Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Interpretazione dei log di IDS ed estrazione di policies per WAF

Relatori
prof. Fulvio Valenza
prof. Riccardo Sisto
prof. Daniele Bringhenti

Candidato
Pietro Ubertini

Anno Accademico 2023-2024

Sommario

Le tecnologie di *Software-Defined Networking* (SDN) e di *Network Function Virtualization* (NFV) rappresentano i punti di partenza per l'esplorazione della nuova frontiera della ricerca: l'Automazione della Sicurezza nelle reti. I paradigmi SDN e NFV garantiscono un alto livello di flessibilità e dinamicità nella definizione e nella configurazione delle reti virtualizzate, fattori essenziali per lo sviluppo di strumenti per la gestione automatizzata delle funzioni di sicurezza.
Attraverso la NFV è possibile separare le funzioni di rete dai dispositivi hardware dedicati, permettendo l'esecuzione di tali funzioni tramite processi software su server di tipo general-purpose. L'impiego di questa tecnologia semplifica significativamente la realizzazione del *Service Graph* (SG), una generalizzazione della *Service Function Chain*: si tratta di una rappresentazione della topologia logica della rete, costituita da funzioni di rete interconnesse tra loro. L'amministratore di rete si occupa solitamente della creazione del SG, allocando e configurando manualmente le *Network Security Functions* (NSFs). La necessità dello sviluppo di un meccanismo di automazione, nasce proprio dal fatto che tra le prime cause di attacchi informatici alle reti moderne c'è la mal configurazione delle funzioni di sicurezza. Gli amministratori tendono infatti ad utilizzare un approccio manuale nell'allocazione e nella configurazione delle funzioni di rete, realizzando soluzioni non ottimali o, nel peggiore dei casi, non corrette. Queste NSFs includono ad esempio *Web Application Firewalls* (WAFs), che hanno un ruolo cruciale nella protezione dei web server della rete che forniscono il servizio.
Alla luce di queste considerazioni, l'obiettivo di questo elaborato è quello di presentare una soluzione esistente al problema sopracitato, estendendone le funzionalità relative ai WAF. Il lavoro di tesi ha infatti contribuito allo sviluppo di *VEREFOO* (VEried REFinement and Optimized Orchestration), un framework che propone un approccio ottimale ed automatizzato per garantire il rispetto dei requisiti di sicurezza di una rete, limitando dunque gli errori umani legati alla configurazione delle NSFs. Partendo da un SG in input e dei *Network Security Requirements* (NSRs), espressi dall'amministratore di rete con un linguaggio di alto livello, VEREFOO effettua la formulazione e la risoluzione di un problema *Max Satisfiability Modulo Theories* (MaxSMT). Sulla base del risultato di tale problema, il framework restituisce in output un SG comprensivo del numero minimo di NSFs allocate, configurando per ogni funzione il minimo numero di *policies* necessarie per soddisfare i NSRs.
Questo elaborato è incentrato sullo studio dell'output degli *Intrusion Detection Systems* (IDSs) e sull'implementazione di un meccanismo in grado di estrarre delle *policies* di protezione da attacchi web, configurate nei WAFs da VEREFOO in base al risultato del problema MaxSMT.

Ringraziamenti

Lo sviluppo di questa tesi rappresenta il culmine di un intenso percorso accademico di studio e di ricerca, durante il quale ho avuto modo di arricchire il mio bagaglio culturale grazie al supporto e all'incoraggiamento di molte persone. Con immensa gratitudine, desidero ringraziare i miei supervisori di tesi: il Prof. Valenza, il Prof. Sisto ed in particolare il Prof. Bringhenti, il cui sostegno si è rivelato di fondamentale importanza durante lo svolgimento della tesi. La loro guida è stata inestimabile.

Indice

Elenco delle figure 8

Elenco dei codici 9

1 Introduzione 12

1.1 Introduzione alla tesi 12

1.2 Struttura dell'elaborato 13

2 Network Security Automation 15

2.1 Software Defined Networking e Network Function Virtualization . . 15

2.1.1 Le architetture SDN 15

2.1.2 La Virtualizzazione delle Funzioni di Rete 17

2.2 VEREFOO, il workflow ed il problema MaxSMT 19

2.2.1 Il Framework 19

2.2.2 Il problema MaxSMT in VEREFOO 19

2.2.3 Il workflow 20

3 Classificazione OWASP e ModSecurity 23

3.1 La classificazione OWASP ed il Core Rule Set 23

3.2 Il WAF scelto: ModSecurity 24

3.2.1 Il modello delle regole e la rappresentazione XML in VEREFOO 26

I modelli 26

Le rappresentazioni XML 27

4 Obiettivo della Tesi 30

4.1 L'interazione tra WAF e IDS 30

4.2 L'estrazione delle OWASP policies dal log IDS 31

5 Analisi degli Intrusion Detection Systems 32
5.1 Snort . . . 32
5.1.1 Introduzione a Snort . . . 32
5.1.2 Definizione delle regole per la produzione di Alert . . . 34
5.1.3 Classificazione degli Alert prodotti da Snort . . . 36
5.1.4 Formati dei file di log di Snort . . . 37
5.2 Suricata . . . 39
5.2.1 Introduzione a Suricata . . . 39
5.2.2 Analogie con Snort nella definizione delle regole e nella classificazione degli Alert . . . 41
5.2.3 Formati dei file di log di Suricata . . . 42

6 Approccio definito per la Policy Extraction 45
6.1 Aggiornamento della classificazione degli Alerts . . . 45
6.2 Strategia di mapping degli Alerts . . . 48
6.3 Modifiche al modello ed alla rappresentazione XML delle regole OWASP . . . 49

7 Implementazione e Validazione della Policy Extraction 53
7.1 L'implementazione . . . 53
7.1.1 Gestione della configurazione in input . . . 54
7.1.2 Funzione di analisi del log ed estrazione delle OWASP policies 55
7.2 Esempi di applicazione e validazione dei risultati . . . 58
7.2.1 Caso d'uso: Scenario A . . . 58
Descrizione dello scenario e della configurazione adottata . . . 59
Simulazione di un attacco DOS e produzione di *alert logs* . . . 60
Estrazione delle *policies* e validazione dei risultati . . . 60
7.2.2 Caso d'uso: Scenario B . . . 62
Estrazione delle *policies* e validazione dei risultati . . . 63
7.3 Valutazione delle prestazioni e dei limiti della soluzione . . . 65
7.3.1 Rendimento in funzione della classificazione . . . 65
7.3.2 Tempi di esecuzione in funzione del formato e della dimensione del log . . . 66
7.3.3 Limiti della soluzione adottata . . . 67

8 Conclusioni 69

Bibliografia 71

A 74

A.1 Tabella di descrizione dei moduli del *Core Rule Set* 74

A.2 Classificazione degli alert adottata in 7.2 76

A.3 Configurazione del file di mapping 77

A.4 Classe *IdsPolicyExtraction.java* 78

A.5 Configurazione di Suricata 82

A.6 Configurazione di Snort 83

A.7 File di log Scenario A 84

A.8 Funzione di test per la validazione dei risultati 87

A.9 Test Scenario A 88

A.10 File di log Scenario B 89

A.11 Service Graph Scenario B 89

A.12 Test Scenario B 91

Elenco delle figure

1.1 Patterns delle violazioni informatiche nel tempo [Verizon DBIR] . . 13

2.1 Architettura di rete SDN 16
2.2 *Service Function Chain* relativa ad una rete tradizionale 18
2.3 *Service Function Chain* della rete virtualizzata tramite NFV e SDN 18
2.4 Architettura di VEREFOO 20
2.5 Esempio di *Allocation Graph* 21

3.1 Classificazione *OWASP Top Ten* del 2021 24

4.1 Esempio di architettura di rete generica con WAF 31

6.1 Esempio di classificazione ibrida per la definizione delle regole negli IDS 48
6.2 Esempio configurazione del file di mapping nello scenario in 6.1 . . 49
6.3 Esempio di conversione di regola OWASP da linguaggio di medio ad alto livello 50

7.1 Schema di rete dello scenario descritto 59
7.2 Schema logico per lo Scenario A 59
7.3 Risultato della funzionalità di *Policy Extraction* per lo Scenario A . 61
7.4 Allocazione del WAF ed installazione delle OWASPprop nello Scenario A 62
7.5 Schema di rete Scenario B 62
7.6 Risultato della funzionalità di *Policy Extraction* per lo Scenario B . 64
7.7 Aggiornamento dell'istanza del WAF dello Scenario B 64
7.8 Esempio di classificazione aggregata 65
7.9 Lista di OWASPprop prodotta 66
7.10 Tempi di estrazione Scenario A (10000 segnalazioni a file) 66
7.11 Tempi di estrazione al variare del numero di segnalazioni e del formato del log 67

Elenco dei codici

3.1 Principali direttive di configurazione e definizione delle regole in ModSec . . . 25
3.2 Schema XSD di un *Network Security Requirement*, . . . 27
codes/nfvSchema.xsd . . . 27
3.3 Schema XSD di Property . . . 28
3.4 Schema XSD di HTTPDefinition . . . 28
3.5 Schema XSD di HTTPDefinition . . . 28
3.6 Schema XSD delle regole OWASP . . . 29
3.7 Schema XSD del WAF . . . 29
5.1 Esempio di utilizzo di *ipvar* all'interno di *snort.conf* . . . 33
5.2 Esempio di utilizzo di *portvar* all'interno di *snort.conf* . . . 34
5.3 Esempio di utilizzo di *var* all'interno di *snort.conf* . . . 34
5.4 Esempio di regola estratto dal *rule set* della *Snort Community* . . . 35
5.5 Contenuto di default del file *classification.conf* . . . 36
5.6 Esempio di *alert* generato da Snort . . . 37
5.7 Attivazione degli *Output Modules* in *snort.conf* . . . 38
5.8 Esempio di log degli alert prodotti da Snort in modalità *full* . . . 38
5.9 Esempio di log degli alert prodotti da Snort in modalità *fast* . . . 39
5.10 Esempio di configurazione dei parametri di rete in Suricata . . . 40
5.11 Inclusione del *rule set* locale nel file *suricata.yaml* . . . 41
5.12 Attivazione della *alert-debug mode* in Suricata . . . 41
5.13 Esempio di classificazione, definizione della regola ed alert prodotto in Suricata . . . 42
5.14 Configurazione del modulo *alert-debug* in Suricata . . . 42
5.15 Attivazione dell'*EVE format* ed esempio di alert prodotto . . . 43
5.16 Attivazione del modulo *fast* ed esempio di alert prodotto . . . 43
6.1 Definizione del *classtype* associato ad attacchi web . . . 46
6.2 Esempio di alert prodotto utilizzando la classificazione di default . . . 46
6.3 Esempio di regola per il rilevamento di un attacco XSS . . . 46
6.4 Esempio di alert prodotto dalla regola 6.3 . . . 46
6.5 Definizione del nuovo *classtype* per l'identificazione di XSS . . . 47
6.6 Aggiornamento della regola 6.3 con il *classtype* appena definito . . . 47
6.7 Esempio di alert prodotto dalla regola 6.6 . . . 47
6.8 Aggiornamento della classificazione installata in Snort e Suricata . . . 47
6.9 Aggiornamento del set di possibili valori per una regola OWASP . . . 50
6.10 Precedente versione di *PropertyDefinition* . . . 51
6.11 Rappresentazione dello schema xsd di una *OWASPprop* . . . 51
6.12 Aggiornamento definizione *PropertyDefinition* . . . 52

6.13 Classe generata automaticamente da AJAX a partire da 6.12 52
7.1 Costruttore della classe implementata 54
codes/IdsPolicyExtraction.java 55
7.2 Regex definite per l'estrazione dei dati 56
7.3 Estrazione della classificazione 56
codes/IdsPolicyExtraction.java 56
7.4 Estrazione degli indirizzi IP 57
7.5 Aggiornamento della lista di *OWASPprop* 57
7.6 Controllo sugli indirizzi IP 58
7.7 Avvio monitoraggio del traffico su VM1 59
7.8 Script Python lanciato per simulare l'attacco DOS 60
codes/scenario1.xml 60
7.9 Rappresentazione XML Scenario A 61
7.10 *Mapper.config* definito dalla classificazione 7.8 65
7.11 Incremento delle righe nel *mapper.config* 66
7.12 Configurazione del *mapper.config* per gli Scenari A e B 68
codes/final_classification.config 76
A.1 Classificazione aggiornata installata sugli IDS 77
codes/final_mapper.config 77
A.2 Configurazione adottata del file *mapper.config* 78
codes/IdsPolicyExtraction.java 78
codes/IdsPolicyExtraction.java 79
codes/IdsPolicyExtraction.java 80
codes/IdsPolicyExtraction.java 81
A.3 *IdsPolicyExtraction.java* 82
codes/suricata_example.conf 82
A.4 Suricata.yaml 83
A.5 custom.rules 83
A.6 Snort.conf 84
A.7 local.rules 84
A.8 Fast log prodotti da Snort e Suricata 84
A.9 Snort Verbose Log 85
A.10 Formato Json del log di Suricata 85
A.11 Suricata Alert Debug 86
A.12 Funzione di test 87
A.13 Funzione di test Scenario A 88
A.14 Fast log Scenario B 89
codes/scenariob.xml 89
codes/scenariob.xml 90
A.15 Service Graph Scenario B 91
codes/scenarioB_test.java 91
A.16 Funzione di test Scenario B 92

Capitolo 1

Introduzione

1.1 Introduzione alla tesi

Tra i paradigmi adottati nelle reti moderne, i più rilevanti sono *Software Defined Networking* (SDN) e *Network Function Virtualization* (NFV). La tecnologia di SDN garantisce una gestione centralizzata delle risorse di rete, separando il controllo della rete stessa dal forwarding dei dati. La NFV ha come obiettivo la virtualizzazione delle funzioni di rete: questo paradigma prevede il disaccoppiamento delle funzioni di rete dai moduli hardware dedicati, permettendo l'esecuzione di tali funzioni da macchine virtuali installate su server di tipo general-purpose. L'impiego combinato di queste tecnologie nella definizione e nella configurazione delle reti moderne garantisce vantaggi significativi in termini di flessibilità, scalabilità e dinamicità della rete, oltre ad una notevole riduzione dei costi dovuta all'assenza di dispositivi hardware dedicati. Ulteriori benefici della NFV risiedono nella creazione dei *Service Graph* (SG). Il SG consiste in una rappresentazione logica della topologia della rete virtuale, costituita da funzioni di rete interconnesse tra loro, ed è solitamente definita dall'amministratore di rete. Nonostante la NFV semplifichi la composizione del SG, si tratta di un processo manuale, dunque suscettibile ad errori da parte dell'amministratore. Tra le funzioni di rete da allocare, ci sono le *Network Security Functions* (NSFs) come firewalls, VPN gateways, Intrusion Detection Systems, di fondamentale importanza per la protezione della rete da attacchi informatici. L'approccio manuale adottato dagli amministratori di rete, porta a soluzioni non ottimali o, nel peggiore dei casi, errate, pregiudicando i tempi di reazione in caso di attacco informatico. Per questo motivo, sono stati sviluppati degli strumenti di gestione delle NSFs in grado di supportare l'amministratore in fase di design della rete. Tuttavia, si tratta di una soluzione parziale del problema, in quanto resta strettamente legata alle competenze tecniche dell'amministratore.
Come si evince dal *Data Breach Investigation Report* [1] del 2024 prodotto da *Verizon*, gli asset più colpiti dagli attacchi informatici restano i *Web Application Servers*. Dall'analisi dell'evoluzione temporale delle violazioni informatiche in figura 1.1, è possibile notare un sostanziale peggioramento degli attacchi dovuti ad errori di configurazione dei sistemi che, con l'evoluzione delle tecnologie adottate, diventano sempre più complicati da gestire dal punto di vista della sicurezza.

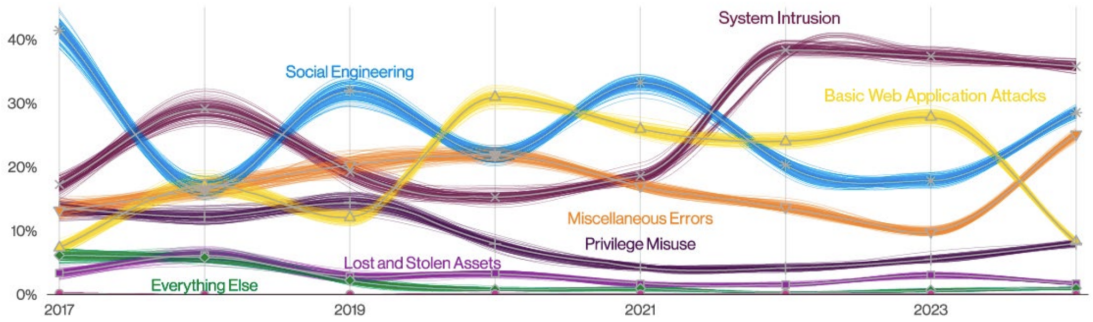

Figura 1.1. Patterns delle violazioni informatiche nel tempo [Verizon DBIR]

Da queste considerazioni, è nata la necessità dello sviluppo di un meccanismo di Automazione della Sicurezza nella configurazione delle reti virtualizzate [2]. *VEREFOO* (VErified REFinement and Optimized Orchestration) rappresenta una soluzione al problema precedentemente descritto, proponendo un approccio automatizzato ed ottimale per l'allocazione e la configurazione delle NSFs all'interno delle reti. Partendo da un SG, associato alla topologia logica di una rete, e da un insieme di *Network Security Requirements* (NSRs) da rispettare, VEREFOO è in grado di produrre un SG in output contenente le NSFs, ciascuna configurata con le relative *policies*, necessarie per il rispetto dei requisiti di sicurezza in input. Per arrivare a questo risultato, VEREFOO sfrutta Z3, un *theorem solver* sviluppato da Microsoft Research, formulando e risolvendo un problema parzialmente pesato di tipo MaxSMT.
Questo lavoro di tesi si concentra sull'estensione delle funzionalità di VEREFOO in modalità *Web Application Security* (WAS). In particolare, considerando gli *Open Source* IDS in commercio, l'elaborato descrive l'implementazione di un meccanismo in grado di interpretare i file di log prodotti dagli IDS. L'obiettivo è quello di estrarre delle *policies* dalle informazioni relative ad attacchi web contenute nei file di log, per poi calarle opportunamente, qualora necessario, nei WAFs allocati nel SG finale da VEREFOO.

1.2 Struttura dell'elaborato

Dopo aver brevemente introdotto nella sezione 1.1 le tematiche affrontate e gli obiettivi raggiunti, il resto della tesi è strutturata come segue:

- Capitolo 2: descrive in maniera dettagliata le tecnologie di NFV e SDN, già introdotte in precedenza, ed il funzionamento di VEREFOO, come soluzione al problema della *Security Automation*;
- Capitolo 3: illustra sinteticamente i principi dell'*Open Web Application Security Project* ed il WAF adottato in VEREFOO per la protezione da attacchi web;
- Capitolo 4: presenta in maniera esaustiva gli obiettivi della tesi;

- Capitolo 5: tratta l'analisi degli IDS in commercio, evidenziando analogie e differenze nella definizione delle regole e nella classificazione delle anomalie rilevate, ed illustra lo studio dei formati dei file di log in output;
- Capitolo 6: illustra le modifiche effettuate al modello dei requisiti di sicurezza e descrive l'approccio adottato per il processo di estrazione delle regole OWASP da calare nei WAFs;
- Capitolo 7: presenta l'implementazione del meccanismo di estrazione delle *policies*, con esempi di applicazione e validazione dei risultati raggiunti.
- Capitolo 8: conclude la tesi descrivendo gli obiettivi raggiunti e fornendo spunti per futuri lavori.

Capitolo 2

Network Security Automation

Come evidenziato nel capitolo introduttivo dell'elaborato 1, le tecnologie di *Software Defined Networking* e *Network Function Virtualization* rappresentano il punto di partenza per lo sviluppo di strumenti di automazione della sicurezza nelle reti virtualizzate [3]. La flessibilità e la dinamicità introdotte nelle reti moderne da questi paradigmi semplificano notevolmente lo sviluppo e l'aggiornamento di nuove funzioni di rete, non essendoci vincoli con dispositivi hardware dedicati tipici delle reti tradizionali.
L'obiettivo di questo capitolo, partendo dalla descrizione approndita delle tecnologie sopra citate, è quello di presentare VEREFOO come soluzione al problema della *Security Automation*, evidenziando i concetti fondamentali ed esponendo il funzionamento del framework.

2.1 Software Defined Networking e Network Function Virtualization

2.1.1 Le architetture SDN

La tecnologia di *Software Defined Networking* (SDN) si basa sulla separazione fisica del controllo della rete dal forwarding dei pacchetti. Le architetture di rete definite in questo contesto, sono costruite su tre livelli: l'*application layer*, l'*infrastructure layer* ed il *control layer* [4].
Partendo dal livello più basso, l'*infrastructure layer* rappresenta l'insieme dei dispositivi fisici appartenenti alla rete. A differenza delle reti tradizionali in cui il *control plane* ed il *data plane* sono gestiti direttamente da switch e router, nelle architetture SDN questi dispositivi sono passivi e rappresentano dei semplici *Forwarding Devices*: la loro unica funzione è quella di inoltrare i pacchetti da un'interfaccia all'altra, basandosi sulle politiche di instradamento definite nel *control layer*.
Il *control layer* rappresenta il livello intermedio ed ospita l'entità architetturale fondamentale per il funzionamento della rete, il *Controller SDN*, responsabile della definizione delle politiche di rete e del forwarding dei pacchetti. Il controller interagisce con gli elementi dell'*infrastructure layer* mediante l'interfaccia di programmazione (API) *OpenFlow*, fornendo un approccio centralizzato in grado di apportare

modifiche alla rete in modo dinamico e semplificato. Oltre alle *southbound interfaces*, le APIs che permettono la comunicazione con l'infrastruttura fisica della rete, il controller fornisce delle *northbound interfaces* per comunicare con l'*application layer*.
L'*application layer* è il livello più alto dell'architettura SDN. Esso ospita le applicazioni esterne in grado di manipolare la rete, comunicando con il Controller SDN tramite le *northbound interfaces*, come mostrato in figura 2.1:

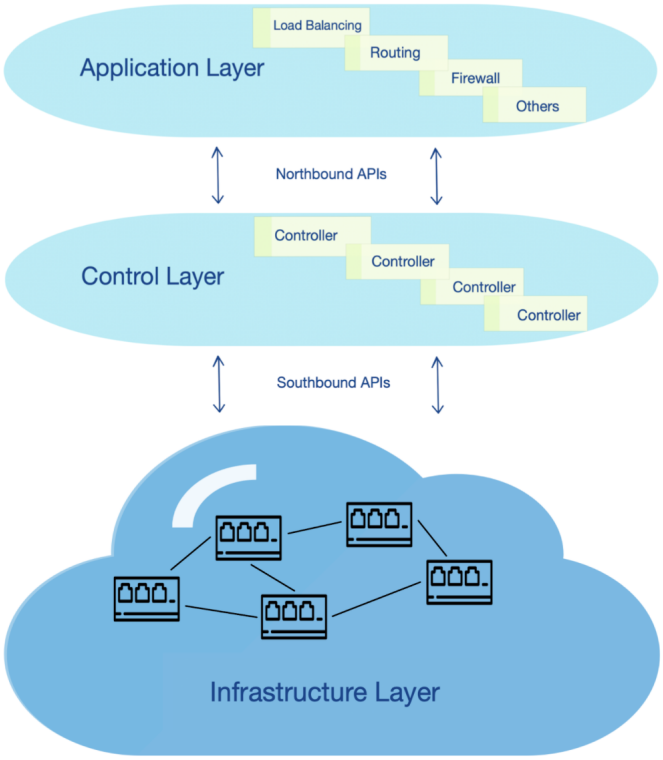

Figura 2.1. Architettura di rete SDN

Questo approccio centralizzato e programmabile delle architetture SDN garantisce una serie di vantaggi in termini di dinamicità e facilità della gestione della rete. Gli amministratori di rete, sfruttando le APIs a disposizione, possono modificare l'infrastruttura ed aggiungere nuove funzionalità senza la necessità di intervenire sugli apparati di rete coinvolti (ad esempio switch e router), in quanto le regole da installare nei *forwarding devices* possono essere programmate in maniera centralizzata. Inoltre, l'utilizzo di semplici switch in sostituzione ai complessi router per il forwarding, riduce significativamente i costi.
La centralità del Controller SDN può tuttavia rappresentare uno svantaggio, in quanto le capacità di calcolo del dispositivo sono limitate rispetto alle dimensioni sempre più estese delle reti moderne; inoltre il controller rappresenta un *single point of failure*. Di conseguenza, per migliorare la robustezza e scalabità della soluzione, sono state ideate architetture SDN di tipo gerarchico [5]: il *control layer* è costruito da più Controller SDN organizzati in modo gerarchico. Ciascuno di questi controller si occupa di una porzione dell'*infrastructure layer* e comunica con gli altri per coordinarsi e prendere decisioni al livello globale per il forwarding.

