



**Politecnico
di Torino**

Master of Science in Computer Engineering

Master Degree Thesis

Prova

Supervisors

prof. Fulvio Valenza
prof. Name Surname
dott. Name Surname
dott. Name Surname

Candidate

Benito MARRA

ACADEMIC YEAR 2023-2024

Summary

Text of the summary

Acknowledgements

Acknowledgement (optional)

Contents

List of Figures	6
List of Tables	7
Listings	8
1 Introduzione	10
1.1 Obiettivo della Tesi	10
1.2 Descrizione della tesi	11
2 Network Security Automation e Verefoo	13
2.1 Service Function Chain	13
2.2 Introduzione a Verefoo	15
2.2.1 Descrizione del modello	15
2.3 Definizione ed esempi delle proprietà di sicurezza	18
3 Docker	19
3.1 Introduzione a Docker	19
3.2 Docker Compose	19
4 Conclusions	20
Bibliography	21

List of Figures

2.1	Esempio di Service Function Chain	14
2.2	Architettura di VEREFOO [1]	16
2.3	Esempi di Service (in alto) e Allocation(in basso) Graphs [1]	17

List of Tables

Listings

2.1	Definizione di una proprietà di sicurezza generica	18
-----	--	----

Chapter 1

Introduzione

1.1 Obiettivo della Tesi

Nell'ultimo decennio diverse tecnologie di rete si sono sviluppate, creando reti sempre più robuste ed efficienti. In questo scenario, una tecnologia in particolare, la "*Network Function Virtualization*"(NFV) ha reso possibile creare delle reti che svolgono le funzioni di sicurezza e trasporto dei dati tramite la virtualizzazione di quest'ultime. Ciò ha permesso di definire le "*Software Defined Network*"(SDN) che introducono la possibilità di controllare le operazioni di rete tramite software.

Sfruttando queste tecnologie è stato sviluppato Verefoo(VERified REfinement and Optimized Orchestration) cioè un framework in grado di riuscire ad implementare nei nodi della rete i vari "*Network Security Requirements*"(NSR) in una topologia predefinita e fornita come input al framework.

Gli obiettivi di questa tesi possono essere riassunti nei 3 punti:

1. Sistemare e migliorare una demo precedentemente sviluppata per mostrare le potenzialità del framework. All'interno di questa demo il framework può accettare solo un determinato requisito di sicurezza di rete ovvero la *Protection Property* cioè la possibilità di far passare il traffico crittografato da un nodo ad un altro della topologia in maniera sicura. Per fare ciò verranno allocate nella topologia dei VPN Gateway in grado di crittare e decrittare il traffico in ingresso ed in uscita.
2. Cercare una soluzione per poter integrare le varie versioni di Verefoo. Prima del lavoro che verrà spiegato più avanti il framework era in grado, in un'unica esecuzione, di allocare solo VPN Gateway oppure di allocare solo FireWall per garantire la *Isolation Property* e la *Reachability Property*, mentre l'obiettivo che si cerca di raggiungere è di riuscire ad allocare contemporaneamente in un'unica esecuzione del framework sia i VPN Gateway che i Firewall necessari al corretto funzionamento della rete.
3. Produrre una nuova demo, diversa da quella del punto 1 in grado di mostrare le nuove capacità del framework. La demo prodotta dovrà essere in grado di poter allocare, in un'unica esecuzione tutti i Firewall e i VPN Gateway definiti dalle proprietà di rete che sono stati forniti come input al framework.

Da questo momento in poi per tutto il documento mi riferirò alla demo da modificare come Demo-A mentre alla demo da implementare come Demo-B.

