli strati hanno avuto un'evoluzione: si è passati

da un'architettura multistrato di tipo *fully-connected* (strato in cui ogni neurone è collegato a tutti i neuroni di un altro strato) a porzioni di neuroni localmente connessi, nelle *Convolutional Neural Networks*, a connessioni ricorrenti tra gli stessi neuroni, nelle *Recurrent Neural Networks*. Tuttavia, più connessioni e di maggiore complessità significa, anche, che la rete ha più parametri da ottimizzare e questo richiede una grande quantità di potenza di calcolo. Tutto ciò ha permesso alle reti neurali profonde di modellare problemi di elevata complessità, come, ad esempio, nel campo del riconoscimento delle immagini.

Attualmente, il *Deep Learning* trova comunemente impiego nelle seguenti aree:

- classificazione di immagini;
- riconoscimento vocale;
- trascrizioni testuali;
- attività di traduzioni linguistiche;
- conversione *text-to-speech*;
- assistenti intelligenti, come Google Assistant o Alexa;
- guida autonoma;
- targeting pubblicitario;
- capacità di rispondere a domande in linguaggio naturale;
- teoria dei giochi;

Concludiamo la sezione con la Figura 2.10 che mostra la differenza architetturale tra una *ANN* ed una *Deep Artificial Neural Network*.

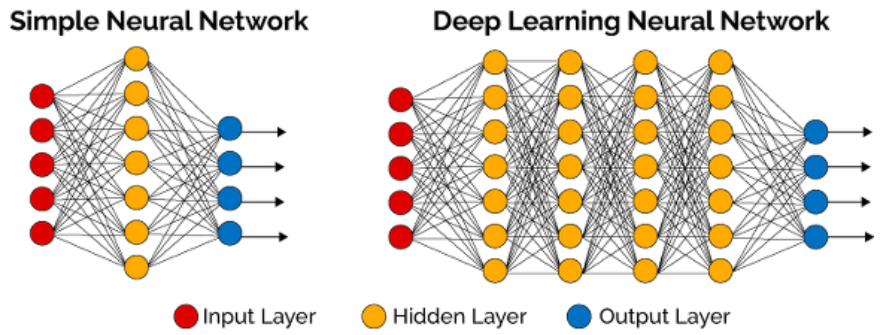


Figura 2.10: Differenze architetturali tra una *ANN* ed una *DANN*

2.5 OWASP Top 10

L'obiettivo di questo lavoro di tesi, come già accennato nell'introduzione, è di rilevare anomalie in un Server Web tramite l'analisi di log. Fatta questa breve premessa, quindi, è importante capire quali sono le anomalie che possono capitare in questo tipo di sistemi; allo stesso modo, è importante comprendere quali sono le minacce ed i possibili attacchi che possono prendere di mira un'applicazione Web.

OWASP Top 10 è un report regolarmente aggiornato dalla *Open Web Application Security Project* (*OWASP*) che si focalizza sui dieci rischi maggiormente critici per la sicurezza delle applicazioni web. L'obiettivo di questo documento è quello di sensibilizzare e mettere in guardia gli sviluppatori riguardo le conseguenze di debolezze che possono compromettere le applicazioni Web. Il documento si basa sui dati di oltre quaranta aziende specializzate nella sicurezza delle applicazioni e su un'indagine di settore che ha coinvolto 500 persone. Ciò fa sì che i dati riguardino le vulnerabilità di centinaia di organizzazioni e più di 100000 applicazioni. Prima di elencare i rischi citati nel documento, gli esperti della *OWASP* forniscono suggerimenti aggiuntivi:

- Mettere in sicurezza un'applicazione Web non significa fermarsi ai primi 10 rischi, ma esistono centinaia di problematiche che possono riguardare la sicurezza.
- Il documento, nel corso degli anni, cambierà e ciò sarà dovuto al naturale evolvere della tecnologia; quindi gli sviluppatori devono continuamente aggiornare il codice per metterlo a riparo da future vulnerabilità.
- Gli sviluppatori devono porsi in maniera positiva davanti al problema della sicurezza in quanto, sia la *OWASP* che altre organizzazioni, mettono a disposizione documenti, guide, e best practice che possono essere consultati per migliorare tale aspetto.
- È opportuno utilizzare, quando possibile, tool automatici. Gli attuali repository sono composti da milioni di righe di codice e le vulnerabilità possono essere davvero ostiche da individuare. In tali casi, è bene affidarsi a persone esperte in *application security* che useranno nella maniera più efficace gli strumenti software per rilevare tali vulnerabilità.
- I responsabili di una software-house deve rendere la sicurezza parte integrante della cultura in tutta l'organizzazione.

La versione più recente della *OWASP Top 10* risale al 2017 ed è mostrata nella Figura 2.11.

Secondo tale documento, i dieci maggiori rischi che affliggono le applicazioni Web sono i seguenti [11]:

- *Injection*: un'iniezione di codice avviene quando un attaccante invia dati non validi all'applicazione Web con l'intenzione di farle fare qualcosa di diverso da quello per cui è stata progettata. Queste vulnerabilità sono molto frequenti e si possono trovare nelle query *SQL*, *LDAP*, *XPath*, nei comandi del sistema operativo, nel parser *XML*, nelle query *ORM*, etc. Tutto ciò che accetta parametri di input può essere potenzialmente vulnerabile ad un attacco di iniezione di codice. L'iniezione di codice può comportare la perdita di dati, la loro divulgazione a soggetti non autorizzati, la perdita di responsabilità o la negazione di accessi. Monitorare, ed eventualmente bloccare, il dato di input è una pratica intelligente per prevenire tale vulnerabilità.
- *Broken Authentication*: una vulnerabilità di questo tipo può permettere ad un attaccante di utilizzare mezzi manuali o automatici per cercare di ottenere il controllo di un account utente o, peggio ancora, per ottenere il controllo completo del sistema. Gli attaccanti possono rilevare problemi nelle fasi di autenticazione con mezzi manuali e sfruttarli con strumenti automatizzati, tramite liste di

OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

Figura 2.11: *OWASP Top 10* del 2017

password e attacchi al dizionario. Per mettersi al riparo da questa minaccia può essere necessario definire una severa politica per le password, rinforzare le operazioni di login, registrazione e recupero delle credenziali, impostare un limite per i tentativi di login e migliorare la gestione della sessione.

- *Sensitive Data Exposure*: l'esposizione di dati sensibili è una delle vulnerabilità più diffuse. Essa consiste nel furto di dati *PII* (informazioni di identificazione personale) che sarebbero dovuti rimanere protetti. Tali furti sono, comunemente, noti come violazioni di dati. Esempio di dati sensibili violati sono: password, numeri di carte di credito, credenziali, informazioni sanitarie, etc. Cifrare sempre i dati in transito e utilizzare connessioni sicure, come HTTPS, permette di prevenire tale minaccia.
- *XML External Entities (XXE)*: *XXE* è un tipo di attacco che prende di mira tutte quelle applicazioni che effettuano il parsing di documenti *XML*. Per default, molti vecchi processori *XML* permettono di specificare una risorsa esterna, ovvero una *URI*, che è dereferenziata e valutata durante il processo di parsing. Gli attaccanti, solitamente, sfruttano tale falla di sicurezza per estrarre dati, eseguire una richiesta remota dal server, eseguire attacchi di denial-of-service, e molto altro ancora. Evitare serializzazione di dati sensibili, usare, quando possibile, un formato meno complesso per i dati, come il *JSON*, e disabilitare le risorse esterne *XML* sono considerate best practice per combattere il problema.
- *Broken Access Control*: un *access control* è un software che controlla l'accesso a determinate informazioni e funzionalità. La violazione di un *access control* con-

sente agli aggressori di aggirare le autorizzazioni e di eseguire comandi come se fossero utenti privilegiati, cioè come amministratori. Ad esempio, un'applicazione Web potrebbe consentire a un utente di cambiare l'account di accesso semplicemente cambiando parte di un URL, senza altre verifiche. Una regola molto semplice, consigliata dalla *OWASP* è "nega per default"; essa suggerisce che, ad eccezione delle risorse pubbliche, vengono negate i permessi di accesso alle risorse per impostazione predefinita;

- *Misconfigurazioni di sicurezza*: gli attaccanti tentano di sfruttare falle non protette, cartelle e file non protetti, pagine inutilizzate e accessi ad account predefiniti al fine di garantirsi un accesso non autorizzato o per acquisire conoscenza del sistema. Cattive configurazioni possono capitare ad ogni livello dello stack di un'applicazione Web, inclusi servizi di rete, Server Web, database, framework di sviluppo, container, etc. In questo caso, scanner automatici sono utili per rilevare tali misconfigurazioni.
- *Cross-Site Scripting (XSS)*: il *cross site scripting* è una vulnerabilità diffusa che riguarda molte applicazioni Web. Gli attacchi *XSS* consistono nell'iniettare, in un sito Web, script maligni lato client che utilizza, per l'appunto, il sito Web come metodo di propagazione. Una vulnerabilità di tipo *XSS*, consente ad un attaccante di iniettare contenuti in un sito Web e di modificarne la visualizzazione, costringendo il browser della vittima a eseguire il codice fornito dall'aggressore durante il caricamento della pagina. *XSS* è presente in circa due terzi di tutte le applicazioni. Se una vulnerabilità *XSS* non viene corretta con una patch, può essere molto pericolosa. In particolare, è possibile distinguere tre tipologie di questa vulnerabilità:
  - *Reflected XSS*: si ha quando i dati forniti dall'utente (di solito tramite form HTML) sono usati immediatamente dallo script lato server per costruire le pagine risultanti senza controllare la correttezza della richiesta.
  - *Stored XSS*: si verifica quando i dati forniti dall'attaccante vengono salvati sul server, e quindi visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione, senza aver eliminato la formattazione HTML dai messaggi visualizzati dagli altri utenti. È la variante più pericolosa.
  - *DOM XSS*: questa tipologia è differente dalle precedenti e si verifica quando un'applicazione usa uno script *JavaScript* lato client che elabora i dati da una fonte non attendibile, scrivendo, di solito, i dati all'interno del *DOM*.
- *Deserializzazione Insicura*: questa minaccia si rivolge alle numerose applicazioni Web che, spesso, serializzano e deserializzano i dati. Serializzare significa trasformare oggetti in byte grezzi, mentre deserializzare significare fare l'operazione opposta. Una vulnerabilità di questo tipo è dovuta alla deserializzazione di dati da fonti non affidabili, e ciò può portare a severe conseguenze, come, ad esempio, attacchi DDoS o esecuzione di codice remoto. Per prevenire tale minaccia si può monitorare la deserializzazione ed implementare controlli di tipo; tuttavia, se non necessario, l'unico modo per proteggersi realmente è quello di evitare la deserializzazione da fonti sconosciute;
- *Usare componenti con vulnerabilità note*: gli attaccanti cercano vulnerabilità in componenti software che possono, poi, utilizzare per orchestrare gli attacchi. Alcuni dei componenti più popolari sono utilizzati su centinaia di migliaia di siti

Web. Un aggressore che dovesse trovare una falla di sicurezza in uno di questi componenti potrebbe sfruttarla su centinaia di migliaia di siti vulnerabili. Per ridurre al minimo il rischio di impiegare componenti con vulnerabilità note, gli sviluppatori dovrebbero rimuovere i componenti inutilizzati dai loro progetti, oltre ad utilizzare solo quelle librerie software che provengono da fonti affidabili e che siano aggiornati;

- *Monitoraggio e Logging insufficiente*: l'importanza della sicurezza di un sito Web non può essere sottovalutata. Anche se la sicurezza al 100% non è un obiettivo realistico, ci sono molti modi per mantenere il sito Web regolarmente monitorato, in modo da poter intervenire immediatamente quando succede qualcosa. Molte applicazioni Web non adottano misure sufficienti per rilevare le violazioni dei dati. Il tempo medio di scoperta di una violazione è di circa 200 giorni dopo che è avvenuta. Questo dà agli aggressori molto tempo per causare danni prima che ci sia una risposta. L'*OWASP* raccomanda agli sviluppatori Web di implementare piani di logging e di monitoraggio, nonché piani di reazione agli incidenti, per assicurarsi di essere messi al corrente degli attacchi alle loro applicazioni;

Come già accennato, *OWASP Top 10* è un documento che viene periodicamente aggiornato, e la versione più recente risale al 2017. La versione del 2017 introduce, rispetto alla precedente datata 2013, due nuove minacce: il problema della serializzazione insicura e l'insufficienza di adeguati strumenti di monitoraggio e logging. Qualora ce ne fosse ancora bisogno, anche *OWASP* pone l'attenzione su attività di monitoraggio e di analisi dei log, al fine di rilevare anomalie ed attività sospette, esattamente come l'obiettivo che questo lavoro di tesi si propone di raggiungere.

3

Strumenti utilizzati

In questo capitolo verranno illustrati gli strumenti software utilizzati nella presente tesi. Nella prima parte si parlerà dello Stack ELK, entrando nel dettaglio dei singoli componenti, ovvero Elasticsearch, Logstash, Kibana e Beats. Nella fase seguente, dopo una breve introduzione a Python, il linguaggio di programmazione utilizzato, verranno descritte le librerie software impiegate nella fase di analisi. In particolare, saranno presentate le principali caratteristiche di Pandas, la libreria utilizzata per la manipolazione dei dati, Scikit-learn, per l'implementazione degli algoritmi di Machine Learning, e, infine, Keras, un framework specifico per reti neurali e Deep Learning. Nell'ultima sezione verrà presentato MISP, una piattaforma open source che favorisce la condivisione di informazioni e indicatori di sicurezza.

3.1 Elastic Stack

Elastic Stack è la piattaforma open source di *log management* utilizzata in questo lavoro di tesi. La piattaforma consiste di quattro componenti fondamentali: *Elasticsearch*, *Logstash*, *Kibana* e *Beats*. La Figura 3.1 mostra l'organizzazione dello stack software.

Come detto, lo stack si basa sull'insieme di quattro software; tuttavia questi non furono sviluppati tutti insieme. Nel 2004, Shay Banon stava lavorando alla costruzione di un motore di ricerca e creò *Compass*, un precursore di *Elasticsearch*. La seconda versione del motore di ricerca fu resa open source e ribattezzata "*Elasticsearch*". La risposta degli utenti fu molto positiva, soprattutto perchè trovarono il software semplice e facile da usare e ciò spinse Banon a fondare la *Elasticsearch Inc.* nel 2012. Nel periodo in cui quest'ultima fu fondata, altri due progetti open source ebbero un successo notevole. Jordan Sissel stava lavorando su *Logstash*, un modulo open source in grado di "ingerire" log ed inviarli ad uno "*stash*" scelto dall'utente, incluso *Elasticsearch*. Rashid Khan, invece, stava lavorando al problema della visualizzazione dei dati e fondò Kibana, una *User Interface* open source.

Shay, Jordan e Rashid si conoscevano da tempo, e ciascuna conosceva i progetti che stavano portando avanti gli altri due. Perciò decisero di unire le forze, dando vita allo *Stack ELK - Elasticsearch*, *Logstash* e *Kibana*. Nel 2015 nacque la necessità di utilizzare degli strumenti che consentissero di inviare, in maniera leggera, dati di

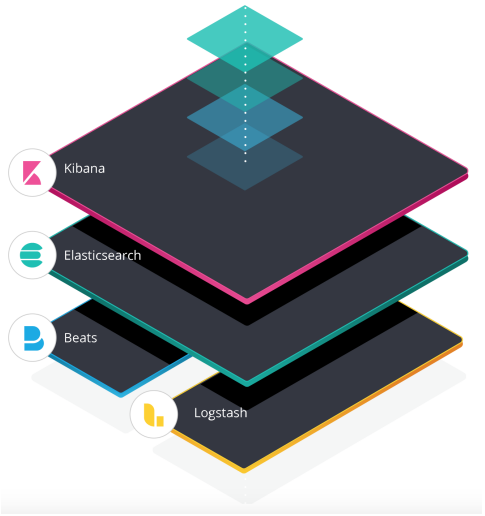


Figura 3.1: *Elastic Stack*

rete, log e metriche tra i software della famiglia *Elastic*: così fu sviluppato *Beats* e la piattaforma prese il nome di *Elastic Stack*. Nello stesso anno, il nome della società fu modificato e passò da *Elasticsearch Inc.* a *Elastic* [12].

Elastic Stack, quindi, è un'insieme di quattro software open source, progettati per fornire una soluzione completa di *log analysis end-to-end* che consente di ricercare, analizzare e visualizzare i log provenienti da diversi sistemi. Lo stack si compone dei seguenti componenti:

- *Elasticsearch*;
- *Logstash*;
- *Kibana*;
- *Beats*.

3.1.1 Elasticsearch

Elasticsearch è un motore di ricerca e di analisi distribuito ed open source che permette di gestire moltissimi tipi di dati, inclusi dati testuali, numerici, geospaziali, strutturati e non strutturati [13]. I punti di forza di *Elasticsearch* sono la velocità, la scalabilità, la natura distribuita e le semplici API REST che mette a disposizione.

La velocità e la scalabilità, unita alla capacità di indicizzare molti tipi di dati, ne consentono l'utilizzo in diversi casi d'uso; tra questi citiamo:

- ricerche aziendali;
- ricerche di siti web;
- *log analysis* e *logging*;
- monitoraggio delle metriche di un'infrastruttura;
- monitoraggio delle performance di un'applicazione;
- analisi di dati geospaziali;

- analisi di business;
- analisi di sicurezza.

Elasticsearch è un motore di ricerca basato su *Apache Lucene*; esso, tuttavia può, anche, essere considerato come un DBMS NoSQL di tipo *document store*. I dati sono rappresentati come veri e propri documenti testuali, espressi in formato JSON, a ciascuno dei quali sono associati un valore identificativo ed un insieme di coppie chiave-valore. I tradizionali DBMS, come, ad esempio, quelli relazionali, non sono realmente progettati per *query full-text* e sicuramente non funzionano bene con dati grezzi, tipicamente non strutturati o semi-strutturati. Sullo stesso hardware, query complesse, che richiederebbero qualche secondo utilizzando il linguaggio SQL, impiegano qualche millisecondo per essere eseguite su *Elasticsearch*. È bene dire, comunque, che la scelta di un qualunque database significa accettare dei compromessi; si può optare su *Elasticsearch* quando si richiede scalabilità (quindi questo tool è ideale per i *big data*), necessità di archiviare un singolo oggetto in maniera flessibile e ricerche veloci; il "costo" da pagare è la rinuncia ad operazioni di *join* e l'assenza del concetto di "transazione", due aspetti che, al contrario, rappresentano due punti di forza, ad esempio, di un sistema relazionale.

I dati grezzi confluiscono in *Elasticsearch* da una grande varietà di sorgenti dati, quali log, applicazioni web e metriche di sistema. Nella fase di *Data Ingestion*, i dati sono analizzati, normalizzati ed arricchiti, per poi essere indicizzati da *Elasticsearch*. Una volta avvenuta l'indicizzazione, gli utenti possono eseguire query complesse ed usare degli aggregatori che forniscono un riepilogo dei lori dati.

In *Elasticsearch* è presente il concetto di indice; quest'ultimo è una raccolta di documenti collegati tra loro. Un documento, memorizzato come JSON, collega un insieme di chiavi (nomi di campi o proprietà) con i valori corrispondenti (stringhe, numeri, valori booleani, date ed altri tipi di dati). *Elasticsearch*, come detto precedentemente, è un motore di ricerca che consente di effettuare *query full-text*; queste sono tipologie di query che consentono di analizzare campi testuali come, ad esempio, il corpo di una email.

Le *query full-text*, in *Elasticsearch*, sono molto efficienti, e ciò è dovuto al concetto di *inverted index*, un aspetto ereditato da *Apache Lucene*, il motore di ricerca su cui si basa. Tale caratteristica esprime il modo in cui i documenti sono correlati. Dati un insieme di documenti, dopo una fase di *text processing* (volto a portare tutto il testo in minuscolo, a rimuovere la punteggiatura e le *stopwords*), un *inverted index* consiste in una lista di parole che rappresentano tutte le parole non ripetute che appaiono nei documenti; a ciascuna parola sarà associata una lista dei documenti in cui essa appare.

La Figura 3.2 mostra la metodologia appena descritta.

L'*inverted index* è simile all'indice dei termini presenti alla fine di un libro, che mostra le pagine in cui un termine è presente o viene discusso. L'*inverted index*, quindi, rende più facile la risoluzione di query testuali a documenti specifici che potrebbero essere correlati, sulla base delle parole chiave presenti nella query, e velocizza il processo di recupero dei documenti limitando lo spazio di ricerca da considerare. Durante il processo di indicizzazione dei documenti, *Elasticsearch* memorizza quest'ultimi e costruisce un *inverted index* per rendere i dati del documento ricercabili in tempo quasi reale.

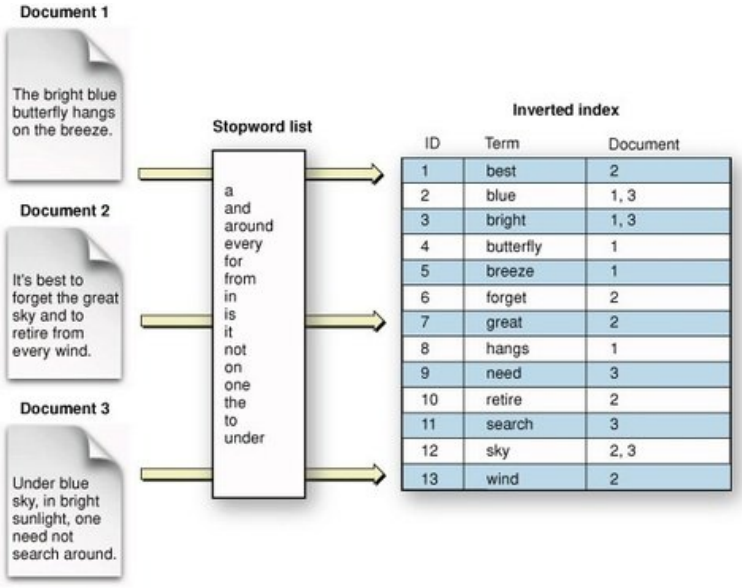


Figura 3.2: Processo di generazione di un *inverted index*

Uno dei notevoli vantaggi di *Elasticsearch* è la possibilità di utilizzare le numerose API REST per integrare, gestire ed interrogare i dati indicizzati in tantissimi modi diversi. Ad esempio, aggiungere un nuovo documento in *Elasticsearch* è semplice, e questo processo può essere automatizzato; infatti, con una semplice richiesta HTTP di tipo POST il documento viene inviato come un oggetto JSON. Anche le query possono essere poste utilizzando il JSON: la query viene inviata come una richiesta HTTP di tipo GET in cui il body della richiesta, in JSON, ne specifica i criteri. Le API REST, quindi, semplificano il recupero, l'invio e la verifica dei dati direttamente da riga di comando, ad esempio, in ambiente *Unix*, utilizzando il software *cURL*. Tuttavia, *Elasticsearch* supporta molti linguaggi di programmazione, e client ufficiali sono disponibili per Java, JavaScript, Go, C#, PHP, Perl, Python e Ruby.

3.1.2 Logstash

Logstash, uno dei prodotti core dell'*Elastic Stack*, è usato per processare, aggregare ed inviare dati ad una specifica destinazione, come, ad esempio, *Elasticsearch*. *Logstash* è una pipeline open source di elaborazione dei dati che consente di fare *data ingestion* da più fonti contemporaneamente, arricchire e trasformare i rispettivi dati, prima di farli indicizzare da *Elasticsearch* [14].

Il punto di forza di *Logstash* è che, in maniera semplice, tramite un singolo file di configurazione, è possibile, dinamicamente, unificare dati da sorgenti diverse, per poi trasformarli, effettuare operazioni di *data cleaning* e *data enrichment*, ed, infine, inviare i dati ad un altro sistema.

Le pipeline di *Logstash* si basano su uno o più file di configurazione. Quest'ultimi sono documenti testuali, espressi in formato JSON, che si trovano solitamente in

ambiente *Unix* nella directory `/etc/logstash/conf.d` con estensione `.conf`. Una pipeline specifica da quali sorgenti recuperare i dati, quali trasformazioni applicare e dove inviarli.

Un file di configurazione di *Logstash* si compone, tipicamente, di tre parti: la sezione di input, i filtri e la sezione di output. Le sezioni di input e di output sono obbligatori, invece i filtri sono opzionali. La Figura 3.3 mostra il flusso di una pipeline in *Logstash*.

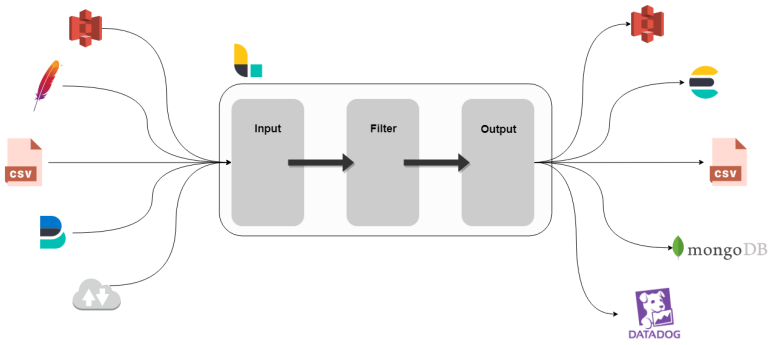


Figura 3.3: Flusso di una pipeline in *Logstash*

La sezione di input

La sezione di input specifica le sorgenti da cui prelevare i dati. *Logstash* è in grado di aggregare dati da più fonti, come file, Server Web Apache, Apache Kafka, etc, e aggiungerli in maniera efficiente ad una coda, per poi processarli da thread e *worker* multipli. Tutto ciò può essere fatto in maniera continuativa ed in streaming.

In generale, ad ogni fonte è associato uno specifico plugin di input. Attualmente, i plugin di input disponibili sono moltissimi e possono essere consultati alla pagina `https://www.elastic.co/guide/en/logstash/current/input-plugins.html`. Alcuni di esempio sono i seguenti:

- *beats*: usato per processare eventi provenienti da *Beats*;
- *file*: usato per leggere dati da un file sul file system;
- *exec*: recupera l'output di un comando della shell come un evento;
- *azure_event_hubs*: riceve eventi da Azure Event Hubs.

Logstash, per ogni dato di input acquisito, crea un evento; al termine di tale fase i dati raccolti saranno posti, di default, all'interno di un campo appositamente generato, ovvero il campo *message*.

I filtri

Mentre i dati viaggiano dalla sorgente all'output, *Logstash* analizza ogni evento, identifica i nomi dei campi per costruire una struttura e li trasforma in un formato comune.

Nella sezione relativa ai filtri, quindi, sono specificate le trasformazioni che devono essere applicate ai dati "*on-the-fly*", ovvero prima che siano inviati al sistema di output. Si può ben capire, quindi, che la sezione relativa ai filtri, seppur opzionale, svolge un ruolo di primo piano.

Anche in questo caso i filtri disponibili, consultabili all'indirizzo `https://www.elastic.co/guide/en/logstash/current/filter-plugins.html`, sono moltissimi. Esempi di filtri molto utilizzati sono i seguenti:

- `grok`: permette di strutturare ed effettuare il parsing di testo arbitrario; `Grok`, attualmente, è il modo migliore per ricavare la struttura da dati strutturati, al fine di poterli analizzare in maniera agevole.
- `mutate`: applica trasformazioni ai campi di un evento. Tramite *mutate* si possono rinominare, rimuovere, sostituire e modificare i campi degli eventi.
- `drop`: permette di rimuovere eventi, in base a determinate condizioni. Un esempio di utilizzo può essere la rimozione di eventi relativi a log di debug.
- `clone`: fa una copia di un evento, con la possibilità di aggiungere o rimuovere campi.
- `geoip`: aggiunge informazioni sulla posizione geografica degli indirizzi IP.
- `fingerprint`: consente di anonimizzare dati sensibili o di escludere completamente determinati campi.

Il plugin di output

La sezione di output è la parte finale di una pipeline *Logstash*. Il filtro di output contiene le direttive che indicano a *Logstash* dove esportare gli eventi, dopo che questi sono stati creati e trasformati tramite i filtri. Come si può ben immaginare, *Elastichearch* rappresenta il naturale sistema di output; tuttavia esistono altre possibilità.

In generale, i filtri di output più utilizzati sono i seguenti:

- `elasticsearch`: invia i dati ad un'istanza di *Elasticsearch*;
- `csv`: i dati sono scritti su un file del file system nel formato CSV;
- `s3`: i dati sono inviati ad Amazon S3.

I plugin di output disponibili possono essere consultati alla pagina `https://www.elastic.co/guide/en/logstash/7.5/output-plugins.html`.

