**Alma Mater Studiorum — Università di Bologna**
**Campus di Cesena**

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea triennale in Ingegneria e Scienze Informatiche

# Studio delle Botnet per mezzo di IDS

Tesi di Laura

Presentata da:
**Manuel Luzietti**
**Matr.0000937684**

Relatore:
**Prof. G D'Angelo**

Correlatore:
**Prof. C Barbone**

**Anno Accademico 2021–2022**

*We shall not cease from exploration*
*And the end of all our exploring*
*Will be to arrive where we started*
*And know the place for the first time.*
*T.S. Eliot*

# Introduzione

Il rapporto dell'Associazione Italiana per la Sicurezza Informatica [1] ha rivelato che nel 2022 sono state individuate ben 7.71 miliardi di minacce *Botnet* in tutto il mondo, mentre gli attacchi *Distributed Denial of Service* (DDoS)[1] in Italia, spesso veicolati attraverso Botnet, sono raddoppiati rispetto all'anno precedente. Dei 6.32 miliardi di *malware* rilevati, 39.88 milioni solo in Italia, il rapporto ha evidenziato una particolare polarizzazione rispetto al 2021 attorno a *Zeus*, *Ramnit* e *QakBot*, tutti in grado di costringere la macchina infetta in una rete di *bot* [2]–[5]. Inoltre, le rilevazioni effettuate dai device appartenenti all'Autonomous System (AS) di Fastweb mostrano che il 3% delle infezioni è causato da *Conficker* (più di 9 milioni di macchine infette stimate [6]), con al secondo posto *Avalanche-Andromeda* (responsabile della distribuzione di circa 80 famiglie di malware), entrambi attribuibili alla classe Botnet.

Questi sono solo una parte dei dati allarmanti che dimostrano quanto le Botnet siano diffuse e permeate nel mondo del *cybercrimine*. Per questo motivo rimangono tutt'ora un importante oggetto di studio da parte di provider di sistemi di sicurezza ICT e ricercatori, all'interno di un contesto in cui l'Italia durante l'anno passato ha registrato il numero più alto di incidenti informatici mai registrato[2].

Le Botnet sono reti costituite da macchine infette, dette bot, che eseguono comandi provenienti da una o più macchine, detti Command and Control Server (C&C Server), impartiti dal botmaster (controllore dei bot). Il concetto alla base di questa tecnologia è così semplice da rendere labile il confine con altri malware dalla struttura simile. Si pensi ad esempio a un ipotetico *Trojan*[3], capace di inviare le informazioni rubate dagli utenti verso server di raccolta dati; questo ne condividerebbe la possibile struttura, seppure non fosse in grado di eseguire comandi. Non è una novità infatti che un malware inizialmente progettato per uno scopo preciso e limitato, venga poi aggiornato in una versione capace di eseguire comandi, trasformandolo in tutto e per tutto in una Botnet, estendendone le capacità col minimo sforzo (ne è un esempio *Ramnit* [5]). Queste reti possono essere utilizzate per una vasta gamma di attività illecite, tra cui Remote Administration Tool (RAT)[4], DDoS, distribuzione di malware, furto di informazioni personali, *spam*, etc.

[1] Attacco informatico in cui si cerca di esaurire le risorse computazionali della vittima attraverso molteplici richieste provenienti da plurime sorgenti

[2] 2.489 incidenti di cui almeno 1500 con severità compresa tra alta e critica

[3] Virus informatico nascosto all'interno di un altro applicativo

[4] Virus informatico capace di permettere accesso attraverso la rete a macchina vittima

L'elaborato che segue si pone come obiettivo lo sullo studio di queste tecnologie, partendo da un punto di vista teorico per poi procedere con l'analisi di due Botnet *open source*[5] in un ambiente controllato appositamente strutturato. La trattazione della tesi sarà organizzata in modo da studiarne in primo luogo le topologie, i protocolli e le tecniche di offuscamento utilizzate dalle Botnet. Successivamente, verranno analizzati il *deployment* dell'infrastruttura e gli strumenti utilizzati, inclusi gli *intrusion detection system* installati per il testing. Verrà quindi presentata una sezione inerente gli Agent[6], prima di proseguire con l'analisi della seconda rete di bot. Infine, verranno analizzati gli approcci di rilevazione utilizzati e considerati, per poi passare a una breve trattazione di altri possibili approcci.

[5]con Botnet open source si intende sviluppata a scopi divulgativi/didattici, con codice sorgente visualizzabile e a cui è possibile contribuire.

[6]Programma attivo in background che raccoglie informazione ed esegue task su macchina monitorata

# Indice

# 1 Studio delle Botnet

Una botnet è una rete che consiste di *host* compromessi chiamati bot che eseguono istruzioni impartite da un altro host detto *master* o *botmaster*, responsabile di coordinare i bot. Il botmaster non comunica direttamente con i bot, infatti esso usa almeno un server C&C che funge da *Proxy* tra di esso e le vittime. Questo garantisce un livello di anonimato aggiuntivo per il master oltre alla possibilità di scalare orizzontalmente le capacità del sistema. Avendo infatti più C&C server permette di avere più capacità di calcolo per gestire numeri elevati di bot oltre a garantire la disponibilità del servizio in caso di *take down* di uno dei C&C server. Moderne topologie di Botnet possono usare i bot che sono già parte della rete come C&C server, rimediando quindi al problema del singolo *point of failure* di una topologia centralizzata. A volte le funzionalità di un bot e quelle di un C&C server possono risiedere in stesso host.

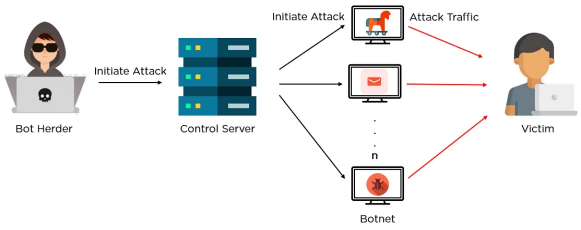



Figura 1.1: esempio di architettura di una Botnet [7].

Le Botnet possono essere di tipo *push based* o *pull based*. Nel primo caso è il server che invia i comandi al bot mentre nel secondo caso è il bot infetto che è incaricato di richiedere i comandi al server periodicamente. La modalità operazionale dipende più che altro dal protocollo in uso[1]. Di seguito verranno presentati i principali schemi comunicazioni, topologie, protocolli e tecniche di occultamento delle Botnet [8], [9].

## 1.1 Schema Comunicazionale

Parte importante della comunicazione della Botnet è il *coordinamento*. I task che non sono automatizzati vengono eseguiti solo dopo aver ricevuto l'apposito comando dal C&C, il che implica una serie di messaggi da e verso il server. I task automatizzati non richiedono

[1] eg. HTTP costruito su modello richiesta-risposta quindi una Botnet basata su questo protocollo sarà di tipo pull based.

il coordinamento con il server di controllo e quindi non generano traffico di rete. Di conseguenza più sono i task automatizzati, meno sarà il traffico di rete necessario per la coordinazione del bot e più sarà difficile da identificare la Botnet attraverso l'analisi del traffico di rete. Ovviamente limitare il coordinamento è a discapito della flessibilità, quindi solo una parte dei task viene automatizzata (ovviamente ci sono eccezioni anche a questo).

Lo schema di comunicazione della Botnet può essere separato in più parti: *propagazione* e *operazione*. Questi passi, dal punto di vista dell'host compromesso, possono avvenire solo sequenzialmente dato che l'infezione deve avvenire prima che il bot possa divenire operazionale.

### 1.1.1 Propagazione

La parte di *propagazione* è la fase in cui si cerca di reclutare nuovi bot. È la prima fase in nel ciclo di vita di un bot, ed è una fase importante dato che la potenza di una Botnet aumenta con la sua grandezza[2]. Un'eccezione a questo sono gli APT (Advanced Persistent Threat) che sono attacchi mirati solitamente, e quindi necessitano solo di un limitato numero di macchine, oltre al fatto che devono rimanere nascoste nel sistema il più a lungo possibile, evitando dunque di propagarsi per non rischiare di essere rilevate.

La propagazione può avvenire attivamente o passivamente. Entrambi i metodi di propagazione implicano che l'host vittima venga infettato con l'eseguibile di riferimento della Botnet.

#### Attiva

Nella propagazione attiva, il bot cerca e sfrutta vulnerabilità in altre macchine per espandersi. Questo può avvenire su comando del C&C o in automatico. A seconda dell'*exploit*[3] selezionato, una prima fase di scansione della rete può essere necessaria.

Una volta che la macchina è infetta, se il task non è automatizzato sarà necessario una messaggio di coordinazione dal C&C server, in cui si specificano parametri per l'exploit e la possibile scansione di rete, oltre ad avviare l'infezione. Alcune Botnet mantengono questi parametri di configurazione in binario o in file separato, non necessitando dunque di un passo di coordinazione.

La scansione di rete è opzionale, e può essere di diversi tipi a seconda delle informazioni che si vuole ottenere e del livello di occultamento che si vuole mantenere. Ad esempio potrebbero essere usate scansioni UDP[4]/TCP[5] per cercare servizi esposti e vulnerabili, o

[2] eg. Botnet capace di attacchi DDoS è più efficiente se ha più bot attraverso il quale generare traffico di rete.
[3] codice usato per sfruttare una vulnerabilità.
[4] User Datagram Protocol.
[5] Transmission Control Protocol.

semplicemente richieste ICMP[6] per limitarsi a identificare quali porte sono attive sull'host target.

Lo step principale è l'infezione, in cui si cerca di abusare i servizi vulnerabili della macchina vittima. Questa fase dipende molto dal tipo di attacco scelto[7] . Dato che durante questa fase la dimensione di dati scambiabili è relativamente limitata, una fase ulteriore di *download* può essere necessario.

Il download può avvenire in più parti, tipica delle *multi stage infection*. In primo luogo la vittima esegue solo un piccolo eseguibile[9] detto *dropper* che si occupa di rimediare l'eseguibile vero e proprio dalla Botnet o da un'altra sorgente. Il binario può anche essere ricevuto criptato, rendendo ulteriormente difficile per i ricercatori riuscire a ottenere l'eseguibile per scopi di *reverse engineering*[10]. Quindi l'eseguibile viene opzionalmente decriptato ed eseguito.

La fase finale di questo processo di propagazione attiva è lo step di *registrazione*. Questo step è opzionale, ma fondamentale se si vuole tener traccia delle dimensioni della Botnet, in caso di Botnet *push based*[11] o per particolari tipologie di Botnet[12].

### Passiva

Con propagazione passiva si intende la distribuzione del malware di riferimento attraverso altri vettori non in controllo della Botnet. Questo include propagazione attraverso email, siti web o dispositivi di archiviazione esterna. Comune a questi meccanismi di infezione è il fatto che la vittima infetta la propria macchina eseguendo il malware, attraverso un click o un azione particolare. La distribuzione via email può avvenire attraverso email di *phishing*. Queste sono email che sembrano legittime, ma che in realtà hanno l'obiettivo di ingannare la vittima ad aprire un file allegato o visitare un particolare sito web maligno. Analoghi a email di phishing sono i siti web di phishing. Pagine web uguali alla loro copia legittima ma appartenenti a domini diversi (tecnica anche detta *domain typosquatting*).

Si possono anche usare vulnerabilità in browser, plugin della vittima o compromettere un sito web che esso visita frequentemente, per causare un *drive by download*[13].

Si noti che in un contesto del genere, solo la ricezione di un email o la visita a un sito web può essere individuata a livello di rete. Inoltre la comunicazione in questo caso usa gli stessi protocolli e modalità del traffico di rete legittimo. Per questo motivo solo attraverso

[6] Internet Control Message Protocol.

[7] eg. eseguire una richiesta contenente un payload maligno o eseguire attacco *bruteforce*[8] su un servizio (eg. *ssh*).

[8] attacco in cui si cerca di indovinare una combinazione in modo esaustivo, testando tutte le combinazioni possibili.

[9] un eseguibile di piccole dimensioni è solitamente meno sospetto.

[10] tecnica di analisi di un binario attraverso appositi strumenti, necessario per capire il flusso di esecuzione di un programma e relative funzionalità senza eseguirlo.

[11] il server deve conoscere la posizione del bot per poter comunicare con esso.

[12] eg. Botnet P2P.

[13] tecnica di attacco che si riferisce al download non intenzionale di software maligno da sito compromesso.

Deep Packet Inspection[14] (DPI) sarebbe possibile individuare contenuto eseguibile e questo è possibile solo se il tutto non è criptato.

Dispositivi di archiviazione esterna possono essere infettati per far sì che il malware si muova tra dispositivi. All'inserimento del media di archiviazione all'interno di un altro *device*, questo avrebbe una copia dell'eseguibile potenzialmente eseguibile, senza alcun traffico di rete o movimenti sospetti.

Una volta che il malware viene eseguito sulla macchina vittima, seguono le fasi di download e registrazione, opzionali e comuni alla propagazione attiva (Sezione 1.1.1).

### 1.1.2 Operazione

La seconda fase è quella *operazionale* in cui si eseguono i veri e propri *task*, a seconda di quali siano gli obiettivi della Botnet. I task possono essere categorizzati in accordo al loro obiettivo come:

- Download dati;
- Upload dati;
- Forward Proxy;
- Reverse Proxy;
- Instruction.

#### Download dati

Con Download ci si riferisce alla memorizzazione di dati in bot, provenienti da sorgenti esterne. Dati possono essere file arbitrari, aggiornamenti della Botnet o malware aggiuntivo. I dati potrebbero perfino essere contenuti illegali atti a screditare la vittima. Questo meccanismo permette anche di offrire la Botnet come servizio di distribuzione di malware a pagamento. In questo modo sviluppatori di malware possono pagare il botmaster per avere un base di partenza di macchine su cui poter installare in modo agile il proprio software maligno.

La capacità di aggiornamento della Botnet permette di adattare la rete a nuove necessità oltre che a difendersi da possibili minacce. Ovviamente questa capacità deve essere ben progettata in quanto può fornire un vettore di attacco a terzi, che possono sfruttare questa *feature* per prendere possesso della Botnet o per rimpiazzarla con un altra.

L'operazione di Download inizia con una prima parte di coordinazione in cui si specifica quale file scaricare e dove cercarlo. Infatti il bot non è costretto a scaricare il file dal C&C server, che quindi non necessita di incorporare meccanismi e protocolli di *file transfer*. Il

[14] analisi dei pacchetti di rete che va oltre alla mera intestazione.

bot potrebbe avere all'interno del binario delle direttive di download da eseguire subito dopo l'infezione, rendendo la fase iniziale di coordinazione superflua. I dati possono anche essere passati all'interno del messaggio di coordinazione o essere inclusi in binario in fase di propagazione, in caso contrario verranno scaricati successivamente.

L'ultima fase è quella in cui si memorizzano i dati in macchina o dove si esegue il software aggiuntivo scaricato, a cui può seguire un messaggio opzionale verso il server.

### Upload dati

L'operazione di Upload dati dal bot verso il server viene utilizzata da tutte quelle tipologie di Botnet che hanno come scopo ultimo la raccolta di dati o il calcolo distribuito. Con raccolta di dati si intende prelevare informazioni dal sistema infetto. Queste informazioni possono essere file arbitrari, informazioni sensibili dell'utente, credenziali d'accesso, informazioni del sistema in uso, etc. Con calcolo distribuito si intende l'upload del risultato di una computazione eseguita in modo distribuito su tutti i nodi infetti della rete.

Questo task inizia con uno step di coordinazione, in cui il C&C server invia comandi con specifiche istruzioni o dati se necessari, che può essere opzionale nel caso l'upload venga eseguito subito dopo la propagazione. Successivamente i dati richiesti sono prelevati dal bot o la computazione richiesta è eseguita. Per concludere i dati sono inviati al server.

### Forward Proxy

Un *forward proxy* è un server capace di connettersi a un host specifico come tramite per un altro host. Questo permette di mantenere l'anonimato dell'host che inizia la richiesta, di fare tunneling delle connessioni o di effettuare connessioni verso host che altrimenti non sarebbero raggiungibili.

Le Botnet possono implementare proxy unidirezionali o bidirezionali. Proxy bidirezionali sono necessari se si vuole il risultato di una richiesta. Alcune Botnet includono SOCKS proxy[15] come servizio, permettendo di utilizzare la Botnet come servizio di anonimato, sfruttando i bot come proxy server generici. Altro uso di proxy bidirezionale è la possibilità di valicare Firewall o NAT infettandoli e usandoli come proxy, fungendo da *bridge* verso i device interni alla rete che altrimenti non sarebbero raggiungibili.

Per creare meno traffico sospetto possibile, la Botnet può anche usare i proxy per fare *protocol encapsulation*, ovvero nascondere un protocollo all'interno di un altro. In questo modo i sistemi di rilevazione devono fare lo sforzo aggiuntivo di essere in grado di interpretare tutti i protocolli possibili se voglio ricostruire la comunicazione originale.

Proxy unidirezionali possono essere usati per effettuare attacchi DDoS. Infatti il botmaster non ha bisogno di controllare ogni messaggio e non è interessato alla risposta di questi. Gli basta comunicare destinazione e il tipo di comunicazione da utilizzare.

[15] proxy server in grado di eseguire forward di connessioni TCP e UDP (da versione 5 in poi) indipendentemente dal protocollo applicativo veicolato.

Un altro modo di usare i proxy unidirezionali è quello di usarli per distribuire email di spam. Durante la fase di coordinazione al master basta invia il *template* della email e una lista degli host da colpire.

A livello di schema comunicazionale, sia quello unidirezionale che bidirezionale iniziano con uno step di coordinazione. Nel caso bidirezionale viene settata la configurazione del proxy con le porte su cui fare *forwarding* e i protocolli da usare. Dato che queste configurazioni possono essere già incluse in binario questo step è opzionale.

Per il proxy unidirezionale invece, che è usato esclusivamente per eseguire task automatizzati, questo passo non è opzionale. Ad esempio se si vuole eseguire un attacco DDoS deve essere impostato il tipo di attacco, il target e la durata. In teoria, potrebbe essere possibile inserire tali informazioni nel codice binario, ma ciò potrebbe rendere questo tipo di attacco troppo rigido, mentre esso per natura richiede di essere improvviso e di durata variabile.

L'ultimo step è lo scambio di dati. In caso il proxy sia bidirezionale, la comunicazione sarà composta di richieste e risposte dove il bot farà da tramite. Nel caso di proxy unidirezionale i dati sono inviati solamente da bot verso il target.

#### Reverse Proxy

Le Botnet possono essere usate per creare un *fast-flux network* (Sezione 1.2.1). Un *fast-flux network* consiste di più *reverse proxy server*[16] che sono usati per connettersi al C&C server.Dato che le connessioni sono verso i proxy, l'identità del C&C server rimanere nascosta, rendendo il server più resistente a possibili *take down*.

La comunicazione di rete necessaria per questo task inizia con uno step di coordinazione in cui il bot è istruito di quali porte aprire e verso quali indirizzi interni connettersi. Dopo questa fase di coordinazione altri host o bot possono connettersi al reverse proxy per comunicare indirettamente col C&C server. Le informazioni di coordinazione non sono preinseribili nel binario del malware per via della natura della Botnet. I bot che fungono da reverse proxy potrebbero infatti non essere mai raggiungibili in uno scenario in cui gli indirizzi di questi dovessero essere prefissati[17].

#### Instruction

Una delle feature presenti nelle Botnet è la capacità di eseguire task a nome del botmaster che non generano traffico di rete. Esempi di task di questo genere sono:

- Terminazione di altri processi;
- Cancellazione di file;

[16] mentre un forward proxy server funge da intermediario per un client, il reverse proxy server agisce come intermediario per il server, ponendosi tra il client e il server.

[17] I bot potrebbero non essere online, il malware può essere neutralizzato o possono esserci problemi di rete.

- Modifiche al traffico di rete dell'host;
- *Disruption* di servizi;
- Creazione di *backdoor*[18];
- Distruzione o criptazione di dati in dispositivi di archiviazione;
- Esecuzione di eseguibili.

La comunicazione necessaria per questo genere di operazione inizia con una fase di coordinazione in cui vengono specificati il task da eseguire ed eventuali parametri. Gli eseguibili della Botnet possono essere accompagnati da file di configurazione contenenti queste istruzioni o possono essere inclusi in binario direttamente, rendendo questa fase opzionale. Ad esempio molte Botnet contengono una lista dei processi di antivirus da chiudere o disabilitare. Dopo lo step di coordinazione vengono eseguiti i comandi ricevuti.

## 1.2 Topologie di rete

La comunicazione tra C&C server e i bot può essere organizzata da un punto di vista topologico nei seguenti modi:

- Centralizzata;
- Decentralizzata;
- Ibrida.

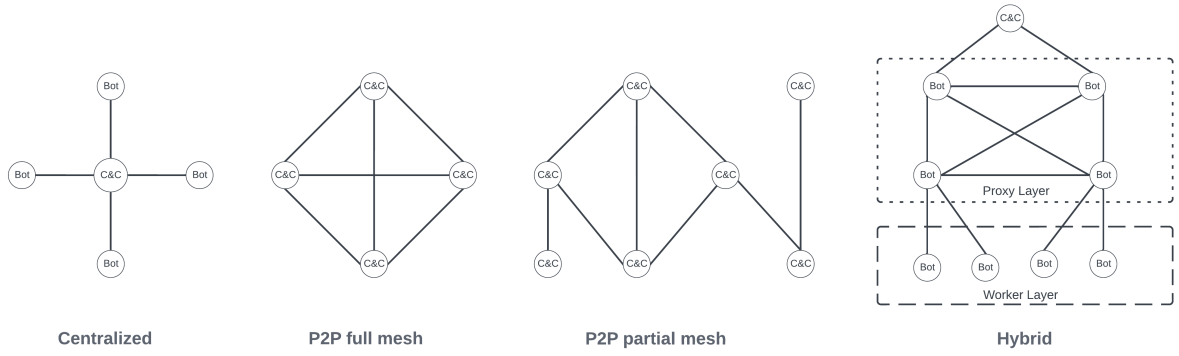



Figura 1.2: Topologie botnet.