1.2 Descrizione della tesi

Dopo aver spiegato nel Capitolo [1] gli obiettivi ed il lavoro prodotto per raggiungerli, il resto della tesi è definito nel seguente modo:

- Nella prima parte del Capitolo [2] si introduce il problema della Network Security Automation e si descrive il framework di Verefoo, ponendo particolare attenzione sul suo funzionamento ad alto e basso livello. Nella seconda parte sono descritte le definizioni delle Proprietà di sicurezza da passare come input al framework, con una spiegazione dettagliata di come queste intervengono nella definizione della topologia finale che verrà fornita come output dal framework. Infine verranno introdotti i grafi che verefoo richiede ed utilizza nella computazione dei vari *NSF*.
- Il Capitolo [3] definisce l'architettura di docker, specificando la differenza tra usare docker per la virtualizzazione e delle semplici macchine virtuali. Successivamente viene fatto un approfondimento sul docker-compose, un tool in grado di poter istanziare più container velocemente tramite script. Nella parte finale viene spiegato come effettuare il networking sui container istanziati, come definirlo tramite docker-compose e come testare le comunicazioni in modo efficiente.
- Il Capitolo [4] descrive i lavori svolti nella Demo-A. Inizialmente viene descritto tramite pezzi di codice lo sviluppo dell'installer prodotto affinché un qualsiasi utente possa utilizzare la demo in maniera pratica ed agile. Nei paragrafi successivi vengono evidenziati i punti critici incontrati, elencando le modifiche apportate affinché essa possa funzionare correttamente. Nell'ultimo paragrafo verranno specificati ulteriori upgrade che si possono inserire nella demo per mettere in mostra in maniera ancora più evidente il lavoro svolto da Verefoo.
- Il Capitolo [5] contiene un breve approfondimento degli obiettivi 2 e 3 della tesi con una descrizione accurata dei vari passi che sono stati svolti prima della soluzione definitiva. In questo capitolo si evidenziano anche le difficoltà che sono emerse lavorando al framework, e verranno proposte alcune soluzioni per poter evitare simili problematiche in futuro.
- Il Capitolo [6] descrive la soluzione finale scelta ed implementata su Verefoo. In questo capitolo viene quindi spiegato, anche tramite frammenti di codice, gli step che il framework eseguirà per produrre in output una rete che soddisfi contemporaneamente tutti i requisiti di sicurezza passati come input.
- Il Capitolo [7] descrive lo sviluppo della Demo-B. In un primo momento si mostra la topologia di rete scelta da virtualizzare, con la finalità di indicare

le nuove funzionalità di verefoo sviluppate al completamento del secondo obiettivo della tesi. Successivamente vengono descritti tutti i passi svolti per implementare la demo, con un commento per il codice che è stato utilizzato. Infine si evidenziano anche i limiti della demo prodotta con alcuni futuri aggiornamenti possibili.

- Il Capitolo [\[8\]](#) elenca i lavori futuri da svolgere all'interno del framework, la necessità di poter implementare soluzioni alternative a quella proposta in questo documento, e i limiti che devono essere superati affinché il framework possa essere utile in un ambiente reale e non solo di testing virtualizzato. Infine vengono descritte le conclusioni del lavoro, con un riassunto generale di tutto ciò che è stato prodotto.

Chapter 2

Network Security Automation e Verefoo

Nel mondo odierno le reti internet hanno rivestito un'importanza sempre maggiore, evolvendosi da piccole e semplici scenari per reti private domestiche a grandi e complicate topologie per le aziende e la comunicazione in tutto il mondo. Trattandosi di un mondo sempre in evoluzione, anche la configurazione e l'installazione di queste reti è diventata sempre più complessa, tanto da far notare sempre di più l'errore umano nelle impostazioni delle reti. Per queste situazioni nasce l'idea di *Network Security Automation*, che pone come obiettivo principale quello di riuscire a rendere la sicurezza delle reti il più possibile autonoma, riducendo la possibilità di errore umano e delegando all'automatizzazione tutte le criticità della configurazione delle varie funzioni di rete.

In questo capitolo si introduce la definizione di ***Security Function Chain (SFC)*** specificando la loro capacità nel migliorare la sicurezza delle reti, successivamente verrà introdotto il framework Verefoo che utilizza le SFC per poter produrre delle topologie di rete robuste e sicure e automatizzare il processo di creazione e configurazione delle reti.

L'ultima sezione del capitolo infine spazia sugli input che il framework accetta, le *Network Security Functions (NSFs)*, cioè tutte le funzioni che la rete deve rispettare come ad esempio filtrare dei pacchetti o criptare del traffico dati.