3.1.3 Kibana

Kibana è un'applicazione open source che si trova in cima all'*Elastic Stack*, fornendo funzionalità di ricerca e visualizzazione per i dati indicizzati in *Elasticsearch*. Inoltre, *Kibana* funge sia da interfaccia utente per il monitoraggio, la gestione e la messa in sicurezza di un cluster *Elastic Stack*, che da hub centralizzato per le soluzioni sviluppate sfruttando lo stack *Elastic* [15]. *Kibana*, quindi, permette di cercare, visualizzare ed interagire con i dati memorizzati negli indici di *Elasticsearch*, per poi effettuare analisi avanzate e visualizzare i dati tramite una varietà di grafici, tabelle e mappe.

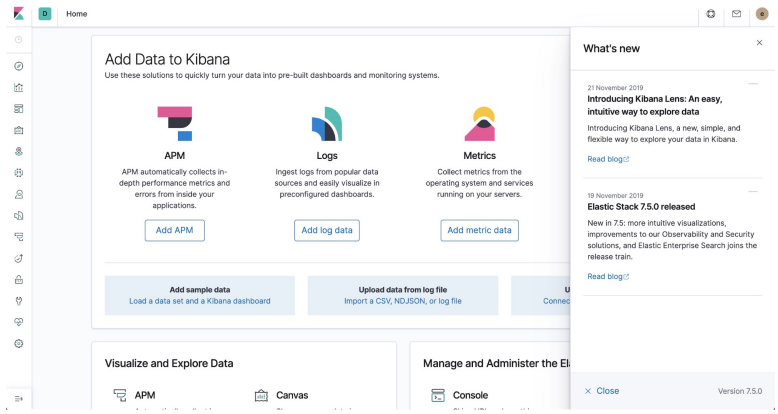


Figura 3.4: Home page di *Kibana* 7.5

Kibana utilizza una semplice interfaccia basata su browser per creare e condividere dashboard dinamiche che visualizzano, immediatamente, i cambiamenti alle query *Elastichsearch*. La Figura 3.4 mostra la home page di *Kibana* 7.5.

La configurazione della piattaforma è molto semplice: una volta installata si può direttamente iniziare a visualizzare i dati associati agli indici di *Elastichsearch*, senza nessuna infrastruttura aggiuntiva richiesta.

Nel menù a sinistra sono presenti le funzioni di *Kibana*; le più rilevanti sono: *Discover*, *Visualize*, *Dashboard* e *DevTools*. Esse verranno illustrate più in dettaglio nelle prossime sottosezioni.

Discover

Il pannello *Discover* consente di esplorare i dati, in maniera interattiva, al fine di individuare relazioni e peculiarità. La Figura 3.5 mostra il pannello *Discover* di *Kibana* 7.5.

In questo pannello, sostanzialmente, è possibile [16]:

- accedere ad ogni documento, il cui indice appartiene al *index pattern* selezionato;
- cercare i dati e filtrare i risultati;
- visionare i dettagli dei documenti che corrispondono ad una ricerca;
- visualizzare i documenti che si sono verificati prima o dopo un documento.

La prima cosa da fare in *Discover* è selezionare un *index pattern* nell'opzione in alto a sinistra. Quest'ultimo definisce quali sono i documenti che devono essere visualizzati; essi, infatti, saranno quelli il cui indice appartiene all'*index pattern* selezionato. Per default, *Discover* mostra i dati degli ultimi 15 minuti; tuttavia, se i dati contengono delle informazioni temporali, è possibile aumentare l'intervallo di tempo modificando l'apposita opzione in alto a destra.

Dopo aver definito i documenti da visualizzare ed impostato l'intervallo temporale, si può iniziare ad effettuare delle ricerche. Quest'ultime vengono effettuate tramite la barra di ricerca in alto; esistono due modalità di ricerca, ovvero usando il *Kibana Query Language*, che offre una sintassi di query semplificata, oppure adottando la sintassi di ricerca di *Lucene*, selezionabile dal menù *KQL*.

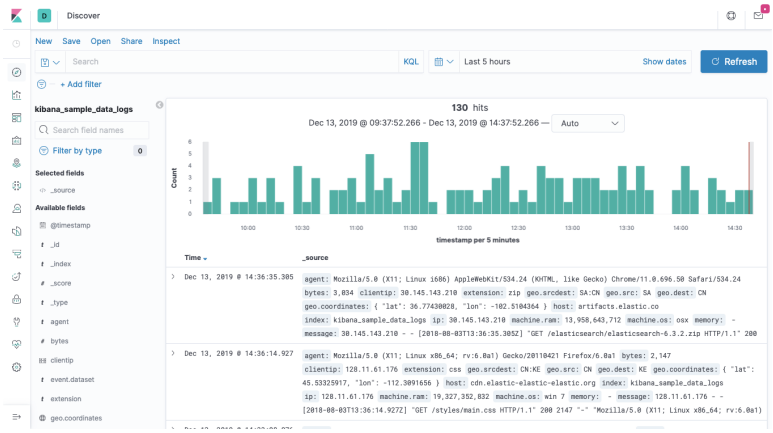


Figura 3.5: Il pannello *Discover* di *Kibana* 7.5

Dopo aver effettuato una ricerca è possibile utilizzare il menù a sinistra, *"Available fields"*, per filtrare quest'ultima rispetto a campi specifici. Quando si clicca un campo nella corrispettiva lista si visualizzeranno i primi cinque valori del campo e la percentuale del numero di documenti che contengono quel valore. La zona centrale di *Discover* contiene la tabella dei documenti; quest'ultima può essere espansa per esaminare un singolo documento insieme ai suoi campi in formato tabellare o JSON. Infine, si può salvare o condividere una ricerca; tramite le azioni *Save* e *Share*, presenti nella barra del menù, è possibile, ad esempio, esportare i dati in un file CSV o creare un link diretto da condividere.

Visualize

Il pannello *Visualize* consente di creare dei grafici relativi ai documenti degli indici di *Elasticsearch*, che potranno, poi, essere aggiunti alle dashboard per l'analisi. I grafici di *Kibana* si basano su query *Elasticsearch*. Utilizzando operatori di aggregazione di *Elasticsearch* per processare i dati, è possibile creare grafici che mostrano tendenze, picchi ed anomalie.

A partire dalla home page di *Kibana*, i passi per creare un grafico sono i seguenti:

1. Aprire il pannello *Visualize*.
2. Fare click su "Crea nuova visualizzazione".
3. Scegliere un tipo di visualizzazione; viene messa a disposizione una grande varietà di grafici, tra cui:
   - *grafici base*: metriche, tabelle, grafici a torta, grafici a barre, linee e aree;
   - *grafici ottimizzati per serie temporali*: *TSVB* e *Timelion*;
   - *mappe*: mappe basate su regioni geografiche o su coordinate, *Elastic Maps* e mappe di calore.
4. Definire una query di ricerca per specificare i dati del grafico:
   - prima di inserire i criteri della ricerca, è necessario selezionare l'*index pattern* per definire quali sono gli indici coinvolti nella ricerca;

- è possibile costruire un grafico a partire da una ricerca salvata; in tal caso, si deve fare click sul nome della ricerca che si decide di utilizzare.

Dashboard

Dashboard è il pannello in cui è possibile creare, modificare e visualizzare le dashboard create. Una dashboard contiene più visualizzazioni e grafici in una sola pagina. La Figura 3.6 mostra un esempio di dashboard in *Kibana* 7.5.

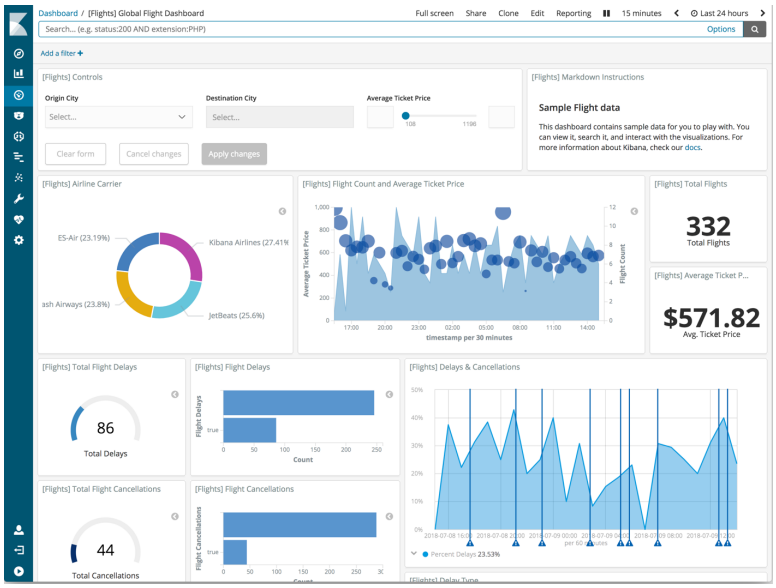


Figura 3.6: Dashboard d'esempio in *Kibana* 7.5

I grafici presenti nelle dashboard possono, poi, essere ulteriormente filtrati tramite una query di ricerca o facendo click su uno dei campi in essi presenti.

Per creare una dashboard è necessario avere dati indicizzati in *Elasticsearch*, un *index pattern* definito e mappe, ricerche salvate ed altre tipologie di grafici. Dopo aver definito i prerequisiti necessari, per creare una dashboard occorre:

1. Aprire il pannello *Dashboard*.
2. Fare click su "Crea nuova dashboard".
3. Fare click su "Aggiungi".
4. Usare "Aggiungi pannelli" per aggiungere elementi alla dashboard: i grafici, le ricerche salvate e le mappe si trovano nella sezione relativa ai pannelli e, dopo averli selezionati, possono essere modificati, spostati e ridimensionati.
5. Quando le modifiche sono concluse, salvare la dashboard tramite l'apposita opzione.

DevTools

DevTools è una delle feature più importanti di *Kibana* e consiste in un insieme di strumenti che consentono di interagire con i dati di un cluster *Elasticsearch*. I tre strumenti sono: la *Console*, il *Search Profiler* ed il *Grok Debugger*.

La *DevTools Console* permette, nella stesso pannello, di eseguire query *Elasticsearch* e di visionare i risultati restituiti. La *Console* è in grado di fornire suggerimenti; quindi, scrivere una query è molto semplice. Come si può vedere dalla Figura 3.7, in *Console* sono presenti due sezioni distinte: una per costruire ed eseguire query *Elasticsearch* ed una per visualizzare i risultati. Nel pannello a sinistra si possono scrivere query *Elasticsearch*; durante la digitazione, la *Console* ci fornirà dei suggerimenti sintattici, rendendo questo processo molto semplice.

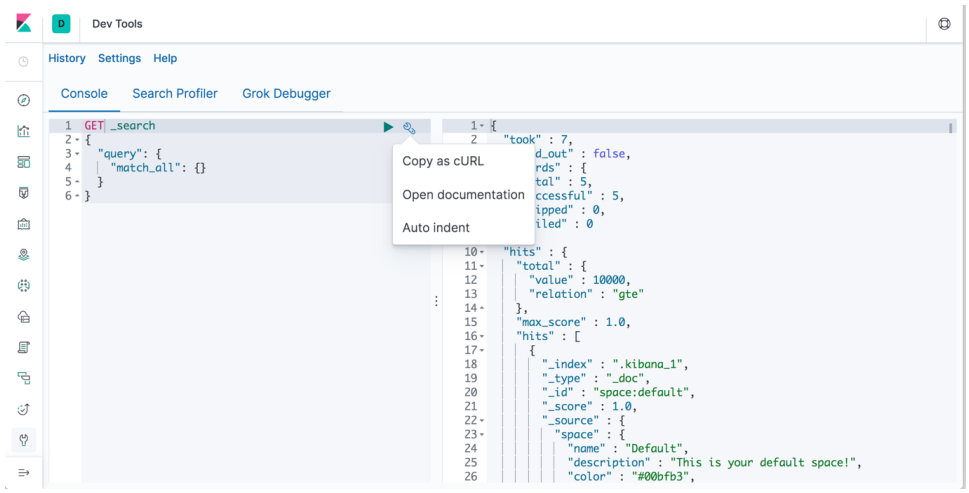


Figura 3.7: La *Console* di *DevTools* in *Kibana* 7.5

Dopo aver creato la query di ricerca, cliccando sulla freccia verde, è possibile avviare l'esecuzione; a seguito di ciò si possono visionare immediatamente i risultati nel pannello a destra.

Query Profiler è uno strumento di *Kibana* che appartiene al pacchetto *X-Pack*. *X-Pack* è un'estensione *Elastic Stack* che fornisce funzionalità aggiuntive riguardo sicurezza, allarme, monitoraggio, reporting, *Machine Learning*, e molto altro. *X-Pack* può essere testato usufruendo di un periodo di prova di 30 giorni al termine del quale, per continuare ad utilizzare le sue funzionalità aggiuntive, sarà necessario acquistare un abbonamento. Con *Search Profiler* si può tracciare le query in esecuzione e individuare quelle che hanno problemi di performance.

Kibana Grok Debugger è uno strumento di supporto per costruire dei pattern di tipo *grok*. Un pattern *grok* può essere utilizzato per strutturare dei dati non strutturati, esattamente allo stesso modo di quanto può essere fatto in *Logstash*. Non è semplice scrivere un pattern *Grok*. Per verificare se i risultati restituiti sono corretti, si dovrebbero effettuare dei test, eseguendo ogni volta dei file di configurazione di

Logstash. *Kibana Grok pattern* rende semplice questo processo perchè simula l'esecuzione di un pattern *Grok*; i risultati di tale simulazione possono essere visualizzati nella stessa pagina.

3.1.4 Beats

Beats è l'ultimo componente che costituisce l'*Elastic Stack*. I *Beats* sono, essenzialmente, degli agenti software leggeri che hanno lo scopo di acquisire dati e spedirli a *Logstash* o *Elasticsearch*. *Beats* è uno strumento software che viene installato nei server da cui si desidera ricevere i dati; in tal modo, si possono inviare i dati di centinaia o migliaia di macchine direttamente al cluster *Elasticsearch* o a *Logstash*, nel caso sia necessario un'ulteriore elaborazione.

Attualmente, sette tipi di *Beats* sono sviluppati ufficialmente da *Elastic*. Questi sono:

- *Filebeat*: è progettato per leggere file da un file system. È particolarmente utile per i file di log delle applicazioni e dei sistemi. In aggiunta, *Filebeat* facilita il processo di configurazione includendo "moduli" per l'acquisizione dei formati di file di log più comuni, come *MySQL*, *Apache*, *NGINX* e altri. Questi moduli riducono la configurazione di *Filebeat* ad un unico comando.
- *Metricbeat*: come suggerisce il nome, *Metricbeat* è usato per raccogliere metriche da server e sistemi. Uno dei maggiori vantaggi di *Metricbeat* è la sua leggerezza; esso può essere installato sui sistemi senza influire sulle prestazioni.
- *Packetbeat*: è un analizzatore di pacchetti di rete e monitora i protocolli di rete per consentire agli utenti di tenere sotto controllo la latenza di rete, gli errori, i tempi di risposta, le prestazioni SLA, i login degli utenti e molto altro.
- *Winlogbeat*: è uno strumento specificatamente progettato per fornire flussi stream di dati in ambiente *Windows*. *Winlogbeat* può leggere log da qualsiasi canale *Windows*, inclusi file di log. Esso può agire come uno strumento di miglioramento della sicurezza, in quanto rende possibile, per un'azienda, tenere sotto controllo tutto ciò che accade su un host *Windows*.
- *Auditbeat*: monitora l'attività degli utenti e dei processi. I dati degli eventi vengono analizzati e inviati, in tempo reale, ad *Elasticsearch* per monitorare la sicurezza dell'ambiente.
- *Heartbeat*: invia i dati necessari per il monitoraggio dei tempi di attività. Monitora i servizi tramite operazioni di *ping* e poi spedisce i dati ad *Elasticsearch* per l'analisi e la visualizzazione. *Heartbeat* può effettuare il *ping* utilizzando i protocolli *ICMP*, *TCP* e *HTTP*. *Heartbeat* garantisce il supporto per *TLS*, autenticazione e proxy. La sua efficiente risoluzione DNS gli permette di monitorare ogni singolo host posto dietro un server con *load balancer*.
- *Functionbeat*: permette di effettuare *data ingestion*, nonchè di trasformare, arricchire ed inviare dati da una qualunque sorgente cloud a *Elasticsearch*.

3.2 Python e Librerie

Python è il linguaggio di programmazione scelto per questo lavoro di tesi. Esso è un linguaggio di programmazione ad alto livello, interpretato e general-purpose. Si

è deciso di utilizzare la Versione 3.6; tuttavia la scelta di utilizzare Python *3.x* è praticamente obbligata, in quanto la Versione *2.x* è in *end-of-life* dal 1° Gennaio 2020.

Attualmente, Python, oltre ad essere uno dei linguaggi di programmazione più utilizzati in assoluto, è lo *standard de facto* per l'analisi dei dati e l'uso di algoritmi di Intelligenza Artificiale.

Esistono molti motivi che hanno portato Python ad essere così popolare; uno è, senza dubbio, la sua semplicità. Durante lo sviluppo di un'applicazione software è necessario concentrarsi sulla qualità del codice per semplificare la manutenzione e gli aggiornamenti. La sintassi di Python consente di esprimere dei concetti semplici, senza sforzi aggiuntivi, rendendo il codice pulito e leggibile. La semplicità del linguaggio porta con sè dei vantaggi ulteriori. In primo luogo, esso fa sì che Python sia un linguaggio semplice da imparare; in secondo luogo, gli sviluppatori possono, così, impiegare completamente il loro tempo nella generazione di modelli di Intelligenza Artificiale, piuttosto che concentrarsi sulle sfumature tecniche del linguaggio.

La scelta di utilizzare Python è, inoltre, motivata dal fatto che permette di accedere ad un numero praticamente infinito di librerie di terze parti e framework, incoraggiando la modularità dei programmi ed il riuso del codice. Ad esempio, implementare algoritmi di Intelligenza Artificiale può essere complicato e richiede molto tempo. È fondamentale avere un ambiente ben strutturato e collaudato, in modo tale da trovare, senza molto sforzo, le migliori soluzioni di *coding*. Molto spesso, per ridurre i tempi di sviluppo, i programmatori si rivolgono ad un certo numero di librerie e strumenti di sviluppo, messi a disposizione da altri sviluppatori per tutti. Nell'ambito dell'analisi dei dati e dell'Intelligenza Artificiale, Python fornisce un ampio set di librerie. Alcune di esse sono:

- *Keras*, *TensorFlow* e *Scikit-learn* per l'Intelligenza Artificiale;
- *NumPy* e *Pandas* l'analisi dei dati general-purpose;
- *SciPy* per elaborazioni avanzate;
- *Matplotlib* per la *data visualization*.

Un fattore ulteriore che contribuisce alla popolarità di Python è che esso permette di sviluppare applicazioni e script compatibili con la maggior parte dei sistemi e piattaforme. Python, infatti, è supportato da molte piattaforme, tra cui Linux, Windows e macOS. Python è un linguaggio di programmazione interpretato; quindi è possibile eseguire lo stesso codice, senza che questo sia ricompilato, su sistemi diversi, purchè si utilizzi lo stesso interprete. In alternativa, a partire dal codice sorgente, si possono creare dei programmi eseguibili per i principali sistemi operativi, consentendo, quindi, la distribuzione del software anche in quei sistemi in cui non è installato un interprete Python.

Infine, come già detto precedentemente, Python è uno dei linguaggi più diffusi in assoluto, e ciò si traduce in una community di sviluppatori molto vasta e attiva; ciò spinge sempre più volontari a sviluppare e pubblicare le loro librerie. La diffusione della community permette, di conseguenza, un notevole supporto per coloro che desiderano trovare soluzioni ai loro problemi di sviluppo.

3.2.1 Pandas

Pandas è una libreria software, scritta in Python, che fornisce strutture dati flessibili, efficienti, efficaci ed espressive. Essa è stata progettata per gestire dati etichettati o strutturati in maniera semplice e veloce [17]. *Pandas* si basa su *Numpy*; anche quest'ultima rappresenta una popolare libreria Python. *Numpy* fornisce supporto per matrici ed array multidimensionali, insieme ad una vasta collezione di operazioni ad alto livello che agiscono, in maniera efficiente, su tali strutture dati [18].

In un processo di *data mining*, molto spesso, prima di definire un modello di Intelligenza Artificiale, la maggior parte del tempo è speso in attività di *data cleaning* e di *data trasformation*; è proprio in queste attività che *Pandas* mostra i suoi punti di forza, mettendo a disposizione degli oggetti che consentono di gestire grandi tabelle di dati in maniera efficace ed efficiente.

Alcune importanti funzionalità di *Pandas* sono:

- gestione flessibile di valori mancanti;
- inserimento e cancellazione di colonne nelle strutture dati;
- allineamento automatico ed esplicito dei dati: i valori possono essere allineati con un insieme di etichette, oppure si può lasciare che *Pandas* li allinei automaticamente;
- possibilità di effettuare operazioni di *merge* e *join* tra più dataset;
- ridimensionamento e pivoting delle strutture dati;
- capacità di manipolare i dati sfruttando gli indici delle strutture dati;
- selezione di un sottoinsieme dei dati rispetto ad etichette o indici;
- supporto per l'analisi di serie temporali;
- ordinamento dei dati in senso crescente o decrescente;
- supporto per il caricamento di dati da CSV, file Exel, file di testo, e molti altri formati;
- filtraggio dei dati rispetto ad una condizione.

In *Pandas* esistono due tipi di strutture dati, ovvero le *Series* ed i *DataFrame*.

Una *Series* è un oggetto monodimensionale, simile ad un array, contenente i dati e le etichette dei dati, chiamati indici. Se non viene specificato esplicitamente un indice per i dati, quest'ultimo assumerà valori che vanno da 0 a `N`, dove `N` rappresenta il numero di elementi contenuti nella *Series*.

Un *DataFrame* è una struttura dati bidimensionale, simile ad un foglio di calcolo elettronico, contenente un insieme di colonne, ciascuna delle quali può ospitare un tipo di dato differente (numerico, testuale, booleano, etc.). Un *DataFrame* ha indici sia per le righe che per le colonne. Come già accennato in precedenza, un *DataFrame* è in grado di ospitare dati bidimensionali; tuttavia, sfruttando l'indicizzazione gerarchica, ovvero un'ulteriore funzionalità di *Pandas*, si possono rappresentare, sempre nel formato tabellare, dati a più dimensioni.

3.2.2 Scikit-Learn

Scikit-Learn è la libreria software di riferimento per implementare algoritmi di *Machine Learning* in Python [19]. Il progetto è stato avviato nel 2007 da David Cournapeau, all'interno del Google Summer of Code e, da allora, molti volontari hanno

portato avanti il lavoro. La libreria è completamente open source. Attualmente si è arrivati alla Versione 0.22, mantenuta da un team di volontari. *Scikit-Learn*, internamente, si basa su *NumPy*, *SciPy* e *matplotlib*, importantissime librerie Python già citate in precedenza. Essa consente di eseguire numerose operazioni e mette a disposizione una grande varietà di algoritmi; ciò fa sì che, insieme ad un'eccellente documentazione riguardo classi, metodi e funzioni, sia relativamente semplice implementare dei modelli di *Machine Learning*.

Scikit-Learn, come detto, offre numerose funzionalità. Alcune di esse sono le seguenti:

- *data pre-processing*;
- algoritmi di riduzione della dimensionalità;
- strumenti per la selezione del modello;
- algoritmi di regressione;
- algoritmi di classificazione;
- algoritmi di *clustering*.

In aggiunta, vengono forniti, per default, dei dataset che consentono di testare i modelli sviluppati.

Scikit-Learn è uno strumento davvero utile; tuttavia, esistono dei task di Intelligenza Artificiale che non sono supportati, infatti, ad esempio, non è possibile implementare reti neurali *Deep* e reti SOM (*Self-Organizing Maps*), individuare regole associative ed implementare algoritmi di *reinforced learning*.

La Figura 3.8 fornisce una panoramica delle funzionalità di *Scikit-Learn* in base al tipo di problema da affrontare.

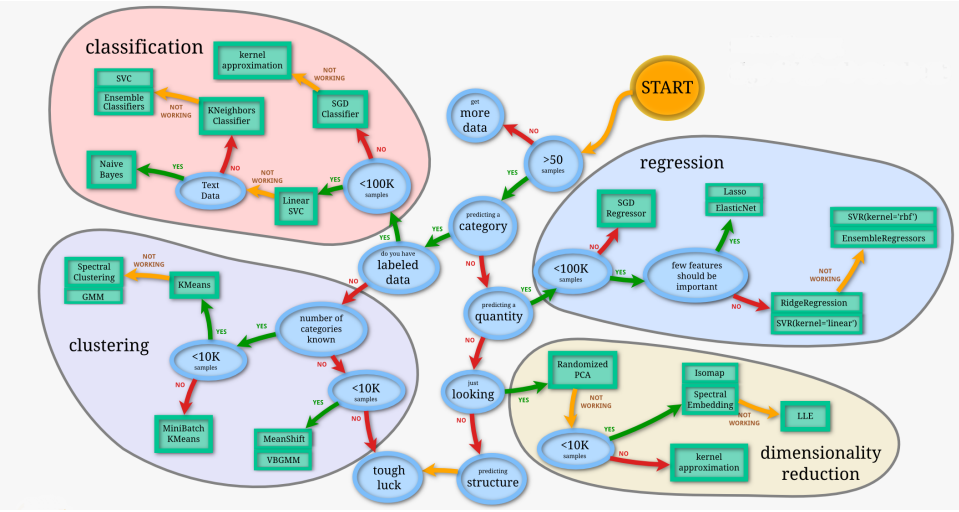


Figura 3.8: Panoramica delle funzionalità offerte da *Scikit-Learn*

Data Pre-Processing

Il package `sklearn.preprocessing` fornisce diverse funzioni che consentono di trasformare i dati grezzi in una rappresentazione più adatta per gli algoritmi di *Machine Learning*. In alcuni casi, ad esempio, può essere necessario trasformare i dati in modo tale che i valori di una colonna abbiano media zero e deviazione standard pari ad 1; in questi casi si può applicare la funzione `StandardScaler`. Come detto, *Scikit-Learn* utilizza, internamente, *NumPy* e, quindi, come si può ben capire, si possono utilizzare solo valori numerici, un'attività di *pre-processing* molto frequente è quella di trasformare feature categoriche in valori numerici, ad esempio utilizzando una specifica funzione che implementi il *One-Hot Encoder*. In altri casi, i dati da utilizzare in fase di analisi possono avere scale diverse e, quindi, può essere necessario un processo di normalizzazione con la funzione `normalize`.

Riduzione della dimensionalità

Quando il numero di dimensioni, o *feature*, è troppo elevato, può essere utile sfruttare algoritmi non supervisionati per ridurre la dimensionalità del problema. La riduzione della dimensionalità consente di selezionare le *features* più importanti di un dataset multidimensionale. *Scikit-Learn* offre diversi approcci alla riduzione della dimensionalità; uno di questi è la *Principal Component Analysis*, o PCA.

Selezione del modello

Quando si addestrano e allenano modelli di *Machine Learning* è necessario suddividere il dataset in *training set* e *testing set*: la funzione `model_selection.train_test_split()` può essere utile in questi casi.

Oltre a suddividere il dataset, *Scikit-Learn* mette a disposizione delle funzioni che possono essere utilizzate per selezionare il miglior modello di *Machine Learning*, come, ad esempio, la *cross-validation*, il *tuning* dei parametri con il *search grid* ed il calcolo delle metriche di performance di un modello (errore quadratico medio, matrice di confusione, *f-measure* e molte altre).

Regressione

Con gli algoritmi di regressione, dati un insieme di *feature* di input, è possibile predire in output un valore numerico.

In *Scikit-Learn*, per quanto riguarda la regressione, ci sono molti algoritmi tra cui scegliere: dalla regressione lineare e polinomiale, fino ad arrivare agli algoritmi di regressione basati su *Support Vector*, alberi decisionali e *Nearest Neighbors*. Si possono, inoltre, utilizzare algoritmi di regressione che si basano su reti neurali (non *Deep*); tuttavia *Scikit-Learn* non può essere considerata all'altezza di librerie specialistiche, come, ad esempio, *TensorFlow*.

Classificazione

Fare classificazione significa predire la classe di apparenza, prestabilita a priori, di un nuovo elemento.