2.1.2 La Virtualizzazione delle Funzioni di Rete

L'idea alla base della tecnologia di *Network Function Virtualization* (NFV) è quella di eliminare la dipendenza delle funzioni di rete dai dispositivi hardware dedicati, permettendo l'implementazione di tali funzioni in termini di processi software eseguiti su server di tipo general purpose. L'utilizzo di questo paradigma semplifica la definizione della *Service Function Chaining* (SFC) da parte dell'amministratore di rete, ovvero la rappresentazione logica delle interconnessioni delle funzioni di rete. Una SFC è composta da una serie di *Service Functions* (SFs): si tratta di componenti logici interconnessi tra loro che integrano le molteplici funzioni di rete. A differenza di ciò che avviene per le reti tradizionali, dove ogni SF è allocata su un modulo hardware dedicato, grazie alla NFV è possibile definire le funzioni di servizio come macchine virtuali allocate su un gruppo di server interconnessi tra loro. Di conseguenza, non esistendo più una corrispondenza univoca tra la funzione di rete ed il dispositivo hardware che la implementa, più SFs possono essere allocate sullo stesso server e condividere le risorse di rete.
L'utilizzo combinato delle tecnologie di NFV e SDN permette di ottenere ulteriori vantaggi. Sfruttando la dinamicità del SDN e la flessibilità della NFV, il processo di definizione di nuove funzionalità di rete risulta molto più semplice: l'aggiunta di una nuova funzione di servizio consiste nell'allocazione di una macchina virtuale su un server, piuttosto che nell'installazione di uno o più moduli hardware dedicati, e nell'aggiornamento delle regole di forwarding dei dispositivi di collegamento, calcolate dal controller SDN e distribuite dinamicamente tramite *Switch SDN*. Dalle figure 2.2 e 2.3 è possibile cogliere la semplificazione ottenuta nella definizione della *Service Function Chain*.

Con l'evoluzione della virtualizzazione tramite SDN e NFV, le reti moderne sono diventate sempre più grandi e complesse, creando nuove opportunità di attacchi informatici e causando un drastico incremento delle violazioni nei sistemi. La flessibilità e la dinamicità offerta dai paradigmi descritti in questo capitolo, non possono essere sfruttate a pieno per le funzioni di sicurezza. Come già accennato nel capitolo introduttivo della tesi, la definizione e la configurazione delle *Network Security Functions* (NSFs) della rete si basano su un approccio manuale, suscettibile quindi ad errori umani. L'amministratore di rete effettua la configurazione delle funzioni di sicurezza, come firewall o IDS, basandosi sull'aspettativa iniziale dei possibili attacchi. In caso di errore o di violazioni inaspettate, l'amministratore dovrà aggiornare le funzioni di sicurezza della rete in modo da prevenire l'insorgenza di ulteriori attacchi; tuttavia questo approccio tradizionale può avere senso per reti relativamente piccole e non per le sempre più grandi e complesse reti virtualizzate moderne. Da queste considerazioni, è nata la necessità di un meccanismo di automazione dell'allocazione e della configurazione delle NSFs.

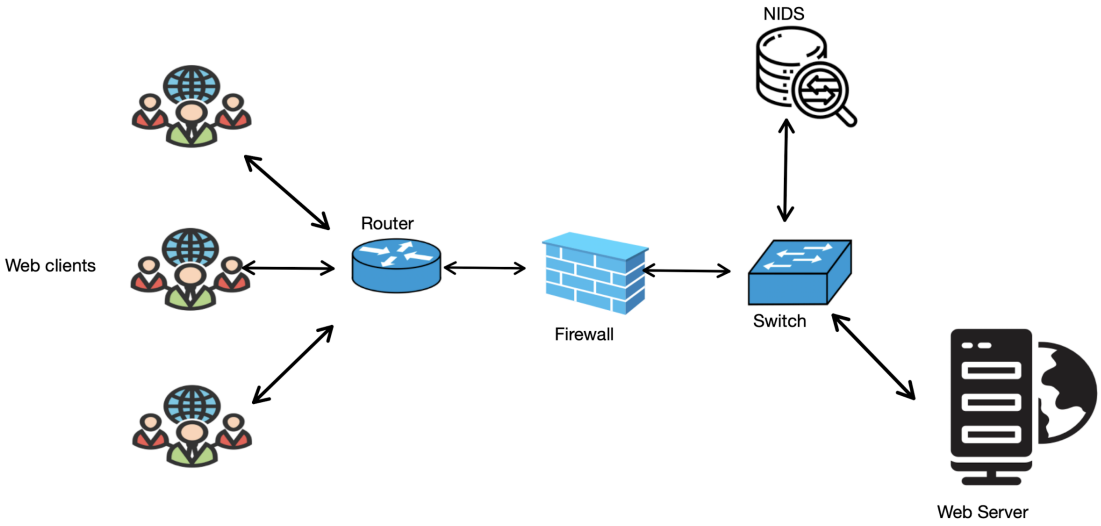

Figura 2.2. *Service Function Chain* relativa ad una rete tradizionale

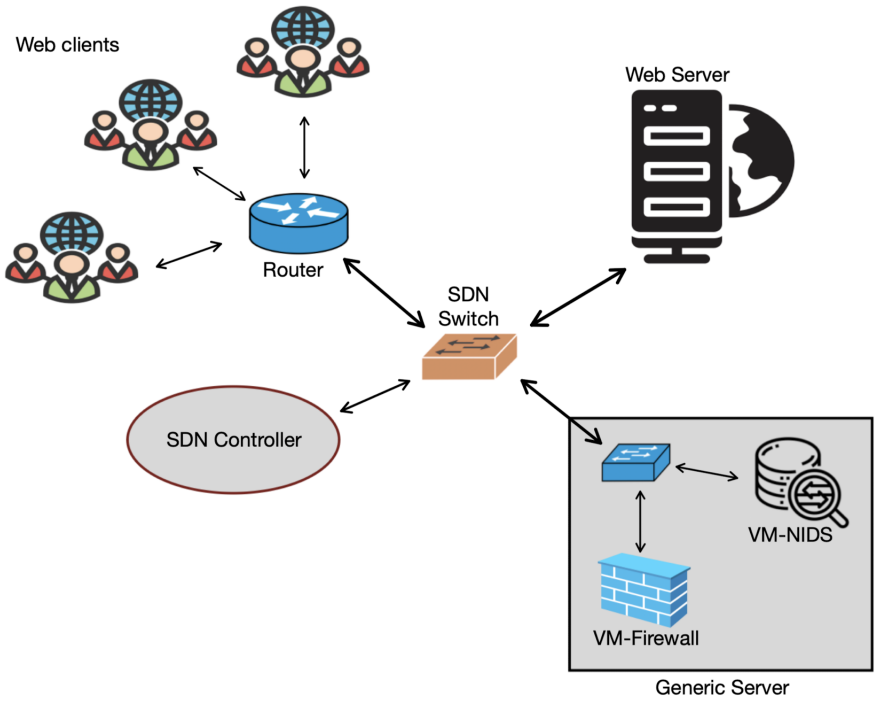

Figura 2.3. *Service Function Chain* della rete virtualizzata tramite NFV e SDN

2.2 VEREFOO, il workflow ed il problema MaxSMT

Nonostante sia un obiettivo di grande rilevanza, attualmente gli strumenti sviluppati nel campo della *Security Automation* hanno delle carenze considerevoli in termini di ottimizzazione e di verifica della correttezza. Unica soluzione in letteratura a questo problema è *VEREFOO* (VEried REFinement and Optimized Orchestration).

2.2.1 Il Framework

VEREFOO è un framework che propone un approccio ottimale ed automatizzato per l'allocazione e la configurazione delle *Network Security Funtions*. Gli aspetti fondamentali che rendono questo framework una soluzione all'avanguardia sono l'ottimizzazione e la verifica formale della correttezza dei risultati ottenuti [2]. Partendo da un set di requisiti di sicurezza in input, espressi con un linguaggio di medio o alto livello seguendo un approccio di tipo *Policy-based Management* (PBM), VEREFOO è in grado di fornire una soluzione formale ed ottimizzata - per l'allocazione e la configurazione delle NSFs - formulando e risolvendo un problema delle teorie del modulo di massima soddisfacibilità. (*Max Satisfiability Modulo Theories* - MaxSMT).

2.2.2 Il problema MaxSMT in VEREFOO

Risolvere un problema della soddisfacibilità proposizionale (SAT) consiste, a partire da una formula proposizionale e da un insieme di variabili booleane, nell'accertarsi che esista almeno una combinazione delle variabili in grado di soddisfare la formula in input; tuttavia la soluzione raggiunta per questo tipo di problema può non essere ottima. Il problema MaxSMT formulato e risolto dal framework è sostanzialmente un'estensione del problema SAT, in cui vengono applicati dei vincoli di ottimizzazione durante la ricerca della soluzione. Dati un insieme di clausole proposizionali ed un insieme di variabili booleane in input, la risoluzione di un problema MaxSMT consiste nell'identificare i valori ottimi, tra quelli assegnabili alle variabili in input, tali da massimizzare la soddisfacibilità delle clausole. Nel contesto di VEREFOO, si tratta di un problema MaxSMT di tipo *partial-weighted*:

- *partial*, poichè le clausole vengono classificate come *hard constraints* e *soft constraints*; quelle di tipo *hard* sono le clausole che devono essere necessariamente soddisfatte, mentre le clausole *soft* possono non essere rispettate

- *weighted*, poichè a ciascuna clausola può essere attribuito un peso; l'assegnazione di un peso ad ognuna delle clausole veicola la ricerca della soluzione ottima da parte del *solver*, che cercherà di massimizzare la somma dei pesi associati alle clausole soddisfatte.

L'insieme delle clausole e delle variabili, alla base della formulazione del problema sopracitato, viene estratto dal framework a partire dal set di *Network Security Requirements* (NSRs) e dal *Service Graph* (SG).

2.2.3 Il workflow

Interpretando la figura 2.4, che rappresenta il modello dell'architettura di VEREFOO, in questa sezione verrà illustrato il funzionamento del framework partendo dai concetti di SG e NSRs.

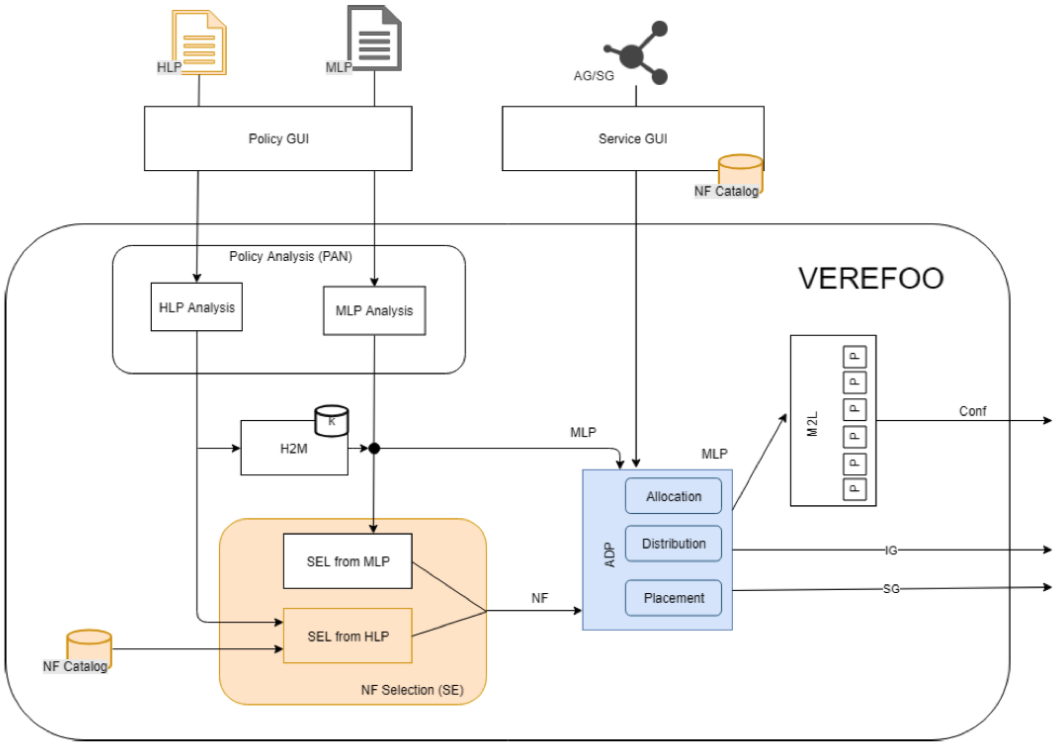

Figura 2.4. Architettura di VEREFOO

Il SG è una generalizzazione della *Service Function Chain* (SFC): consiste in una descrizione logica della topologia di rete, definita dal progettista per rappresentare l'interconnessione delle SFs. Lo schema risultante è più raffinato rispetto ad una SFC, in quanto possono coesistere percorsi diversi tra ciascuna coppia di funzioni di rete, con la possibilità di generare dei loop nel grafo. Il *service designer* fornisce il SG in input al framework mediante una *Service GUI*. Questo SG, composto da semplici funzioni di rete senza alcuna capacità di sicurezza, viene processato automaticamente da VEREFOO per costruire un *Allocation Graph* (AG): si tratta di una trasformazione della topologia logica del SG dove, tra ogni coppia di funzioni di rete virtuali, viene inserito un *Allocation Place* (AP) come mostrato in figura 2.5.
È proprio in questi APs che, in base al risultato del problema MaxSMT, verranno allocate le NSFs tali da soddisfare i requisiti di sicurezza in input. Il framework

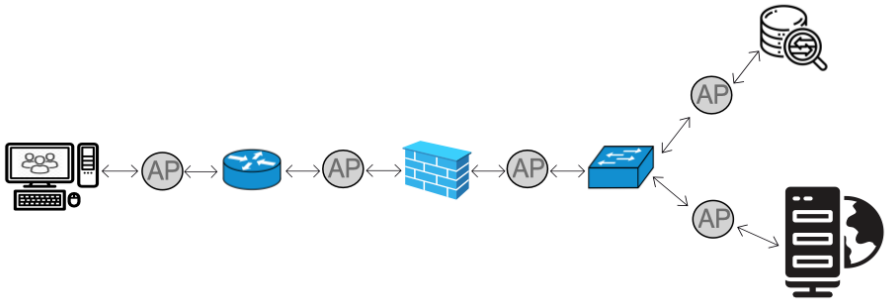

Figura 2.5. Esempio di *Allocation Graph*

permette inoltre di ricevere direttamente un AG in input, in modo da forzare l'allocazione di una funzione di sicurezza in una specifica posizione della rete.

I NSRs identificano i requisiti di sicurezza che devono essere soddisfatti e sono espressi in un linguaggio di medio o alto livello a seconda del grado di competenza dell'amministratore. Per gli utenti meno esperti, la creazione dei NSRs è facilitata grazie ad una *Policy GUI*, che permette di esprimere i requisiti con un linguaggio di alto livello e flessibile.

In fase preliminare, VEREFOO analizza i NSRs in input tramite il modulo PAN (*Policy ANalysis*): l'obiettivo di questo blocco del framework è quello di valutare la presenza di conflitti tra le *policies* in input e creare il set minimale di requisiti da rispettare. Il PAN accetta NSRs sia di alto che di medio livello; in caso di errori nella definizione delle policies che non possono essere automaticamente risolti direttamente dal modulo, viene ritornato un messaggio di non applicabilità dei requisiti all'utente per richiedere la riformulazione dei criteri di sicurezza.
Se i requisiti sono espressi con un linguaggio di alto livello, vengono successivamente riformulati tramite il modulo *High to Medium* (H2M) in un set minimale di NSRs di medio livello, i quali conterranno tutte le informazioni necessarie per l'allocazione e la configurazione delle VNFs.
Una volta formulato il set di policies, il modulo *Network Function Selection* (SE) determina il set ottimale di funzioni di rete in grado di soddisfare i requisiti. Sfruttando un catalogo predefinito di NSFs, questo modulo effettua la selezione delle funzioni necessarie sulla base di due criteri: le funzioni selezionate devono fornire le capacità richieste per soddisfare i requisiti in input ed è necessario che siano disponibili le risorse fisiche dove allocare tali funzioni. Questo modulo svolge dunque un ruolo fondamentale ed introduce un primo grado di ottimizzazione selezionando le funzioni in base al costo in termini di risorse di rete.
I NSRs di medio livello, la lista delle funzioni selezionate ed il SG originale (o l'AG) vengono elaborati dal modulo ADP (*Allocation, Distrubition and Placement module*). Si tratta dell'elemento principale dell'architettura del framework, il cui obiettivo è quello di calcolare il SG finale arricchito con le NSFs sulla base degli input precedentemente descritti. Sfruttando Verigraph [6], un framework esterno utilizzato per la valutazione del SG, e z3Opt, un solver sviluppato da Microsoft Research per la risoluzione del problema MaxSMT, il modulo ADP è in grado di:

- orchestrare l'allocazione nell'AG delle funzioni di sicurezza selezionate dal modulo SE, in modo da soddisfare i NSRs espressi attraverso il linguaggio di medio livello. Questo stadio del processo è definito come *Allocation Phase*
- in contemporanea alla fase di allocazione, distribuire le policies espresse in linguaggio di medio livello sulle NSFs allocate (*Distrubution Phase*); il set di policies installato non è necessariamente coincidente con il set di NSRs in input, poichè le policies vengono minimizzate seguendo dei criteri di ottimizzazione
- installare sull'infrastruttura fisica le VNFs che implementano il set delle funzioni di rete complessivo, quello contenente le funzioni di rete del SG di partenza e le funzioni di sicurezza allocate, e trasformare allo stesso tempo le policies configurate da medio livello a basso livello. Durante la *Placement Phase*, il processo di installazione delle VNFs è effettuato minimizzando il consumo delle risorse dell'infrastruttura fisica.

Capitolo 3

Classificazione OWASP e ModSecurity

L'evoluzione tecnologica dell'ultimo trentennio è stata fortemente condizionata dall'incessante aumento dell'utilizzo delle applicazioni web: l'impiego quotidiano delle web app, in contesti lavorativi, educativi o di semplice intrattenimento, ha spostato l'attenzione degli utenti malevoli sui web servers, che rappresentano gli asset più colpiti dagli attacchi informatici [1]. Di conseguenza, il *Web Application Firewall* ha assunto un ruolo fondamentale nella sicurezza dell'infrastruttura di rete che ospita l'applicazione.
Dopo aver precedentemente esposto nel capitolo 2 il problema della *Security Automation* ed aver presentato VEREFOO come possibile soluzione, in questa parte dell'elaborato verranno inizialmente descritti gli obiettivi dell'organizzazione *OWASP*, punto di riferimento per lo sviluppo e la protezione delle applicazioni web, per poi analizzare il funzionamento di ModSecurity, il WAF attualmente supportato da VEREFOO.

3.1 La classificazione OWASP ed il Core Rule Set

Molte delle criticità delle applicazioni web derivano dal fatto che si tende a trascurare la valutazione della sicurezza durante il design e l'implementazione delle applicazioni stesse. Ne consegue lo sviluppo di soluzioni con vulnerabilità più o meno complesse che richiedono ingenti sforzi economici per essere patchate. A tal proposito, ad inizi anni 2000 è nata l'*Open Web Application Security Project* (OWASP), un'organizzazione no-profit con l'obiettivo di fornire linee guida e strumenti sulla sicurezza delle applicazioni web.
Uno dei progetti più rilevanti è la classificazione *OWASP Top Ten*, riconosciuta al livello mondiale dagli sviluppatori come punto di partenza per lo sviluppo di codice più sicuro. Le aziende software dovrebbero adottare queste linee guida e valutare la sicurezza delle applicazioni durante l'intero processo di sviluppo, al fine di ridurre i costi in caso di correzioni. Questa classificazione comprende i dieci principali fattori di rischio per la sicurezza delle applicazioni web e viene aggiornata periodicamente. In figura 3.1, è rappresentata la classifica aggiornata, rilasciata dall'organizzazione nel 2021. Per ciascuna voce dell'elenco, OWASP fornisce un'accurata descrizione e

possibili approcci per la prevenzione.

Figura 3.1. Classificazione *OWASP Top Ten* del 2021

L'impiego di un WAF permette di aggiungere un livello di sicurezza esterno per compensare la presenza di vulnerabilità nell'applicazione web. Tuttavia, la configurazione di questa funzione di sicurezza può essere complessa, poichè prevede la selezione e l'aggiornamento di regole e *signatures* per l'analisi del traffico e la protezione del web server che esegue l'applicazione.
Per facilitare la configurazione dei WAFs, OWASP ha rilasciato il *ModSecurity Core Rule Set*: si tratta di un set di regole per il rilevamento delle tipologie di attacco più comuni, come *SQL Injection*, *Cross Site Scripting*, *Denial of Service* e così via [7]. Inizialmente pensato per *ModSecurity* (ModSec), il WAF supportato da VEREFOO che sarà descritto nel prossimo paragrafo, questo set di regole è compatibile con altre tipologie di WAFs. Sotto la supervisione dell'organizzazione, il *Core Rule Set* è stato sviluppato dalla comunità degli esperti di sicurezza seguendo le linee guida della classificazione *OWASP Top Ten* [8].

3.2 Il WAF scelto: ModSecurity

ModSec è tra gli *open source Web Application Firewall* più utilizzati nell reti moderne. Questo tipo di WAF garantisce protezione per le applicazioni web, monitorando il traffico in tempo reale e fornendo un meccanismo di logging completo delle richieste e delle risposte HTTP. La protezione offerta da ModSec è basata su tre diversi approcci:

- modello negativo o *Blacklisting*, che consiste nel bloccare solo le comunicazioni per cui è stata definita una regola, consentendo le altre;

- modello positivo o *Whitelisting*, che al contrario blocca tutto il traffico in entrata, eccetto i pacchetti che rispettano determinati requisiti specificati tramite opportune regole;

- *Known Weakness and Vulnerabilities*, tramite il quale è possibile proteggere le applicazioni web vulnerabili per cui non è stata ancora rilasciata ed installata una patch, senza intervenire sul codice sorgente

Scendendo nel dettaglio, il WAF è basato su un file *main.conf* di configurazione, contenente le informazioni relative alla modalità di funzionamento attiva ed alle regole da applicare al traffico. La definizione dei parametri di configurazione e dei criteri di sicurezza si basa su un linguaggio di alto livello specifico per ModSec, che viene interpretato internamente dal WAF tramite un *rule engine*. È inoltre possibile definire più file di configurazione, ciascuno contente le proprie regole in base alle funzioni, ed includerli nel *main.conf* seguendo un approccio modulare: è possibile attivare la protezione contro una specifica categoria di attacco web scaricando il *ModSecurity Core Rule Set* dal repository ufficiale [9] ed includendo nel *main.conf* il file di regole relativo all'attacco considerato. Nel codice 3.1 sono elencate le principali direttive di configurazione e definizione delle regole:

```
# Direttive di ModSec
# Attivazione Core Rule set e waf mode
Include /apache/conf/crs/crs-setup.conf
Include /apache/conf/crs/rules/*.conf
SecRuleEngine On

# Attivazione dell'analisi del body nelle richieste HTTP
SecRequestBodyAccess On

# Attivazione dell'analisi del body nelle risposte HTTP
SecResponseBodyAccess On

# Azione di Default in whitelisting mode.
SecDefaultAction "deny, log, status"

# Definizione regola di web filtering per bloccare '/etc/password'
SecRule REQUEST_URI "/etc/password" "id:50001"

# Definizione regola di web filtering per bloccare le richieste che
    contengono '..' nell'URL
SecRule REQUEST_URI "\.\./" "id:20002"

# Definizione regola di web filtering per bloccare le richieste proveniente
    da uno specifico IP
SecRule REMOTE_ADDR "^192\.168\.1\.250$" "id:50006"

# Definizione regola di time filtering per bloccare il traffico tra due date
    specifiche
SecRule TIME_DAY "^([2](4|5|6))$" "id:15"
```

Codice 3.1. Principali direttive di configurazione e definizione delle regole in ModSec

3.2.1 Il modello delle regole e la rappresentazione XML in VEREFOO

Durante la progettazione della funzionalità di WAF del framework, è stato sviluppato un modello ad-hoc per esprimere le regole descritte nel precedente paragrafo come NSRs in input a VEREFOO. Tale modello prevede il disaccoppiamento delle regole di *web e time filtering* dalle regole per la mitigazione di attacchi web, definite come regole *OWASP*.

I modelli

Le regole di *web e time filtering* condividono il seguente formato:

[*Action, IPsrc, IPdst, PORTsrc, PORTdst, HTTPmethod, URL, Domain, TIMEstart, TIMEend*]

dove

- *Action* rappresenta l'azione da applicare al traffico in caso di match della regola; i possibili valori sono *Allow* e *Deny*, dove *Allow* equivale alla condizione di raggiungibilità
- *IPsrc* e *IPdst* indicano gli indirizzi sorgente e destinazione per cui è definita la regola
- *PORTsrc* e *PORTdst* specificano le porte sorgente e destinazione per cui è definita la regola
- *HTTPmethod* corrisponde al metodo HTTP utilizzato nella richiesta per cui è definita la regola; i possibili valori sono *GET, HEAD, DELETE, POST, PUT, PATCH*
- *URL* e *Domain* sono le stringhe che rappresentano rispettivamente URL e dominio per cui è applicata la regola
- *TIMEstart* e *TIMEend* sono le stringhe temporali per specificare un intervallo in caso di *time filtering rule*

Le regole OWASP sono invece modellate con la seguente sintassi:

ACTIVATE rule_file

dove *rule_file* rappresenta il file di configurazione da attivare in ModSec per la protezione dalla specifica categoria di attacchi web. Ad esempio, la regola per attivare la protezione da un attacco di tipo DOS sarà "ACTIVATE DOS.conf". Dopo aver risolto il problema MaxSMT, le *policies* di tipo OWASP da calare nel WAF verranno tradotte da medio a basso livello, e saranno attivate aggiungendo i corrispondenti file di configurazione nella direttiva *include* di ModSec.