2.1 Service Function Chain

All'interno delle reti è possibile far passare il traffico in maniera *End-to-End*, *Site-to-Site* o *End-to-Site*. Durante la comunicazione nelle reti moderne è solito far transitare i pacchetti attraverso nodi che si occupano di funzioni specifiche all'interno della rete (Ad esempio un Packet Filter o un Network Address Translator NAT), che sono necessari per poter far rispettare alla rete determinate caratteristiche l'utente richiede. I nodi che sono adibiti a svolgere le funzioni prendono il nome di *Service Function (SF)* ed il collegamento di più nodi adibiti a SF viene definito *Service Function Chain (SFC)*. Una definizione formale di SF e SFC è stata presentata nel RFC 7665 [2] che definisce le seguenti:

- **Service Function:** Una funzione che è responsabile del trattamento specifico dei pacchetti ricevuti. Una Service Function può agire su varie livelli di uno stack di protocollo (ad esempio, al livello di rete o ad altri livelli OSI). Come componente logica, una SF può essere realizzata come un elemento virtuale o essere incorporata in un elemento di rete fisico. Una o più SF possono essere incorporate nello stesso elemento di rete. Possono esistere più occorrenze della funzione di servizio nello stesso dominio amministrativo.
- **Service Function Chain** Una Service Function Chain definisce un insieme ordinato di funzioni di servizio astratte e vincoli di ordinamento che devono essere applicati a pacchetti e/o frame e/o flussi selezionati come risultato di una classificazione. Un esempio di una Service Function astratta è un "firewall". L'ordine implicito potrebbe non essere una progressione lineare poiché l'architettura consente SFC che si ramificano su più di un percorso e consente anche casi in cui c'è flessibilità nell'ordine in cui le Service Function devono essere applicate.

La possibilità di definire SF separate e di combinarle fra loro nell'ordine che si preferisce garantisce alle reti la possibilità di essere flessibili e scalabili facilmente. Come infatti è descritto dalla definizione di SFC la concatenazione di più SFC permette ramificazioni su più percorsi, garantendo molteplici comunicazioni fra due host con caratteristiche di sicurezza differenti. Per comprendere meglio il concetto di SFC, un esempio fornito è il seguente:



Figure 2.1. Esempio di Service Function Chain

Come si può notare, vi sono diversi elementi all'interno di questo esempio. Per quanto riguarda i vari SF possiamo trovare i seguenti:

- **Firewall:** Si occupa di fare da packet filter fra i due webclient, per filtrare solo i pacchetti che effettivamente sono necessari alla comunicazione
- **Monitor:** Si occupa di monitorare il traffico in transito fra i due nodi, può essere un nodo da interrogare in caso di problematiche all'interno della rete per controllare che il firewall svolga il suo ruolo correttamente.
- **VPN Gateway:** Si occupa di criptare e decriptare il traffico. In questa topologia è fondamentale la loro presenza in quanto vi è un nodo considerato non affidabile, di conseguenza tutte le comunicazioni che passano attraverso quel nodo sono criptografate.

L'unione dei tre SF in questo specifico ordine definisce una Service Function Chain. E' importante notare che se l'ordine fosse stato diverso (ad esempio mettendo prima i 2 VPN Gateway e dopo il firewall) la SFC risultante sarebbe stata diversa da quella di partenza, garantendo una ramificazione.

2.2 Introduzione a Verefoo

Il potenziamento progressivo delle tecnologie appena descritte ha portato rapidamente allo sviluppo di reti che automatizzavano i lavori di configurazione che solitamente venivano svolti manualmente. Un esempio di queste nuove tecnologie viene svolto da VEREFOO[1](VERified REfinement and Optimized Orchestration), un framework che si pone diversi obiettivi fra i quali la definizione ad alto livello dei requisiti di sicurezza di rete, l'allocazione automatica ed ottimale delle varie Service Function per ottenere la maggior efficienza di rete allocando le minori risorse possibili, e la configurazione automatica delle varie *Network Security Functions*, eliminando l'errore umano che in reti di grandi dimensioni è solito capitare. Come è possibile intuire, la sfida di produrre una rete configurata automaticamente e correttamente è molto difficile da ottenere, perciò per assicurare la correttezza formale dei risultati ottenuti da Verefoo l'intero framework utilizza un metodo formale che si basa sulla risoluzione di un *Maximum Satisfiability Modulo Theories* (MaxSMT) tramite l'engine Z3 di Microsoft.