Anche in questo caso, *Scikit-Learn*, mette a disposizione una grande varietà di algoritmi: *k-nearest neighbors* (k-NN), alberi decisionali, *Support Vector Machine* (SVM), classificatore bayesiano, e molti altri. Alcuni classificatori hanno il suffisso *CV* nel loro nome: questa notazione sta a significare che implementano la cross-validation per default. *Scikit-Learn* consente, anche, di utilizzare degli algoritmi, ad esempio, il `VotingClassifier` o il `BaggingClassifier`, che permettono di implementare un *Ensemble Learning*. Questo vuol dire utilizzare classificatori con caratteristiche differenti e tuttavia uniti in un certo modo per riuscire a massimizzare le prestazioni usando i punti di forza di ognuno e limitando le corrispettive debolezze.

Clustering

Clustering significa raggruppare oggetti omogenei, senza sapere, a priori, il numero di *cluster*; il *clustering*, quindi, rientra nell'ambito dell'apprendimento non supervisionato.

In *Scikit-Learn* sono presenti numerosi algoritmi di *clustering*: dai più famosi *K-Means* e *DBSCAN*, fino ad arrivare agli algoritmi di *clustering* gerarchico, agglomerativi, e molti altri.

3.2.3 Keras

Keras è una libreria open source, scritta in Python, che fornisce API di alto livello per implementare reti neurali e *Deep Neural Network*, sfruttando, ad un livello inferiore, le funzionalità di *TensorFlow*, *CNTK*, o *Theano* [20]. *Keras* è uno dei framework per il *Deep Learning* più utilizzati, in quanto, grazie alla sua facilità d'uso e alla sua modularità, consente di progettare reti neurali in maniera semplice e veloce. *Keras* può essere eseguita indistintamente su CPU e GPU e, inoltre, fornisce supporto per reti neurali convoluzionali, reti neurali ricorrenti e una combinazione di quest'ultime.

Le caratteristiche di *Keras* possono essere riassunte nei seguenti punti:

- *User-friendly*: *Keras*, riprendendo un suo famoso motto, fornisce "API progettate per gli esseri umani, non per le macchine". In *Keras* la semplicità d'uso e la *user experience* sono delle caratteristiche fondamentali. *Keras* adotta delle best practice per ridurre il più possibile la curva di apprendimento: offre API semplici e coerenti, minimizza il numero di azioni dell'utente, necessarie per eseguire le più comuni operazioni, e fornisce un feedback chiaro e preciso riguardo gli errori in cui gli utenti possono incappare.
- *Modularità*: in *Keras*, un modello è inteso come una sequenza o un grafo di moduli autonomi e completamente configurabili che possono essere collegati insieme senza molte restrizioni. Nello specifico, gli strati di una rete neurale, le funzioni di costo, gli ottimizzatori, gli schemi iniziali e le funzioni di attivazione sono tutti moduli autonomi che possono essere combinati per creare nuovi modelli.
- *Facile estensibilità*: *Keras* consente di aggiungere nuovi moduli, intesi come classi e funzioni, in maniera semplice, mentre i moduli esistenti forniscono esempi esaustivi per la loro comprensione. Essere in grado di creare facilmente nuovi moduli consente un'espressività totale, rendendo *Keras* adatto per analisi molto avanzate.

- *Scritto in Python*: Python è utilizzato per implementare le reti neurali in *Keras* e ciò si traduce in una scrittura compatta e leggibile, nella semplicità nel debug e in una facile estensione verso altri sistemi.

Dei quattro punti cardini citati in precedenza, sicuramente, la principale ragione che ha contribuito alla popolarità di *Keras* è stata la possibilità di sviluppare modelli complessi in maniera semplice e senza molto sforzo. Oltre alla facilità di apprendimento e di costruzione del modello, *Keras* offre strumenti che consentono l'integrazione con diversi *engine* di *back-end* (*TensorFlow*, *CNTK*, *Theano*, *MXNet*, e *PlaidML*), fornisce un robusto supporto per moltissime GPU e, inoltre, consente un training di dati in ambiente distribuito.

Come menzionato in precedenza, *Keras* è una libreria software che fornisce API di alto livello che permettono di implementare reti neurali. Le operazioni di basso livello, come, ad esempio, i prodotti tra tensori e le convoluzioni, non sono fatte da *Keras*; tali operazioni sono eseguite da *engine* di *back-end*, sui quali, poi, si basa *Keras*. Con una visione semplificata, possiamo affermare che gli *engine* di *back-end*, che si trovano ad un livello inferiore, si occupano di fare i calcoli e di definire in maniera complessa i modelli, mentre *Keras* fornisce delle semplici API che consentono di astrarre la complessità dei modelli stessi, al fine di fornire una rapida sperimentazione. Attualmente, *Keras* supporta tre implementazioni di *engine* di *back-end*: il backend *TensorFlow*, il back-end *Theano* e il back-end *CNTK*. *TensorFlow* è l'*engine* di *back-end* di default, tuttavia questa opzione può essere modificata in qualunque momento. La Figura 3.9 mostra la struttura architetturale di *Keras*.

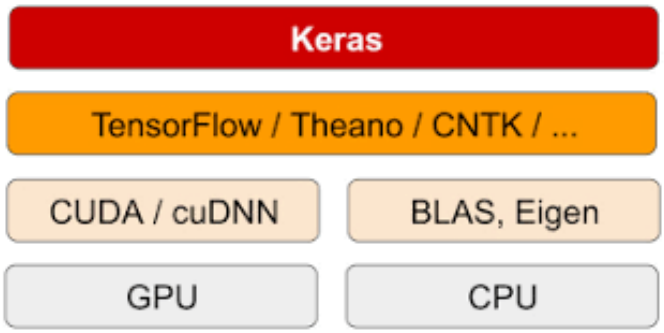


Figura 3.9: Architettura di Keras

Dal punto di vista pratico, in *Keras*, la struttura dati fondamentale è il *model*, che definisce il modo in cui organizzare i vari strati. La più semplice tipologia di modello è quello sequenziale che equivale ad uno stack lineare di strati. Per strutture più complesse si può utilizzare la *Keras functional API*. In linea generale, i passi da compiere per definire un modello sono i seguenti: dopo aver definito la tipologia del modello, sequenziale o tramite la *functional API*, gli strati della rete possono essere aggiunti, in maniera molto semplice, richiamando la funzione `.add()`. A questo punto, tramite la funzione `.compile()`, si può eseguire la configurazione del processo di apprendimento, definendo in maniera opportuna la funzione costo,

l'ottimizzatore e la metrica da tenere in considerazione. Seguono, infine, la fase di training, con il metodo `.fit()` e la fase di predizione su nuovi elementi, con il metodo `.predict()`.

Concludiamo questa sezione ricordando che *Keras* è supportato da Google, Microsoft, Amazon, Apple, Nvidia, Uber, e molte altre multinazionali, a testimonanzia del fatto che tale progetto è una soluzione di riferimento nell'ambito delle reti neurali e del *Deep Learning*.

3.3 MISP

MISP *(Malware Information Sharing Platform and Threat Sharing)* è una piattaforma software open source che facilita lo scambio e la condivisione di indicatori di compromissione (*Indicators of Compromise* o IOC) riguardo attacchi informatici, malware, vulnerabilità e truffe finanziarie all'interno di una community di utenti fidati [21]. Christophe Vandeplas trovava frustante che gli IOC venissero condivisi via e-mail o tramite documenti PDF, in quanto non potevano essere elaborati da sistemi automatici e, quindi, nel Giugno del 2011, fu avviato il progetto MISP, con l'obiettivo di consentire una migliore condivisione delle informazioni di sicurezza. La piattaforma condivisa di MISP è un modello distribuito contenenti informazioni tecniche e generiche che possono essere condivise all'interno di community pubbliche, semi-private o private. Nelle definizioni precedenti è stato anticipato il concetto di IOC: gli indicatori di compromissione, o IOC, sono delle porzioni di dati che, con molta probabilità, segnalano la presenza di un'intrusione in un sistema informatico; esempi di IOC sono signature, indirizzo IP e valore hash di un malware, URL o domini di una botnet, e così via.

Lo scambio di informazioni di sicurezza dovrebbe portare a un rilevamento più rapido degli attacchi informatici e a migliorare i sistemi di rilevamento, riducendo, allo stesso tempo, i falsi positivi. La condivisione delle informazioni è la chiave per una rapida ed efficace individuazione degli attacchi. Molto spesso, organizzazioni simili sono prese di mira dagli stessi malintenzionati. MISP rende più facile la condivisione consentendo, anche, un'analisi collaborativa ed impedendo, quindi, di svolgere del lavoro già fatto da qualcun altro in precedenza.

MISP, tra le altre cose, può essere utilizzato anche sfruttando le API REST messe a disposizione ed è facilmente estensibile tramite *misp-modules*. Inoltre, sono state sviluppate librerie aggiuntive, come, ad esempio, *PyMISP* che consente di interagire con MISP tramite Python.

In MISP, l'elemento centrale è il concetto di evento: un evento incapsula un insieme di informazioni contestualmente legate. Un evento, in MISP, si riferisce ad un evento di sicurezza e contiene tutte le informazioni che sono state raccolte. La Figura 3.10 mostra un evento MISP d'esempio, incluse tutte le informazioni correlate.

Un evento è descritto tramite i suoi attributi; gli attributi possono essere qualsiasi cosa che aiuti a descrivere l'evento, inclusi IOC, vulnerabilità e qualsiasi altra informazione correlata. Gli attributi, in MISP, possono essere indicatori di rete, ad esempio indirizzi IP, indicatori di sistema, ad esempio una certa stringa in memoria, o, anche, i dettagli di un account bancario. Ad ogni attributo è sempre associato un

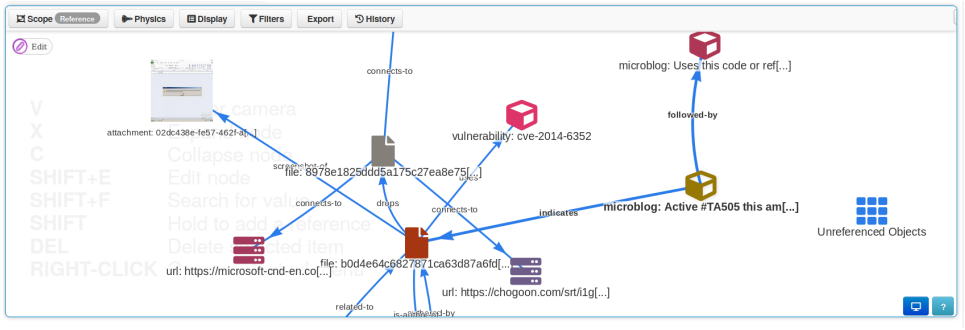


Figura 3.10: Esempio di evento in MISP

tipo, ad esempio “MD5” o “url”, che ne specifica la natura, ed una categoria, che ne definisce il contesto. In MISP è presente, inoltre, il concetto di oggetto: un oggetto è una combinazione di singoli attributi. La creazione degli oggetti MISP, con i relativi attributi, si basa su casi d’uso reali nell’ambito della sicurezza informatica.

Utilizzare MISP come piattaforma di *information sharing*, oltre a garantire i benefici di cui si è parlato precedentemente, permette di usufruire di numerose funzionalità. Le più importanti sono:

- Un database, che consente di archiviare IOC, informazioni tecniche e generiche riguardo malware, incidenti di sicurezza e attaccanti.
- La correlazione automatica, che consente di individuare relazioni tra attributi ed indicatori. Gli strumenti di correlazione includono correlazioni tra più attributi e correlazioni più avanzate, come, ad esempio, la correlazione di tipo *fuzzy hashing* (viene rivelato se input diversi hanno sequenze di byte identici nello stesso ordine). La correlazione è, in ogni caso, un’opzione che può essere disabilitata.
- Modello dei dati flessibile, dove gli oggetti complessi possono essere collegati insieme.
- Modalità facilitata per la condivisione dei dati utilizzando diversi modelli di distribuzioni. MISP può sincronizzare automaticamente eventi e attributi da istanze diverse.
- Interfaccia grafica intuitiva che consente, agli utenti finali, di creare, aggiornare e correlare eventi. Una specifica sezione permette di creare e visualizzare le relazioni tra oggetti e attributi. Vengono, inoltre, offerte funzionalità di filtraggio avanzate che consentono di analizzare in maniera semplice eventi ed attributi.
- I dati sono memorizzati in un formato strutturato e sono supportati numerosi indicatori di sicurezza.
- I dati presenti in MISP possono essere esportati in molti modi diversi: file di testo, CSV, OpenIOC (formato open source per la condivisione di informazioni), XML e JSON. Si possono, inoltre, generare regole per *Intrusion Detection System* (sono supportati, per default, *Snort*, *Suricata* e *Bro*).
- I dati possono essere importati da numerose tipologie di sorgenti: testo libero, OpenIOC, e molti altri.
- Possibilità di importare ed integrare in MISP qualsiasi feed di *threat intelligence* di terze parti o feed OSINT (Open Source INTelligence). Molti feed sono già

inclusi per default nell'installazione di MISP. Esempi di feed già inclusi sono: lista di *exit-node* di TOR, liste di IP in black-list, liste di domini malevoli, e così via.
- Insieme di API flessibili per integrare MISP con soluzioni proprietarie. Si può, ad esempio, utilizzare *PyMISP*, una libreria Python che consente di recuperare, aggiungere o aggiornare gli eventi di attributi, eseguire ricerche per attributi, e gestire "sample" di malware.
- Tramite *misp-modules* è possibile creare dei moduli, scritti in Python, che permettono di estendere MISP con servizi propri.

4

Modello di Anomaly Detection in uno scenario controllato

Il capitolo corrente illustra le fasi che si sono rese necessarie per lo sviluppo di un modello di Anomaly Detection, in uno scenario controllato, al fine di individuare le sessioni Web anomale. Innanzitutto, viene definito lo scenario di riferimento. Successivamente, viene descritto il dataset utilizzato nella fase di analisi. Dopo di ciò viene trattata la fase di esplorazione dei dati, al fine di individuare le caratteristiche salienti. A questo punto, viene definito l'approccio proposto in fase di analisi e vengono presentati gli algoritmi di Machine Learning che si è deciso di utilizzare. Le successive quattro sezioni illustrano le varie di fasi di cui si compone il processo di analisi. Il capitolo si conclude con la descrizione, e una successiva discussione, dei risultati ottenuti.

4.1 Scenario

L'obiettivo della presente tesi, come accennato nell'introduzione, è quello utilizzare algoritmi di Intelligenza Artificiale per classificare il traffico Web. L'intento, quindi, è quello di sviluppare un modello che definisca il profilo del comportamento standard; il comportamento di un utente che si discosta dalla norma sarà segnalato come anomalia. Riprendendo i concetti di *Anomaly Detection*, nel contesto delle applicazioni Web, un'anomalia potrebbe indicare l'utilizzo di uno scanner automatico per la ricerca di vulnerabilità, un attacco *bruteforce* per rubare le credenziali di un utente oppure, molto semplicemente, un certo insieme di azioni, non malevoli, che, nel complesso, risultano singolari; possiamo dire, quindi, che, nelle applicazioni Web, un attacco informatico è considerato come anomalia; tuttavia, non è sempre vero il viceversa.

Per facilitare la trattazione e lo sviluppo del modello, in questa fase, si è utilizzato un dataset semplificato che descriveremo nelle sezioni successive.

I log utilizzati in questa analisi, insieme ad altre informazioni specifiche, sono confidenziali e, quindi, vista la presenza di un NDA (*Non Disclosure Agreement*), non possono essere mostrati.

4.2 Dataset

Il dataset utilizzato per la generazione e la sperimentazione del modello di *Anomaly Detection* consiste in un insieme di log provenienti da un Server Web Nginx. Nello specifico, per le generazione del modello, si è utilizzato un dataset composto da 10550 richieste, che coprono un periodo di 10 mesi, mentre, per la fase di valutazione, è stato utilizzato un altro dataset, proveniente dallo stesso Server Web, composto da 1265 richieste, registrate in un periodo temporale successivo rispetto al primo, e che coprono un periodo di due settimane; l'idea, quindi, è quella di utilizzare una base storica di conoscenza per sviluppare il modello di comportamento normale, per poi utilizzare un altro insieme di dati, proveniente dallo stesso server, per la valutazione.

I dataset sono dei file di testo che contengono le richieste processate dal server. Nei file di log, per ogni richiesta, come già accennato nel Capitolo 2, vengono memorizzate le seguenti informazioni: *indirizzo IP*, *identd*, *userid*, *timestamp*, *richiesta* (comprendente *metodo*, *risorsa* e *protocollo*), *response code*, *content length*, *referer* e *user-agent*.

Come si può ben capire dai numeri citati in precedenza, dal punto di vista del volume dei dati, non si tratta di un problema di "*Big Data*", e ciò consente una manipolazione di questi ultimi senza molti sforzi; in aggiunta, per agevolare la costruzione e la sperimentazione del modello, entrambi i dataset hanno delle facilitazioni che riducono notevolmente la successiva fase di *Data Pre-Processing*. In particolare:

- I dataset sono stati etichettati da un esperto del dominio; ciò significa che sono stati identificati i singoli log che rappresentano un attacco o, in generale, un qualcosa di insolito. Relativamente all'analisi di log, la maggior parte delle volte non si ha a disposizione un dataset etichettato e, in questo caso, conoscere la classe di appartenenza dei singoli log permette, in primo luogo, di distinguere in maniera netta cosa è ammesso e cosa no, e, successivamente, apre un ventaglio di possibilità riguardo agli algoritmi di Intelligenza Artificiale da utilizzare.
- Il sito Web è semplice, sia in termini di lista di pagine di cui si compone sia nel modo in cui la Web Application è strutturata.
- Ai dati è già stata applicata una fase di *Data Cleaning*. In particolare, entrambi i dataset non contengono traffico generato da *bot*, *crawler* e *spider*. Tipicamente, la maggior parte del traffico Web è generato, in termini generali, da bot di motori di ricerca che hanno l'obiettivo di indicizzare il Web; questo tipo di traffico è fasullo, perché generato da agenti software automatici, facilmente individuabili e che non costituiscono un comportamento anomalo o malevolo.

4.3 Data Exploration

I log, come detto nella sezione precedente, risiedono in file di testo che, per loro natura, non consentono di esplorare, in maniera agevole, i dati a disposizione. In un processo di analisi, la fase di *Data Exploration* è molto importante perché consente di capire cosa contiene un dataset, le sue caratteristiche e la sua struttura dei dati al

fine di scoprire informazioni utili; inoltre, molto spesso, in questa fase, si utilizzano degli strumenti di *Data Visualization*, che consentono, tramite tabelle, grafici e metriche, di capire quali sono le caratteristiche chiave di un dataset.

Kibana è lo strumento di *Data Visualization* utilizzato e, di conseguenza, ciò significa configurare un *Elastic Stack* composto da *Elasticsearch*, *Logstash* e *Kibana*. La scelta di caricare i dati grezzi in *Elasticsearch* e, in generale, di adottare i software della famiglia *Elastic*, piuttosto che essere una spiacevole necessità, rappresenta un punto di forza. In questo caso, l'utilizzo dei software della famiglia *Elastic* consente di beneficiare di numerosi vantaggi, i più rilevanti dei quali sono i seguenti:

- *Elasticsearch* è un database NoSQL, di tipo *document store*, in grado di lavorare con grandi quantità di dati in maniera efficiente ed efficace; inoltre, allo stesso tempo, *Elasticsearch* è un motore di ricerca molto efficiente per query testuali o *query full-text*; esso è, quindi, ideale per l'analisi dei log; un ulteriore vantaggio sta nel fatto che esso può essere utilizzato tramite API REST e, di conseguenza, può essere integrato in moltissimi sistemi.
- *Logstash* funge da intermediario tra i dati grezzi ed *Elasticsearch*, consentendo di pulire, trasformare ed arricchire i dati "*on-the-fly*".
- *Kibana*, come detto, è uno strumento di *Data Visualization*, e, quindi, è ideale per la fase di esplorazione. In *Kibana* è possibile visualizzare nel dettaglio un singolo log, creare dei grafici che favoriscono la visualizzazione e utilizzare la *Console* di *DevTools* per definire query complesse.

4.3.1 Pipeline Logstash

Dopo aver installato e configurato i software della famiglia *Elastic*, per caricare i dati in *Elasticsearch*, come specificato nel Capitolo 2, è necessario definire una pipeline *Logstash* tramite le definizione di un file di configurazione con estensione `.conf`.

Il file di configurazione di *Logstash*, che definisce la pipeline necessaria per caricare i dati grezzi in *Elasticsearch*, è mostrato nel Listato 4.1.

```
1   ##Input section
2   input {
3     file {
4          path => "/path_to_logs/*"
5          type => "nginx"
6          sincedb_path => "/dev/null"
7          start_position => "beginning"
8       }
9   }
10
11  ##Filter section
12  filter {
13      grok {
14           match => { "message" => ["%{IP:clientip} %{GREEDYDATA:BindDN} \[%{GREEDYDATA:timestamp}\] \"%{WORD:method
                  } %{GREEDYDATA:path_request} \HTTP/%{NUMBER:httpversion}\" %{NUMBER:response} %{NUMBER:bytes} %{
                  QUOTEDSTRING:httpreferer} %{QUOTEDSTRING:httpuseragent}"]}
15           overwrite => [ "message" ]
16         }
17      mutate {
18          convert => ["response", "integer"]
19          convert => ["bytes", "integer"]
20          }
21      geoip {
22          source => "clientip"
23          target => "geoip"
24          }
25      date {
26          match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
27          target => "@timestamp"
28          }
29      useragent {
```

```
30          source => "httpuseragent"
31      }
32  }
33
34  ##Output section
35  output {
36    elasticsearch {
37      hosts => ["http://localhost:9200/"]
38      index => "logstash-nginx_%{+YYYY-ww}"
39    }
40  }
```

Listato 4.1: Pipeline *Logstash* utilizzata per caricare i dati grezzi in *Elasticsearch*

Come si può notare, la pipeline è composta da tre sezioni: la sezione di input, che specifica le sorgenti dati, quella relativa ai filtri, in cui vengono definite le trasformazioni da applicare, e quella di output, che indica dove dovranno essere inviati i dati.

La sezione di input

Nella sezione di input viene utilizzato il plugin *file*, in quanto i dati grezzi, ovvero i log, sono contenuti in file di testo. Nel plugin *file*, l'opzione `path` è obbligatoria e specifica il file o i file (viene supportata la notazione *wildcard*) da cui prelevare i dati. Gli altri parametri sono opzionali e sono: `type`, che specifica il tipo dell'evento, `sincedb_path`, che viene usato per tenere traccia dei file, e `start_position`, che indica da dove Logstash dovrebbe iniziare a leggere i file (può essere "*beginning*" o "*end*").

I filtri

La parte relativa ai filtri è, senza dubbio, la sezione più importante della pipeline *Logstash*, perchè consente di trasformare, pulire ed applicare processi di *Data Enrichment* ai dati, prima che essi siano spediti al plugin di output. È bene ricordare che il filtri sono applicati a cascata agli eventi e, quindi, l'ordine dei vari filtri è molto importante. Nello specifico i plugin utilizzati durante questa fase sono:

- *grok*: è uno dei più importanti plugin messi a disposizione da *Logstash*. Esso consente di analizzare del testo libero e strutturarlo in un certo formato. *Logstash*, per ogni dato di input acquisito, crea un evento, e il valore del campo "*message*" equivale al singolo log contenente tutte le informazioni ad esso correlate. In un file di testo contenente le richieste ricevute da un Server Web Nginx, le informazioni relative al singolo log, quali indirizzo IP, timestamp e così via, sono separate da uno spazio bianco, ed è questa la struttura che il plugin deve riconoscere. Con l'opzione `match` stiamo definendo un mapping tra un certo pattern ed il valore del campo "*message*"; il pattern specifica che, dato un evento (equivalente ad un singolo log), la prima stringa rappresenta l'indirizzo IP, poi vi è uno bianco, poi una seconda stringa che rappresenta il *BindDN*, poi ancora uno spazio bianco, e così via. Con l'opzione "*overwrite*" specifichiamo che, dato un evento, tutte le informazioni che non combaciano con il mapping definito devono sovrascrivere il valore del campo *message*. Come dicevamo, l'ordine dei filtri è importante; ciò significa che, dopo che un evento passa per il filtro corrente, esistono nuovi campi i cui valori sono quelli estrapolati grazie al pattern.

- *mutate*: è utilizzato per apportare delle modifiche a specifici campi. Nel caso in esame è stato utilizzata l'opzione `convert`, che permette di modificare il tipo dei valori associati ad un certo campo. In questo caso, si è deciso di modificare il tipo associato a "*response*" e "*bytes*", considerando entrambi come campi di tipo *integer*.
- *geoip*: questo filtro permette di aggiungere informazioni riguardo la posizione geografica di un indirizzo IP, in base ai dati presenti nel database GeoLite2 di Maxmind. Utilizzare questo filtro consente, quindi, di arricchire l'informazione riguardo l'indirizzo IP da cui proviene una certa richiesta; in questo modo, automaticamente, è possibile risalire, in maniera approssimativa, ma comunque utile, alla nazione, alla regione e alla città da cui proviene una certa richiesta. Il filtro *geoip* è stato utilizzato specificando due opzioni: *source* indica il campo contenente l'informazione riguardo l'indirizzo IP, mentre *target* specifica in quale campo *Logstash* dovrebbe memorizzare le informazione aggiuntive.
- *date*: questo filtro è utilizzato per analizzare le date dei campi, e quindi per utilizzare una certa data o timestamp per un evento. In questo caso si è deciso di utilizzare il filtro *date* per utilizzare, come timestamp dell'evento, il valore associato al campo *timestamp*, estratto grazie al filtro *grok*. Per fare ciò è stata utilizzata l'opzione `match`, in cui si sono specificati due parametri: il primo rappresenta il campo che contiene l'informazione temporale, mentre il secondo definisce il formato del timestamp, in modo tale che si possa riconoscere quale parte corrisponde al giorno, al mese, all'anno, e così via. Con l'opzione `target` si indica il campo in cui sarà memorizzato il timestamp; in questo caso, si è deciso di aggiornare il valore del campo `@timestamp` associato all'evento.
- *useragent*: questo filtro, come *geoip*, è utilizzato per effettuare *Data Enrichment* di certe informazioni. Come si può ben capire dal nome, *useragent* consente di analizzare una stringa, che rappresentava lo useragent di una richiesta, e strutturarla in un formato predefinito. Il filtro aggiunge ulteriori informazioni, come la famiglia del browser utilizzata (Chrome, Firefox, Safari, etc.), il sistema operativo (inclusa la versione) dell'utente, ed il dispositivo utilizzato. Esso richiede una sola opzione obbligatoria, ovvero `source`, che indica quale campo contiene le informazioni relative allo useragent da analizzare.

La sezione di output

La sezione di output indica il sistema in cui inviare i dati trasformati nella fase precedente. In questo caso, adottando completamente lo stack della famiglia *Elastic*, si è deciso di utilizzare il plugin di output *elasticsearch* in modo tale da memorizzare i log nell'omonimo database. Chiaramente, se si prevede di utilizzare *Kibana* nelle fasi successive, utilizzare il plugin *elasticsearch* è la scelta ideale. Nel plugin *elasticsearch* sono state specificate due opzioni. L'opzione `hosts` specifica indirizzo e porta dell'istanza di *Elasticsearch* a cui inviare i dati (*Elastic stack* è installato in locale, quindi l'indirizzo è *localhost* e la porta è quella di default, ovvero la 9200), mentre `index` specifica l'indice da associare agli eventi. L'opzione `index` merita un'attenzione particolare; in questo caso si è deciso di utilizzare un indice settimanale; ciò significa che tutti gli eventi (ovvero i documenti nel database) che, in base al valore del timestamp, appartengono ad una certa settimana saranno raggruppati nello stesso indice; tuttavia, si può anche creare un unico indice che raggruppi

tutti gli eventi, oppure creare indici annuali, mensili, settimanali e così via. Come già anticipato, `index` permette di creare nomi di indici generati dinamicamente; in questo caso, la struttura dell'indice è `logstash-nginx_YYYY-ww`, in cui l'anno e la settimana sono calcolati in base al timestamp dell'evento.

4.3.2 Kibana

Dopo aver caricato i dati in *Elasticsearch*, questi ultimi possono essere visualizzati in maniera agevole sfruttando le funzionalità di *Kibana*. I due dataset, uno utilizzato nella fase di *training* e l'altro per il *testing*, sono stati caricati adottando lo stesso file di configurazione di *Logstash*, con la differenza che, nel primo la struttura dell'indice è `logstash-nginx_YYYY-ww`, mentre, nel secondo, è `newlog_YYYY-ww`.

La Figura 4.1 mostra il pannello *Discover* di *Kibana*, in cui sono i presenti i documenti associati all'indice `logstash-nginx`; la Figura 4.2, invece, mostra i documenti associati all'indice `newlog`.