[18]rendere la macchina accessibile dall'attaccante attraverso la rete.

### 1.2.1 Centralizzata

Nella topologia centralizzata ogni bot si connette direttamente a un unico server. Per una topologia del genere spesso sono usati protocolli di comunicazione semplici come Internet Relay Chat (IRC) (Sezione 1.3.1) o HyperText Markup Language (HTTP). Per questo motivo ne risulta anche semplificato il *set-up*, dato che questi protocolli non hanno particolari esigenze da soddisfare. Una topologia del genere gode anche di bassa latenza, dato che bot e server si scambiano messaggi direttamente, e di alta scalabilità per via della semplicità della rete.

Tuttavia questa semplicità di rete è a discapito della robustezza. Infatti una struttura centralizzata ha un singolo *point of failure* e il *take down* del server principale di comando e controllo comporterebbe la rottura dell'intera rete Botnet. Il singolo *point of failure* può essere mitigato in parte utilizzando più C&C server. Un'altra tecnica utilizzata per mitigare il problema della robustezza è il *fast-flux* che verrà descritto nella Sezione 1.2.1.

Altro punto a sfavore di questa topologia è il fatto che gli indirizzi dei C&C server devono essere inseriti all'interno del binario del bot. Questo è un rischio per il botmaster che potrebbe perdere il controllo del C&C server in caso questi indirizzi dovessero essere scovati da un'analista di sicurezza informatica. Una possibile soluzione potrebbe essere criptare od offuscare il codice del bot. Tuttavia anche questa soluzione risulterebbe temporanea in quanto non garantirebbe la sicurezza che questi indirizzi non vengano trovati a seguito di un analisi approfondita da parte di uno specialista. Una tecnica spesso usata per evitare l'*hardcoding* degli indirizzi dei server è il Domain Generation Algorithm (DGA) che verrà analizzato nella Sezione 1.2.1.

#### Fast-flux network

In un Fast-flux network (Figura 1.3) il servizio Domain Name System (DNS) di risoluzione degli *hostname* viene usato come *multiplexer* e *load balancer*. La macchina infetta non comunica direttamente col server usando il suo indirizzo IP ma punterà a risolvere un hostname attraverso risoluzione DNS. Al record DNS in cui vi è registrato l'hostname da risolvere saranno associati molteplici indirizzi IP appartenenti a più bot della Botnet. Così facendo si sfrutta le capacità native di *load balancing* dei server DNS per ciclare sugli indirizzi IP assegnati. I bot così associati a questo hostname sono gli unici a conoscenza dell'indirizzo IP del C&C server e quindi sono gli unici in grado di comunicare con esso. Quando una macchina infetta ha necessità di comunicare con il server di controllo, risolve l'hostname associato ottenendo l'indirizzo IP di un bot che farà da proxy (reverse proxy) nella comunicazione verso il server. In questo modo si sposta il singolo point of failure dal server C&C al DNS server.

Per rendere più complessa la rete ed eliminare anche questo point of failure si può creare un *double-flux* network. Il double-flux usa lo stesso schema per ciclare i *name server*

autoritativi che risolveranno l'hostname richiesto, traducendolo come visto in uno dei bot che farà da proxy verso il C&C server.

Si noti che registrare ed eliminare un record DNS sono operazioni veloci e quindi è possibile variare velocemente i bot che fungono da proxy o name server.

Anche utilizzando questa tecnica rimane comunque il problema dell'hardcoding degli hostname all'interno del binario del bot. Per risolvere anche questo problema spesso questa tecnica viene associata a *Domain Generation Algorithm* (DGA) che verrà di seguito descritto.

#### Domain Generation Algorithm (DGA)

Domain Generation Algorithm è un algoritmo utilizzato per creare domini partendo da un valore di input variabile [10]. Molte Botnet usano questo algoritmo per generare e registrare in automatico questi domini per il loro C&C Server.

Quando il bot vuole cercare di contattare il server, usa l'algoritmo per generare un numero significativo di domini che proverà a risolvere. Il master che è a conoscenza dei dettagli dell'algoritmo, può preregistrare uno dei domini generati per essere contattato con successo dai bot. Così facendo non è necessario preinserire domini all'interno dell'eseguibile. Molti DGA usano il tempo corrente come input, rendendo necessario che ogni bot abbia l'orologio di sistema sincronizzato per poter funzionare correttamente. Se per esempio un bot dovesse generare 50.000 domini da testare, un ente governativo dovrebbe preregistrare tutti i 50.000 domini per poter evitare che il botmaster riesca nel suo intento. Inoltre va considerato che se le comunicazioni tra bot e server dovessero usare cifratura a chiave pubblica, anche preregistrando correttamente uno dei domini che la Botnet usa per coordinare i bot, non si riuscirebbe a prenderne il controllo in quanto la comunicazione verrebbe negata per via delle firme digitali non combacianti con quelle del botmaster.

Un esempio di algoritmo DGA può essere il seguente [10]:

1: $i \leftarrow 0$
2: $domain \leftarrow$ ''
3: $year, month, day \leftarrow getCurrentDate()$
4: **while** $i < 16$ **do**
5: $\quad year \leftarrow ((year \hat{} 8 \cdot year) \gg 11) \hat{} ((year \,\&\, \texttt{0xFFFFFFF0}) \ll 17)$
6: $\quad month \leftarrow ((month \hat{} 4 \cdot month) \gg 25) \hat{} 16 \cdot (month \,\&\, \texttt{0xFFFFFFF8})$
7: $\quad day \leftarrow ((day \hat{} (day \ll 13)) \gg 19) \hat{} ((day \,\&\, \texttt{0xFFFFFFFE}) \ll 12)$
8: $\quad domain \leftarrow domain + chr(((year \hat{} month \hat{} day) \% 25) + 97)$
9: **end while**

### 1.2.2 Decentralizzata (P2P)

Una topologia decentralizzata è tipica di una rete di soli bot dove ogni bot può essere un potenziale C&C Server. Questo meccanismo in cui qualsiasi nodo può potenzialmente

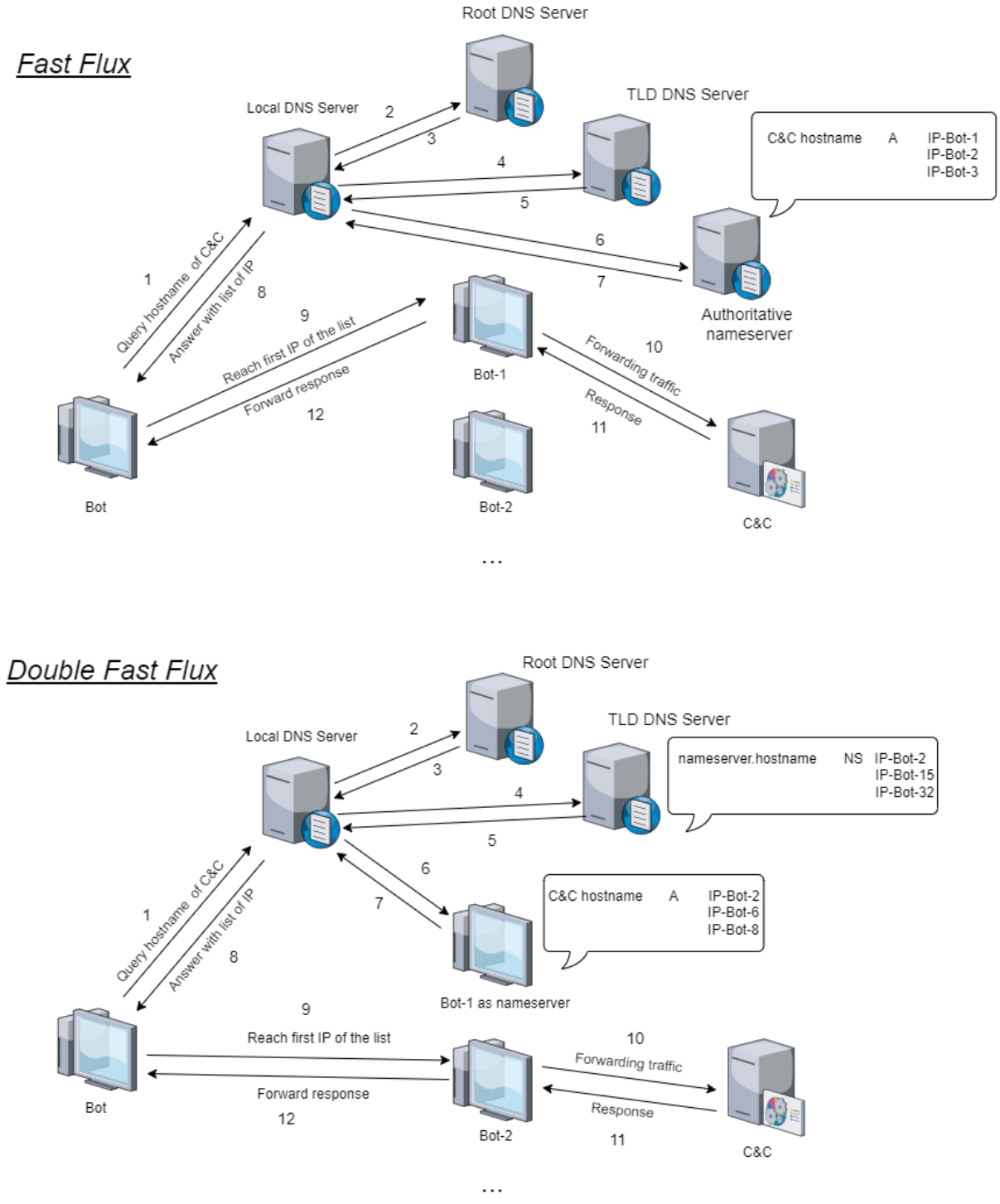



Figura 1.3: Fast flux and double fast flux network.

essere server è tipico delle reti Peer-To-Peer (P2P). Ogni bot è connesso ad almeno un altro bot e i comandi del C&C possono raggiungere ogni nodo della rete se ogni bot ha l'abilità di passare i comandi ai nodi direttamente connessi.

Una rete decentralizzata può essere a maglia parzialmente connessa o completamente connessa.

Una topologia a maglia completamente connessa ha il vantaggio di essere a bassa latenza dato che i comandi non devono essere passati a nodi intermedi per arrivare a tutti i nodi. Un altro vantaggio è che una rete del genere risulta molto robusta in quanto togliere un numero arbitrario di nodi non influenza negativamente la comunicazione di rete degli altri nodi. Tuttavia, considerando che questa topologia richiederebbe un altro numero di connessioni per una rete composta di molti nodi, risulterebbe non scalabile. I sistemi operativi infatti hanno limiti sul numero di connessioni che possono aprire e quindi viene limitato il numero massimo di bot in rete. Inoltre il numero di connessioni influisce sulla visibilità in rete della Botnet. Aggiungere e togliere nodi alla rete necessita di un alto numero di messaggi di coordinazione e per questo motivo spesso le Botnet P2P non sono a maglia completamente connessa.

Una topologia a maglia parzialmente connessa non ha i problemi delle topologie a maglia completamente connessa appena visti ma ha comunque dei fattori negativi. Dato che un bot non è connesso a tutti gli altri bot, l'eliminazione di nodi topologicamente importanti dalla rete può causare la completa interruzione delle comunicazioni per una parte dei device connessi. Implementare topologie P2P risulta difficile a causa principalmente del dover garantire una distribuzione affidabile dei comandi verso tutti i bot oltre al problema di trovare i *peer* iniziali.

Il problema della ricerca iniziale dei peer può essere ovviata inserendo la lista in eseguibile di bot. Un altro modo è quello di usare tecnologie pubbliche già esistenti per memorizzare e in seguito reperire la lista di peer (eg. usare cache server di applicazioni P2P esistenti). Il point of failure della rete si sposta così alla lista di peer iniziali. Un'altra alternativa alle liste di peer pubbliche è la possibilità di scansionare l'intera rete Internet in cerca di peer.

Dato che la rete non è a maglia completa, deve anche essere garantito il trasporto affidabile dei comandi che in questo caso non è nativo di questa topologia di rete. Per questo motivo spesso si usano protocolli P2P già noti che hanno questa funzionalità già implementata. Se si vuole creare la Botnet attraverso un protocollo *ad-hoc*, le funzionalità di consegna affidabile dei messaggi e di *routing* devono essere implementate per poter garantire il corretto funzionamento della rete.

Dato che non tutti i dispositivi sono direttamente raggiungibili da Internet a causa di possibili Firewall o NAT, spesso le Botnet classificano i bot in due categorie: *super node* e *NAT-node*. A seguito dell'infezione la connettività viene rilevata e solo i NAT-node sono usati per attività maligne mentre i super node sono usati solo come tramite per le comunicazioni con i bot C&C.

### 1.2.3 Ibrida

Uno dei modi per ovviare alle problematiche di una rete decentralizzata e allo stesso tempo di fruire dei vantaggi che offre come topologia, è quello di combinarla con la topologia centralizzata per avere i vantaggi di entrambe. Invece di connettere direttamente i bot al C&C server, un livello di proxy viene aggiunto. Questo livello di proxy è costituito da bot connessi in P2P.

I bot che eseguono i task costituiscono un ulteriore livello.

L'assegnamento di un bot al ruolo di worker o proxy è eseguito considerando le proprietà di connettività dei bot analizzati.

Per aumentare ulteriormente la sicurezza del C&C Server si potrebbe aggiungere un ulteriore livello di proxy a discapito di una maggiore latenza.

Altre Botnet ibride sono possibili sfruttando topologie diverse per task diversi. Una Botnet potrebbe ad esempio sfruttare topologia P2P per lo scambio di messaggi e topologia centralizzata con un web server per il download di dati.

## 1.3 Protocolli

Di seguito verranno brevemente descritti i protocolli più comunemente usati a livello comunicazionale e operazionale nelle Botnet.

### 1.3.1 Protocolli Comunicazionali

Con Protocolli comunicazionali si intende quei protocolli usati all'interno della Botnet per la comunicazione tra bot e Server, all'interno di una rete centralizzata, o tra bot e bot all'interno di una Botnet P2P. Ovviamente la scelta della topologia pone restrizioni sulle possibilità di scelta del protocollo comunicazionale.

Le Botnet spesso usano protocolli già esistenti, semplificando lo sviluppo della rete.

Il protocollo scelto può anche essere creato su misura. Questo permette di soddisfare esigenze specifiche come un maggior mimetismo[19] o restrizioni di rete[20].

#### Internet Relay Chat (IRC)

Internet Relay Chat è un protocollo a livello applicativo di messaggistica *text-based*, basato su architettura client-server. Venne inizialmente creato per lo sviluppo di applicazioni chat su Internet [11], in particolare inizialmente usato su Bulletin Board System (BBS)[21]. IRC inizia a essere sviluppato nell'anno 1989. Nell'anno 1993 J.Oikarinen pubblica l'RFC 1459 in cui vengono pubblicate le specifiche del protocollo.

[19] eg. si potrebbe creare un protocollo che mimi il traffico legittimo.

[20] eg. molte reti di corporazioni non permettono l'uso di particolari tipi di protocolli.

[21] sistema telematico client-server creato negli anni '70 prima che i Web Browser venissero realizzati. L'utente si connetteva a un server centrale per condividere e prelevare risorse.

Il client comunica con il Server che è responsabile di consegnare i messaggi ad altri client e server. Messaggi possono essere scambiati client-client o verso un gruppo di client attraverso *canali*[22]. I canali possono essere protetti da password.

Il protocollo supporta anche il trasferimento di file.

Questo protocollo facilita lo sviluppo di una Botnet basata su topologia centralizzata. Il botmaster ad esempio potrebbe inviare i comandi verso un canale di bot, dove il C&C in questo caso ricopre il ruolo di Server centrale incaricato di distribuire il messaggio a tutti i bot interessati. I canali come già detto, possono essere protetti da password per evitare che terzi prendano il controllo della Botnet. La capacità di file-transfer può essere sfruttata per distribuire malware, aggiornamenti della Botnet, file di configurazione, etc.

Tra gli svantaggi di questo protocollo ci sono il fatto che è facilmente identificabile e che non è un protocollo che viene comunemente usato nelle attività di tutti i giorni (nonostante sia un protocollo molto diffuso su Internet) e quindi potrebbe essere bloccato preventivamente via Firewall[23].

### HTTP

Uno dei protocolli a livello applicativo più utilizzati su Internet. Usato per la trasmissione di pagine web da web server a web client su Internet. È basato su architettura client-server dove la comunicazione di gruppo, al contrario di IRC, non è possibile. Visto il largo uso nella rete Internet, sono presenti tantissime implementazioni già pronte di client e server, rendendo lo sviluppo di una Botnet basata su protocollo HTTP facilitata.

Vista la struttura di HTTP può essere usato agilmente per creare una Botnet con topologia centralizzata.

Implementando un HTTP server e un HTTP client su ogni bot è possibile creare una rete P2P. Nonostante questo sia possibile attraverso HTTP, si dovrebbe comunque risolvere alcuni dei problemi noti alle reti P2P come rendere la comunicazione tra bot affidabile e la gestione dei messaggi replicati. Una possibile soluzione può essere creare dei *layer di rete* aggiuntivi sopra HTTP. Si potrebbe creare quindi un livello P2P per aggiungere informazioni necessarie per il trasporto affidabile dei messaggi a ogni nodo e per prevenire messaggi replicati. Un altro livello di routing opzionale potrebbe essere implementato per gestire l'invio selettivo di un messaggio a uno specifico bot non conoscendo il percorso esatto all'interno della rete P2P. Con il livello di routing si potrebbe anche implementare il trasporto affidabile, rendendo opzionale il livello P2P precedentemente descritto.

All'interno di questi livelli aggiuntivi si potrebbe anche aggiungere la possibilità di far rimbalzare il messaggio tra un numero arbitrario di nodi prima di consegnare il messaggio

[22] gruppo di client a cui viene assegnato un nome. Un messaggio verso un canale arriva a tutti i client appartenenti al gruppo rappresentato.

[23] Spesso questo è il caso delle reti corporative.

a destinazione, rendendo più complessa l'identificazione dell'origine del messaggio. Questa tecnica è nota col nome di Stepping Stones (Sezione 1.4).

Il fatto che HTTP sia largamente utilizzato su Internet rende lo scambio di messaggi attraverso questo protocollo non sospetto.

La sua variante HTTPS permette l'uso di comunicazione HTTP all'interno di una connessione cifrata, rendendo quasi[24] impossibile analizzare il contenuto dei messaggi da parte di strumenti di rilevazione.

#### Server Message Block

Server Message Block (SMB) è un protocollo originariamente specificato da Microsoft, IBM e Intel per la condivisione di file, stampanti, porte seriali e comunicazioni di varia natura attraverso la rete [12].

Basato su architettura client-server. Un client si connette a un server autenticandosi con credenziali utente o come anonimo. Una volta connesso può richiedere una lista di *share* disponibili. Con *share* si intende una lista di file o servizi accessibili.

SMB implementa anche meccanismi di comunicazione tra processi via network.

Questo protocollo può anche essere usato per infettare passivamente altri host. Se infatti un file condiviso è modificabile, un utente malintenzionato potrebbe inserire codice maligno all'interno di questo che in seguito potrebbe venir eseguito da un utente ignaro (propagazione passiva) in seguito al download sulla propria macchina.

Questo protocollo spesso viene bloccato dai *gateway* che provvedono accesso a Internet, motivo per cui viene principalmente usato dalle Botnet come protocollo di comunicazione in rete locale.

#### Protocolli P2P

Come già detto nella sezione inerente le topologie decentralizzate di Botnet (1.2.2), i protocolli P2P esistenti vengono spesso adottati per evitare di dover sviluppare protocolli P2P da zero. Questi protocolli possono essere usati per creare una rete P2P separata o per nascondere la rete di bot all'interno di una rete P2P esistente. Un esempio di protocollo usato è WASTE [13], originariamente creato come protocollo di messaggistica distribuita e file sharing. Un altro esempio di protocollo usato è Kademlia [14], che gestisce la struttura della rete, la comunicazione tra nodi e il modo in cui lo scambio di informazioni è effettuato.

#### Protocolli ad-hoc

La creazione di un protocollo *ex novo* permette di creare una comunicazione su misura che rispecchi le esigenze e gli obiettivi che la Botnet vuole perseguire.

[24] i firewall posso usare una tecnica chiamata *HTTPS interception* dove si sfrutta lo stesso principio degli attacchi Man In The Middle (MITM)[25] per analizzare il contenuto dei pacchetti HTTPS.

La maggior parte dei protocolli creati appositamente risulta basata su UDP o TCP a livello di trasporto. Ci sono anche casi osservati di protocolli basati su ICMP. Come già discusso precedentemente a seconda della topologia scelta si devono implementare tutte le accortezze necessarie al corretto funzionamento della rete.

Se si desidera una rete P2P è necessario implementare la comunicazione affidabile dei messaggi e gestire i messaggi duplicati. La questione dei messaggi duplicati può essere ovviata associando un identficativo (ID) ai comandi. Il trasporto affidabile dei messaggi deve essere gestita anche nel caso si utilizzi UDP come protocollo a livello di trasporto. Questo può essere garantito ad esempio implementando ritrasmissione dei messaggi e l'invio di acknowledgment. Un protocollo basato su UDP con consegna affidabile dei messaggi avrebbe anche il vantaggio di poter permettere più connessioni contemporaneamente in quanto nessuna connessione deve essere mantenuta al contrario di TCP.