Questo, essendo basato su 3 pilastri quali *Ottimizzazione*, *Ottimalità* e *Correttezza Formale*, Verefoo si pone 2 obiettivi da soddisfare:

1. L'allocazione ottimale dei vari NSF's
2. La configurazione ottimale dei vari NSF's.

2.2.1 Descrizione del modello

Il modello di Verefoo, descritto in figura 2.2 richiede in input due elementi fondamentali:

- **Service Graph:** Una topologia logica delle funzioni della rete che insieme formano un collegamento end-to-end. A differenza delle SFC il Service Graph può avere diversi percorsi per collegare due punti nella rete presentando la ramificazione tipica della concatenazione di SFC. Durante la creazione del Service Graph non è necessario specificare nessun requisito di sicurezza, come ad esempio Firewall, Monitors o Filtering Databases.
- **Network Security Requirements:** Sono i requisiti di sicurezza che la rete in output dovrà avere dopo la computazione di Verefoo. Questo elemento è fondamentale per costruire il modello di MaxSMT da risolvere tramite Z3. Allo stato attuale, Verefoo consente di avere 3 requisiti di sicurezza principali:
 1. **Reachability Property:** Indica che un nodo Y di destinazione deve essere raggiungibile da un nodo di partenza X in almeno un percorso della topologia.
 2. **Isolation Property:** Indica che un nodo Y di destinazione **NON** deve essere raggiungibile da un nodo di partenza X in tutti i possibili percorsi all'interno della topologia.

3. **Protection Property:** Indica che la comunicazione tra un nodo di partenza X ed un nodo di destinazione Y deve essere sicura. In questo requisito è anche possibile specificare un nodo definito "*Untrusted Node*" ovvero un nodo che potrebbe essere un possibile punto di debolezza nella rete e che quindi deve poter vedere solo il traffico criptografato.