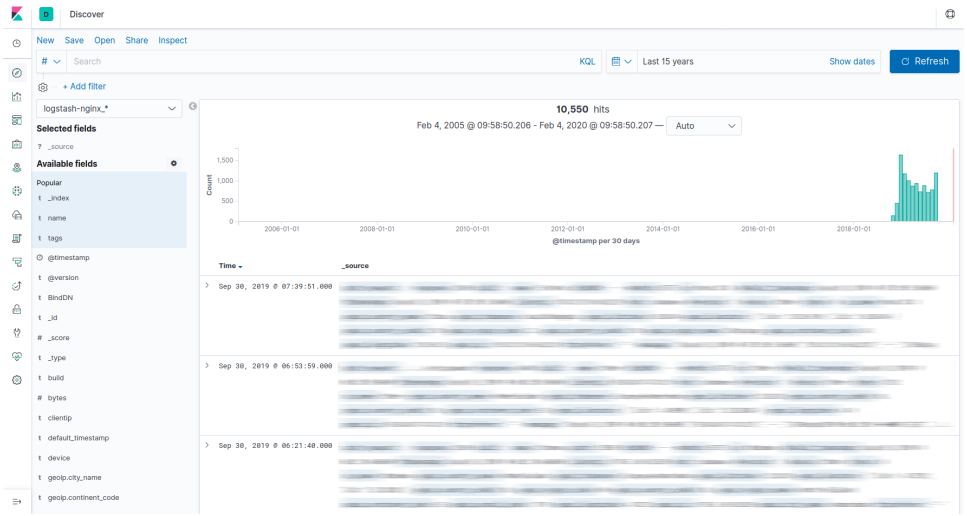


Figura 4.1: Documenti associati all'indice `logstash-nginx` in *Discover*

Kibana, tramite un'interfaccia Web, fornisce delle funzionalità molto utili per esplorare i dati, a partire dal pannello *Discover*, in cui è possibile visualizzare la serie temporale dei documenti, fino ad arrivare ai tantissimi grafici che possono essere utilizzati per rappresentare, in maniera migliore, i dati. I grafici e gli strumenti di visualizzazione sono molto comodi ed efficaci; tuttavia, lo strumento principale utilizzato in questa fase è la *Console*, presente nel pannello *DevTools*. La *Console* consente, tramite query testuali, di eseguire analisi molto spinte e complesse; essa permette di effettuare ricerche, aggiungere o rimuovere documenti associati ad indici, e molte altre operazioni. Durante questa fase, ovviamente, le query riguardano ricerche sui documenti; le tipiche operazioni sono: ordinamento, selezione di un sottoinsieme di campi, filtro dei dati in base a determinate condizioni (sia temporali sia

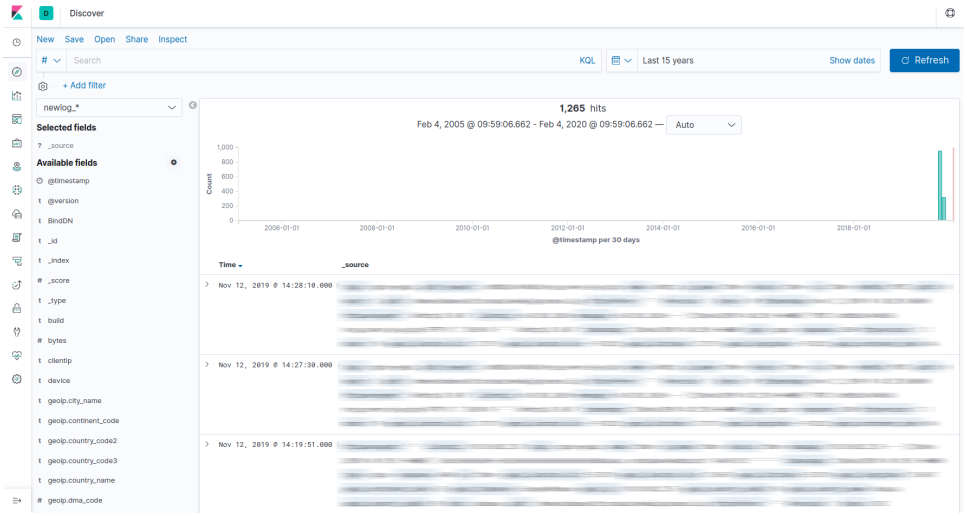


Figura 4.2: Documenti associati all'indice `newlog` in *Discover*

basate su valori), aggregazione dei dati, combinazione di filtraggio e aggregazioni, e molto altro ancora. In questo caso, tipiche query di ricerca effettuate sono: indirizzi IP più frequenti, selezione delle richieste anomale che provengono dall'Italia con *response code* compreso tra 200 e 299, pagine più richieste, richieste effettuate da uno specifico IP ordinate in maniera decrescente rispetto al timestamp, e così via.

Nel Listato 4.2 viene mostrato il codice relativo ad una query effettuata nella *Console*.

```
GET logstash-nginx_*/_search
{
  "sort": [{"@timestamp": {"order": "desc"}}],
  "_source": ["geoip.country_name", "geoip.continent_code", "os", "method", "path_request", "bytes", "response", "
        clientip", "name", "device", "httpreferer", "timestamp"],

  "query": {
     "bool": {
        "filter":    [
             {
               "term":{
                 "label":"anomaly"
               }
             },

             {
               "term":{
                 "geoip.country_name.keyword":"Italy"
               }
             },

             {
               "range": {
                 "response": {
                   "gte": 200,
                   "lt": 300
                 }
               }
             }

          ]
         }
      },

  "aggs": {
    "path": {
      "terms": {
```

```
37          "field": "path_request.keyword",
38          "size": 10
39             }
40          }
41       }
42    }
```

Listato 4.2: Codice di una query di ricerca effettuata tramite la *Console*

Come si può notare, una query è una richiesta HTTP in cui il body definisce i criteri della ricerca. La query mostrata nel Listato 4.2 restituisce i documenti, ovvero i log, ordinati in senso decrescente rispetto a `@timestamp`, le cui richieste sono etichettate come anomale, che provengono dall'Italia e il cui *response code* è compreso tra 200 e 299. In aggiunta, per ciò che riguardava i documenti selezionati, si richiede, tramite un'aggregazione, le 10 pagine Web più richieste. Come si può immaginare, la complessità di questo tipo di query non è banale; tuttavia, utilizzando la sintassi delle query *DSL* di *Elasticsearch*, è possibile definire analisi di questo tipo. In riferimento al Listato 4.2, valgono le seguenti considerazioni:

- La sintassi per eseguire ricerche di documenti è: `GET /<index>/_search`; si tratta, quindi, di una richiesta HTTP con metodo GET. Il parametro `<index>`, che si riferisce all'indice da utilizzare nella ricerca, supporta la notazione *wildcard*; ciò significa che, con `logstash-ngnix-*`, stiamo considerando tutti gli indici dei documenti che corrispondono a quel pattern.
- Il body della richiesta ne definisce i criteri; in particolare:
  - con il parametro `sort` indichiamo che i documenti restituiti devono essere ordinati; in questo caso, l'ordinamento deve avvenire in maniera decrescente rispetto al campo `@timestamp`.
  - con il parametro `_source` specifichiamo quali campi del documento JSON intendiamo visualizzare nei risultati.
  - il parametro `query` definisce il criterio della query; al suo interno, sono definiti tre filtri: il primo specifica che il valore del campo "`label`" deve essere pari ad "`anomaly`", il secondo specifica che siamo interessati alle sole richieste che provengono dall'Italia, mentre il terzo indica che si devono considerare solo le richieste il cui *response code* è compreso, grazie al parametro `range` tra 200 e 300, con l'intervallo chiuso a sinistra e aperto a destra.
  - il parametro `aggs` indica che intendiamo effettuare un'aggregazione. In tal caso, l'aggregazione considererà solo i risultati filtrati dalle condizioni precedenti. Per ciò che ci riguarda, si è deciso di aggregare i dati rispetto al campo `path_request`, con `size` pari a 10; ciò vuol dire che si richiedono le 10 pagine più frequenti.

Si può affermare che, tramite le funzionalità offerte da *Kibana*, ovvero il pannello *Discover*, i grafici e, sopratutto, la *Console*, è stato possibile visionare i dati in maniera migliore. Ciò ci ha consentito di verificare, in maniera semplice, che, effettivamente, ai dataset è già stata applicata una fase di *Data Cleaning*.

4.4 Approccio Proposto

Dopo aver capito quali dati si hanno a disposizione, e prima di procedere con le fasi successive del processo di analisi, è bene definire in maniera chiara e precisa qual è l'approccio scelto per raggiungere l'obiettivo.

L'obiettivo è definire un profilo del comportamento standard, in modo tale che, se un nuovo comportamento si discosta da quello standard, allora sarà considerato come anomalo. In un'applicazione Web, il comportamento di un utente è caratterizzato dall'insieme di azioni che egli compie all'interno di una sessione. In questo contesto, una sessione contiene tutte le attività svolte da un certo utente in un determinato intervallo temporale. L'idea, quindi, è quella di considerare le sessioni degli utenti, definire il profilo standard tenendo conto delle caratteristiche delle sessioni identificate come normali e, di conseguenza, quando le caratteristiche di una nuova sessione sono molte diverse da quelle considerate "normali", la nuova sessione sarà identificata come anomalia. Come detto più volte in precedenza, i dati a disposizione sono le singole richieste degli utenti e, quindi, è necessario ricostruire la sessione di un utente, che ne definisce il suo comportamento.

Un approccio alternativo, che non si è scelto di utilizzare, poteva essere quello di basare l'analisi sul singolo log; tuttavia si è preferito considerare la sessione di un utente per le seguenti ragioni:

- Un Server Web, potenzialmente, è in grado di ricevere migliaia di richieste al secondo; quindi sarebbe stato molto oneroso gestire il meccanismo di analisi in tempi accettabili.
- Considerare la sessione permette di avere una visione d'insieme del comportamento di un utente, e ciò potrebbe rivelare dei comportamenti anomali. Ad esempio, considerando un'analisi basata sul singolo log, un tentativo di login fallito potrebbe essere considerata una situazione "normale"; al contrario, considerando un'analisi basata sulla sessione, se la sessione di un utente è caratterizzata da centinaia di tentativi di login falliti, allora si potrebbe supporre che qualcuno stia effettuando un attacco di *bruteforce*, e quindi è necessario segnalare l'anomalia.

La sessione di un utente è caratterizzata dalla scelta di una variabile libera, ovvero l'intervallo temporale da considerare. Molto spesso, si considera un intervallo temporale di 30 minuti, scelta condivisa anche da Google nella sua piattaforma di analisi Web Google Analytics. Ricostruire la sessione di utente, a partire da un file di log, non è difficile; tipicamente, essa può essere ricostruita grazie alle informazioni su indirizzo IP e user-agent. È vero che, in una richiesta HTTP, l'indirizzo IP e lo user-agent possono essere falsificati (*spoofing attack*); tuttavia, ciò non può essere evitato e, quindi, si suppone che le informazioni siano originali, consentendo, quindi, di identificare correttamente la sessione.

La sessione utente, pertanto, è identificata univocamente da un indirizzo IP e da uno user-agent e contiene tutto il traffico generato in un intervallo temporale di 30 minuti.

Dell'approccio proposto rimane da definire quali sono le caratteristiche con cui è possibile descrivere una sessione, al fine di individuare anomalie; tuttavia, questa parte verrà illustrata nelle fase di *Feature Extraction*.

4.5 Algoritmi Utilizzati

Gli algoritmi di Intelligenza Artificiale rappresentano la soluzione ideale per individuare anomalie nei dati e, in questo caso, per rilevare le sessioni anormali degli utenti. I dataset utilizzati in questa analisi sono etichettati; tale caratteristica consente di sfruttare molte tipologie di algoritmi. Si è deciso di utilizzare tre algoritmi di *Machine Learning* che si basano su tre approcci differenti. Gli algoritmi scelti sono: *SVM*, o *Support Vector Machine*, che è un algoritmo di apprendimento supervisionato; *Isolation Forest* che è, al contrario, un algoritmo di apprendimento non supervisionato; *One-Class SVM* che adotta un approccio ibrido rispetto ai due precedenti.

4.5.1 SVM

Support Vector Machine è un algoritmo di *Machine Learning* supervisionato, tipicamente utilizzato per la classificazione. In questo algoritmo, ogni dato è rappresentato in uno spazio N-dimensionale (dove N è il numero di feature), in cui il valore di ogni feature rappresenta una coordinata. In un problema di classificazione a due classi, come, ad esempio, classe "normale" e classe "anomala", l'obiettivo di questo algoritmo è quello di trovare l'iperpiano che separa le due classi nella maniera migliore, come mostrato nella Figura 4.3.

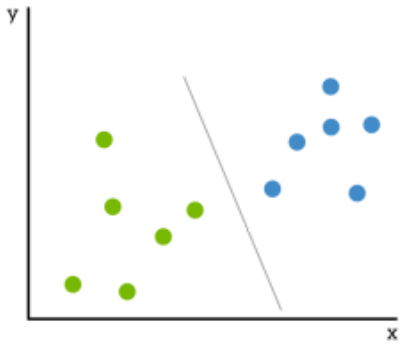


Figura 4.3: Retta che separa gli elementi delle due classi in un problema bidimensionale

Come si può notare nella Figura 4.3, esistono infinite linee in grado di separare i punti verdi da quelli blu. Il miglior iperpiano è quello che:

- Separa gli elementi delle due classi;

- Massimizza il margine, ovvero la distanza tra i *support vector*. I *support vector* sono i punti delle due classi che sono più vicini all'iperpiano. Massimizzare il margine è fondamentale perchè più la distanza tra i *support vector* è maggiore e più siamo sicuri di non commettere errori. I punti vicini al bordo sono quelli con il più alto grado di incertezza e, quindi, è bene massimizzare tale distanza.

  I concetti appena espressi sono mostrati nella Figura 4.4.

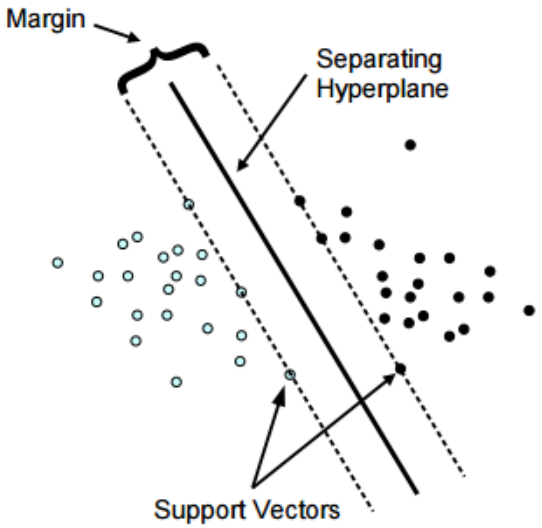


Figura 4.4: Iperpiano, margine e *support vector* in un problema bidimensionale

In uno spazio a 2 dimensioni, l'iperpiano è rappresentato da una retta; in uno spazio a 3 dimensioni, l'iperpiano è rappresentato da un piano a due dimensioni, e così via.

In generale, determinare l'iperpiano "ottimo" non è così semplice come mostrato negli esempi precedenti; ciò è dovuto al fatto che, nel mondo reale, la maggior parte dei problemi non è linearmente separabile. Ciò vuol dire che, molto spesso, non è possibile determinare un iperpiano in grado di separare perfettamente gli elementi di una classe dall'altra. In uno spazio a due dimensioni, un problema non linearmente separabile indica che non esiste una retta in grado di separare perfettamente gli elementi di una classe dall'altra. La differenza tra le due tipologie di problemi è mostrata nella Figura 4.5.

Uno dei punti di forza di *SVM* è che può trattare problemi in cui i dati non sono linearmente separabili. Per trattare questi tipi di problemi è necessario introdurre due concetti: il *Soft-Margin* e i *Kernel Trick*.

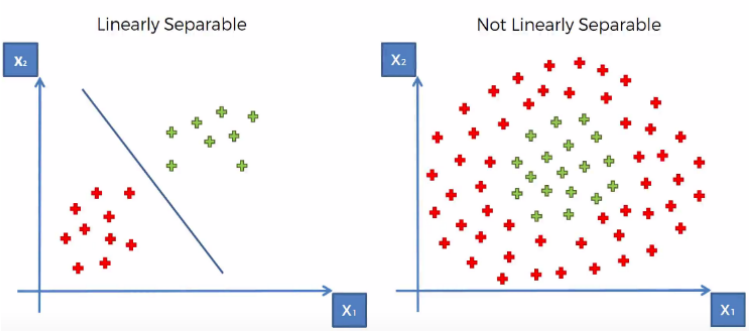


Figura 4.5: Separabilità in un problema bidimensionale

Il *Soft-Margin* è un'evoluzione del bordo decisionale, che può essere utilizzato quando i dati non sono linearmente separabili; si tratta di un iperpiano che tenta di separare gli elementi delle due classi, tollerando uno o più elementi misclassificati. Il *Soft-Margin* utilizza un parametro, detto variabile di *Slack*, che consente di considerare i punti misclassificati o rumorosi che si trovano vicino al bordo decisionale; il concetto di "vicinanza" dipende dal valore di questa variabile. Il *Soft-Margin* tenta di bilanciare il *trade-off* tra la minimizzazione degli elementi misclassificati e l'individuazione dell'iperpiano che massimizza il margine. La Figura 4.6 mostra un esempio di *Soft-Margin*.

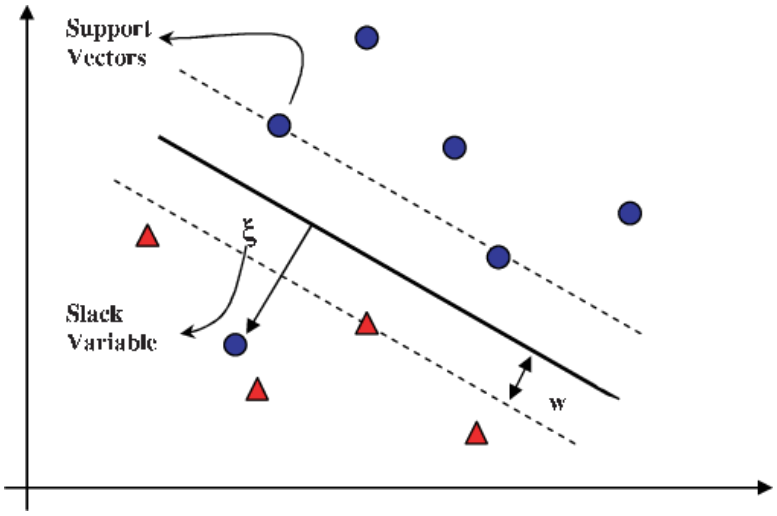


Figura 4.6: Esempio di *Soft-Margin*

I *Kernel Trick* sono delle funzioni che permettono di definire dei bordi decisionali

non lineari. In genere, nei problemi reali, i dati non sono linearmente separabili o leggermente non separabili, come visto nel *Soft-Margin*. Senza scendere nel dettaglio, dato un problema in cui i dati non sono linearmente separabili, un *Kernel Trick* è una funzione che consente di trasformare i dati a N-dimensioni in uno spazio a dimensioni maggiori, in cui i dati sono linearmente separabili e, quindi, in cui è possibile trovare una soluzione. Dallo spazio a più dimensioni si utilizza, ancora una volta, la funzione *Kernel* per trasformare la soluzione ed adattarla allo spazio dimensionale originale.

La Figura 4.7 mostra il meccanismo di *Kernel Trick*.

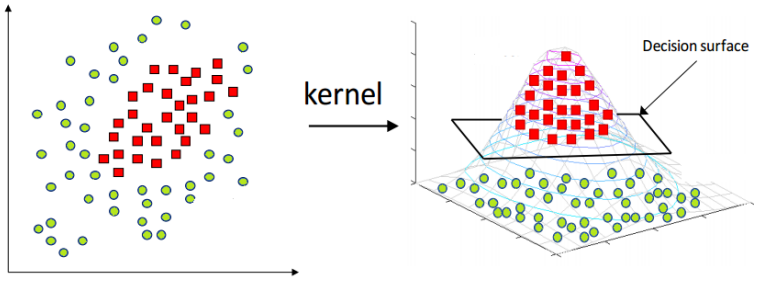


Figura 4.7: Meccanismo di *Kernel Trick*

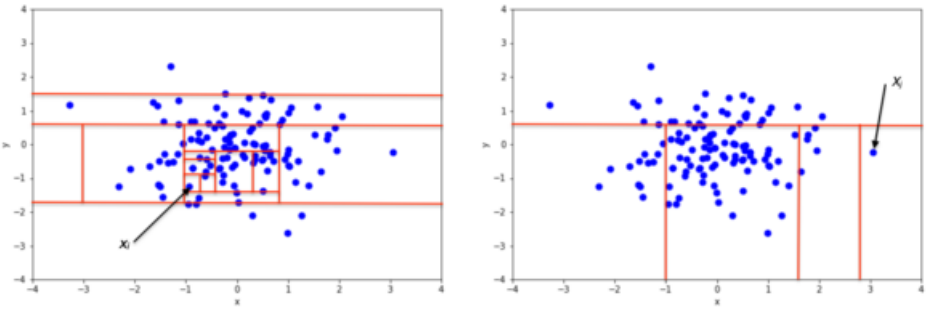
Nella presente tesi, come si può immaginare, l'algoritmo *SVM* è stato utilizzato per fare classificazione binaria, ovvero distinguere le sessioni "normali" da quelle "anomale".

4.5.2 Isolation Forest

Isolation Forest è un algoritmo non supervisionato progettato, nativamente, per rilevare anomalie. È un algoritmo sviluppato recentemente, nel 2008, e proposto da Fei Tony Liu, Kai Ming Ting e Zhi-Hua Zhou [22]. Gli autori suggeriscono che i punti anomali hanno delle caratteristiche precise: sono pochi, cioè rappresentano la minoranza dei dati, e sono diversi, ovvero i valori degli attributi sono molto differenti rispetto ai valori dei dati normali.

Dato che le anomalie sono poche e hanno valori molto differenti, esse sono facilmente isolabili rispetto ai dati tipici. Considerato un dataset, l'algoritmo costruisce un insieme di alberi decisionali, chiamati *Isolation Trees*; e le anomalie sono quei punti che, mediamente, sono individuati con il percorso più breve.

L'algoritmo, come detto, si basa sulla considerazione che, in un dataset, i dati anomali possono essere facilmente separati, cioè isolati, dalle istanze normali. L'*Isolation Forest* genera ricorsivamente delle partizioni sui campioni selezionando, in maniera casuale, un attributo, e poi, ancora in maniera casuale, un valore di *split*, compreso tra il valore massimo e quello minimo consentito per quell'attributo. La Figura 4.8 mostra il numero partizioni necessarie per isolare un punto normale da un'anomalia.



(a) Isolamento di un dato normale (b) Isolamento di un'anomalia

Figura 4.8: Meccanismo di isolamento dell'*Isolation Forest*

Come si può notare dalla Figura 4.8, le anomalie, per essere individuate, richiedono un numero inferiore di partizioni rispetto ai dati normali. La generazione ricorsiva di partizioni può essere rappresentata tramite una struttura ad albero, chiamata *Isolation Tree*. Il numero di partizioni richieste per isolare un punto può essere interpretato come la lunghezza del percorso, a partire dalla radice dell'albero, necessaria per raggiungere il nodo terminale.

Un *Isolation Tree* è un albero con le seguenti proprietà:

- ogni nodo dell'albero, può essere un nodo terminale, senza nessun figlio, oppure un nodo interno con una condizione di test ed esattamente due nodi figli;
- un nodo di test è costituito da un attributo e da un valore, la cui condizione determina l'attraversamento su uno dei due nodi figli.

Per concludere, le principali caratteristiche dell'*Isolation Forest* sono:

- È stato progettato nativamente per individuare anomalie; quindi, è ottimizzato, di default, per questo tipo di attività.
- Ha ottime performance e bassa complessità: la struttura dati fondamentale utilizzata in questo algoritmo è l'albero. L'albero è una struttura dati molto efficiente e l'isolamento di un punto viene effettuato tramite degli *split* basati sui valori di determinati attributi; tale caratteristica fa sì che la complessità computazionale dell'algoritmo sia bassa e, che si possa operare con dataset ad alta dimensionalità.
- Può operare anche con sole istanze normali dei dati; in altre parole l'algoritmo funziona bene anche se il dataset non contiene alcun dato anomalo. Il rilevamento di un'anomalia su cui si basa l'*Isolation Forest*, infatti, prende in considerazione esclusivamente la lunghezza media del percorso necessaria per isolarla.

Come si può immaginare, *Isolation Forest* è stato utilizzato nel nostro caso per individuare le sessioni anomale.

4.5.3 One-Class SVM

Il *One-Class SVM* può essere considerato come un caso particolare di *SVM*, in quanto si basa un approccio diverso. Esso rientra tra gli algoritmi di *One-Class Classification*.

One-Class Classification è un caso particolare di classificazione in cui l'algoritmo tenta di identificare, tra tutti gli elementi, quelli che appartengono ad una specifica classe; l'algoritmo, quindi, apprende da un dataset contenente solo gli oggetti di una certa classe. Algoritmi di *One-Class Classification* sono tipicamente utilizzati quando non si ha a disposizione una quantità sufficiente di elementi di ogni classe, oppure quando si conoscono perfettamente le caratteristiche dei membri di una classe e li si vogliono distinguere dal resto degli elementi. Nell'ambito dell'obiettivo della presente tesi, la visione proposta dagli algoritmi di *One-Class Classification* ha perfettamente senso; l'idea è quella di addestrare l'algoritmo sulle sessioni considerate "normali" in modo tale che tutto ciò che si discosta dal profilo standard sarà segnalato come "anomalo".

Il *One-Class SVM* è un algoritmo di *One-Class Classification* che si basa sull'*SVM*. Esso utilizza un approccio ibrido rispetto ai due algoritmi illustrati nelle sottosezioni procedenti. L'algoritmo fu proposto per primo da Schölkopf [23] che ne illustrò il funzionamento. Il funzionamento interno di *One-Class SVM* è analogo a quanto visto nel classico *SVM*; tuttavia, ci sono delle differenze: in questo caso si cerca di individuare un iperpiano che massimizzi la distanza tra l'origine e i punti rappresentati nello spazio delle feature. In fase di classificazione, ciò si traduce in una funzione binaria che restituisce "*+1*" per i dati normali e "*-1*" per gli *outlier*. Inoltre, rispetto al classico *SVM*, viene introdotto un nuovo parametro, indicato con ν, che stabilisce un limite superiore per la frazione di *outlier* presenti nei dati di *training*. Il parametro ν, quindi, fornisce una stima della percentuale di *outlier* che ci si aspetta di trovare nei dati di *training* definire correttamente il valore di questo parametro è fondamentale per una corretta classificazione.

Per concludere, *One-Class SVM* è stato utilizzato per individuare le sessioni "anomale", cioè quelle sessioni che sono molto differenti dagli esempi utilizzati nella fase di *training*, che, invece, rappresentano le sessioni "normali".

4.6 Data Extraction

Dopo aver individuato gli algoritmi di *Machine Learning* che si è scelto di utilizzare, nella fase di *Data Extraction* si predispongono i dati in modo tale che quest'ultimi possano essere utilizzati dai vari algoritmi in maniera corretta.

Gli algoritmi che si è deciso di utilizzare influenzano, anche, la scelta riguardo quali dati adottare per definire i modelli di Intelligenza Artificiale. Infatti, il modello prodotto tramite *SVM*, essendo un algoritmo di apprendimento automatico supervisionato, necessita dei dati di entrambi le classi, ovvero i log considerati "normali" insieme a quelli considerati "anomali". Al contrario, il *One-Class SVM*, durante la fase di *training*, utilizza i dati associati esclusivamente a istanze normali. L'*Isolation Forest* può lavorare sia con dati misti sia con le sole istanze normali; quindi si è deciso di utilizzare gli stessi dati utilizzati dall'*SVM*.

In questa analisi si è deciso di tentare di rilevare anomalie considerando esclusivamente il traffico corretto, ovvero valutando solo quelle richieste che hanno ricevuto un *response* HTTP di tipo *2xx*. Questa scelta è motivata dal fatto che le sessioni anomale composte da traffico lecito sono quelle che, molto probabilmente, nascondono un attacco o, sebbene non anomale, necessitano di maggiori indagini; ci si è, quindi, concentrati su quei log che hanno ricevuto un *response* di tipo *2xx*, escludendo gli altri.