Le rappresentazioni XML

A partire dai modelli precedentemente descritti, sono stati sviluppati i corrispondenti schemi XSD per la definizione delle funzioni di sicurezza e dei NSRs nella rappresentazione XML, l'input reale del framework. Nel frammento di codice 3.2, è illustrato il modello di rappresentazione XML dei requisiti di sicurezza:

```
<xsd:element name="PropertyDefinition">
       <xsd:complexType>
              <xsd:sequence>
                     <xsd:element name="Property" type="Property"
                         minOccurs="0" maxOccurs="unbounded"/>
                     <xsd:element name="OWASPprop" minOccurs="0"
                         maxOccurs="unbounded">
                            <xsd:complexType>
                                   <xsd:attribute name="value" type="OWASP"
                                       use="required" />
                                   <xsd:attribute name="graph"
                                       type="xsd:long" use="required"/>
                                   <xsd:attribute name="src"
                                       type="xsd:string" use="required" />
                                   <xsd:attribute name="dst"
                                       type="xsd:string" use="required"/>
                                   <xsd:attribute name="isSat"
                                       type="xsd:boolean"/>
                            </xsd:complexType>
                     </xsd:element>
              </xsd:sequence>
       </xsd:complexType>
</xsd:element>
```

Codice 3.2. Schema XSD di un *Network Security Requirement*,

PropertyDefinition è l'elemento che permette di definire un NSR e può contenere un elemento di tipo *Property* oppure di tipo *OWASPprop* - o entrambi. Gli elementi *Property* si riferiscono alle regole di *Web/Time filtering*, mentre quelli di tipo *OWASPprop* si riferiscono alle regole per la protezione da attacchi web. Come mostrato nel codice 3.2.1, gli elementi di tipo *Property* sono definiti come:

```
<xsd:complexType name="Property">
       <xsd:choice>
              <xsd:element name="HTTPDefinition" type="HTTPDefinition"
                  minOccurs="0"/>
              <xsd:element name="POP3Definition" type="POP3Definition"
                  minOccurs="0"/>
       </xsd:choice>
       <xsd:attribute name="name" type="P-Name" use="required"/>
       <xsd:attribute name="graph" type="xsd:long" use="required"/>
       <xsd:attribute name="src" type="xsd:string" use="required"/>
       <xsd:attribute name="dst" type="xsd:string" use="required"/>
       <xsd:attribute name="lv4proto" type="L4ProtocolTypes" default="ANY"/>
       <xsd:attribute name="src_port" type="xsd:string"/>
       <xsd:attribute name="dst_port" type="xsd:string"/>
       <xsd:element name="TIMEprop" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
              <xsd:attribute name="timestart" type="xsd:string"
                  use="required"/>
```

```
            <xsd:attribute name="timeend" type="xsd:string" use="required"/>
            </xsd:complexType>
      </xsd:element>
      <xsd:attribute name="isSat" type="xsd:boolean"/>
      <xsd:attribute name="body" type="xsd:string"/>
</xsd:complexType>
```

Codice 3.3. Schema XSD di Property

dove

- *HTTPDefinition* contiene a sua volta i campi necessari per la definizione della regola che agisce al livello HTTP

```
<xsd:complexType name="HTTPDefinition">
      <xsd:attribute name="url" type="xsd:string" />
      <xsd:attribute name="domain" type="xsd:string" />
      <xsd:attribute name="body" type="xsd:string" />
      <xsd:attribute name="options" type="xsd:string" />
      <xsd:attribute name="httpmethod" type="xsd:string" />
</xsd:complexType>
```

Codice 3.4. Schema XSD di HTTPDefinition

- *POP3Definition* è costruito in maniera analoga ad *HTTPDefinition* e si riferisce al protocollo POP3. La direttiva *choice* che racchiude entrambe le definizioni serve per la condizione di mutua esclusività: una *Property* può contenere una regola che agisce su HTTP o POP3
- *name* è il nome della *Property*
- *graph* indica l'identificativo del grafo di riferimento che contiene la regola
- *src* e *dst* descrivono gli indirizzi IP sorgente e destinazione
- analogamente *src_port* e *src_dst* rappresentano le porte di sorgente e destinazione
- *isSat* è un attributo booleano il cui valore dipende dal risultato del problema MaxSMT. Se la condizione è soddisfatta, questo valore sarà a true
- *TIMEprop* rappresenta l'elemento XML utilizzato per il *time filtering*. Gli attributi *timestart* e *timeend* descrivono l'intervallo temporale durante il quale la regola deve essere attiva

```
      <xsd:element name="TIMEprop" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
            <xsd:attribute name="timestart" type="xsd:string"
                use="required"/>
            <xsd:attribute name="timeend" type="xsd:string"
                use="required"/>
            </xsd:complexType>
      </xsd:element>
```

Codice 3.5. Schema XSD di HTTPDefinition

La struttura delle *OWASPprop* è stata definita direttamente all'interno dello schema XSD 3.2 relativo all'elemento *PropertyDefinition*:

```
<xsd:element name="OWASPprop" minOccurs="0"
    maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:attribute name="value" type="OWASP"
                    use="required" />
                <xsd:attribute name="graph"
                    type="xsd:long" use="required"/>
                <xsd:attribute name="src"
                    type="xsd:string" use="required" />
                <xsd:attribute name="dst"
                    type="xsd:string" use="required"/>
                <xsd:attribute name="isSat"
                    type="xsd:boolean"/>
        </xsd:complexType>
</xsd:element>
```

Codice 3.6. Schema XSD delle regole OWASP

Come specificato nel capitolo 2, partendo dagli XML descritti, VEREFOO analizzerà i requisiti in input per formulare e risolvere il problema MaxSMT, producendo una rappresentazione XML in output relativa al SG finale, comprensiva delle funzioni di sicurezza propriamente configurate per il rispetto dei NSRs. Nel caso di allocazione di un WAF, nel SG finale verrà definito un nodo con attributo *functional_type=WEB_APPLICATION_FIREWALL*, che a sua volta conterrà un elemento *web_application_firewall* definito nel seguente schema XSD 3.7:

```
<xsd:element name="web_application_firewall">
       <xsd:complexType>
              <xsd:choice>
                     <xsd:element ref="waf_elements" minOccurs="0"
                         maxOccurs="unbounded" />
                     <xsd:element name="owasp_rule" type="xsd:string"
                         minOccurs="0" maxOccurs="unbounded" />
              </xsd:choice>
              <xsd:attribute name="defaultAction" type="ActionTypes"/>
       </xsd:complexType>
</xsd:element>
```

Codice 3.7. Schema XSD del WAF

Capitolo 4

Obiettivo della Tesi

Sulla base dei concetti espressi nei precedenti capitoli 2 e 3, in questa parte dell'elaborato verrà descritto nel dettaglio l'obiettivo del lavoro di tesi. Partendo dall'implementazione della funzione di *Web Application Firewall* in VEREFOO, lo scopo di questo documento è quello di proporre un meccanismo in grado di perfezionare i requisiti di sicurezza passati in input dall'utente con delle *policies* OWASP, estratte da uno o più file di log prodotti da un *Intrusion Detection System*.

4.1 L'interazione tra WAF e IDS

Entrambe le tecnologie di IDS e WAF sono state sviluppate al fine di proteggere le applicazioni web da attacchi informatici, ma operano su differenti livelli architetturali della rete. Un IDS lavora sul *network layer* e consente il rilevamento di anomalie e la tempestiva segnalazione: monitorando il traffico della rete, questa tecnologia è in grado di rilevare flussi di pacchetti malevoli o insoliti sulla base di specifici pattern. Una volta rilevata una minaccia, il sistema, tramite un opportuno meccanismo di *alerting*, segnalerà all'amministratore di rete la ricezione del traffico anomalo e potenzialmente associato ad un attacco. Un IDS fornisce inoltre un meccanismo di *logging*, mediante il quale vengono memorizzate le informazioni relative ai pacchetti che hanno generato l'*alert* in uno o più file di log. Grazie a questo servizio, l'amministratore potrà analizzare il flusso di pacchetti anomalo dai file di log e prendere le opportune precauzioni per la protezione del sistema. I WAFs invece, come già illustrato nel precedente capitolo 3 nel caso di ModSecurity, operano sull'*application layer*, monitorando e filtrando le richieste HTTP ed offrendo una soluzione mirata per la protezione delle applicazioni web.
Non avendo accesso al payload delle richieste HTTP dirette al web server, un IDS potrebbe non essere in grado di rilevare specifiche tipologie di attacchi ad applicazioni web, in quanto ogni applicazione ha determinate vulnerabilità che possono essere sfruttate costruendo vettori di attacco ad-hoc; tuttavia recenti tecnologie di NIDS sono parzialmente in grado di identificare gli attacchi web tramite opportune regole generiche. Analogamente, lavorando su un livello dell'architettura protocollare differente, un WAF non riuscirebbe a identificare attacchi del *network layer*. Per questo motivo, nelle reti moderne vengono allocate entrambe le funzionalità

come mostrato in figura 4.1, in modo da raggiungere un livello di sicurezza globalmente ottimale:

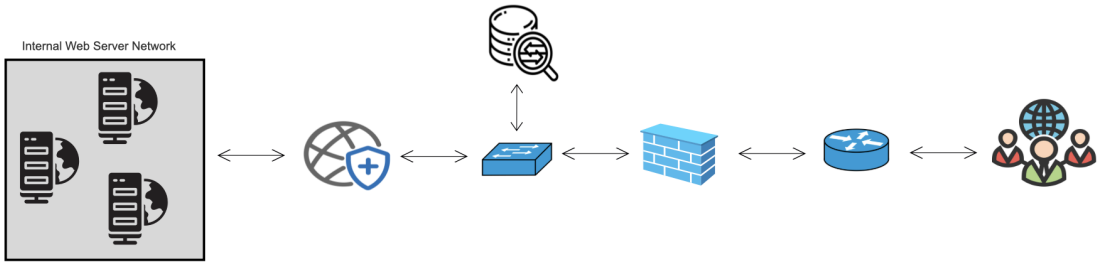

Figura 4.1. Esempio di architettura di rete generica con WAF

4.2 L'estrazione delle OWASP policies dal log IDS

Con questo lavoro di tesi si vuole sfruttare la complementarietà tra i WAFs e gli IDSs, al fine di automatizzare la selezione e l'installazione delle *policies* relative ad attacchi web in ModSec. Attraverso questa nuova funzionalità, un amministratore di rete, ricevuta l'opportuna segnalazione dall'IDS, sarà in grado di sfruttare VEREFOO per l'installazione o l'aggiornamento delle *policies* di sicurezza del WAF tramite i file di log prodotti dall'IDS stesso: a partire dal SG e dai NSRs in input, utilizzando il meccanismo di estrazione delle regole OWASP dai file di log, il framework sarà in grado di integrare il set minimale dei requisiti di sicurezza da rispettare con le regole estratte dai log, calcolare la distribuzione ottima delle *policies* tramite *z3* ed includere i corrispondenti file di configurazione del *Core Rule Set* in ModSecurity. Il raggiungimento dell'obiettivo ha richiesto i seguenti step:

- analisi degli IDS. Durante questa fase, sono stati analizzati i principali *open source* IDS, studiandone la configurazione ed il funzionamento
- indagine sui formati dei log di output per ciascuno degli IDS considerati
- definizione dell'approccio per l'implementazione della funzionalità, sulla base delle analogie riscontrate tra i formati di output dei vari IDS
- implementazione e validazione della soluzione sviluppata mediante l'analisi di esempi pratici

Capitolo 5

Analisi degli Intrusion Detection Systems

Lo studio degli *Intrusion Detection Systems* rappresenta il punto di partenza per la definizione dell'approccio adottato per il meccanismo di *Policy Extraction*. Questa parte dell'elaborato tratta l'analisi degli *open source* IDS in commercio, descrivendone in maniera esaustiva la configurazione, il modello per la definizione delle regole ed i formati dei file di log prodotti in output. L'approfondimento delle caratteristiche appena citate si basa sull'indagine condotta sui rispettivi manuali degli IDS, arricchita con test di configurazione ed esempi esplicativi.

5.1 Snort

5.1.1 Introduzione a Snort

Snort è un sistema di rilevamento e prevenzione delle intrusioni sviluppato da *Sourcefire*, ora parte di *CISCO*, che permette l'analisi del traffico in tempo reale ed il logging del flusso dei pacchetti. A seconda delle necessità, questo strumento può operare in 3 modalità:

- *sniffer mode*, catturando tutti i pacchetti del traffico di rete e mostrandoli in output su schermo; utile per la risoluzione dei problemi di rete
- *packet logger mode*, simile alla precedente; configurandolo in questa modalità, Snort fornisce un servizio di logging del traffico memorizzando il contenuto dei pacchetti all'interno di file di log in diversi formati
- *Network Intrusion Detection System* (NIDS), la più utilizzata per Snort. Analizzando il traffico di rete sulla base di opportune regole e *signatures*, in questa modalità l'IDS è in grado di segnalare all'amministratore di rete il rilevamento di traffico anomalo ed eventualmente, a seconda della disposizione all'interno della rete e della configurazione delle regole, di lavorare come un *Intrusion Prevention System* (IPS) o come WAF - nell'ambito della protezione di applicazioni web - bloccando immediatamente i pacchetti malevoli. Non avendo

accesso completo alle richieste HTTP, caratteristica tipica di un WAF, l'efficacia di Snort in questo contesto è limitata, poichè basata su regole generiche e di conseguenza non esaustive per la mitigazione di attacchi web. Tuttavia come già citato nella sezione 4.1, la soluzione migliore è quella di integrare la prima linea di difesa di Snort, specifica per il *network layer*, con una tecnologia di WAF (ad esempio ModSec), raggiungendo un livello di protezione globalmente migliore.

Il funzionamento di Snort si basa su un file principale denominato *snort.conf*. All'interno di tale file, è possibile specificare una serie di parametri di configurazione dell'IDS, tramite opportune direttive, al fine di selezionare la modalità di utilizzo, definire le regole e gestire i formati degli eventuali log di output prodotti. Le variabili di rete sono di particolare interesse per l'obiettivo della tesi, in quanto rappresentano parte degli attributi da specificare in fase di definizione delle regole da applicare al traffico, che verranno trattate nel paragrafo successivo. Il linguaggio utilizzato nel file *snort.conf* prevede il seguente modello per l'inizializzazione di una variabile:

keyword var_name value

Il campo *value* identifica il valore assegnato alla variabile, mentre *var_name* non è altro che un'etichetta mediante la quale si accede alla stessa. La *keyword* rappresenta invece la direttiva utilizzata per specificare il tipo di variabile e può essere:

- *ipvar*, per l'assegnazione di un indirizzo o un range di indirizzi IP come mostrato nel codice 5.1. Utilizzando questo tipo di variabile è possibile definire un gruppo di indirizzi IP da utilizzare nella definizione di una regola, evitando quindi di dover scrivere nel *rule set* configurato la stessa regola per ogni indirizzo dell'intervallo

```
# Setup the network addresses you are protecting
ipvar HOME_NET 192.168.0.0/24
# Set up the external network addresses. Leave as "any" in most
    situations
ipvar EXTERNAL_NET any
# List of web servers on your network
ipvar HTTP_SERVERS $HOME_NET
```

Codice 5.1. Esempio di utilizzo di *ipvar* all'interno di *snort.conf*

La prima istruzione dell'esempio, mostra la definizione del range di indirizzi IP sfruttando la notazione CIDR della rete 192.168.0.0/24, etichettata come *HOME_NET*. In maniera analoga, la seconda istruzione definisce la variabile *EXTERNAL_NET*, settandola ad *any*, valore utilizzato per indicare che la variabile può assumere qualsiasi valore. Nella terza ed ultima istruzione viene invece mostrato come creare una variabile *HTTP_SERVERS*, assegnandole lo stesso pool di indirizzi IP definito nella prima istruzione.

- *portvar*, per la definizione di una porta o di un pool di porte. Anche in questo caso, per evitare di configurare una regola per ciascuna porta da analizzare in corrispondenza di un pacchetto in input, è possibile utilizzare la sintassi mostrata nel codice 5.2 per la definizione di un range di porte

```
# List of ports you run web servers on
portvar HTTP_PORTS
    [80,81,311,383,591,593,901,1220,1414,1741,1830,2301,2381,2809,3037,
     3128,3702,4343,4848,5250,6988,7000,7001,7144,7145,7510,7777,7779,
     8000,8008,8014,8028,8080,8085,8088,8090,8118,8123,8180,8181,8243,
     8280,8300,8800,8888,8899,9000,9060,9080,9090,9091,9443,9999,11371,
     34443,34444,41080,50002,55555]
# List of ports you want to look for SHELLCODE on.
portvar SHELLCODE_PORTS !80
```

Codice 5.2. Esempio di utilizzo di *portvar* all'interno di *snort.conf*

- *var*, per le istanze di variabili generiche come stringhe, percorsi e numeri. Attraverso la definizione di queste variabili, è possibile ad esempio includere dei file esterni contenenti le regole di analisi del traffico tramite direttiva *include* come in 5.3, seguendo un approccio modulare

```
var RULE_PATH /path/to/rule_set
# Including the rule file using the RULE_PATH variable
include $RULE_PATH/dos.rules
include ..\Snort\rules\local.rules
```

Codice 5.3. Esempio di utilizzo di *var* all'interno di *snort.conf*

5.1.2 Definizione delle regole per la produzione di Alert

Dopo aver analizzato le principali direttive di configurazione delle variabili di rete, in questo paragrafo è illustrato in maniera approfondita il modello adottato per la definizione delle regole.
Snort utilizza un linguaggio semplice, flessibile e potente per la descrizione delle regole. Il manuale ufficiale dell'IDS [10] evidenzia come ciascuna regola sia composta da due principali sezioni logiche, l'intestazione e le opzioni. L'intestazione o *rule header*, contiene informazioni riguardo l'azione da intraprendere, il protocollo, gli indirizzi IP e le porte sorgente e destinazione del pacchetto ricevuto. La sezione delle *rule options* invece contiene i messaggi di avviso e tutte le informazioni necessarie per descrivere quali parti del payload del pacchetto in analisi devono essere ispezionate, affinchè il *detection engine* sia in grado di decidere se l'azione specificata nell'header della regola deve essere intrapresa o meno. Le opzioni possono a loro volta essere classificate in:

- generiche, ovvero quelle opzioni che vengono utilizzate per fornire un contesto aggiuntivo alla regola
- *Payload Detection Rule options*, opzioni specifiche per l'analisi del payload del pacchetto in esame
- *Non-Payload Detection Rule options*, utilizzate per specificare i criteri di valutazione dei dati non appartenenti al payload dei pacchetti
- *Post-Detection Rule options*, per indicare un'azione da intraprendere dopo che una regola è stata attivata

Al fine di descriverne dettagliatamente il modello, è stata estratto un esempio di regola dal *rule set* della community di Snort [11]. Si tratta di una policy di sicurezza configurata al fine di segnalare un possibile *shell code injection attack*, eseguito sfruttando la vulnerabilità CVE-1999-0509 [12]:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SERVER-WEBAPP
    bash access"; flow:to_server,established; content:"/bash";
    fast_pattern:only; http_uri; metadata:policy max-detect-ips drop, ruleset
    community, service http; reference:cve,1999-0509;
    reference:url,www.cert.org/advisories/CA-1996-11.html;
    classtype:web-application-activity; sid:885; rev:21;)
```

Codice 5.4. Esempio di regola estratto dal *rule set* della *Snort Community*

Scendendo nel dettaglio, l'intestazione della regola nel codice 5.4 è composta da:

- *alert*, ovvero l'azione che l'IDS deve intraprendere. In questo caso, Snort produrrà un messaggio di *alert* su console o nel file di log - o in entrambi, dipende dalla configurazione dell'output che affronteremo nella prossima sezione dell'elaborato - contenente un messaggio di segnalazione, nel caso in cui il pacchetto in input farà *match* con la regola
- *tcp*, il protocollo che Snort dovrà analizzare
- *$EXTERNAL_NET*, la variabile di tipo *ip_var* descritta nel paragrafo precedente e configurata nel file *snort.conf*. Indica l'indirizzo o il range di indirizzi IP che rappresentano l'origine del traffico in input
- *any*, segnala che Snort deve ispezionare il pacchetto in ingresso a prescindere dalla porta di inoltro
- —>, l'indicatore per il flusso del traffico. In questo caso specifico, si tratta di traffico di tipo *Inbound*, in quanto i pacchetti provengono dalla *EXTERNAL_NET* in direzione dei server HTTP, definiti mediante la variabile di rete *HTTP_SERVERS*. Dall'analisi del manuale, un aspetto da tenere in considerazione è che Snort non permette la definizione di regole esclusivamente sull'*Outbound Traffic*: per ogni regola sul traffico in uscita deve essercene una sul traffico in ingresso.
- *$HTTP_SERVERS*, il range di indirizzi IP di destinazione precedentemente descritto
- *$HTTP_PORTS*, il set delle porte di destinazione del flusso di pacchetti

La parte delle *rule options* segue l'intestazione della regola ed è indicata all'interno di parentesi tonde. Nonostante le opzioni di tipo generico non siano necessarie, in quanto non condizionano la fase di *detection* da parte di Snort, il manuale dell'IDS suggerisce un ampio utilizzo delle *generic rule options* per aggiungere ulteriori informazioni all'output prodotto, facilitando l'interpretazione degli alerts da parte dell'amministratore di rete. Nell'esempio 5.4, le opzioni generiche più rilevanti sono:

- *msg*, che rappresenta il messaggio contenuto nell'alert prodotto da Snort in caso di corrispondenza tra la regola ed il pacchetto
- *sid*, un identificativo univoco assegnato ad una specifica regola
- *rev*, che specifica il *revision number* di una data regola
- *reference*, per aggiungere informazioni riguardo report di attacchi e vulnerabilità per cui è stata configurata la regola. Nell'esempio, l'opzione è utilizzata sia specificare il riferimento nella *Common Vulnerability and Exposure* (CVE), sia per passare l'url di un report di attacco, effettuato sfruttando la vulnerabilità associata alla regola
- *classtype*, utilizzata per classificare la regola

5.1.3 Classificazione degli Alert prodotti da Snort

L'attributo *classtype* è di fondamentale importanza tra quelli appena descritti, in quanto rappresenta il punto cardine dell'approccio adottato per il conseguimento dell'obiettivo di tesi. Tale opzione viene utilizzata nella definizione delle regole per classificarle in base al tipo di attacco ad esse associato ed assegnare una priorità [13]. I valori attribuibili al campo *classtype* sono enumerati in un file di configurazione di Snort, *classification.config*:

```
# Snort Standard Classification
# Classification Format:
# config classification: "classtype", "classtype_description", "priority"
#
config classification: not-suspicious,Not Suspicious Traffic,3
config classification: unknown,Unknown Traffic,3
config classification: web-application-activity,access to a potentially
    vulnerable web application,2
config classification: bad-unknown,Potentially Bad Traffic, 2
config classification: attempted-recon,Attempted Information Leak,2
config classification: successful-recon-limited,Information Leak,2
config classification: successful-recon-largescale,Large Scale Information
    Leak,2
config classification: attempted-dos,Attempted Denial of Service,2
config classification: successful-dos,Denial of Service,2
config classification: attempted-user,Attempted User Privilege Gain,1
config classification: unsuccessful-user,Unsuccessful User Privilege Gain,1
config classification: successful-user,Successful User Privilege Gain,1
config classification: attempted-admin,Attempted Administrator Privilege
    Gain,1
config classification: successful-admin,Successful Administrator Privilege
    Gain,1
```

Codice 5.5. Contenuto di default del file *classification.conf*

Tramite la direttiva *config classification* è possibile inizializzare una nuova entry nel file, seguendo il formato:

classtype, *classtype_description*, *priority*

dove il primo attributo rappresenta la stessa etichetta utilizzata nell'opzione *classtype* per la definizione delle regole, *priority* è un numero intero utilizzato per la gestione delle priorità ed infine il campo *classtype_description* contiene una breve descrizione della classe di attacco.

L'aspetto interessante da considerare è che le regole e gli *alert* generati condividono la classificazione. In caso di match di un pacchetto malevolo con una regola, l'IDS segnalerà l'accaduto producendo un *alert* in output. Tra le informazioni contenute in tale segnalazione - ad esempio un messaggio di allarme, la data e l'ora del rilevamento, la priorità - ci sarà il campo *Classification*. Questo attributo conterrà la descrizione della classe di attacco, quella definita all'interno del file *classification.config*, associata al *classtype* utilizzato come opzione nella definizione della regola che ha generato l'*alert*. In riferimento alla regola definita nel codice 5.4, mantenendo la configurazione output di default, Snort produrrà un *alert* del tipo:

```
[**] [1:9000001:0] SERVER-WEBAPP bash access [**]
[Classification: access to a potentially vulnerable web application]
    [Priority: 2]
01/04-12:34:49.778290 10.0.0.1:1118 -> 30.0.5.2:0
TCP TTL:64 TOS:0x0 ID:24934 IpLen:20 DgmLen:44
*******F Seq: 0x76F90A04 Ack: 0x63CAAA78 Win: 0x200 TcpLen: 20
```

Codice 5.6. Esempio di *alert* generato da Snort

Analizzando la segnalazione in 5.6

- SERVER-WEBAPP bash access è il messaggio specificato nell'opzione *msg* della regola
- Classification: access to a potentially vulnerable web application è la descrizione definita in *classification.config*, associata alla classe *web-application-activity* utilizzata nella definizione della regola
- Priority: 2 è la priorità specificata nel file di configurazione per il corrispondente *classtype*

5.1.4 Formati dei file di log di Snort

Negli esempi precedenti è stata considerata la configurazione di default di Snort, che utilizza un formato di decodifica ASCII per la stampa di *full alerts*; tuttavia l'IDS fornisce diverse modalità di output per le segnalazioni. Questo paragrafo si concentra sul meccanismo di *logging* degli alerts, aspetto fondamentale per l'interpretazione dei log nella *Policy Extraction*, descrivendo la configurazione di Snort ed i possibili formati di output.
I file di log delle segnalazioni vengono prodotti dall'IDS sfruttando gli *Output Modules*. Si tratta di plugins eseguiti internamente da Snort in seguito ad un alert, permettendo una rappresentazione della segnalazione più flessibile e facilmente interpretabile da parte dell'amministratore di rete. I moduli di output sono attivabili direttamente in *snort.conf*, tramite istruzioni con il seguente formato:

output plugin: log_file_name.ids

All'interno del file di configurazione è possibile specificare più plugins di output da attivare, come mostrato nel codice 5.7, i quali vengono eseguiti in sequenza al verificarsi del rilevamento di un pacchetto malevolo:

```
##################################################
# Step #6: Configure output plugins
# For more information, see Snort Manual, Configuring Snort - Output Modules
##################################################
output alert_full: snortVerboseLog.ids
output alert_fast: snortFastLog.ids
```

Codice 5.7. Attivazione degli *Output Modules* in *snort.conf*

L'utilizzo di questa configurazione di Snort genererà dei file di log relativi ai moduli di output attivati, *alert_full* e *alert_fast*, nella *logging directory* di default.
Il plugin *alert_full* prevede che per ciascuna segnalazione venga memorizzato anche l'header del pacchetto che ha generato l'*alert*. In riferimento all'esempio del codice 5.7, oltre al file *snortVerboseLog.ids*, per ogni IP verranno creati dei file per la raccolta del dump dei pacchetti che hanno scaturito la segnalazione. La creazione di questi file rallenta notevolmente l'esecuzione di Snort, di conseguenza tale metodo di output può essere considerato solo in condizioni di traffico leggero; in ogni caso, il manuale dell'IDS scoraggia fortemente l'utilizzo di questo plugin. Il codice 5.8 è un esempio di file di log prodotto da Snort, in modalità *alert_full*:

```
[**] [1:9000001:0] SERVER-WEBAPP bash access [**]
[Classification: access to a potentially vulnerable web application]
    [Priority: 2]
01/04-12:34:49.778290 10.0.0.1:1118 -> 30.0.5.2:0
TCP TTL:64 TOS:0x0 ID:24934 IpLen:20 DgmLen:44
*******F Seq: 0x76F90A04 Ack: 0x63CAAA78 Win: 0x200 TcpLen: 20

[**] [1:9000001:0] FIN Scan [**]
[Classification: Detection of a Network Scan] [Priority: 3]
01/04-12:34:50.781845 10.0.0.3:1119 -> 30.0.5.2:0
TCP TTL:64 TOS:0x0 ID:51964 IpLen:20 DgmLen:44
*******F Seq: 0x77EF291E Ack: 0x55080E29 Win: 0x200 TcpLen: 20

[**] [1:491:5] FTP Bad login [**]
[Classification: Deprecated protocol usage or protocol attack detected]
    [Priority: 1]
01/04-12:34:51.252082 10.0.0.2 -> 30.0.5.2:18163
TCP TTL:128 TOS:0x0 ID:1743 IpLen:20 DgmLen:82 DF
***AP*** Seq: 0xFACD9194 Ack: 0x4E2AB853 Win: 0xFADA TcpLen: 32
TCP Options (3) => NOP NOP TS: 13957 7457681
```

Codice 5.8. Esempio di log degli alert prodotti da Snort in modalità *full*

Questo formato di output prevede che ciascuna segnalazione sia stampata all'interno del file su più righe, ognuna delle quali contiene rispettivamente le seguenti informazioni: il messaggio di alert, la descrizione della classificazione e la priorità, il timestamp di creazione ed il contenuto dell'header del pacchetto che ha generato la segnalazione.

Attivando il plugin *alert_fast* invece, gli alerts sono memorizzati su un'unica riga all'interno del file di log. Questa modalità di *alerting* è molto più rapida della precedente, in quanto non c'è la necessità di stampare tutti gli headers dei pacchetti e viene prodotto un unico file di log, il cui nome è specificato nell'istruzione di attivazione del plugin.

```
01/04-11:34:49.778290 [**] [1:9000001:0] FIN Scan [**] [Classification:
    Detection of a Network Scan] [Priority: 3] {TCP} 10.0.0.1:1118 ->
    30.0.5.2:0
01/04-11:34:50.781845 [**] [1:9000001:0] FIN Scan [**] [Classification:
    Detection of a Network Scan] [Priority: 3] {TCP} 10.0.0.3:1119 ->
    30.0.5.2:455
01/04-11:34:51.785327 [**] [1:9000001:0] FIN Scan [**] [Classification:
    Detection of a Network Scan] [Priority: 3] {TCP} 10.0.0.1:1118 ->
    30.0.5.2:455
01/04-11:34:52.380494 [**] [1:491:5] FTP Bad login [**] [Classification:
    Deprecated protocol usage or protocol attack detected] [Priority: 1]
    {TCP} 10.0.0.2:21 -> 30.0.5.2:18167
01/04-11:35:03.381792 [**] [1:491:5] FTP Bad login [**] [Classification:
    Deprecated protocol usage or protocol attack detected] [Priority: 1]
    {TCP} 10.0.0.2:21 -> 30.0.5.2:18167
```

Codice 5.9. Esempio di log degli alert prodotti da Snort in modalità *fast*

Nell'esempio 5.9, per ciascuna segnalazione è stata memorizzata una riga nel file di log seguendo il formato:

timestamp [**] [sid] alert msg [**] [classification] [priority] {protocol}
ipSrc:portSrc —>ipDst:portDst

5.2 Suricata

5.2.1 Introduzione a Suricata

Suricata è un sistema di rilevamento e prevenzione delle intrusioni sviluppato dall'organizzazione *Open Information Security Foundation* (OISF). Questa tecnologia è in grado di analizzare il traffico in tempo reale, fornendo funzionalità avanzate per il rilevamento delle minacce - anche le più complesse - e per la protezione da attacchi informatici. L'efficienza e la scalabilità assicurate da questo strumento derivano dal supporto nativo al multithreading: sfruttando l'architettura multi-core dei sistemi, Suricata garantisce migliori prestazioni in reti ad alto flusso di traffico; inoltre il meccanismo di *Deep Packet Inspection* adottato da questo strumento è tendenzialmente migliore del *detection engine* di Snort, in quanto la definizione dei criteri di monitoraggio del traffico - trattata nel successivo paragrafo - può essere estesa mediante scripting.
In maniera analoga a Snort, anche Suricata può lavorare in diverse modalità, progettate per specifiche esigenze nell'ambito della protezione dei sistemi. Le principali sono:

- *Network Security Monitoring.* In questa modalità Suricata registra i flussi di dati in entrata, permettendo l'ispezione del traffico e l'identificazione di

pattern anomali da parte dell'amministratore; simile alla *packet logger mode* di Snort

- come IDS per il monitoraggio passivo del traffico, fornendo meccanismi di *alerting* e *logging* per la segnalazione e la registrazione delle anomalie riscontrate

- come IPS per il blocco dei pacchetti malevoli in ingresso. Se posizionato *inline* rispetto al percorso del traffico, Suricata è in grado di lavorare come IPS e quindi bloccare i pacchetti sospetti tramite monitoraggio attivo del flusso, sulla base di opportune regole e *signatures*. Anche in questa modalità sono erogati i medesimi servizi di segnalazione e *logging*, tipici della modalità IDS

Questo strumento si basa sul file di configurazione *suricata.yaml*, attraverso il quale è possibile specificare tutti i parametri di configurazione necessari riguardo la modalità di utilizzo, le regole per l'analisi del traffico ed i meccanismi di output. Tale file è strutturato in più sezioni, sintetizzate nella seguente lista:

- sezione introduttiva, in cui viene specificato il modello di esecuzione e vengono settati i parametri di *networking*. In questa parte del file, viene selezionata l'interfaccia da analizzare e vengono inizializzate le variabili di rete coinvolte, come mostrato nel codice 5.10

```
##
## Step 1: Inform Suricata about your network
##
vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.1.0/28]"
    EXTERNAL_NET: "!$HOME_NET"
    #EXTERNAL_NET: "any"

    HTTP_SERVERS: "$HOME_NET"
    SMTP_SERVERS: "$HOME_NET"
    SQL_SERVERS: "$HOME_NET"
#...
  port-groups:
    HTTP_PORTS: "80"
    SHELLCODE_PORTS: "!80"
    ORACLE_PORTS: 1521
    SSH_PORTS: 22
#...
```

Codice 5.10. Esempio di configurazione dei parametri di rete in Suricata

È inoltre possibile includere ulteriori file YAML, in modo da definire una configurazione modulare.

- sezione di definizione delle regole. In maniera simile a Snort, è possibile includere dei *rule set* esterni direttamente all'interno del file di configurazione, utilizzando un approccio modulare:

```
## Configure Suricata to load Suricata-Update managed rules.
##
default-rule-path: C:\\Program Files\\Suricata\\rules\\
rule-files:
    - C:\\Program Files\\Suricata\\rules\\custom.rules
```

Codice 5.11. Inclusione del *rule set* locale nel file *suricata.yaml*

- sezione relativa all'output, per la configurazione del meccanismo di *logging* delle segnalazioni. Di particolare interesse ai fini del lavoro di tesi, sono le istruzioni per l'attivazione dei formati di output dei file di log, che verranno analizzati nei successivi paragrafi:

```
outputs:
  # a full alert log containing much information for signature writers
  # or for investigating suspected false positives.
  - alert-debug:
      enabled: yes
      filename: alert-debug.log
      append: yes
      #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'
```

Codice 5.12. Attivazione della *alert-debug mode* in Suricata

5.2.2 Analogie con Snort nella definizione delle regole e nella classificazione degli Alert

Sebbene Snort e Suricata presentino delle sostanziali differenze in termini di funzionalità, architettura e prestazioni, le due tecnologie hanno dei punti in comune di fondamentale importanza ai fini degli obiettivi di tesi:

- la definizione delle regole. Il modello di definizione delle regole adottato da Snort rappresenta di fatto uno standard supportato dalla maggior parte degli IDS in commercio. Analizzando il manuale ufficiale di Suricata [14] è infatti possibile verificare che il modello di definizione dei criteri di analisi del traffico è del tutto analogo a quello descritto nel paragrafo 5.1.2. La caratteristica innovativa di Suricata in questo contesto è che permette di estendere le capacità di rilevamento delle minacce, personalizzando le regole mediante *lua scripting* [15]: questi script sono implementati costruendo funzioni di *match* ad-hoc e vengono passati nella definizione della regola attraverso l'opzione *lua*

- la classificazione degli alert. Il sistema di classificazione delle regole e degli alert generati è congruente a quello di Snort: tra i file di configurazione inclusi in *suricata.yaml* c'è il file *classification.config*, dove sono enumerati i possibili valori dell'opzione *classtype* - utilizzati come opzione nella definizione delle regole

```
# classification.config file content
configuration classification: misc-activity,Misc activity,3


# local.rules file content
alert tcp $HOME_NET 21 -> $EXTERNAL_NET any (msg:"ET POLICY FTP Login
    Successful (non-anonymous)";
    flow:from_server,established;flowbits:isset,ET.ftp.user.login;
    flowbits:isnotset,ftp.user.logged_in;flowbits:set,ftp.user.logged_in;
    content:"230 "; pcre:!"/^230(\s+USER)?\s+(anonymous|ftp)/smi";
    classtype:misc-activity; reference:urldoc.emergingthreats.net/2003410,;
    reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/POLICY_FTP;
    sid:2003410; rev:7;)

# Event/Alert log
10/26/10-10:13:42.904785 [**] [1:2003410:7] ET POLICY FTP Login Successful
    (non-anonymous) [**][Classification: Misc activity[Priority: 3] {TCP}
    192.168.0.109:21 -> x.x.x.x:34117

# Suricata YAML 1.1
...
# Network variable definition
vars:
```

Codice 5.13. Esempio di classificazione, definizione della regola ed alert prodotto in Suricata

5.2.3 Formati dei file di log di Suricata

Le due tecnologie di IDS considerate presentano delle analogie anche per quanto riguarda la configurazione dell'output ed il formato dei log degli alerts. Suricata supporta diversi moduli per il *logging* delle segnalazioni e delle informazioni relative al traffico di rete. Come già accennato nel paragrafo 5.2.1 di introduzione all'IDS, queste modalità permettono di generare log in vari formati, personalizzabili mediante *lua scripting* al fine di facilitarne l'interpretazione e studiati per la compatibilità con altri sistemi di monitoraggio e analisi.
Tra i principali meccanismi di output per le segnalazioni, i più utilizzati sono:

- *alert-debug*. Simile al plugin *alert-full* di Snort, questa modalità viene utilizzata per identificare eventuali falsi positivi e per effettuare debug delle *signatures* configurate. La direttiva di attivazione è specificata nel codice 5.14:

```
    outputs:
    # a full alert log containing much information for signature writers
    # or for investigating suspected false positives.
    - alert-debug:
        enabled: yes
        filename: alert-debug.log
        append: yes
        #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'
    ...
```

Codice 5.14. Configurazione del modulo *alert-debug* in Suricata

Essendo un formato di output molto prolisso, questa modalità rallenta le prestazioni dell'IDS a causa della grande mole di dati da memorizzare

- *Extensible Event Format* (EVE). Si tratta del formato più utilizzato, in quanto consiste in una rappresentazione JSON dell'alert su un'unica linea del file di log:

```
outputs:
    # Extensible Event Format (nicknamed EVE) event log in JSON
        format
    - eve-log:
        enabled: yes
        filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
        filename: eve.json
        # Enable for multi-threaded eve.json output
    ...
# EVE.JSON LOG EXAMPLE
{"timestamp":"2024-03-29T17:24:17.503655+0200","flow_id":
    474332883502324,"in_iface":"\\Device\\
    NPF_{BE187254-6E60-4F3E-A4FD-EBC79D4FF6ED}",
    "event_type":"alert","src_ip":"10.0.0.3","src_port":18071,
    "dest_ip":"30.0.5.2","dest_port":21,"proto":"TCP",
    "pkt_src":"wire/pcap","alert":{"action":"allowed",
    "gid":1,"signature_id":5,"rev":0,
    "signature":"LOCAL DOS SYN packet flood inbound, Potential DOS",
    "category":"Detection of a Denial of Service
        Attack","severity":2},
    "direction":"to_server","flow":{"pkts_toserver":1,
    "pkts_toclient":0, "bytes_toserver":174,"bytes_toclient":0,
    "start":"2024-03-29T17:24:17.503655+0200","src_ip":"10.0.0.3",
    "dest_ip":"30.0.5.2","src_port":18071,"dest_port":21}}
```

Codice 5.15. Attivazione dell'*EVE format* ed esempio di alert prodotto

La versatilità del formato JSON permette l'integrazione con diversi strumenti di analisi e di visualizzazione semplificata degli alerts.

- *fast*. Come specificato nel file *suricata.yaml*, si tratta di un formato di output degli alerts di tipo line-based simile al *fast.log* di Snort:

```
# SURICATA.YAML CONFIGURATION FILE
outputs:
# a line based alerts log similar to Snort's fast.log
- fast:
    enabled: yes
    filename: fast.log
    append: yes
    #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'
...
# FAST.LOG LOG EXAMPLE
03/29/2024-17:24:12.530097 [**] [1:5:0] LOCAL DOS SYN packet flood
    inbound, Potential DOS [**] [Classification: Detection of a
    Denial of Service Attack][Priority: 2] {TCP} 10.0.0.2 ->
    30.0.5.2:8080
...
```

Codice 5.16. Attivazione del modulo *fast* ed esempio di alert prodotto

Questa modalità è la più leggera dal punto di vista computazionale. La semplicità e la leggibilità del formato degli alert prodotti, rendono la *fast mode* la più adatta per la segnalazione tempestiva della anomalie rilevate.

Capitolo 6

Approccio definito per la Policy Extraction

Sulla base delle analogie nella definizione delle regole e nella classificazione degli alerts riscontrate nel precedente capitolo 5, in questa parte dell'elaborato verrà descritto l'approccio utilizzato per la definizione del meccanismo di *OWASP Policy Extraction* implementato in VEREFOO. In particolare verranno presentate l'estensione della classificazione adottata dagli IDS per l'output delle segnalazioni, la logica di mapping degli alerts per l'attivazione dei corrispondenti moduli del *Core Rule Set* e le modifiche al modello ed alla rappresentazione XML delle regole OWASP.

6.1 Aggiornamento della classificazione degli Alerts

Come anzidetto nel capitolo 4, l'obiettivo finale del lavoro di tesi è quello di implementare una funzionalità che estragga dai file di log degli IDS delle *policies* per la protezione da attacchi web, seguendo le direttive del progetto *OWASP Top Ten*. Estrarre una *policy* da una segnalazione dell'IDS, consiste sostanzialmente nell'interpretare una entry del file di log per l'identificazione del tipo di attacco associato all'alert, per poi definire una regola di tipo OWASP - seguendo il modello descritto nella sezione 3.2.1 - per la protezione contro tale attacco. A tal proposito, in virtù dello studio condotto nel precedente capitolo sui formati degli alerts prodotti in output dagli IDS, è stato scelto il campo *Classification* come chiave di lettura delle segnalazioni. Occorre tuttavia fare delle precisazioni in merito.
Nonostante le recenti versioni di Suricata e Snort permettano di identificare traffico malevolo relativo ad attacchi web, si tratta di strumenti ideati per il rilevamento di anomalie al livello di *network* o *transport* layer. Di conseguenza la classificazione di default adottata dagli IDS non è congruente con quella definita nella *OWASP Top Ten*, requisito per l'obiettivo di tesi. In riferimento al codice 5.5, che rappresenta il contenuto del file *classification.conf*, l'unico *classtype* attinente ad attacchi web è:

```
# FORMAT:
# config classification: "classtype", "classtype_description", "priority"
config classification: web-application-activity,access to a potentially
    vulnerable web application,2
```

Codice 6.1. Definizione del *classtype* associato ad attacchi web

Adottando questa configurazione, pacchetti malevoli associati a diversi tipi di attacchi web produrranno alerts in output del tipo:

```
03/11-22:31:00.454871 [**] [1:25101:1] DOS flood denial of service attempt
    [**] [Classification: access to a potentially vulnerable web application]
    [Priority: 2] {TCP} 10.0.0.3:80 -> 30.0.5.2
```

Codice 6.2. Esempio di alert prodotto utilizzando la classificazione di default

aventi lo stesso campo *Classification*, a prescindere dal tipo di attacco. Dato che l'obiettivo finale della funzionalità è quello di attivare una *policy* di protezione ad-hoc contro uno specifico attacco web, è stato necessario aumentare la granularità della classificazione installata sugli IDS.
Per chiarire al meglio il concetto, supponiamo di voler definire una regola per l'identificazione di un *Cross Site Scripting Attack* sfruttando la classificazione di default. L'unico *classtype* associabile ad attacchi web definito in tale classificazione è *web-application-activity*, quindi un possibile esempio di regola potrebbe essere il seguente:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"<script> tag detected";
    flow:to_server,established; content:"<script>"; nocase; http_uri;
    content:"</script>"; nocase; http_uri;
    classtype:web-application-activity; sid:1000001; rev:1;)
```

Codice 6.3. Esempio di regola per il rilevamento di un attacco XSS

Questa *policy* cerca negli URI delle richieste HTTP analizzate le sequenze di apertura e di chiusura dei tag $<script>$, comuni in questa tipologia di attacco. In caso di match di un pacchetto con la regola 6.3, verrà prodotta una segnalazione del tipo:

```
01/04-11:35:03.381792 [**] [1:491:5] <script> tag detected [**]
    [Classification: access to a potentially vulnerable web application]
    [Priority: 1] {TCP} 10.0.0.2:21 -> 30.0.5.2:80
```

Codice 6.4. Esempio di alert prodotto dalla regola 6.3

dove il campo *Classification* conterrà appunto la descrizione configurata per il tipo *web-application-activity*. In questo scenario, a causa della mancanza di informazioni specifiche, la definizione di un criterio che permetta di identificare puntualmente il tipo di attacco web sarebbe troppo difficoltosa: si potrebbe ad esempio attivare la produzione di alert di tipo *verbose* e definire il meccanismo di identificazione sulla base del payload dei pacchetti, ma il grado di complessità aumenterebbe esponenzialmente.

Sfruttando la direttiva *conf classification*, supponiamo ora di inizializzare un nuovo *classtype* nel file di configurazione dell'IDS:

```
config classification: xss-attack, Cross site scripting attack detected, 1
```

Codice 6.5. Definizione del nuovo *classtype* per l'identificazione di XSS

Aggiornando il contenuto del file *classification.conf* come in 6.5, è possibile ridefinire la regola di identificazione per attacchi XSS nel seguente modo:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"<script> tag detected";
    flow:to_server,established; content:"<script>"; nocase; http_uri;
    content:"</script>"; nocase; http_uri; classtype:xss-attack; sid:1000001;
    rev:1;)
```

Codice 6.6. Aggiornamento della regola 6.3 con il *classtype* appena definito

producendo in output alerts del tipo:

```
01/04-11:35:03.381792 [**] [1:491:5] XSS attempt detected [**]
    [Classification: Cross site scripting attack detected] [Priority: 1]
    {TCP} 10.0.0.2:21 -> 30.0.5.2:80
```

Codice 6.7. Esempio di alert prodotto dalla regola 6.6

Configurando un'etichetta di classe - da utilizzare nella definizione delle regole di monitoraggio del traffico - per una specifica tipologia di attacco, la segnalazione prodotta conterrà informazioni di contesto aggiuntive che faciliteranno il processo di interpretazione dell'alert stesso e renderanno di conseguenza più efficiente il meccanismo di *Policy Extraction*.
Sulla base delle precedenti considerazioni, il primo passo è stato dunque quello di aggiornare la classificazione adottata dagli IDS, creando etichette di classe congruenti con la classificazione OWASP ed il Modsec CRS:

```
# CLASSIFICATION
# FORMAT:
# config classification: "classtype", "classtype_description", "priority"
#
# The "classtype" values will be used as options in the rule definitions. In
    this way, the produced alerts can be classified and prioritized,
    according to this configuration.
# When IDS triggers an alert, the related output line in the log file will
    contain a "[Classification: $classtype_description]" field.

config classification: protocol-attack, Deprecated protocol usage or protocol
    attack detected, 1
config classification: multipart-attack, Multipart attack detected, 1
config classification: lfi-attack, LFI detected, 1
config classification: rfi-attack, RFI detected, 1
config classification: code-injection, Malicous code injection detected, 1
config classification: php-injection, PHP malicoius code injection detected, 1
config classification: sql-injection, SQL malicous code injection detected, 1
config classification: xss-attack, Cross site scripting attack detected, 1
config classification: session-fixation-attack, Session fixation attack
    detected, 1
config classification: data-leak, Possible data leaks, 1
config classification: web-application-attack, WA attack detected, 1
```

Codice 6.8. Aggiornamento della classificazione installata in Snort e Suricata

Per semplicità non è stata gestita la priorità associata a ciascuna classe di attacco, settando il valore di default *1*. Tuttavia questo valore è sovrascrivibile direttamente nella definizione delle regole attraverso l'opzione *priority*.

6.2 Strategia di mapping degli Alerts

Come già sottolineato in più occasioni, la scelta di estendere la classificazione installata sugli IDS si basa sulla necessità di identificare un attacco specifico e definire successivamente una regola OWASP puntuale, così da attivare il modulo del Modsec CRS corrispondente al termine dell'esecuzione di VEREFOO. Tuttavia, sebbene la seguente configurazione comporti delle perdite in termini di prestazioni, si potrebbe pensare di utilizzare la classificazione di default degli IDS per la definizione delle regole e la produzione degli alerts per poi attivare, a prescindere dal tipo di attacco, tutti i moduli del CRS. Si tratta della condizione di caso peggiore, in quanto verrebbero attivati tutti i moduli del CRS in ModSec, pur dovendo difendere l'applicazione da un sottoinsieme di possibili attacchi per cui è vulnerabile. Analogamente, si potrebbe configurare una soluzione ibrida in cui vengano definite etichette di classe ad-hoc per l'identificazione di attacchi specifici e *classtypes* generici che raggruppino set di attacchi simili. Per chiarire il concetto, di seguito viene illustrata una generalizzazione della classificazione illustrata nel codice 6.8:

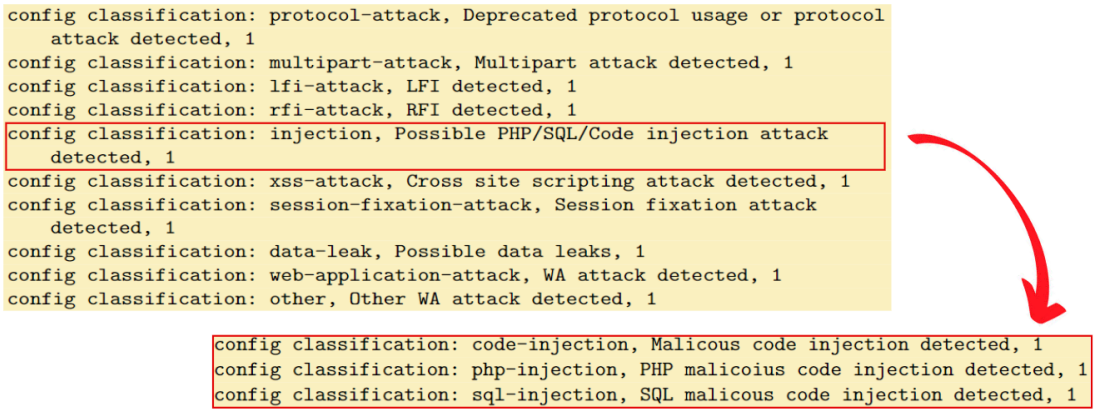

Figura 6.1. Esempio di classificazione ibrida per la definizione delle regole negli IDS

Nella configurazione in figura 6.1, anzichè definire tre etichette di classe distinte per *PHP, Code ed SQL injection*, viene inizializzato un unico *classtype* per raggruppare queste tipologie di attacco. In questo scenario, non potendo discriminare tra le diverse categorie di *injection attacks*, il meccanismo di *Policy Extraction* dovrà estrarre più regole OWASP a fronte di un'unica segnalazione in input.
Per garantire questa flessibilità nella classificazione e tenere traccia dell'eventuale corrispondenza *uno a molti* tra il singolo alert e le regole OWASP, si è optato per l'utilizzo di un nuovo file di configurazione, chiamato *mapper.config*, da passare in input a VEREFOO. Questo file permetterà di mappare ciascuna *classtype_description* definita in *classification.config* con la corrispondente lista di OWASP *policies*, da integrare ai NRSs in input al framework.

Ogni entry inizializzata in questo file segue il formato:

classtype_description —>OWASP_rule1, OWASP_rule2, ... , OWASP_ruleN

Supponendo di essere nello scenario illustrato in figura 6.1, un possibile esempio di configurazione del file *mapper.config* è:

```
# MAPPER CONFIGURATION FILE
##### FORMAT : CLASSTYPE description -> comma separated list of the OWASP
    properties to be activated #####
# $classtype description -> $OWASPprop1, $OWASPprop2, ..... , $OWASPpropn

Deprecated protocol usage or protocol attack detected -> PROTOCOL-ATTACK,
    PROTOCOL-ENFORCEMENT
Multipart attack detected -> MULTIPART-ATTACK
LFI detected -> APPLICATION-ATTACK-LFI
RFI detected -> APPLICATION-ATTACK-RFI
Possible PHP/SQL/Code injection attack detected -> APPLICATION-ATTACK-RCE,
    DATA-LEAKAGES, APPLICATION-ATTACK-PHP, DATA-LEAKAGES-PHP,
    APPLICATION-ATTACK-SQLI, DATA-LEAKAGES-SQL
Cross site scripting attack detected -> APPLICATION-ATTACK-XSS
Session fixation attack detected -> APPLICATION-ATTACK-SESSION-FIXATION
Possible data leaks -> DATA-LEAKAGES
Other WA attack detected -> BLOCKING-EVALUATION, DATA-LEAKAGES-IIS,
    DOS-PROTECTION, SCANNER-DETECTION
WA attack detected -> *
```

Figura 6.2. Esempio configurazione del file di mapping nello scenario in 6.1

Nella figura 6.2, avendo configurato un'unica etichetta di classificazione relativa ad attacchi *PHP, SQL e Code injection*, nel file *mapper.config* è stata inizializzata una singola entry per raggruppare tutte le corrispondenti regole OWASP da selezionare. Attraverso il placeholder *, viene invece specificato che per tutte le segnalazioni aventi *WA attack detected* nel campo *Classification*, verranno selezionate tutte le possibili regole OWASP.

6.3 Modifiche al modello ed alla rappresentazione XML delle regole OWASP

In fase di configurazione del file *mapper.config*, bisogna tenere in considerazione due aspetti fondamentali:

- l'identificativo della regola. Dato che, dopo aver risolto il problema MaxSMT, VEREFOO tradurrà il set minimale di *policies* OWASP da linguaggio di medio a basso livello, è stato scelto, in fase di progetto del modello delle regole relative ad attacchi web, di utilizzare la stessa nomenclatura del CRS. Nello specifico, ciascuna regola OWASP è identificata attraverso il nome del relativo modulo nel CRS, in modo da creare una corrispondenza biunivoca e facilitare il processo di traduzione precedentemente descritto:

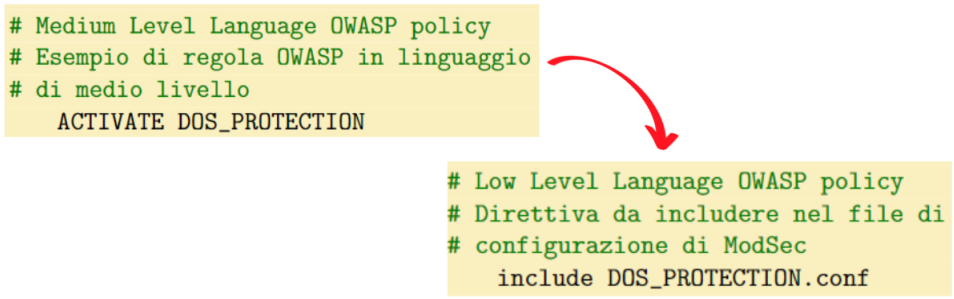

Figura 6.3. Esempio di conversione di regola OWASP da linguaggio di medio ad alto livello

- il livello di protezione offerto dal modulo del CRS selezionato. La costruzione del file *mapper.config* è un processo manuale, di conseguenza suscettibile ad errori da parte dell'amministratore di rete. Per configurare una corretta corrispondenza tra la descrizione dell'etichetta di classe e la lista delle regole OWASP da selezionare, vengono brevemente descritti in appendice - seguendo le indicazioni del sito ufficiale del CRS [16] - i livelli di protezione offerti per ciascun modulo del *Core Rule Set*. Partendo dalla tabella in A.1, sarà possibile selezionare la corretta lista di *OWASPprop* da configurare per ciascun descrittore di classe in *mapper.config*

Tenendo presente che il ModSec CRS è aggiornato periodicamente, la fase successiva del lavoro di tesi è stata quella di modificare il set di possibili valori di una regola OWASP e la corrispondente rappresentazione XML:

```
<xsd:simpleType name="OWASP">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="SCANNER-DETECTION"/>
        <xsd:enumeration value="DOS-PROTECTION"/>
        <xsd:enumeration value="PROTOCOL-ENFORCEMENT"/>
        <xsd:enumeration value="PROTOCOL-ATTACK"/>
        <xsd:enumeration value="MULTIPART-ATTACK"/>
        <xsd:enumeration value="APPLICATION-ATTACK-LFI"/>
        <xsd:enumeration value="APPLICATION-ATTACK-RFI"/>
        <xsd:enumeration value="APPLICATION-ATTACK-RCE"/>
        <xsd:enumeration value="APPLICATION-ATTACK-PHP"/>
        <xsd:enumeration value="APPLICATION-ATTACK-GENERIC"/>
        <xsd:enumeration value="APPLICATION-ATTACK-XSS"/>
        <xsd:enumeration value="APPLICATION-ATTACK-SQLI"/>
        <xsd:enumeration value="APPLICATION-ATTACK-SESSION-FIXATION"/>
        <xsd:enumeration value="APPLICATION-ATTACK-JAVA"/>
        <xsd:enumeration value="BLOCKING-EVALUATION"/>
        <xsd:enumeration value="DATA-LEAKAGES"/>
        <xsd:enumeration value="DATA-LEAKAGES-SQL"/>
        <xsd:enumeration value="DATA-LEAKAGES-JAVA"/>
        <xsd:enumeration value="DATA-LEAKAGES-PHP"/>
        <xsd:enumeration value="DATA-LEAKAGES-IIS"/>
        <xsd:enumeration value="WEB-SHELLS"/>
    </xsd:restriction>
</xsd:simpleType>
```

Codice 6.9. Aggiornamento del set di possibili valori per una regola OWASP

Dall'analisi dei modelli e delle rappresentazioni XML descritti nella sezione 3.2.1 dell'elaborato, sono stati inoltre aggiornati gli schemi xsd degli elementi funzionali, definendo un nuovo tipo complesso *OWASPprop*.
Nell'implementazione di VEREFOO viene utilizzata l'*API AJAX* per la generazione automatica delle classi Java associate agli elementi XML, costituenti il SG in input. Dato che, nella precedente versione del framework, la struttura dell'elemento *OWASP* era esplicitamente definita all'interno dello schema xsd di ogni elemento funzionale - ad esempio in *PropertyDefinition*, *web_application_firewall* e così via - l'API generava, per ciascuno di tali elementi, una sottoclasse di tipo *OWASP* accessibile indirettamente. Ad esempio, considerando il seguente schema xsd relativo alla precedente versione di *PropertyDefinition*:

```
<xsd:element name="PropertyDefinition">
       <xsd:complexType>
              <xsd:sequence>
                     <xsd:element name="Property" type="Property"
                         minOccurs="0" maxOccurs="unbounded"/>
                     <xsd:element name="OWASPprop" minOccurs="0"
                         maxOccurs="unbounded">
                            <xsd:complexType>
                                   <xsd:attribute name="value" type="OWASP"
                                       use="required" />
                                   <xsd:attribute name="graph"
                                       type="xsd:long" use="required"/>
                                   <xsd:attribute name="src"
                                       type="xsd:string" use="required" />
                                   <xsd:attribute name="dst"
                                       type="xsd:string" use="required"/>
                                   <xsd:attribute name="isSat"
                                       type="xsd:boolean"/>
                            </xsd:complexType>
                     </xsd:element>
              </xsd:sequence>
       </xsd:complexType>
</xsd:element>
```

Codice 6.10. Precedente versione di *PropertyDefinition*

AJAX creerebbe all'interno della classe *PropertyDefinition* una sottoclasse *PropertyDefinition.OWASP*, rendendo necessarie - nell'implementazione della funzionalità di *Policy Extraction* - operazioni di mapping aggiuntive per il confronto tra le diverse sottoclassi di tipo *OWASP*. Per questo motivo, è stato creato il seguente *complexType* chiamato *OWASPprop*, estendendo la definizione del *simpleType* nel codice 6.9:

```
<xsd:complexType name="OWASPprop">
   <xsd:simpleContent>
       <xsd:extension base="OWASP">
       <xsd:attribute name="IPsrc" type="xsd:string" use="required"/>
       <xsd:attribute name="IPdst" type="xsd:string" use="required"/>
       </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

Codice 6.11. Rappresentazione dello schema xsd di una *OWASPprop*

Questo nuovo tipo complesso può essere richiamato direttamente nella definizione degli elementi funzionali seguendo un approccio modulare - senza quindi la necessità di definire esplicitamente i singoli attributi che caratterizzano una proprietà OWASP - rendendo univoca la definizione di *OWASPprop*. Ad esempio, aggiornando lo schema 6.10 nel seguente modo:

```
<xsd:element name="PropertyDefinition">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Property" type="Property" minOccurs="1"
    maxOccurs="unbounded"/>
            <xsd:element name="OWASPprop" type="OWASPprop" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Codice 6.12. Aggiornamento definizione *PropertyDefinition*

la libreria AJAX produrrà una classe del tipo:

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "property",
    "owasPprop"
})
@XmlRootElement(name = "PropertyDefinition")
public class PropertyDefinition {

    @XmlElement(name = "Property", required = true)
    protected List<Property> property;
    @XmlElement(name = "OWASPprop")
    protected List<OWASPprop> owasPprop;

    ...
}
```

Codice 6.13. Classe generata automaticamente da AJAX a partire da 6.12

contenente una semplice lista di *OWASPprop*, che risulterà confrontabile con la lista di un altro elemento funzionale.

Capitolo 7

Implementazione e Validazione della Policy Extraction

Dopo aver descritto le strategie per l'interpretazione degli alerts e la selezione delle corrispondenti regole OWASP da attivare, in questa parte dell'elaborato verrà illustrata l'implementazione del meccanismo di *Policy Extraction*. In particolare, verrà inizialmente descritta la classe Java *IdsPolicyExtraction.java*, evidenziando la funzionalità offerta da ogni metodo di tale classe, per poi testare e validare la corretta estrazione delle regole OWASP attraverso esempi applicativi.

7.1 L'implementazione

La classe *IdsPolicyExtraction.java* in A.4 è composta dai seguenti metodi:

- il costruttore della classe
  IdsPolicyExtraction*(String mapperConfigPath)*
  che accetta come parametro il percorso del file di mapping delle regole OWASP. Questo metodo scansiona *mapper.config* per tenere traccia della corrispondenza tra ciascun descrittore di classe - definito in *classification.config* - e la relativa lista di *policies* da attivare tramite una HashMap, seguendo la logica descritta nel successivo paragrafo

- il metodo
  public Boolean ***checkIfOWASP****(String classtype_description)*
  per verificare se è stata definita la corrispondenza tra una certa *classtype_description* ed una lista di regole da selezionare nella HashMap interna

- il metodo
  public List<OWASP> ***getOWASPpropList****(String classtype_description)*
  che ritorna la lista di regole OWASP memorizzata nella HashMap relativa alla chiave *classtype_description*

- il metodo
  private boolean ***networkContains****(String ip, String networkAddress)*

per verificare che un indirizzo IP sia contenuto all'interno di un range di indirizzi

- il metodo

  public List<OWASPprop> ***getFromIdsLog****(...)*

  che rappresenta il cuore del meccanismo di estrazione. Sulla base della struttura dati definita dal costruttore, analizza ogni riga del file di log in input ed estrae, attraverso dei criteri che saranno illustrati nelle sezioni successive, le *OWASPprop* da integrare ai NRS in ingresso a VEREFOO.

7.1.1 Gestione della configurazione in input

Di seguito viene illustrata l'implementazione del costruttore della classe:

```
public IdsPolicyExtraction(String mapperConfigPath) {
    try {
        this.hashMap = new HashMap<String, ArrayList<OWASP>>();
        File mapconFile = new File(mapperConfigPath);
        Scanner fileReader = new Scanner(mapconFile);

        // Scanning mapper.config
        while (fileReader.hasNextLine()) {
            String entryString = fileReader.nextLine();
            if(!entryString.startsWith(regexEntryLog)
            && !entryString.isEmpty()) {
                String[] tokens = entryString.split(regexIPsplit);
                if (tokens.length == 2) {
                    //For each classtype description,
                    // the list of OWASP properties
                    hashMap.put(tokens[0],
                        new ArrayList<OWASP>(Arrays.stream(tokens[1]
                        .split(regexOWASPlist))
                        .map(x -> OWASP.fromValue(x))
                        .collect(Collectors.toList())));
                } else System.err.println("Wrongly configured
                                           OWASP map file\n");
            }
        }
        fileReader.close();
    }  catch (FileNotFoundException e) {
        System.out.println("OWASP map configuration file needed\n");
        e.printStackTrace();
    }
}
```

Codice 7.1. Costruttore della classe implementata

All'interno di tale metodo, viene letto *mapper.config* per inizializzare una HashMap interna e tenere traccia della corrispondenza tra ciascun *classtype_description* e la corrispondente lista di regole OWASP. Richiamando il formato delle righe di tale file, descritto nel capitolo 6:

classtype_description —>OWASP_rule1, OWASP_rule2, ... , OWASP_ruleN

ciascuna riga letta viene divisa in due parti, utilizzando il separatore —>. La prima delle due parti rappresenta il descrittore di classe, la seconda invece l'elenco delle regole OWASP da memorizzare. Ciascuna stringa di questo elenco viene passata a ***fromValue****(String)* - metodo della classe OWASP, generata automaticamente dalla libreria AJAX - per ottenere una lista di oggetti OWASP. Infine il metodo effettua l'operazione di *put* nella HashMap interna, settando il descrittore di classe come chiave e la lista OWASP calcolata come valore.

7.1.2 Funzione di analisi del log ed estrazione delle OWASP policies

Una volta inizializzata la HashMap interna, occorre chiamare ***getFromIdsLog****(...)* per avviare il processo di estrazione. Questo metodo ritorna una lista di *policies* di tipo OWASPprop da sottoporre al solver e richiede una serie di parametri in input:

- *List<OWASPprop> propList*. Si tratta delle lista di *policies* da aggiornare, contente le OWASPprop estratte dai NSRs in input
- *String logPath*, il percorso del file di log prodotto dall'IDS da esaminare
- *List<Node> nodeList*. Dato che a ciascuna *policy* di tipo OWASPprop è associata una coppia di indirizzi IP sorgente e destinazione, occorre verificare se l'alert in analisi si riferisce ad un nodo della rete del SG. Per questo motivo è stato necessario passare la lista dei nodi, opportunamente filtrata nel *VerefooSerializer*, per effettuare il controllo appena descritto.

Dovendo gestire le diverse tipologie di log ampliamente descritte nel capitolo 5, l'analisi degli alerts si basa su una serie di *regular expressions* definite staticamente all'interno della classe: dato che, a seconda del formato attivato, un alert può essere memorizzato nel file di log su una o più righe, l'estrazione dei dati relativi ad una OWASPprop avviene in maniera sequenziale aggiornando opportunamente delle variabili temporanee. In particolare, per ogni entry del log esaminata vengono aggiornati una serie di Matcher basati sulle regex:

```
private final static String regexClassification = "\\[Classification:
        (.*?)\\]|\"category\":\"(.*?)\"|^ALERT CLASS
        \\[(\\d{0,})\\]:(.*?)$";
private final static String ipString = "(\\b(?:(?:25[0-5]|2[0-4][0-9]
        |[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4]
        [0-9]|[01]?[0-9][0-9]?)\\b)";
private final static String suricataJsonIP = "(\"src_ip\":\""+
        ipString+"\",\"src_port\":(.*?),\"dest_ip\":\""+
        ipString+"\",\"dest_port\":(.*?),)";
private final static String snortIP = "("+ipString+":?\\b(\\d{0,20000})
        ?\\b -> "+ipString+":?\\b(\\d{0,20000})?\\b)";
private final static String suricataIPsrc = "^SRC IP:(\\s*)"+ipString,
                            suricataIPdst = "^DST IP:(\\s*)"+ipString;
private final static String regexIP = snortIP+"|"+suricataJsonIP+"|"+
        suricataIPsrc+"|"+suricataIPdst;
```

```
private final static String regexEntryLog = "#",
                            regexIPsplit = " -> ",
                            regexOWASPlist = ",\\s*";
```

Codice 7.2. Regex definite per l'estrazione dei dati

Seguendo questa strategia, l'implementazione del metodo può essere concettualmente suddivisa in 3 macrosezioni per:

- estrarre la descrizione della classe. In questa parte del codice viene sfruttata un'istanza della classe Matcher, chiamata *mClasstype*, per catturare i dati relativi alla classificazione dell'alert tramite la variabile *regexClassification*:

```
        // matching the Classification field
        if (currentClasstype.isEmpty()) {
            mClasstype = classificationPattern
                .matcher(logEntry);
            if (mClasstype.find()) {
                // if it's a (snortLog || suricataFastLog)
                //          -> checkIfOWASP(mClasstype.group(1));
                // else if it's a JSON log
                //          -> checkIfOWASP(mClasstype.group(2))
                // else it's a SuricataDebug
                currentClasstype = mClasstype.group(1) != null ?
                    mClasstype.group(1) :
                        mClasstype.group(2) != null ?
                        mClasstype.group(2) : mClasstype.group(4)
                        .replaceFirst("\\s*", "");
            }
    }
```

Codice 7.3. Estrazione della classificazione

  L'eventuale valore catturato dal Matcher fornisce indicazioni sul formato del campo *Classification* dell'alert e di conseguenza sulla tipologia del file di log. Sulla base del risultato del metodo *mClasstype.find()*, la variabile temporanea *currentClasstype* - utilizzata per memorizzare la descrizione della classificazione estratta dalla entry corrente - viene opportunamente aggiornata in 7.3 attraverso una sequenza di operatori ternari

- ricavare la coppia di indirizzi IP sorgente e destinazione. In maniera del tutto analoga alla precedente sezione, nel codice 7.4 viene aggiornato il Matcher *mIpAddresses* per l'estrazione degli indirizzi IP sorgente e destinazione, relativi ai pacchetti che hanno generato l'alert corrente:

```
    // matching ipSrc and ipDst
    mIpAddresses = ipPattern.matcher(logEntry);
    if (mIpAddresses.find()) {
        if (currentIPsrc.isEmpty()) {
            // Assigning the current ipSrc,
            // according to the log format
            currentIPsrc = mIpAddresses.group(1) != null ?
                mIpAddresses.group(2) : mIpAddresses.group(6) != null ?
                mIpAddresses.group(7) :
                mIpAddresses.group(12);
```

```
        }
        if (currentIPdst.isEmpty()) {
            // Assigning the current ipDst, according
            // to the log format
            currentIPdst = mIpAddresses.group(1)
                != null ? mIpAddresses.group(4) :
                mIpAddresses.group(6) != null ? mIpAddresses.group(9) :
                mIpAddresses.group(13) != null ? mIpAddresses.group(14)
                : "";
        }
    }
```

Codice 7.4. Estrazione degli indirizzi IP

- combinare i precedenti dati per inizializzare una nuova *policy*. In quest'ultima sezione viene costruito un nuovo oggetto OWASPprop e, previo controllo sugli indirizzi IP e sulla presenza di eventuali duplicati nella lista finale da ritornare, viene aggiunto in *propList*, la lista finale ritornata dalla funzione:

```
    if (!currentIPsrc.isEmpty() && !currentIPdst.isEmpty()
        && this.checkIfOWASP(currentClasstype)) {
        String ipSrc = new String(currentIPsrc),
                ipDst = new String(currentIPdst);
        // if the (src, dst) IP addresses identify
        // two Service Graph nodes
        if(containsIp(nodeStrings, ipSrc) &&
            containsIp(nodeStrings, ipDst)) {
            this.getOWASPpropList(currentClasstype)
                .forEach(x -> {
                // if the OWASP property is already
                // stored for (ipSrc, ipDst), skip
                if (propList.stream().filter(p ->
                        p.getValue().equals(x) &&
                        (p.getIPsrc().equals(ipSrc)||
                            p.getIPsrc().equals("*"))
                    && p.getIPdst().equals(ipDst))
                        .count() == 0) {
                    // otherwise, add the new OWASP
                    // prop for the pair (ipSrc, ipDst)
                    OWASPprop o = new OWASPprop();
                    o.setIPsrc(ipSrc);
                    o.setIPdst(ipDst);
                    o.setValue(x);
                    propList.add(o);
                }
            });
        }
        currentIPsrc = ""; currentIPdst = "";
        currentClasstype="";
    }
```

Codice 7.5. Aggiornamento della lista di *OWASPprop*

Dato che all'interno del SG possono essere schematizzati singoli *endpoint*, identificati tramite indirizzo IP univoco, piuttosto che intere porzioni di rete utilizzando opportunamente il placeholder *, è stato necessario implementare

un metodo ad-hoc per la verifica degli indirizzi IP estratti dall'alert:

```
private boolean networkContains(String ip, String networkAddress) {
    if (networkAddress.endsWith("*")) {
        IpAddressMatcher ipMatcher;
        networkAddress =
            networkAddress.endsWith("*.*") ?
            networkAddress.endsWith("*.*.*") ?
            networkAddress.replace("*.*.*", "0.0.0/8") :
            networkAddress.replace("*.*", "0.0/16") :
            networkAddress.replace("*", "0/24");
        ipMatcher = new IpAddressMatcher(networkAddress);
        return ipMatcher.matches(ip);
    }
    return false;
}

private boolean containsIp(List<String> nodes, String ip) {
    for(String n : nodes) {
        if (n.endsWith("*")) {
            // Network Address
            if(networkContains(ip, n)) return true;
        } else {
            if(n.equals(ip)) return true;
        }
    }
    return false;
}
```

Codice 7.6. Controllo sugli indirizzi IP

Il metodo *containsIp(...)* scansiona la lista di IP relativi ai nodi del SG. Se l'IP dell'n-esimo nodo termina con il placeholder * - identificando quindi una porzione di rete - viene chiamata la funzione *networkContains(...)* per verificare se l'IP estratto dall'alert è contenuto nel range di IP associato al nodo corrente. Altrimenti, in caso l'n-esimo IP sia un indirizzo puntuale, si fa il semplice confronto con l'indirizzo estratto dall'alert.
Terminata l'operazione di controllo sugli indirizzi IP, il metodo procede con la verifica della presenza di eventuali duplicati e, in caso negativo, crea una nuova istanza di OWASPprop con i dati estratti per poi aggiungerla alla lista da ritornare.

7.2 Esempi di applicazione e validazione dei risultati

7.2.1 Caso d'uso: Scenario A

A partire da uno schema di rete di esempio, dalla configurazione degli IDS e dalla definizione del file di mapping degli alerts, in questa sezione verrà riprodotto un semplice attacco DOS - all'*application layer* - utilizzando l'applicativo *hping*, per poi testare la funzionalità di *Policy Extraction* e validare i risultati ottenuti.