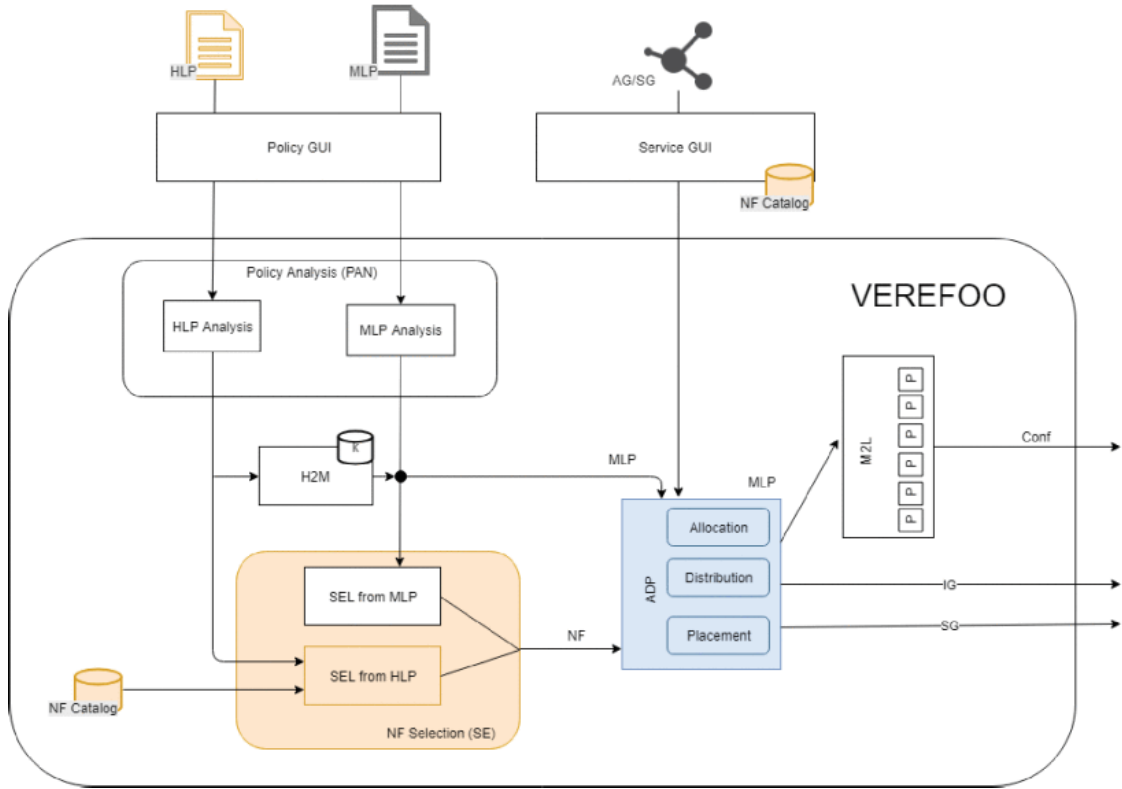


Figure 2.2. Architettura di VEREFOO [1]

Successivamente all'input, il framework esegue una serie di passi che possono essere così riassunti:

1. **Fase di controllo dell'input:** In questa fase Verefoo accetta l'input passato sotto forma di file XML e controlla la coerenza dell'input fornito. Il framework oltre ad accettare l'input composto da Service Graph e NSRs accetta anche la possibilità di fornire un *Allocation Graph* al posto del Service Graph (figura 2.3). La differenza fondamentale fra i due è che nel secondo oltre ai vari nodi della rete descritti nel Service Graph si specificano anche dei nodi aggiuntivi, chiamati *Allocation Places* che rappresentano i punti nella topologia dove è possibile istanziare una funzione di sicurezza di rete.
2. **Fase di analisi del modulo PAN:** qui Verefoo esegue un'analisi delle policy che sono state passate in input. Più specificatamente viene controllato che i vari NSRs siano coerenti fra loro, evidenziando eventuali errori (ad esempio non si può avere una Reachability Property ed una Isolation Property con gli

stessi nodi di partenza e destinazione). Alla fine dell'analisi delle policy viene prodotto il numero minimo di vincoli che devono essere rispettati affinché la topologia soddisfi i NSRs richiesti. In caso di errore un report viene solitamente fornito in output per comprendere il perché un determinato input non è soddisfabile.

3. **Trasformazione a Medium Level Language:** Una volta definiti ad alto livello i vincoli da far rispettare alla topologia, questi vengono tradotti da un linguaggio di alto livello ad uno di medio livello tramite il componente H2M.
4. **Selezione delle Network Function:** Ricevuto l'output dal modulo H2M il modulo Network Functions Selection (SE) si occupa di selezionare da un catalogo le NSF necessarie a rispettare i requisiti di alto e medio livello. Questo catalogo è anche accessibile al designer della rete nella fase di design disponibile nella Service GUI di Verefoo.
5. **Allocazione, Distribuzione e Piazzamento:** Durante questo passo del framework viene eseguito, come suggerito dal titolo, l'allocazione delle varie funzioni di rete calcolate tramite il modulo NF Selection e i vincoli di medio livello tradotti nell'H2M. Questo compito è affidato al modulo ADP che è il cuore di Verefoo, perché decide in quali punti della rete e con quali configurazioni le varie NFs devono essere allocate. L'ADP produce quindi un nuovo Service Graph nel quale sono allocate anche le funzioni di sicurezza di rete, e produce dei file di configurazione per ciascuna funzione allocata nel nuovo Service Graph. Queste configurazioni sono create in un linguaggio di basso livello grazie al modulo M2L presente all'interno dell'ADP.