I dati, come detto nelle sezioni precedenti, sono stati caricati in *Elasticsearch*. *Elasticsearch* offre un client ufficiale per Python, consentendone, quindi, l'integrazione. Si è, pertanto, codificata una funzione, mostrata nel Listato 4.3, che utilizza il client Python di *Elasticsearch* per definire una query che restituisce i dati memorizzati nel database.

```
1   def exportTrainData(file_output):
2       es = Elasticsearch([{'host': 'localhost', 'port': 9200}])
3       res = es.search(index='logstash-nginx_*',
4                       scroll='2m',
5                       body={
6                           "_source": constant.my_fieldnames,
7                           "query": {
8                               "bool": {
9                                   "filter": [{
10                                      "term": {
11                                        "label": "normal"
12                                      }
13                                    },
14                                    {
15                                      "range": {
16                                          "response": {
17                                              "gte": 200,
18                                              "lt": 300
19                                          }
20                                      }
21                                    }
22                                ]
23                            }
24                        }
25                      })
26      scroll_id = res['_scroll_id']
27      scroll_size = len(res['hits']['hits'])
28      output_file_csv = file_output
29
30      with open(output_file_csv, mode='w+') as output:
31          writer = csv.writer(output)
32          writer.writerow(constant.my_fieldnames)
33          while scroll_size > 0:
34              for result in res['hits']['hits']:
35                  data = []
36                  flat_result = helper.flattenDict(result['_source'], '', '')
37                  for field in constant.my_fieldnames:
38                      data.append(flat_result.get(field))
39                  writer.writerow(data)
40              res = es.scroll(scroll='2m', scroll_id=scroll_id)
41              scroll_id = res['_scroll_id']
42              scroll_size = len(res['hits']['hits'])
```

Listato 4.3: Codice della funzione che estrae i dati da *Elasticsearch*

Il Listato 4.3 si riferisce alla funzione definita per estrarre i dati che saranno, poi, utilizzati per l'algoritmo *One-Class SVM*. Come si può notare, un oggetto `Elasticsearch`, per essere istanziato, richiede *host* e *porta* in cui l'omonimo database è installato. Successivamente tramite la funzione `search`, si definisce la query; i parametri principali definiti nella funzione `search` sono `index`, che specifica l'indice o il pattern dell'indice coinvolto nella ricerca, e `body`, che ne definisce i criteri. Il valore del parametro `body` equivale al *body* di una query eseguita tramite la *Console* di *Kibana*; come si può notare dal codice, si è deciso di recuperare solo quei log etichettati come "normali" e il cui *response code* è compreso tra 200 e 299. Il resto

del codice permette di memorizzare i dati recuperati in un file CSV, per utilizzarli nelle fasi successive.

4.7 Session Extraction

L'obiettivo, in questa fase, è stato quello di ricostruire le sessioni degli utenti utilizzando i log recuperati nella fase precedente.

La sessione, infatti, definisce il comportamento di un utente e corrisponde al traffico generato in un certo intervallo di tempo. In questo contesto, si considera un intervallo temporale di 30 minuti e si suppone che un utente sia individuato univocamente dal suo indirizzo IP e dallo useragent.

La sessione è caratterizzata, quindi, da tutti i log generati da un utente, individuato univocamente grazie ad un indirizzo IP e uno useragent, in una finestra temporale di 30 minuti.

Tramite Pandas, una libreria Python molto utilizzata per la manipolazione dei dati, l'estrazione delle sessioni degli utenti si riduce ad una funzione composta da poche righe di codice, come mostrato nel Listato 4.4

```
1   def sessionExtraction(rawdata_filename, session_dir):
2       df = pd.read_csv(rawdata_filename, keep_default_na=False)
3       while not df.empty:
4           ip_target = df.iloc[0]['clientip']
5           ua_target = df.iloc[0]['httpuseragent']
6           format = '%d/%b/%Y:%H:%M:%S %z'
7           first_timestamp = pd.to_datetime(df.iloc[0]['timestamp'], format=format)
8           is_a_session_log = (df.clientip == ip_target) & (df.httpuseragent == ua_target) & (pd.to_datetime(df.
                    timestamp, format=format) - first_timestamp <= pd.Timedelta(30, unit='m'))
9           df_session = df[is_a_session_log]
10          df_session.to_csv(session_dir + 'session-' + str(last_timestamp) + '.csv')
11          df = df.drop(df[is_a_session_log].index)
```

Listato 4.4: Codice della funzione che ricostruisce le sessioni degli utenti

Come si può osservare in questo listato, il file CSV, che contiene i log grezzi, prodotto durante la fase di *Data Extraction*, è caricato in un *DataFrame*, grazie alle funzionalità offerte da *Pandas*. Successivamente, il meccanismo per estrarre le sessioni degli utenti è il seguente: fino a quando il *DataFrame* non è vuoto, si legge la prima riga (equivale ad un singolo log) e si memorizza il valore dell'indirizzo IP e dello useragent utilizzato; successivamente si definisce una condizione booleana che seleziona tutte le righe del *DataFrame* relative alle richieste effettuate in una finestra di 30 minuti con lo stesso IP e lo stesso useragent; le righe del *DataFrame* che individuano la sessione di un utente sono memorizzate in un nuovo file CSV e, poi, eliminate dal *DataFrame* originale.

4.8 Feature Extraction

Dell'approccio proposto rimane da definire quali sono le caratteristiche con cui è possibile descrivere una sessione. Insieme ad un esperto di dominio, durante questa fase, sono state definite le feature che caratterizzano una sessione. Gli algoritmi di *Machine Learning*, utilizzati nella fase successiva, sono stati implementati tramite

la libreria Python *Scikit-Learn* che, per sua natura, richiede che tutti i valori delle feature siano numerici; ciò significa che, in alcuni casi, è stato necessario codificare le informazioni testuali o categoriche in informazioni numeriche. È bene specificare che l'applicazione Web, a cui fanno riferimento i log presi in considerazione, non gestisce un meccanismo di autenticazione e, quindi, nei dati grezzi, il valore del campo `userid` è sempre mancante; inoltre, per tutti i log, anche i valori dei campi `identd` e `referer` (che specificano l'URL di provenienza) sono mancanti; pertanto, non si è potuto sfruttare tali informazioni.

Dopo un'attenta valutazione, la Tabella 4.1 mostra quali sono le feature individuate per caratterizzare una sessione. Come si può notare, la maggior parte di esse sono di natura statistica.

Feature Generali

Rientrano in questa categoria le feature *Richieste* e *Bytes*. La prima è pari al numero di richieste effettuate nella sessione, mentre la seconda è la somma dei valori del campo *bytes* di tutte le richieste.

Feature Temporali

Le feature calcolate utilizzando il campo *timestamp* delle richieste sono *durata_time*, *media_time*, *mediana_time*, *min_time*, *max_time* e *devstand_time*. La prima feature è calcolata come la differenza, in secondi, tra la prima e l'ultima richiesta, mentre le altre feature considerano il tempo intercorso tra una richiesta e la successiva.

Feature Basate Sul Metodo HTTP

Le feature *GET*, *POST*, *HEAD*, *TRACE* e *OTHER* sono calcolate considerando i metodi delle richieste di una sessione.

Feature Basate Sul *Response Code*

Le feature *n2xx* e *n3xx* sono calcolate considerando i *response code* delle richieste di una sessione.

Feature Basate Sulle Pagine Richieste

La maggior parte delle feature individuate si basano sulle pagine richieste in una sessione. Questo non è strano; infatti, un tipico "mezzo" attraverso il quale un attacco verso un' applicazione Web viene sferrata è proprio tramite la pagina richiesta. *SQL Injection*, *Cross Site Scripting (XSS)*, *Remote Code Execution* e *Shell Shock* sono esempi di attacchi Web che sfruttano particolari codifiche delle richieste HTTP. Le feature che si basano sulle pagine richieste risultano, quindi, di notevole importanza.

Nelle feature *avg_length*, *mediana_length*, *min_length*, *max_length* e *devstand_length* si considera la lunghezza, in termini di numero di caratteri, delle pagine richieste in una sessione; si è pensato di considerare queste feature in quanto, tipicamente,

Richieste	Totale numero di richieste HTTP
Bytes	Totale Byte scambiati
durata_time	Durata, in secondi, della sessione
media_time	Media, in secondi, del tempo intercorso tra richieste successive
mediana_time	Mediana, in secondi, del tempo intercorso tra richieste successive
min_time	Minimo, in secondi, del tempo intercorso tra richieste successive
max_time	Massimo, in secondi, del tempo intercorso tra richieste successive
devstand_time	Deviazione standard, in secondi, del tempo intercorso tra richieste successive
GET	Numero di richieste GET
POST	Numero di richieste POST
TRACE	Numero di richieste TRACE
HEAD	Numero di richieste HEAD
OTHER	Numero di richieste con altri metodi HTTP
n2xx	Numero di richieste con *response code* 2xx
n3xx	Numero di richieste con *response code* 3xx
avg_length	Media del numero di caratteri delle pagine richieste
mediana_length	Mediana del numero di caratteri delle pagine richieste
min_length	Minimo numero di caratteri delle pagine richieste
max_length	Massimo numero di caratteri delle pagine richieste
devstand_length	Deviazione standard del numero di caratteri delle pagine richieste
avg_argument	Media del numero di argomenti tra le richieste
min_argument	Minimo numero di argomenti tra le richieste
max_argument	Massimo numero di argomenti tra le richieste
avg_special	Media del numero di caratteri speciali delle pagine richieste
mediana_special	Mediana del numero di caratteri speciali delle pagine richieste
min_special	Minimo numero di caratteri speciali delle pagine richieste
max_special	Massimo numero di caratteri speciali delle pagine richieste
devstand_special	Deviazione standard del numero di caratteri speciali delle pagine richieste
avg_digits	Media del numero di cifre utilizzate tra le richieste
mediana_digits	Mediana del numero di cifre utilizzate tra le richieste
min_digits	Minimo numero di cifre utilizzate tra le richieste
max_digits	Massimo numero di cifre utilizzate tra le richieste
devstand_digits	Deviazione standard del numero di cifre utilizzate tra le richieste
num_chrome	Numero di richieste effettuate tramite Chrome
num_firefox	Numero di richieste effettuate tramite Firefox
num_safari	Numero di richieste effettuate tramite Safari
num_edge	Numero di richieste effettuate tramite Microsoft Edge o IE
num_opera	Numero di richieste effettuate tramite Opera
num_uc	Numero di richieste effettuate tramite UC Browser
num_others	Numero di richieste effettuate tramite altri browser
cont_*xx*	*One Hot Encoder*, su base continente, della nazione da cui proviene la richiesta
num_windows	Numero di richieste effettuate tramite Windows
num_apple	Numero di richieste effettuate tramite OSX e iOS
num_android	Numero di richieste effettuate tramite Android
num_linux	Numero di richieste effettuate tramite S.O. Linux *based*
num_os_others	Numero di richieste effettuate tramite altri S.O.
ip_reputation	Score che indica la reputazione dell'indirizzo IP

Tabella 4.1: Feature di una sessione Web

le richieste anomale possono utilizzare delle richieste molto strane, che si traducono in sequenze di caratteri molto lunghe.

Nelle feature *avg_argument*, *min_argument* e *max_argument* si considera il numero di argomenti utilizzati nelle richieste di una sessione.

Nelle feature *avg_special*, *mediana_special*, *min_special*, *max_special* e *devstand_special* si considera il numero di caratteri speciali utilizzati nelle pagine richieste di una sessione; si è pensato di considerare queste feature in quanto l'uso di caratteri speciali, in alcuni casi, può nascondere comportamenti sospetti.

Nelle feature *avg_digits*, *mediana_digits*, *min_digits*, *max_digits* e *devstand_digits*

si considera il numero di cifre utilizzate nelle pagine richieste di una sessione; si è pensato di considerare tali feature in quanto, in alcuni casi, alcuni caratteri, quali, ad esempio, lo spazio, l'apice singolo ed altri, sono codificati utilizzando delle cifre numeriche.

Feature Basate Su Browser

Le feature *num_chrome*, *num_firefox*, *num_safari*, *num_chrome*, *num_edge*, *num_oepra* e *num_others* sono calcolate considerando il browser utilizzato, individuato grazie alla stringa *useragent*.

Feature Spaziali

Nella sezione relativa a *Logstash*, sfruttando il filtro *geoip*, si erano già riusciti ad ottenere informazioni spaziali sulla base dell'indirizzo IP; tuttavia, *Scikit-learn* non è in grado di operare su feature testuali o categoriche; si richiede, quindi, che l'informazione spaziale sia, in qualche modo, trasformata in una feature numerica. Si è deciso di codificare l'informazione riguardo la nazione di provenienza della richiesta con un *One-Hot Encoder* rispetto al continente; ciò significa che si sono aggiunte tante variabili quanto sono i continenti e se, ad esempio, una sessione proviene dall'Italia, la corrispondente variabile, che rappresenta il continente "Europa", assumerà un valore pari a uno, mentre tutte le altre variabili avranno valore zero.

Feature Basate Su Sistema Operativo

Le feature *num_windows*, *num_apple*, *num_android*, *num_linux* e *num_os_others* sono calcolate considerando il sistema operativo utilizzato, individuato grazie alla stringa *useragent*.

4.8.1 IP Reputation e MISP

L'*IP Reputation* è una feature che merita una menzione particolare. Essa rappresenta un valore che indica la reputazione di un indirizzo IP. Un indirizzo IP potrebbe avere una cattiva reputazione quando vengono rilevate attività sospette, come, ad esempio, server di spam, o virus, che vengono inviati da quell'IP. In genere, per determinare la reputazione di un indirizzo IP si richiede una grande mole di informazioni; si è, quindi, deciso di utilizzare la piattaforma MISP.

MISP è una piattaforma software che facilita la condivisione e lo scambio di informazioni di sicurezza. Dopo l'installazione, MISP, nella configurazione di default, include un insieme di *feed* OSINT pubblici. Questi *feed* possono essere utilizzati come fonte di correlazione per gli eventi e gli attributi, senza necessità di importarli direttamente nel sistema. Il sistema di *feed* adottato da MISP consente una facile correlazione tra gli eventi, ma anche di confrontare in maniera veloce gli eventi provenienti da *feed* diversi. Sono inclusi, come detto, per default, moltissimi *feed*; tra questi citiamo: lista di IP in blacklist, lista di URL compromessi, lista di domini che contengono malware, lista di nodi TOR, lista di IP di spam, e così via. Tutti i *feed*

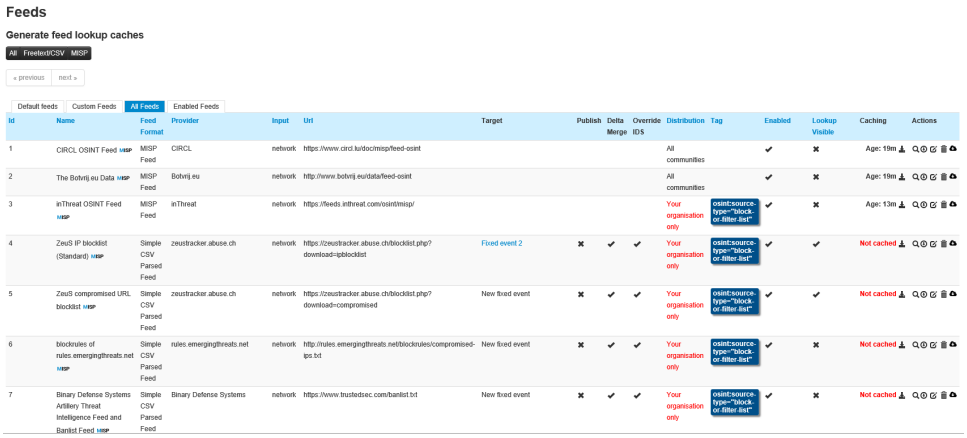


Figura 4.9: Parte dei *feed* disponibili in MISP

disponibili possono essere visualizzati nella pagina `https://www.misp-project.org/feeds/`.

La Figura 4.9 mostra una parte dei *feed* disponibili in MISP.

I *feed* di MISP sono stati utilizzati come fonte per determinare la reputazione dell'indirizzo IP della sessione considerata.

Le funzionalità di MISP possono essere estese progettando dei moduli software; quest'ultimi sono software autonomi che permettono di espandere MISP con altri servizi. I moduli sono scritti in Python 3 e possono essere utilizzati tramite semplici API. L'obiettivo è quello di espandere facilmente le funzionalità di MISP senza modificare i componenti *core*. Le API per utilizzare i moduli sono disponibili indipendentemente dall'installazione o dalla configurazione di MISP.

Si è quindi creato un modulo MISP in modo tale che, dato un indirizzo IP, lo score rappresentante la reputazione di quest'ultimo è calcolato come il numero di liste in cui compare. Il Listato 4.5 mostra il codice relativo al modulo MISP creato.

```
import json, requests
from pymisp import ExpandedPyMISP

misperrors = {'error': 'Error'}
mispattributes = {'input': ['ip'], 'output': ['text']}
moduleinfo = {'version': '1.0', 'author': 'Paolo Grossetti',
              'description': 'Simple expansion service to calculate a score for IP reputation',
              'module-type': ['expansion', 'hover']}

misp_url = "https://misp-instance:443"
misp_key = 'MY_MISP_KEY'

def ipScore(misp, param):
    num_blacklist = len(param)
    total_blacklist = misp.get_users_statistics().get('stats').get('event_count')
    return num_blacklist/total_blacklist

def handler(q=False):
    if q is False:
        return False
    try:
        request = json.loads(q)
    except Exception as e:
        misperrors["error"] = "Errore nel caricamento del json. Errore: "+str(e)
        return misperrors
    if request.get('ip'):
        iptarget = request['ip']
    else:
        return False
    try:
        misp = ExpandedPyMISP(misp_url, misp_key, ssl=False)
    except Exception as e:
        misperrors["error"] = "Errore nella creazione dell\' oggetto MISP. Errore: "+str(e)
        return misperrors
    result = misp.search(controller='attributes', value=iptarget, type_attribute=['ip-src', 'ip-dst'])
    if len(result.get('Attribute')) > 0:
        score = ipScore(misp, result.get('Attribute'))
    else:
        score = 0
    r = {'results': [{'types': mispattributes['output'],
                      'values':[str(score)]}]}
    print(r)
    return r

def introspection():
    return mispattributes

def version():
    return moduleinfo
```

Listato 4.5: Modulo MISP per l'IP Reputation

Un modulo MISP è composto, almeno, da tre funzioni:

- `introspection`: è una funzione che restituisce un dizionario degli attributi di input e output supportati dal modulo;
- `handler`: è una funzione che definisce il comportamento del modulo. Essa riceve in input un documento JSON e restituisce un dizionario con uno o più valori;
- `version`: è una funzione che restituisce un dizionario con la versione ed i relativi metadati, inclusa la potenziale configurazione richiesta dal modulo.

Come si può notare nel Listato 4.5, la funzione `handler` utilizza *ExpandedPyMISP*, un client Python che permette di interrogare, appunto in Python, la piattaforma MISP per calcolare lo score di un indirizzo IP; tale score sarà pari a zero se l'IP non compare in nessuna lista, altrimenti sarà pari al numero di liste in cui quell'IP compare.

Nella fase di *Data Extraction*, il calcolo della reputazione dell'indirizzo IP che ha generato una sessione si riduce ad una semplice richiesta HTTP al modulo descritto in precedenza, avendo l'accortezza di fornire il valore dell'IP in un documento JSON; il codice sorgente è mostrato nel Listato 4.6.

```
1   def ipReputation(param):
2       ip = param[0]
3       try:
4           buffer = BytesIO()
5           c = pycurl.Curl()
6           c.setopt(c.URL, 'http://127.0.0.1:6666/query')
7           c.setopt(pycurl.POST, 1)
8           header = ['Content-Type:application/json']
9           c.setopt(pycurl.HTTPHEADER, header)
10          body_as_dict = {"module": "myiprep", "ip": ip}
11          body_as_json_string = json.dumps(body_as_dict)
12          body_as_file_object = StringIO(body_as_json_string)
13          c.setopt(pycurl.READDATA, body_as_file_object)
14          c.setopt(pycurl.POSTFIELDSIZE, len(body_as_json_string))
15          c.setopt(c.WRITEDATA, buffer)
16          c.perform()
17          c.close()
18          response = buffer.getvalue()
19          decoded_response = response.decode('iso-8859-1')
20          score = int(json.loads(decoded_response).get('results')[0]['values'])
21          return score
22      except pycurl.error:
23          raise pycurl.error("Non sono riuscito a calcolare lo score di questo IP: "+str(ip)+"  Controlla le
                impostazioni di MISP")
```

Listato 4.6: Codice del metodo che effettua una richiesta al modulo MISP per il calcolo della reputazione dell'IP

4.9 Training del Modello

I dataset utilizzati nella fase di *Training* dei modelli di *Machine Learning* sono prodotti effettuando *feature extraction* delle singole sessioni, presenti, come detto nella fase di *Session Extraction*, in singoli file CSV, e poi aggregando i vari risultati in un unico file.

SVM, *Isolation Forest* e *One-Class SVM* sono algoritmi di *Machine Learning*; ciascuno di essi si basa su un approccio diverso: la differenza di approccio, di conseguenza, influenza il modo in cui i tre algoritmi devono apprendere dagli esempi di *training*.

SVM è un algoritmo di apprendimento supervisionato e ciò richiede che l'algoritmo, nella fase di *training*, impari a distinguere le caratteristiche delle sessioni etichettate come "normali" da quelle etichettate come "anomale". Chiaramente, è bene specificare che la sessione di un utente è considerata anomala se contiene del traffico etichettato come "anomalia". *SVM*, quindi, richiede, nella fase di *training*, dati che appartengono ad entrambi le classi. In un apprendimento supervisionato, molto spesso, si applicano tecniche di *cross-validation* che hanno l'obiettivo di valutare quanto il modello sia performante indipendentemente dai dati di *training*, ovvero viene valutata la proprietà di *generalizzazione*. Nel caso di *SVM*, prima della effettiva fase di *Training*, è stata applicata la *10-Fold Cross Validation*. L'algoritmo *SVM*, implementato da *Scikit-Learn*, richiede di specificare una serie di parametri. I più importanti sono:

- *C*: è il parametro di regolarizzazione; definisce il "costo" che si paga nel misclassificare i dati di *training*.
- *kernel*: specifica il tipo di kernel che deve essere usato dall'algoritmo. Esso può essere: "*linear*", "*poly*", "*rbf*" (valore di default), "*sigmoid*" o "*pre-computed*".
- *class_weight*: specifica il "peso" di ciascuna classe in base alla proporzione degli elementi. Per default, con il valore"*balanced*", il "peso" di ogni classe è calcolato automaticamente in base alle etichette dei dati.

Isolation Forest è un algoritmo di apprendimento automatico non supervisionato; ciò significa che le informazioni sulla classe di appartenenza delle sessioni non sono utilizzate. Nell'*Isolation Forest*, durante la fase di *training*, vengono costruiti gli *Isolation Tree*, ovvero l'insieme di alberi che saranno, poi, utilizzati per individuare le anomalie nel fase di *testing*. Queste ultime saranno quelle che, mediamente, sono isolate con un numero minore di *split*. L'algoritmo *Isolation Forest*, implementato da *Scikit-learn*, richiede la specifica di alcuni parametri:

- *n_estimators*: indica il numero di *Isolation Tree* da considerare;
- *contamination*: indica la quantità di contaminazione nel dataset, ovvero la proporzione di *outlier*. Questo parametro consente di definire una soglia che sarà utilizzata per la valutazione di uno *score* grezzo.

One-Class SVM è un algoritmo che utilizza un approccio ibrido rispetto agli algoritmi precedenti. Tale algoritmo risulta essere non supervisionato; tuttavia, richiede, nella fase di *training*, elementi che appartengono ad una sola classe. In questo caso, quindi, si è deciso di allenare l'algoritmo esclusivamente sulle sessioni "normali", in modo tale che, nella fase di *testing*, riesca a riconoscere le sessioni "anomale". Il *One-Class SVM* è, anch'esso, disponibile in *Scikit-Learn*, e i più importanti parametri di configurazione sono:

- *kernel*: specifica il tipo di kernel che deve essere usato dall'algoritmo (come nell'*SVM*);
- *degree*: indica il grado della funzione kernel polinomiale ("*poly*"). Questo parametro è ignorato negli altri tipi di kernel;
- ν: definisce il limite superiore della percentuale di *outlier* che ci si aspetta di trovare nei dati. Deve essere un valore compreso tra 0 e 1 (quest'ultimo escluso) e, per default, vale 0.5.

In generale, i valori delle feature dei dataset utilizzati appartengono a scale diverse; ad esempio, la feature *Bytes*, che rappresenta la quantità totale di byte scambiati in una sessione, può avere dei valori molto differenti rispetto a *Richieste*, che indica il numero totale di richieste effettuate; questa caratteristica del dataset può influenzare i risultati. Un'ulteriore differenza dei tre algoritmi scelti è che, l'*Isolation Forest* utilizza alberi decisionali, mentre *SVM* e *One-Class SVM* si basano sul calcolo di distanze nello spazio. Ciò significa che, nel primo caso l'algoritmo effettua degli *split* sulla base dei valori delle feature, e ciò fa sì che le scale diverse dei valori delle feature siano ininfluenti, mentre, nel secondo caso, i diversi range dei valori delle feature sono rilevanti e, quindi, prima della fase di *Training*, i dati devono essere standardizzati tramite un'apposita funzione offerta da *Scikit-Learn*. La standardizzazione consente di portare tutti i valori delle feature di un dataset ad una scala comune, senza falsificare le differenze nel range di valori; si è deciso di preferire la standardizzazione rispetto alla normalizzazione, in quanto quest'ultima richiede la conoscenza dei valori massimi e minimi di ogni feature, caratteristica che, per i valori delle feature individuate, non era possibile avere.

4.10 Testing e Valutazione

Il dataset utilizzato nella fase di *Testing*, come detto nelle sezioni precedenti, è composto da 1265 richieste effettuate in due settimane di attività. Anche per questo dataset sono state eseguite le fasi di *Data Exploration*, *Data Extraction*, *Session Extraction*, *Feature Extraction* e si generato un *testset* utilizzato per valutare i modelli di *Machine Learning* sviluppati nella fase precedente.

Per ogni algoritmo, i migliori risultati sono riportati nella Tabella 4.2.

Algoritmo	*Accuracy*	*Precision (Norm.)*	Recall *(Norm.)*	F1-Score *(Norm.)*	Precision *(Anom.)*	Recall *(Anom.)*	F1-Score *(Anom.)*
SVM con C=1.0, kernel=RBF e class_weight=balanced	0.99	0.98	1.00	0.99	1.00	1.00	1.00
Isolation Forest con n_estimators=500 e contamination=auto	0.90	0.33	0.34	0.34	0.95	0.94	0.94
One-Class SVM con kernel=RBF e ν=0.005	0.99	1.00	0.86	0.93	0.99	1.00	0.99

Tabella 4.2: Risultati ottenuti

Come si può notare da questa tabella tutti gli algoritmi utilizzati hanno una *Accuracy* molto elevata. Si può dire che l'*Isolation Forest* è l'algoritmo che si comporta nella maniera peggiore, in quanto, sebbene offra una accuratezza molto alta, non riesce a distinguere tra le sessioni normali e quelle anomale. Infatti, il modello tendenzialmente predice una classe come "anomala", sbagliando frequentemente sulla sessione normale. *SVM* e *One-Class SVM* hanno ottimi risultati sia per quanto riguarda l'*Accuracy* complessiva sia per le altre metriche.

Analizzando i risultati ottenuti e gli approcci utilizzati dai vari algoritmi si possono fare delle considerazioni finali: il modello di *Anomaly Detection*, che sfrutta l'algoritmo *SVM*, come si può vedere dai risultati, è il migliore, e ciò significa che esso riesce molto bene, grazie agli esempi forniti durante la fase di *training*, a distinguere le sessioni normali da quelle anomale. Tuttavia, gli attaccanti, continuamente, affinano le loro tecniche al fine di eludere i controlli; ciò significa che un attaccante potrebbe compiere un insieme di azioni, "inedite" e malevoli, tali per cui il modello non riesce a rilevare il comportamento sospetto; ciò significa che questo modello deve essere continuamente aggiornato.

Il modello di *Anomaly Detection* basato su *One-Class SVM*, invece, utilizza un approccio differente: l'algoritmo apprende le caratteristiche delle sessioni valutate come "normali" e segnalerà come "anomalia" tutte le sessioni che si discostano dal profilo del comportamento standard; ciò significa che, in futuro, se un attaccante dovesse ideare un nuovo attacco, la sessione anomala, seppur "inedita", sarà rilevata.

Alla luce di quanto detto sopra, il modello di *Anomaly Detection* basato su *One-Class SVM*, seppur offra delle performance leggermente inferiori rispetto a quello basato su *SVM*, è da preferire.