Il payload dei messaggi può essere in binario o testo. Un payload formato testo necessita di un *parser* per comprendere i comandi così codificati, oltre a essere più facile da analizzare a livello di reverse engineering.

Uno dei punti a sfavore dei protocolli creati su misura è che questi non sono usati all'interno di traffico di rete legittimo e quindi potrebbe essere identificato.

### 1.3.2 Protocolli Operazionali

Oltre ai protocolli usati per il normale funzionamento della Botnet, sono necessari anche altri protocolli per svolgere i task operazionali atti a raggiungere l'obiettivo preposto. Di seguito verranno presentati alcuni esempi di protocolli usati per le più comuni attività di una Botnet:

- Per inviare spam via email deve essere in grado di poter usare Simple Mail Transport Protocol (SMTP). SMTP è un protocollo slient-server usato per inviare messaggi di posta elettronica attraverso server dedicato (Server SMTP). Dato che molti server di posta elettronica richiedono autenticazione solo per inviare messaggi fuori dalla rete locale, questa risulta la principale causa di spam. Un bot potrebbe infatti sfruttare la sua posizione all'interno della rete per inviare email di spam attraverso il mail server comune, verso altri nodi della rete locale. Per questo motivo molti email provider bloccano email provenienti da IP di reti private;

- HTTP e HTTPS vengono usati per eseguire *click fraude*, ovvero simulare il click su un URL o annuncio pubblicitario col fine di alterarne le statistiche di visita della risorsa. In questo modo il botmaster può fare profitto grazie al numero falsato di visite all'annuncio o pagina web;

- HTTP, UDP e TCP vengono anche usati per eseguire attacchi DDoS in cui si cerca di sovraccaricare il network del servizio target. Un altro modo di sovraccaricare il

servizio è attraverso l'invio di richieste computazionali intensive con l'intento di saturare la capacità di calcolo dell'applicazione target.

## 1.4 Tecniche di occultamento e offuscamento di comunicazione

Di seguito verranno brevemente raccolte alcune delle tecniche usate dalle Botnet per nascondere le comunicazioni od offuscarne il contenuto. Con offuscamento si intende rielaborare il contenuto in modo che risulti complesso comprenderne il significato, usato spesso per rendere più difficili i tentativi di reverse engineering o di analisi.

### Canali nascosti

Considerando che, come già detto, i protocolli costruiti ad-hoc sono facilmente individuabili in quanto non figurano comunemente nella rete Internet e che i protocolli già esistenti generano traffico di rete che non dovrebbe esistere in scenario normale, si possono usare i canali nascosti per occultare la comunicazione. Con canali nascosti si intende instaurare una comunicazione sfruttando mezzi che non sono usati a scopo comunicativo. Un esempio è usare gli header dei segmenti TCP o dei datagrammi del protocollo IP per comunicare informazioni o comandi.

### Crittografia

L'ispezione del contenuto del traffico di rete permette la ricerca di pattern di messaggi di Botnet conosciute, portando a una possibile identificazione dei device infetti o del C&C server. Per questo motivo spesso le Botnet usano crittografia a livello di rete per fare in modo che terzi non riescano a ispezionare il contenuto delle comunicazioni.

Un esempio di protocolli usati a tal fine sono HTTPS o TLS. Sebbene sia possibile creare protocolli crittografici su misura, ciò pone il rischio che una falla nel protocollo crittografico lo renda completamente inefficace. Per questo motivo i botmaster devono preoccuparsi di usare protocolli aggiornati che non presentino vulnerabilità note nel caso decidano di utilizzare protocolli già esistenti.

### Molteplici protocolli

Per adattare il bot alla rete in cui risiede e quindi rendere la Botnet meno identificabile è possibile creare un livello di rete variabile capace di usare molteplici protocolli. Avendo una vasta scelta di protocolli, il bot potrebbe scegliere di usare quello più popolare all'interno della rete in cui risiede per rendere la comunicazione meno sospetta.

### Compressione

La compressione dei dati viene comunemente usata per ridurre la dimensione di file. Tuttavia nel comprimere i dati ne offusca anche il contenuto in quanto, nonostante l'informazione originaria rimanga intatta all'interno dei dati compressi, questi risultano non interpretabili a meno di conoscer l'algoritmo di decompressione.

Dato che questa tecnica non richiede alcuna chiave, chiunque conosca l'algoritmo di compressione può decomprimere i dati e quindi risalire al contenuto originario.

### Steganografia

La steganografia è un'altra tecnica di offuscamento. Essa consiste nel nascondere informazioni all'interno di *contenitori* che non sono pensati per la comunicazione. Simile a livello di logica ai *canali nascosti* ma che usa file (eg. immagini, video, documenti, etc.) per comunicare al posto degli attributi dei pacchetti di rete.

In passato sono state proposte Botnet capaci di sfruttare questo principio all'interno dei social network, incapsulando le informazioni dei comandi all'interno di immagini JPEG postate dall'utente [15].

Altro esempio peculiare di botnet di questo tipo è quella ideata da A.Compagno e collaboratori, che hanno sfruttato i messaggi di testo all'interno dei social network, usando caratteri non visualizzati dai web browser [15].

### Stepping Stones

Tecnica usata per aumentare la segretezza del C&C server che consiste nell'aggiungere un numero arbitrario di nodi intermedi tra il server e i bot destinatari dei comandi. Questo può essere ottenuto implementando un layer di routing che permette di far rimbalzare i messaggi tra un numero desiderato di nodi prima di consegnarlo a destinazione.

### Rootkit

I rootkit sono un tipo di software progettato per nascondere le attività di un malware all'interno del sistema. Spesso utilizzano funzionalità di api hooking (Capitolo 7) per intercettare le chiamate a funzioni di libreria del sistema al fine di nascondere l'esecuzione di file malevoli o attività di essi.

# 2 Infrastruttura di testing

Prima di poter analizzare le botnet dinamicamente, è necessario progettare ed eseguire il *deploy* di un infrastruttura sicura, in cui è possibile eseguire i malware relativi alla botnet oggetto di studio senza rischiare di compromettere dispositivi al di fuori del perimetro designato.

Il deploy dell'infrastruttura (Figura 2.1) è stato eseguito su un Server VMWare ESXi (2.1) raggiungibile attraverso Virtual Private Network (VPN), in quanto situato all'interno della rete dell'Università di Bologna, Campus di Cesena. Attraverso le capacità di virtualizzazione del server sono state create una macchina Linux per ospitare il C&C server e due macchine che verranno infettate con malware per trasformarli in bot attivi. Le macchine bot in questione sono una basata su Linux e l'altra su Windows. Queste macchine sono assegnate a una rete VLAN isolata così che non siano in grado di comunicare con altri dispositivi al di fuori delle macchine soggette al test. Chiameremo questa rete *botnet Vlan*.

Per far in modo che le macchine sotto testing potessero occasionalmente interfacciarsi alla rete Internet in modo controllato è stato aggiunto un Firewall (2.2). La macchina in questione usa ClearOs (2.2.1) come sistema operativo e ha due Network Interface Card (NIC). Una delle NIC è collegata a una VLAN connessa a Internet e ha indirizzo IP assegnatogli attraverso Dynamic Host Configuration Protocol (DHCP). L'altra NIC è collegata alla botnet Vlan e ha indirizzo IP statico assegnatogli manualmente.

Una volta configurato il Firewall, le tabelle di routing dei bot e del C&C server sono state modificate per fare in modo che il Firewall operasse da gateway per questi dispositivi. In questo modo quando i dispositivi all'interno della botnet Vlan cercheranno di comunicare all'esterno della rete il Firewall filtrerà le comunicazioni.

ClearOS ha diversi moduli installabili per configurare le funzionalità di Firewall. In questo caso è stato utilizzato un modulo basato su Iptables (2.2.3).

Oltre alle funzioni di Firewall sono state usate le funzionalità di server DHCP che il sistema operativo offre attraverso modulo dedicato. Per mezzo di questo sono stati assegnati gli indirizzi IP alle macchine interne alla botnet Vlan.

In seguito alla conferma della connettività Internet delle macchine interne alla botnet Vlan, sulle macchine Linux è stato installato per comodità un ambiente Desktop, di cui erano privi.

In fine per velocizzare il deploy dei nodi della botnet nelle rispettive macchine e per facilitare il testing è stato installato Docker all'interno delle macchine della botnet Vlan.

Per lo scambio di file con macchine fuori dal perimetro del server ESXi è possibile usare SSH Tunneling, a patto che il dispositivo interessato sia connessa alla stessa VPN del server.

Di seguito verranno descritte più nel dettaglio le principali tecnologie sopra citate.

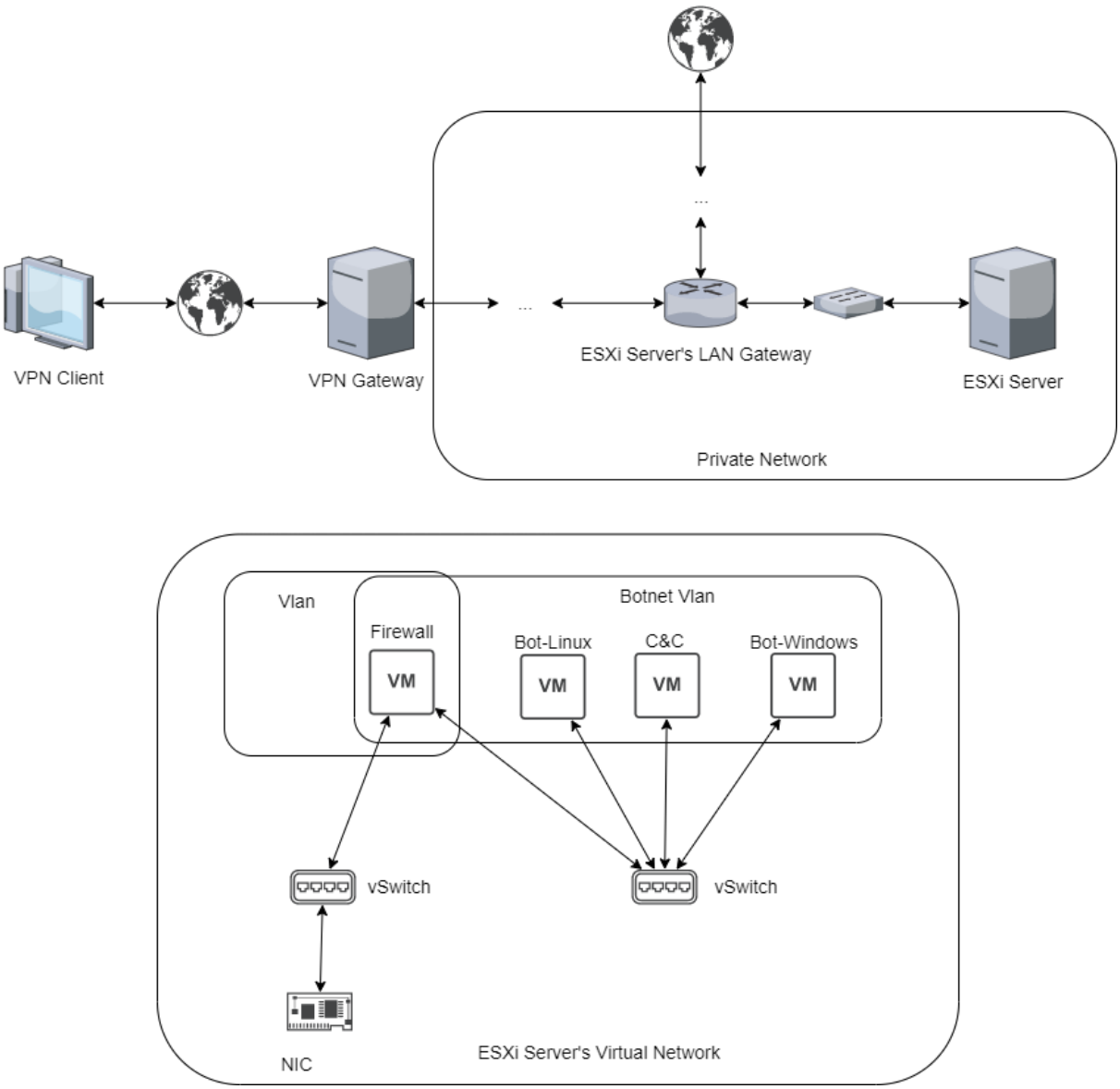



Figura 2.1: Infrastruttura di testing iniziale.

## 2.1 Server VMWare ESXi

Il server sul quale sono stati effettuati i test usa ESXi (Elastic Sky X integrated) [16] sviluppato da VMWare come *hypervisor bare metal* per la creazione e gestione di macchine virtuali.

Un hypervisor, anche conosciuto come *virtual machine monitor*, è un tipo di software di virtualizzazione che supporta la creazione e il management di macchine virtuali [17], [18].

Consente a un computer host di supportare diverse *guest Virtual Machine (VM)* , condividendo le risorse come memoria e potenza di calcolo. Ha il compito di astrarre il lato software di un computer dal suo hardware traducendo richieste da risorse fisiche a virtuali. L'hypervisor tratta le risorse di elaborazione (tra cui CPU, memoria, *storage*) come un pool di risorse facilmente riposizionabili tra i guest esistenti o su nuove VM. A seconda delle esigenze le risorse vengono suddivise e trasferite dall'ambiente fisico alle varie macchine virtuali. Durante l'esecuzione di una VM, quando un utente o un programma invia un'istruzione che richiede ulteriori risorse dall'ambiente fisico, l'hypervisor programma la richiesta in base alle risorse del sistema fisico, in modo che il sistema operativo e le applicazione della macchina virtuale possano accedere al pool condiviso di risorse fisiche.

Esistono due tipi di hypervisor (Figura 2.2):

**tipo 1 o *bare metal*** come un leggero sistema operativo che esegue direttamente sull'hardware di macchina host;

**tipo 2 o *hosted*** livello software che viene eseguito sopra un sistema operativo già funzionante, come una normale applicativo.

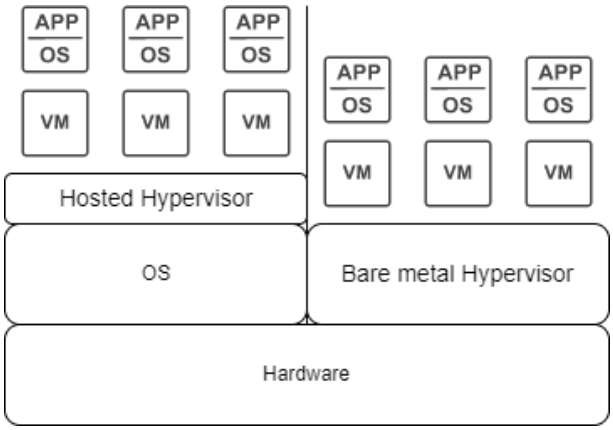



Figura 2.2: Tipologie di hypervisor.

Le principali differenze tra questi sono che l'hypervisor di tipo 1 è installato nell'unità di archiviazione della macchina host ed esegue direttamente sull'hardware come farebbe un sistema operativo, mentre il tipo 2 è installato come un normale software ed esegue sopra un sistema operativo di cui sfrutta le capacità di virtualizzazione. L'hypervisor bare metal tipicamente risulta essere più sicuro in quanto non in contatto con il sistema operativo. Infatti in caso di malfunzionamento del sistema operativo o in caso risulti vulnerabile a possibili attacchi comprometterebbe la sicurezza dell'intera struttura virtualizzata.

Inoltre i *tipo 1* risultano più efficienti e veloci in quanto le richieste di risorse tra hardware e hypervisor non devono passare attraverso il livello aggiuntivo del sistema operativo che ne fa da tramite.

ESXi offre anche capacità di virtualizzazione di rete [19]. Attraverso la virtualizzazione di rete, è possibile creare reti virtuali di macchine che utilizzino gli stessi protocolli delle

reti fisiche, sia all'interno di un singolo host ESXi Server che tra diversi host, per scopi di sviluppo, testing o per deployment in produzione. Le componenti principali della rete virtualizzata di ESXi sono i virtual switch e i virtual adapter. I virtual switch permettono alle macchine virtuali all'interno di un singolo host di comunicare tra loro senza la necessità di hardware di rete aggiuntivo, utilizzando gli stessi protocolli degli switch fisici. Inoltre, gli switch virtuali di ESXi supportano le VLAN, compatibili con le implementazioni standard dei vendor di terze parti. Ogni macchina virtuale può essere configurata con uno o più adattatori Ethernet virtuali, ognuno dei quali ha un proprio indirizzo IP e MAC, conferendo alle macchine virtuali le stesse proprietà di quelle fisiche dal punto di vista della rete.

## 2.2 Firewall

Un Firewall è un dispositivo per la sicurezza di rete che permette di filtrare il traffico in entrata e in uscita in base a regole prestabilite dall'utente [20]. Costituiscono una barriera tra le reti interne, sicure e controllate, e le reti esterne che possono essere affidabili o meno. In questo caso il Firewall verrà usato per l'esatto opposto, col fine ultimo di salvaguardare la rete esterna. Infatti è la rete interna che sarà considerata pericolosa in quanto sarà popolata da host infetti potenzialmente in grado di espandere l'infezione in rete.

Un Firewall può essere costituito da un componente hardware, software o entrambi. Tra i diversi tipi di Firewall ci sono:

**Packet filtering** È il tipo di Firewall più comune, anche chiamato *network layer firewall* o *stateless firewall*. Tratta ogni frame o pacchetto individualmente, accettandolo o negandolo all'interno del network di destinazione in base a regole prestabilite. Tra i criteri che è possibile comparare con le informazioni dei pacchetti ci sono: tipo di protocollo, indirizzo sorgente o destinazione, porte di origine o destinazione, direzione del traffico, etc.;

**Circuit-level gateway** Monitora l'*handshake* TCP tra l'host locale e quello remoto per determinare se la sessione che sta per essere inizializzata è legittima in base all'affidabilità del sistema remoto. Non ispeziona i pacchetti di sua iniziativa. Applica meccanismi di sicurezza quando si stabilisce una connessione. Una volta che questa è stabilita i pacchetti possono fluire senza ulteriori controlli. Il server che riceve richieste filtrate da questo Firewall vedrà nei messaggi di rete le informazioni del Firewall e non quelle del client originario, nascondendone l'identità;

**Statefull filtering** Mantiene traccia di tutte le connessioni che passano attraverso esso e può determinare se un pacchetto è l'inizio di una connessione, parte di una connessione già stabilita, o un pacchetto invalido. A tal scopo tiene un record in *cache* per ogni connessione aperta. Quando il primo pacchetto di una nuova connessione

occorre, il Firewall ne compara le informazioni con delle regole prestabilite e su questa base decide se bloccare o accettare la connessione con i relativi pacchetti;

**Application layer** Un Application Layer Firewall, anche chiamato Application proxy Firewall, protegge le risorse di rete filtrando messaggi a livello applicativo. Più complicato di un *packet filtering Firewall* in quanto esso è in grado di comprendere il protocollo applicativo e i dati, oltre a poter intercettare ogni informazione destinata all'applicazione. Sulla base delle informazioni disponibili per prendere decisioni, può autenticare utenti e giudicare se dati che dovrebbero fluire possono presentare una possibile minaccia. Combina alcune degli attributi del packet filtering Firewall con quelli del circuit-level gateway Firewall. Capace di filtrare pacchetti non solo per i protocolli a cui è destinato ma anche sulla base di altre caratteristiche.

L'application proxy Firewall viene detto proxy service mentre la macchina host che li esegue viene chiamato *application gateway.*

Anche se risulta più sicuro di altre soluzione Proxy, può impattare negativamente la performance di rete;

**Multilayer inspection** Questo tipo di firewall è in grado di analizzare i pacchetti a livello di rete, trasporto e applicazione, offrendo un elevato livello di controllo sul traffico di rete. Inoltre, il firewall multilayer inspection è in grado di mantenere lo stato dei pacchetti, ovvero di ricordare a quali connessioni appartengono, in modo da consentire un controllo più preciso su quali pacchetti vengono autorizzati a passare attraverso il firewall, in modo analogo agli statefull filtering Firewall.

Navigando sul web si possono trovare numerose tecnologie atte allo scopo. In questo caso è stato scelto ClearOS come sistema operativo capace di eseguire le funzionalità di filtraggio del traffico di rete.

### 2.2.1 ClearOS

ClearOS [21] è un sistema operativo open source basato su CentOS. Ideato per le piccole medie imprese che vogliono soluzioni flessibili. ClearOS offre diverse capacità tra cui Intrusion Detection, Content Filtering, Firewall, Bandwith Management, Domain Controller, Mail Server, DNS Server, NAT, DHCP Server, Web Server, e molto altro. Il punto forte di ClearOS è la facilità di utilizzo oltre alla flessibilità. Tutte le capacità di questa piattaforma sono distribuite sotto forma di moduli scaricabili al bisogno attraverso una comodissima web GUI. Oltre ai moduli gratuiti, ClearOS offre soluzioni a pagamento più adatte al mondo *enterprise*. In questo modo l'azienda interessata può comprare solo i moduli che desidera ed espandersi al bisogno acquistando ulteriori moduli.