Descrizione dello scenario e della configurazione adottata

Per simulare l'attacco DOS è stato scelto lo schema di rete in figura 7.1:

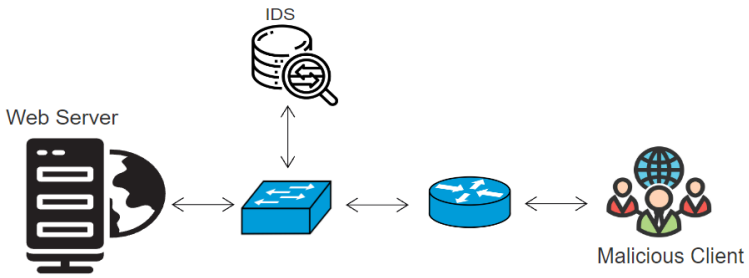

Figura 7.1. Schema di rete dello scenario descritto

Si tratta di un semplice scenario in cui c'è una sottorete che ospita un web server, il cui traffico è monitorato a valle da un IDS, ed un web client. Per replicare tale ambiente di simulazione sono state istanziate due macchine virtuali, VM1 e VM2, raffigurate in 7.2.
La VM1 rappresenta il web server su cui gira l'applicazione web da proteggere; per semplificare l'implementazione, piuttosto che creare e configurare un'ulteriore macchina virtuale, Snort e Suricata sono stati installati direttamente su VM1. La VM2 invece rappresenta il web client, dal quale verrà lanciato l'attacco DOS per la produzione degli *alert logs* in VM1.

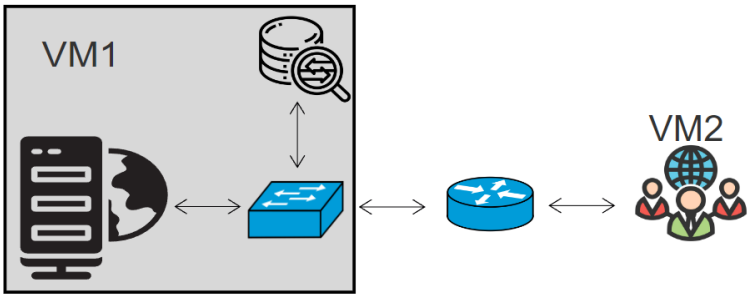

Figura 7.2. Schema logico per lo Scenario A

Nelle sezioni A.5 e A.6 è stato illustrato il processo di configurazione degli IDS sulla macchina virtuale che ospita il web server. Per attivare il monitoraggio del traffico, da terminale - posizionandosi nelle rispettive directory di installazione degli IDS - sono stati lanciati i comandi:

```
# Terminal 1: running Suricata
suricata -c suricata_installation_path/suricata.yaml -i <interface>

# Terminal 2: running Snort
snort -c snort_installation_path/etc/snort.conf -i <interface>
```

Codice 7.7. Avvio monitoraggio del traffico su VM1

Simulazione di un attacco DOS e produzione di *alert logs*

Dopo aver configurato ed avviato gli IDS su VM1, è stato lanciato l'attacco DOS tramite lo script *dos-attack.py*:

```
import os
# Edit target_ip with the web server IP address
target_ip = "30.0.0.5."
os.system("hping3 -c 10000 -d 120 -S -w 64 -p 21 --flood --rand-source "+
    target_ip)
```

Codice 7.8. Script Python lanciato per simulare l'attacco DOS

Il tool *hping3* è un semplice strumento per generare traffico personalizzato da linea di comando. Nel codice in 7.8:

- l'opzione *-c* è utilizzata per specificare il numero di pacchetti da inviare
- tramite *-d* viene specificata la dimensione del payload
- *-S* specifica che saranno inviati messaggi SYN
- *-w* per descrivere la dimensione della finestra TCP
- il campo *-p* specifica la porta
- *–flood* per mandare i pacchetti il più velocemente possibile
- *30.0.0.5* rappresenta invece l'indirizzo IP di destinazione, quello della VM1 su cui gira l'applicazione

Avendo installato delle regole ad-hoc per il rilevamento di questo tipo di attacco, gli IDS genereranno gli alerts nei file di log - seguendo i formati attivati nei rispettivi file di configurazione degli IDS. L'output prodotto dagli IDS è illustrato nella sezione A.7 dell'appendice.

Estrazione delle *policies* e validazione dei risultati

Ricevute le opportune segnalazioni da parte dell'IDS, è possibile finalmente sfruttare la funzionalità di *Policy Extraction* di VEREFOO. A tal proposito, è stata innanzitutto creata la seguente rappresentazione XML:

```
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="../../xsd/nfvSchema.xsd">
       <graphs>
              <graph id="0">
                     <node functional_type="WEBCLIENT" name="10.0.0.1">
                            <neighbour name="30.0.0.5" />
                            <configuration description="VM1-WebClient"
                                   name="confA">
                                   <webclient nameWebServer="30.0.5.2" />
                            </configuration>
```

```
                    </node>
                    <node functional_type="WEBSERVER" name="30.0.0.5">
                        <neighbour name="10.0.0.1" />
                        <configuration description="VM2-WebServer"
                            name="confB">
                            <webserver>
                                <name>30.0.0.5</name>
                            </webserver>
                        </configuration>
                    </node>
                </graph>
        </graphs>
        <Constraints>
                <NodeConstraints />
                <LinkConstraints />
        </Constraints>
        <PropertyDefinition>
                <OWASPprop IPsrc="10.0.0.1"
                    IPdst="30.0.0.5">BLOCKING-EVALUATION</OWASPprop>
                <OWASPprop IPsrc="10.0.0.1"
                    IPdst="30.0.0.5">APPLICATION-ATTACK-XSS</OWASPprop>
        </PropertyDefinition>
        <ParsingString></ParsingString>
</NFV>
```

Codice 7.9. Rappresentazione XML Scenario A

Questo schema rappresenta il SG comprensivo dei NSRs, l'input del framework. All'interno del campo *PropertyDefinition* sono state specificate due OWASPprop da allocare. Per testare la funzionalità di *Policy Extraction* è stata creata una classe di test - illustrata in A.12 - che, a partire dal SG, dal file per il mapping degli alerts e dai file di log, verifica la corretta estrazione delle OWASPprop. Nel caso dello Scenario A considerato, l'output prodotto da tale test è:

```
Testing policy extraction...
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/scenarioAtesi/snortFastLog.ids:     13ms
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/scenarioAtesi/snortVerboseLog.ids:     3ms
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/scenarioAtesi/suricataDebug.ids:     9ms
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/scenarioAtesi/suricataJson.ids:     2ms
************************************************
List of policies:
BLOCKING_EVALUATION, 10.0.0.1, 30.0.0.5
APPLICATION_ATTACK_XSS, 10.0.0.1, 30.0.0.5
DOS_PROTECTION, 10.0.0.1, 30.0.0.5
```

Figura 7.3. Risultato della funzionalità di *Policy Extraction* per lo Scenario A

Avendo definito due NSRs nell'elemento *PropertyDefinition* del SG, la lista di *policies* in input alla funzione di estrazione *getFromIdsLog()* conterrà inizialmente le OWASPprop *BLOCKING_EVALUATION* e *APPLICATION_ATTACK_XSS*. Al termine del processo di estrazione, viene correttamente aggiunta la policy di protezione *DOS_PROTECTION* - estratta dagli alert contenuti nei file di log A.7 - alla lista di OWASPprop in output.
Aggiornando i NSRs nella rappresentazione XML 7.2.1 con il set di OWASPprop appena calcolato per poi sottoporlo a VEREFOO, il framework produrrà un SG in output corrisponde al seguente schema:

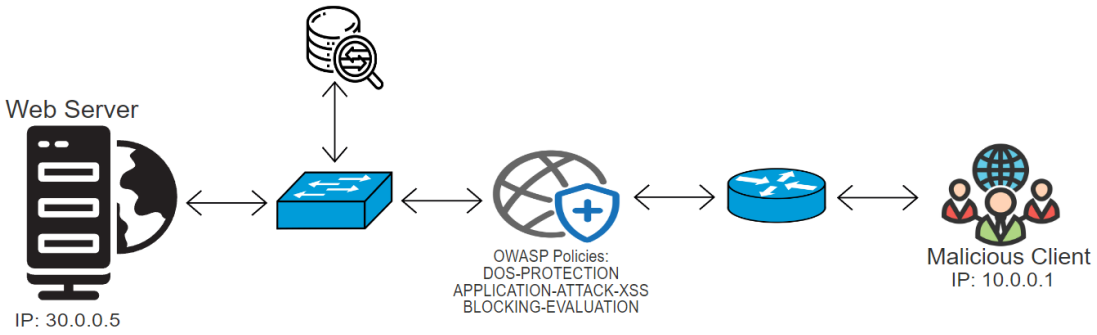

Figura 7.4. Allocazione del WAF ed installazione delle OWASPprop nello Scenario A

Dopo la risoluzione del problema MaxSMT, VEREFOO allocherà un WAF nel SG e calerà il set minimale di OWASPprop, calcolato partendo dai NRSs in input integrati con le *policies* estratte dai log.

7.2.2 Caso d'uso: Scenario B

Sfruttando la medesima classificazione e la stessa configurazione del file di mapping dello Scenario A, in questa sezione verrà testata nuovamente la funzionalità di *Policy Extraction* per verificare i seguenti aspetti:

- l'esclusione di alerts non correlati ai nodi della rete
- l'assenza di duplicati nella lista di OWASPprop selezionate
- la corretta selezione di OWASPprop in caso di presenza di sottoreti nel SG
- la corretta selezione delle OWASPprop installate nelle istanze di WAF del SG

A tal proposito è stato costruito il SG in A.11 che corrisponde al seguente schema di rete:

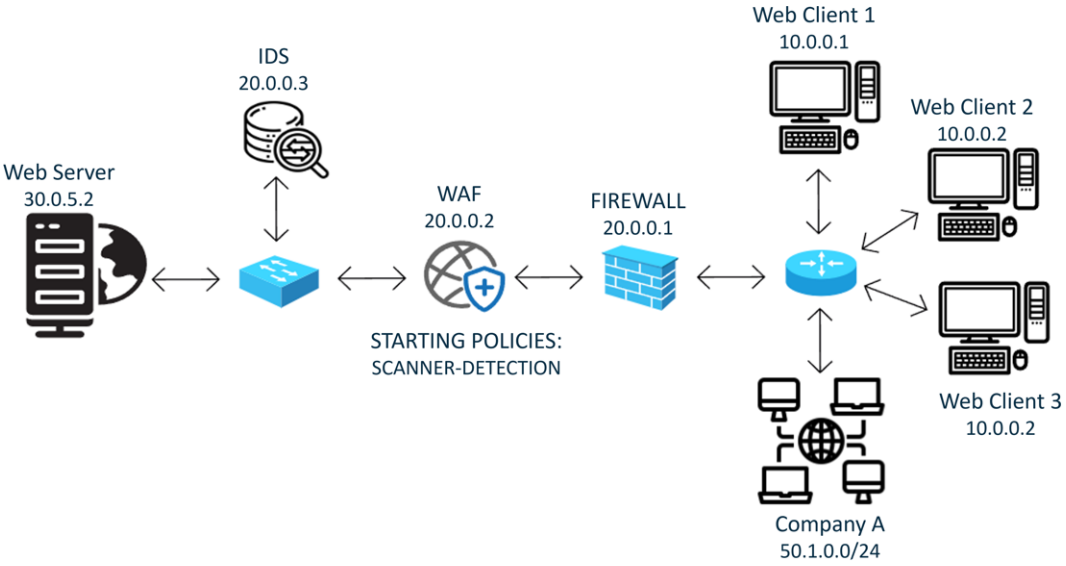

Figura 7.5. Schema di rete Scenario B

In questo scenario il SG contiene un'istanza di *WEB_APPLICATION_FIREWALL*, configurato con l'OWASPprop *SCANNER-DETECTION* per il rilevamento e la protezione contro scanner di rete.
In aggiunta, oltre ai tre semplici client *WEB_CLIENT1* e *WEB_CLIENT2* e *WEB_CLIENT3*, è stata configurata la rete della *Company A* identificata dall'IP *50.1.0.**.
Nella sezione *PropertyDefinition* sono state infine passate le OWASPprop *BLOCKING_EVALUATION* e *APPLICATION_ATTACK_XSS* come NSRs da rispettare.

Estrazione delle *policies* e validazione dei risultati

Esaminando il SG ed i NSRs definiti, è possibile intuire che la lista iniziale delle OWASPprop - calcolata dalla funzione di test implementata - è costituita da:

- *BLOCKING_EVALUATION, 10.0.0.1, 30.0.5.2* (NSRs)
- *APPLICATION_ATTACK_XSS, 10.0.0.2, 30.0.5.2*, (NSRs)
- *SCANNER_DETECTION, *, 30.0.5.2* (configurata nell'istanza di WAF del SG)

Analizzando il log A.14 prodotto dall'IDS, ci si aspetta che la funzione di *Policy Extraction*:

- ignori i primi tre alerts del file di log, poichè è già definita la *policy SCANNER_DETECTION* per ogni IP sorgente
- estragga dal quarto e dal quinto alert le OWASPprop *PROTOCOL_ATTACK* e *PROTOCOL_ENFORCEMENT* per IP sorgente *10.0.0.2*, integrandole poi nella lista da ritornare
- estragga dal sesto e dal settimo alert l'OWASPprop *DOS_PROTECTION* per IP sorgente *10.0.0.1*, *10.0.0.2*
- ignori l'ottavo alert, poichè estrarrebbe un duplicato
- estragga un'unica *policy* di protezione da attacchi DOS dai successivi due alerts, con IP sorgente *10.0.0.3*
- estragga *DOS_PROTECTION* per l'IP 50.1.0.48, appartenente alla *Company A*
- ignori l'ultimo alert, poichè relativo a nodi di rete esterni al SG

Dando il SG A.11 ed il file di mapping A.3 in input alla classe di test A.12 - appositamente implementata per lo Scenario B - si ottiene il seguente output:

```
Testing Scenario B policy extraction...
**************************************************
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/scenarioBtesi/log.ids:     15ms
BLOCKING_EVALUATION, 10.0.0.1, 30.0.5.2
APPLICATION_ATTACK_XSS, 10.0.0.2, 30.0.5.2
SCANNER_DETECTION, *, 30.0.5.2
PROTOCOL_ATTACK, 10.0.0.2, 30.0.5.2
PROTOCOL_ENFORCEMENT, 10.0.0.2, 30.0.5.2
DOS_PROTECTION, 10.0.0.1, 30.0.5.2
DOS_PROTECTION, 10.0.0.2, 30.0.5.2
DOS_PROTECTION, 10.0.0.3, 30.0.5.2
DOS_PROTECTION, 50.1.0.48, 30.0.5.2
```

Figura 7.6. Risultato della funzionalità di *Policy Extraction* per lo Scenario B

perfettamente congruente con la precedente analisi. Integrando i NSRs del SG con le OWASPprop appena estratte e sottoponendo il nuovo SG a VEREFOO, il framework produrrà in output la rappresentazione XML dello schema di rete in figura 7.7, aggiornando le *policies* dell'istanza del WAF con il set minimale di OWASPprop calcolato dal solver:

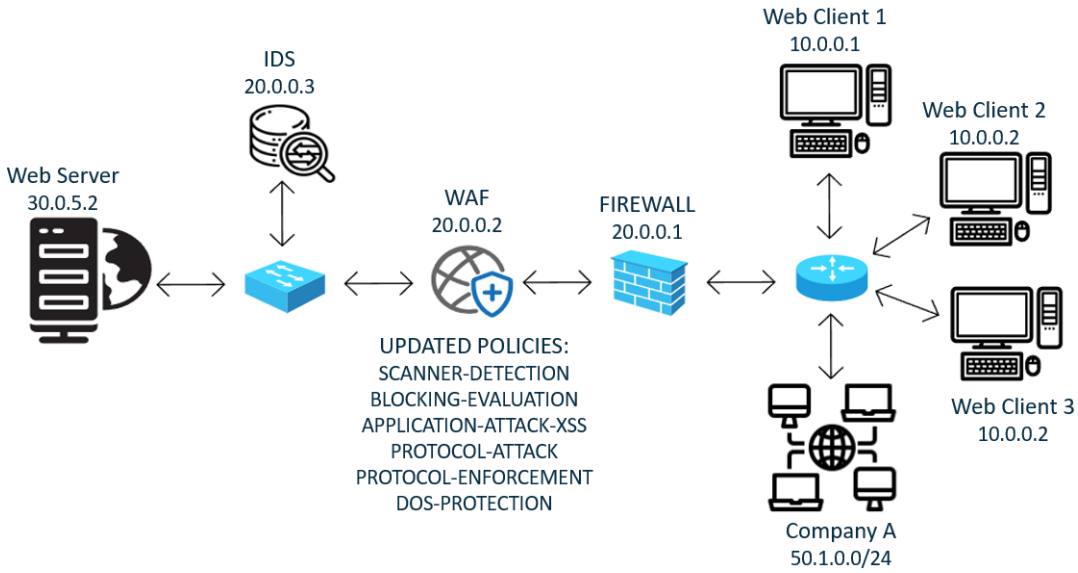

Figura 7.7. Aggiornamento dell'istanza del WAF dello Scenario B

7.3 Valutazione delle prestazioni e dei limiti della soluzione

Come da titolo, in quest'ultimo paragrafo verranno valutate le prestazioni del meccanismo implementato al fine di evidenziarne i limiti. In particolare, verrà stimato il rendimento della *Policy Extraction* in base ai seguenti due criteri:

- granularità della classificazione
- dimensione e formato del log

7.3.1 Rendimento in funzione della classificazione

Nel capitolo 6, è stato descritto come la soluzione implementata sia fortemente dipendente dalla configurazione della classificazione installata sugli IDS, che a sua volta condiziona la realizzazione del file di mapping degli alerts utilizzato dalla funzionalità di estrazione. Negli esempi di applicazione mostrati precedentemente è stata utilizzata una classificazione sufficientemente granulare, in modo da selezionare solo *OWASPprop* relative al tipo di attacco che ha generato l'alert. Diminuendo progressivamente la granularità, si avranno a disposizione sempre meno *classtypes* per la definizione di regole relative ad attacchi web negli IDS, causando un'aggregazione sempre maggiore delle entry configurate nel file *mapper.config*.
Per chiarire il concetto, si consideri la seguente classificazione aggregata:

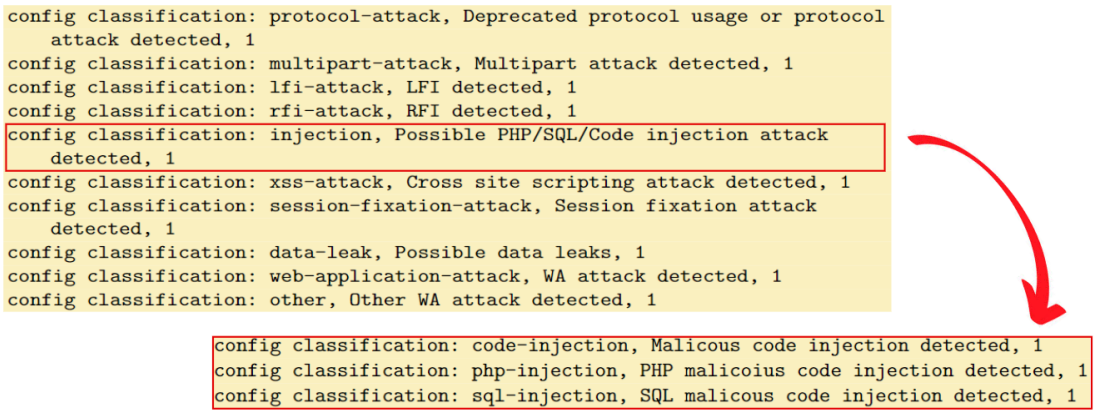

Figura 7.8. Esempio di classificazione aggregata

Avendo creato un'unica etichetta di classe per attacchi di tipo *Injection*, il file di mapping per la configurazione del meccanismo di *Policy Extraction* conterrà la seguente riga:

```
# Mapper.config
Possible PHP/SQL/Code injection attack detected -> APPLICATION_ATTACK-RCE,
    DATA-LEAKAGES, APPLICATION-ATTACK-PHP, DATA-LEAKAGES-PHP,
    APPLICATION-ATTACK-SQLI, DATA-LEAKAGES-SQL
...
```

Codice 7.10. *Mapper.config* definito dalla classificazione 7.8

Dunque in caso di rilevamento ad esempio di un attacco di tipo *SQL Injection*, l'IDS produrrà un alert settando il campo *Classification* con il descrittore di classe - quello relativo all'etichetta *Injection* in figura 7.8 - e la funzionalità di *Policy Extraction* produrrà una lista di OWASPprop estratte del tipo:

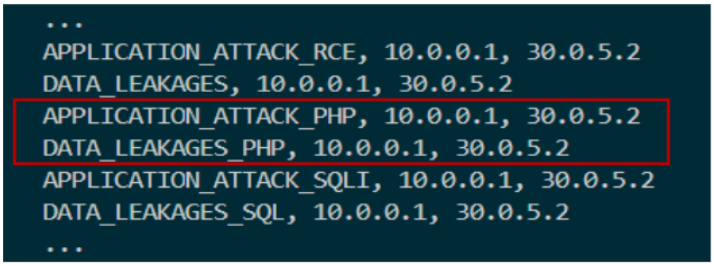

Figura 7.9. Lista di OWASPprop prodotta

Supponendo che l'applicazione sia stata sviluppata seguendo i criteri definiti dal progetto OWASP e non presenti vulnerabilità sfruttabili tramite *PHP Injection*, la selezione e l'eventuale installazione al termine dell'esecuzione di VEREFOO della OWASPprop relativa a questa tipologia di attacco causerebbe un degrado delle prestazioni nel WAF. In questo scenario, installare le OWASPprop *APPLICATION-ATTACK-PHP* e *DATA-LEAKAGES-PHP* nell'istanza del WAF consiste nell'attivare i due moduli del CRS corrispondenti; tenendo presente che all'aumentare dei moduli attivati cresce il carico di lavoro di ModSec, attivando *rule set* superflui si introduce quindi un fattore di degrado nelle prestazioni.
Discorso analogo nel caso opposto: se da una parte viene generato un *overhead* a carico del WAF, dall'altra, l'eccesiva granularità della classificazione, comporterebbe l'incremento del carico di lavoro della funzionalità di *Policy Extraction*. Intensificare il livello di dettaglio della classificazione comporta un aumento delle corrispondenze da definire nel *mapper.config*, che a sua volta implica un maggior consumo di risorse da parte del meccanismo implementato:

```
# Mapper.config
...
Deprecated HTTP protocol usage -> PROTOCOL_ENFORCEMENT
HTTP Request Smuggling detected -> PROTOCOL_ATTACK, PROTOCOL_ENFORCEMENT
HTTP Response Splitting -> PROTOCOL_ATTACK, PROTOCOL_ENFORCEMENT
...
```

Codice 7.11. Incremento delle righe nel *mapper.config*

In particolare aumenterebbe la dimensione della HashMap interna istanziata dalla funzionalità di *Policy Extraction*, utilizzata per tenere traccia delle corrispondenze. Sulla base di questa analisi, per evitare un eccessivo degrado delle prestazioni, occorre definire il giusto *trade-off* nel livello di dettaglio della classificazione degli IDS.

7.3.2 Tempi di esecuzione in funzione del formato e della dimensione del log

Analizzando le tipologie di log offerte da Snort e Suricata nel capitolo 5, è stato evidenziato come sostanzialmente i vari formati differiscano sulla base delle informazioni da memorizzare. Al variare del formato scelto, aumenta la quantità di dati associata a ciascun alert e di conseguenza il numero righe scritte nel file di log per ciascuna segnalazione. Dato che il meccanismo di analisi ed estrazione degli alerts sfrutta una lettura sequenziale del file di log, il formato utilizzato rappresenta dunque un criterio di valutazione delle prestazioni.
Per chiarire il concetto, si considerino i file di log A.7 prodotti nello Scenario A. In seguito al rilevamento di un pacchetto associato all'attacco DOS, per ciascuna segnalazione gli IDS hanno prodotto un numero di righe specifico a seconda del formato del log:

IDS	**Formato del Log**	**# di righe per Alert**
Snort	Fast	1
Snort	AlertFull	5
Suricata	Fast	1
Suricata	Eve Format (JSON)	1
Suricata	Debug	52

Dai dati raccolti in tabella per lo Scenario A, è possibile notare che il file di log prodotto da Snort in *Alert-Full mode* memorizza la singola segnalazione su un numero di righe cinque volte maggiore rispetto ai log di tipo *single line* (*Snort/Suricata Fast, Suricata EVE*). Analogamente, il log prodotto da Suricata in modalità *Debug* stampa ciascun alert con un numero di righe cinquanta volta maggiore. In generale, questi fattori moltiplicativi non sono fissi, ma dipendono dalla dimensione del payload del pacchetto che ha generato l'alert.

