POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Configurazione automatica di Web Firewall

Relatori
Prof. Fulvio Valenza
Prof. Riccardo Sisto
Dott. Daniele Bringhenti

Candidato
Giosuè Gisina

Anno Accademico 2022-2023

Alla mia famiglia

Ringraziamenti

Grazie alla mia famiglia, ai miei genitori. Non solo per questo traguardo ma per il loro supporto in tutto ciò che ho fatto e che farò.

Grazie alla persona che c'è stata durante questi anni.

Sommario

La Virtualizzazione delle Funzioni di Rete (in inglese Network Functions Virtualization, NFV) rappresenta una tecnologia avanzata che permette di separare le funzioni di rete dall'hardware specializzato. Questo disaccoppiamento consente l'eliminazione dei dispositivi hardware dedicati per ciascuna funzione di rete, favorendo invece l'esecuzione di tali funzioni tramite processi software su server generici.

Uno dei risultati che si possono ottenere attraverso la NFV è la creazione di un Service Graph, cioè una rappresentazione di come le varie funzioni di rete sono disposte ed interconnesse tra loro. Tuttavia, sorge un problema: la creazione del Service Graph è solitamente compito di un amministratore di rete, che si occupa manualmente dell'allocazione e della configurazione delle Network Security Functions (NSF). Queste NSF includono, ad esempio, web-application firewall, essenziali per proteggere la rete da minacce di sicurezza informatica. Purtroppo, le operazioni manuali eseguite dall'amministratore rendono il processo suscettibile ad errori umani.

L'obiettivo della presente tesi è quello di presentare una possibile soluzione al problema precedentemente descritto. Questa tesi ha difatti contribuito allo sviluppo di VEREFOO (VErified REFinement and Optimized Orchestration), un framework che si propone di fornire un approccio automatizzato per garantire la sicurezza di rete e risolvere il problema sopra menzionato. Il principale contributo fornito dal presente lavoro di tesi è rivolto ai WAFs (Web Application Firewalls) ed alla loro applicabilità nell'ambito di VEREFOO.

Lo scopo principale di VEREFOO è quello di automatizzare ed ottimizzare l'allocazione e la configurazione delle NSF necessarie per soddisfare i requisiti di sicurezza della rete. Questi requisiti possono essere espressi dall'amministratore di rete utilizzando un linguaggio di alto livello, rendendo l'intero processo più intuitivo.

L'ottimizzazione compiuta da VEREFOO si basa sulla formulazione di un problema MaxSMT (Satisfiability Modulo Theories) e la sua successiva risoluzione. L'obiettivo è soddisfare un insieme di clausole definite "hard" che devono essere sempre rispettate e di massimizzare la somma dei pesi attribuiti alle clausole "soft", che invece non è necessario che siano soddisfatte. VEREFOO utilizza la combinazione di clausole hard e soft per

stabilire dove, all'interno del Service Graph, è necessario assegnare determinate NSF, minimizzando il numero di istanze (di NSF) ed il numero di regole calate all'interno di ogni istanza.

Un suggerimento per ottimizzare la configurazione di un WAF è fornito all'interno della tesi: in futuro, sarà essenziale estendere le funzionalità per supportare il carattere jolly "*" per le variabili di tipo stringa nel modello z3. Questa funzionalità è cruciale per ridurre il numero di configurazioni presenti all'interno di un WAF, consentendo un'implementazione ancora più efficiente.

Contents

Elenco delle figure 9

1 Introduzione 11
- 1.1 Struttura del documento . . . 12
- 1.2 Citazione fonti . . . 12

2 SDN e NFV 15
- 2.1 Software Defined Networking . . . 15
- 2.2 Network Function Virtualization . . . 17

3 VEREFOO 21
- 3.1 Il workflow di Verefoo . . . 22
- 3.2 Maximum Satisfiability Modulo Theories . . . 24
  - 3.2.1 Il problema SMT . . . 24
  - 3.2.2 Il problema MaxSMT . . . 25
  - 3.2.3 Introduzione al solver z3 . . . 25

4 Web Application Firewall 27
- 4.1 Obiettivo della tesi . . . 27
- 4.2 Il concetto . . . 28
- 4.3 Modalità di utilizzo . . . 29
- 4.4 Possibili soluzioni per WAF . . . 30
  - 4.4.1 ModSecurity . . . 31
  - 4.4.2 Snort . . . 33
  - 4.4.3 Squid . . . 36
  - 4.4.4 ModSecurity vs Snort vs Squid . . . 37

5 Modellizzazione dei Requisiti di Sicurezza 39
- 5.1 Il modello . . . 40

5.1.1 Funzionalità di web-filtering e time-filtering . . . 40
5.1.2 Time filtering: utilizzo di TIMEstart e TIMEend . . . 42
5.1.3 Protezione da attacchi web . . . 43
5.2 Rappresentazione XML . . . 44
5.2.1 HSPL e MSPL . . . 44
5.2.2 Requisiti . . . 45
5.2.3 Schema di output . . . 47
5.3 Esempi di configurazione . . . 49
5.4 Modellizzazione problema MaxSMT . . . 52
5.4.1 Funzione match . . . 53
5.4.2 Modello per i Network Security Requirements . . . 54
5.4.3 Algoritmo di calcolo per flussi massimali . . . 55
5.4.4 Allocazione e configurazione firewall . . . 56

6 Implementazione e Validazione 59
6.1 Implementazione . . . 59
6.1.1 Requisiti per l'implementazione . . . 59
6.1.2 Il processo . . . 59
6.2 Validazione . . . 60
6.2.1 Caso d'uso n.1 . . . 61
6.2.2 Caso d'uso n.2 . . . 62
6.2.3 Caso d'uso n.3 . . . 63

7 Conclusione e lavori futuri 69

Bibliografia 71

Elenco delle figure

2.1 Possibile schema logico di un'architettura SDN. 17
2.2 Esempio di una Service Function Chain relativa al solo utilizzo di tecnologia SDN. 18
2.3 Esempio di una Service Function Chain relativa all'utilizzo combinato di tecnologia SDN e NFV. 18

3.1 Esempio di Service Graph. 22
3.2 Esempio di Allocation Graph creato a partire dal Service Graph. 22
3.3 Rappresentazione del modello del framework VEREFOO. 23

4.1 Differenza tra un Network Firewall (PF) e un Web Application Firewall. . 28

5.1 Rappresentazione grafica Allocation Graph per esempi di configurazione. . 50
5.2 Esempio n.1: rappresentazione grafica dell'allocazione di WAF (web filtering). 50
5.3 Esempio n.2: rappresentazione grafica dell'allocazione di WAF (web filtering). 51
5.4 Esempio n.3: rappresentazione grafica dell'allocazione di WAF (time filtering). 52
5.5 Esempio n.4: rappresentazione grafica dell'allocazione di WAF (OWASP properties). 53
5.6 Algoritmo di calcolo per flussi massimali. 56

6.1 Allocation Graph per caso d'uso implementazione n.1 61
6.2 Grafo per caso d'uso implementazione n.1 61
6.3 Allocation Graph per caso d'uso implementazione n.1 63
6.4 Grafo per caso d'uso implementazione n.2 - soluzione non ottima 64
6.5 Grafo per caso d'uso implementazione n.2 - soluzione ottima 65
6.6 Service Graph per caso d'uso implementazione n.1 66
6.7 Grafo per caso d'uso implementazione n.3 - soluzione non ottima 67
6.8 Grafo per caso d'uso implementazione n.3 - soluzione ottima 68

Capitolo 1

Introduzione

Per il prosieguo della presente tesi è fondamentale introdurre due fondamentali concetti nell'ambito delle reti di telecomunicazioni moderne: la NFV (Network Function Virtualization) e l'SDN (Software Defined Networking). In un panorama in continua evoluzione, queste tecnologie hanno guadagnato sempre più importanza nella trasformazione e nell'ottimizzazione delle reti di comunicazione. La NFV, a differenza dell'approccio tradizionale basato su dispositivi hardware dedicati, mira a virtualizzare le funzioni di rete, consentendo di eseguirle su server virtuali, migliorando così l'agilità e la flessibilità delle reti. D'altra parte, l'SDN è un paradigma che separa il controllo della rete dal forwarding dei dati, permettendo una gestione centralizzata e programmatica delle risorse di rete. L'integrazione sinergica di NFV e SDN apre la strada a una serie di benefici significativi, tra cui una maggiore scalabilità, riduzione dei costi operativi, implementazione rapida di nuovi servizi e un migliore adattamento alle esigenze dinamiche degli utenti.

Questa ricerca sfrutta le potenzialità di queste tecnologie per la creazione di un Service Graph (SG). Un SG è la combinazione di più funzioni di rete collegate tra loro. E' evidente che, tramite l'utilizzo della NFV, è molto più semplice creare un SG e risulta anche molto più flessibile modificarlo.

Il framework VEREFOO (VErified REFinement and Optimized Orchestration) rappresenta una soluzione innovativa che mira a raffinare le politiche di sicurezza, traducendo i requisiti ad alto livello in espressioni di complessità intermedia. L'obiettivo è ottenere un'allocazione automatizzata ed ottimale delle Funzioni di Sicurezza di Rete necessarie per soddisfare i vincoli di sicurezza all'interno di una topologia indicata. Inoltre, il framework si concentra sull'ottimizzazione della distribuzione delle regole di politica assegnate alle funzioni allocate e sulla corretta implementazione delle funzioni virtualizzate nella rete di supporto.

L'idea cardine di VEREFOO è quella di limitare il più possibile gli errori commessi

dalla componente umana nella creazione di un service graph, lasciando questo compito ad una entità automatizzata, cioè VEREFOO. Il framework, infatti, riceve in input una descrizione logica della rete ed un set di Security Policies e fornisce un SG con le NSF (Network Security Functions) già allocate e configurate in modo appropriato.

Per raggiungere tali risultati, VEREFOO risolve un problema di MaxSMT sfruttando il solver z3, un risolutore di teoremi sviluppato da Microsoft Research, e Verigraph, un framework sviluppato presso il Politecnico di Torino, specializzato nella verifica dei requisiti per scenari di Virtual Network Embedding (VNE).

Il presente lavoro di tesi si è focalizzato sull'estensione del modulo ADP di VEREFOO. In particolare ci si è concentrati sulla progettazione ed implementazione di un Web Application Firewall (WAF). I WAF, infatti, possono essere allocati da VEREFOO per soddisfare i requisiti di sicurezza ricevuti in input.

1.1 Struttura del documento

Questo lavoro di tesi è stato diviso nei seguenti capitoli, la cui sequenza logica segue la stessa sequenza temporale con cui è stata svolta la tesi:

- **Capitolo 2**: presenta in due sezioni separate, in maniera esaustiva, i concetti già introdotti di NFV ed SDN;
- **Capitolo 3**: illustra l'obiettivo e il funzionamento del framework VEREFOO e delle sue componenti;
- **Capitolo 4**: descrive l'obiettivo della tesi ed analizza le funzionalità dei WAF in commercio, giungendo alla conclusione, a fine capitolo, di quale sia il software più adatto a soddisfare le necessità di VEREFOO;
- **Capitolo 5**: tratta l'analisi e la modellizzazione dei requisiti di sicurezza;
- **Capitolo 6**: tratta il processo di implementazione del WAF, per quanto concerne l'attivazione di regole OWASP;
- **Capitolo 7**: conclude il presente documento di tesi illustrando cosa è stato raggiunto e quali sono i possibili lavori futuri.

1.2 Citazione fonti

Di seguito vengono citate le fonti riportate in bibliografia che si sono dimostrate di fondamentale importanza per uno studio preliminare per il presente lavoro di tesi.

Automazione per la sicurezza nelle reti:

1. Automation for network security configuration: state of the art and research trends [1]
2. Towards Security Automation in Virtual Networks [2]

Verefoo (in generale):

1. A novel approach for security function graph configuration and deployment [3]
2. A novel abstraction for security configuration in virtual networks [4]

Verefoo (firewall) e Gestione Firewall:

1. Automated optimal firewall orchestration and configuration in virtualized networks [5]
2. Automated firewall configuration in virtual networks [6]
3. Optimizing distributed firewall reconfiguration transients [7]
4. Introducing programmability and automation in the synthesis of virtual firewall rules [8]
5. A demonstration of VEREFOO: an automated framework for virtual firewall configuration [9]
6. Automating the configuration of firewalls and channel protection systems in virtual networks [10]
7. An Optimized Approach for Assisted Firewall Anomaly Resolution [11]

Altro:

1. Towards a fully automated and optimized network security functions orchestration [12]
2. OWASP ModSecurity Core Rule Set [13]
3. Adding Support for Automatic Enforcement of Security Policies in NFV Networks [14]
4. User-oriented Network Security Policy Specification [15]

Capitolo 2

SDN e NFV

Questo capitolo ha lo scopo di introdurre due concetti fondamentali per il prosieguo dell'analisi. Le Software-Defined Networks (SDN) e la Networks Functions Virtualization (NFV) attribuiscono un peso di grande rilievo al concetto di software e nel come esso può, in parte, sostituire l'hardware. Questo porta ad una serie di vantaggi tra cui la semplicità di sviluppo di una nuova funzionalità e la facilità con la quale il software può essere riprogrammato ogni qualvolta serva effettuare una modifica. I concetti di SDN e NFV verranno di seguito trattati in maniera dettagliata.

Altro concetto fondamentale che è bene chiarire prima di inserirsi nell'analisi dettagliata delle due tecnologie è quello di Service Functions Chain.

Una Service Function altro non è che un componente logico, interconnesso solitamente con altri componenti, che può integrare svariate funzioni di rete.

Il Service Function Chaining (SFC) è il processo di connessione dei servizi di rete (virtuali) in una catena utilizzando la programmabilità del Software-Defined Networking (SDN). La possibilità di configurare automaticamente le connessioni di rete virtuale per gestire vari flussi di traffico è uno dei vantaggi derivanti dall'utilizzo di SFC.

2.1 Software Defined Networking

Il Software Defined Networking (SDN) è un paradigma di gestione delle reti di telecomunicazioni che si basa sulla separazione del controllo della rete dal forwarding dei dati. Nelle reti tradizionali i dispositivi di rete come switch e router eseguono sia il controllo che il forwarding dei pacchetti dati. In un'architettura SDN, il controllo della rete è decentralizzato e gestito da un'entità centrale chiamata Controller SDN.

Il Controller SDN è responsabile della definizione delle politiche di rete e dell'elaborazione delle regole di instradamento dei pacchetti. Esso interagisce con gli elementi di

rete mediante un'interfaccia di programmazione ben definita (API), come OpenFlow, consentendo di configurare dinamicamente le regole di routing e le politiche di rete in modo centralizzato.

I dispositivi di rete (come switch, router e access point) all'interno di un'architettura SDN vengono denominati "Forwarding Devices" e hanno una funzione essenzialmente passiva. Essi inoltrano i pacchetti dati in base alle regole e politiche stabilite dal Controller SDN.