Una volta installato, configurato e impostato come gateway all'interno della botnet Vlan, è stato installato il modulo aggiuntivo di Server DHCP, per configurare in automatico

gli indirizzi di rete delle macchine interne alla Vlan. In seguito è stato anche testato il modulo DNS, che in fase di testing non è stato utilizzato in quanto i dispositivi in rete venivano referenziati per indirizzo IP, e alcuni moduli gratuiti di Intrusion Detection System (IDS). Tuttavia i moduli di IDS gratuiti non sono stati ritenuti all'altezza del compito che avrebbero dovuto svolgere e sono stati quindi disinstallati. Va comunque considerato che non sono state testate le soluzione a pagamento che sicuramente sarebbero state più adatte al contesto. In fine prima di selezionare un modulo Firewall tra le opzioni disponibili offerte da ClearOS, sono stati studiati Netfilter (2.2.2), Iptables (2.2.3), Nftables (2.2.4) e Firewalld (2.2.5). Una volta comprese queste tecnologie è stata scelta una soluzione (modulo) basata su Iptables (Figura 2.3).

Attraverso il modulo di Firewall sono state scritte delle regole, con sintassi uguale a quella di Iptables, per le seguenti security policy:

- Non permettere traffico ipv6 da, verso, attraverso il Firewall;
- Permetti traffico da e verso la web GUI del Firewall;
- Permetti traffico DHCP da e verso il Firewall;
- Permetti traffico DNS da e verso il Firewall;
- Permetti traffico SSH da e verso il Firewall;
- Permetti traffico da e verso gli Intrusion Detection System[1];
- Nega tutto il traffico non precedentemente permesso da/verso/attraverso il Firewall.

Quando era necessario connettere le macchine della Vlan a Internet, venivano disattivate le regole opportune per permettere il traffico di rete.

### 2.2.2 Netfilter

Netfilter [22], [23] è un framework per Kernel Linux 2.4.x e superiori, che permette di intercettare e manipolare i pacchetti di rete. Attraverso Netfilter è possibile realizzare diverse funzionalità di rete tra cui:

- Stateless firewall;
- Statefull firewall;
- Tutti i tipi di traduzione di indirizzi IP o porte (eg. NAT).

[1] la macchina con ClearOS è stata utilizzata per comodità come punto di interfacciamento con la web GUI del Firewall e le web GUI degli IDS così che si potesse avere il controllo della rete attraverso un unico web browser.

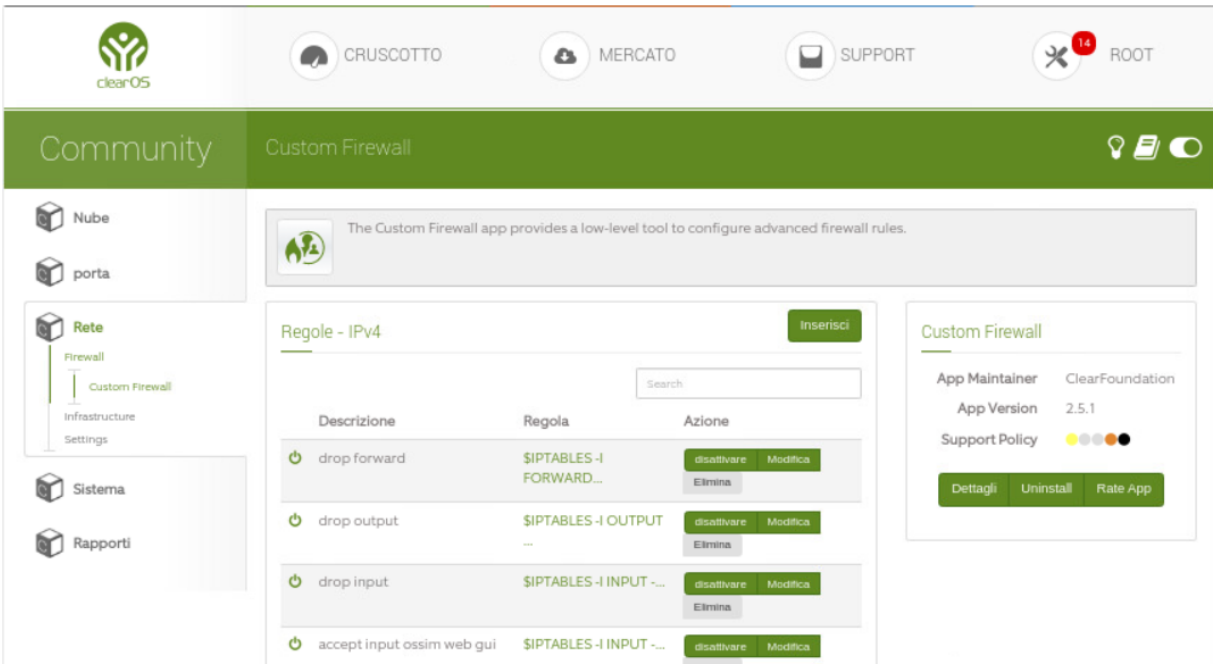



Figura 2.3: Interfaccia web di modulo firewall all'interno di ClearOS.

Attraverso gli *hook* di Netfilter i moduli del Kernel possono registrare funzioni di *callback* in diverse posizioni chiave lungo il Linux network stack. Quando un pacchetto progredisce attraverso lo stack (in entrata o in uscita), al raggiungimento di uno di questi punti chiave, l'hook associato a questa posizione specifica si attiverà, chiamando le funzioni di callback dei moduli Kernel registrati presso di esso. In particolare gli hook sono:

**NF_IP_PRE_ROUTING** Questo hook verrà attivato da qualsiasi pacchetto in entrata subito dopo essere entrato nello stack di rete. Questo hook viene elaborato prima che venga presa qualsiasi decisione di instradamento relativa a dove inviare il pacchetto;

**NF_IP_LOCAL_IN** Questo hook verrà attivato dopo il routing di un pacchetto in ingresso, se il pacchetto è destinato al sistema locale;

**NF_IP_FORWARD** Questo hook verrà attivato dopo il routing di un pacchetto in ingresso, se il sistema deve eseguire il forward del pacchetto;

**NF_IP_LOCAL_OUT** Questo hook verrà attivato da qualsiasi traffico in uscita dal sistema locale, nel momento in cui entra nel network stack;

**NF_IP_POST_ROUTING** Questo hook verrà attivato da qualsiasi traffico in uscita o se il sistema deve fare il forward del pacchetto, dopo che il routing è avvenuto e prima che il pacchetto venga spedito in linea.

I moduli Kernel registrati in questo modo, possono assegnare un numero di priorità per identificare deterministicamente in quale ordine verranno eseguite le funzioni di callback.

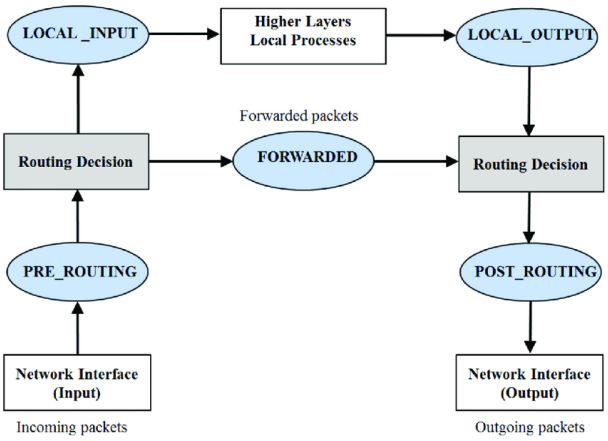



Figura 2.4: Flusso del pacchetto attraverso il Linux network stack con Netfilter hook [24].

Quando una funzione di callback viene eseguita, il modulo può decidere di alterare il pacchetto responsabile dell'esecuzione della funzione. Al termine dell'esecuzione, inoltra al framework la propria decisione sul futuro del pacchetto. Il pacchetto può quindi essere scartato, ignorato, inserito in coda per poter essere maneggiato da software che appartengono allo *spazio utente*[2], o potrebbe proseguire il suo corso attraverso lo stack.

Per interagire con il framework Netfilter dunque è necessario sviluppare un modulo Kernel che possa registrare funzioni di callback presso gli hook o scrivere un applicativo che interagisca con la coda nello spazio utente (user mode).

Per semplificare il processo e non dover specificatamente scrivere un software per filtrare i pacchetti spesso si usano programmi come Iptables e Nftables. Attraverso questi software è possibile definire insiemi di regole di filtraggio dei pacchetti.

### 2.2.3 Iptables

Iptables usa tabelle per organizzare queste regole. Nello specifico le tabelle *built-in* di Iptables sono:

**Filter table** Per controllare il flusso dei pacchetti che entra ed esce dal sistema;

**NAT table** Per ridirigere connessioni verso altre interfacce di rete;

**Mangle table** Per modificare header di pacchetti;

**Raw table** Provvede meccanismo per marcare pacchetti ed escluderli dal tracking di connessione[3];

[2]Netfilter è un framework che esegue a livello di Kernel in Kernel mode e quindi l'utente che esegue software in user mode non potrebbe interagire direttamente con questo.

[3]Netfilter riesce a identificare un pacchetto come parte di una connessione e Iptables quindi sfrutta questa capacità in automatico.

**Security table** Usata per interagire con SELinux[4].

Ogni tabella è a sua volta organizzata in *chain* che contengono le vere e proprie regole. Mentre le tabelle rappresentano lo scopo generale delle regole che contengono, le chain rappresentano i Netfilter hook a cui si riferiscono e da cui vengono attivati. Le chain quindi determinano quando le regole al loro interno verranno valutate. I nomi delle chain fanno riferimento ai nomi degli hook a cui sono associati e sono:

- Prerouting;
- Input;
- Forward;
- Output;
- Postrouting.

Dato che alcune operazioni/decisioni hanno senso solo in alcuni punti del network stack, non ogni tabella ha tutte le precedenti chain registrate presso i rispettivi Netfilter hook. Ad esempio la Filter table hanno solo le Input, Output e Forward chain registrate.

Come detto le regole sono posizionate all'interno delle chain delle tabelle specifiche. Quando una chain è chiamata dal rispettivo hook, i dettagli del pacchetto vengono comparati con le regole con ordine. Ogni regola è divisa in due parti. La prima parte è quella interessata dei criteri di match con le informazioni del pacchetto. Tra i criteri di match ci sono il protocollo usato, la sorgente, la destinazione, lo stato della connessione, le porte utilizzate, e molto altro. La seconda parte invece detta *target* specifica un azione da compiere sul pacchetto. Tra i target più comuni ci sono Accept, Reject, Drop e Jump.

Oltre alle chain built-in è anche possibile creare chain personalizzate. Queste non sono associate ad hook Netfilter e sono raggiungibili solo attraverso il target Jump, ovvero saltando da una chain built-in a questa. Quando un pacchetto esegue il match con una regola avente target Jump, il controllo può saltare all'interno di chain definite dall'utente. In caso non vi sia match all'interno di queste, il controllo tornerebbe alla chain d'origine. In questo modo è possibile aumentare l'organizzazione del sistema.

### 2.2.4 Nftables

Nftables [25] è il successore di Iptables, basato anch'esso sul framework Netfilter. Risulta più flessibile, scalabile ed efficiente nel classificare i pacchetti. La struttura è uguale a quella di Iptables, ovvero è composto di table, che contengono chain, che a loro volta contengono regole. Tra i vantaggi che offre rispetto a Iptables ci sono:

[4]Security Enhanced Linux è un modulo di sicurezza del Kernel Linux che fornisce insieme di strumenti di sicurezza.

- Una singola regola può avere più target mentre Iptables può averne solo uno per regola;
- Possibilità di creare regole più complesse attraverso strutture dati come map, set, concatenazioni, diminuendo quindi il numero di regole da controllare e quindi aumentando l'efficienza;
- Non ha table e chain predefinite, sfrutta combinazione di parametri selezionabili come il tipo di hook e lo scopo per determinarne il comportamento;
- Più facile da configurare;
- Possibilità di gestire regole IPv4 e IPv6 attraverso lo stesso tool, mentre in Iptables le configurazioni sono gestite separatamente[5].

### 2.2.5 Firewalld

Firewalld [26] è uno strumento di gestione del Firewall che sfrutta Nftables o Iptables. È il tool di gestione Firewall predefinito di RHEL 7, CentOS, Fedora e altre distribuzioni Linux. Esso agisce come un'interfaccia front-end e non è quindi un Firewall *stand-alone*. Firewalld è strutturato in zone. Le interfacce di rete o range di indirizzi IP possono essere assegnati a zone specifiche. Una zona dichiara quali porte o servizi sono permessi o bloccati, oltre a poter definire ulteriori regole più complesse. Le zone sono formulate come file XML e sono posizionate all'interno della directory di Firewalld. Il funzionamento alla base di Firewalld è indipendente dal fatto che venga utilizzato Nftables o Iptables. Al boot esegue uno script Python che converte e carica le regole, configurate con Firewalld, in Nftables o Iptables. Sono anche disponibili interfacce grafiche che ne semplificano ulteriormente l'utilizzo.

## 2.3 Docker

In uno scenario di normale sviluppo di un'applicazione *end-to-end*, dove si vogliono utilizzare diversi servizi che devono cooperare tra loro, ci possono essere le seguenti problematiche da tenere in considerazione:

**Compatibilità/dipendenze** I diversi servizi dell'applicativo possono richiedere differenti versioni della stessa libreria. Le librerie devono essere compatibili con il sistema operativo e l'hardware sottostante. Inoltre queste hanno dipendenze che devono essere soddisfatte. Il problema di soddisfare tutte queste esigenze contemporaneamente per permettere il corretto funzionamento dell'applicativo è conosciuto come *Matrix from hell*;

[5] viene usato tool *iptables* per le configurazioni IPv4 e *ip6tables* per quelle IPv6.

**Tempi di Setup** La configurazione di un nuovo ambiente di sviluppo da zero sottintende che lo sviluppatore debba assicurarsi di avere il giusto sistema operativo e le librerie necessarie nelle versioni corrette, e che le dipendenze siano risolte correttamente. Questo processo può essere laborioso e non banale, oltre che richiedere una considerevole quantità di tempo;

**Differenti ambienti di sviluppo** Essendoci diversi ambienti di sviluppo, test e produzione, non è detto che, una volta completata una modifica, l'applicativo funzioni correttamente quando viene eseguito in un ambiente differente.

Questi sono alcuni dei motivi per cui può essere utile containerizzazione i servizi. Con containerizzazione si intende raggruppare il codice software e tutti i relativi componenti necessari, come librerie, framework e altre dipendenze, in modo che risultino isolate in un proprio contenitore, detto *Container*.

In questo modo è possibile eseguire e spostare il software o l'applicazione contenuta all'interno in qualsiasi ambiente o infrastruttura indipendentemente dal sistema operativo eseguito. Lo scopo quindi della containerizzazione è il poter eseguire applicativi, servizi, processi ovunque, in qualsiasi momento e in un ambiente isolato, risolvendo in questo modo le problematiche sopra citate.

Un Container può essere visto come un processo che dà all'utente l'illusione di eseguire sopra un sistema operativo, si parla infatti di *lightweight process virtualization*. In particolare sfruttano componenti del Kernel Linux per virtualizzare, isolare e gestire le risorse. I

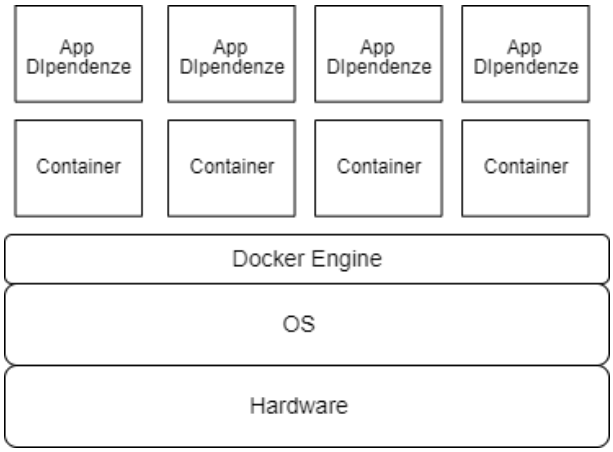



Figura 2.5: Docker containers.

Container condividono il Kernel del sistema operativo sul quale eseguono, così facendo non hanno bisogno di dover virtualizzare il sistema operativo per poter eseguire componenti software [27]. Essi *vivono* solo per il tempo dell'esecuzione del processo o del servizio che ospitano. Per questi motivi dunque sono più leggeri e meno dispendiosi in termini di potenza di calcolo di una macchina virtuale, oltre a risultare più veloci in termini di *boot* e *shutdown*.

Dato che i Container sono soluzioni a basso livello solitamente difficili da utilizzare, si usano soluzioni software come Docker [28] che sfruttano le capacità del Kernel Linux per crearli e gestirli.

Docker usa le Docker Image, che rappresentano una serie di istruzioni da eseguire sequenzialmente, al fine di creare un Docker Container [27]. Contengono inoltre il codice, le librerie necessarie, tool, dipendenze e tutto quello che è indispensabile per eseguire l'applicativo o il servizio contenuto. Quando viene eseguita una Docker Image, si creano una o più istanze di Container, usando quell'immagine come template.

Le Docker Image vengono create a partire da file di configurazione, detti Dockerfile. Nel Dockerfile viene specificata la Docker Image di partenza e tutte le operazioni, che su questa verranno eseguite, atte a costruire il template per i futuri Container. Per un esempio di Dockerfile si osservi la sezione di codice 2.1. Quando viene eseguita la *build*

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
```

Listing 2.1: Esempio di Dockerfile.

dell'immagine, ogni operazione all'interno del Dockerfile costituisce un livello che viene posta sopra al livello dell'operazione precedente Figura 2.6. Ogni livello contiene solo le differenze dal livello precedente. In questo modo se la build dovesse fallire, al tentativo successivo riprenderebbe dal livello in cui si era interrotto. Questo meccanismo permette anche di accelerare i tempi in caso di *rebuild* in seguito a piccoli cambiamenti nel Dockerfile, in quanto i livelli verranno ricostruiti dal primo cambiamento a seguire. I livelli sono *read*

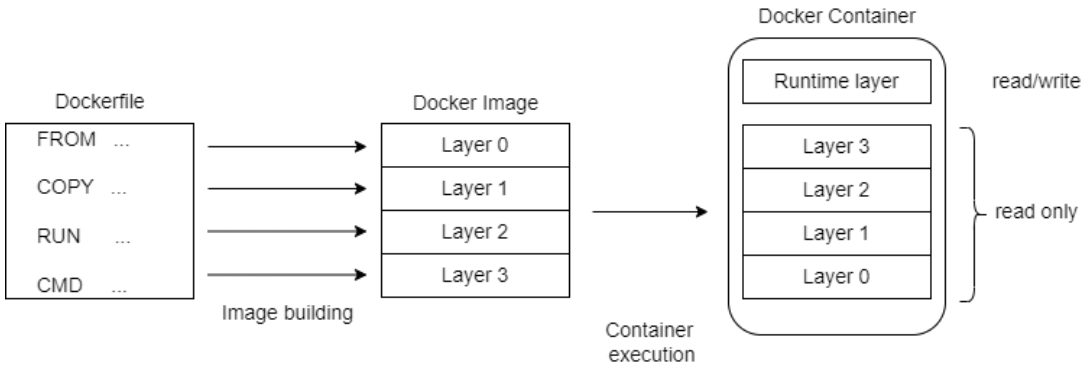



Figura 2.6: Creazione dei livelli dell'immagine a partire dal Dockerfile ed esecuzione del container.

*only* e sono condivisi tra tutte le istanze di Container della stessa immagine. Quando viene eseguito un Container, si crea un ulteriore livello che è *readable* e *writeable*, che chiameremo 'livello container'. Quando viene modificato un file read only all'interno di un livello non

modificabile, si crea una copia locale di quel file nel livello container e la modifica viene eseguita sulla copia[6].

Quando si crea un nuovo file questo viene aggiunto al livello container. Nel momento in cui il Container viene rimosso o finisce la propria esecuzione, tutto il contenuto del livello container viene cancellato. Se è richiesta persistenza di alcune directory e del relativo contenuto è necessario eseguire il *mount* di una directory dell'host all'interno del container con direttive apposite. Questo equivale a livello logico all'associare una directory dell'host a una directory del Container, così che al termine dell'esecuzione i dati all'interno della directory del Container persistano nella directory dell'host.

A livello di rete, Docker, crea di default tre network a cui i Container possono essere associati:

**Bridge** Rete interna privata, solitamente con indirizzo 172.17.0.0/16. Se non viene specificato diversamente un Container viene associato a questo network di default;

**None** Un Container associato a questa tipologia di rete non potrà comunicare con nessun altro Container;

**Host** Il Container viene associato direttamente a rete di host.

I Container possono comunicare tra loro se sono associati alla stessa rete, permettendo quindi di creare applicativi complessi in cui più servizi comunicano tra loro.

Entrando nel dettaglio (Figura 2.7), il network 'Bridge' è un bridge di rete, chiamato di default *docker0*, creato quando viene inizializzato il servizio Docker. Ogni container appartiene ad un namespace di rete, per fare in modo che abbia una copia dello stack di rete dell'host e che sia isolato. I container associati al network 'Bridge', sono in realtà collegati al bridge docker0 attraverso interfacce *vEth*. Quando un container ha necessità di comunicare all'esterno dell'host viene eseguito Network Address Translation (NAT). I container associati al network 'Host', sono in realtà associati allo stesso namespace di rete dell'host.

Se un Container è associato a una rete interna, può essere raggiunto dal network dell'host che lo ospita attraverso un'operazione di *port mapping*. In questo modo si può associare una porta del Container a una porta dell'host, quindi un pacchetto che fluisce attraverso la porta specifica dell'host verrà consegnato in automatico alla porta specifica del Container.

Docker può prelevare le immagini da repository di immagini Docker detti Registry, che possono contenere anche centinaia di migliaia di immagini di applicazioni e servizi pronti per essere scaricati ed eseguiti in Container.

Per sviluppare applicazioni complesse potrebbe essere necessario eseguire molteplici Container contemporaneamente. Per evitare di dover eseguire manualmente ogni container, esistono tool di orchestrazione dei Container. Un esempio di questo genere di tool è *Docker*

[6]tecnica definita come *Copy on write*.

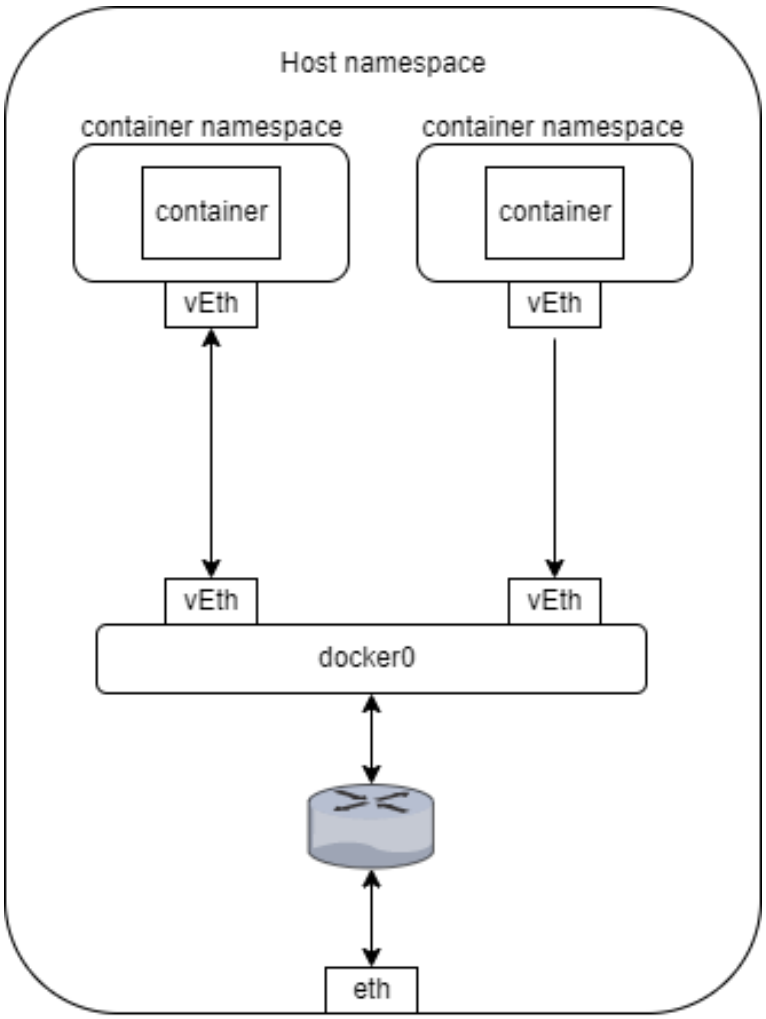



Figura 2.7: Docker network.

*Compose*, che permette grazie a un unico file di configurazione (sezione di codice 2.2) in formato YAML di eseguire contemporaneamente molteplici Container.