5

Modello di Anomaly Detection in un caso reale

Il presente capitolo illustra lo sviluppo di un modello di Anomaly Detection basato su algoritmi di Machine Learning e Deep Learning in uno scenario reale. Nella prima sezione viene definito lo scenario di riferimento, specificando le differenze riscontrate rispetto all'analisi del capitolo precedente. Nella seconda sezione si fornisce una descrizione del dataset considerato. Seguono, poi, una serie di sezioni in cui si descrivono, in modo dettagliato, tutte le fasi necessarie per lo sviluppo del modello. Il capitolo si conclude, infine, con la descrizione dei risultati ottenuti.

5.1 Scenario

Nel capitolo precedente si sono sviluppati dei modelli di *Anomaly Detection* al fine di individuare i comportamenti anomali degli utenti in un'applicazione Web.

I dati utilizzati nel Capitolo 4 provengono da un Server Web reale, ma sono da considerarsi sperimentali; il modesto quantitativo (complessivamente, circa 12000 richieste), l'elevata qualità (ai dati era già stata applicata una fase di *Data Cleaning*) e il vantaggio di avere dati etichettati sono delle facilitazioni che hanno accelerato il processo di sviluppo dei modelli.

Fatta questa importante premessa, diventa interessante capire come si comportano i modelli, prodotti nel capitolo precedente, in un caso reale.

L'obiettivo, quindi, è valutare e migliorare i modelli di *Anomaly Detection* sviluppati nel capitolo precedente in uno scenario d'utilizzo reale. L'obiettivo dei modelli di *Anomaly Detection* è immutato: definire un profilo standard in modo tale da segnalare come anomalia il comportamento di un utente che differisce dalla norma.

Anche in questo caso, i log puntuali, utilizzati nella fase di analisi, sono confidenziali e non possono essere mostrati.

5.2 Dataset

Il dataset utilizzato in questa analisi consiste in un insieme di log provenienti da un Server Web di produzione in *hosting* su *Plesk*. Nello specifico, il dataset è composto

da 465975 log, registrati in un periodo di tre mesi e mezzo di attività. *Plesk* offre servizi di *Web Hosting* e utilizza un Server Web Apache insieme ad un *reverse proxy* Nginx. I Server Web Apache adottano il formato testuale standard per i log, ovvero il *Common Log Format*. Ciò significa che, come nell'analisi precedente, per ogni log vengono memorizzate le seguenti informazioni: *indirizzo IP*, *identd*, *userid*, *timestamp*, *metodo*, *risorsa*, *protocollo*, *response code*, *content length*, *referer* e *user-agent*.

Tipicamente, in uno scenario reale, i dati sono grezzi e non etichettati, come il dataset preso in esame in questa analisi. Analizzare dati grezzi è, sicuramente, più difficile rispetto a trattare dataset creati *ad hoc* da utilizzare con algoritmi di Intelligenza Artificiale.

L'analisi di un dataset in un caso d'uso reale non è un problema banale; in questo caso si sono affrontate diverse criticità. Innanzitutto, la mole di dati a disposizione è notevole e richiede, per la gestione, strumenti adatti; poi, i dati sono completamente grezzi, senza alcuna attività di *Data Cleaning* iniziale; infine, come detto, i dati non sono etichettati.

La complessità del dataset, quindi, ha richiesto, nelle fasi successive, pesanti attività di *Data Pre-Processing*.

5.3 Data Exploration

Nel processo di analisi, con un dataset molto ampio, la fase di *Data Exploration* risulta essenziale. Anche in questo caso si sono utilizzati i software della famiglia *Elastic*: i dati grezzi sono stati caricati in *Elasticsearch* sfruttando una pipeline *Logstash*, per poi visualizzarli in *Kibana*. Per caricare i dati grezzi in *Elasticsearch* si è utilizzato lo stesso file di configurazione di *Logstash* che è stato mostrato nel capitolo precedente, ad eccezione della struttura dell'indice che, in questo caso, è `production_YYYY-ww` (struttura indice settimanale). Il codice sorgente della pipeline *Logstash* è riportato nel Listato 5.1.

```
1   input {
2     file {
3          path => "/path_to_logs/*"
4          type => "apache"
5          sincedb_path => "/dev/null"
6          start_position => "beginning"
7       }
8   }
9
10  filter {
11      grok {
12           match => { "message" => ["%{IP:clientip} %{GREEDYDATA:BindDN} \[%{GREEDYDATA:timestamp}\] \"%{WORD:method
                } %{GREEDYDATA:path_request} \HTTP/%{NUMBER:httpversion}\" %{NUMBER:response} %{NUMBER:bytes} %{
                QUOTEDSTRING:httpreferer} %{QUOTEDSTRING:httpuseragent}"]}
13          overwrite => [ "message" ]
14          }
15      mutate {
16          convert => ["response", "integer"]
17          convert => ["bytes", "integer"]
18          }
19      geoip {
20          source => "clientip"
21          target => "geoip"
22          }
23      date {
24          match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
25          target => "@timestamp"
26          }
27      useragent {
28          source => "httpuseragent"
```

```
29      }
30    }
31
32    output {
33      elasticsearch {
34        hosts => ["http://localhost:9200/"]
35        index => "production_%{+YYYY-ww}"
36      }
37    }
```

Listato 5.1: Pipeline *Logstash* utilizzata per caricare i dati grezzi in *Elasticsearch*

Uno dei principali vantaggi dei software della famiglia *Elastic* è che sono in grado di gestire grandi quantità di dati e, come ci si aspettava, *Elasticsearch*, *Logstash* e *Kibana* sono stati in grado di gestire senza problemi il dataset che, si ricorda, è composto da oltre 450000 log.

La Figura 5.1 mostra il pannello *Discover* di *Kibana* in cui sono presenti i documenti associati all'indice `production`.

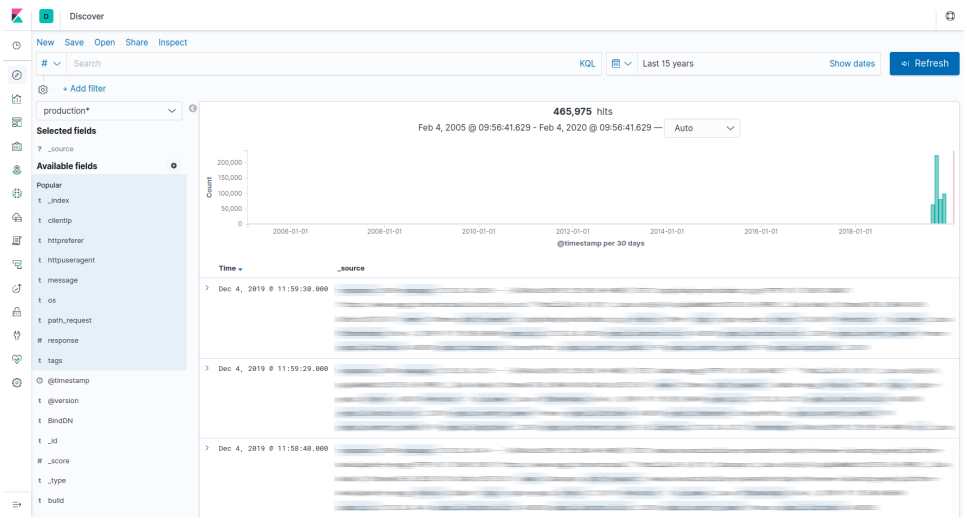


Figura 5.1: Documenti associati all'indice `production` in *Discover*

Come si può notare dal Listato 5.1, nella sezione relativa ai filtri, si è utilizzato il plugin *useragent*; esso consente di arricchire l'informazione relativa allo useragent di una richiesta HTTP estraendo, automaticamente, sistema operativo, browser utilizzato e altre informazioni. Il plugin *useragent* memorizza l'informazione sul dispositivo utilizzato nel campo *name* ed è in grado, nella maggior parte di casi, di individuare *spider* e *web crawler*. Tramite gli strumenti offerti da *Kibana* ci si è, quindi, concentrati sull'analisi del campo *name*; si è notato che il 48% delle richieste totali sono state effettuate da strumenti automatici che hanno l'obiettivo di scandagliare il Web, al fine di indicizzarlo. Come ci si poteva aspettare, i *Web Crawler* più frequenti presenti nel dataset sono Googlebot, bingbot, YandexBot, AhfresBot, SeznamBot, Applebot, e così via.

La successiva indagine ha preso come riferimento la risorsa richiesta, analizzando il relativo campo `path_request`. Una pagina Web è progettata utilizzando diversi

componenti: il documento vero e proprio, che contiene le informazioni della pagina, dei file JavaScript utilizzati per implementare script lato client, fogli di stile (CSS) e font dei caratteri utilizzati per la formattazione delle pagine, e file multimediali (immagini, video e suoni) usati per fornire una gradevole comunicazione. Ciò significa che la richiesta di una singola pagina Web si traduce, molto spesso, in richieste multiple volte a scaricare tutte le risorse necessarie. Il dataset utilizzato in tale analisi contiene dati grezzi e, quindi, in esso vengono registrate tutte le richieste, sia dei singoli documenti sia di tutte le risorse statiche necessarie. Utilizzando la *Console* di *Kibana* si è definita una query per individuare tutte le richieste relative a risorse statiche. Il Listato 5.2 mostra il codice sorgente della query DSL. Dall'approfondimento è emerso che il 25% delle richieste totali riguarda risorse statiche.

```
GET production_*/_search
{
  "_source": ["path_request", "httpreferer", "response"],
  "query": {
    "bool": {
      "must": [
                {
                  "query_string": {
                  "default_field": "path_request.keyword",
                  "query": "*.css OR *.js OR *.mp4 OR *.jpg OR *.jpeg OR *.png OR *.gif OR *.svg OR *.woff2 OR
                         *.woff OR *.ttf* OR *.ico OR *.webm OR *.ogg OR *map OR *.pdf",
                  "analyze_wildcard": true
                  }
                }
              ]
            }
        }
}
```

Listato 5.2: Codice sorgente della query DSL utilizzata per individuare risorse statiche

Un ultimo approfondimento ha reso la fase di *Data Exploration* fondamentale per il prosieguo dell'analisi. Ricordiamo che, tra i filtri utilizzati nella pipeline *Logstash*, il plugin *geoip* consente di localizzare geograficamente l'indirizzo IP da cui proviene una certa richiesta. Analizzando la nazione di provenienza dei log si è notato che tutte le richieste provenivano dalla Francia, e questo particolare è stato considerato alquanto strano, in quanto i log riguardano un sito Web italiano. Un'indagine approfondita ha portato alla luce che tutte le richieste erano effettuate da 8 indirizzi IP distinti, tutti localizzati in Francia. Dopo maggiori analisi si è scoperto che *Plesk*, il servizio che ospita il Server Web, utilizza Nginx come *reverse proxy* e, a causa di una misconfigurazione, nei log non è stato riportato l'indirizzo IP del mittente reale della richiesta, bensì quelli dei Server Web Nginx, localizzati in Francia. L'episodio descritto in precedenza consente di fare due considerazioni: la prima è che lavorare con dati reali non è banale e possono capitare dei problemi di questo tipo; la seconda è che un'adeguata fase di *Data Exploration* è essenziale.

La problematica riscontrata fa sì che, in sostanza, non si ha a disposizione l'informazione sull'indirizzo IP dell'utente; tuttavia si è deciso, ugualmente, di continuare il processo di analisi.

5.4 Approccio Proposto

L'obiettivo, in questa analisi, è valutare e migliorare l'approccio proposto nel capitolo precedente. L'idea, quindi, anche in questo caso, è considerare la sessione degli utenti e tentare di definire il profilo di comportamento corretto: la condotta di un utente, definita tramite le azioni che compie, che si differenzia dal profilo sarà considerata come anomala. Un'anomalia potrebbe rappresentare un attacco oppure un comportamento che, seppur innocuo, merita un approfondimento.

Una sessione corrisponde al traffico generato, in un certo intervallo di tempo, da un utente identificato univocamente dal suo indirizzo IP e da uno *useragent*. Nel dataset preso in esame l'informazione relativa all'indirizzo IP di un utente, per quanto detto nella sezione precedente, non è affidabile; quindi si è tentato di ricostruire la sessione utente considerando esclusivamente la stringa *useragent*; questa scelta, ovviamente, introduce un'approssimazione in quanto lo *useragent* potrebbe non essere sufficiente per identificare un utente e, di conseguenza, si potrebbero introdurre degli errori. Un'ulteriore conseguenza, come si vedrà nei capitoli successivi, è che si deve rinunciare alle feature che si basano sull'indirizzo IP.

Concludiamo questa sezione specificando che, anche per questa analisi, per le sessioni si è deciso di considerare, al massimo, a partire dalla prima richiesta effettuata da un utente, una finestra temporale di 30 minuti.

5.5 Algoritmi Utilizzati

Negli scenari reali, tra cui rientra il dataset in esame, molto spesso, non si hanno a disposizione dati etichettati. In questo caso, etichettare l'intero dataset, composto da oltre 450000 singole richieste, non è fattibile e si potrebbero commettere degli errori; un discorso simile può essere fatto giudicando le sessioni. L'impraticabilità di definire delle etichette, quindi, rende impossibile utilizzare algoritmi di apprendimento automatico supervisionato.

Focalizzandosi sulle sessioni, risulta difficile valutare quando un insieme di log sono anomali, mentre, al contrario, si potrebbe individuare un sottoinsieme delle sessioni e, a partire da quelle giudicate normali, definire un profilo del comportamento standard. L'idea, quindi, è quella di adottare una metodologia simile al *One-Class Classification*: è difficile definire in maniera chiara quando una sessione è anomala, quindi si considerano un insieme sufficienti di sessioni, valutate positivamente, che rappresentano il profilo standard corretto, e tutte le sessioni che si discostano da quest'ultimo saranno valutate come anomale. L'approccio appena descritto ha influenzato, quindi, la scelta degli algoritmi; in questa analisi si è deciso di adottare degli algoritmi di Intelligenza Artificiale semi-supervisionati e non supervisionati.

Gli algoritmi utilizzati per implementare la metodologia descritta sono: *Isolation Forest*, *One-Class SVM*, *Gaussian Mixture Models* e *Autoencoder*. L'ultimo algoritmo, ovvero l'*Autoencoder*, è una *Deep Artificial Neural Network*; l'elevata quantità di dati e la complessità del problema hanno spinto a valutare delle soluzioni basate su reti neurali.

5.5.1 Isolation Forest

L'*Isolation Forest* è un algoritmo di apprendimento automatico non supervisionato ed è stato già introdotto nel capitolo precedente. Si rimanda alla Sezione 4.5.2 per una descrizione dettagliata.

5.5.2 One-Class SVM

One-Class SVM è un algoritmo di apprendimento automatico che utilizza un approccio ibrido. È considerato un algoritmo non supervisionato; tuttavia richiede, durante la fase di addestramento, elementi di una specifica classe. Questo algoritmo è stato già descritto nella Sezione 4.5.3, a cui si fa riferimento.

5.5.3 Gaussian Mixture Models

Il *Gaussian Mixture Models*, o GMM, è un algoritmo di clustering probabilistico e rientra tra gli algoritmi di apprendimento automatico non supervisionato. Esso è un caso particolare di *mixture model*. Un *mixture model* è un modello probabilistico utilizzato per rappresentare la presenza di sotto-popolazioni all'interno di una popolazione più ampia. L'individuazione della sotto-popolazione non richiede l'utilizzo di dati d'esempio che la identifichino e si basa, quindi, su un approccio *unsupervised*. In maniera formale, possiamo dire che un *mixture model* corrisponde ad un mix di distribuzioni (*mixture distribution*) che, complessivamente, rappresenta la distribuzione di probabilità delle osservazioni nella popolazione complessiva. In alcuni casi, i dati che stiamo tentando di modellare sono molti complessi e, ad esempio, potrebbero essere multimodali; ciò significa che esistono numerose, ma differenti, regioni alcune delle quali hanno un'alta probabilità, mentre in altre la probabilità è bassa. In questo ambito, quindi, si potrebbero modellare i dati tramite mix di distribuzioni differenti, in cui ciascuna di esse ha una semplice forma parametrica; in questo contesto, le distribuzioni sono anche chiamate componenti. Si ipotizza, quindi, che ogni punto appartenga ad uno dei componenti e si tenta di inferire delle informazioni da ogni componente separatamente. Il *Gaussian Mixture Models* adotta un modello di tipo *mixture model* in cui le diverse componenti vengono modellate come distribuzioni gaussiane.

Il *Gaussian Mixture Models* viene spesso utilizzato nel clustering dei dati; uno dei principali vantaggi è che può essere impiegato per *hard-clustering* e *soft-clustering*. La differenza tra *hard-clustering* e *soft-clustering* è che, nel primo, ogni elemento è assegnato esattamente ad un cluster, mentre, nel secondo, un dato può appartenere a più di un cluster. Nella variante *hard-clustering*, *GMM* assegna un dato a quel cluster, identificato da una distribuzione gaussiana, che massimizza la probabilità a posteriori del componente. Definito un *GMM*, quindi, l'algoritmo assegna gli elementi ai cluster in modo tale da produrre la massima probabilità a posteriori; ogni punto è assegnato esattamente ad un cluster. In aggiunta, si può utilizzare *GMM* per eseguire un *clustering* più flessibile sui dati, ovvero di tipo *soft* o *fuzzy clustering*. In questa variante, l'algoritmo assegna all'elemento, per ogni cluster, uno score grezzo. Il valore dello score indica l'affinità dell'elemento a quel cluster. A differenza di un *hard-clustering*, il metodo è più flessibile si può assegnare un punto a più di

un cluster. In questo caso, lo score è rappresentato dalla probabilità a posteriori di ogni componente.

Come per gli altri algoritmi di clustering, *GMM* richiede di specificare, a priori, il numero di cluster da considerare; esso, invece, specifica il numero di componenti.

Un *Gaussian Mixture Models* è composto da K gaussiane, dove K indica il numero di cluster. Ciascuna gaussiana k è caratterizzata dai seguenti parametri:

- la media μ che definisce il centro;
- la deviazione standard σ che definisce la larghezza;
- la probabilità π che definisce il "peso" dei componenti, ovvero quanto grande o piccola debba esser la funzione gaussiana.

La Figura 5.2 mostra i parametri appena descritti.

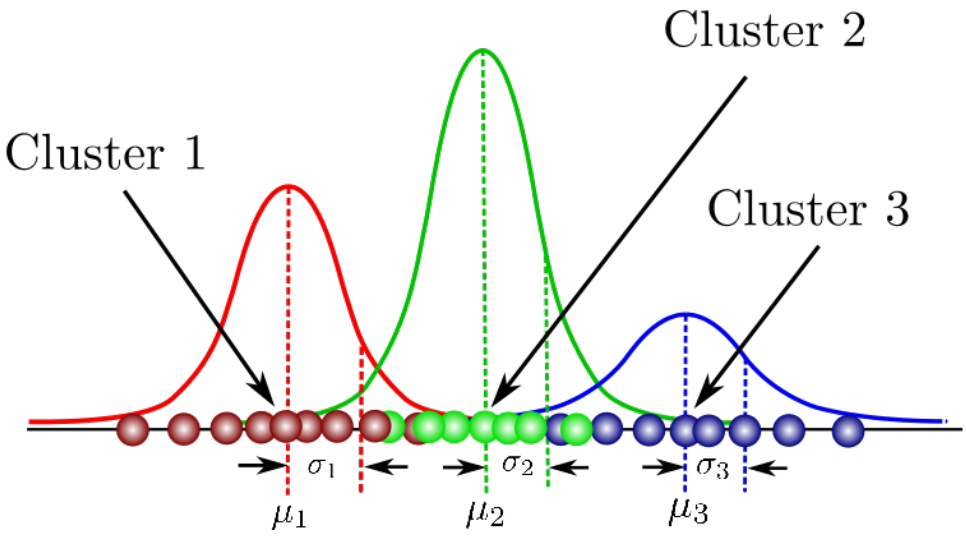


Figura 5.2: Esempio di *GMM* con 3 componenti

Come si può notare dalla Figura 5.2, ogni gaussiana descrive i dati contenuti in ciascuno dei tre cluster. Le probabilità di ciascuna gaussiana devono essere tali per cui deve essere soddisfatto il vincolo:

$$\sum_{k=1}^{K} \pi_k = 1$$

In generale, la funzione che descrive la distribuzione normale o gaussiana è la seguente:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Come si può notare, la distribuzione normale dipende dalla media μ e dalla varianza σ^2.

In termini dell'input x, la funzione scritta sopra è chiamata funzione densità di probabilità e fornisce la probabilità di osservare l'input x, data una certa distribuzione normale con i relativi parametri.

Nel *Gaussian Mixture Models* si considerano più distribuzione gaussiane e, quindi, la trattazione si complica. Dato un GMM con K componenti, la probabilità di osservare l'input x vale:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \sigma_k^2)$$

con

$$\sum_{k=1}^{K} \pi_k = 1$$

La trattazione si complica ulteriormente se, invece di considerare un singolo elemento di input x, si considera $X = \{x_1,..,x_n\}$, cioè un vettore composto da N elementi di input.

La prima equazione esprime che un certo elemento di input x è una combinazione lineare di K gaussiane. Ogni gaussiana è mediata dal termine π_k che rappresenta il "peso" corrispondente. La seconda equazione è un vincolo sui pesi delle gaussiane: la somma deve fare uno. In un *GMM*, quindi, la probabilità di osservare un dato dipende dal valore dei pesi e dei parametri di ciascuna gaussiana, ovvero media e varianza. Per individuare i valori di pesi, medie e varianze non si utilizza una formula diretta, bensì un algoritmo iterativo, detto algoritmo EM (*Expectation-Maximization*); quest'ultimo consente di determinare i parametri di modelli quando certe informazioni sono mancanti. L'algoritmo EM è composto da due fasi, ovvero:

- fase di *Expectation*: in questa fase, per ogni elemento di input, viene calcolata la probabilità a posteriori per ogni componente;
- fase di *Maximization*: vengono aggiornati i valori dei pesi, delle medie e delle varianze calcolate durante la fase precedente;

L'obiettivo dell'algorimto EM è quello di massimizzare la funzione densità di probabilità rispetto ai valori dei pesi, delle medie e delle varianze.

Concludiamo tale sezione specificando che, molto spesso, per semplificare i calcoli, la probabilità a posteriori non viene calcolata direttamente, ma si utilizza la *log-likelihood*, ovvero il logaritmo naturale della probabilità a posteriori.

5.5.4 Autoencoder

Un *Autoencoder* è una classe di reti neurali addestrata per tentare di copiare gli elementi di input nello strato di output. La Figura 5.3 mostra l'architettura tipica di un *Autoencoder*.

Gli *Autoencoder* sono *Artificial Neural Networks* in grado di apprendere efficienti rappresentazioni dei dati di input, chiamati *codings*, senza alcuna supervisione; si tratta, quindi, di un approccio completamente *unsupervised*. I *codings*, in genere, sono descritti con un numero inferiore di feature rispetto ai dati di input e, di conseguenza, utilizzare tale tipologia di reti neurali può essere utile per ridurre

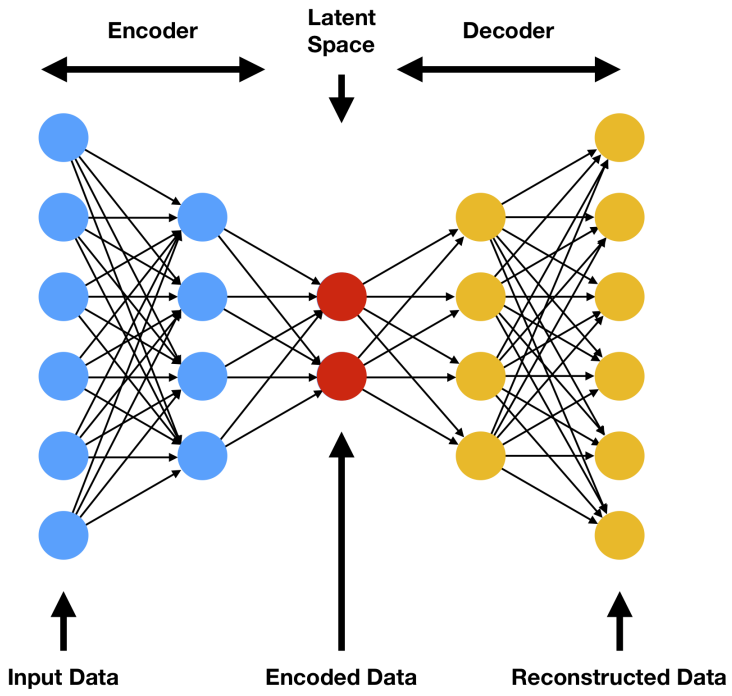


Figura 5.3: Esempio di architettura di un *Autoencoder*

la dimensionalità del problema. In maniera differente, *Autoencoder* sono utilizzati come potenti *feature detector*.

Un *Autoencoder* è in grado generare casualmente nuovi dati che sembrano molto simili a quelli di *training*; questo approccio prende il nome di modello generativo. Si potrebbe, ad esempio, addestrare la rete su un insieme di immagini di visi umani e sarebbe, così, in grado di generare nuovi volti.

In maniera sorprendente, l'obiettivo di un *Autoencoder* è concettualmente semplice: apprendere dagli esempi in modo tale da copiare gli elementi di *training* in output. L'approccio potrebbe sembrare banale; tuttavia, vincolare la rete a determinate condizioni può essere molto difficile. Per esempio, si possono limitare le dimensioni della rappresentazione interna o si possono aggiungere, in input, dati rumorosi in modo tale da allenare la rete a recuperare i dati originali; tali vincoli impediscono all'*Autoencoder* di copiare banalmente gli input direttamente sugli output, costringendolo ad apprendere modi efficienti di rappresentazione dei dati. Un *Autoencoder*, quindi, "osserva" i dati di input, li converte in una efficiente rappresentazione interna e restituisce in output un qualcosa che, si spera, sia molto simile ai dati iniziali, come mostrato nella Figura 5.4.

Un *Autoencoder* si compone sempre di due parti: un encoder, o *recognition network*, che converte gli input in una rappresentazione interna, e un decoder, o *generative network*, che converte la rappresentazione interna in output. Come si può vedere dalle figure, un *Autoencoder* ha, in genere, ha la stessa struttura di una rete

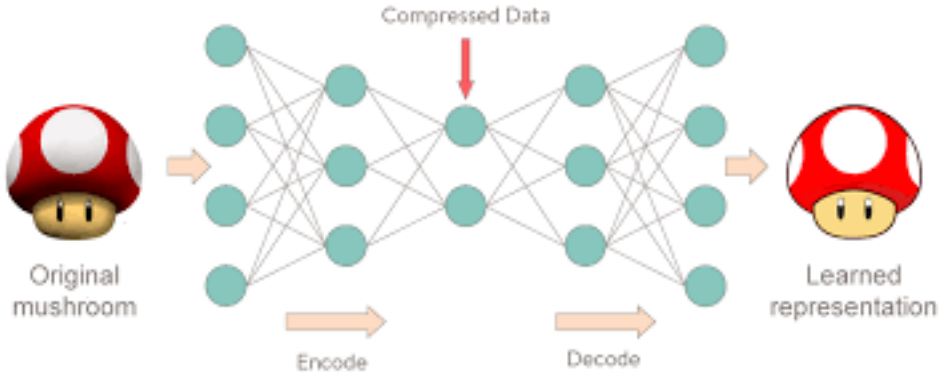


Figura 5.4: Applicazione di un *Autoencoder* nell'ambito della *Computer Vision*

Multi-Layer Perceptron (MLP), ad eccezione del numero di neuroni nello strato di output che deve essere uguale a quelli dello strato di input. I neuroni di uscita della rete sono chiamati, anche, neuroni di ricostruzione in quanto un *Autoencoder* tenta di ricostruire gli elementi di input e la funzione di costo contiene un errore di ricostruzione che penalizza il modello quando i dati ricostruiti differiscono da quelli di input.

Un *Autoencoder*, come già anticipato in precedenza, è una rete composta da uno strato di input, uno o più strati nascosti ed uno strato di output; con più di uno strato nascosto si entra nell'ambito delle *Deep Artificial Neural Networks*. Tra gli strati nascosti, quello più interno definisce la rappresentazione degli elementi di input con un numero di feature inferiore. La rappresentazione interna, tipicamente, ha una dimensionalità inferiore rispetto ai dati di input; in tal caso, si parla di *Autoencoder Undercomplete*. In un *Autoencoder Undercomplete* la rappresentazione interna utilizza, intenzionalmente, un numero inferiore di neuroni in modo tale che l'encoder apprenda una rappresentazione compressa degli input che, poi, esso dovrà ricostruire.