Questo approccio decentralizzato e programmabile dell'SDN offre notevoli vantaggi in termini di flessibilità, gestione semplificata e scalabilità. La separazione del controllo e del forwarding permette di apportare modifiche alla rete in modo rapido e dinamico, adattandola alle mutevoli esigenze degli utenti. Inoltre, l'SDN facilita l'implementazione di nuove funzionalità di rete e la creazione di servizi personalizzati, in quanto le regole possono essere programmate e modificate in modo centralizzato senza dover intervenire sui singoli dispositivi di rete.

Un altro vantaggio dell'SDN è la possibilità di sfruttare l'intelligenza del Controller SDN per ottimizzare il traffico di rete, migliorando le prestazioni complessive e riducendo i tempi di latenza. Il Controller può analizzare il traffico di rete in tempo reale e prendere decisioni di routing intelligenti per ottimizzare il flusso dei pacchetti.

I componenti di inoltro dei dati, che possono essere switch white label invece di router complessi o dispositivi personalizzati dal fornitore, sono ciò che costituisce l'infrastruttura di rete. La loro unica funzione è inoltrare i pacchetti di input alle porte di output appropriate lungo il percorso per raggiungere le destinazioni finali il più rapidamente possibile, poiché il concetto chiave della tecnologia SDN è proprio l'efficienza di inoltro dei pacchetti.

Ogni switch dispone di una tabella di inoltro che contiene un elenco di regole. Queste regole consentono allo switch di filtrare solo i pacchetti che rispettano le regole e specificano le azioni che lo switch deve intraprendere su tali pacchetti (come l'inoltro a una porta specifica, l'inoltro alla piattaforma del controller, l'inoltro a una velocità specifica, l'eliminazione del pacchetto, spingendo o spuntando un campo, sovrascrivendo o modificando un campo di intestazione).

In conclusione, l'SDN rappresenta un approccio rivoluzionario nella gestione delle reti di telecomunicazioni, consentendo una maggiore agilità, scalabilità e facilità di gestione delle reti. La sua architettura programmabile e basata su Controller offre un'elevata automazione e personalizzazione, aprendo la strada a nuove opportunità per l'evoluzione delle reti verso un futuro sempre più intelligente e dinamico.

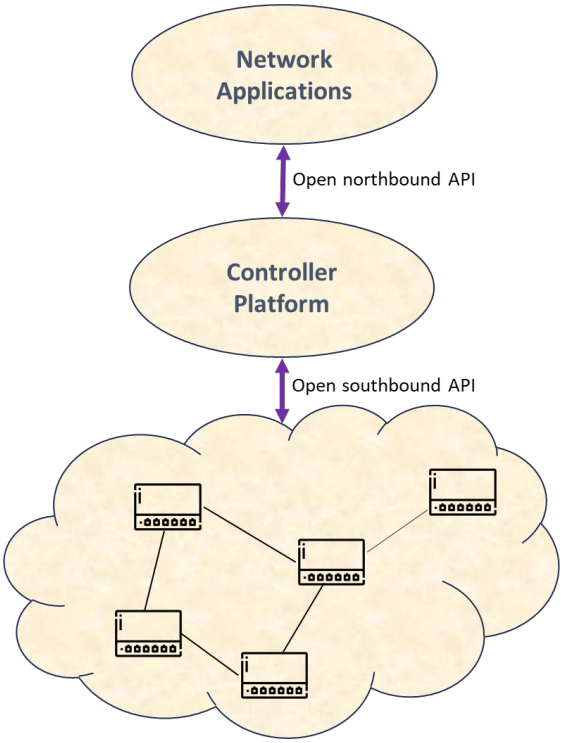

Figura 2.1. Possibile schema logico di un'architettura SDN.

2.2 Network Function Virtualization

Il concetto fondamentale della tecnologia NFV (Network Function Virtualization) è che ciascuna funzione è un processo software che può essere eseguito su un server generico anziché su hardware dedicato aggiuntivo. Di conseguenza, mentre SDN si concentra sulla costruzione di canali di inoltro tramite software, NFV mira alla virtualizzazione dell'informatica, in particolare, per quanto riguarda il nostro caso, delle funzioni di rete.

Dal concetto di NFV discende immediatamente quello di VNF, Virtualized Network Function. Mentre con NFV si intende la tecnologia, con VNF si intende nello specifico il blocco funzione virtualizzato. Il blocco Virtualized Network Function (VNF) si distingue per le particolari funzioni software che sfruttano software di virtualizzazione per funzionare senza la necessità di hardware specializzato.

Grazie alla tecnologia NFV non è necessario che ciascuna funzione nella Service Function Chain sia una scatola hardware specifica; può invece trattarsi di una funzione software collocata su un server, grazie alla tecnologia di virtualizzazione delle funzioni di rete. La catena delle funzioni di servizio modellata nella 2.2 può ora essere visualizzata come nella 2.3, dove le funzioni sono macchine virtuali che possono essere distribuite sullo stesso

server anziché su dispositivi hardware più unici.

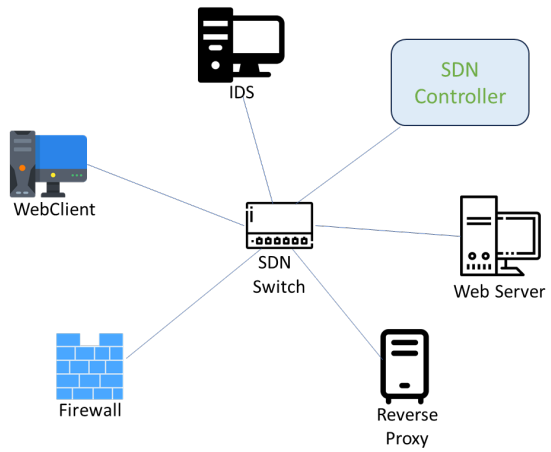

Figura 2.2. Esempio di una Service Function Chain relativa al solo utilizzo di tecnologia SDN.

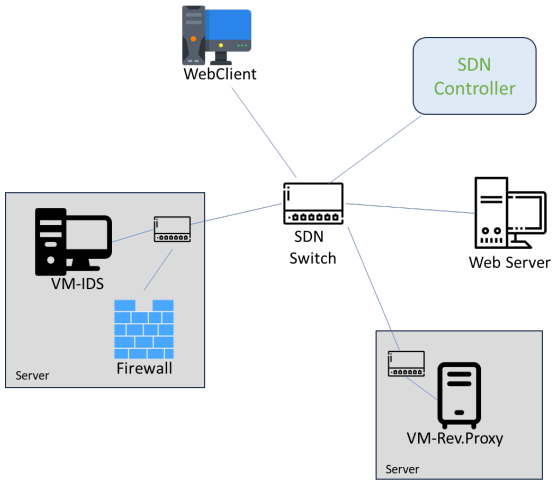

Figura 2.3. Esempio di una Service Function Chain relativa all'utilizzo combinato di tecnologia SDN e NFV.

Come si può ben notare, quindi, grazie all'utilizzo della tecnologia NFV, combinata all'SDN, si ottengono ulteriori vantaggi. Esaminiamone qualcuno:

1. uno switch software che può essere configurato come switch SDN fornisce connettività dinamica per le funzioni all'interno dello stesso server;

2. Le risorse informatiche sono condivise da tutti i servizi virtualizzati in esecuzione sullo stesso server e possono essere allocate dinamicamente in base alla domanda;

3. poiché non è necessario acquistare o configurare alcuna unità hardware, una nuova funzione può essere fornita con molta flessibilità semplicemente creando una nuova macchina virtuale o Docker nell'host del sistema operativo;

Capitolo 3

VEREFOO

Gli obiettivi principali del framework noto come VEREFOO (VErified REfinement and Optimized Orchestration) sono il perfezionamento dei requisiti di sicurezza di rete (NSRs) di alto livello, l'allocazione ottimale e la configurazione automatica delle funzioni di sicurezza di rete (NSFs) scelte per soddisfare i vincoli di sicurezza e il posizionamento di ciascuna funzione di rete virtuale (nel caso della presente tesi si tratta di Web Application Firewalls) all'interno dell'Allocation Graph, una topologia creata a partire dal Service Graph (entrambi i concetti saranno spiegati di seguito). Per risolvere il problema delle teorie del modulo di massima soddisfacibilità (MaxSMT) in un ambiente cloud, si utilizza il solver z3Opt.

Terminologia:

- ***SERVICE GRAPH*** La cache web, il bilanciamento del carico e il monitoraggio del traffico sono alcuni esempi delle funzioni di rete che compongono questa topologia logica e vengono utilizzate da un progettista di servizi, o dalla persona responsabile della definizione del servizio, per assemblare un intero end- servizio completo. In altre parole, un Service Graph è essenzialmente una generalizzazione dell'idea di Service Function Chain (SFC). Mentre una SFC è una catena di funzioni di rete, un Service Graph è una topologia più sofisticata, in cui possono esistere diversi percorsi tra ciascuna coppia di punti finali e possono essere inclusi alcuni loop.

- ***ALLOCATION GRAPH*** Si tratta di una topologia logica che è stata prodotta a partire da un Service Graph e si distingue per avere più nodi e lo stesso insieme di funzioni di rete del Service Graph. Queste parti aggiuntive fungono da luoghi di allocazione in cui vi è *la possibilità* , non per forza la necessità, di inserire una funzione di sicurezza della rete se questa è la posizione migliore per essa.

Per rendere ancor più chiaro il concetto e la differenza tra i due grafi si riporta di seguito un esempio. Viene rappresentato in figura 3.1 un esempio di Service Graph molto semplice, con due switch e cinque funzioni di rete generiche collegate agli switch come mostrato.

Da esso si può ricavare l'Allocation Graph, riportato in figura 3.2, semplicemente posizionando degli Allocation Point su ogni link che collega switch-funzione e switch-switch.

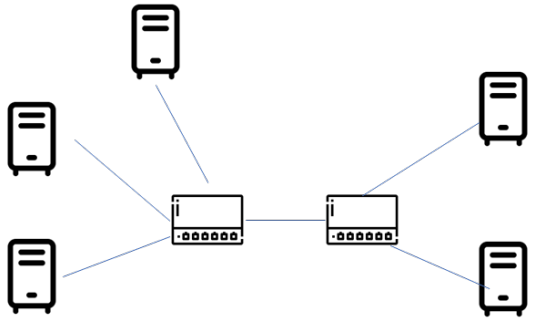

Figura 3.1. Esempio di Service Graph.

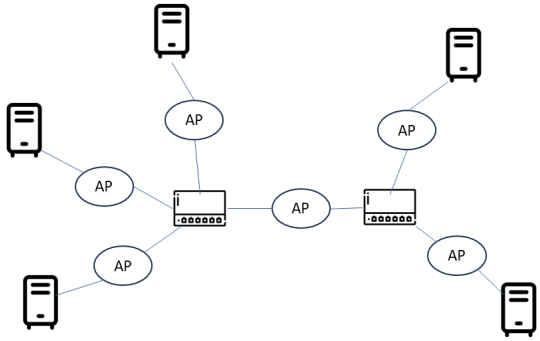

Figura 3.2. Esempio di Allocation Graph creato a partire dal Service Graph.

3.1 Il workflow di Verefoo

In questa sezione viene esposto ed analizzato il modello del framework VEREFOO. La figura 3.3 consente di avere una rappresentazione grafica delle componenti logiche del framework. Spieghiamo, inizialmente, quali sono i possibili input che il sistema accetta e dopo passiamo ad analizzare nel dettaglio ogni componente logica di VEREFOO. Inputs:

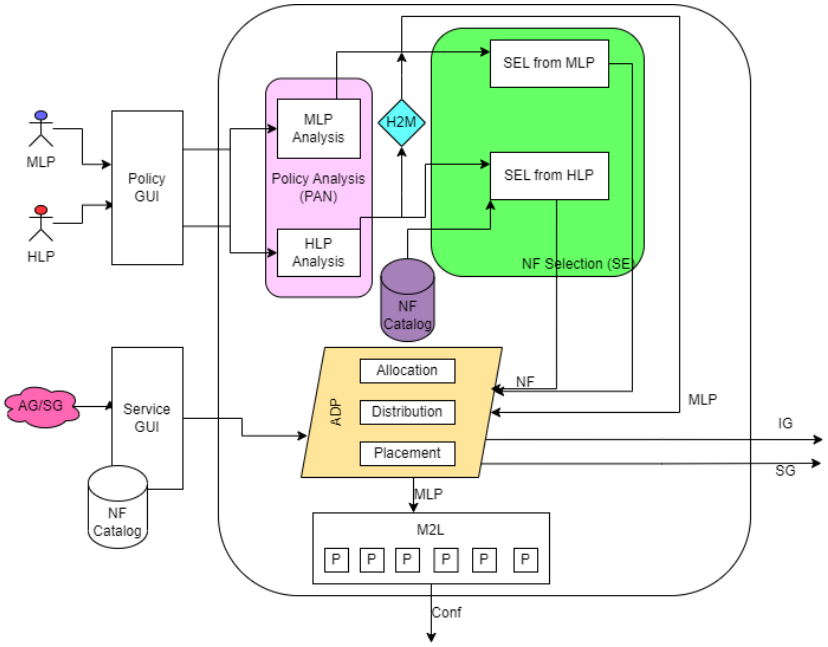

Figura 3.3. Rappresentazione del modello del framework VEREFOO.

- un insieme di requisiti di sicurezza di rete (NSRs, Network Security Requirements) che utilizzano un linguaggio di alto o medio livello a seconda del livello di competenza dell'utente per esprimere le restrizioni di sicurezza che devono essere soddisfatte;

- un SG o, in alternativa, direttamente un AG attraverso una GUI, che dà accesso ad un Catalogo delle Funzioni di Rete dal quale l'utente può scegliere quali funzioni – semplici funzioni di rete o anche Funzioni di Sicurezza di Rete – allocare immediatamente sul proprio grafico. Questo è il caso se ci si sente sicuri di specificare fin dall'inizio le posizioni dei luoghi di assegnazione in cui devono essere installate le funzioni di sicurezza della rete.

Passiamo ad analizzare nel dettaglio le componenti del framework.

- Il componente centrale dell'architettura è il modulo Allocation, Distribution and Placement (ADP), la cui funzione è calcolare un grafico di servizio con le funzioni di sicurezza di rete aggiunte o un grafico fisico utilizzando i requisiti di sicurezza di livello medio, un elenco di NSRs e il SG originale o direttamente l'AG come input.

  L'elenco delle regole di policy di medio livello che devono essere applicate a ciascuna istanza della funzione di rete è un altro output dell'elemento ADP. Il modulo

Medium-to-Low (M2L) genera quindi la corrispondente configurazione di basso livello che dipende dalla precisa implementazione della funzione del vendor.

- Il modulo NF Selection (SE) svolge un ruolo cruciale nel determinare quali funzioni di sicurezza di rete sono necessarie per soddisfare i requisiti di sicurezza di rete di alto e medio livello selezionandole da un elenco predefinito, che è lo stesso elenco a cui ha accesso il progettista del servizio tramite la GUI del servizio. Pur non avendo conoscenza della topologia del Grafico di Allocazione, questo passaggio può richiedere una procedura di ottimizzazione attraverso la quale viene scelta la migliore raccolta di Funzioni di Sicurezza della Rete, e questo elemento può costituire una potenziale limitazione.