Tenendo dunque presente che il numero di righe associate a ciascuna segnalazione è funzione del formato del log e determina le dimensioni complessive del file da analizzare, l'utilizzo di log di tipo *verbose* comporta un incremento esponenziale dei tempi di esecuzione della *Policy Extraction*. Utilizzando la funzione di test A.13 costruita per lo scenario A, è possibile infatti verificare come i tempi di estrazione delle *policies* peggiorino drasticamente per l'analisi dei formati di tipo *verbose*:

```
Testing Snort and Suricata Fast Log policy extraction...
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/speedcompare/fastSnort.ids:     333ms
**************************************************
Testing Suricata Json Log policy extraction...
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/speedcompare/fastSnort.ids:     220ms
**************************************************
Testing Snort Verbose Log policy extraction...
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/speedcompare/verboseSnort.ids:     449ms
**************************************************
Testing Suricata Debug Log policy extraction...
OWASP Policies extraction time from log ./testfile/PolicyExtraction/logs/speedcompare/suricataDebug.ids:     4039ms
```

Figura 7.10. Tempi di estrazione Scenario A (10000 segnalazioni a file)

Il seguente grafico mostra infatti come i tempi di esecuzione (su scala logaritmica) varino significativamente in funzione del numero delle segnalazioni e del formato del log:

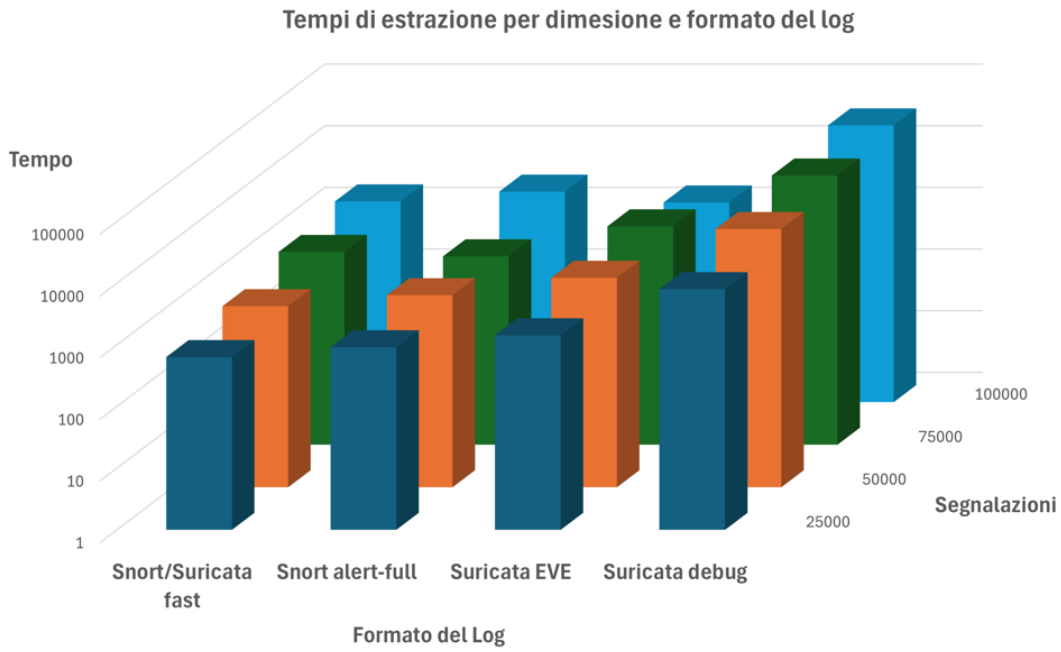

Figura 7.11. Tempi di estrazione al variare del numero di segnalazioni e del formato del log

7.3.3 Limiti della soluzione adottata

L'analisi delle prestazioni condotta nei precedenti paragrafi ha infine evidenziato quelli che rappresentano i limiti della soluzione adottata per l'implementazione. Dal punto di vista dei risultati, in 7.3.1 è stato illustrato come la granularità della classificazione degli IDS condizioni fortemente il rendimento della *Policy Extraction* da una parte, ed il lavoro a carico dell'eventuale istanza di WAF dall'altra. La definizione di una possibile soglia sul livello di granularità da adottare dipende dal contesto e richiede conoscenze approfondite da parte dell'amministratore di rete.
Altro fattore che rappresenta un limite per la soluzione implementata, è l'utilizzo della libreria AJAX. Se da una parte le API di questa libreria permettono la generazione automatica di classi Java associate agli elementi XML del SG, dall'altra limitano le operazioni di confronto tra istanze degli stessi oggetti. Questo aspetto è dovuto al fatto che l'implementazione generata per i metodi di confronto, non utilizzano il valore degli oggetti ma un identificativo univoco come chiave per le operazioni di comparazione. Sebbene questa problematica sembrerebbe facilmente risolvibile implementando l'*override* dei metodi *hashCode()* ed *equals()* per ciascun elemento XML, occorre tener conto del seguente aspetto: ad ogni nuova compilazione del framework, le API generano una nuova implementazione delle classi, sovrascrivendo il codice delle precedenti versioni. Di conseguenza, seppur limitando le prestazioni in termini di tempo di esecuzione, nella soluzione adottata è stato

scelto di utilizzare semplici liste di oggetti come strutture dati di appoggio, in modo da mappare gli oggetti in esame prima dell'operazione di confronto effettiva.

Dal punto di vista della sicurezza, il limite della soluzione adottata risiede nell'approccio manuale utilizzato nel processo di configurazione del meccanismo di estrazione. Dovendo infatti creare un file per mappare ogni descrittore di classe associato ad attacchi web con una lista di OWASPprop, la mancata definizione di una delle corrispondenze comporterebbe un'interpretazione incompleta del log in input.
Si consideri il seguente *mapper.config*, creato a partire dalla classificazione A.2:

```
Detection of a Denial of Service Attack -> DOS-PROTECTION
Deprecated protocol usage or protocol attack detected -> PROTOCOL-ATTACK,
    PROTOCOL-ENFORCEMENT
Multipart attack detected -> MULTIPART-ATTACK
LFI detected -> APPLICATION-ATTACK-LFI
RFI detected -> APPLICATION-ATTACK-RFI
Remote code injection detected -> APPLICATION-ATTACK-RCE, DATA-LEAKAGES
PHP malicious code injection detected -> APPLICATION-ATTACK-PHP,
    DATA-LEAKAGES-PHP
Cross site scripting attack detected -> APPLICATION-ATTACK-XSS
SQL malicious code injection detected -> APPLICATION-ATTACK-SQLI,
    DATA-LEAKAGES-SQL
Session fixation attack detected -> APPLICATION-ATTACK-SESSION-FIXATION
Possible data leaks -> DATA-LEAKAGES
Other WA attack detected -> SCANNER-DETECTION
```

Codice 7.12. Configurazione del *mapper.config* per gli Scenari A e B

Ipotizzando la mancata configurazione della entry:

Detection of a Denial of Service Attack -> DOS-PROTECTION

la HashMap interna della classe ***IdsPolicyExtraction(...)*** non memorizzerà tale corrispondenza. Di conseguenza, a fronte di un alert relativo ad un attacco DOS, il metodo ***getFromIdsLog(...)*** estrarrà correttamente il campo *Classification* dalla segnalazione in esame, per poi scartare l'alert a causa dell'esito negativo dell'operazione di ricerca dell'etichetta di classe nella HashMap (7.5).

Capitolo 8

Conclusioni

Il lavoro di tesi descritto in questo documento ha dunque contribuito all'estensione delle funzionalità fornite dal framework VEREFOO, strumento all'avanguardia nell'Automazione della Sicurezza nelle reti. In particolare, sulla base della naturale coesistenza dei WAF e degli IDS nelle reti moderne, è stata estesa la capacità del framework modellando ed implementando un meccanismo di estrazione che, a partire dai file di log prodotti dagli IDS, è in grado di interpretare le segnalazioni ed estrapolare delle *policies* di protezione da attacchi web, configurate successivamente nelle istanze di WAF della rete.
La definizione dell'approccio alla *Policy Extraction* ha richiesto una fase di studio preliminare di VEREFOO, concentrandosi in particolar modo sulle precedenti scelte progettuali fatte durante lo sviluppo del framework in merito ai WAF. Successivamente, sulla base dell'analisi condotta sui principali *open source* IDS in commercio, sono stati selezionati Snort e Suricata come punti di riferimento. Per entrambe le tecnologie di rilevamento del traffico sono stati evidenziati i principi di funzionamento e di configurazione, ponendo particolare attenzione sul meccanismo di classificazione delle segnalazioni adottato dagli IDS e sui possibili formati delle segnalazioni prodotte nei file di log.
Terminato l'approfondimento sulle tecnologie di IDS, è stata definita una strategia per l'interpretazione delle segnalazioni e l'estrazione dei corrispondenti requisiti di sicurezza basata sui seguenti punti chiave:

- l'aggiornamento della classificazione degli alerts adottata dagli IDS
- la definizione di un modello per mappare la corrispondenza tra un particolare tipo di segnalazione ed un requisito di sicurezza, attraverso un nuovo file di configurazione

A partire dunque dall'aggiornamento della classificazione e della definizione del file di configurazione per il mapping, è stato sviluppato il software. In questa fase è stato integrata l'implementazione del modello dei WAF e delle regole OWASP relative ad una precedente versione del framework, adattandolo alle esigenze dell'obiettivo di tesi, per poi realizzare il meccanismo di estrazione. Questo lavoro presenta una prima versione della funzionalità di *Policy Extraction*, la quale offre possibilità di sviluppo di lavori futuri in termini di ottimizzazione.

A partire infatti dai limiti della soluzione descritti in 7.3.3, è possibile ad esempio estendere la strategia adottata per l'interpretazione delle segnalazioni utilizzando più chiavi di lettura, rendendo ancora più accurato il processo di estrazione.
Dato che la costruzione del file *mapper.config* è basata su un approccio manuale, quindi suscettibile ad errori da parte dell'amministratore, uno spunto per un possibile lavoro futuro riguarda la definizione di un meccanismo in grado di derivare automaticamente il file di mapping a partire dalla classificazione degli IDS, seguendo le direttive e la nomenclatura dell'*OWASP CRS*.
Dal punto di vista del codice invece, si potrebbero implementare dei metodi adhoc per il confronto di istanze delle classi generate automaticamente dalla libreria AJAX: richiamando quanto descritto nel precedente capitolo, i metodi di confronto generati automaticamente dalle API utilizzano un identificativo univoco come chiave di comparazione; questo aspetto rende inconsistenti le operazioni di confronto diretto tra le istanze di tipo OWASPprop, rendendo quindi necessario l'utilizzo di strutture dati di appoggio. Aggiornando dunque la libreria o implementato l'*override* dei metodi *equals()* e *hashCode()* per gli elementi coinvolti, non sarebbero più necessarie operazioni di mapping intermedie e si potrebbero utilizzare strutture dati più performanti - dal punto di visto delle operazioni di ricerca e confronto - delle liste attualmente adottate.

Bibliografia

[1] C. D. Hylender, P. Langlois, A. Pinto, and S. Widup, "Data breach investigation report," Verizon, Tech. Rep., 2024. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir

[2] D. Bringhenti, R. Sisto, and F. Valenza, "Towards security automation in virtual networks," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 326–331.

[3] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Towards a fully automated and optimized network security functions orchestration," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–7.

[4] H. Abdelgader Eissa, K. A. Bozed, and H. Younis, "Software defined networking," in *2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2019, pp. 620–625.

[5] E. Amiri, E. Alizadeh, and K. Raeisi, "An efficient hierarchical distributed sdn controller model," in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, 2019, pp. 553–557.

[6] G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "A framework for verification-oriented user-friendly network function modeling," *IEEE Access*, vol. 7, pp. 99 349–99 359, 2019.

[7] O. W. A. S. Project, "Owasp modsecurity core rule set." [Online]. Available: https://owasp.org/www-project-modsecurity-core-rule-set/

[8] ——, "Owasp top ten." [Online]. Available: https://owasp.org/www-project-top-ten/

[9] ——, "Modsec core rule set official repository." [Online]. Available: https://github.com/coreruleset/coreruleset

[10] T. S. Team, "Snort 2.9.16 manual." [Online]. Available: https://www.snort.org/documents#OfficialDocumentation

[11] "Snort community-rules." [Online]. Available: https://www.snort.org/downloads#rules

[12] NIST, "Cve-1999-0509 detail." [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-1999-0509

[13] T. S. Team, "classtype option." [Online]. Available: https://docs.snort.org/rules/options/general/classtype

[14] OISF, "Suricata rule format." [Online]. Available: https://docs.suricata.io/en/latest/rules/intro.html#

[15] ——, "Lua scripting for detection." [Online]. Available: https://docs.suricata.io/en/latest/rules/lua-detection.html#lua-detection

[16] O. W. A. S. Project, "Core rule set description." [Online]. Available: https://coreruleset.org/docs/rules/rules/

Appendice

Appendice A

A.1 Tabella di descrizione dei moduli del *Core Rule Set*

Modulo CRS	**Descrizione**
DOS-PROTECTION.conf	Per rilevare e bloccare attacchi DOS di livello 7 contro il server
SCANNER-DETECTION.conf	Set di regole per il rilevamento di scanner della rete
PROTOCOL-ENFORCEMENT.conf	Per il blocco di richieste HTTP malformate o associate a vecchie versioni del protocollo
PROTOCOL-ATTACK.conf	Per la protezione da attacchi specifici contro il protocollo HTTP, ad esempio *HTTP Request Smuggling* e *HTTP Response Splitting*
APPLICATION-ATTACK-LFI.conf	Per la prevenzione di attacchi di tipo *Local File Inclusion*: indurre l'applicazione ad eseguire o esporre file del server, rivelando informazioni riservate ed aprendo le porte ad attacchi XSS e *remote code execution*
APPLICATION-ATTACK-RFI.conf	Per la prevenzione di attacchi di tipo *Remote File Inclusion*: indurre l'applicazione ad eseguire risorse remote, compromettendo il server

APPLICATION-ATTACK-RCE.conf	Regole di sicurezza per la protezione contro attacchi di *Remote Command execution*: indurre l'applicazione ad eseguire singoli comandi o interi script attraverso shell Unix o Powershell del server
APPLICATION-ATTACK-PHP.conf	Per la gestione di attacchi *PHP injection*: indurre l'applicazione ad eseguire script PHP malevoli
APPLICATION-ATTACK-XSS.conf	Regole per la protezione contro attacchi XSS
APPLICATION-ATTACK-SQLI.conf	File di configurazione per la prevenzione di *SQL injection attacks*: aggirare un *query interpreter* mal configurato per l'esecuzione di query SQL malevole
APPLICATION-ATTACK-SESSION-FIXATION.conf	Contenente le *signatures* relative a attacchi di tipo *Session Fixation*: instaurare una sessione utente valida con l'applicazione, sfruttando vulnerabilità nella gestione del meccanismo di autenticazione
APPLICATION-ATTACK-JAVA.conf	Regole di protezione contro attacchi Java e Javascript
DATA-LEAKAGES.conf	Per offrire protezione contro fughe di dati generiche
DATA-LEAKAGES-SQL.conf	Per prevenire fughe di dati da server SQL di backend, generalmente combinato con il modulo APPLICATION-ATTACK-SQLI.conf
DATA-LEAKAGES-JAVA.conf	Analogamente al caso precedente, è utilizzato per proteggere il server da fughe di dati dovute alla presenza di vulnerabilità nel codice JAVA dell'applicazione
DATA-LEAKAGES-PHP.conf	Prevenzione di fughe di dati dovute a attacchi *PHP injection*; solitamente combinato con il modulo APPLICATION-ATTACK-PHP.conf

DATA-LEAKAGES-IIS.conf	Per la protezione contro gli attacchi che sfruttano le vulnerabilità del *Microsoft IIS*
APPLICATION-ATTACK-GENERIC.conf	Set di regole per la prevenzione di altri tipi di attacchi, non trattati nei precedenti moduli

A.2 Classificazione degli alert adottata in 7.2

```
# IDS classification.config
# config classification:shortname,short description,priority
#

config classification: not-suspicious,Not Suspicious Traffic,3
config classification: unknown,Unknown Traffic,3
config classification: bad-unknown,Potentially Bad Traffic, 2
config classification: attempted-recon,Attempted Information Leak,2
config classification: successful-recon-limited,Information Leak,2
config classification: successful-recon-largescale,Large Scale Information
    Leak,2
config classification: attempted-dos,Attempted Denial of Service,2
config classification: successful-dos,Denial of Service,2
config classification: attempted-user,Attempted User Privilege Gain,1
config classification: unsuccessful-user,Unsuccessful User Privilege Gain,1
config classification: successful-user,Successful User Privilege Gain,1
config classification: attempted-admin,Attempted Administrator Privilege
    Gain,1
config classification: successful-admin,Successful Administrator Privilege
    Gain,1

# NEW CLASSIFICATIONS
config classification: rpc-portmap-decode,Decode of an RPC Query,2
config classification: shellcode-detect,Executable code was detected,1
config classification: string-detect,A suspicious string was detected,3
config classification: suspicious-filename-detect,A suspicious filename was
    detected,2
config classification: suspicious-login,An attempted login using a suspicious
    username was detected,2
config classification: system-call-detect,A system call was detected,2
config classification: tcp-connection,A TCP connection was detected,4
config classification: trojan-activity,A Network Trojan was detected, 1
config classification: unusual-client-port-connection,A client was using an
    unusual port,2
config classification: network-scan,Detection of a Network Scan,3
config classification: denial-of-service,Detection of a Denial of Service
    Attack,2
config classification: non-standard-protocol,Detection of a non-standard
    protocol or event,2
config classification: protocol-command-decode,Generic Protocol Command
    Decode,3
```

```
config classification: web-application-activity,access to a potentially
    vulnerable web application,2
config classification: web-application-attack,WA attack detected,1
config classification: misc-activity,Misc activity,3
config classification: misc-attack,Misc Attack,2
config classification: icmp-event,Generic ICMP event,3
config classification: inappropriate-content,Inappropriate Content was
    Detected,1
config classification: policy-violation,Potential Corporate Privacy
    Violation,1
config classification: default-login-attempt,Attempt to login by a default
    username and password,2

# Update
config classification: targeted-activity,Targeted Malicious Activity was
    Detected,1
config classification: exploit-kit,Exploit Kit Activity Detected,1
config classification: external-ip-check,Device Retrieving External IP
    Address Detected,2
config classification: domain-c2,Domain Observed Used for C2 Detected,1
config classification: pup-activity,Possibly Unwanted Program Detected,2
config classification: credential-theft,Successful Credential Theft Detected,1
config classification: social-engineering,Possible Social Engineering
    Attempted,2
config classification: coin-mining,Crypto Currency Mining Activity Detected,2
config classification: command-and-control,Malware Command and Control
    Activity Detected,1

# Custom
config classification: protocol-attack, Deprecated protocol usage or protocol
    attack detected, 1
config classification: multipart-attack, Multipart attack detected, 1
config classification: lfi-attack, LFI detected, 1
config classification: rfi-attack, RFI detected, 1
config classification: code-injection, Remote code injection detected, 1
config classification: php-injection, PHP malicious code injection detected, 1
config classification: sql-injection, SQL malicious code injection detected, 1
config classification: xss-attack, Cross site scripting attack detected, 1
config classification: session-fixation-attack, Session fixation attack
    detected, 1
config classification: data-leak, Possible data leaks, 1
config classification: other, Other WA attack detected, 1
```

Codice A.1. Classificazione aggiornata installata sugli IDS

A.3 Configurazione del file di mapping

```
# MAPPER CONFIGURATION FILE

# This file contains, for each 'classtype description' field configured in
    "snort_installation_path/etc/classification.conf", the corresponding OWASP
# properties to be activated. The efficiency of the mapper is strongly
    related to the classification.conf file configuration:
```

```
# the more the classification is fine graded, the more the Mapper will be
    able to properly select the OWASP policies.

##### FORMAT : CLASSTYPE description -> comma separated list of the OWASP
    properties to be activated #####
# $classtype description -> $OWASPprop1, $OWASPprop2, ..... , $OWASPpropn

# Check the OWASPprop.java class file for the OWASP property possible values.

Detection of a Denial of Service Attack -> DOS-PROTECTION
Deprecated protocol usage or protocol attack detected -> PROTOCOL-ATTACK,
    PROTOCOL-ENFORCEMENT
Multipart attack detected -> MULTIPART-ATTACK
LFI detected -> APPLICATION-ATTACK-LFI
RFI detected -> APPLICATION-ATTACK-RFI
Remote code injection detected -> APPLICATION-ATTACK-RCE, DATA-LEAKAGES
PHP malicious code injection detected -> APPLICATION-ATTACK-PHP,
    DATA-LEAKAGES-PHP
Cross site scripting attack detected -> APPLICATION-ATTACK-XSS
SQL malicious code injection detected -> APPLICATION-ATTACK-SQLI,
    DATA-LEAKAGES-SQL
Session fixation attack detected -> APPLICATION-ATTACK-SESSION-FIXATION
Possible data leaks -> DATA-LEAKAGES
Other WA attack detected -> SCANNER-DETECTION

# Usefull links:
# https://coreruleset.org/docs/rules/rules/ : OWASP Core-Rule set description
# https://github.com/coreruleset/coreruleset/tree/main/rules : ModSec OWASP
    Core-Rule set
```

Codice A.2. Configurazione adottata del file *mapper.config*

A.4 Classe *IdsPolicyExtraction.java*

```
package it.polito.verefoo.functions;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Collectors;

import org.springframework.security.web.util.matcher.IpAddressMatcher;

import it.polito.verefoo.jaxb.Node;
import it.polito.verefoo.jaxb.OWASPprop;
import it.polito.verefoo.jaxb.OWASP;
```

```
public class IdsPolicyExtraction {
    private HashMap<String, ArrayList<OWASP>> hashMap;
    private final static String regexClassification = "\\[Classification:
            (.*?)\\]|\"category\":\"(.*?)\"|^ALERT CLASS
            \\[(\\d{0,})\\]:(.*?)$";
    private final static String ipString = "(\\b(?:(?:25[0-5]|2[0-4][0-9]
            |[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4]
            [0-9]|[01]?[0-9][0-9]?)\\b)";
    private final static String suricataJsonIP = "(\"src_ip\":\""+
            ipString+"\",\"src_port\":(.*?),\"dest_ip\":\""+
            ipString+"\",\"dest_port\":(.*?),)";
    private final static String snortIP = "("+ipString+":?\\b(\\d{0,20000})
            ?\\b -> "+ipString+":?\\b(\\d{0,20000})?\\b)";
    private final static String suricataIPsrc = "^SRC IP:(\\s*)"+ipString,
                                suricataIPdst = "^DST IP:(\\s*)"+ipString;
    private final static String regexIP = snortIP+"|"+suricataJsonIP+"|"+
            suricataIPsrc+"|"+suricataIPdst;

    private final static String regexEntryLog = "#",
                                regexIPsplit = " -> ",
                                regexOWASPlist = ",\\s*";

    public IdsPolicyExtraction(String mapperConfigPath) {
        try {
            this.hashMap = new HashMap<String, ArrayList<OWASP>>();
            File mapconFile = new File(mapperConfigPath);
            Scanner fileReader = new Scanner(mapconFile);

            // Scanning mapper.config
            while (fileReader.hasNextLine()) {
                String entryString = fileReader.nextLine();
                if(!entryString.startsWith(regexEntryLog)
                && !entryString.isEmpty()) {
                    String[] tokens = entryString.split(regexIPsplit);
                    if (tokens.length == 2) {
                        //For each classtype description,
                        // the list of OWASP properties
                        hashMap.put(tokens[0],
                            new ArrayList<OWASP>(Arrays.stream(tokens[1]
                            .split(regexOWASPlist))
                            .map(x -> OWASP.fromValue(x))
                            .collect(Collectors.toList())));
                    } else System.err.println("Wrongly configured
                                                OWASP map file\n");
                }
            }
            fileReader.close();
        }  catch (FileNotFoundException e) {
            System.out.println("OWASP map configuration file needed\n");
            e.printStackTrace();
        }
    }

    public Boolean checkIfOWASP(String classtype) {
        return hashMap.keySet().contains(classtype) ? true : false;
    }
```

```
public List<OWASP> getOWASPpropList(String classtype) {
    return hashMap.keySet().contains(classtype) ? hashMap.get(classtype) :
 null;
}

private boolean networkContains(String ip, String networkAddress) {
    if (networkAddress.endsWith("*")) {
        IpAddressMatcher ipMatcher;
        networkAddress =
            networkAddress.endsWith("*.*") ?
            networkAddress.endsWith("*.*.*") ?
            networkAddress.replace("*.*.*", "0.0.0/8") :
            networkAddress.replace("*.*", "0.0/16") :
            networkAddress.replace("*", "0/24");
        ipMatcher = new IpAddressMatcher(networkAddress);
        return ipMatcher.matches(ip);
    }
    return false;
}

private boolean containsIp(List<String> nodes, String ip) {
    for(String n : nodes) {
        if (n.endsWith("*")) {
            // Network Address
            if(networkContains(ip, n)) return true;
        } else {
            if(n.equals(ip)) return true;
        }
    }
    return false;
}

public List<OWASPprop> getFromIdsLog(List<OWASPprop> propList,
       List<Node> nodeList, String logPath) {
try {
    File log = new File(logPath);
    Scanner fileReader = new Scanner(log);
    Pattern classificationPattern = Pattern
                    .compile(regexClassification);
    Pattern ipPattern = Pattern.compile(regexIP);
    Matcher mClasstype, mIpAddresses;
    List<String> nodeStrings = nodeList.stream()
                .map(n -> n.getName())
                .collect(Collectors.toList());
    String currentClasstype = new String(),
            currentIPsrc = new String(),
            currentIPdst = new String();

    while(fileReader.hasNextLine()) {
        String logEntry = fileReader.nextLine();

    // Getting OWASPprop from log entry string:
    // matching the Classification field
    if (currentClasstype.isEmpty()) {
        mClasstype = classificationPattern
            .matcher(logEntry);
```

```
                .matcher(logEntry);
            if (mClasstype.find()) {
                // if it's a (snortLog || suricataFastLog)
                //          -> checkIfOWASP(mClasstype.group(1));
                // else if it's a JSON log
                //          -> checkIfOWASP(mClasstype.group(2))
                // else it's a SuricataDebug
                currentClasstype = mClasstype.group(1) != null ?
                    mClasstype.group(1) :
                        mClasstype.group(2) != null ?
                        mClasstype.group(2) : mClasstype.group(4)
                        .replaceFirst("\\s*", "");
            }
    }

    // matching ipSrc and ipDst
    mIpAddresses = ipPattern.matcher(logEntry);
    if (mIpAddresses.find()) {
        if (currentIPsrc.isEmpty()) {
            // Assigning the current ipSrc,
            // according to the log format
            currentIPsrc = mIpAddresses.group(1) != null ?
                mIpAddresses.group(2) : mIpAddresses.group(6) != null ?
                mIpAddresses.group(7) :
                mIpAddresses.group(12);
        }
        if (currentIPdst.isEmpty()) {
            // Assigning the current ipDst, according
            // to the log format
            currentIPdst = mIpAddresses.group(1)
                != null ? mIpAddresses.group(4) :
                mIpAddresses.group(6) != null ? mIpAddresses.group(9) :
                mIpAddresses.group(13) != null ? mIpAddresses.group(14)
                : "";
        }
    }

    if (!currentIPsrc.isEmpty() && !currentIPdst.isEmpty()
        && this.checkIfOWASP(currentClasstype)) {
        String ipSrc = new String(currentIPsrc),
                ipDst = new String(currentIPdst);
        // if the (src, dst) IP addresses identify
        // two Service Graph nodes
        if(containsIp(nodeStrings, ipSrc) &&
            containsIp(nodeStrings, ipDst)) {
            this.getOWASPpropList(currentClasstype)
                .forEach(x -> {
                // if the OWASP property is already
                // stored for (ipSrc, ipDst), skip
                if (propList.stream().filter(p ->
                        p.getValue().equals(x) &&
                        (p.getIPsrc().equals(ipSrc)||
                            p.getIPsrc().equals("*"))
                    && p.getIPdst().equals(ipDst))
                        .count() == 0) {
                    // otherwise, add the new OWASP
```

```
                    // prop for the pair (ipSrc, ipDst)
                    OWASPprop o = new OWASPprop();
                    o.setIPsrc(ipSrc);
                    o.setIPdst(ipDst);
                    o.setValue(x);
                    propList.add(o);
                }
            });
        }
        currentIPsrc = ""; currentIPdst = "";
        currentClasstype="";
    }
            }
            fileReader.close();
            return propList;
        } catch (FileNotFoundException e) {
            System.out.println("Error opening the IDS log file\n");
            e.printStackTrace();
        }
        return null;
    }
}
```