```
version: "3.9"
services:
  web:
      build: .
      ports:
          - "8000:5000"
      volumes:
          - .:/code
      environment:
          FLASK_DEBUG: "true"
  redis:
      image: "redis:alpine"
```

Listing 2.2: Esempio di file di configurazione Docker Compose per l'esecuzione di molteplici container.

# 3 Intrusion Detection System (IDS)

Un Intrusion Detection System è un software o hardware che identifica azioni non legittime su computer, col fine di preservare la sicurezza del sistema. In particolare ha l'obiettivo di identificare traffico di rete e uso di computer non legittimi, che non potrebbero essere identificati da un normale Firewall. Si possono identificare principalmente due tipi di IDS [29]:

- Signature-based IDS (SIDS);
- Anomaly-based Intrusion Detection System (AIDS).

Con SIDS si intende strumenti di rilevazione basati su tecniche di *pattern matching* per scovare attacchi già noti. Quando la *signature*[1] di un evento combacia con la signature di un'intrusione nota all'interno del database di signature, viene generato un segnale d'allarme.

Questi solitamente hanno un'ottima accuratezza per quanto riguarda la rilevazione di minacce già note ma non possono rilevare *attacchi zero-day*[2] dato che non esistono signature di tale attacco sino a quanto la signature di questo viene estratto e aggiunto al database. Gli approcci SIDS soffrono anche nell'identificare le varianti di malware noti, in quanto variazioni nel codice potrebbero comportare una signature differente al malware originale.

Approcci tradizionali di SIDS sfruttano i *log*[3] di una macchina host, generati dal sistema o applicazioni software, e l'analisi del traffico di rete per scovare azioni, comandi, pattern riconducibili a malware noti.

L'approccio Anomaly-based parte dal presupposto che il comportamento non legittimo differisca da quello legittimo. Viene creato il modello di comportamento del sistema. Le tecniche usate per effettuare appartengono alle seguenti categorie:

- Machine Learning;
- Statistical-based;
- Knowledge-based.

[1] attraverso l'estrazione di *feature* prefissate si può generare la firma di un particolare evento per usarla come identificativo univoco di questo.

[2] attacchi informatici che sfruttano vulnerabilità di cui non si conosce l'esistenza o che non sono state ancora risolte.

[3] documentazione inerente un determinato evento che ha avuto esito

Il modello viene creato in una prima fase di *training* in cui viene osservato il normale comportamento del sistema. Segue una fase di *testing* in cui si usa un nuovo data set per stabilire l'accuratezza del sistema nel rilevare intrusioni. Ogni significativa deviazione del comportamento osservato dal modello è considerata un'anomalia e trattata come un'intrusione.

Il vantaggio primario di questo approccio è la capacità di rilevare attacchi zero-day dato che è basato sul comportamento e non sulle signature.

Un'altra classificazione possibile degli IDS è basata sul tipo di sergenti dati che utilizzato per le rilevazioni. In particolare si parla di:

- Host-Based IDS (HIDS);
- Network-Based IDS (NIDS).

Gli HIDS analizzano i log originati dal sistema host e dalle *sorgenti di Auditing*[4]. Sono capaci di individuare minacce che non usano traffico di rete.

I NIDS monitorano il traffico di rete estratto attraverso appositi strumenti di cattura. Possono essere usati per monitorare molteplici computer su stessa rete al contrario di HIDS che sono installati su una macchina specifica. Sono capaci di individuare minacce esterne prima che si propaghino tra dispositivi. A seconda della rete da monitorare possono essere richieste capacità di hardware elevate per gestire correttamente alti traffici di rete.

Il giusto deploy di molteplici NIDS in determinati punti strategici della rete in combinazione con HIDS e Firewall possono provvedere a una protezione robusta sia ad attacchi esterni che interni.

Di seguito verranno analizzati i due NIDS utilizzati ma prima è necessaria una piccola premessa sulla duplicazione del traffico di rete, operazione necessaria per poter analizzare una copia dei dati che fluiscono attraverso la rete.

## 3.1 Duplicazione del traffico di rete

La duplicazione del traffico di rete può avvenire in due modi [30]:

- Inline;
- Mirroring.

Con mirroring si intende utilizzare le capacità built-in degli switch o router, mentre con inline si intende utilizzare un dispositivo apposito di duplicazione del traffico lungo la linea di rete.

[4]sorgenti di eventi generati in seguito a controlli specifici.

### 3.1.1 Port Mirroring

Il port mirroring è una funzionalità spesso già integrata nei router e switch a livello enterprise. Il traffico che passa attraverso una porta dello switch o router viene 'specchiato' verso un'altra porta. La porta selezionata per eseguire l'output della copia del traffico è detta Switch Port Analyzer (SPAN) o *mirror port*. Dato che il compito primario di switch o router è il forwarding dei pacchetti, la copia dei dati è data con bassa priorità. Questo può portare a drop di pacchetti anche quando vi è basso volume di traffico. Va considerato inoltre che queste porte utilizzano slot sullo switch o router che potrebbero altrimenti venir utilizzati per gli scopi di forwarding.

In caso di configurazioni errate, l'utilizzo di una porta SPAN potrebbe intaccare la performance di rete.

Usa una singola porta di uscita per aggregare più link e i relativi canali di up-link e down-link. Così facendo è molto probabile che la linea di output si saturi portando a perdita dei pacchetti.

Usato solitamente per monitorare bassi volumi di dati in luoghi in cui non è presente un Test Access Port.

### 3.1.2 Test Access Port

Il Test Access Port (TAP) è un device di cattura del traffico di rete posizionato lungo la linea cablata. Attraverso a uno splitter interno è capace di copiare i dati e reindirizzarli verso porte di uscita.

Separa canali di up-link e down-link diminuendo la possibilità di drop di pacchetti. Il TAP è in grado quindi di creare una copia esatta del flusso bidirezionale a pieno rate di linea, che ne consegue nella piena fedeltà di monitoraggio. Ci sono TAP passivi e attivi. Quelli passivi non sono alimentati e quindi in caso di malfunzionamenti della linea elettrica il sistema può comunque funzionare correttamente. Tuttavia i TAP passivi in rame distorcono il segnale rendendoli inadoperabili per linee con velocità superiori a un gigabit al secondo. Esistono TAP passivi ottici per ovviare al problema, al costo di un indebolimento del segnale ottico sulla linea. Quelli attivi sono alimentati da corrente elettrica e ritrasmettono e duplicano il segnale senza introdurre distorsioni.

I TAP passivi non richiedono configurazioni o interventi particolari, offrendo continuo accesso ai dati.

### 3.1.3 Port mirroring via software

Altri approcci meno scalabili sono possibili, come ad esempio l'utilizzo di software e tool per eseguire la copia e l'inoltro del traffico di rete. In questo caso è stato usato Iptables per fare test di port mirroring via software, utilizzando la seguente regola:

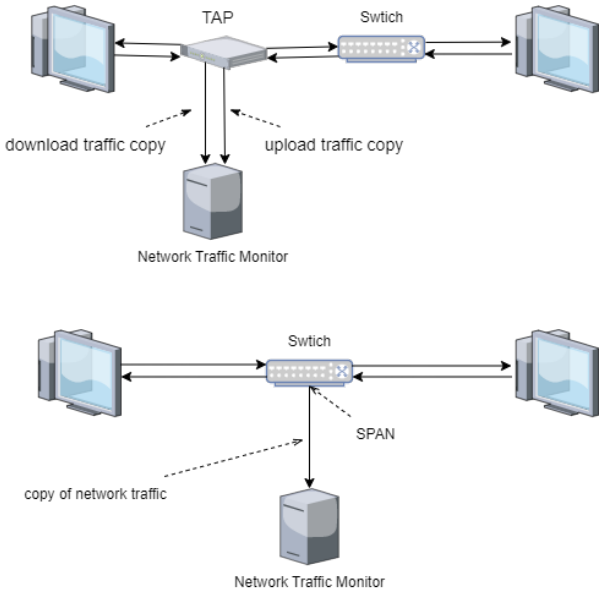



Figura 3.1: Esempio di applicazioni di SPAN e TAP.