- Se i requisiti di sicurezza specificati sono espressi in un linguaggio di alto livello, il modulo "High to Medium" (H2M) li riformula per produrre un insieme di requisiti di sicurezza di rete di medio livello. Tali requisiti contengono tutti i dati necessari per la creazione delle policy per le Funzioni di Sicurezza di Rete che vengono automaticamente allocate sul grafo e la configurazione di basso livello delle VNF installate successivamente;

- Il modulo Policy Analysis (PAN), che prende come input i requisiti di sicurezza della rete, ha lo scopo di eseguire un'analisi dei conflitti, determinare se qualcuno dei requisiti è in conflitto e stabilire l'insieme minimo di restrizioni che devono essere rispettate. Nei casi in cui i requisiti di sicurezza in input contengano errori che non possono essere corretti mediante questa procedura automatica ma richiedono una riformulazione da parte dell'utente, questo può fornire una tempestiva segnalazione di non applicabilità al progettista del servizio.

3.2 Maximum Satisfiability Modulo Theories

3.2.1 Il problema SMT

Il problema della soddisfacibilità proposizionale (SAT) prende in input una formula proposizionale e un insieme di variabili booleane ed accerta se esiste o meno una combinazione di valori per queste variabili tali che si possa garantire la soddisfacibilità delle formule. Un risolutore SAT deve solo identificare una risposta tra tutte quelle possibili e non è tenuto a determinare la migliore combinazione possibile di variabili proposizionali. Dato che questo ultimo concetto risulta per noi essere una necessità, si procede con la descrizione del problema MaxSMT.

3.2.2 Il problema MaxSMT

Il problema delle teorie del modulo di massima soddisfacibilità (MaxSMT) è un'estensione del problema SMT nel contesto dell'ottimizzazione, dove l'obiettivo è identificare **i migliori valori** delle variabili che massimizzano la massima soddisfacibilità delle clausole, avendo in input un insieme di clausole.

Esistono alcune varianti di MaxSMT:

- MaxSMT *weighted*, dove è possibile attribuire pesi diversi ad ogni clausola e, di conseguenza, la ricerca dell'ottimo verrà fatta tenendo presente di voler massimizzare il peso totale delle clausole soddisfatte;
- MaxSMT *partial*, dove alcune clausole, che chiameremo *hard-constraints*, sono obbligatorie e devono per forza essere soddisfatte. Mentre altre clausole possono non essere soddisfatte;
- MaxSMT *weighet-partial*, che risulta essere una combinazione delle prime due varianti. In questa terza possibilità possiamo sia attribuire pesi diversi alle clausole, sia dichiarare queste come *hard* o *soft*.

Il problema che affronteremo in questo lavoro di tesi è l'ultimo: il *weighted-partial MaxSMT*. Esso, nello specifico caso del framework VEREFOO, ha il compito di selezionare le funzioni di rete necessarie, posizionarle correttamente e stabilire la configurazione ottimale sulla base di un set di policy di sicurezza passate in ingresso e di un Service Graph.

3.2.3 Introduzione al solver z3

Al fine di risolvere i problemi di SMT (e MaxSMT) Microsoft Research ha creato un solver noto come z3.

Z3 trasforma un insieme di formule, dopo averle ricevute attraverso un'interfaccia, in un file SMTLIB2 sulla base di un linguaggio ed una semantica specificati a livello internazionale per offrire uno sfondo uniforme per le teorie SMT. Quindi, nel tentativo di ridurre il tempo di calcolo complessivo o di arrivare a un risultato non ideale, tenta inizialmente di utilizzare strategie come la pre-elaborazione o l'euristica. Infine si avvale di un particolare risolutore (come SAT o Fixedpoint) per ottenere una soluzione o, nel caso sia richiesta un'ottimizzazione, la migliore soluzione possibile; in particolare, z3 utilizza un motore di ottimizzazione noto come z3Opt per il processo di ottimizzazione.

Capitolo 4

Web Application Firewall

4.1 Obiettivo della tesi

Sulla base dell'introduzione fatta nei capitoli 2 e 3, in questo paragrafo verrà descritto nello specifico qual è l'obiettivo del presente documento. Com'è stato già trattato nel precedente capitolo, il framework VEREFOO ha come obiettivo quello di allocare funzioni di sicurezza nella rete sulla base dei requisiti di sicurezza passati come input. L'obiettivo della tesi è, in particolare, quello di allocare una particolare funzione di sicurezza: i Web Application Firewalls. Per raggiungere questo obiettivo è stato modificato il comportamento del modulo ADP. Si analizzano adesso step-by-step i passaggi tramite i quali è stato perseguito l'obiettivo:

1. Sono stati analizzati i più popolari WAF open-source in commercio. L'analisi, in particolare, è stata fatta su tre soluzioni che presentano aspetti differenti tra loro: ModSecurity, Squid e Snort. L'analisi è stata conclusa con la decisione di utilizzare ModSecurity in quanto possiede caratteristiche idonee alle necessità di VEREFOO.

2. Il secondo passo è stato creare un modello, dapprima preliminare, in seguito dettagliato. Il modello comprende la definizione formale dei requisiti di sicurezza, differenziando due tipologie di regole:

   (a) regole per il web e time filtering;

   (b) regole per l'attivazione dell'OWASP-crs. Le regole comprendono attributi ed azioni.

3. Il terzo passo è stato ridefinire il modello del problema di MaxSMT, apportando le necessarie modifiche ai vincoli del modello già esistente (per i packet filter).

4. Il quarto passo è stato implementare la funzionalità di attivazione delle regole OWASP. Tale funzionalità permette di inserire in input, oltre al grafo di rete, policy di sicurezza che indicano su quale flusso deve essere attivata una determinata regola. Il framework dopo aver effettuato il processo di ottimizzazione tramite il solver z3, inserisce il WAF nell'Allocation Point corretto.

4.2 Il concetto

Il concetto principale su cui si basa lo sviluppo della presente tesi è quello di Web Application Firewall. Nel seguente capitolo verrà presentata una descrizione dei WAF (Web Application Firewall), partendo dal cos'è un WAF, come funziona, quali sono le migliori soluzioni in questo momento e quale abbiamo scelto di adottare.

Una tecnologia di sicurezza chiamata web application firewall (WAF) è progettata per proteggere le applicazioni web da numerosi pericoli e intrusioni su Internet. Al fine di rilevare e ridurre possibili problemi di sicurezza, funge da barriera tra un'applicazione Web e Internet monitorando e filtrando sia i dati in entrata che quelli in uscita.

Come mostrato in figura 4.1, a differenza di un Packet Filter che filtra il traffico di rete analizzandolo fino al livello Trasporto (TCP, UDP), un WAF riesce ad analizzare il traffico fino a livello applicativo (HTTP/HTTPs), di conseguenza è possibile applicare regole più dettagliate. Ovviamente il WAF ha anche visibilità sui campi del livello trasporto, quindi fa anche il lavoro di un network firewall.

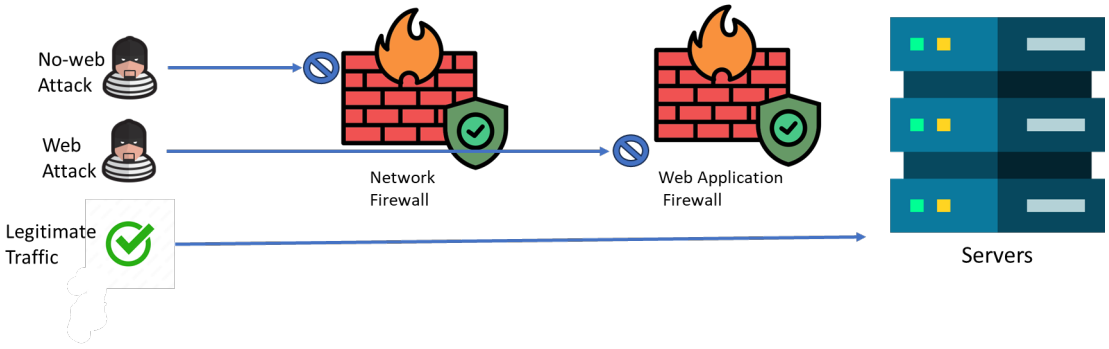

Figura 4.1. Differenza tra un Network Firewall (PF) e un Web Application Firewall.

Di seguito vengono descritte più in dettaglio le caratteristiche e le finalità di un web application firewall.

- **Protezione contro gli attacchi alle applicazioni Web**: le apparecchiature di sicurezza delle applicazioni Web (WAF) vengono generalmente utilizzate per proteggere i siti Web da una varietà di pericoli online, tra cui, a titolo esemplificativo:
  - Introducendo *query SQL* dannose, è possibile rilevare e fermare i tentativi di influenzare il database di un'applicazione web.
  - Prevenire l'inserimento di script pericolosi nei siti web in modo che non possano essere eseguiti da utenti incauti (*cross-site scripting, o XSS*).
  - Blocca le richieste falsificate che potrebbero comportare lo svolgimento di attività illegali per conto di un utente (*falsificazione di richieste cross-site o CSRF*).
  - Blocca i tentativi di accesso indesiderati indovinando ripetutamente utenti e password utilizzando attacchi di *forza bruta*.
  - Attacchi *DDoS* (Distributed Denial of Service): riducono gli effetti degli sciami di traffico destinati a mettere a dura prova le capacità del servizio online.
- **Monitoraggio e filtraggio** del traffico: i WAF esaminano costantemente il traffico in entrata e in uscita per individuare tendenze o anomalie che potrebbero essere segnali di un attacco. Per garantire che le richieste dannose vengano bloccate prima che raggiungano l'applicazione web, analizzano sia il traffico HTTP che HTTPs.
- Gli **approcci analitici** basati su regole e comportamenti vengono utilizzati insieme dai WAF. Quando si rilevano modelli di attacco noti, le tecniche basate su regole utilizzano set di regole predeterminate, mentre l'analisi comportamentale cerca modelli di traffico sospetti in base al comportamento previsto dell'applicazione.
- **Registrazione e reporting**: conservando registri approfonditi del traffico online e degli incidenti di sicurezza, gli amministratori possono tenere d'occhio e valutare eventuali rischi o vulnerabilità. Per le esigenze di risposta agli incidenti e di conformità, queste informazioni sono utili.
- **Policy** e personalizzazione: i WAF possono essere personalizzati per soddisfare i requisiti particolari di un'applicazione web. Per bilanciare sicurezza e usabilità, gli amministratori possono progettare policy di sicurezza, impostare regole e modificare le impostazioni.

4.3 Modalità di utilizzo

Per raggiungere i propri obiettivi di sicurezza, i WAF possono utilizzare una varietà di strategie, tra cui metodi di blacklist, whitelist, basati su firma e basati su regole.

Il *blacklisting* è la modalità con la quale vengono bloccati i flussi di traffico che fanno match con almeno una delle regola della blacklist. Tutto il resto del traffico è permesso.

Il *whitelisting* è la modalità con la quale vengono ammessi i flussi di traffico che fanno match con almeno una delle regole riportate nella whitelist. Tutto il resto del traffico è bloccato.

Blacklisting e whitelisting sono due approcci speculari. E' facile, tuttavia, notare che l'approccio whitelisting è più sicuro in quanto previene falle di sicurezza legate a dimenticanze e/o errori da parte dell'operatore che configura il firewall.

Si riporta qualche esempio pratico per chiarire la differenza tra i due approcci presentati.

Whitelisting

- Whitelist IP Situazione: solo i dipendenti che lavorano negli uffici dell'azienda dovrebbero avere accesso al dashboard amministrativo di un sito di e-commerce. Whitelist: solo gli indirizzi IP collegati agli uffici dell'azienda possono accedere alla dashboard da parte del WAF. *L'accesso è negato a tutti gli altri indirizzi IP.*

- Whitelist Metodi HTTP Situazione:il modulo di contatto di un'applicazione web dovrebbe consentire solo richieste POST. Whitelist: il WAF implementa una regola che limita la capacità del modulo di contatto di gestire le richieste HTTP solo alle richieste POST, vietando tutti gli altri metodi HTTP.

Blacklisting

- Blacklist IP Situazione: un sito Web viene preso di mira da numerosi tentativi di accesso dannosi provenienti da un determinato indirizzo IP. Aggiungendo l'indirizzo IP offensivo a una blacklist, il WAF blocca l'accesso da quella fonte da quel momento in poi, lasciando concessi tutti gli altri accessi.

4.4 Possibili soluzioni per WAF

In questo capitolo studieremo l'analisi fatta circa le possibili proposte opensource di Web Application Firewalls.

Nello specifico esamineremo tre soluzioni: ModSecurity, SNORT e Squid.

In seguito all'analisi svolta, ModSecurity si è rivelata la scelta finale in quanto molto più adattabile alle nostre esigenze.

4.4.1 ModSecurity

ModSecurity: introduzione

Modsecurity è un Web Application Firewall open-source diventato oramai molto utilizzato in diverse organizzazioni in tutto il mondo. Modsecurity, abbreviato modsec, è nato come modulo per il server web Apache. Attualmente viene integrato principalmente con Apache ma può essere usato anche con altre soluzioni (come ad esempio NGINX, Microsoft IIS).

L'uso principale è quello comune a tutti i WAFs, protezione da attacchi di tipo web tra cui SQL injection, XSS, DDoS, eccetera.

Il sistema di rilevamento delle minacce usato da modsec è basato su Regole: esso utilizza un set di regole predefinite e/o personalizzate per identificare comportamenti sospetti.

Per configurare le regole di modsecurity si utilizza uno specifico linguaggio di configurazione basato sulle *SecRules*.

Di seguito si elencano le principali caratteristiche di ModSecurity:

- **Logging HTTP requests:** modsec permette di fare logging (cioè salva i dati) **completo** delle richieste/risposte http. Questa è una caratteristica che rende modsec diverso dagli altri WAFs poiché questi ultimi fanno il logging solo parziale (non salvano il body delle richieste, rendendo possibili attacchi tramite POST);
- **Monitoring real time;**
- **Whitelisting / Blacklisting:** può funzionare sia con un comportamento positivo che negativo;
- **Motore delle regole specifico:** possiede un proprio *rule-engine* che permette di scrivere regole secondo uno specifico linguaggio di configurazione. Questo consente di semplificare la sintassi di regole logicamente molto complesse;
- **Trasparente alla topologia di rete:** l'installazione di modsecurity non richiede nessuna modifica alla topologia di rete in cui viene configurato.

ModSecurity: funzionamento

Modsec basa il suo funzionamento su alcuni file di configurazione. Possono essere creati più file di configurazione diversi, con l'obiettivo di mantenerli separati in base alle funzioni, e poi includere quelli che si vogliono attivare nel file *main.conf*.

Nel seguito di questo paragrafo si descrive in parte il linguaggio di configurazione usato da modsec.