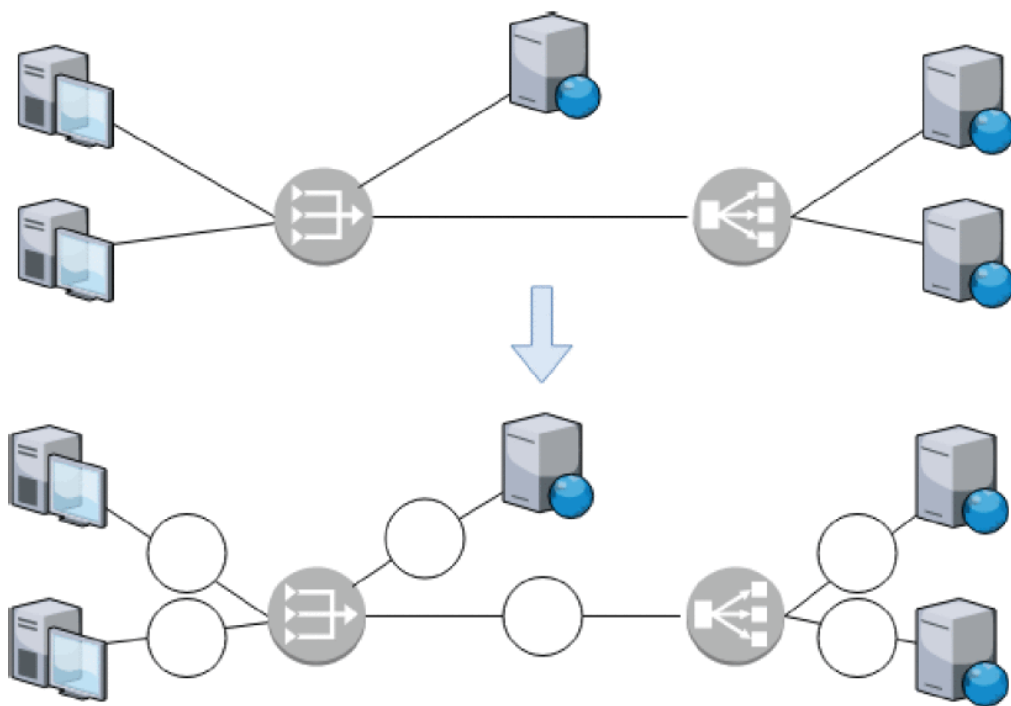


Figure 2.3. Esempi di Service (in alto) e Allocation(in basso) Graphs [1]

2.3 Definizione ed esempi delle proprietà di sicurezza

Le proprietà di sicurezza sono il secondo input che può essere fornito a Verefoo per stabilire i vincoli necessari affinché si possa creare una rete sicura. A differenza del primo parametro di input, la definizione della proprietà di sicurezza è responsabilità dell'amministratore della rete, in quanto deve decidere tramite le possibilità offerte da Verefoo e dalla GUI ad esso associata quante e quali proprietà inserire nella propria rete per garantire la sicurezza richiesta.

Per garantire flessibilità al framework è possibile inserire le definizioni sia con un linguaggio a basso livello definendo rispetti IP, porte e protocolli da accettare o rifiutare che con un linguaggio ad alto livello nel quale i vari elementi della rete verranno definiti da dei numeri che poi saranno mappati a degli IP.

I requisiti prodotti all'interno di verefoo saranno formulati con la seguente definizione:

Listing 2.1. Definizione di una proprietà di sicurezza generica

<code>[ruleType, IPSrc, IPDst, portSrc, portDst, transportProto]</code>

- **ruleType**: specifica quale proprietà stiamo definendo. Allo stato attuale del framework ci sono 3 possibili opzione: Reachability Property, Isolation Property e Protection Property
- **IPSrc**: definisce da quale nodo
- **IPDst**:
- **portSrc**:
- **portDst**:
- **transportProto**:

Chapter 3

Docker

3.1 Introduzione a Docker

Parte di docker.

3.2 Docker Compose

Parte di Docker compose.

description [\[3\]](#)

Chapter 4

Conclusions

Conclusion and future works

Bibliography

- [1] D. Brighenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, “Towards a fully automated and optimized network security functions orchestration,” in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–7.
- [2] J. M. Halpern and C. Pignataro, “Service function chaining (SFC) architecture,” *RFC*, vol. 7665, pp. 1–32, 2015. [Online]. Available: <https://doi.org/10.17487/RFC7665>
- [3] D. Brighenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, “Automated optimal firewall orchestration and configuration in virtualized networks,” in *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–7. [Online]. Available: <https://doi.org/10.1109/NOMS47738.2020.9110402>