```
iptables -t mangle -I PREROUTING -j TEE --gateway 100.0.0.2
```

Questo comando aggiunge una regola all'inizio della chain di *Prerouting* nella tabella *Mangle* in cui specifica di inviare una copia del pacchetto al nodo di rete specificato.

## 3.2 Security Onion

Security Onion [31] è una piattaforma gratuita e open source per Network Security Monitoring (NSM) ed Enterprise Security Monitoring (ESM) [32]. Con ESM si intende l'aggiunta delle capacità di monitoring degli *end-point* alla soluzione NSM.

Nella figura 3.2 si può vedere un esempio di utilizzo di Security Onion all'interno di un network enterprise. In questo caso il traffico di rete viene duplicato e inviato attraverso porte TAP a un Security Onion che ne esegue l'analisi. Nel mentre anche i log dei sistemi monitorati vengono raccolti e analizzati. In questo modo si hanno funzioni di NIDS e di HIDS.

Di seguito verranno brevemente descritti i principali software che vengono utilizzati all'interno di Security Onion per poi analizzare l'architettura distribuita di tale IDS e il suo deploy all'interno dell'infrastruttura di testing.

### 3.2.1 Tool integrati

Di seguito verranno descritti i principali tool utilizzati da Security Onion durante il funzionamento all'interno di un'architettura distribuita. L'architettura in questione verrà meglio analizzata successivamente.

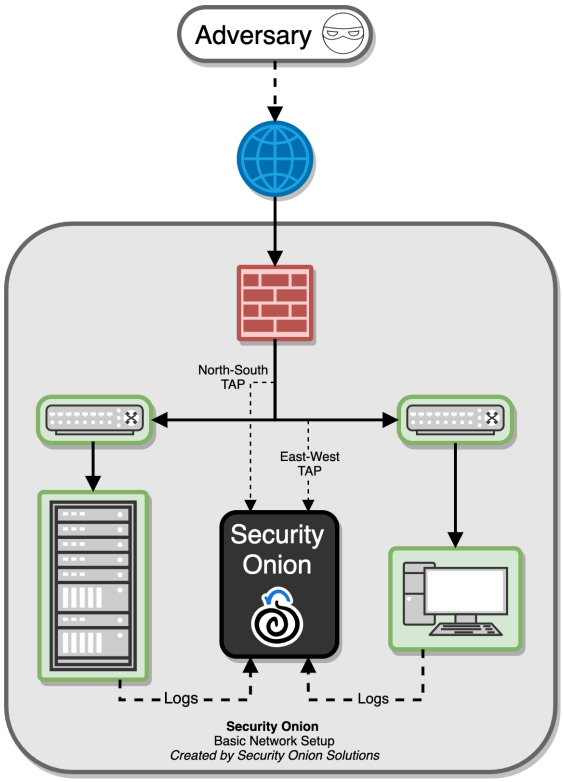



Figura 3.2: Esempio di utilizzo di Security Onion all'interno di un network enterprise [33].

### Suricata

Suricata [34] è un NIDS open source, maturo e robusto. Ispeziona il traffico di rete usando regole e linguaggio di signature oltre ad avere supporto a LUA [35] scripting per rilevazione di minacce complesse. Attraverso l'integrazione di linguaggio di scripting è possibile, ad esempio, elaborare il payload di un pacchetto di rete prima di cercare possibili fingerprint match, aumentando le difese contro le tecniche di offuscamento. Suricata è anche in grado di agire attivamente in caso di intrusione, per questo motivo è anche classificato come Intrusion Prevention System (IPS). Inoltre è anche in grado di memorizzare PCAP file[5] per analisi successive. Viene quindi principalmente usato come SIDS all'interno di Security Onion.

### ZEEK

ZEEK [36] è un tool di analisi del traffico di rete. Al contrario di Suricata non è ottimizzato per la ricerca di signature e non fornisce memorizzazione della cattura del traffico di rete. Il suo scopo è interpretare il traffico di rete e generare notevoli volumi di informazioni riguardanti il comportamento della rete, fornendo visibilità e contesto, oltre a offrire capacità come:

[5] file contenenti pacchetti catturati dal traffico di rete.

- Analisi della performance di rete;
- Profiling di host e servizi;
- Anomaly detection.

Security Onion viene affiancato a Suricata per la rilevazione di minacce attraverso l'analisi del traffico di rete. Mentre Suricata permette di identificare attacchi basandosi sul signature matching, ZEEK permette di reperire il contesto dell'attacco, offrendo tutti i metadata necessari per analizzare al meglio la situazione e poter quindi agire di conseguenza.

### Stenographer

Stenographer [37] è un utility per la cattura del traffico di rete cui scopo è memorizzare tutto il traffico catturato su disco per scopi di intrusion detection e incident response. Fornisce un'implementazione ad alta performance della scrittura dei pacchetti da NIC a disco e gestisce la cancellazione automatica quando la memoria si satura. Offre inoltre metodi per la lettura rapida ed efficiente di set specifici di pacchetti oltre a un linguaggio di query.

Si può accedere a cattura dei pacchetti memorizzai attraverso una qualsiasi interfaccia PCAP o attraverso Command Line Interface (CLI).

### Strelka

Strelka [38] è un sistema di scansione di file in tempo reale, utilizzato per la ricerca e rilevazione di minacce o per analisi di incident response. Il suo scopo è l'estrazione di file e metadati su larga scala. È in grado di utilizzare regole YARA[6] per l'identificazione di minacce note.

Nel Security Onion, viene utilizzato per analizzare i file estratti da ZEEK e Suricata durante l'analisi del traffico di rete.

### Sensoroni

Sensoroni [39] è un sistema software per il monitoraggio e l'interrogazione di server o dispositivi remoti. È composto da un server e uno o più agenti. Gli agenti sono eseguiti sui dispositivi remoti (sensori) ed eseguono i *job* che sono stati impartiti e accodati dal server. Il server offre un'interfaccia utente per gli amministratori per eseguire task come il monitoraggio dei sensori e la gestione dei job.

[6] le regole YARA usano pattern come sequenze di byte o stringhe riconducibili a malware noti.

### Elastic Stack

Con Elastic Stack [40] si intende una collezione di software open source appartenenti alla compagnia Elastic. Il suo scopo è aiutare l'utente a raccogliere dati da qualsiasi sorgente e in qualsiasi formato per poter poi eseguire ricerche, analisi e visualizzazioni dei dati. Questa collezione era precedentemente chiamata come ELK stack, dove ELK rappresenta l'acronimo delle sue componenti software, ovvero:

**Elasticsearch** Motore di ricerca e analisi distribuito, basato su interfaccia RESTful e usato per moltissimi casi d'uso. Componente che rappresenta il cuore dell'Elastic Stack. Mantiene e indicizza i dati per ricerche veloci e precise. In Security Onion riceve i log e ne effettua il parsing prima di memorizzarli e indicizzarli;

**Logstash** Motore di data processing server-side basato su pipeline, capace di raccogliere dati da moltitudine di sorgenti, trasformali e inoltrarli. In Security Onion trasporta i log, su cui non è ancora stato effettuato parsing, a Elastic Search;

**Kibana** Interfaccia GUI (Figura 3.3) che permette di visualizzare i dati memorizzati in Elastic Search e di navigare l'Elastic Stack. Permette di visualizzare in tempo reale flussi di dati di notevoli dimensioni e di aumentarne la comprensione attraverso grafici.

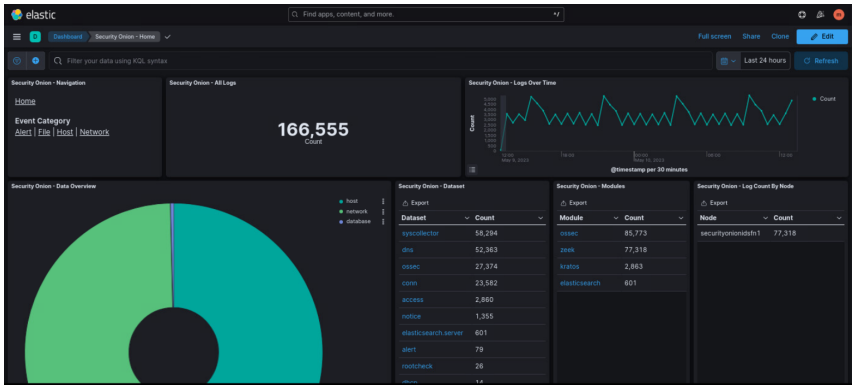



Figura 3.3: Kibana.

L'ELK stack si è evoluto nell'Elastic Stack in seguito all'aggiunta di Beats alla collezione. I Beats sono installati come agenti su dispositivi remoti e vengono utilizzati per inviare dati di diversa natura (log, metriche, dati di rete, etc.) direttamente a Elastic Search o indirettamente attraverso Logstash. Security Onion in particolare usa FileBeat per inoltrare log a Elastic Search.

Lo stack viene principalmente usato per la gestione di ingenti quantità di dati o in caso siano necessarie capacità di ricerca avanzate sui dati. In Security Onion viene utilizzato per inoltrare, ricevere, memorizzare, indicizzare, visualizzare e analizzare i vari log generati.

### ElastAlert

ElastAlert [41] è un framework per l'allerta su anomalie, picchi o altri pattern di interesse all'interno dei dati custoditi in Elasticsearch. Interroga Elasticsearch e fornisce un meccanismo di allerta in tempo reale con molteplici tipi di output.

### Curator

Curator [42] è un utility utilizzata all'interno di Elastic Stack per la gestione degli indici di Elasticsearch. In particolare:

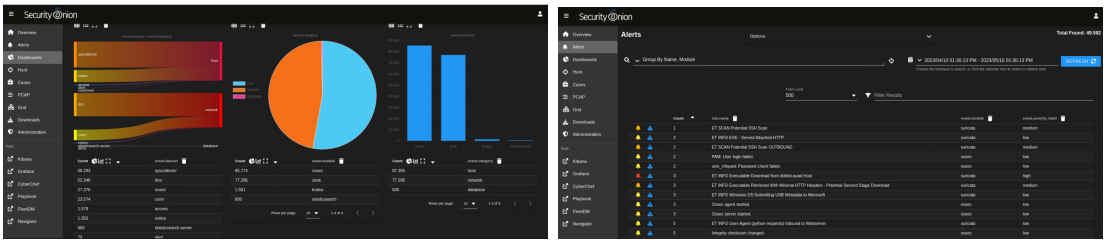
1. Ottiene lista di indici da cluster, che verrà chiamata *actionable*;
2. Itera su lista di filtri definiti da utente per rimuovere progressivamente indici dalla lista actionable;
3. Esegue diverse azioni su elementi rimasti a seguito del filtraggio.

### Redis

Redis [43] è uno store open source di strutture dati memorizzate in memoria, utilizzato come database, cache e message broker. Supporta strutture dati come stringhe, hash, liste, insiemi, insiemi ordinati, etc. I dataset vengono mantenuti in memoria principale rendendo l'accesso estremamente veloce, ma i dati possono anche essere resi persistenti all'occorrenza.

Security Onion utilizza Redis come contenitore temporaneo dei log ricevuti da Logstash. I log così memorizzati vengono poi reperiti da Logstash e consegnati a Elastic Search.

### Security Onion Console (SOC)



(a) Dashboard. (b) Alerts.

Figura 3.4: Alcune delle visualizzazioni dell'interfaccia web di Security Onion.

SOC è l'interfaccia web attraverso cui un'analista può accedere ai vari tool di analisi offerti dalla piattaforma. L'interfaccia web è divisa più sezioni. Tra le sezioni built-in vi sono:

**Alerts** Visualizza gli *alert* che Security Onion sta generando. È possibile visualizzarne i dettagli e i PCAP a cui sono associati o aggiungere l'alert a dei Case;

**Dashboard** Permette di avere un overview generale del sistema. Personalizzabile all'occorrenza;

**Hunt** Simile alla Dashboard ma più mirata alla ricerca di minacce;

**Cases** Interfaccia per gestire i Case. Attraverso i Case è possibile raggruppare più alert che si ritengono essere associati per studiare nel dettaglio la possibile intrusione o anomalia;

**PCAP** Permette di analizzare i PCAP raccolti con Stenographer, anche in relazione a un particolare evento;

**Grid** Permette di controllare lo status dei nodi appartenenti al sistema Security Onion.

Oltre alle sezioni elencate vi sono collegamenti a tool che hanno interfacce web separate come Kibana, tra cui:

**Grafana** Permette di visualizzare attraverso grafici lo status e informazioni vitali del sistema come l'uso della CPU, RAM, traffico di rete, etc. [44];

**Playbook** Consente di creare una lista di strategie di rilevazione, dette *Play*. Ogni Play è composto di obiettivo, configurazione e *follow-up*. L'obiettivo è l'oggetto da rilevare (eg. un'anomalia, un'intrusione) e necessita della configurazione necessaria per poter essere rilevato. Nel caso di Security Onion le configurazioni sono basate su ElastAlert. Con follow-up si intende le azioni da eseguire per validare e/o correggere quanto accaduto.

I risultati delle Play sono disponibili nelle sezioni Haunt, Dashboard e Kibana. Quando il risultato di una Play è di gravità elevata viene anche generato un alert visualizzabile nell'apposita sezione. Quando una Play viene attivata, la configurazione ElastAlert viene inserita in produzione;

**ATT&CK Navigator** Il Navigator ATT&CK è progettato per fornire una navigazione e un'annotazione di base delle tecniche appartenenti alla *matrice MITRE ATT&CK*[7] [45]. Può essere utilizzato per visualizzare la copertura difensiva, la pianificazione del team di attacco/difesa, la frequenza delle tecniche rilevate e molto altro. Inoltre si integra con Playbook, permettendo di visualizzare le Play in relazione alle tecniche coperte nella matrice ATT&CK per avere un'idea ancora più chiara della copertura difensiva [46].

[7] matrice standardizzata di tattiche e tecniche di minacce informatiche.

#### Host-based visibility tools

Security Onion permette anche la raccolta di molti tipi di log generati dagli host che si desiderano monitorare. L'host può scegliere tra diverse tecnologie agent-based per effettuare diversi task di controllo sull'host e inviare i relativi log a Security Onion.

Attraverso specifiche tecnologie installate sugli host da monitorare e l'analisi dei relativi log è possibile implementare il lato HIDS di Security Onion già accennato in precedenza. In alternativa, si potrebbe decidere di inviare log arbitrari usando una delle seguenti tecnologie:

**Osquery** Osquery [47] è un agent capace di astrarre l'host, sul quale è installato, a database relazionale. L'agent può quindi essere interrogato usando sintassi SQL per avere informazioni sul sistema. Questo può anche essere usato inviare i log a Security Onion;

**Wazuh** Wazuh [48] è un agent con capacità di HIDS. Verrà approfondito meglio nella sezione inerente gli agent. Permette anche l'inoltro di log a Security Onion;

**FileBeat** Come già spiegato precedentemente, FileBeat [49] è in grado di fornire trasporto di log dedicato;

**Syslog** Syslog [50] è un protocollo di rete usato per l'invio di log semplici. Utile nel caso non sia possibile installare un agent sull'end-point.

### 3.2.2 Architettura

Security Onion dispone di diverse Architetture possibili per adattarsi alle diverse esigenze e risorse, tuttavia l'architettura raccomandata è quella distribuita che verrà di seguito analizzata.

L'architettura distribuita offre maggior scalabilità e una maggior performance in quanto basta aggiungere nodi per poter maneggiare più traffico di rete o sorgenti di log.

Si compone di tre tipologie di nodi:

- Forward Node;
- Manager Node;
- Search Node.

Il Forward Node è un sensore che esegue Zeek, Suricata, Stenographer e Wazuh per generare log. In particolare, come detto, Zeek e Suricata analizzano il traffico di rete, Wazuh funge da HIDS sull'end-point e Stenographer memorizza i PCAP all'interno di esso. Tutti i log vengono inviati attraverso FileBeat a Manager Node. I PCAP vengono

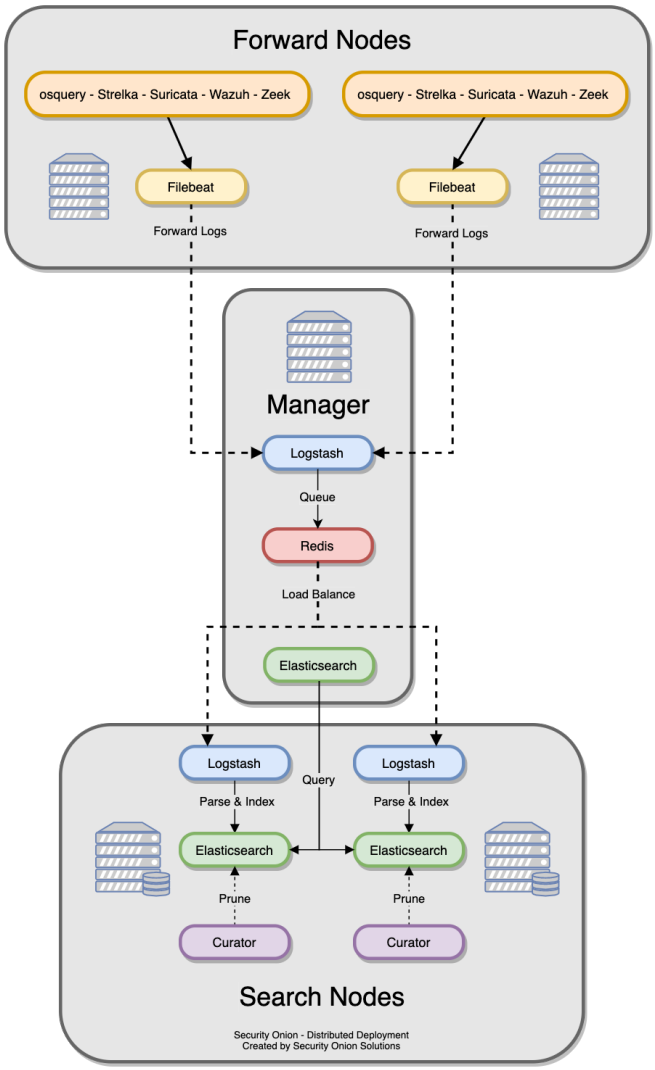



Figura 3.5: Architettura di installazione distribuita di Security Onion [51].

all'occorrenza reperiti dal Forward Node attraverso il Sensoroni agent. Strelka analizza all'occorrenza i file estratti da Zeek.

Il Manager node riceve attraverso Logstash i log inviati dai sensori e li mette in coda su Redis. Hanno anche un'istanza di Elasticsearch che viene usata per memorizzare i Case e le configurazioni centrali oltre a effettuare le query sulle istanze ElasticSearch dei Search Node. Esegue inoltre:

- Security Onion Console per permettere a un'analista di interfacciarsi con esso;
- Kibana per visualizzare i log indicizzati con Elasticsearch;
- Curator per gestire gli indici ed eseguire operazioni su essi;
- ElastAlert per generare alert in Elasticsearch;
- Wazuh come istanza HIDS locale;

Attraverso il Manager Node l'analista può interfacciarsi con Security Onion per eseguire query e reperire/analizzare dati.

Il Search Node usa Logstash per consumare log da coda in Manager Node per poi eseguirne il parsing e l'indicizzazione con ElasticSearch. Usa Curator per gestire gli indici e ha Wazuh installato come agent HIDS. Quando un analista esegue una query su Manager Node, questo esegue a sua volta query su Search Node e ne restituisce il risultato. L'aggiunta di Search Node può avvenire attraverso *cross cluster search*, dove ogni Search Node è indipendente, o attraverso *clustering* tradizionale in cui tutti i Search Node si uniscono logicamente per formare un singolo cluster.

### 3.2.3 Deploy e requisiti Hardware

In questa sezione verrà inizialmente descritto in che modo i vari nodi dell'infrastruttura utilizzano le risorse di calcolo, per poi passare all'elenco dei requisiti minimi di sistema [52] dei singoli nodi e l'effettivo dispiegamento delle risorse invece utilizzato sulle macchine virtuali. Le risorse sono state calcolate considerando 1 Gbps come massimo carico di rete. In conclusione seguirà una breve analisi del deploy dell'IDS all'interno dell'infrastruttura di testing.

#### Manager Node

Il Manager Node utilizza la CPU principalmente per ricevere i log in ingresso e posizionarli in coda su Redis, eseguire web server e aggregare i risultati ottenuti dai vari Search Node in seguito a query.

Le risorse della RAM vengono usate in particolare da esecuzione di Logstash e Redis. Le dimensioni della RAM impattano direttamente sulla dimensione della coda Redis.

La memoria secondaria invece per utilizzi generali di sistema operativo e per mantenere la Dashboard di Kibana.

| | CPU (N. core) | RAM (GB) | DISK (GB) |
|---|---|---|---|
| Minimi | 4-8 | 16 | 200-1000 |
| Utilizzati | 8 | 16 | 200 |

Tabella 3.1: Requisiti minimi e risorse effettivamente assegnate al Manager Node.

### Search Node

I Search Node usano la CPU principalmente per eseguire il parsing e l'indicizzazione dei log, ovviamente la richiesta di CPU aumenta all'aumentare del numero di log da elaborare.

La quantità di RAM a disposizione influenza direttamente la velocità di ricerca all'interno degli indici e l'affidabilità del sistema di ricerca. I programmi principalmente coinvolti nell'utilizzo della RAM sono Logstash ed Elasticsearch.

La memoria secondaria viene utilizzata per memorizzare metadati indicizzati. All'aumentare dello spazio a disposizione è possibile aumentare i tempi di conservazione dei dati.

| | CPU (core) | RAM (GB) | DISK (GB) |
|---|---|---|---|
| Minimi | 4-8 | 16-64 | >200 |
| Utilizzati | 8 | 16 | 200 |

Tabella 3.2: Requisiti minimi e risorse effettivamente assegnate al Search Node.

### Forward Node

La CPU di un Forward Node viene utilizzata per analizzare e memorizzare il traffico di rete. All'aumentare del traffico di rete aumenta il consumo del tempo di CPU necessario per l'analisi dei dati.

Si stima che un worker Suricata o Zeek possa elaborare 200 Mbps, ciò implica che per gestire 1 Gbps di traffico di rete siano necessari almeno 10 worker e quindi almeno 10 core. A questi vanno aggiunti i core necessari per eseguire altri processi fondamentali come Stenographer, etc.

La RAM impiegata dipende molto dai servizi attivi, dal tipo di traffico monitorato, dal traffico attuale e dall'ammontare di pacchetti scartati che si considera accettabile. Viene principalmente utilizzata per scrivere cache e processare il traffico di rete.

La memoria secondaria dipende da quanti dati si vogliono mantenere memorizzati per analisi future e dal periodo di retention[8] ritenuto opportuno. Se ad esempio si volesse mantenere il traffico di rete di un'intera giornata di monitoraggio, supponendo che il traffico di rete ricevuto abbia una media di 1 Gbps, allora sarebbero necessari circa 11 TB di spazio di archiviazione solo per mantenere i PCAP.

| | CPU (N. core) | RAM (GB) | DISK (GB) |
|---|---|---|---|
| Minimi | $\approx 12$ | 16-128 | variabile |
| Utilizzati | 13 | 64 | 500 |

Tabella 3.3: Requisiti minimi e risorse effettivamente assegnate al Forward Node.

### Deploy

Il primo passo di deploy dell'IDS all'interno dell'infrastruttura di testing è stato creare una VM all'interno del server ESXi e installarci sopra il Manager Node. Il Manager Node ha un'interfaccia di rete collegata alla Vlan con accesso a Internet.

In seguito è stato installato il Search Node all'interno di un'altra VM. Questa ha una sola interfaccia di rete collegata alla stessa Vlan del Manager Node. Il nodo di ricerca è stato quindi collegato al Manager.

Infine è stata creata un'ulteriore VM in cui è stato installato il Forward Node. Questa ha due interfacce di rete. Una collegata alla stessa rete del Manager Node, mentre l'altra è collegata alla botnet Vlan di cui si vuole monitorare il traffico di rete. In particolare quest'ultima deve essere in modalità promiscua[9]. In una situazione normale sarebbe necessario duplicare il traffico di rete attraverso TAP o SPAN e inviarne una copia all'interfaccia del Forward Node per poterlo analizzare, ma in questo caso particolare non è necessario. Infatti grazie al network virtualizzato del server ESXi basta impostare il virtual switch in modalità promiscua e tutto il traffico di rete sarà catturabile.

Una volta terminata l'installazione sono state aggiunte regole al Firewall integrato del Security Onion per poter accedere all'interfaccia web del Manager Node. Una volta impostato l'indirizzo di rete da monitorare è stato accertato che i servizi funzionassero correttamente. In particolare sono state attivate tutte le regole built-in di Suricata e sono stati effettuati dei semplici test di scansione con Nmap[10] [53] per verifica che le scansioni venissero effettivamente rilevate.

[8] specifica dopo quanto tempo i dati verranno cancellati.
[9] in modalità promiscua l'interfaccia di rete lascia passare anche il traffico non rivolto all'interfaccia stessa, verso la cpu.
[10] software utilizzato per eseguire scansioni di rete.

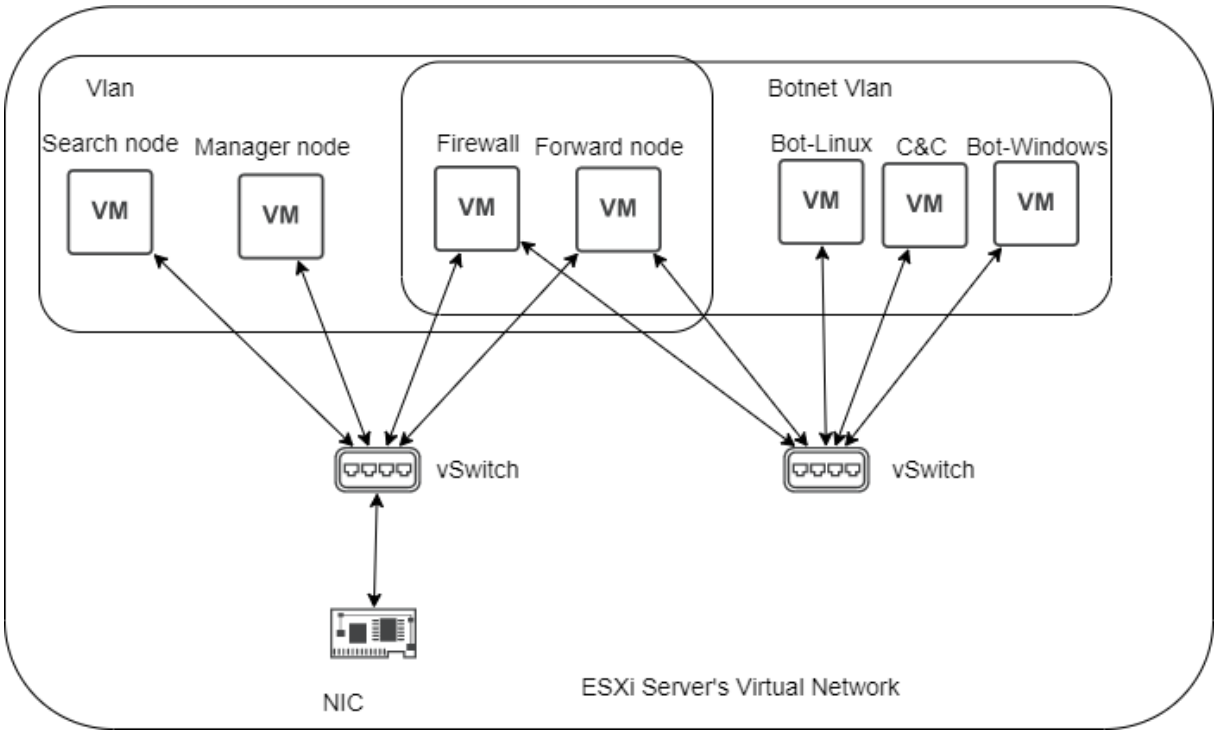



Figura 3.6: Deploy dei nodi di Security Onion all'interno del network virtuale dell'infrastruttura di testing.

### 3.2.4 SALT

Salt [54] è il componente principale di Security Onion incaricato del management di tutti gli altri nodi del sistema IDS.

Salt è un framework per l'automazione cui funzionalità principali sono:

- Capacità di gestire configurazioni di nodi da remoto;
- Esecuzione di comandi e prelievo di dati su nodi remoti.

Oltre a queste capacità offre comunque tantissime altre funzionalità come Job Management, Job Scheduling, File Server, orchestration, etc.

In particolare la funzionalità di Configuration Management offerta permette di centralizzare le configurazioni di tutti i nodi dell'infrastruttura, oltre ad astrarre l'infrastruttura a un insieme di dati.

Basato su architettura Master-Minions, dove master controlla uno o più minion. Comandi sono generati da master verso gruppo di minion, che eseguono task e ne ritornano i risultati/dati a master.

I messaggi tra Master e Minion avvengono attraverso *ZeroMQ message bus* [55]. La comunicazione tra Master e Minion è criptata, utilizzando crittografia a chiave pubblica per l'autenticazione del Minion e crittografia a chiave privata per scambio di dati.

Salt è basato su agent installati sui minion ma è anche possibile gestire i nodi attraverso SSH senza dover installare necessariamente un agent apposito (agentless system).

Il framework è scritto in Python ma il sistema di management delle configurazioni è di fatto language-agnostic, ovvero aperto a qualsiasi tipo di linguaggio. Di default usa YAML[11] [56] e Jinja[12] [57] template per le configurazioni.

Tra le feature offerte da Salt vi sono anche:

**Fault tollerance** Minion può connettersi a molteplici master contemporaneamente;

**Configuration Management scalabile** Il Design realizzato permette il management di migliaia di nodi;

**Parallel execution** Comandi eseguibili in parallelo;

**Python API** Interfaccia di programmazione semplice con design modulare ed estendibile.

### Remote execution

Come accennato Salt usa modello master-client in cui master inoltra comandi a client che provvede a eseguirli.

Entrando nel dettaglio, il Salt Master è un server che esegue un particolare processo chiamato *salt-master*. Esso inoltra comandi a server (Salt Minion) registrati presso il master che eseguono processo chiamato *salt-minion*.

Salt può anche essere visto come un modello *publisher-subscriber*. Il Master pubblica *job* che devono essere eseguiti e Minion si iscrivono a questi job.

Quando un Minion termina l'esecuzione di un job, invia dati in risposta al Master. Salt usa due porte in particolare su ogni nodo per le comunicazioni tra master e minion. Queste porte lavorano in contemporanea per ricevere e inviare dati verso il message bus. Il Message Bus usa ZeroMQ per creare una topologia di rete asincrona per la comunicazione più veloce possibile. In particolare usa due canali:

**Pub Channel** Canale usato da master per inviare job a minion. Basato su paradigma pub/sub;

**Req Channel** Canale usato da minion per inviare dati a master. Usato principalmente per eseguire *fetch* di file e inviare risultati di job.

La sintassi di un comando eseguibile da remoto verso uno o più minion è della seguente forma:

```
salt '<target>' <module>.<function> [arguments]
```

[11] linguaggio per serializzazione di dati in maniera comprensibile all'utente, utilizzato spesso per file di configurazione.
[12] template engine.

### Configuration Management

Il Configuration Management è eseguito dichiarando in quale stato dovrebbe essere un minion specifico, questo stato è chiamato Salt State. Questi altro non sono che file YAML, attraverso i quali è possibile eseguire il management dell'infrastruttura. Un esempio di semplice Salt State:

```
1       install_network_packages:
2           pkg.installed:
3               - pkgs:
4                   - rsync
5                   - lftp
6                   - curl
```

L'esempio mostra un Salt State usato per installare o mantenere l'installazione di alcuni pacchetti. I file Salt State hanno estensione *.sls*.

Per portare manualmente un minion specifico in questo stato si può usare il comando:

```
salt "<nome minion>" state.apply <Salt State>
```

Ovviamente non è pensabile eseguire il precedente comando manualmente per portare ogni minion in un determinato stato, soprattutto per ambienti in cui vi sono migliaia di minion con migliaia di State file. Per questo motivo viene usato il *Top File*.

Il Top File è uno State File usato per mappare i Salt State ai vari minion. In breve, esso descrive dove determinati Salt State devono essere applicati. Un esempio di Top File è il seguente:

```
1       base:
2         '*':
3            - common
4         'minion1':
5             - sample
```

Questo Top File associa il Salt State *common.sls* a ogni minion e *sample.sls* al minion *minion1* Attraverso il comando che segue è possibile applicare il Top File a tutti i minion:

```
salt * state.highstate
```

I Salt State possono essere parametrizzati attraverso l'uso di *pillar* e meccanismi di template. I pillar sono file in master che contengono dati che devono essere distribuiti a minion. Questi dati possono essere ad esempio dati sensibili, password, configurazioni, chiavi crittografiche, e molto altro. I pillar sono associati ai vari Minion con un apposito *Top File* dedicato, in modo analogo a quanto avviene con i Salt State. Un esempio di pillar contenenti id di utenti:

```
users:
  thatch: 1000
  shouse: 1001
  utahdave: 1002
  redbeard: 1003
```

I dati dei pillar possono essere referenziati all'interno dei Salt State usando dei template, come ad esempio il seguente Salt State:

```
{% for user, uid in pillar.get('users', {}).items() %}
{{user}}:
  user.present:
    - uid: {{uid}}
{% endfor %}
```

Questo State usa la sintassi di template Jinja per creare una logica più complessa, in particolare prende i dati contenuti all'interno del pillar precedentemente definito per assicurarsi che all'interno del minion, a cui verrà applicato lo State, siano presenti gli user id elencati nel pillar.

#### Salt in Security Onion

Come già detto Salt è usato all'interno di Security Onion per il management di tutti i nodi all'interno dell'infrastruttura.

Molte delle opzioni di Security Onion sono configurabili attraverso specifici pillar file.

Modificando i Pillar è possibile cambiare le configurazioni di tool. Tra i tool che è possibile modificare attraverso le impostazioni nei Pillar vi sono: Filebeat, Suricata, Zeek, gestione degli alert, etc.

Oltre al mantenere le configurazioni per i vari software, attraverso Salt è anche possibile eseguire comandi sui nodi e controllarne lo stato, permettendo controllo totale sull'infrastruttura Security Onion.

## 3.3 Ossim

Ossim [58] è un prodotto Security Information and Event Management (SIEM) open source sviluppato da AT&T Cybersecurity (in passato denominata AlienVault). Un sistema SIEM colleziona dati attingendo da diverse fonti di log all'interno dell'infrastruttura. Attraverso l'analisi dei dati è capace di generare allarmi in caso di eventi sospetti, oltre a poter generare report. Ossim viene integrato con Open Threat Exchange (OTX), una piattaforma in cui utenti possono scambiarsi informazioni su host maligni.

È in grado di supportare log di diversa natura e da molteplici sorgenti. È anche possibile inoltrare coppia di traffico di rete.

Ossim è costituito di tre elementi:

**Sensore** Primo device ad avere un contatto con i log. Raccoglie log e informazioni che riceve e lì converte in eventi per poi inviarli a Server;

**Server** Processa eventi ricevuti da sensore ed esegue operazioni come correlazione di eventi, pattern recognition e calcolo del rischio. Inoltre esegue anche controlli sfruttando informazioni di OTX;

**Logger** Nodo sul quale vengono memorizzati i log. Comprime, firma e memorizza informazioni in accordo alle retention policy.

Ossim è incentrato sul concetto di Asset. Un Asset è un qualsiasi elemento dotato di indirizzo IP. Ogni Asset aggiunto deve passare attraverso i seguenti moduli per poter avere il massimo livello di accuratezza di output da Ossim:

**Asset Discovery** Modulo che identifica Asset e porte note. Cerca inoltre di mantenere un inventario degli Asset identificati;

**Vulnerability Assessment** Identifica vulnerabilità in Asset attraverso scansioni. Le scansioni possono essere più approfondite attraverso autenticazione;

**Intrusion Detection** Costituito da NIDS, HIDS e File Integrity Monitor (FIM). Utilizza tool come Snort per identificare intrusioni in traffico di rete in generale e Suricata per intrusioni su traffico web based;

**Behavioral Monitoring** Modulo in cui vengono analizzate le anomalie di rete. Utilizza modulo Netflow per identificare cause di spikes di rete, propagazione di malware, etc. Possiede anche capacità di monitoraggio dell'availability di host e servizi;

**Security Intelligence** Modulo in cui processamento/computazione di eventi è eseguito. Tutte le info prese da moduli precedenti vengono analizzate al suo interno per creare report.

Ossim e l'intera infrastruttura è configurabile e gestibile una volta installato attraverso l'interfaccia web.

## Deploy e requisiti Hardware

Ossim è stato utilizzato come IDS da affiancare a Security Onion. È stato utilizzato principalmente per poter confrontare i risultati di Security Onion con un secondo IDS e per approfondire un sistema IDS alternativo.

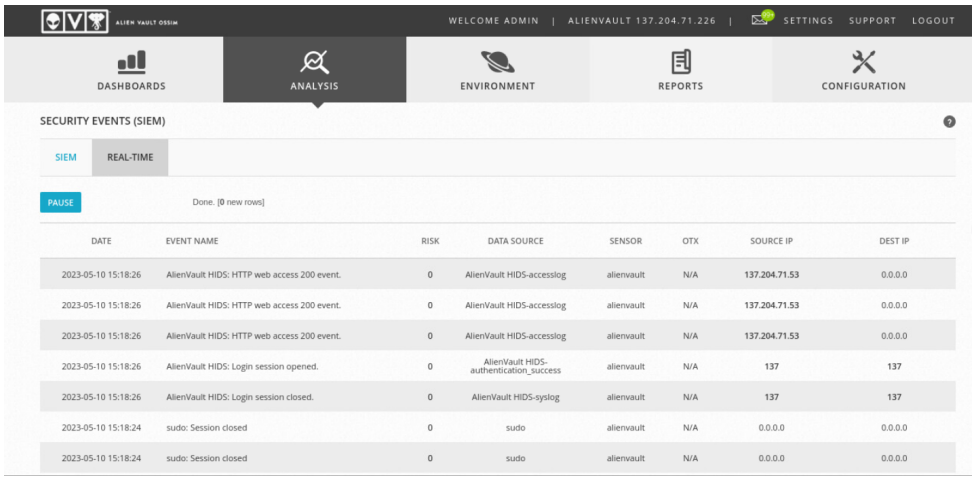



Figura 3.7: Interfaccia web di visualizzazione dei log.

Per semplicità ne è stato eseguito il deploy in formato stand-alone, ovvero con tutti e tre gli elementi che lo compongono in stesso host.

La macchina virtuale sul quale è stato installato OSSIM ha due interfacce di rete:

- NIC di management collegata alla Vlan con accesso a Internet per potersi interfacciare con la web Gui;
- NIC collegata alla botnet Vlan in modalità promiscua, necessaria per sniffare il traffico di rete.

I requisiti Hardware richiesti e soddisfati per gestire dai 1000 ai 2000 eventi al secondo sono i seguenti:

| | CPU (N. core) | RAM (GB) | DISK (GB) |
|---|---|---|---|
| Minimi | 8 | 16-24 | 500-1000 |
| Utilizzati | 8 | 24 | 500 |

Tabella 3.4: Requisiti minimi e risorse effettivamente assegnate ad Ossim.

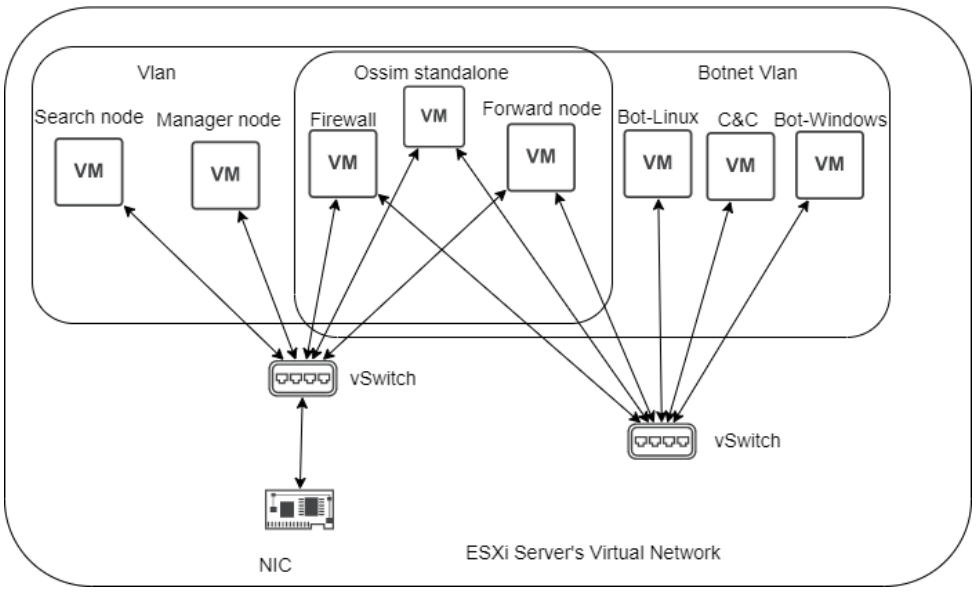



Figura 3.8: Deploy di Ossim in modalità standalone all'interno dell'infrastruttura di testing.

# 4 Studio Botnet 1 - Byob

Byob [59] è una botnet open source scritta in Python, creata appositamente per scopi educativi e di testing. Si compone di quattro categorie di moduli principali:

- Server Module;
- Client Module;
- Post-exploitation Modules;
- Core Modules.

I moduli Core racchiudono le funzionalità principali di Byob e sono utilizzate all'interno del client module e del server module.

Il modulo server viene utilizzato per inizializzare il C&C server. Utilizza Flask [60] come framework per le funzionalità di web server e dispone di un database Sqlite [61] per mantenere i dati e le sessioni dei bot anche a distanza di tempo e in presenza di disconnessioni. Offre controllo sui bot attraverso console collegata via *reverse-shell*[1] o attraverso interfaccia web dedicata. Basato su topologia centralizzata con paradigma push-based (il server invia i comandi ai bot).

Il modulo client è utilizzato per creare i payload che verranno utilizzato per generare i bot. In particolare il payload è un multi-stage installer che verrà analizzato meglio nel dettaglio. Il bot così generato sarà in grado di importare moduli necessari e arbitrari direttamente dal server e di caricarli in memoria, non lasciando così traccie su disco. Questo meccanismo permette al bot di non avere dipendenze pregresse se non avere Python installato. Tuttavia per ovviare anche a questa dipendenza è possibile creare eseguibili che includano all'interno, oltre al payload, un interprete Python stand-alone, così che neanche Python risulti più una dipendenza. Il payload può anche essere opzionalmente criptato e opzionalmente può abortire l'esecuzione se rileva che il sistema sul quale sta eseguendo è una macchina virtuale.

I moduli post-exploitation sono moduli built-in che possono essere importati ed eseguiti su bot e offrono funzionalità operative molto interessanti. Tra questi possiamo trovare:

**Persistence** Cerca di rendere il malware persistente all'interno del sistema che lo ospita;

[1] shell remota a cui viene reindirizzato input e output verso host che vuole accedervi.

**Packet Sniffer** Cattura traffico di rete;

**Escalate Privileges** Tenta di ottenere privilegi di amministratore;

**Port Scanner** Scansiona traffico di rete;

**Keylogger** Cattura digitazioni della tastiera;

**Screenshot** Cattura immagine del monitor;

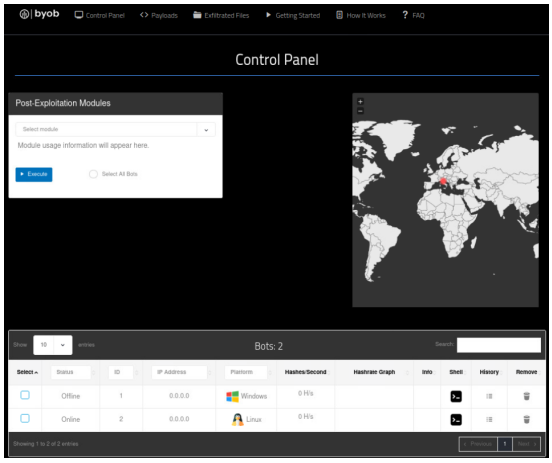
**Outlook** Utilizza Outlook client per leggere/scrivere email;

**Process Control** Per elencare/ricercare/terminare processi;

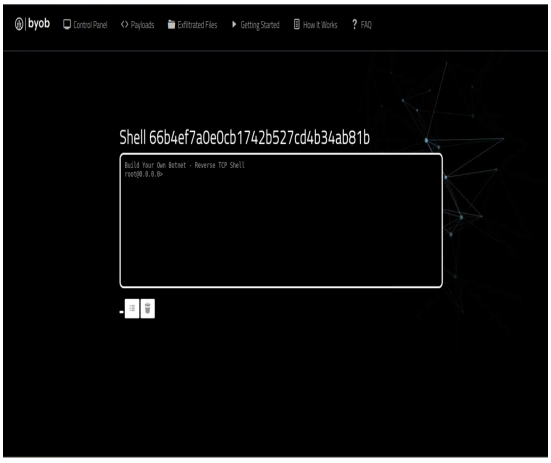
**iCloud** Cerca account autenticati a iCLoud su macOS;

**Miner** Utilizza un miner per minare criptovalute;

Oltre a questi moduli built-in, il bot può anche importare moduli creati ad-hoc dal botmaster.




(a)

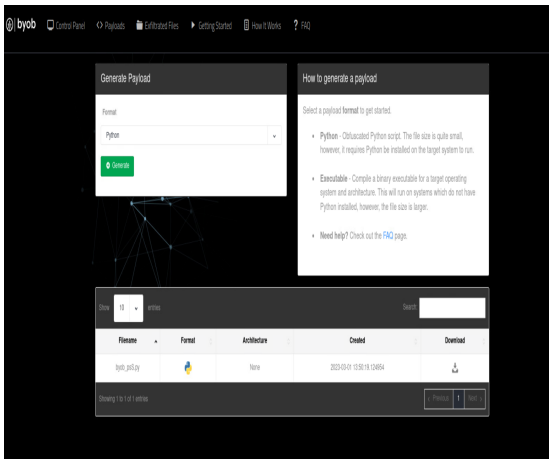



(b)

Figura 4.1: (a) Home page; (b) Pagina shell.

L'interfaccia web del C&C è dotata di sistema di autenticazione, per impedire che terzi possano prenderne il controllo facilmente. Quindi prima di poter interagire con le funzionalità che offre l'interfaccia web è necessario eseguire il login nell'apposita pagina web. Nella Figura 4.1a si può vedere la schermata di controllo principale dell'interfaccia web di Byob. In basso è presente una schermata con i bot attualmente collegati. Sul lato sinistro vi è un pannello con i moduli di post-exploitation utilizzabili sui bot del pannello sottostante, mentre a destra si può controllare la posizione geografica del bot.

Cliccando sull'icona del terminale in corrispondenza di un bot nel pannello inferiore, reindirizza il browser verso una schermata in cui viene emulato un terminale. Questo è




(a)


(b)

Figura 4.2: (a) Schermata di generazione dei payload; (b) Byob console.

collegato al bot attraverso reverse-shell e può essere usato per operare direttamente sul bot.

In Figura 4.2a si può osservare la schermata di creazione dei payload. I payload possono essere eseguibili compilati o script Python. Una volta creati possono essere scaricati da questa pagina web.

Byob può essere anche eseguito nella sua variante minimale da console (Figura 4.2b).

Attraverso il pannello di controllo o le console è possibile controllare i bot ed eseguire le classiche funzionalità RAT o eseguire moduli Python arbitrari, rendendo di fatto possibile eseguire qualsiasi operazione[2] sulle macchine vittime.

## 4.1 Dettagli implementativi

Nella seguente sezione verranno descritti più nel dettaglio i principali componenti su cui è basata la botnet.

### 4.1.1 Server

Lo script *Server.py* è il responsabile dell'esecuzione del server e di tutte le sue funzionalità. In Figura 4.3 si può vedere un diagramma di sequenza che ne descrive il comportamento.

Al momento dell'esecuzione crea tre sottoprocessi che eseguiranno in background:

**Packages handler** Mantiene un server HTTP su porta dedicata incaricato di fornire i Python Package richiesti da bot;

**Module handler** Mantiene un server HTTP su porta dedicata incaricato di fornire Moduli Python richiesti da bot;

[2]per alcune operazioni può essere necessaria l'autenticazione come amministratore.

**Request handler** Gestisce le richieste POST di upload inviate da bot, salvando i file in apposite directory.

Successivamente, esegue il C&C server su un'altra porta dedicata. Il C&C server crea un thread che si occupa di gestire le nuove connessioni inizializzate dai bot. Per ogni nuovo bot connesso crea una sessione, cui informazioni vengono salvate in database e aggiornate a ogni connessione.

Una volta creato il thread, si occupa di gestire i comandi inseriti su console di controllo dei bot e le richieste dell'interfaccia web. Come già osservato nell'introduzione di questo capitolo, Byob può eseguire anche senza interfaccia web e in tal caso la console non è inizialmente collegata a un bot. La console emulata dell'interfaccia web, invece, è già associata al bot specifico.

Quando Byob esegue in console, ha a disposizione dei comandi, di cui la versione con interfaccia web non ha bisogno, dato che sono integrati all'interno delle pagine web (eg. controllare le sessioni attive).

Durante il ciclo di vita del C&C, la console rimane in attesa di input. Quando viene inserito un comando, ne effettua il parsing per controllare che sia un comando disponibile e nel caso lo esegue con eventuali parametri annessi. Una volta collegata alla sessione del bot, questa rimane in attesa di comandi da inoltrare. La versione web usa uno script Javascript per inviare una richiesta POST contenenti i comandi digitati al backend che si occupa di inoltrarlo al bot. La versione console invece crea un thread che è incaricato di inviare i comandi direttamente al bot, e li inoltra solo in risposta a richieste di polling da parte del bot. Si può quindi affermare che la versione web sia push-based mentre l'altra pull-based.

Da questo punto in avanti le funzionalità di console e console web emulata sono identiche, la versione web infatti sfrutta le funzionalità riadattate di Byob versione console.

Il server invia comandi e riceve risultati sotto forma di dizionari contenenti altre informazioni come l'id del bot.

### 4.1.2 Payload

L'infezione attraverso payload è strutturata su più parti consequenziali.

Il file che la vittima esegue è il *Dropper*. Il Dropper scarica dal server lo Stager e lo esegue. Le righe di codice Python responsabili del download sono inizialmente serializzate, compresse e codificate in base64, per rendere più difficile l'identificazione di codice malevolo da parte strumenti di rilevazione.

Lo Stager, a sua volta, scarica il Payload dal server. Di default il server invia il payload in chiaro, ma è possibile criptare il contenuto del Payload, in questo caso lo Stager si preoccupa di decriptarlo prima di eseguirlo.

Il Payload è il cuore del malware.

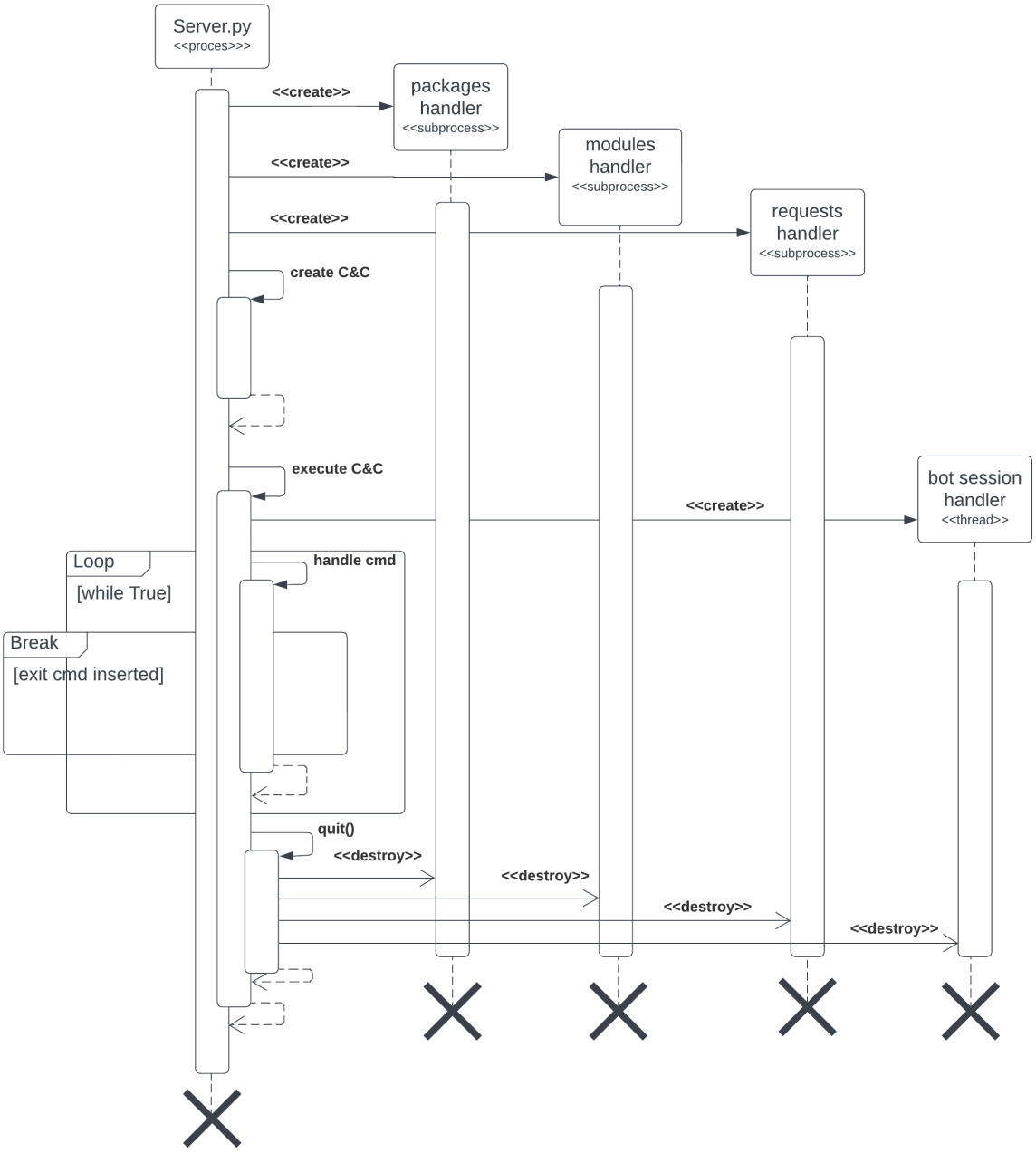



Figura 4.3: Diagramma di sequenza di Byob.

Una volta eseguito instaura una connessione col server e inizializza due thread, incaricati di ricercare e importare moduli e package da remoto, per poi entrare in un loop in attesa di task da eseguire.

Alla ricezione di un task, esegue il parsing e ne esegue comando (Figura 4.4), cui risultato viene incapsulato e inoltrato al server. Nel caso il server operi senza interfaccia web, viene eseguito un thread incaricato di eseguire polling verso il server, a cui il server occasionalmente risponderà con un task.

In caso il comando ricevuto necessiti un modulo o package, questo viene importato da remoto.

L'importazione di un modulo (o package) avviene aggiungendo un oggetto di classe Loader, contenenti i metodi *find_module* e *load_module*, alla lista di oggetti *sys.meta_path* e in seguito eseguendo l'import manuale del modulo. Normalmente, quando l'interprete Python cerca di importare un modulo, naviga la lista *sys.meta_path* ed esegue il metodo *find_module* di ciascun oggetto. Questo metodo restituisce un loader del modulo selezionato nel caso lo trovi. Un loader è un oggetto che implementa il metodo *load_module*. Una volta che il loader è stato trovato, ne viene eseguito il metodo *load_module* che importa il modulo all'interno del contesto.

Byob sfrutta questo meccanismo. Quando esegue l'import manuale di un modulo remoto, l'interprete si imbatte nel loader inserito da Byob all'interno della lista *sys.meta_path* ed esegue il suo metodo di ricerca. Questo cerca il modulo all'interno della lista di moduli disponibili precedentemente estratta dal server, e nel caso lo trovi ritorna un'istanza di se stesso. Dato che la classe loader di Byob è fornita anche del metodo *load_module*, può eseguirlo per importare il modulo da remoto.

Quando il bot è incaricato di eseguire l'upload di un file verso il server, invia una richiesta POST al sottoprocesso del server, incaricato di gestire le richieste POST, sulla porta dedicata.

## 4.2 Testing e debug

Per eseguire il testing, il server è stato containerizzato con Docker. Il build dell'immagine è stato eseguito attraverso il seguente Dockerfile:

Listing 4.1: Byob server Dockerfile.