- SecRuleEngine

```
#si vuole attivare modsec in modalità logging:
#con "detectiononly" fa solo logging dei flussi di traffico
SecRuleEngine DetectionOnly



#per attivare modsec in modalità WAF:
SecRuleEngine On
```

- SecRequestBodyAccess e SecResponseBodyAccess

```
#Si vuole far analizzare a modsec anche il body delle richieste
#http, in questo modo si evitano attacchi che usano POST
SecRequestBodyAccess On

#allo stesso modo, se serve, è possibile attivare l'analisi del
#body anche per le risposte
SecResponseBodyAccess On
```

- SecDefaultAction

```
#si vuole operare in modalità whitelisting
#l'azione di default da configurare è DENY
#quando un pacchetto fa match con la regola ritorna il codice 406
SecDefaultAction "deny, log, status:406"
```

- SecRule: è la keyword per inserire una regola in modsec

```
#blocca l'apertura di /etc/passwd
SecRule REQUEST_URI "/etc/passwd" "id:50001"

#blocca tutte le richieste che nell'URL hanno ".."
SecRule REQUEST_URI "\.\./" "id:20002"

#blocca le richieste provenienti da uno specifico indirizzo IP
SecRule REMOTE_ADDR "^192\.168\.1\.250$" "id:50006"
```

REQUEST URI è la variabile che, inserita dopo *SecRule*, permette di analizzare l'intero URL e controllare se il pattern riportato dopo di essa fa match.

ModSecurity ha anche la possibilità di fare **time filtering** attraverso l'uso del set di variabili TIME = TIMEDAY, TIMEHOUR, TIMEMIN, TIMESEC, TIMEMON, ...;

```
#match con tutti i flussi di traffico
#tra il 24 e il 26 di ogni mese
SecRule TIME_DAY "^([2](4|5|6))$" "id:15"
```

Con questo termina l'introduzione al linguaggio di configurazione di modsecurity.

OWASP ModSecurity Core Rule Set

In questo capitolo vediamo cosa sono le OWASP CRS per ModSecurity [13].

L'OWASP CRS, noto anche come OWASP Core Rule Set, è una raccolta di linee guida e configurazioni di sicurezza con l'obiettivo di individuare e affrontare minacce e vulnerabilità comuni nelle applicazioni web.

Sotto la guida di OWASP, la comunità della sicurezza informatica ha sviluppato e mantiene queste linee guida come iniziativa open source. Forniscono agli sviluppatori e agli esperti di sicurezza delle applicazioni web risorse già pronte per difendere le applicazioni web da tipologie di attacchi frequenti.

Dopo aver installato modsec sul server web vanno integrati in esso i file di configurazione OWASP. Una volta scaricati i file di configurazione dal repository ufficiale OWASP, si possono integrare tramite la direttiva *SecRuleInclude*. Oltre a caricarle ed attivarle già pronte è possibile modificarle, con lo scopo di adattarle alle esigenze dell'applicazione web.

4.4.2 Snort

Snort: introduzione

Snort è un sistema di prevenzione delle intrusioni (*Network Intrusion Detection System, NIDS*) che comprende le funzionalità di un Web Application Firewall. Funziona analizzando il traffico di rete in tempo reale, identificando i pacchetti di dati che rappresentano una minaccia per il sistema e attuando misure di sicurezza per prevenire l'attacco.

Snort funziona analizzando il traffico di rete alla ricerca di firme o pattern che corrispondono a intrusioni o attacchi noti. Il workflow di Snort può essere così riassunto:

1. **Cattura del traffico:** acquisisce il traffico dal segmento di rete su cui viene installato e configurato.

2. **Analisi e rilevamento:** il traffico acquisito viene analizzato in tempo reale. I pacchetti catturati vengono confrontati con i pattern definiti. Se qualcuno di essi trova una corrispondenza viene generato un avviso e l'evento viene registrato.

3. **Azione:** in base a come viene configurato, Snort può anche intraprendere alcune azioni di risposta all'evento come il blocco immediato del traffico sospetto.

4.

Snort può operare in 3 modalità:

1. Sniffer mode: fa solo sniffing dei pacchetti riportandoli in output;
2. packet logger mode: salva i pacchetti che fanno match su disco;
3. NIDS (Network Intrusion Detection System)

Nel prossimo paragrafo esamineremo il funzionamento di Snort per quanto riguarda la modalità NIDS, ovvero quella che per funzionalità più si avvicina ad un WAF.

SNORT: funzionamento

Il funzionamento di Snort si basa sul file di configurazione *snort.conf*. In questo file vanno inserite le regole da applicare ai pacchetti con le relative azioni da prendere in caso di match.

Si mostrano di seguito le principali caratteristiche del linguaggio di configurazione di snort.

Le varibili var, portvar e ipvar rappresentano uno strumento fondamentale per la configurazione e dunque la creazione di regole in snort. Queste variabili consentono di definire in modo flessibile parametri che influenzano il comportamento di Snort durante l'analisi del traffico.

- **var:** la variabile *var* viene utilizzata per definire variabili generiche all'interno delle regole snort. Queste variabili possono rappresentare diversi tipi di dato: stringhe, numeri, indirizzi IP ecc.

```
#definire la variabile RULES_PATH che indica il percorso rules/
var RULES_PATH/
```

- **portvar:** la variabile *portvar* è progettata per la gestione delle porte di rete. Essa permette di definire una o un pool di porte che possono essere utilizzate all'interno delle regole.

```
#definisco il pool di porte MY_PORTS 22, 80 e da 1024 a 1050
#la negazione si fa con !port_number/list
portvar MY_PORTS [22,80,1024:1050]
```

- **ipvar:** la variabile *ipvar* è utilizzata per gestire singoli indirizzi IP o range di indirizzi. E' utile quando si desidera definire gruppi di indirizzi IP che possono essere usati all'interno delle regole, così da non dover riscrivere ogni volta un'intera lista di indirizzi.

```
#definisco il pool di indirizzi MY_NET da 192.168.1.0 a
#192.168.1.255 e da 10.1.1.0 a 10.1.1.255
#qui viene usata la notazione CIDR (si possono anche indicare
#indirizzi singoli o combinazioni CIDR / singoli)
ipvar MY_NET [192.168.1.0/24,10.1.1.0/24]
```

Si propone di seguito un esempio di regola in snort, con descrizione:

```
alert tcp any any -> $MY_NET $MY_PORTS (flags:S; msg:"SYN packet";)
#Questa regola viene utilizzata per identificare i pacchetti TCP con il
#flag SYN impostato.
```

Di seguito una spiegazione dei vari elementi della regola:

- "alert" indica che Snort deve generare un allarme se viene rilevata una corrispondenza con la regola.
- "tcp" specifica il protocollo che Snort deve analizzare.
- "any any" indica che Snort deve analizzare tutti gli indirizzi IP e le porte sorgente.
- "->" indica la direzione del traffico, in questo caso è il traffico in entrata (richieste),

<- indica traffico inbound/outbound. *N.B. in snort non è possibile fare una regola esclusivamente sull'outbound traffic.*

- "$MY_NET" e "$MY_PORTS" sono le variabili definite dall'utente viste precedentemente

che specificano l'indirizzo IP di rete e le porte che Snort deve monitorare con questa regola.

- "flags:S" indica che Snort deve cercare i pacchetti TCP con il flag SYN impostato.
- "msg: "SYN packet"" è un messaggio di allarme personalizzato che Snort genererà se

viene rilevata una corrispondenza con la regola. In questo caso, il messaggio è "SYN packet";

4.4.3 Squid

Squid: introduzione

Squid è un software di caching proxy che ha il compito di spezzare le richieste tra gli utenti e i server web. Quando un utente richiede una pagina web, Squid ne fa una copia e la memorizza nella cache per poi fornirla velocemente in caso di richieste successive. In questo modo si riduce il traffico sulla rete e si migliorano le prestazioni.

Per quanto riguarda gli aspetti di web filtering, Squid offre un framework di controllo degli accessi, che consente di limitare l'accesso ai contenuti web in base alle politiche di sicurezza dell'organizzazione. Grazie alle opzioni di autenticazione, Squid consente di verificare le credenziali degli utenti e garantire che solo quelli autorizzati possano accedere a determinati contenuti. Permette di configurare il file di log (caratteristica presente anche in modsecurity), che consente di registrare tutte le attività dei proxy e delle applicazioni web. Questo è particolarmente utile per monitorare l'utilizzo delle risorse web, individuare eventuali problemi di sicurezza e analizzare le prestazioni del sistema.

Il file di configurazione di squid è *squid.conf*, un file di testo che contiene tutte le opzioni di configurazione del proxy. Tramite squid.conf si riesce a configurare porte http, flussi di traffico in entrata e uscita ed altro.

Squid: funzionamento

Nel presente paragrafo si riportano le variabili principali usate in squid.conf e alcuni esempi di configurazione.

- access_control: questa opzione consente di definire le regole di accesso ai contenuti web, in base all'indirizzo IP dell'utente, al tipo di richiesta, all'URL richiesto e altro ancora.
- authentication: questa opzione consente di configurare l'autenticazione degli utenti, in base a diversi metodi come NTLM, Basic, Digest, LDAP e altri.
- cache_dir: questa opzione consente di definire la posizione e le caratteristiche della cache, come la dimensione massima, la politica di sostituzione dei dati e altro ancora.
- http_port: questa opzione consente di definire la porta su cui Squid ascolta le richieste HTTP dei client.
- refresh_pattern: questa opzione consente di definire le regole di aggiornamento della cache, in base alle caratteristiche del contenuto web, come l'URL, il tipo di file, la dimensione e altro ancora.

- logformat: questa opzione consente di definire il formato del file di log, in modo da registrare solo le informazioni rilevanti per la specifica attività di monitoraggio.

Tramite la keyword *acl* si possono definire insiemi di indirizzi, porte su cui applicare successivamente le regole.

```
acl trusted_ports port 21   #ftp port
acl trusted_ports port 80   #http port
```

Tramite *http_access + deny/allow* si definisce cosa è consentito e cosa no. In questo esempio utilizziamo un approccio whitelisting, blocchiamo tutto il traffico e consentiamo solo quello "trusted".

```
http_access allow trusted_ports
http_access deny all
```

Per fare URL filtering in squid bisogna creare una lista di indirizzi e/o pattern che si vogliono bloccare/consentire e, successivamente, applicare una regola su di essi come nel seguente esempio:

```
#blocca tutti gli URL contenuti nel file block_list.txt

acl block_list dstdomain "/etc/squid/block_list.txt"
http_access deny block_list
```

Per fare TIME filtering bisogna creare una ACL time come in questo esempio:

```
#blocca tutto il traffico del lunedì (M), mercoledì (W)
#e Venerdì (F) dalle ore 08:00 alle 13:00.
acl blocked_hours time MWF 08:00-13:00
```

4.4.4 ModSecurity vs Snort vs Squid

Terminata la descrizione di Modsecurity, Snort e Squid si prosegue, in questo paragrafo, analizzando i pro e i contro di ogni soluzione spiegando il perché, in conclusione, è stato scelto ModSecurity.

La scelta finale ricade su ModSecurity perché si presta meglio ai bisogni del framework VEREFOO. Esso permette di fare web filtering e time filtering in maniera semplificata rispetto ad altre soluzioni grazie al suo linguaggio di configurazione. Inoltre, tramite le OWASP CRS, permette di attivare file di configurazione già precompilati per la protezione contro la maggior parte degli attacchi web noti. Snort a differenza dei WAFs, essendo un NIDS, non lavora principalmente su HTTP(s). Questo rende il suo utilizzo non indicato

per lo scopo del presente lavoro di tesi. Squid è invece un proxy server open source che fornisce funzionalità di caching e controllo dell'accesso. Esso può essere configurato per filtrare il traffico di rete, bloccare il contenuto inappropriato e prevenire gli attacchi di rete. Non essendo nello specifico un WAF, risulta più complicata la sua configurazione per le esigenze di VEREFOO.

Capitolo 5

Modellizzazione dei Requisiti di Sicurezza

Nel seguente capitolo verrà presentata una descrizione del modello dei *Requisiti di Sicurezza della rete (NSRs, Network Security Requirements)*. Questi rappresentano il secondo input del framework VEREFOO, oltre al Service Graph. In particolare, gli NSRs rappresentano dei vincoli di connettività tra due *end-points*: se un end-point deve poter raggiungere un altro utilizzando almeno un percorso consentito nel grafico, allora deve avere la *condizione di raggiungibilità*; in caso contrario, deve avere la *condizione di isolamento*.

L'obiettivo di questo capitolo è fornire un modello per la creazione di regole di sicurezza che permettano di gestire le seguenti categorie di funzionalità:

- attacchi di tipo web (SQL injection, XSS, DDoS, ecc.);
- funzionalità di web filtering;
- funzionalità di time filtering;

Prima di proseguire con il prossimo paragrafo che analizza nel dettaglio il modello sviluppato occorre fare una premessa. Il modello è stato sviluppato in modo tale da disaccoppiare due diverse categorie di regole:

1. la prima che soddisfa, insieme, funzionalità di web filtering e di time filtering;
2. la seconda che soddisfa solo la protezione da attacchi web (tramite le OWASP crs).

5.1 Il modello

5.1.1 Funzionalità di web-filtering e time-filtering

Ogni requisito di sicurezza r $\in R$ è rappresentato dal seguente formato:

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

- Azione $\in$ {Allow, Deny} Allow equivale alla condizione di raggiungibilità. Consente il passaggio del traffico di rete da un end-point all'altro.

Deny equivale alla condizione di isolamento. Blocca il passaggio del traffico di rete da un end-point all'altro.

- IPsrc: è l'indirizzo IP sorgente della comunicazione per cui è definita la regola;
- IPdst: è l'indirizzo IP destinazione della comunicazione per cui è definita la regola;
- PORTsrc: è il numero di porta o il range di porte sorgente della comunicazione per cui è definita la regola;
- PORTdst: è il numero di porta o il range di porte destinazione della comunicazione per cui è definita la regola;
- HTTPmethod $\in$ {GET, HEAD, DELETE, POST, PUT, PATCH} è il request method http utilizzato nella comunicazione per cui è definita la regola;
- URL: è la stringa che rappresenta l'URL per cui è definita la regola;
- Domain: è la stringa che rappresenta il dominio per cui è definita la regola;
- TIMEstart: è la stringa che rappresenta il primo estremo (inizio) dell'intervallo di tempo per cui è definita la regola;
- TIMEend: è la stringa che rappresenta il secondo estremo (fine) dell'intervallo di tempo per cui è definita la regola;

La **wildcard *** ha il significato di *"doesn't care"*. I successivi esempi chiariscono in modo esaustivo i diversi casi di utilizzo di *.

- **IP** L'indirizzo 10.0.0.0/24 che, in notazione CIDR, rappresenta l'intervallo di indirizzi utilizzabili tra 10.0.0.0 e 10.0.0.255 può essere rappresentato con **10.0.0.***. Allo stesso modo il generico indirizzo x.y.0.0/16 può essere rappresentato con x.y.*.*.