5.6 Data Extraction

La scelta dell'approccio utilizzato e, di conseguenza, dei vari algoritmi di Intelligenza Artificiale adottati, ha influenzato, anche in questo caso, la fase di estrazione dei dati da *Elasticsearch*. Come detto nella Sezione 5.4, si è deciso di adottare un approccio in cui i vari algoritmi sono allenati con un sottoinsieme di sessioni, giudicate normali, in modo tale da segnalare come anomalie le sessioni degli utenti che differiscono dal profilo standard. Questa scelta fa sì che tutti algoritmi scelti nella sezione precedente utilizzino lo stesso insieme di dati di *training*.

Grazie alla fase di *Data Exploration* si è dimostrato che il 25% delle richieste totali riguarda le risorse statiche, tipicamente utilizzate per strutturare pagine Web. Immagini, fogli di stile, script JavaScript, font ed altri file statici sono componenti di una pagina Web, ma che non si è deciso di considerare nella presente analisi. La richiesta di una specifica pagina Web, come accennato in precedenza, corrisponde a

più richieste, necessarie per recuperare le risorse; si è, quindi, deciso di non considerare i log relativi a richieste di questi tipi di file in quanto, per questa analisi, non aggiungono un significativo contributo informativo e contribuiscono ad aumentare il volume delle informazioni da processare.

Una differenza rispetto all'analisi illustrata nel capitolo precedente è che, in questa analisi, si è deciso di recuperare dal database *Elasticsearch* tutte le richieste il cui *response code* è compreso tra 200 e 499 (nell'analisi precedente solo quelle *2xx*). Questa scelta è motivata dal fatto che l'applicazione Web è progettata in maniera tale che, quando si richiede una pagina che non esiste, si viene reindirizzati su una pagina standard e, quindi, questo tipo di richiesta corrisponde ad una richiesta con *response 3xx*, seguita da una con *response 2xx*; tuttavia, si è deciso anche di considerare, nella query di estrazione, richieste con *response 4xx* al fine di individuare quei log in cui il meccanismo descritto non avviene.

Anche in questo caso, come nell'analisi del capitolo precedente, si è sfruttato il client Python *elasticsearch* che consente, tramite semplice API, di recuperare i dati presenti nel database.

Il Listato 5.3 mostra il codice relativo alla funzione Python codificata per estrarre i dati.

```
def exportTrainData(file_output):
    es = Elasticsearch([{'host': 'localhost', 'port': 9200}])
    res = es.search(index='production_2019-3*',
                    scroll='2m',
                    body={
                        "sort": [{"@timestamp": {"order": "asc"}}],
                        "_source": constant.my_fieldnames,
                        "query": {
                            "bool": {
                              "must_not": [
                                {
                                  "query_string": {
                                  "default_field": "path_request.keyword",
                                    "query": "*.css OR *.js OR *.mp4 OR *.jpg OR *.jpeg OR *.png OR *.gif OR *.svg OR
                                              *.woff2 OR *.woff OR *.ttf OR *.ico OR *.webm OR *.ogg OR *map OR *.pdf",
                                  "analyze_wildcard": True
                                  }
                                }
                              ],
                              "filter":
                              [
                                {
                                    "range": {
                                        "response": {
                                            "gte": 200,
                                            "lt": 500
                                        }
                                    }
                                }
                              ]
                            }
                          }
                    })
    scroll_id = res['_scroll_id']
    scroll_size = len(res['hits']['hits'])
    output_file_csv = file_output
    with open(output_file_csv, mode='w+') as output:
        writer = csv.writer(output)
        writer.writerow(constant.my_fieldnames)
        while scroll_size > 0:
            for result in res['hits']['hits']:
                data = []
                flat_result = flattenDict(result['_source'], '', '')
                for field in constant.my_fieldnames:
                    data.append(flat_result.get(field))
                writer.writerow(data)
            res = es.scroll(scroll='2m', scroll_id=scroll_id)
            scroll_id = res['_scroll_id']
            scroll_size = len(res['hits']['hits'])
```

Listato 5.3: Codice della funzione utilizzata per estrarre i dati da *Elasticsearch*

Come si può notare dal Listato 5.3, la query restituisce i documenti associati all'indice `production` e il parametro `query` della funzione `search` ne specifica i criteri: devono essere restituiti tutti i documenti, ovvero i log, che, per quanto riguarda il campo *path_request*, non contengono determinati tipi di file (individuati dalle relative estensioni) e il cui valore del campo *response* deve essere compreso tra 200 e 500, quest'ultimo non incluso.

Una menzione particolare la merita il parametro `scroll` della funzione `search`. Questo parametro specifica che si è utilizzato l'API *Scroll* di *Elasticsearch*: questo tipo di API è altamente consigliata quando una query restituisce più di 10000 documenti; in *Elasticsearch* uno dei parametri di configurazione è `index.max_result_window`, che specifica il numero massimo di documenti da restituire; per default, tale parametro è impostato a 10000; aumentare il suo valore potrebbe causare problemi all'istanza *Elasticsearch* e, quindi, non è consigliata la modifica; quando una query restituisce più di 10000 documenti, è caldamente consigliato utilizzare l'API *Scroll*, che consente di restituire blocchi di dati, fino a quando tutti i documenti coinvolti non sono stati restituiti.

Il resto del codice, mostrato nel Listato 5.3, consente di salvare i singoli log in un unico file CSV da utilizzare nelle fasi successive.

5.7 Session Extraction e Data Filtering

L'obiettivo della fase di *Session Extraction* è quello di ricostruire la sessione degli utenti a partire dai log grezzi. La sessione utente, intesa come traffico generato in un certo intervallo di tempo, ne definisce il comportamento. Come accennato nelle sezioni precedenti, in questa analisi, la difficoltà nasce dall'impossibilità di utilizzare l'informazione sull'indirizzo IP per l'identificazione di un utente. L'unica informazione a disposizione per individuare un utente è la stringa *useragent*. In una richiesta HTTP, *useragent* specifica dettagli sul sistema operativo, sul motore di rendering, sul browser e sulla piattaforma hardware utilizzata. Come si può ben pensare, le informazioni contenute nello *useragent* sono numerose e dettagliate e, quindi, con un certo grado di approssimazione, tale campo ci consente di identificare un utente. Nel concetto di sessione l'intervallo temporale scelto è, anche per questa analisi, 30 minuti.

Sebbene le informazioni presenti nello *useragent* siano piuttosto significative, potrebbe capitare di avere utenti diversi che, tramite la combinazione di dispositivo, browser e sistema operativo, siano identificati dalla stessa stringa, portando ad un'errata definizione del concetto di sessione; tale situazione, considerando una finestra temporale di 30 minuti, non è poi così remota. Senza poter sfruttare la preziosa informazione sull'indirizzo IP, si è deciso, quindi, di utilizzare un'euristica che irrigidisce il concetto di sessione, al fine evitare delle falsificazioni. L'idea, su cui si basa l'euristica, è quella di considerare ugualmente una finestra temporale di 30 minuti; tuttavia, si richiede che, tra tutte le richieste di cui si compone una sessione, il tempo intercorso tra una richiesta e la successiva non sia maggiore di 5 minuti. L'idea, quindi, è quella di evitare delle contaminazioni tra sessioni differenti, imponendo che le diverse richieste siano a distanza, al massimo, di 5 minuti, pena la creazione di una nuova "sessione".

Un'ultima considerazione può essere fatta riguardo il traffico generato da Web crawler che, si ricorda, corrisponde al 48% del traffico totale. Nella sezione precedente avevamo filtrato il traffico da richieste a file statici, per poi salvare i log grezzi in un file CSV; a partire da quest'ultimo, si è deciso, inoltre, si escludere il traffico, e quindi le sessioni, generate dagli spider Web. Questa decisione è motivata dal fatto che i Web crawler sono facilmente identificabili analizzando lo *useragent*, il corrispondente traffico non è generato da persone reali e, sopratutto, il modello di *Anomaly Detection* che si intende realizzare non mira ad individuare tali sistemi automatici. Per individuare le sessioni generate dagli spider Web si è, innanzitutto, ricostruita la relativa sessione a partire dai log grezzi e, successivamente, si è codificata una funzione Python che, analizzando le informazioni dello *useragent* decide che, se l' 80% delle richieste della sessioni si riconducono a noti sistemi di Web crawler, allora quella sessione va scartata e, quindi, non considerata nelle fasi successive, in quanto si ritiene che, con molta probabilità, sia stata generata da sistemi automatici. Quanto appena detto è codificato nella funzione Python definita nel Listato 5.4.

La generazione delle sessioni utilizzando l'euristica e la considerazione fatta sul traffico generato da Web crawler sono aspetti condivisi sia per i dati utilizzati nella fase di *training* sia per quelli adottati nella fase di *testing*.

```
1   def isBotSessions(dataframe):
2       name = dataframe.loc[:, 'name']
3       device = dataframe.loc[:, 'device']
4       if (len(device[device.str.lower().str.contains('spider')]) or
5           len(name[name.str.lower().str.contains('bot')]) or len(
6                   name[name.str.lower().str.contains('crawler')]) or len(
7                   name[name.str.lower().str.contains('crawl')]) or len(
8                   name[name.str.lower().str.contains('scrapy')]) or len(
9                   name[name.str.lower().str.contains('qwantify')]) or len(
10                  name[name.str.lower().str.contains('datanyze')]))/(len(dataframe.index)) > 0.8:
11          return True
12      else:
13          return False
```

Listato 5.4: Codice della funzione utilizzata per filtrare le sessioni generate da noti Web crawler

Tramite Pandas, la complessità di estrarre le sessioni con le specifiche condizioni definite in precedenza si riduce ad una funzione composta da poche righe di codice, come mostrato nel Listato 5.5.

```
1   def sessionExtraction(rawdata_filename, session_dir):
2       df = pd.read_csv(rawdata_filename, keep_default_na=False)
3       format = '%d/%b/%Y:%H:%M:%S %z'
4       df['timestamp'] = pd.to_datetime(df['timestamp'], format=format)
5       df['timestamp'] = pd.to_datetime(df['timestamp'], utc=True)
6       while not df.empty:
7           first_timestamp = df.iloc[0]['timestamp']
8           useragent_target = df.iloc[0]['httpuseragent']
9           df_window_session = df[(df.iloc[:]['timestamp'] - first_timestamp <= pd.Timedelta(30, unit='m'))]
10          session_user = (df_window_session.httpuseragent.values == useragent_target)
11          df_session = df_window_session[session_user]
12          are_requests_between_60s = (df_session.iloc[:]['timestamp'] - df_session.iloc[:]['timestamp'].shift()).
                fillna(pd.Timedelta(seconds=0)) <= pd.Timedelta(300, unit='s')
13          if len(are_requests_between_60s[(are_requests_between_60s.values == False)]) > 0:
14              min_index_false = are_requests_between_60s[(are_requests_between_60s.values == False)].idxmin()
15              df_session = df_session.loc[:min_index_false - 1]
16          if not isBotSessions(df_session):
17              df_session.to_csv(session_dir + 'session~' + str(first_timestamp) + '.csv')
18          df = df.drop(df_session.index)
```

Listato 5.5: Codice della funzione che ricostruisce le sessioni degli utenti

Come nell'analisi del capitolo precedente, ogni sessione individuata viene memorizzata in un apposito file CSV da utilizzare nelle successive fasi.

5.7.1 Data Filtering

L'attività di *Data Filtering* è un'attività esclusiva per le sessioni che comporranno il training set; essa è necessaria in quanto l'approccio proposto, basato su *One-Class Classification*, richiede che i dati di *training* rispecchino il più possibile il comportamento corretto che, quindi, ne definisce il profilo standard. In questa fase si sono apprese informazioni da un esperto di dominio, con l'obiettivo di filtrare ulteriormente le sessioni individuate nella sezione precedente. Ad esempio, l'applicazione Web è sviluppata utilizzando un certo framework PHP, tale per cui, nelle richieste, determinati tipi di file non possono comparire. Pertanto, si è definito un insieme di euristiche con l'obiettivo di filtrare ulteriormente le sessioni che saranno utilizzate per definire il profilo di comportamento corretto.

5.8 Feature Extraction

Nella fase di *Feature Extraction* si sono individuate le caratteristiche da utilizzare per descrivere la sessione. Come nel caso precedente, le feature individuate sono di natura statistica. Rispetto all'analisi svolta nel capitolo precedente, questa fase ha subito delle modifiche importanti. Per quanto detto nelle sezioni precedenti, l'impossibilità di utilizzare l'indirizzo IP fa sì che le feature che si basano su questa informazione siano inutilizzabili. Escludendo le informazioni geografiche, non poter sfruttare l'*IP Reputation* strutturato tramite MISP rappresenta, senza dubbio, una grossa perdita. In questa fase, inoltre, alcune feature, utilizzate in precedenza, sono state soppiantate in quanto non si sono ritenute utili in questa analisi, mentre altre, inedite, sono state aggiunte. Nel dataset preso in esame l'informazione sul *referer*, che specifica la pagina di provenienza di una richiesta, è disponibile, e quindi si è deciso di utilizzarla. La Tabella 5.1 mostra quali sono le feature individuate per la presente analisi.

La feature *OPTIONS* è stata aggiunta per categorizzare, in maniera migliore, i metodi delle richieste.

La feature *n4xx* rappresenta una novità che si è deciso di introdurre. Per il calcolo di questa feature si è considerato il numero di richieste che riguardano la pagina vetrina, a cui si viene reindirizzati quando viene richiesta una pagina che non esiste.

Le feature *num_ref_unknown* e *num_ref_site* sono delle novità perchè considerano il *referer*. La prima feature è calcolata considerando il numero di richieste della sessione in cui il *referer* è mancante, mentre la seconda è definita come il numero di richieste che provengono da un'altra sezione dell'applicazione Web.

Le feature *num_ok_req* e *num_bad_req* sono state definite grazie all'aiuto di un esperto di dominio e sono calcolate considerando le risorse, in termini di pagine, richieste dall'utente. La prima feature si basa su quali tipi di richieste, che riguardano certi file, sono ammesse, mentre la seconda, al contrario, considera quali tipologie di file non sono ammessi.

Richieste	Totale numero di richieste HTTP
Bytes	Totale Bytes scambiati
media_time	Media, in secondi, del tempo intercorso tra richieste successive
mediana_time	Media, in secondi, del tempo intercorso tra richieste successive
min_time	Minimo, in secondi, del tempo intercorso tra richieste successive
max_time	Massimo, in secondi, del tempo intercorso tra richieste successive
devstand_time	Deviazione standard, in secondi, del tempo intercorso tra richieste successive
GET	Numero di richieste GET
POST	Numero di richieste POST
OPTIONS	Numero di richieste OPTIONS
HEAD	Numero di richieste HEAD
OTHER	Numero di richieste con altri metodi HTTP
n2xx	Numero di richieste con *response code* 2xx
n3xx	Numero di richieste con *response code* 3xx
n4xx	Numero di richieste con *response code* 4xx
avg_length	Media del numero di caratteri delle pagine richieste
mediana_length	Mediana del numero di caratteri delle pagine richieste
min_length	Minimo numero di caratteri delle pagine richieste
max_length	Massimo numero di caratteri delle pagine richieste
devstand_length	Deviazione standard del numero di caratteri delle pagine richieste
avg_argument	Media del numero di argomenti tra le richieste
min_argument	Minimo numero di argomenti tra le richieste
max_argument	Massimo numero di argomenti tra le richieste
avg_special	Media del numero di caratteri speciali delle pagine richieste
mediana_special	Mediana del numero di caratteri speciali delle pagine richieste
min_special	Minimo numero di caratteri speciali delle pagine richieste
max_special	Massimo numero di caratteri speciali delle pagine richieste
devstand_special	Deviazione standard del numero di caratteri speciali delle pagine richieste
avg_digits	Media del numero di cifre utilizzate tra le richieste
mediana_digits	Mediana del numero di cifre utilizzate tra le richieste
min_digits	Minimo numero di cifre utilizzate tra le richieste
max_digits	Massimo numero di cifre utilizzate tra le richieste
devstand_digits	Deviazione standard del numero di cifre utilizzate tra le richieste
num_ref_unknown	Numero di richieste effettuate con *referer* mancante
num_ref_site	Numero di richieste effettuate in cui il *referer* è l'app Web
num_ok_req	Numero di richieste lecite
num_bad_req	Numero di richieste di file non ammessi

Tabella 5.1: Feature individuate per la presente sessione. In verde sono riportate le nuove feature individuate

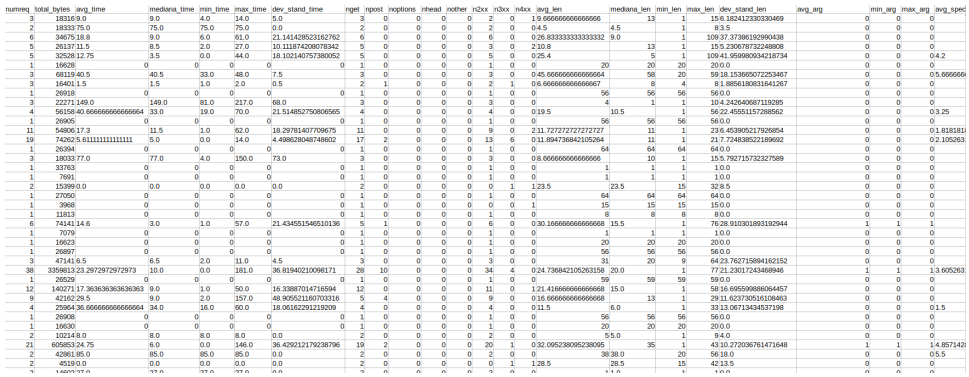
5.9 Training del Modello

Il *training-set* utilizzato in questa fase è ottenuto effettuando *feature extraction* delle singole sessioni, per poi aggregare i risultati in un unico file CSV. Una porzione del *training-set* prodotto è mostrato nella Figura 5.5.

Isolation Forest, *One-Class SVM*, *Gaussian Mixture Models* e *Autoencoder* sono algoritmi di Intelligenza Artificiale che si basano su approcci diversi, ma che, in questo caso, utilizzano lo stesso *training-set*. Dei dati complessivi, ne è stata utilizzata una porzione per il *training*, mentre il resto è stato riservato per il *testing*. I primi tre algoritmi sono stati implementati sfruttando le funzionalità offerte da *Scikit-Learn*, mentre, per l'*Autoencoder*, si è utilizzato il framework *Keras*.

Isolation Forest

L'*Isolation Forest* è un algoritmo di *Machine Learning* non supervisionato che si basa su alberi di ricerca ed è stato già introdotto, insieme ai relativi parametri,



numreq	total_bytes	avg_time	mediana_time	min_time	max_time	dev_stand_time	nget	npost	noptions	nhead	nother	n2xx	n3xx	n4xx	avg_len	mediana_len	min_len	max_len	dev_stand_len	avg_arg	min_arg	max_arg	avg_spec
3	18316	9.0	9.0	4.0	14.0	5.0	3	0	0	0	0	2	0	1	9.666666666666666	13	1	15	6.182412330330469		0	0	0
2	18333	75.0	75.0	75.0	75.0	0.0	2	0	0	0	0	2	0	0	4.5	4.5	1	8	3.5		0	0	0
6	34675	18.8	9.0	6.0	61.0	21.141428523162762	6	0	0	0	0	6	0	0	6.833333333333332	9.0		1	109.37	37361929990438	0	0	0
5	26137	11.5	8.5	2.0	27.0	10.111874208078342	5	0	0	0	0	3	0	0	2	10.8	13	1	15.5	23067873224808	0	0	0
5	32528	12.75	3.5	0.0	44.0	18.102140757380052	5	0	0	0	0	5	0	0	5	25.4	5	1	109.41	95998093421873	4	0	0	4.2
1	16628	0	0	0	0	0	1	0	0	0	0	1	0	0	0	20	20	20	20.0	0	0	0	
3	68119	40.5	40.5	33.0	48.0	7.5	3	0	0	0	0	3	0	0	5.666666666666664	58	20	59	18.153665072253467		0	0	5.666666
3	16401	1.5	1.5	1.0	2.0	0.5	2	1	0	0	0	2	1	0	6.666666666666667	8	4	8	1.885618083164126	7	0	0	0
1	26918	0	0	0	0	0	1	0	0	0	0	1	0	0	56	56	56	56	56.0	0	0	0	
3	22271	149.0	149.0	81.0	217.0	68.0	3	0	0	0	0	3	0	0	4	1	1	1	10.4	2426406871192985	0	0	0
4	56158	40.666666666666664	33.0	19.0	70.0	21.514852750806565	4	0	0	0	0	4	0	0	9.5	10.5	1	56	22.455511572885	62	0	0	3.25
1	26905	0	0	0	0	0	1	0	0	0	0	1	0	0	56	56	56	56	56.0	0	0	0	
11	54806	17.3	11.5	1.0	62.0	18.297814077096754	11	0	0	0	0	9	0	2	11.727272727272727	11	1	23	6.453905217926854		0	0	1.818181
19	74262	5.611111111111111	5.0	0.0	14.0	4.498628048748602	17	2	0	0	0	13	6	0	11.894736842105264	11	1	21	7.724838522189692		0	0	2.105263
1	26394	0	0	0	0	0	1	0	0	0	0	1	0	0	64	64	64	64	64.0	0	0	0	
3	18033	77.0	77.0	4.0	150.0	73.0	3	0	0	0	0	3	0	0	8.666666666666666	10	1	15	5.792715732327589		0	0	
1	33763	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	1	1	1.0	0	0	0	
1	7691	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	1	1	1.0	0	0	0	
2	15399	0.0	0.0	0.0	0.0	0.0	2	0	0	0	0	0	1	1	1.23.5	23.5		15	32.8.5	0	0	0	
1	27050	0	0	0	0	0	1	0	0	0	0	1	0	0	64	64	64	64	64.0.0	0	0	0	
1	3968	0	0	0	0	0	1	0	0	0	0	0	0	1	15	15	15	15	15.0.0	0	0	0	
1	11813	0	0	0	0	0	1	0	0	0	0	1	0	0	8	8	8	8	8.0.0	0	0	0	
6	74141	14.6	3.0	1.0	57.0	21.434551546510136	5	1	0	0	0	6	0	0	30.166666666666668	15.5	1	76.28	91030189319229	44	1	1	1
1	7079	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	1	1	1.0.0	0	0	0	
1	19623	0	0	0	0	0	1	0	0	0	0	1	0	0	20	20	20	20	20.0.0	0	0	0	
1	26897	0	0	0	0	0	1	0	0	0	0	1	0	0	56	56	56	56	56.0.0	0	0	0	
3	47141	6.5	6.5	2.0	11.0	4.5	3	0	0	0	0	3	0	0	31	20	9	64.23	76271589416215	2	0	0	0
38	335981	23.297297972973	10.0	0.0	181.0	36.81940210098171	28	10	0	0	0	34	4	0	24.736842105263158	20.0		1	77.21	23017243468946	1	1	1.3.605263
1	26529	0	0	0	0	0	1	0	0	0	0	1	0	0	59	59	59	59	59.0.0	0	0	0	
12	140271	17.363636363636363	9.0	1.0	50.0	16.3388701417169	12	0	0	0	0	11	0	0	1.21.416666666666668	15.0		1	58.16	69559988604457	0	0	0
9	42162	29.5	9.0	2.0	157.0	48.90552116070331	6	5	4	0	0	9	0	0	16.666666666666668	13	1	29	11.623730516109463		0	0	0
4	25964	36.666666666666664	34.0	16.0	60.0	18.06162291219209	4	0	0	0	0	4	0	0	11.5	6.0		1	33	13.067134345371	98	0	0	1.5
1	26908	0	0	0	0	0	1	0	0	0	0	1	0	0	56	56	56	56	56.0.0	0	0	0	
1	19630	0	0	0	0	0	1	0	0	0	0	1	0	0	20	20	20	20	20.0.0	0	0	0	
2	10214	8.0	8.0	8.0	8.0	0.0	2	0	0	0	0	2	0	0	5	5.0		1	9	4.0	0	0	0
21	605853	24.75	6.0	0.0	146.0	36.429212179238796	19	2	0	0	0	20	1	0	32.09523809523809	5	35	1	43	10.272036761471648	1	1	1.4.857142
2	42861	85.0	85.0	85.0	85.0	0.0	2	0	0	0	0	2	0	0	38	38.0		20	56	18.0	0	0	0.5.5
2	4519	0.0	0.0	0.0	0.0	0.0	2	0	0	0	0	0	1	1	1.28.5	28.5		15	42	13.5	0	0	0
2	14602	77.0	77.0	77.0	77.0	0.0	2	0	0	0	0	2	0	0	11.0		1	1	10.0		0	0	0

Figura 5.5: Porzione dei dati utilizzati nella fase di *training*

nel capitolo precedente. In questa analisi, l'*Isolation Forest*, nella fase di *training*, "costruisce" l'*Isolation Tree* utilizzando solo gli elementi di una sola classe, ovvero le sessioni "normali", a differenza dell'analisi del capitolo precedente in cui erano stati utilizzati elementi di entrambi le classi.

One-Class SVM

One-Class SVM è un algoritmo di *Machine Learning*, anch'esso, descritto nel capitolo precedente. Per questo tipo di algoritmo non sono state introdotte novità: *One-Class SVM*, nella fase di *training*, apprende esclusivamente le caratteristiche delle sessioni "normali".

Gaussian Mixture Models

Gaussian Mixture Models (nel seguito, *GMM*) è un algoritmo di *clustering* non supervisionato e probabilistico. *GMM* è un algoritmo di *clustering* tipicamente utilizzato per raggruppare elementi che appartengono a gruppi diversi. In esso, il *clustering* può essere di tipo *hard*, se ogni elemento è assegnato esattamente ad un *cluster*, o *soft*, se un elemento può appartenere a più di un *cluster*. In questa analisi, la peculiarità è che si intende utilizzare il *Gaussian Mixture Models* in versione *One-Classification*. Il *Gaussian Mixture Models* utilizza un insieme di distribuzioni gaussiane e ciascuna identifica un insieme di elementi; ogni gaussiana è caratterizzata da un "peso", dalla media e dalla varianza e sulla combinazione di questi valori si calcola la funzione densità di probabilità (*PDF*) che indica la probabilità di osservare un dato di input x, date le distribuzioni normali. Utilizzare *GMM* in ottica *One-Class* significa sfruttare, nella fase di *training*, solo elementi di una certa classe. Dopo aver definito il numero di componenti del *mixed model*, l'algoritmo determinerà i valori dei "pesi", delle medie e delle varianze in modo tale da ottenere, con i diversi componenti, i vari gruppi di dati omogenei; ciò fa sì che sia possibile determinare una *threshold*, basata sui valori della funzione densità di probabilità, in modo tale che, se, nella fase di *testing*, per un nuovo dato, il valore della funzione supera il limite imposto, vorrà dire che quel dato è anomalo. Nel *Gaussian Mixture Model* implementato da *Scikit-learn* i parametri fondamentali sono tre:

- *Components*: indica il numero di componenti, ovvero il numero di cluster, da considerare.
- *Covariance_type*: indica il tipo di varianza da usare. Deve essere uno tra "*full*", "*tied*", "*diag*" o "*spherical*". Questa possibilità è fondamentale in quanto ci consente di scegliere la forma dei *cluster*, cosa che, ad esempio, nell'algoritmo *K-Means*, basando su centroidi, non è possibile, in quanto i *cluster* hanno obbligatoriamente forma sferica.
- *Init_params*: indica il modo in cui inizializzare i valori dei "pesi", delle medie e delle varianze. Di default è impostata l'opzione *kmeans*, il che vuol ire che tali valori sono inizializzati sfruttando l'algoritmo *K-Means*.

Chiaramente, la principale difficoltà consiste nell'individuare il numero corretto di componenti: in genere, si utilizzano le metriche *AIC* e *BIC* (entrambe disponibili in *Scikit-Learn*) per la valutazione.

Autoencoder

Autoencoder è un algoritmo che si basa sull'utilizzo delle reti neurali. Per implementare questo tipo di rete si è utilizzato *Keras*, un framework che ha consentito di definire la rete in maniera semplice e veloce. Un *Autoencoder* tenta di ricostruire le informazioni basandosi sugli esempi di *training*. L'idea su cui si basa il modello *Anomaly Detection* che sfrutta questo algoritmo è molto semplice: nella fase *training* vengono forniti in input solo i dati relativi alle sessioni "normali', in modo tale che la rete impari a ricostruirle; sulla base dell'errore di ricostruzione degli elementi di input verrà definita una *threshold*, in modo tale che, nella fase di *testing*, un nuovo elemento sarà identificato come anomalo se l'errore di ricostruzione di quest'ultimo supera il limite definito.