Codice A.3. *IdsPolicyExtraction.java*

A.5 Configurazione di Suricata

Per la replica dello scenario 7.2.1, occore configurare Suricata seguendo questi passi:

- modificare il file *suricata_installation_path/suricata.yaml*. Innanzitutto bisogna configurare le variabili di rete, definire la directory di salvataggio dei file di log, attivare una o più modalità di output ed infine includere mediante opportuna direttiva i file contenenti le regole. Di seguito è illustrato il contenuto del file di configurazione in riferimento allo schemo di rete in figura 7.1:

```
# Suricata YAML 1.1
...
# Network variable definition
vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.1.0/28]"
    EXTERNAL_NET: "!$HOME_NET"
...
# The default logging directory. Any log or output file will be
# placed here if it's not specified with a full path name. This can be
# overridden with the -l command line parameter.
default-log-dir: C:\\Program Files\\Suricata\\log
...
# Configure the type of alert (and other) logging you would like.
outputs:
  # a full alert log containing much information for signature writers
```

```
  # or for investigating suspected false positives.
  - alert-debug:
      enabled: yes
      filename: alert-debug.log
      append: yes
      #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'
  # a line based alerts log similar to Snort's fast.log
  - fast:
      enabled: yes
      filename: fast.log
      append: yes
      #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'

  # Extensible Event Format (nicknamed EVE) event log in JSON format
  - eve-log:
      enabled: yes
      filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
      filename: eve.json
...
```

Codice A.4. Suricata.yaml

- installare la *classification.config* aggiornata A.2. Dopo aver definito le nuove etichette di classe seguendo le direttive del capitolo 6, per installare l'aggiornamento è sufficiente sostituire il file di classificazione presente nella directory di installazione di Suricata (o modificarlo in loco con i nuovi *classtypes*)

- definire la regola. Sfruttando la classificazione appena installata, occore infine definire la regola per il rilevamento dell'attacco DOS, salvandola in uno dei file della directory *suricata/rules*. Per semplicità è stato definito il nuovo file in cui salvare esclusivamente le regole per il rilevamento dell'attacco DOS:

```
#CUSTOM RULES
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"LOCAL DOS SYN packet
    flood inbound, Potential DOS"; flow:to_server; flags: S,12;
    threshold: type both, track by_dst, count 5000, seconds 5;
    classtype:denial-of-service; sid:5;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"LOCAL DOS SYN packet
    flood outbound, Potential DOS"; flow:to_server; flags: S,12;
    threshold: type both, track by_dst, count 5000, seconds 5;
    classtype:denial-of-service; sid:6;)
```

Codice A.5. custom.rules

A.6 Configurazione di Snort

La configurazione di Snort prevede i medesimi passaggi descritti nella precedente sezione A.5. Come già accennato nel capitolo 5, Snort e Suricata sono del tutto congruenti nella configurazione della classificazione degli alerts e delle regole. Dunque occorre:

- modificare *snort_installation_path/etc/snort.conf*, settando i valori delle variabili di rete, configurando la directory dei log in output tramite direttiva

config logdir, attivare i formati di log di output opportuni ed includere il rule set contenente la regola per il rilevamento dell'attacco DOS:

```
# Setup the network addresses you are protecting
ipvar HOME_NET any
# Set up the external network addresses. Leave as "any" in most
    situations
ipvar EXTERNAL_NET any
...
# Setup the output log directory
config logdir: C:\Snort\log
...
# Setup alert mods and output files
output alert_full: snortVerboseLog.ids
output alert_fast: snortFastLog.ids
...
include ..\Snort\rules\local.rules
...
```

Codice A.6. Snort.conf

- aggiornare il file *snort_installation_path/etc/classification.config*, utilizzando la medesima classificazione decritta in A.2

- aggiornare il rule set locale con la seguente regola per il rilevamento dell'attacco DOS simulato:

```
#CUSTOM RULES
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"LOCAL DOS SYN packet
    flood inbound, Potential DOS"; flow:to_server; flags: S,12;
    threshold: type both, track by_dst, count 5000, seconds 5;
    classtype:denial-of-service; sid:5;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"LOCAL DOS SYN packet
    flood outbound, Potential DOS"; flow:to_server; flags: S,12;
    threshold: type both, track by_dst, count 5000, seconds 5;
    classtype:denial-of-service; sid:6;)
```

Codice A.7. local.rules

A.7 File di log Scenario A

```
06/19/2024-15:58:51.917284 [**] [1:5:0] LOCAL DOS SYN packet flood inbound,
    Potential DOS [**] [Classification: Detection of a Denial of Service
    Attack] [Priority: 3] {TCP} 10.0.0.1 -> 30.0.0.5:21
06/19/2024-15:58:56.953604 [**] [1:5:0] LOCAL DOS SYN packet flood inbound,
    Potential DOS [**] [Classification: Detection of a Denial of Service
    Attack] [Priority: 3] {TCP} 10.0.0.1 -> 30.0.0.5:21
06/19/2024-15:59:01.959055 [**] [1:5:0] LOCAL DOS SYN packet flood inbound,
    Potential DOS [**] [Classification: Detection of a Denial of Service
    Attack] [Priority: 3] {TCP} 10.0.0.1 -> 30.0.0.5:21
...
```

Codice A.8. Fast log prodotti da Snort e Suricata

```
[**] [1:5:0] LOCAL DOS SYN packet flood inbound, Potential DOS [**]
[Detection of a Denial of Service Attack] [Priority: 3]
06/19/2024-15:58:51.917284 10.0.0.1 -> 30.0.0.5:21
TCP TTL:64 TOS:0x0 ID:24934 [... header info ]

[**] [1:5:0] LOCAL DOS SYN packet flood inbound, Potential DOS [**]
[Detection of a Denial of Service Attack] [Priority: 3]
06/19/2024-15:58:56.953604 10.0.0.1 -> 30.0.0.5:21
TCP TTL:64 TOS:0x0 ID:51964 [... header info ]

[**] [1:5:0] LOCAL DOS SYN packet flood inbound, Potential DOS [**]
[Detection of a Denial of Service Attack] [Priority: 3]
06/19/2024-15:59:01.959055 10.0.0.1 -> 30.0.0.5:21
TCP TTL:64 TOS:0x0 ID:29773 [... header info ]
...
```

Codice A.9. Snort Verbose Log

```
{"timestamp":"2024-06-19T15:58:51.917284+0200","flow_id":1124957789213973,
"in_iface":"\\Device\\NPF_{BE187254-6E60-4F3E-A4FD-EBC79D4FF6ED}",
"event_type":"alert","src_ip":"10.0.0.1","src_port":6097,
"dest_ip":"30.0.0.5","dest_port":21,"proto":"TCP","pkt_src":"wire/pcap",
"alert":{"action":"allowed","gid":1,"signature_id":5,"rev":0,"signature":
"LOCAL DOS SYN packet flood inbound, Potential DOS","category":"Detection
 of a Denial of Service Attack","severity":2},"direction":"to_server",
 "flow":{"pkts_toserver":1,"pkts_toclient":0,"bytes_toserver":174,
 "bytes_toclient":0,"start":"2024-06-19T15:58:51.917284+0200",
 "src_ip":"10.0.0.1","dest_ip":"30.0.0.5","src_port":6097,
 "dest_port":21}}

{"timestamp":"2024-06-19T15:58:56.953604+0200","flow_id":155052205281891,
"in_iface":"\\Device\\NPF_{BE187254-6E60-4F3E-A4FD-EBC79D4FF6ED}",
"event_type":"alert","src_ip":"10.0.0.1","src_port":33584,
"dest_ip":"30.0.0.5","dest_port":21,"proto":"TCP","pkt_src":"wire/pcap"
"alert":{"action":"allowed","gid":1,"signature_id":5,"rev":0,"signature":
"LOCAL DOS SYN packet flood inbound, Potential DOS","category":"Detection
 of a Denial of Service Attack","severity":2},"direction":"to_server",
 "flow":{"pkts_toserver":1,"pkts_toclient":0,"bytes_toserver":174,
 "bytes_toclient":0,"start":"2024-06-19T15:58:56.953604+0200","src_ip":
 "10.0.0.1","dest_ip":"30.0.0.5","src_port":33584,"dest_port":21}}

{"timestamp":"2024-06-19T15:59:01.959055+0200","flow_id":1585836379258007,
"in_iface":"\\Device\\NPF_{BE187254-6E60-4F3E-A4FD-EBC79D4FF6ED}",
"event_type":"alert","src_ip":"10.0.0.1","src_port":60062,
"dest_ip":"30.0.0.5","dest_port":21,"proto":"TCP","pkt_src":"wire/pcap",
"alert":{"action":"allowed","gid":1,"signature_id":5,"rev":0,"signature"
:"LOCAL DOS SYN packet flood inbound, Potential DOS","category":"Detection
 of a Denial of Service Attack","severity":2},"direction":"to_server",
 "flow":{"pkts_toserver":1,"pkts_toclient":0,"bytes_toserver":174,
 "bytes_toclient":0,"start":"2024-06-19T15:59:01.959055+0200",
 "src_ip":"10.0.0.1","dest_ip":"30.0.0.5","src_port":60062,
 "dest_port":21}}
...
```

Codice A.10. Formato Json del log di Suricata

```
+================
TIME: 06/19/2024-15:58:51.917284
PKT SRC: wire/pcap
SRC IP: 10.0.0.1
DST IP: 30.0.0.5
PROTO: 6
SRC PORT: 6097
DST PORT: 21
TCP SEQ: 55467304
TCP ACK: 231075530
FLOW: to_server: TRUE, to_client: FALSE
FLOW Start TS: 06/19/2024-15:58:51.917284
FLOW PKTS TODST: 1
FLOW PKTS TOSRC: 0
FLOW Total Bytes: 174
FLOW IPONLY SET: TOSERVER: TRUE, TOCLIENT: FALSE
FLOW ACTION: DROP: FALSE
FLOW NOINSPECTION: PACKET: FALSE, PAYLOAD: FALSE, APP_LAYER: FALSE
FLOW APP_LAYER: DETECTED: FALSE, PROTO 0
PACKET LEN: 174
PACKET:
 0000 22 32 4D 07 11 48 A0 78 17 AA D8 D1 08 00 45 00 "2M..H.x ......E.
 0010 00 A0 16 D6 00 00 40 06 03 9F F6 E5 A7 51 C0 A8 ......@. .....Q..
 0020 01 04 17 D1 00 15 03 4E 5D 28 0D C5 EE CA 50 02 .......N ](....P.
 0030 00 40 25 A6 00 00 58 58 58 58 58 58 58 58 58 58 .@%...XX XXXXXXXX
 0040 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0050 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0060 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0070 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0080 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0090 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 00A0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXX
ALERT CNT: 1
ALERT MSG [00]: LOCAL DOS SYN packet flood inbound, Potential DOS
ALERT GID [00]: 1
ALERT SID [00]: 5
ALERT REV [00]: 0
ALERT CLASS [00]: Detection of a Denial of Service Attack
ALERT PRIO [00]: 2
ALERT FOUND IN [00]: PACKET
ALERT IN TX [00]: N/A
PAYLOAD LEN: 120
PAYLOAD:
 0000 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0010 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0020 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0030 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0040 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0050 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0060 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXX XXXXXXXX
 0070 58 58 58 58 58 58 58 58 XXXXXXXX
+================
...
```

Codice A.11. Suricata Alert Debug

A.8 Funzione di test per la validazione dei risultati

```
private static List<OWASPprop> test(String serviceGraph,
                    String mapperConfPath, String logPath)
                    throws Exception, BadGraphError{
    JAXBContext jc = JAXBContext.newInstance( "it.polito.verefoo.jaxb" );
    // create an Unmarshaller
    Unmarshaller u = jc.createUnmarshaller();
    SchemaFactory sf = SchemaFactory.newInstance(XMLConstants.
    W3C_XML_SCHEMA_NS_URI);
    Schema schema = sf.newSchema( new File( "./xsd/nfvSchema.xsd" ));
    u.setSchema(schema);
    NFV root = (NFV) u.unmarshal(new FileInputStream(serviceGraph));
    VerefooNormalizer norm = new VerefooNormalizer(root);
    root = norm.getRoot();
    IdsPolicyExtraction i = new IdsPolicyExtraction(mapperConfPath);
    List<OWASPprop> testResult = new ArrayList<>();

    for(Graph g : root.getGraphs().getGraph()) {
        List<OWASPprop> owaspProps = root.getPropertyDefinition()
            .getOWASPprop().stream().collect(Collectors.toList());

        g.getNode().stream().filter(n -> n.getFunctionalType().value()
                                        .equals("WEB_APPLICATION_FIREWALL"))
                                        .forEach(waf -> waf.getConfiguration()
                .getWebApplicationFirewall().getOWASPprop().forEach(o -> {
                OWASP value = o.getValue();
                String ipSrc = o.getIPsrc(), ipDst = o.getIPdst();

                if (owaspProps.stream().filter(p -> p.getValue().equals(value)
                    && (p.getIPsrc().equals(ipSrc)
                        ||p.getIPsrc().equals("*"))
                    && p.getIPdst().equals(ipDst)).count() == 0) {
                    OWASPprop waf_prop = new OWASPprop();
                    waf_prop.setValue(value);
                    waf_prop.setIPsrc(ipSrc);
                    waf_prop.setIPdst(ipDst);
                    owaspProps.add(waf_prop);
                }
           }));

        long beginAll=System.currentTimeMillis();
        testResult =i.getFromIdsLog(owaspProps, g.getNode(), logPath);
        long endAll=System.currentTimeMillis();
        System.out.println("OWASP Policies extraction time from log "
                           +logPath+":     " + (endAll-beginAll)+"ms" );
    }
    return testResult;
}
```

Codice A.12. Funzione di test

A.9 Test Scenario A

```
@Test
public static void testScenarioA(){
    try {
        List<String> expected = new ArrayList<>();
        expected.add("BLOCKING_EVALUATION, 10.0.0.1, 30.0.0.5");
        expected.add("APPLICATION_ATTACK_XSS, 10.0.0.1, 30.0.0.5");
        expected.add("DOS_PROTECTION, 10.0.0.1, 30.0.0.5");

        System.out.println("
    **************************************************");
        System.out.println("Testing policy extraction...");

        List<String> res1 = test(scenarioA_sg, scenarioA_mapper,
                            scenarioA_fast)
                            .stream().map(x -> x.getValue()+", "+
                                x.getIPsrc()+", "+x.getIPdst())
                            .collect(Collectors.toList());
        assertTrue(res1.containsAll(expected));

        List<String> res2 = test(scenarioA_sg, scenarioA_mapper,
                            scenarioA_snortDebug)
                            .stream().map(x -> x.getValue()+", "+
                                x.getIPsrc()+", "+x.getIPdst())
                            .collect(Collectors.toList());
        assertTrue(res2.containsAll(res1));

        List<String> res3 = test(scenarioA_sg, scenarioA_mapper,
                            scenarioA_suricataDebug)
                            .stream().map(x -> x.getValue()+", "+
                                x.getIPsrc()+", "+x.getIPdst())
                            .collect(Collectors.toList());
        assertTrue(res3.containsAll(res1));

        List<String> res4 = test(scenarioA_sg, scenarioA_mapper,
                            scenarioA_suricataJson)
                            .stream().map(x -> x.getValue()+", "+
                            x.getIPsrc()+", "+x.getIPdst())
                            .collect(Collectors.toList());
        assertTrue(res4.containsAll(res1));

        System.out.println("
    **************************************************");
        System.out.println("List of policies:");
        res1.forEach(x -> System.out.println(x));
    } catch (Exception e) {
        fail(e.toString());
    }
}
```

Codice A.13. Funzione di test Scenario A

A.10 File di log Scenario B

```
01/04-11:34:49.778290 [**] [1:9000001:0] FIN Scan [**] [Classification:
    Detection of a Network Scan] [Priority: 3] {TCP} 10.0.0.1:1118 ->
    30.0.5.2:0
01/04-11:34:50.781845 [**] [1:9000001:0] FIN Scan [**] [Classification:
    Detection of a Network Scan] [Priority: 3] {TCP} 10.0.0.3:1119 ->
    30.0.5.2:455
01/04-11:34:51.785327 [**] [1:9000001:0] FIN Scan [**] [Classification:
    Detection of a Network Scan] [Priority: 3] {TCP} 10.0.0.1:1118 ->
    30.0.5.2:455
01/04-11:19:34.380494 [**] [1:491:5] FTP Bad login [**] [Classification:
    Deprecated protocol usage or protocol attack detected] [Priority: 1]
    {TCP} 10.0.0.2:21 -> 30.0.5.2:18167
01/04-11:19:34.381792 [**] [1:491:5] FTP Bad login [**] [Classification:
    Deprecated protocol usage or protocol attack detected] [Priority: 1]
    {TCP} 10.0.0.2:21 -> 30.0.5.2:18167
03/11-21:09:27.683976 [**] [1:25101:1] DOS flood denial of service attempt
    [**] [Classification: Detection of a Denial of Service Attack] [Priority:
    2] {TCP} 10.0.0.1:50124 -> 30.0.5.2:80
03/11-21:09:27.808135 [**] [1:25101:1] DOS flood denial of service attempt
    [**] [Classification: Detection of a Denial of Service Attack] [Priority:
    2] {TCP} 10.0.0.2:50121 -> 30.0.5.2:80
03/11-22:30:50.531840 [**] [1:9000010:0] ICMP Packet found [**]
    [Classification: Detection of a Network Scan] [Priority: 3] {ICMP}
    10.0.0.3 -> 30.0.5.2
03/11-22:31:00.454871 [**] [1:25101:1] DOS flood denial of service attempt
    [**] [Classification: Detection of a Denial of Service Attack] [Priority:
    2] {TCP} 10.0.0.3:80 -> 30.0.5.2
03/11-22:31:00.454871 [**] [1:25101:1] DOS flood denial of service attempt
    [**] [Classification: Detection of a Denial of Service Attack] [Priority:
    2] {TCP} 10.0.0.3:80 -> 30.0.5.2
03/11-22:31:00.674865 [**] [1:25101:1] DOS flood denial of service attempt
    [**] [Classification: Detection of a Denial of Service Attack] [Priority:
    2] {TCP} 50.1.0.48:80 -> 30.0.5.2
...
```

Codice A.14. Fast log Scenario B

A.11 Service Graph Scenario B

```
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../xsd/nfvSchema.xsd">
<graphs>
<graph id="0">
<node functional_type="WEBCLIENT" name="10.0.0.1">
       <neighbour name="20.0.0.4" />
       <configuration description="WEB-CLIENT1"
              name="confA">
              <webclient nameWebServer="30.0.5.2" />
       </configuration>
</node>
```

```
<node functional_type="WEBCLIENT" name="10.0.0.2">
       <neighbour name="20.0.0.4" />
       <configuration description="WEB-CLIENT2"
              name="conf1">
              <webclient nameWebServer="30.0.5.2" />
       </configuration>
</node>

<node functional_type="WEBCLIENT" name="10.0.0.3">
       <neighbour name="20.0.0.4" />
       <configuration description="WEB-CLIENT3"
              name="conf1">
              <webclient nameWebServer="30.0.5.2" />
       </configuration>
</node>

<node functional_type="WEBCLIENT" name="50.1.0.*">
       <neighbour name="20.0.0.4" />
       <configuration description="COMPANY-A"
              name="conf1">
              <webclient nameWebServer="30.0.5.2" />
       </configuration>
</node>

<node functional_type="NAT" name="20.0.0.4">
       <neighbour name="50.1.0.*" />
       <neighbour name="10.0.0.1" />
       <neighbour name="10.0.0.2" />
       <neighbour name="10.0.0.3" />
       <neighbour name="20.0.0.1" />
</node>

<node functional_type="FIREWALL" name="20.0.0.1">
       <neighbour name="20.0.0.4" />
       <neighbour name="20.0.0.2" />
</node>

<node functional_type="WEB_APPLICATION_FIREWALL" name="20.0.0.2">
       <neighbour name="20.0.0.5" />
       <neighbour name="20.0.0.4" />
       <configuration description="WAF"
              name="conf1">
              <web_application_firewall defaultAction="ALLOW" >
                     <OWASPprop IPsrc="*"
                         IPdst="30.0.5.2">SCANNER-DETECTION</OWASPprop>
              </web_application_firewall>
       </configuration>
</node>

<node functional_type="FORWARDER" name="20.0.0.5">
       <neighbour name="20.0.0.2" />
       <neighbour name="20.0.0.3" />
       <neighbour name="30.0.5.2" />
</node>
```

```
<node functional_type="TRAFFIC_MONITOR" name="20.0.0.3">
       <neighbour name="20.0.0.5" />
</node>

<node functional_type="WEBSERVER" name="30.0.5.2">
       <neighbour name="20.0.0.5" />
       <configuration description="WEB-SERVER"
              name="confB">
              <webserver>
                     <name>30.0.5.2</name>
              </webserver>
       </configuration>
</node>

</graph>
</graphs>
<Constraints>
<NodeConstraints />
<LinkConstraints />
</Constraints>
<PropertyDefinition>
<Property graph="0" name="IsolationProperty" src="10.0.0.1"
dst="30.0.5.2" />
<Property graph="0" name="IsolationProperty" src="10.0.0.2"
dst="30.0.5.2" />
<OWASPprop IPsrc="10.0.0.1" IPdst="30.0.5.2">BLOCKING-EVALUATION</OWASPprop>
<OWASPprop IPsrc="10.0.0.2"
    IPdst="30.0.5.2">APPLICATION-ATTACK-XSS</OWASPprop>
</PropertyDefinition>
<ParsingString></ParsingString>
</NFV>
```

Codice A.15. Service Graph Scenario B

A.12 Test Scenario B

```
@Test
    public static void testScenarioB(){
        try {
            List<String> expected = new ArrayList<>();
            expected.add("BLOCKING_EVALUATION, 10.0.0.1, 30.0.5.2");
            expected.add("APPLICATION_ATTACK_XSS, 10.0.0.2, 30.0.5.2");
            expected.add("SCANNER_DETECTION, *, 30.0.5.2");
            expected.add("DOS_PROTECTION, 10.0.0.2, 30.0.5.2");
            expected.add("DOS_PROTECTION, 10.0.0.3, 30.0.5.2");
            expected.add("DOS_PROTECTION, 10.0.0.1, 30.0.5.2");
            expected.add("DOS_PROTECTION, 50.1.0.48, 30.0.5.2");
            expected.add("PROTOCOL_ATTACK, 10.0.0.2, 30.0.5.2");
            expected.add("PROTOCOL_ENFORCEMENT, 10.0.0.2, 30.0.5.2");

            System.out.println("Testing Scenario B policy extraction...");
            System.out.println("
    ***************************************************");
```

```
            List<String> res = test(scenarioB_sg,
                                    mapperConfigPath,
                                    scenarioB_log)
                               .stream().map(x -> x.getValue()+","+
                                                 x.getIPsrc()+", "+
                                                 x.getIPdst())
                               .collect(Collectors.toList());
            assertTrue(res.containsAll(expected));

            res.forEach(x -> System.out.println(x));
        } catch (Exception e) {
            fail(e.toString());
        }
    }
```

Codice A.16. Funzione di test Scenario B