- **PORT** Si può specificare una porta, un range di porte o tutte le porte (tramite la wildcard *).

```
"Blocca tutto il traffico diretto alla porta destinazione 8080"
other_fields=*, PORTdst="8080", Azione="deny"

"Accetta tutto il traffico in uscita dalle porte sorgente 1080-3128"
other_fields=*, PORTsrc="1080-3128", Azione="allow"
```

- Per quanto riguarda il **transport protocol** si dà per scontato che venga usato TCP.
- **HTTPmethod** E' possibile inserire più di un request method per volta, quindi nel caso in cui si volesse fare una regola simile ma con *GET* e *POST* basta inserire nel campo "URL" la stringa *"GET, POST"*. Anche nel campo HTTPmethod è consentito l'uso della wildcard *. Questo significa che, quando utilizzata, verrà creata una regola che fa match con tutti i 6 request methods supportati.
- **URI** Per fare filtering sull'URI si può specificare una stringa o una regex. Permette di valutare anche le query strings inserite per passare parametri dopo il "?" (es. /index.php? p=X). Dato che il modello permette l'inserimento nel campo URI sia di stringhe che di regex vi è la necessità di distinguere tra i due casi. Lo si fa utilizzando l'operatore *@rx* prima di inserire una espressione regolare. Altrimenti viene valutata una stringa normale.

```
"Blocca le richieste che hanno la stringa /examples/HacK.php nell'URI"
other_fields=*, URI="/examples/HacK.php", Azione="deny"


"Blocca le richieste il cui URI inizia per "hack""
other_fields=*, URI="@rx \^/hack", Azione="deny"
```

- **Domain** Questo campo della regola permette di considerare il dominio delle request lines valutate dal waf.

```
Esempio:
Viene passata la request line "http://www.example.com/index.php?p=X"
Si vuole bloccare tutto il traffico verso il dominio example.com:

other_fields=*, Domain="example.com", Azione="deny"
```

Per la descrizione degli attributi TIMEstart e TIMEend si riserva un paragrafo a parte.

5.1.2 Time filtering: utilizzo di TIMEstart e TIMEend

I due campi TIMEstart e TIMEend permettono di fare time filtering. Essi consentono, infatti, di definire l'intervallo di tempo (definito da inizio e fine) per cui la regola cercherà corrispondenza con i flussi di traffico. Durante il processo di modellizzazione di questi due campi sono state analizzate due soluzioni.

- Soluzione n.1

Sia TIMEstart che TIMEend devono corrispondere a valori temporali definiti. Soluzione semplice ma poco flessibile. Esempio:

```
# Blocca tutto il traffico ricevuto in qualsiasi giorno di agosto 2023
  other_fields=*, TIMEstart="2023:08:01:00:00:00",
  TIMEend="2023:08:31:23:59:59", Azione="deny"
```

- Soluzione n.2

Si potrebbe pensare ad una soluzione piu complessa sulla scia delle variabili di modsecurity: soluzione più complessa ma che permette maggiore flessibilità. Modsecurity prevede una sintassi con l'utilizzo degli operatori *@gt, @lt, @ge, @le*, i quali permettono di creare regole su orizzonti temporali non propriamente definiti, ad es. "il giorno 01 maggio 2023 *e tutto ciò che viene prima*". In questo modello, al posto degli operatori di modsecurity, viene fatto uso della wildcard * nel seguente modo: almeno uno dei due campi temporali (rispettivamente, TIMEstart e/o TIMEend) deve essere per forza definito da un intervallo temporale specifico (data, ora, ...). L'altro può invece utilizzare la wildcard *. Se definisco TIMEstart ed uso la wildcard * su TIMEend significa che voglio fare match con l'orizzonte temporale che va *da TIMEstart a tutto ciò che viene dopo* (equivalente a @ge). Se definisco TIMEend ed uso la wildcard * su TIMEstart significa che voglio fare match con l'orizzonte temporale che va *fino a TIMEend e tutto ciò che viene prima* (equivalente a @le).

Esempio:

```
# Blocca tutto il traffico che ricevi FINO AL 31 agosto 2023:
other_fields=*, TIMEstart=*, TIMEend="2023:08:31:23:59:59",
Azione="deny"
```

```
# Blocca tutto il traffico che ricevi A PARTIRE DAL 01 agosto 2023:
other fields=*, TIMEstart="2023:08:01:00:00:00", TIMEend=*,
Azione="deny"
```

5.1.3 Protezione da attacchi web

Nelle precedenti sezioni di questo capitolo abbiamo analizzato il modello delle policy per web e time filtering. In questa sezione analizziamo il modello per le protezioni contro gli attacchi di tipo web, tramite l'utilizzo del OWASP-crs. Il processo di attivazione di una regola owasp in modsecurity è il seguente:

1. si scarica il core rule set dal sito ufficiale;
2. si estraggono i file di configurazione (relativi a diversi tipi di attacchi) in una directory del web server;
3. si configura modsecurity in modo tale da utilizzare le regole inserendo una include dentro il file di configurazione: *Include "/path/to/coreruleset/*.conf"*

Il nostro modello:

L'utente ha la possibilità di attivare uno o più file di configurazione owasp tramite la seguente sintassi:

```
ACTIVATE conf_file
```

where *conf_file* ∈ *conf_files* = {DOS-PROTECTION", "SCANNER-DETECTION", "PROTOCOL-ENFORCEMENT", "PROTOCOL-ATTACK", "MULTIPART-ATTACK", "APPLICATION-ATTACK-LFI", "APPLICATION-ATTACK-RFI", "APPLICATION-ATTACK-RCE", "APPLICATION-ATTACK-PHP", "APPLICATION-ATTACK-NODEJS", "APPLICATION-ATTACK-XSS", "APPLICATION-ATTACK-SQLI", "APPLICATION-ATTACK-SESSION-FIXATION", "APPLICATION-ATTACK-JAVA", "BLOCKING-EVALUATION", "DATA-LEAKAGES", "DATA-LEAKAGES-SQL", "DATA-LEAKAGES-JAVA", "DATA-LEAKAGES-PHP", "DATA-LEAKAGES-IIS", "DATA-LEAKAGES"}

ciò significa che se si vogliono attivare più file di configurazioni si dovrà ripetere la keyword "ACTIVATE".

Esempio:

```
# attiva la protezione contro attacchi di tipo DDoS e SQLinjection
→ ACTIVATE sql.conf
→ ACTIVATE DOS.conf
```

Nella fase di traduzione dal linguaggio di medio livello a quello di basso livello verranno valutati i nomi dei file che sono stati inseriti dopo la keyword “ACTIVATE” e saranno riportati nella include all’interno del file di configurazione di modsecurity.

5.2 Rappresentazione XML

In questa sezione si vede com’è stato aggiornato il modello per la configurazione dei parametri di input del framework.

5.2.1 HSPL e MSPL

Prima di addentrarci nell’analisi del modello, facciamo chiarezza sui diversi livelli di linguaggio usati in questo lavoro di tesi.

Basato su un paradigma soggetto-verbo-oggetto-parametri, HSPL (High Security Policy Language) è un linguaggio utilizzato per la progettazione e l’astrazione di politiche di alto livello; ogni statement è caratterizzato da un soggetto che indica il soggetto che intende far rispettare questa esigenza di sicurezza della rete, un verbo che designa l’azione richiesta, e un oggetto target dell’azione, seguito da altri argomenti per offrire maggiori informazioni.

La caratteristica chiave di HSPL è che deve consentire la formulazione di espressioni semplici che gli utenti finali possano comprendere e utilizzare senza conoscenze specializzate o approfondite. L’idea infatti sarebbe quella di prevedere uno strumento ausiliario opzionale, quale ad esempio un elenco predeterminato di security policies che l’utente può selezionare.

MSPL (Medium Security Policy Language) è un linguaggio che richiede la capacità di fornire tutti i dati necessari per la configurazione delle funzioni di rete nella fase successiva. Di conseguenza, il suo pubblico target è la comunità tecnica, come il responsabile della sicurezza. In ogni caso, per evitare di essere associati a implementazioni particolari, il modo in cui le regole MSPL vengono presentate deve essere astratto e indipendente dalle loro specificità.

Dal linguaggio di medio livello si prevede, successivamente, di passare ad un Low-Level language specifico per la tecnologia utilizzata. Ciò consente di passare da un linguaggio definito per il framework e comprensibile all’utente ad un linguaggio di configurazione della specifica tecnologia utilizzata. Questo aspetto non è però trattato in questo lavoro di tesi. Ci limitiamo qui a descrivere il MSPL, illustrando gli schemi XSD che definiscono la struttura dei file di configurazione in XML passati come input.

5.2.2 Requisiti

In questo paragrafo analizziamo lo schema XSD dei requisiti di sicurezza passati in input.

```
<xsd:element name="PropertyDefinition">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Property" type="Property" minOccurs="1" maxOccurs="
       unbounded"/>
   <xs:element name="OWASPprop" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xsd:simpleContent>
      <xsd:extension base="xs:string">
        <xsd:restriction base="xs:string">
             <xsd:enumeration value="DOS-PROTECTION"/>
             <xsd:enumeration value="SCANNER-DETECTION"/>
             <xsd:enumeration value="PROTOCOL-ENFORCEMENT"/>
             <xsd:enumeration value="PROTOCOL-ATTACK"/>
             <xsd:enumeration value="MULTIPART-ATTACK"/>
         <xsd:enumeration value="APPLICATION-ATTACK-RFI"/>
         <xsd:enumeration value="APPLICATION-ATTACK-RCE"/>
             <xsd:enumeration value="APPLICATION-ATTACK-PHP"/>
             <xsd:enumeration value="APPLICATION-ATTACK-NODEJS"/>
             <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="DATA-LEAKAGES"/>
        </xsd:restriction>
       </xsd:extension>
      </xsd:simpleContent>
     </xs:complexType>
```

```
34      </xs:element>
35     <xsd:element name="IPsrc" type="xs:string"/>
36     <xsd:element name="IPdst" type="xs:string"/>
37    </xsd:sequence>
38   </xsd:complexType>
39 </xsd:element>
```

PropertyDefinition è l'elemento principale che permette di inserire un requisito di sicurezza. Esso può essere o di tipo *Property* **oppure** *OWASPprop*.

- Property è l'elemento che permette di inserire un requisito di web/time filtering;
- OWASPprop è l'elemento che consente l'inserimento di una regola OWASP per la protezione da attacchi web specifici. Nello schema XSD riportato precedentemente è presente la lista completa delle possibili OWASPprop che il framework mette a disposizione. Oltre alla stringa che identifica la regola da attivare vi sono altre due stringhe (*IPsrc e IPdst*) che permettono di identificare il flusso di traffico interessato dalla regola.

Di seguito si riporta lo schema XSD dell'elemento *Property:*

```
1 <xsd:complexType name="Property">
2         <xsd:choice>
3                 <xsd:element name="HTTPDefinition" type="HTTPDefinition" minOccurs="0"/
4                 <xsd:element name="POP3Definition" type="POP3Definition" minOccurs="0"/
5         </xsd:choice>
6         <xsd:sequence>
7         <xsd:element name="TIMEprop" minOccurs="0" maxOccurs="unbounded">
8         <xsd:complexType>
9                 <xsd:attribute name="timestart" type="xsd:string" use="required"/>
10                <xsd:attribute name="timeend" type="xsd:string" use="required"/>
11        </xsd:complexType>
12        </xsd:element>
13        </xsd:sequence>
14    <xsd:attribute name="name" type="P-Name" use="required"/>
15        <xsd:attribute name="graph" type="xsd:long" use="required"/>
16        <xsd:attribute name="src" type="xsd:string" use="required"/>
17        <xsd:attribute name="dst" type="xsd:string" use="required"/>
18        <xsd:attribute name="src_port" type="xsd:string"/>
19        <xsd:attribute name="dst_port" type="xsd:string"/>
```

```
20          <xsd:attribute name="isSat" type="xsd:boolean"/>
21          <xsd:attribute name="body" type="xsd:string"/>
22  </xsd:complexType>
```

Property, come anticipato, è l'elemento che ci permette di inserire regole per il web filtering ed il time filtering. Analizziamo nello specifico i campi:

- HTTPDefinition è una struttura che contiene a sua volta i campi per la definizione di una regola che agisce a livello di protocollo HTTP: url, domain, body, options e httpmethod.

```
1  <xsd:complexType name="HTTPDefinition">
2      <xsd:attribute name="url" type="xsd:string" />
3      <xsd:attribute name="domain" type="xsd:string" />
4      <xsd:attribute name="body" type="xsd:string" />
5      <xsd:attribute name="options" type="xsd:string" />
6          <xsd:attribute name="httpmethod" type="xsd:string" />
7  </xsd:complexType>
```

- POP3Definition segue lo stesso concetto di HTTPDefinition ma per il protocollo POP3. Data la presenza di *choice* nella definizione, HTTPDefinition e POP3Definition sono in una condizione di esclusività, se nella PropertyDefinition vi è uno, non può esserci l'altro e viceversa;
- TIMEprop è l'elemento che ci permette di fare filtering sul tempo. Esso è infatti costituito dai due campi: *timestart* e *timeend*. Entrambi sono in formato stringa e rappresentano, rispettivamente, istante di inizio e di fine;
- name è il nome che viene attribuito alla property;
- graph rappresenta il grafo di riferimento di questa property;
- src e dst sono, rispettivamente, l'indirizzo IP sorgente e destinazione;
- src_port e src_dst sono, rispettivamente, la porta sorgente e destinazione;
- isSat è un booleano che varrà *true* se il problema MaxSMT è soddisfatto, *false* altrimenti;

5.2.3 Schema di output

Il framework, sulla base degli input, calcola il risultato che, se necessario, prevederà uno o più WAF da inserire nella topologia di rete. In questo paragrafo analizziamo lo schema XSD del Web Application Firewall.

```
<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>
```

La struttura che si ottiene come output del processo (*web_application_firewall)* può essere di due tipi:

- la lista dei campi del WAF che verrà inserito nella topologia:

```
<xsd:element name="waf_elements">
    <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="id" type="xsd:long" minOccurs="0" />
          <xsd:element name="action" type="ActionTypes" minOccurs="0" default="DEN
          <xsd:element name="ipsrc" type="xsd:string" />
          <xsd:element name="ipdst" type="xsd:string" />
          <xsd:element name="portsrc" type="xsd:string" minOccurs="0" />
          <xsd:element name="portdst" type="xsd:string" minOccurs="0" />
          <xsd:element name="httpmethod" type="xsd:string" minOccurs="0" />
          <xsd:element name="domain" type="xsd:string" minOccurs="0" />
          <xsd:element name="url" type="xsd:string" minOccurs="0" />
          <xsd:element name="timestart" type="xsd:string" minOccurs="0" />
          <xsd:element name="timeend" type="xsd:string" minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

- **oppure** la stringa che rappresenta la regola owasp che richiesta (*owasp_rule)*.

Infine, il campo *DefaultAction* può assumere i seguenti valori:

- *ALLOW or DENY* nel caso di waf_elements;
- *ACTIVATE* nel caso di owasp_rule.

5.3 Esempi di configurazione

In questo paragrafo si mostrano alcuni esempi di configurazione per dare un'idea più chiara di quale sia l'output del framework VEREFOO dati determinati input.

Per i successivi esempi si prenda in considerazione il seguente grafo. Viene utilizzato direttamente un Allocation Graph (con gli Allocation Point già inseriti) anziché un Service Graph solo per una maggior chiarezza grafica.