Autoencoder è, a tutti gli effetti, una rete neurale e, quindi, i parametri in gioco sono molti. Si devono configurare numerosi parametri, i più importanti dei quali sono:

- il numero di strati nascosti, il numero di neuroni e la funzione di attivazione per ciascuno;
- il numero di epoche di *training*;
- il *learning rate*;
- la funzione di *loss*;
- la funzione di ottimizzazione;
- la dimensione di *batch*.

In genere, per una rete neurale si devono scegliere anche il numero di neuroni dello strato di input e di output, tuttavia, in questo caso, essendo l'*Autoencoder* una rete neurale utilizzata per ricostruire le informazioni di ingresso, il numero di neuroni dello strato di input e di output è uguale e pari al numero di feature individuate.

5.10 Testing e Valutazione

Gli algoritmi scelti condividono la caratteristica di un apprendimento *unsupervised*. Definire delle metriche oggettive per valutare un algoritmo di apprendimento automatico non supervisionato è difficile a causa, appunto, della mancanza di una etichetta. Per consentire una semplice valutazione dei risultati, quindi, si è deciso di codificare una funzione che, attraverso regole statiche, consente, in maniera approssimativa, ma comunque efficace, di assegnare un'etichetta per i soli dati di *testing*. La funziona verifica se tra le richieste della sessioni sono presenti *SQL Injection*, *Cross-Site Scripting*, *shell shock*, iniezione di codice PHP, un numero anomalo di richieste, e altri tipi attacchi Web. Il codice della funzione è mostrato nel Listato 5.6.

```
1   def isAnomaly(dataframe):
2       path_request = dataframe.loc[:, 'path_request']
3       param_response = dataframe.loc[:, 'response']
4       method = dataframe.loc[:, 'method']
5       if len(method[method.str.contains('HEADOPTIONSTRACE')]) > 0:
6           return True
7       elif (len(param_response[param_response.between(400, 499, inclusive=True)]) + len(path_request[path_request.str
                .lower().str.contains("/page-not-found")]))/dataframe.values.shape[0] > 0.6:
8           return True
9       elif dataframe.values.shape[0] > 100:
10          return True
11      elif is_broken_auth_and_session_management(path_request):
12          return True
13      elif is_insecure_direct_object_reference(path_request):
14          return True
15      elif is_local_file_inclusion(path_request):
16          return True
17      elif is_php_code_injection(path_request):
18          return True
19      elif is_web_shells_attack(path_request):
20          return True
21      elif is_shell_shock(path_request):
22          return True
23      elif is_xss(path_request):
24          return True
25      elif is_sqli(path_request):
26          return True
27      elif len(path_request[path_request.str.contains(constant.bad_request_filetypes, regex=True)]) > 0:
28          return True
29      else:
30          return False
```

Listato 5.6: Codice della funzione che etichetta approssimativamente le sessioni di *testing*

È bene specificare che la funzione presente nel Listato 5.6 è esclusivamente utilizzata per la valutazione dei risultati; questi ultimi si ottengono confrontando il risultato della funzione con quello restituito dal modello di *Anomaly Detection*.

Per ogni algoritmo, i migliori risultati sono riportati nella Tabella 5.2.

Isolation Forest

Tra gli algoritmi utilizzati, *Isolation Forest* ha le performance peggiori. L'*Accuracy* complessiva è accettabile; tuttavia, le metriche riguardo la classe "anomalia" non sono sufficienti. Il miglior modello di questo algoritmo è ottenuto con 256 *Isolation Tree* e il valore del parametro *contamination* impostato ad *"auto"*.

Come nell'analisi precedente, i risultati ottenuti con questo tipo di algoritmo non sono sufficienti e, quindi, si è deciso di approfondire questa aspetto. Da ulteriori

Algoritmo	Accuracy	Precision (Norm.)	Recall (Norm.)	F1-Score (Norm.)	Precision (Anom.)	Recall (Anom.)	F1-Score (Anom.)
Isolation Forest con iTree=256 e contam.=auto	0.79	0.85	0.91	0.88	0.19	0.12	0.15
One-Class SVM con kernel=RBF e ν=0.02	0.79	0.99	0.75	0.86	0.42	0.97	0.58
GMM con comp.=20, tipo cov.="full" e init_params="kmeans"	0.97	0.98	0.99	0.98	0.94	0.86	0.90
Autoencoder	0.95	0.99	0.95	0.97	0.78	0.94	0.85

Tabella 5.2: Risultati ottenuti

indagini, è emerso che le performance dell'*Isolation Forest*, implementato da *Scikit-Learn*, sono strettamente dipendenti dal parametro *contamination*; quest'ultimo, si ricorda, fornisce una stima della quantità di *outlier* presenti ed è una delle opzioni richieste nella fasi di *training*.

È difficile definire correttamente il valore del parametro: il *training-set*, infatti, contiene esclusivamente elementi della classe "normale" e, di conseguenza, il valore di *contamination* deve essere molto basso; tuttavia, nel *testing-set* la proporzione tra gli elementi di classe "normale" e "anomalia" non è la stessa, portando, quindi, a risultati scadenti. In generale, in uno scenario reale, le anomalie possono essere numerose, anche in numero maggiore rispetto alle situazioni normali; quindi la soluzione basata su *Isolation Forest* non è consigliata.

L'implementazione offerta da *Scikit-Learn* fa sì che *Isolation Forest* sia da preferire quando, approssimativamente, la percentuale di *outlier*, tra *training* e *testing* set, non cambia; l'algoritmo basa il proprio funzionamento su alberi di ricerca e, quindi, rimane il vantaggio che esso non necessita di ulteriori operazioni di standardizzazione o normalizzazione.

One-Class SVM

Il modello di *Anomaly Detection* basato su *One-Class SVM* riesce a migliorare i risultati ottenuti rispetto all'algoritmo precedente. Le migliori performance dell'algoritmo si sono ottenute con il kernel *RBF* e il valore di ν pari 0.02. Complessivamente i risultati sono buoni. Tuttavia, la *f-measures* della classe "anomalia" è insufficiente; ciò è dovuto ad una bassa *precision*. Un basso valore della *precision* della classe "anomalia" sta a significare che si hanno numerosi falsi positivi, ovvero molte sessioni "normali" sono, in realtà, rilevate come anomale.

Gaussian Mixture Models

Il modello *Anomaly Detection* basato su *Gaussian Mixture Models* di tipo *One-Class* presenta dei risultati molto buoni.

La migliore configurazione dell'algoritmo si ha per i seguenti valori dei parametri:

- *Components=20*, ovvero i dati di *training* sono stati raggruppati in 20 cluster;
- *Covariance_tipe="full"*;, questa opzione specifica che ogni componente, ovvero ogni cluster, può assumere una forma propria;
- *init_params="kmeans"*; ciò che i valori dei pesi, delle medie e delle varianze sono inizializzati utilizzando l'algoritmo *K-Means*.

Il concetto di *Anomaly Detection* che si è utilizzato per il presente algoritmo è il seguente: durante la fase di *training* vengono generate 20 distribuzioni gaussiane, ciascuna delle quali con un "peso", una media e una varianza. Dopo aver determinato tali parametri è possibile calcolare la funzione densità di probabilità in cui gli elementi di *training*, appartenenti alla classe "normale", assumeranno determinati valori. Si è, quindi, deciso di definire, sulla base dei dati di *training*, una *threshold* pari al 95esimo percentile dei valori della funzione densità di probabilità. Per i dati di *testing* viene calcolata la funzione densità di probabilità con i parametri individuati nella fase di *training*; quelli in cui il valore supera la *threshold* definita saranno rilevati come anomali.

Questa metodologia, come si può notare dai risultati, consente di avere *Accuracy* complessiva molto elevata, e anche le metriche di *f-measure* di ciascuna classe sono molto positive.

Autoencoder

Il modello di *Anomaly Detection* basato su *Autoencoder* rappresenta una soluzione con ottimi risultati. *Autoencoder*, basandosi su reti neurali, richiede di scegliere numerosi parametri. I migliori risultati sono stati ottenuti con la seguente configurazione:

- Rete *fully-connected feedforward* in cui lo strato di input e quello di output sono composti da un numero di neuroni pari al numero di feature individuate; si sono utilizzati sei strati nascosti in cui, rispettivamente, il numero di neuroni utilizzati è 32, 16, 8, 16 e 32. La funzione di attivazione di ogni strato della rete è la funzione lineare, ovvero corrisponde ad una combinazione degli input con i pesi sinaptici.
- Numero di epoche di training = 400;
- *Learning-rate* = 0.001;
- Funzione di *loss*: errore quadratico medio;
- Funzione di ottimizzazione: discesa stocastica del gradiente;
- Dimensione di *batch* = 128.

Anche per questo modello si è deciso di utilizzare una *threshold*: essa è basato sul 95esimo percentile dell'errore di ricostruzione degli elementi di *training*. Dato un elemento di *testing*, dopo averlo sottoposto alla rete, si calcola l'errore di ricostruzione e, se quest'ultimo supera il limite definito, allora sarà rilevato come anomalo. I risultati mostrati nella Tabella 5.2 confermano la bontà della metodologia appena descritta: l'*Accuracy* complessiva è elevata e, anche rispetto alle altre metriche, i risultati sono soddisfacenti.

6

Discussione in merito al lavoro svolto

Il capitolo corrente illustra l'analisi SWOT, una metodologia di pianificazione strategica utilizzata per valutare i punti di forza, i punti di debolezza, le opportunità e le minacce del lavoro svolto nella presente tesi. Verrà, poi, fornita una breve descrizione riguardo le lezioni apprese durante le varie fasi del progetto.

6.1 SWOT Analysis

La *SWOT Analysis* è uno strumento utilizzato per valutare i punti di forza (*Strengths*), i punti di debolezza (*Weaknesses*), le opportunità (*Opportunities*) e le minacce (*Threats*) di un progetto svolto all'interno di una organizzazione. Essa, tipicamente, viene rappresentata tramite una matrice, come nella Figura 6.1.

La *SWOT Analysis* è utilizzata in molte organizzazioni per capire e analizzare gli aspetti rilevanti, positivi e negativi, di un progetto valutando sia le caratteristiche interne che quelle dell'ambiente competitivo esterno.

Durante la fase di valutazione si tenta di identificare i fattori chiave considerati importanti per il raggiungimento di un obiettivo. Nella matrice *SWOT* le informazioni chiave sono raggruppate in due categorie:

- fattori interni: punti di forza e debolezze;
- fattori esterni: opportunità e minacce presenti all'esterno dell'organizzazione.

6.1.1 Strengths

I punti di forza rappresentano le caratteristiche di successo del progetto. Nella presente tesi, si è deciso di definire un modello di *Anomaly Detection* definendo un profilo basato sulle sessioni degli utenti considerate "normali". Tale approccio presenta i seguenti, importanti, vantaggi:

- Il modello è in grado di rilevare vulnerabilità *0-day*; quest'ultima è una vulnerabilità critica, non nota allo sviluppatore, che dovrebbe fornire una *patch* o una soluzione immediatamente o, appunto, in "zero giorni". Nel modello proposto, basandosi su algoritmi di Intelligenza Artificiale, non c'è una distinzione

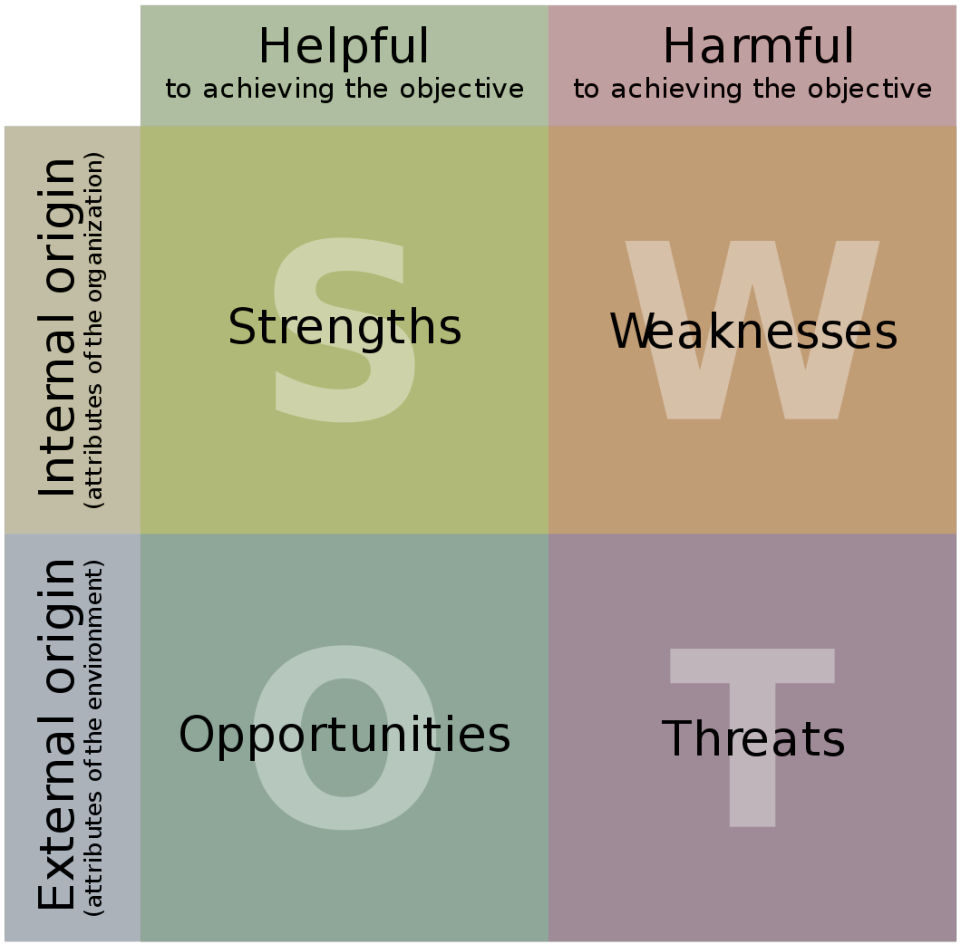


Figura 6.1: Matrice *SWOT*

tra intrusioni note e nuove; se il comportamento rilevato dal modello è insolito da quello normale si è in grado di rilevare intrusioni che sfruttano vulnerabilità *0-day*. La caratteristica discussa non è scontata; infatti, altri sistemi, per il rilevamento di attività sospette, utilizzano un approccio basato su *signature*, ovvero utilizzano un grande database di attacchi noti. Ciò presenta degli svantaggi: considerare solo attacchi noti fa sì che non sia possibile rilevare intrusioni che sfruttano *0-day* e, inoltre, il database deve essere periodicamente aggiornato per inserire i campioni dei nuovi attacchi scoperti, ma ciò fa sì che il database diventi sempre più grande portando, col tempo, ad un degrado delle prestazioni.

- Il modello richiede di definire solo il comportamento corretto, in ottica *white-list*. Ciò significa che si considerano un insieme di attività normali monitorate fino ad un determinato momento, poi si utilizzano gli algoritmi in modo tale che imparino a riconoscere le caratteristiche delle sessioni "normali" e, infine, quelle attività che non sono conformi al modello saranno rilevate come anomali.

Il modello, quindi, è basato su un approccio *whitelist*: sono considerate lecite solo quelle sessioni simili al profilo standard, rilevando come anomale tutte le altre. Per la definizione del modello non c'è, quindi, necessità di considerare gli attacchi o le altre situazioni anomale.

6.1.2 Weaknesses

Le debolezze indicano gli aspetti vulnerabili del progetto e, quindi, quelli che si dovrebbero migliorare.

Il modello di *Anomaly Detection* proposto si basa su due aspetti fondamentali: la definizione del profilo di comportamento standard e il concetto di sessione utente.

Definire il profilo standard richiede una conoscenza del dominio al fine di classificare il traffico degli utenti. Gli algoritmi utilizzati richiedono, durante la fase di *training*, le caratteristiche degli elementi di una sola classe, ovvero le sessioni "normali". Molto spesso è difficile definire il concetto di "normalità", e ciò può portare ad includere delle sessioni anomale che introducono delle contaminazioni nel modello. La maniera migliore per selezionare, esclusivamente, le sessioni "normali" è quella di esplorare i dati, al fine di individuare correlazioni, regole ed intuizioni in modo tale da considerare solo le informazioni utili. Quest'attività, come si può ben immaginare, richiede molto tempo, necessita di una profonda conoscenza da parte di un esperto e risulta essere, di conseguenza, uno degli svantaggi della metodologia.

Il modello proposto definisce il comportamento di un utente sulla base delle caratteristiche statistiche della relativa sessione; quest'ultima è caratterizzata da un intervallo temporale che, nelle analisi precedenti, si è considerato pari a 30 minuti. Analizzare la sessione di un utente, quindi, non permette di rilevare anomalie *online*, cioè non consente di individuare subito, o quasi subito, anomalie che compaiono nel sistema. In uno scenario d'utilizzo reale, inizialmente, a partire dai log grezzi, si ricostruiscono le sessioni e poi si utilizza il modello per valutare se sono presenti sessioni anomale; questo scenario indica che il modello proposto è ideato per funzionare *offline* o, al limite, con un scarto, rispetto al tempo corrente, pari all'intervallo temporale scelto per la sessione; cioè, ad esempio, se si considera una sessione di 30 minuti, come minimo, si potranno analizzare le sessioni generate in quest'ultima finestra temporale. L'approccio scelto, di conseguenza, potrebbe portare ad un ritardo nel rilevamento di un'anomalia che, in alcuni casi, non è accettabile.

6.1.3 Opportunities

Le opportunità sono delle condizioni esterne che favoriscono l'adozione del progetto e che, quindi, si devono tentare di cogliere.

Il 24 Maggio 2018 è entrato in vigore il Regolamento Generale sulla Protezione dei Dati, noto come GDPR, un regolamento emanato dall'Unione Europea, che illustra le norme in materia di trattamento di dati personali e privacy. Gli articoli 33 e 34 del regolamento illustrano gli obblighi e le norme che devono essere rispettate in caso di un'intrusione informatica. La violazione di dati personali è definita come la violazione di sicurezza che comporta, accidentalmente o in modo illecito. la distribuzione, la perdita, la modifica, la divulgazione non autorizzata o l'accesso ai dati personali trasmessi, conservati, o comunque trattati. In caso di intrusione,

il titolare del trattamento dei dati avrà l'obbligo legale di rendere note le fughe di dati all'autorità nazionale e di comunicarle entro 72 ore da quando ne è venuto a conoscenza. Il titolare del trattamento, inoltre, deve descrivere le misure adottate o di cui propone l'adozione, per porre rimedio alla violazione dei dati personali e per attenuarne i possibili effetti negativi.

Il modello di *Anomaly Detection* proposto rappresenta, quindi, uno strumento utile per garantire la conformità al regolamento, permettendo di rilevare e monitorare il comportamento degli utenti di un'applicazione Web.

Il lavoro nella presente tesi, inoltre, sfrutta, come già detto in precedenza, algoritmi di Intelligenza Artificiale nell'ambito della *Cybersecurity*. L'Intelligenza Artificiale sta trasformando le attività legate alla sicurezza informatica. Negli ultimi anni, la *Cybersecurity* e l'Intelligenza Artificiale stanno diventando un binomio indissolubile per contrastare le minacce informatiche ed evitare attacchi informatici e disservizi che, in generale, possono affliggere l'ecosistema IT di un'organizzazione. L'Intelligenza Artificiale al servizio della sicurezza informatica permette di potenziare le capacità predittive dei sistemi di controllo e, quindi, consente alle organizzazioni pubbliche e private di usufruire di strumenti in grado di contrastare i sempre più sofisticati attacchi informatici.

Il lavoro svolto nella presente tesi tratta degli argomenti che rappresentano alcuni dei più rilevanti trend dell'attuale periodo storico e, senza dubbio, anche del prossimo futuro.

6.1.4 Threats

Le minacce sono le condizioni esterne che ostacolano la buona riuscita del progetto. La *Cybersecurity* rappresenta uno dei maggiori topic degli anni recenti e ciò fa sì sempre più aziende stiano effettuando notevoli investimenti in questo ambito. *IBM*, *Splunk* e *MixMode* sono esempi di grandi aziende che hanno sviluppato piattaforme software molto potenti che sfruttano algoritmi di Intelligenza Artificiale per rilevare anomalie nei dati. La concorrenza, quindi, offre strumenti simili e rappresenta una prima minaccia che può oscurare le funzionalità offerte dalla presente soluzione.

Un'ulteriore minaccia esterna riguarda lo sviluppo del modello di *Anomaly Detection*; in quest'ultimo si richiede la conoscenza specifica del comportamento standard di un utente e tale requisito fa sì che la soluzione proposta non sia velocemente adattabile ad altri scenari. In particolare, durante la fase di *training*, il modello apprende le caratteristiche delle sessioni valutate come "normali". Lo svantaggio consiste nel fatto che non esistono regole prestabilite per valutare la sessione di un utente come normale o anomala; ciò che è considerato "normale" in un'applicazione Web può essere considerato insolito per un'altra, e viceversa. Il comportamento standard, quindi, dipende dal caso specifico, ovvero richiede di valutare e selezionare quelle sessioni considerate lecite.

Un'ultima minaccia riguardo il "ciclo di vita" dell'algoritmo. Il modello di *Anomaly Detection* si basa, come detto, su un insieme di sessioni utente valutate positivamente. Nel corso del tempo, a causa di modifiche alla struttura dell'applicazione o di altri cambiamenti, il comportamento degli utenti può cambiare. L'algoritmo, quindi, periodicamente, deve essere aggiornato in modo tale da essere in

grado di rilevare tali cambiamenti. L'aggiornamento del modello richiede una nuova progettazione utilizzando i dati più recenti.

6.2 Lezioni apprese

Il lavoro svolto nella presente tesi ha permesso di fare delle esperienze riguardo numerose metodologie adottate e ha consentito di sperimentare diversi strumenti software.

In questa tesi si è sviluppato un modello di *Anomaly Detection* utilizzando dei dati grezzi provenienti da uno scenario reale. Ciò ha consentito di acquisire consapevolezza riguardo i diversi problemi che possono capitare in questo contesto.

Una diretta conseguenza di quanto appena detto riguarda il fatto che, nei processi di analisi con dati di questo tipo, le fasi di preparazione ed esplorazione occupano la maggior parte del tempo e incidono, in maniera decisiva, nelle performance del modello. In particolare, la fase di *Data Exploration* ha consentito di individuare delle problematiche nei log, riguardo l'indirizzo IP del mittente, ignote anche al responsabile dell'applicazione Web, e che hanno richiesto importanti modifiche nelle fasi di analisi successive.

7

Conclusioni e uno sguardo al futuro

Nella seguente tesi è stato proposto un modello di *Anomaly Detection*, basato su algoritmi di *Machine Learning* e *Deep Learning*, al fine di individuare attività sospette in un Server Web.

Lo sviluppo del modello ha richiesto una metodologia che si basa sull'analisi dei log grezzi; questi ultimi contengono una serie di informazioni riguardo gli utenti che hanno generato del traffico Web. L'analisi dei log è una fase complessa che richiede strumenti adatti; pertanto, per tale attività, si è deciso di sfruttare i software della famiglia *Elastic*. In particolare, *Kibana*, la piattaforma di *Data Visualization* utilizzata, fornisce un insieme di tool che ha consentito di effettuare ricerche e visualizzazioni dei dati. L'approccio proposto si basa sulla definizione di un profilo del comportamento standard e corretto di un utente. Le attività di un utente sono valutate analizzando la sua sessione, ovvero il traffico che ha generato in uno specifico intervallo di tempo. La metodologia adottata ha, quindi, richiesto, a partire dai log grezzi, di ricostruire l'insieme di attività svolte da un utente e, poi, individuare un insieme di caratteristiche che consentono di descrivere una sessione. Successivamente si sono utilizzati degli algoritmi di Intelligenza Artificiale per definire un modello che apprenda il profilo standard; le sessioni che si discostano da quest'ultimo saranno rilevate come anomalie.

Questa tesi si compone di due parti principali. Nella prima si è sviluppato un modello basato su dati sperimentali che, tramite delle facilitazioni che presentava, ha accelerato e favorito il processo di analisi. Per questo modello è importante sottolineare l'integrazione con MISP, una piattaforma software utilizzata per la condivisione e lo scambio di informazioni di sicurezza, e l'utilizzo di algoritmi *supervised* e *unsupervised* di *Machine Learning*, implementati grazie a *Scikit-learn*.

Nella seconda parte si è utilizzato un dataset contenente i log di un Web Server di produzione proveniente da uno scenario reale, al fine di valutare e migliorare il modello precedente. L'analisi, in tal caso, è stata molto più complessa e si sono potute comprendere le difficoltà che si possono rilevare quando si effettuano analisi di dati reali. In particolare, a causa della complessità del dataset e di errori presenti nei log, in alcune fasi, si è resa necessaria una ri-progettazione della metodologia. Per questo modello, una novità rispetto al precedente, consiste nel fatto che si è utilizzato *Keras*, una libreria open source, che ha consentito di implementare un modello di *Anomaly Detection* basato su una *Deep Artificial Neural Networks*.

In entrambi i casi descritti, i risultati ottenuti sono soddisfacenti. Nello specifico, il secondo caso si basa su uno scenario reale, quindi, il modello di *Anomaly Detection* sviluppato può essere utilizzato e testato nell'ambiente per individuare comportamenti anomali o sospetti degli utenti dell'applicazione Web.

Il modello proposto, in futuro, può essere ulteriormente migliorato. L'attuale modello richiede, per rilevare nuovi pattern di comportamenti corretti, una nuova progettazione. Un possibile sviluppo futuro, quindi, è quello di progettare una metodologia in grado di aggiornare, in maniera incrementale, il modello, al fine di apprendere continuamente nuovi pattern di comportamenti corretti.

In alternativa, si potrebbe decidere di perseguire lo stesso obiettivo cambiando approccio; Nel nostro caso, infatti, si è deciso di raggruppare i log in sessioni utente; tuttavia, un'idea alternativa da esaminare potrebbe consistere nel definire un'analisi basata sulla sequenza di log prodotti da una certa esecuzione di un utente.

Riferimenti bibliografici

1. What is Cyber-Security? https://www.kaspersky.com/resource-center/definitions/what-is-cyber-security, 2020.
2. Number of connected devices reached 22 billion, where is the revenue? https://www.helpnetsecurity.com/2019/05/23/connected-devices-growth/, 2019.
3. Cybersecurity: a global problem. https://www.forbes.com/sites/forbestechcouncil/2017/01/17/why-cybersecurity-should-be-the-biggest-concern-of-2017/, 2019.
4. The real cost of data breaches in 2019 (and how to avoid them). https://www.kaspersky.com/blog/secure-futures-magazine/cybersecurity-economics-report-2019/28913/, 2019.
5. How to Hunt For Security Threats. https://www.gartner.com/smarterwithgartner/how-to-hunt-for-security-threats/, 2018.
6. A Framework for Cyber Threat Hunting. https://www.threathunting.net/files/framework-for-threat-hunting-whitepaper.pdf, 2016.
7. A Practical Model for Conducting Cyber Threat Hunting. https://www.sans.org/reading-room/whitepapers/threathunting/practical-model-conducting-cyber-threat-hunting-38710, 2018.
8. Anomaly Detection : A Survey. http://cucis.ece.northwestern.edu/projects/DMS/publications/AnomalyDetection.pdf, 2009.
9. What is NGINX? https://www.nginx.com/resources/glossary/nginx/, 2020.
10. Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly, 2017.
11. OWASP. *OWASP Top 10 -2017: The Ten Most Critical Web Application Security Risks*. 2017.
12. Our Story. https://www.elastic.co/about/history-of-elasticsearch, 2020.
13. What is Elasticsearch? https://www.elastic.co/what-is/elasticsearch, 2020.
14. Logstash: Centralize, transform & stash your data. https://www.elastic.co/products/logstash, 2020.
15. What Is Kibana? https://www.elastic.co/what-is/kibana, 2020.
16. Discover. https://www.elastic.co/guide/en/kibana/current/discover.html, 2020.
17. What is it? https://github.com/pandas-dev/pandas, 2020.
18. What is NumPy? https://docs.scipy.org/doc/numpy-1.13.0/user/whatisnumpy.html, 2020.
19. Scikit-Learn. https://github.com/scikit-learn/scikit-learn, 2020.
20. Keras: The Python Deep Learning library. https://keras.io/, 2020.

21. Quick Start MISP. https://www.circl.lu/doc/misp/quick-start/, 2020.
22. Isolation Forest. https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf, 2020.
23. Support Vector Method for Novelty Detection. https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf, 2020.
24. The Hunter Strikes Back: The SANS 2017 Threat Hunting Survey. https://www.sans.org/reading-room/whitepapers/threathunting/paper/37760, 2017.