```
<graphs>
        <graph id="0">
                <node functional_type="WEBCLIENT" name="10.10.10.10">
                        <neighbour name="1.1.1.1" />
                </node>

                <node functional_type="WEBCLIENT" name="20.20.20.20">
                        <neighbour name="2.2.2.2" />
                </node>

                <node functional_type="WEBSERVER" name="100.1.1.1">
                        <neighbour name="1.1.1.1" />
                        <neighbour name="2.2.2.2" />
                </node>

                <node name = "1.1.1.1">
                        <neighbour name="10.10.10.10" />
                        <neighbour name="100.1.1.1" />
                </node>

                <node name = "2.2.2.2">
                        <neighbour name="20.20.20.20" />
                        <neighbour name="100.1.1.1" />
                </node>
        </graph>
</graphs>
```

Esempio n.1 - fig. 5.2

- "Il client 10.10.10.10 non può fare richieste GET al server 100.1.1.1 all'URL *www.fakewebsite.com/hack*";
- "Il client 20.20.20.20 non può fare richieste POST al server 100.1.1.1

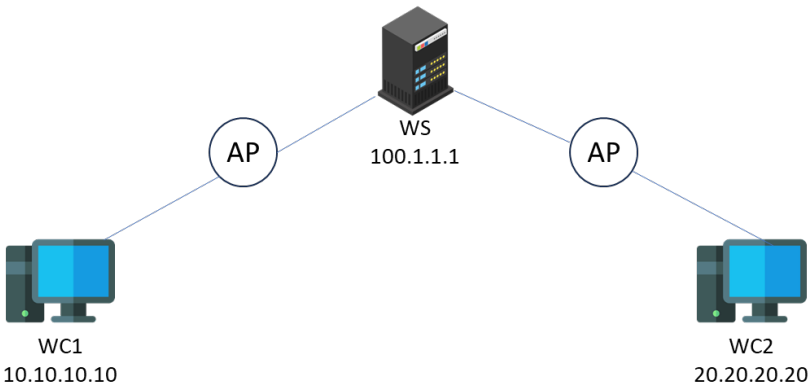

Figura 5.1. Rappresentazione grafica Allocation Graph per esempi di configurazione.

```
<PropertyDefinition>
<Property graph="0" name="IsolationProperty" src="10.10.10.10" dst="100.1.1.1">
   <HTTPDefinition httpmethod="GET" domain="fakewebsite.com" url="/hack" />
</Property>

<Property graph="0" name="IsolationProperty" src="20.20.20.20" dst="100.1.1.1">
    <HTTPDefinition httpmethod="POST" />
</Property>
</PropertyDefinition>
```

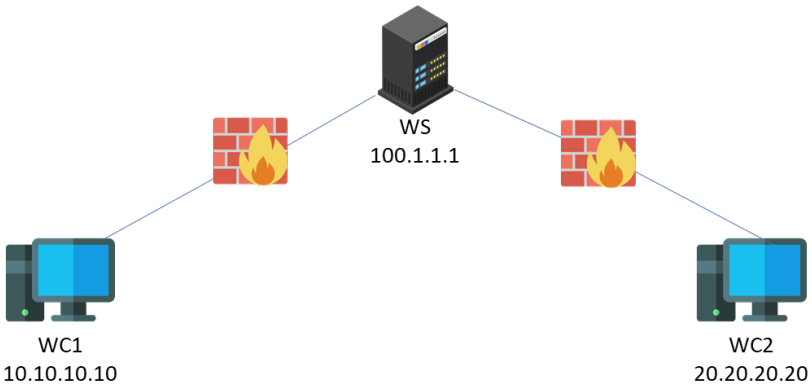

Figura 5.2. Esempio n.1: rappresentazione grafica dell'allocazione di WAF (web filtering).

Esempio n.2 - fig. 5.3

- Il client 10.10.10.10 non può fare richieste GET e POST al server 100.1.1.1 su URL che iniziano per “hack”.

```
    #uso di regular expressions per la definizione del campo URI

<PropertyDefinition>
        <Property graph="0" name="IsolationProperty" src="10.10.10.10" dst="100.1.1.1">
            <HTTPDefinition httpmethod="GET" domain="fakewebsite.com" url="@rx␣^/hack" />
            <HTTPDefinition httpmethod="POST" domain="fakewebsite.com" url="@rx␣^/hack" />
        </Property>
</PropertyDefinition>
```

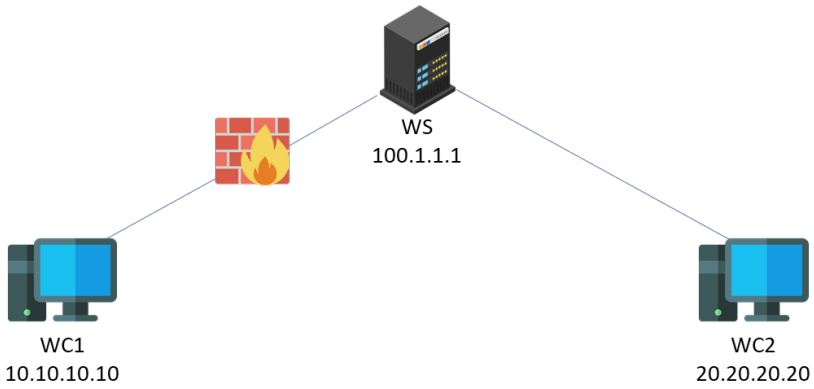

Figura 5.3. Esempio n.2: rappresentazione grafica dell’allocazione di WAF (web filtering).

Esempio n.3 - fig. 5.4

- Il client 20.20.20.20 non può fare richieste al server 100.1.1.1 nell’intervallo di tempo che va dal 01 agosto 2023 al 31 agosto 2023;
- Il client 10.10.10.10 non può fare richieste al server 100.1.1.1 dal 01 agosto 2023 **in poi**

```
# uso del time filtering con entrambi i tempi (start & end) definiti
# uso del time filtering con uno dei due estremi temporali
 (start OPPURE end) non definito

<PropertyDefinition>
```

```
<Property graph="0" name="IsolationProperty" src="20.20.20.20" dst="100.1.1.1">
    <TIMEprop timestart="2023:08:01:00:00:00" timeend="2023:08:31:23:59:59" />
</Property>
<Property graph="0" name="IsolationProperty" src="10.10.10.10" dst="100.1.1.1">
    <TIMEprop timestart="2023:08:01:00:00:00" timeend="*" />
</Property>
</PropertyDefinition>
```

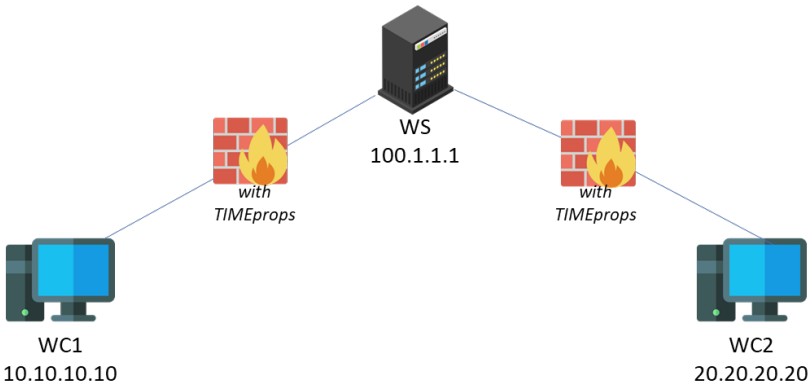

Figura 5.4. Esempio n.3: rappresentazione grafica dell'allocazione di WAF (time filtering).

Esempio n.4 - fig. 5.5

- Si vogliono includere i file di configurazione per modsecurity per la protezione di attacchi DoS e data leakage configurati rispettivamente sui flussi di traffico da WB1 a WS e da WB2 a WS.

```
# Protezione da attacchi utilizzando le OWASP-crs

<PropertyDefinition>
        <OWASPprop value = "DOS-PROTECTION" IPsrc = "10.10.10.10" IPdst = "100.1.1.1" /
        <OWASPprop value = "DATA-LEAKAGES" IPsrc = "20.20.20.20" IPdst = "100.1.1.1" />
</PropertyDefinition>
```

5.4 Modellizzazione problema MaxSMT

In 3.2 sono stati introdotti i problemi SMT e MaxSMT: cosa sono e le loro varianti. In questa sezione viene analizzato il modello in [6] in parte modificato per adattarlo al nuovo

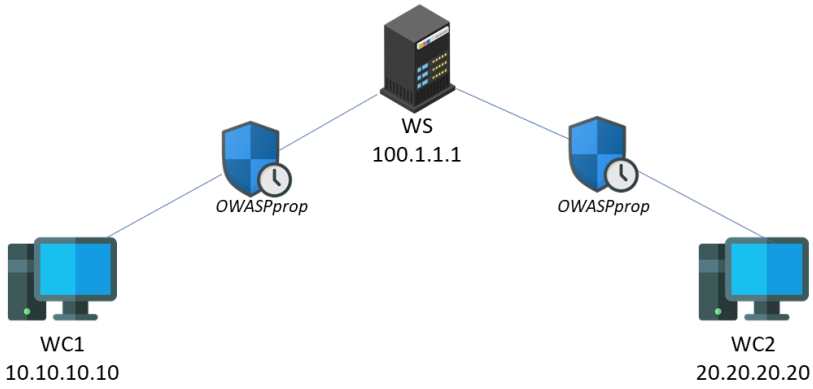

Figura 5.5. Esempio n.4: rappresentazione grafica dell'allocazione di WAF (OWASP properties).

modello proposto in questo lavoro di tesi.

Si riportano, per chiarezza, i formati delle regole per il modello di questa tesi:

```
#web & time filtering
IPsrc | IPdst | PORTsrc | PORTdst | HTTPmethod | URL | Domain | TIMEstart | TIMEend | ACT

#OWASP
IPsrc | IPdst | OWASPprop
```

5.4.1 Funzione match

La funzione match permette di verificare se un pacchetto fa match o meno con una specifica regola. Si prenda in considerazione la generica regola *"r"* e il generico pacchetto *"pk0"* all'istante che si sta analizzando. La funzione match ritorna *TRUE* se e solo se sono verificate *tutte* le condizioni riportate nella formula seguente:

$$
\begin{aligned}
r.match(pk0) \Leftrightarrow & pk0.IPsrc \subseteq r.IPsrc \\
& \wedge\ pk0.IPdst \subseteq r.IPdst \\
& \wedge\ pk0.PORTsrc \subseteq r.PORTsrc \\
& \wedge\ pk0.PORTdst \subseteq r.PORTdst \\
& \wedge\ pk0.HTTPmethod \subseteq r.HTTPmethod \\
& \wedge\ pk0.URI \subseteq r.URI \\
& \wedge\ pk0.Domain \subseteq r.Domain \\
& \wedge\ pk0.TIMEstart \subseteq r.TIMEstart \\
& \wedge\ pk0.TIMEend \subseteq r.TIMEend
\end{aligned}
$$

5.4.2 Modello per i Network Security Requirements

Si definisca R_S il set di NSRs per il web filtering e time filtering. Si ipotizzi che R_S sia conflict-free. Ogni $r \in R_S$ è formata dalla coppia *(C, a)*, dove C rappresenta la condizione per il match e a l'azione da eseguire in caso di match.

$$C = IPsrc, IPdst, PORTsrc, PORTdst, HTTPmethod, URI, Domain, TIMEstart, TIMEend$$

Si definisca Q_S il set di regole OWASP attivabili dal framework. Ogni $q \in Q_S$ è formata dalla coppia *(O, a)*, dove O rappresenta la precisa regola e a l'azione da eseguire che, in questo caso, è sempre la default action (vedasi dopo per dettagli su questo aspetto).

$$O = OWASPprop$$

Sia $f = [e_s, t_{s,a}, ..., t_{k,d}, e_d]$ un flusso di traffico generico, dove e_s rappresenta l'end-point sorgente, $t_{s,a}$ rappresenta il flusso di traffico tra l'end-point sorgente e il secondo nodo direttamente collegato ad esso, e_d rappresenta l'end-point destinazione dell'intero flusso di traffico e$t_{k,d}$ il flusso di traffico dal penultimo nodo all'end-point finale.

Di seguito si riportano le condizioni affinché il flusso f soddisfi il requisito C:

1)$\alpha(e_s) \subseteq C.IPsrc \wedge \alpha(e_d) \subseteq C.IPdst$
2)$t_{s,a} \subseteq (C.IPsrc, *, C.PORTsrc, *, *, *, *, *, *)$
3)$t_{k,d} \subseteq (*, C.IPdst, *, C.PORTdst, HTTPmethod, URI, Domain, TIMEstart, TIMEend$

α è la funzione che, dato un end-point come parametro, ritorna il suo indirizzo IP (o il range di indirizzi). Di conseguenza, la 1) verifica che gli indirizzi IP di sorgente e destinazione corrispondano, rispettivamente, agli IPsrc e IPdst della regola. La 2) e la 3) verificano, rispettivamente, che i flussi di traffico *dalla* sorgente e *alla* destinazione contengano nello specifico i campi riportati nella regola.

Simili considerazioni possono essere fatte per il requisito O. Di seguito si riportano le condizioni affinché $f = [e_s, t_{s,a}, ..., t_{k,d}, e_d]$ soddisfi il requisito O:

1)$\alpha(e_s) \subseteq C.IPsrc \wedge \alpha(e_d) \subseteq C.IPdst$

2)$t_{s,a} \subseteq (C.IPsrc, *)$

3)$t_{k,d} \subseteq (*, C.IPdst)$

Occorre precisare che le OWASP properties possono essere attivate anche su base di un flusso più dettagliato (cioè che tiene conto anche di metodo http, uri ecc.). In questo lavoro di tesi è stato creato un modello su una versione più semplice della configurazione mantenendo solo IP sorgente e destinazione che identificano il flusso di interesse.

Per quanto riguarda l'*azione a*, essa dipende dal comportamento di default adottato.

AZIONE PER IL REQUISITO C:

a = {ALLOW, DENY} default_behaviour = {blacklisting, whitelisting}

blacklisting → a=ALLOW

whitelisting → a=DENY

AZIONE PER IL REQUISITO O:

b = default_behaviour = {owasp_enforcement} → riguarda solo il processo di attivazione della X-regola

5.4.3 Algoritmo di calcolo per flussi massimali

Il concetto di flusso massimale permette di considerare un particolare set di flussi con lo scopo di semplificare l'analisi e rendere il calcolo più efficiente. Può essere preso in considerazione solo l'insieme dei flussi massimali che soddisfano $r.C$, il quale, pur risultando inferiore al set F_r, è ugualmente rappresentativo. Il sottoinsieme di F_r, noto come F_r^M (set di flussi massimali), è definito come comprendente *solo i flussi che non sono sottoflussi di nessun altro flusso in* F_r.

Passando ai flussi massimi si riduce il numero di diverse istanze che devono essere prese in considerazione e, di conseguenza, il numero di vincoli dati in ingresso al modello. In altre parole, ci sono meno flussi da prendere in considerazione. I flussi massimali hanno un altro vantaggio: la loro generazione avviene prima che venga formulato il problema di MaxSMT. Di conseguenza, le variabili che compongono il modello di flusso non sono

libere quando viene formulato il problema MaxSMT poiché sono già stati loro assegnati determinati valori. Mantenendo minimo il numero di variabili libere, lo spazio di ricerca del problema MaxSMT viene limitato e le prestazioni migliorano.

Prendendo in considerazione l'algoritmo 5.6 per il calcolo dei flussi massimali, ripreso da [6], vengono proposte di seguito alcune modifiche per adattarlo al presente lavoro di tesi sui WAF.

Algorithm 1. Computation of F_r^M

Input: a requirement r, and an AG G_A, **Output:** F_r^M

1: $F_r^M = \emptyset$
2: **for each** $p = [n_0, n_1, \ldots, n_{m+1}] \in paths(r, G_A)$ **do**
3: $F \leftarrow \{[n_0, t^r_{0,1}, n_1, \text{true}, n_2, \ldots, \text{true}, n_{m+1}]\}$
4: **for** $i = 1, 2, \ldots, m$ **do**
5: $F \leftarrow \{l + [b_i \wedge b'_i, n_i] + l' | l + [b_i, n_i] + l' \in F, b'_i \in \{\mathcal{I}_i^a, \mathcal{I}_i^d\}\}$
6: $F \leftarrow \{l + [b_i \wedge b'_i, n_i] + l' | l + [b_i, n_i] + l' \in F, b'_i \in \{\mathcal{D}_{ij}\}\}$
7: $F \leftarrow \{l + [b_i, n_i, b_{i+1} \wedge \mathcal{T}_i(b_i), n_{i+1}] + l' | l +$
$[b_i, n_i, b_{i+1}, n_{i+1}] + l' \in F\}$
8: $F' \leftarrow \{l + [t^r_{m,m+1} \wedge b_{m+1}, n_{m+1}] \mid l + [b_{m+1}, n_{m+1}] \in F\}$
9: **for** $i = m, m-1, \ldots, 1$ **do**
10: $F' \leftarrow \{l + [b_i \wedge \mathcal{T}_i^{-1}(b_{i+1}), n_i, b_{i+1}] + l' | l +$
$[b_i, n_i, b_{i+1}] + l' \in F'\}$
11: **if** $F \neq F'$ **then**
13: $F \leftarrow F'$
13: **goto** line 4
14: $F_r^M \leftarrow F_r^M \cup F$
15: **return** F_r^M

Figura 5.6. Algoritmo di calcolo per flussi massimali.

L'insieme di percorsi iniziali è formato da tutti i possibili paths che hanno $e_s = n_0$ come nodo iniziale e $e_d = n_{m+1}$ come nodo finale, dove e_s ha come indirizzo IP uno di quelli contenuti in $r.C.IPsrc$ e e_d ha come indirizzo IP uno di quelli contenuti in $r.C.IPdst$.

$$t^r_{0,1} = (\alpha(n_0) \wedge r.C.IPsrc, *, r.C.PORTsrc, *, *, *, *, *, *)$$
$$t^r_{m,m+1} = (*, \alpha(n_{m+1}) \wedge r.C.IPdst, *, r.C.PORTdst, HTTPmethod, URI,$$
$$Domain, TIMEstart, TIMEend)$$

5.4.4 Allocazione e configurazione firewall

$1. \forall a_h \in A_A . Soft(allocated(a_h) = false, c_h)$

Questa condizione permette di minimizzare il numero di firewalls allocati dal framework. La condizione è del tipo *Soft(f, c)*, dove f è una funzione e c è il peso associatole. Infatti,

dal momento che un firewall può essere allocato in qualsiasi AP, bisogna inserire una condizione per cui in ogni AP è preferibile che non venga allocato nessun firewall (condizione migliore).

Potrebbe essere necessaria l'allocazione di un firewall, in un dato AP, solo per per settare una o più regole OWASP. Anche in questo caso vale lo stesso concetto di minimizzazione basato sui soft constraints.

La minimizzazione del numero di regole calate in ogni firewall dipende dall'azione di default che viene impostata. Per questo motivo, nella decisione della regola di default, si applica la seguente logica:

Si consideri z il numero di *reachability* requirements e y il numero di *isolation* requirements. Se $z > y$ l'azione di default viene impostata ad "$ALLOW$" , altrimenti a "$DENY$". In questo modo si riduce il numero di regole da dover calare. Nel caso in cui z sia uguale a y, l'azione di default è indifferente da un punto di vista del numero di regole ma un approccio di tipo whitelisting è sempre più sicuro.

Capitolo 6

Implementazione e Validazione

Come si evince dai precedenti capitoli, l'analisi è stata svolta considerando due gruppi di funzionalità diversi: *web e time filtering* da un lato, *owasp-rules protection* dall'altra.

La fase di implementazione, descritta nel seguente capitolo, è stata svolta per quanto riguarda la sola protezione da attacchi web tramite regole Owasp.

6.1 Implementazione

6.1.1 Requisiti per l'implementazione

In questa sezione si introducono i requisiti necessari per l'implementazione, prima di procedere con l'analisi delle fasi con cui sono forniti gli input e l'ouput che da essi verrà generato.

Il sistema prevede due tipi di dati in ingresso:

- un Service Graph (SG), con cui si fornisce la topologia di rete su cui devono essere inserite le funzioni di sicurezza (in questo caso trattasi di web-application-firewall);
- N requisiti di sicurezza, con cui vengono fornite le specifiche di sicurezza che devono essere soddisfatte tramite l'inserimento dei WAF e delle regole specifiche per ognuno di essi.

6.1.2 Il processo

Forniti gli input del sistema, il framework provvede a calcolare il numero di WAF necessari e la loro posizione scegliendo tra gli Allocation Point del nuovo Allocation Graph.

Per chiarire meglio il concetto si propone di seguito un esempio. Partendo dalla seguente definizione di una proprietà:

```
<PropertyDefinition>
    <Property>
        <EnumList>DOS-PROTECTION</EnumList>
        <graph>0</graph>
        <isSat>true</isSat>
        <src>10.0.0.2</src>
        <dst>20.0.0.1</dst>
    </Property>
</PropertyDefinition>
```

La proprietà prevede l'inserimento della regola owasp per la protezione da attacchi di tipo "Denial of Service". Il grafo di riferimento su cui va applicata la regola è specificato dal numero 0. Il flusso di riferimento su cui va applicata la regola è quello identificato da IP sorgente e IP destinazione.

La prima cosa che il framework fa è quella di creare i vincoli da passare al modulo per l'ottimizzazione (corrispondente a Z3).

Il vincolo è del tipo *isolation-requirement* ed è associato ad un singolo flusso caratterizzato dalla quadrupla *IPsrc - PORTsrc - IPdst - PORTdst*.

A questo punto sarà il solver Z3 ad occuparsi di trovare una soluzione che soddisfi tutti i vincoli "hard" e che fornisca la garanzia di correttezza.

Si possono presentare due casi:

- Z3 riesce a soddisfare i vincoli *hard*: viene fornito in output un messaggio di soddisfacibilità ed il grafo completo con le nuove funzioni di sicurezza incluse;

- Z3 non riesce a soddisfare i vincoli *hard*: viene fornito in output un messaggio di non soddisfacibilità.

6.2 Validazione

In questa sezione verranno presentati alcuni casi d'uso per mostrare come il framework si comporta in base a diversi dati in input.

Il primo esempio è molto semplice. I dati passati in input sono volutamente minimali per esprimere il concetto base.

Successivamente verranno proposti esempi più complessi per mostrare il workflow del framework in contesti anche molto diversi tra loro.

6.2.1 Caso d'uso n.1

Si prenda come esempio l'allocation graph in fig. 6.1.

La proprietà in input è la seguente:

```
<PropertyDefinition>
    <Property>
        <EnumList>DOS-PROTECTION</EnumList>
        <src>10.10.10.10</src>
        <dst>100.1.1.1</dst>
    </Property>
</PropertyDefinition>
```

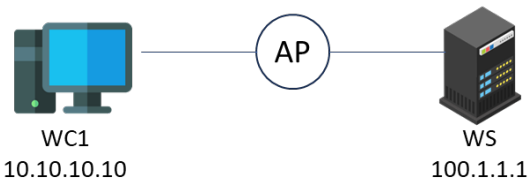

Figura 6.1. Allocation Graph per caso d'uso implementazione n.1

A partire dall'AG in fig. 6.1 e dalla proprietà passata in ingresso il requisito può essere soddisfatto e vi è solo una possibile soluzione.

E' presente infatti un solo Allocation Point ed una sola regola da soddisfare. Il framework provvederà dunque ad inserire n.1 *WAF* e n.1 *rules*.

Figura 6.2. Grafo per caso d'uso implementazione n.1

Come mostrato in figura 6.2, in questo caso viene inserito un firewall nell'unico AP disponibile e viene inserita in esso la proprietà di isolamento per la protezione da attacchi di tipo Denial of Service.

6.2.2 Caso d'uso n.2

Si prenda come esempio l'allocation graph in fig. 6.3 e la seguente proprietà:

```
<PropertyDefinition>
    <Property>
        <EnumList>DATA-LEAKAGES</EnumList>
        <src>10.10.10.10</src>
        <dst>100.1.1.1</dst>
    </Property>
    <Property>
        <EnumList>APPLICATION-ATTACK-RFI</EnumList>
        <src>20.20.20.20</src>
        <dst>100.1.1.1</dst>
    </Property>
</PropertyDefinition>
```

Per semplificare la rappresentazione dell'Allocation Graph si riporta un solo Allocation Point.

Dagli input riportati, ci sono due soluzioni possibili. Si riportano di seguito le due soluzioni.

1. Soluzione possibile ma non ottima La soluzione, presentata tramite il grafo in figura 6.4, prevede l'inserimento di n.2 WAF.

   Nel firewall di sinistra (sul link del web-client 10.10.10.10) verrà inserità l'apposita regola di protezione per attacchi di tipo *data-leakages*.

   Nel firewall di destra (sul link del web-client 20.20.20.20) verrà inserità l'apposita regola di protezione per attacchi di tipo *remote file inclusion*.

2. soluzione ottima La soluzione, presentata tramite il grafo in fig. 6.5, prevede l'inserimento di n.1 WAF in un link comune tra i due webclient. In questo modo è possibile inserire entrambe le regole, per i due diversi webclient, in un singolo firewall.

   In realtà esiste una soluzione analoga a questa che prevede l'inserimento del firewall sul link direttamente connesso al webserver. Tuttavia, tale soluzione non viene considerata in quanto, come già detto, risulta essere analoga a questa.

L'ultima soluzione proposta è dunque quella ottima. In quanto permette di ottimizzare il numero di firewall inserendone uno anziché due.

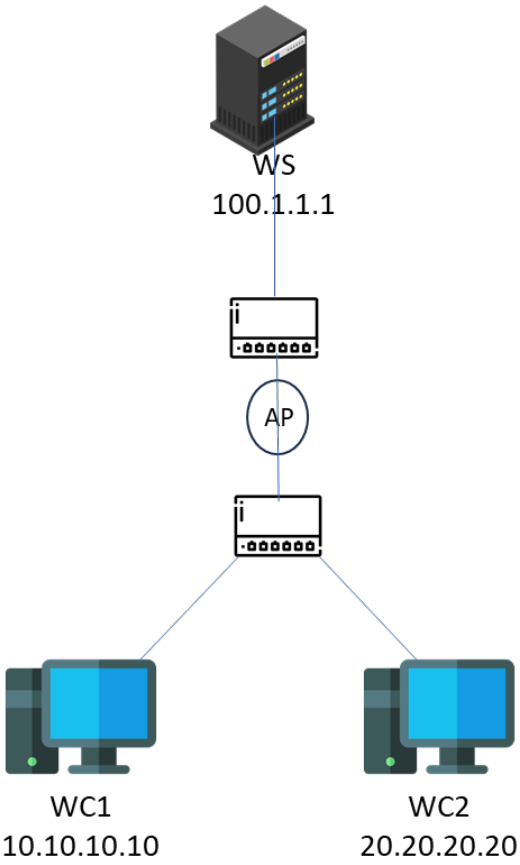

Figura 6.3. Allocation Graph per caso d'uso implementazione n.1

6.2.3 Caso d'uso n.3

Si prenda come esempio il service graph in fig. 6.6 e la seguente proprietà:

```
<PropertyDefinition>
    <Property>
        <EnumList>DATA-LEAKAGES</EnumList>
        <src>10.10.10.10</src>
        <dst>100.1.1.1</dst>
    </Property>
    <Property>
        <EnumList>APPLICATION-ATTACK-RFI</EnumList>
        <src>20.20.20.20</src>
        <dst>100.1.1.1</dst>
```

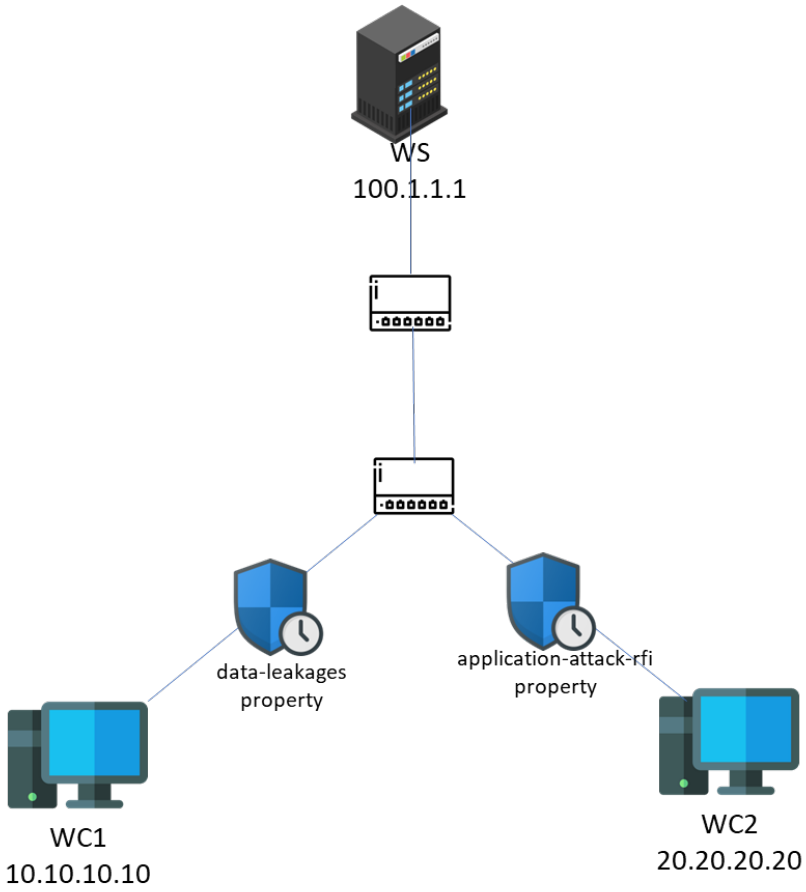

Figura 6.4. Grafo per caso d'uso implementazione n.2 - soluzione non ottima

```
11      </Property>
12      <Property>
13          <EnumList>APPLICATION-ATTACK-PHP</EnumList>
14          <src>30.30.30.30</src>
15          <dst>100.1.1.1</dst>
16      </Property>
17      <Property>
18          <EnumList>APPLICATION-ATTACK-PHP</EnumList>
19          <src>30.30.30.30</src>
20          <dst>200.2.2.2</dst>
21      </Property>
22      <Property>
```

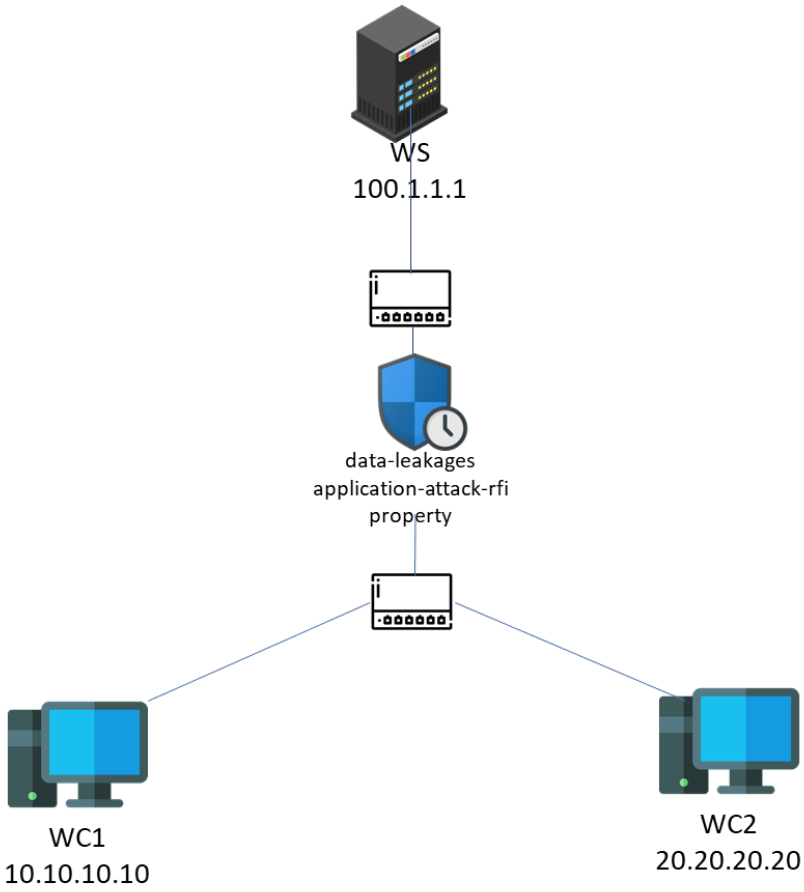

Figura 6.5. Grafo per caso d'uso implementazione n.2 - soluzione ottima

```
23          <EnumList>DOS-PROTECTION</EnumList>
24          <src>40.40.40.40</src>
25          <dst>200.2.2.2</dst>
26      </Property>
27  </PropertyDefinition>
```

Similmente allo *usecase n.2*, in questo caso si riporta il service graph, anziché l'allocation graph, in modo tale da rendere la rappresentazione più snella e semplice da leggere.

In questo scenario si ha n.2 webservice a cui potrebbero essere indirizzati i vari flussi di traffico. Questo vuol dire che i webclient presenti nella rete potrebbero essere soggetti, come in questo esempio, a delle regole (uguali o diverse) in base al webserver di riferimento.

Di seguito si propone, in fig. 6.7, una delle possibili soluzioni corrette ma non ottimali:

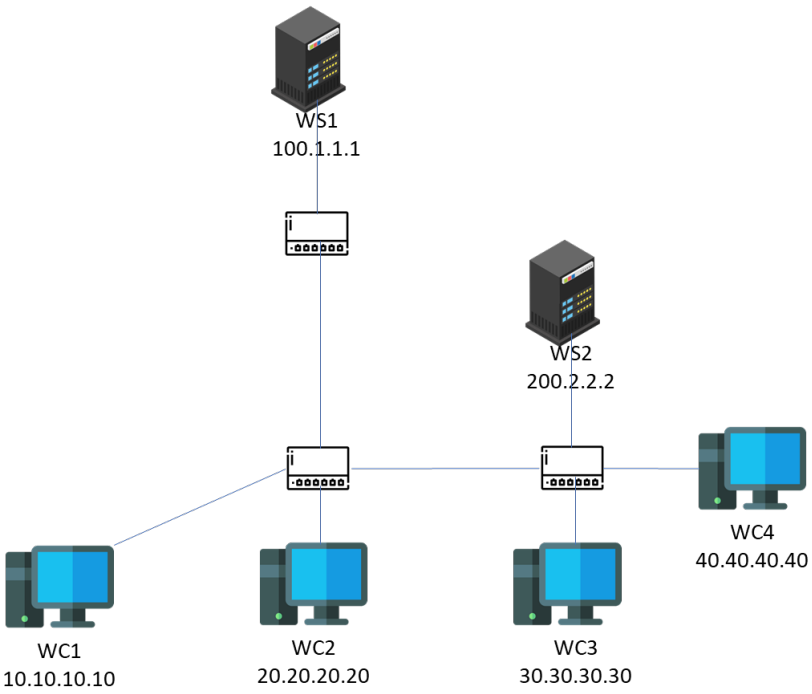

Figura 6.6. Service Graph per caso d'uso implementazione n.1

Tale soluzione prevede l'inserimento di n.4 Web Application Firewall, dividendo su di essi le regole necessarie per i n.4 webclient.

La soluzione ottima prevede invece l'inserimento di soli due Web Application Firewall, come mostrato di seguito in figura 6.8.

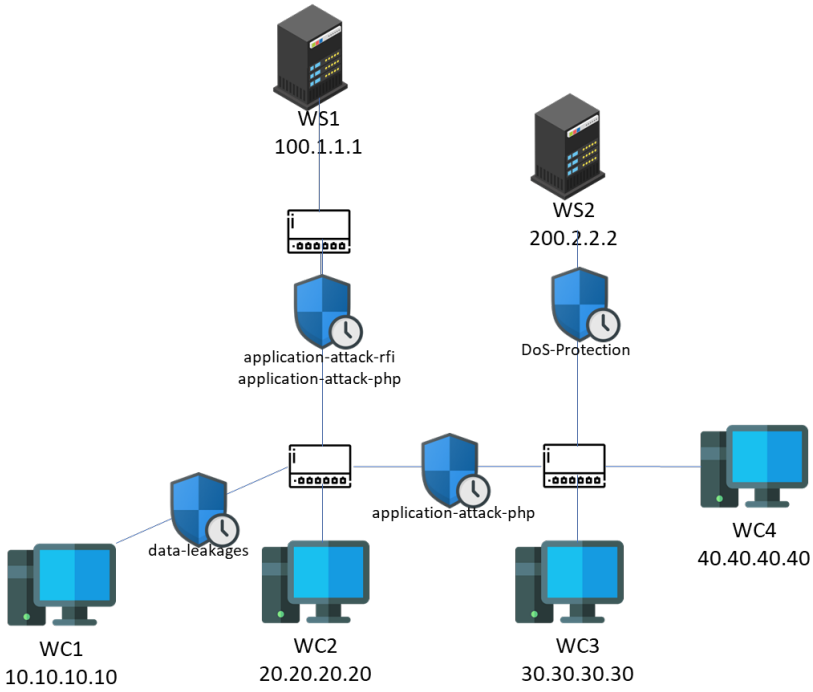

Figura 6.7. Grafo per caso d'uso implementazione n.3 - soluzione non ottima

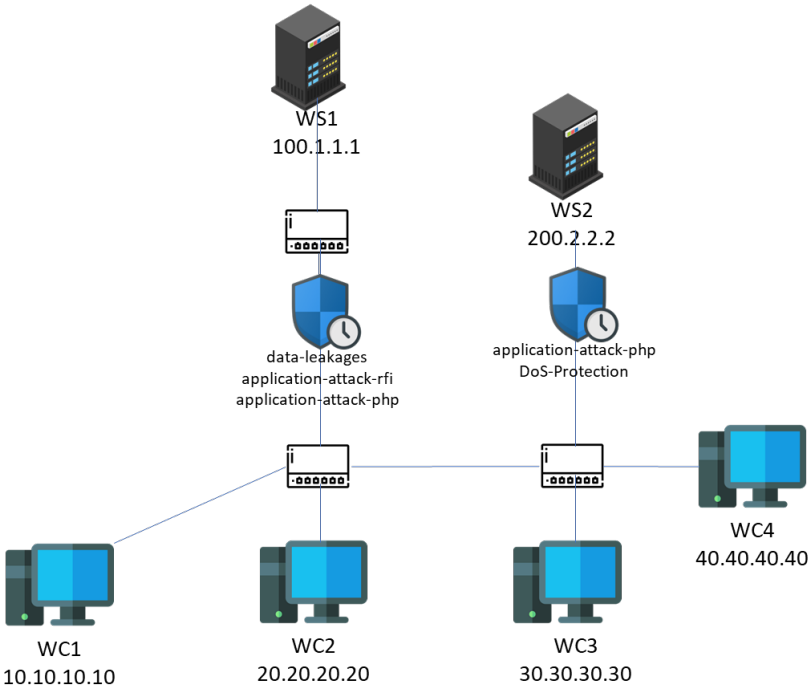

Figura 6.8. Grafo per caso d'uso implementazione n.3 - soluzione ottima

Capitolo 7

Conclusione e lavori futuri

Il lavoro di tesi presentato in questo documento si è concentrato sull'estensione delle funzionalità del framework VEREFOO. L'obiettivo principale è stato quello di estendere il framework con funzionalità di web filtering e protezione da attacchi web.

Il lavoro può essere considerato ancora in fase iniziale. La maggior parte del carico di lavoro è stata dedicata difatti allo studio delle principali soluzioni WAF in commercio. Questo studio preliminare è risultato necessario per le successive scelte progettuali.

Nello specifico, lo studio si è concentrato sull'analisi del problema: voler estendere il framework al livello applicativo (http) così da poter lavorare anche su filtraggio web. Per prima cosa sono state analizzate le funzionalità dei principali WAF in commercio. Questo è stato fatto in ottica progettuale così da permettere l'interazione futura di VEREFOO con altri strumenti (nello specifico caso di questa tesi è stato usato ModSecurity).

Successivamente è stato sviluppato il modello, definendo il formato degli attributi del sistema (le regole di sicurezza). Questo è stato fatto prendendo già in considerazione le scelte progettuali fatte in precedenza, ovvero l'uso futuro di modsecurity. Senza queste prerogative, infatti, il modello sarebbe stato astratto e difficilmente applicabile senza ulteriori modifiche ed adattamenti futuri.

Dalla definizione del modello delle regole è stato definito l'MSPL (medium-level security policy language), ovvero il linguaggio di medio livello utilizzato per definire le policy di sicurezza in VEREFOO. Per raggiungere tale obiettivo sono stati modificati gli schemi XSD che definiscono il modello del MSPL in VEREFOO.

Infine si è passati alla fase di sviluppo del software. Qui ci si è concentrati su una parte della modellizzazione fatta in precedenza. E' stato sviluppato infatti una versione preliminare del WAF concepito in fase di modelling, la quale permette l'inserimento di regole OWASP (simili a requisiti di isolamento) su flussi di traffico determinati dalla tupla *(IPsrc, IPdst, PORTsrc, PORTdst)*.

Il lavoro, come detto finora, è preliminare e non completo. Il lavoro futuro da svolgere è relativo al completamento delle funzionalità previste e progettate in fase di modellizzazione.

Bibliografia

[1] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza. «Automation for network security configuration: state of the art and research trends». In: *ACM Computing Surveys 56 (3), 1-37* (2023) (cit. a p. 13).

[2] Daniele Bringhenti, Riccardo Sisto, Fulvio Valenza. «Towards Security Automation in Virtual Networks». In: *IEEE 9th International Conference on Network Softwarization* (2023) (cit. a p. 13).

[3] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza. «A novel approach for security function graph configuration and deployment». In: *IEEE 7th International Conference on Network Softwarization* (2021) (cit. a p. 13).

[4] Daniele Bringhenti, Riccardo Sisto, Fulvio Valenza. «A novel abstraction for security configuration in virtual networks». In: (2023) (cit. a p. 13).

[5] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, Jalolliddin Yusupov. «Automated optimal firewall orchestration and configuration in virtualized networks». In: *IEEE Network Operations and Management Symposium* (April 2020) (cit. a p. 13).

[6] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, Jalolliddin Yusupov. «Automated firewall configuration in virtual networks». In: *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING* (2022) (cit. alle pp. 13, 52, 56).

[7] Daniele Bringhenti, Fulvio Valenza. «Optimizing distributed firewall reconfiguration transients». In: (2022) (cit. a p. 13).

[8] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, Jalolliddin Yusupov. «Introducing programmability and automation in the synthesis of virtual firewall rules». In: *6th IEEE Conference on Network Softwarization* (2020) (cit. a p. 13).

[9] Daniele Bringhenti, Riccardo Sisto, Fulvio Valenza. «A demonstration of VEREFOO: an automated framework for virtual firewall configuration». In: *IEEE 9th International Conference on Network Softwarization* (2023) (cit. a p. 13).

[10] Daniele Bringhenti, Riccardo Sisto, Fulvio Valenza. «Automating the configuration of firewalls and channel protection systems in virtual networks». In: *IEEE 9th International Conference on Network Softwarization* (2023) (cit. a p. 13).

[11] Daniele Bringhenti, Lucia Seno, Fulvio Valenza. «An Optimized Approach for Assisted Firewall Anomaly Resolution». In: *IEEE Access 11* (2023) (cit. a p. 13).

[12] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, Jalolliddin Yusupov. «Towards a fully automated and optimized network security functions orchestration». In: *IEEE International Conference on Computing* (October 2019) (cit. a p. 13).

[13] OWASP. «OWASP ModSecurity Core Rule Set». In: () (cit. alle pp. 13, 33).

[14] Cataldo Basile, Fulvio Valenza, Antonio Lioy, Diego R. Lopez, Antonio Pastor Perales. «Adding Support for Automatic Enforcement of Security Policies in NFV Networks». In: *IEEE-ACM TRANSACTIONS ON NETWORKING* (September 2019) (cit. a p. 13).

[15] Fulvio Valenza, Antonio Lioy. «User-oriented Network Security Policy Specification». In: *JOURNAL OF INTERNET SERVICES AND INFORMATION SECURITY* (May 2018) (cit. a p. 13).