```
FROM ubuntu:bionic

WORKDIR /app
COPY . .
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get install -y build-essential cmake && \
    apt-get install -y python3.6 python3-pip && \
```

```
WORKDIR /app/web-gui
RUN python3.6 -m pip install pip --upgrade && \
    python3.6 -m pip install -r requirements.txt && \
    rm -rf .gitignore .travis.yml README.md requirements.txt \
        startup.sh docker-pyinstaller*

EXPOSE 5000 1337 1338 1339
ENTRYPOINT python3.6 run.py --debug
```

Il Dockerfile specifica che al momento del build deve essere copiato il sorgente del server all'interno del container, che devono essere installate tutte le dipendenze necessarie e che devono essere esposte le porte 5000, 1337, 1338, 1339 del Container. Per eseguire il container contenente il server è stato utilizzato il seguente *bash* script:

```
containers=`docker ps -a | grep -A 1 byob | cut -b -12`;
for x in $containers; do
    docker stop $x && docker rm $x ;
done;
docker run --name byob-web -it  -p 5000:5000 -p 1337:1337 \
    -p 1338:1338 -p 1339:1339 -v ./byob:/app    byobserver ;
```

Lo script interrompe l'esecuzione di tutti i container Byob esistenti e li rimuove prima di eseguire un nuovo container. La porta 5000 dell'host viene mappata alla porta del C&C mentre le restanti sono mappate ai servizi di distribuzione dei moduli/package e gestione delle richieste POST. La directory contenente il sorgente è stata mappata a una directory dell'host in modo che fosse facile modificarne il codice all'occorrenza e per fare in modo che alla rimozione del container la directory fosse persistente sulla macchina host. Gli argomenti *-it* sono utilizzati per rendere possibile l'interazione con la shell di debug del container.

Purtroppo il software ha presentato diversi problemi e bug da risolvere nella sua versione *out of the box*.

La funzionalità di import dei package da remoto risulta rotta. In particolare la ricerca confonde i moduli con i package, non riuscendo quindi mai a trovare ciò che si desidera poi importare. Il problema è stato aggirato ma non risolto, evitando di perdere troppo tempo su questioni secondarie, importando manualmente da locale i package necessari.

I moduli di post-exploitation sono compatibili solo con alcuni degli OS e la maggior parte risultava non funzionante. La botnet è stata testata sia con bot Windows che con bot Linux. Tra i moduli testati e non funzionanti vi sono i moduli per eseguire screenshot, keylogging, cattura del traffico di rete e uso della Webcam.

Analizzando il modulo Keylogger si è potuto notare che il modulo risulta scritto in modo errato, forse perché utilizza una versione deprecata di API della libreria che sfruttava per eseguire il suo compito.

Tra i moduli che invece risultano funzionanti vi erano quello utilizzato per eseguire scansioni delle porte di rete e quello per la gestione dei processi.

Le funzionalità di RAT funzionano correttamente. In particolare è stata testata l'esecuzione di codice Python arbitrario attraverso riga di comando o attraverso l'import di moduli creati appositamente.

La creazione del Payload criptato non funziona correttamente.

Le funzionalità di geolocalizzazione, l'import di script di Google Analytics e font scaricati da Internet sono stati rimossi in quanto si appoggiano a servizi web esterni non raggiungibili dall'interno del perimetro di testing, provocando il crash dell'applicativo e ritardi nell'esecuzione.

A seguito di una fase di risoluzione dei problemi che ha richiesto un considerevole impiego di tempo, si è ritenuto opportuno procedere alla verifica con IDS delle sole funzionalità che al momento risultavano operative, per poi procedere con lo studio di un'ulteriore botnet.

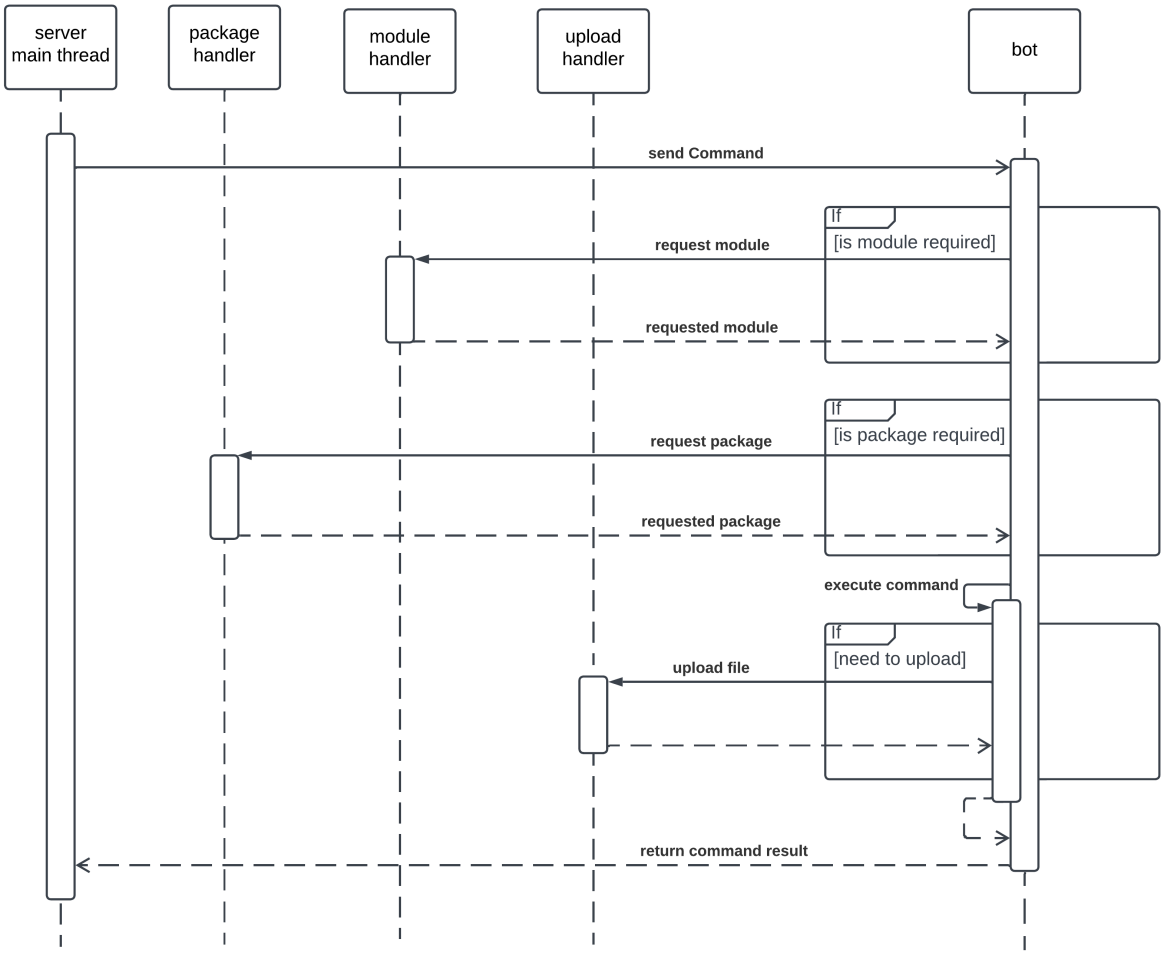
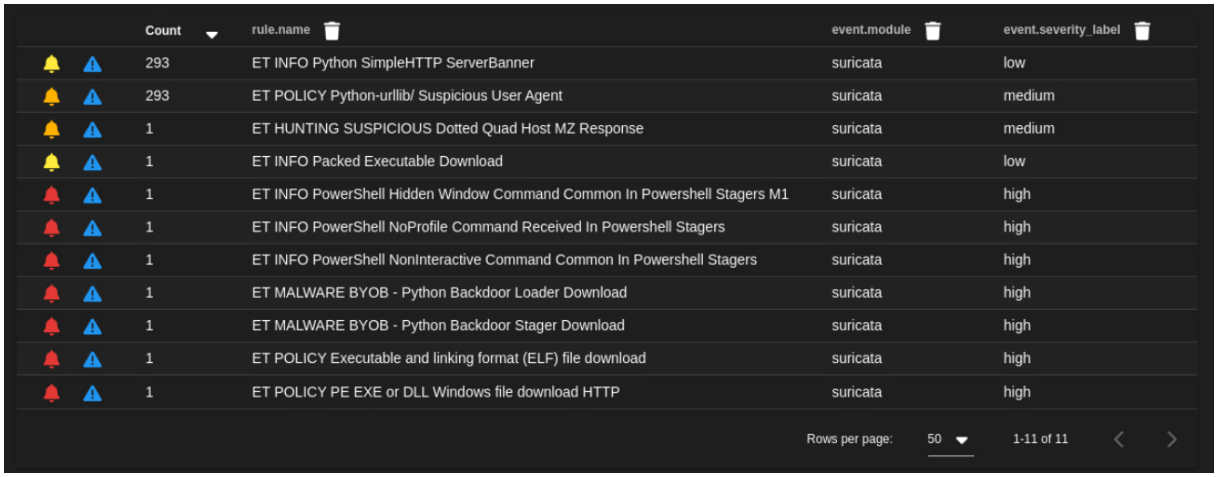



Figura 4.4: esecuzione di un comando ricevuto dal C&C.

## 4.3 Rilevazioni con IDS

In Figura 4.5 si possono vedere gli eventi generati da Security Onion attraverso il modulo Suricata al momento dell'esecuzione del Payload. Gli eventi sono autoesplicativi e fanno riferimento in particolare a:

- Download di eseguibile;
- Richiesta web verso indirizzo IP;
- Rilevazione di Python all'interno dello user-agent;
- Rilevazione di codice di classe Loader di Byob;
- Rilevazione di codice di Stager di Byob.



| Count | rule.name | event.module | event.severity_label |
|---|---|---|---|
| 293 | ET INFO Python SimpleHTTP ServerBanner | suricata | low |
| 293 | ET POLICY Python-urllib/ Suspicious User Agent | suricata | medium |
| 1 | ET HUNTING SUSPICIOUS Dotted Quad Host MZ Response | suricata | medium |
| 1 | ET INFO Packed Executable Download | suricata | low |
| 1 | ET INFO PowerShell Hidden Window Command Common In Powershell Stagers M1 | suricata | high |
| 1 | ET INFO PowerShell NoProfile Command Received In Powershell Stagers | suricata | high |
| 1 | ET INFO PowerShell NonInteractive Command Common In Powershell Stagers | suricata | high |
| 1 | ET MALWARE BYOB - Python Backdoor Loader Download | suricata | high |
| 1 | ET MALWARE BYOB - Python Backdoor Stager Download | suricata | high |
| 1 | ET POLICY Executable and linking format (ELF) file download | suricata | high |
| 1 | ET POLICY PE EXE or DLL Windows file download HTTP | suricata | high |

Rows per page: 50 1-11 of 11

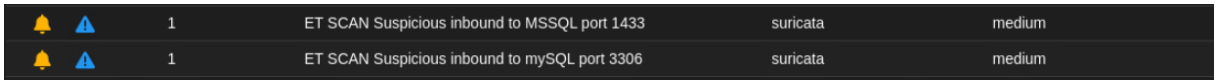
Figura 4.5: Rilevazione Security Onion dell'esecuzione del Payload.

Gli eventi più significativi fanno riferimento alla rilevazione di eseguibili e codici maligno noto. Se il download del payload fosse stato criptato nessuna di queste rilevazioni avrebbe avuto luogo tranne il riferimento all'indirizzo IP.

In Figura 4.6 si possono osservare le rilevazioni di Security Onion e Ossim in merito all'esecuzione del modulo di scansione di rete.

Le funzionalità di RAT attraverso reverse-shell e l'utilizzo di moduli Python generici costruiti ad-hoc non sono stati rilevati. Questo perché Suricata non ha fingerprint che facciano riferimento ai moduli da noi creati, mentre i comandi inoltrati attraverso reverse-shell sono criptati con AES-256 e non vi è modo di identificare quel traffico di rete come maligno sfruttando solo SIDS.

Seppure il modulo di mining non sia stato testato a pieno, in quanto richiedeva interazioni con la rete Internet, al momento dell'esecuzione il download dell'eseguibile di un noto malware dal C&C. Essendo un malware noto, di cui esistono fingerprint in database pubblici, Strelka è stato in grado di identificarlo come si può osservare da una sezione del log riportato in Figura 4.7.




(a) Security Onion.


(b) Ossim.

Figura 4.6: Rilevazioni della scansione di rete eseguita da bot attraverso modulo apposito.

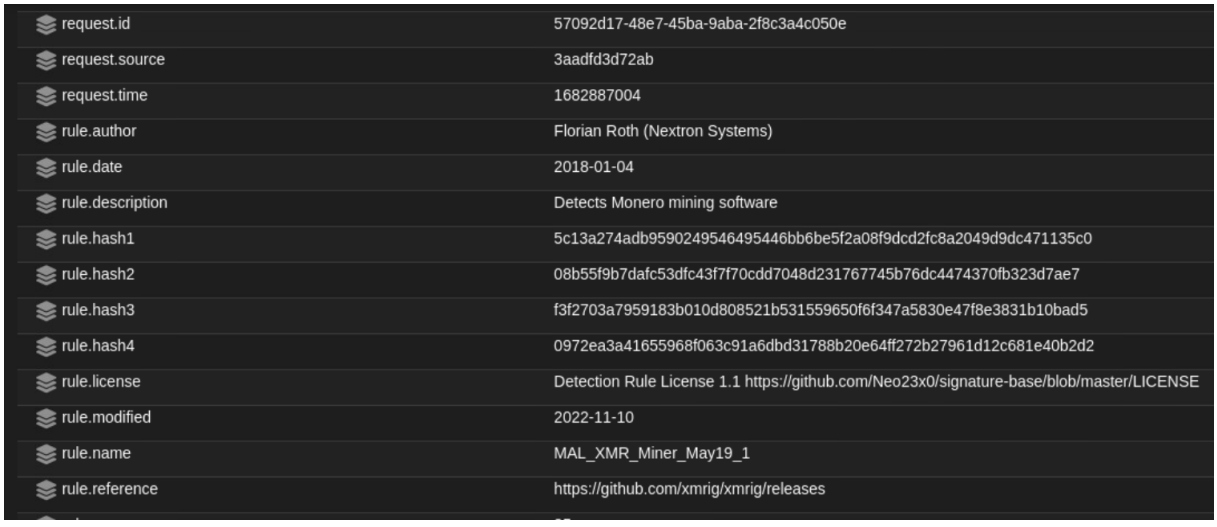



Figura 4.7: Rilevazione di import di malware noto.

# 5 Agent

Un agente o agent è un software definito a largo modo come un programma che può esercitare l'autorità di un individuo o di un'organizzazione, lavorare autonomamente per raggiungere un obiettivo e interagire con altri agenti e con l'ambiente circostante [62]. Esso si compone di codice e informazioni sullo stato del sistema per seguire computazioni. Inoltre, richiede una piattaforma sui cui eseguire.

Gli agent possono essere statici o mobili. Quelli statici rimangono fissi in una singola piattaforma mentre quelli mobili sono in grado di fermare la computazione, spostarsi su un'altra piattaforma e riprendere con l'esecuzione del proprio codice.

La piattaforma d'origine di un agent è chiamata *home platform* e normalmente è l'ambiente considerato più sicuro per l'agent. Una piattaforma può avere molteplici luoghi in cui più agent possono interagire.

La tecnologia degli agenti mobili ha tratto beneficio dal lavoro svolto sugli agenti intelligenti e sullo sviluppo di sistemi software capaci di supportare codice su hardware eterogeneo (eg. Java e Java Virtual Machine sono un esempio di sistema software capace di portare codice eseguibile su qualsiasi sistema operativo). Con agente intelligente si intende agenti statici autonomi capaci di applicare la conoscenza del dominio dell'applicazione. Gli agenti intelligenti incorporano la capacità di decomporre e risolvere problemi in modo collaborativo. Gli agenti osservano il loro ambiente, ragionano sulle proprie azioni e su quelle degli altri agenti, interagiscono con altri agenti ed eseguono le proprie azioni in modo concorrente con altri agenti. Attraverso le interazioni possono scambiarsi informazione tramite un linguaggio di comunicazione degli agenti e possono collaborare per creare una conoscenza comune su un argomento.

Anche se gli agenti mobili mantengono le caratteristiche di autonomia e collaborazione degli agenti intelligenti, l'accento è sulle caratteristiche di mobilità, spesso basandosi su algoritmi semplici e diretti per il ragionamento e la collaborazione attraverso una meno elaborata interpretazione dei messaggi.

La capacità di spostare una computazione tra diversi nodi offre vantaggi rispetto agli attuali sistemi di rilevamento delle intrusioni che si basano su computazioni staticamente posizionate. Per poter applicare la tecnologia degli agenti mobili alla rilevazione delle intrusioni, i nodi partecipanti (cioè gli host e i dispositivi di rete) devono avere installata una piattaforma per agenti. Poiché molti sistemi di agenti funzionano su una vasta gamma di hardware e software, questo requisito non è difficile da soddisfare. Con la tecnologia

degli agenti mobili, i nodi di raccolta, di aggregazione interna e di comando e controllo non devono essere continuamente residenti sulla stessa macchina fisica. Ad esempio, un agente mobile può funzionare come nodo di aggregazione e spostarsi in qualsiasi posizione fisica nella rete che sia migliore per i suoi scopi. Il paradigma degli agenti mobili offre inoltre la possibilità di specializzazione: gli agenti possono essere diversi per funzioni diverse, ognuno alla ricerca di attacchi distinti e di conseguenza elaborando i dati in modo specifico.

La tecnologia degli agenti mobili può potenzialmente superare alcune limitazioni intrinseche degli IDS che utilizzano solo componenti statici. Ciò non significa che le caratteristiche degli agenti mobili in sé siano sufficienti per ottenere miglioramenti. Quando si applicano agenti mobili a questo dominio di applicazione, sono comunque necessarie scelte di progettazione accurate per sfruttare le loro caratteristiche. In particolare, il tipo di coordinamento del livello di conoscenza richiesto per rilevare e rispondere alle intrusioni pone molte esigenze sugli agenti, tra cui localizzare altri agenti con le capacità necessarie, comunicare efficacemente con loro utilizzando un vocabolario comune e coordinare le azioni da intraprendere per affrontare congiuntamente una situazione data. In passato sono stati identificati diversi vantaggi nell'utilizzo degli agenti mobili rispetto ai loro omologhi statici [9], [63], [64]:

**Diminuire la latenza di rete** Grazie alla loro capacità di essere inviati per eseguire operazioni direttamente al punto di interesse remoto, gli agenti mobili sono in grado di rispondere in tempo reale ai cambiamenti dell'ambiente. Oltre a rilevare e diagnosticare potenziali intrusioni di rete, gli agenti mobili possono fornire meccanismi di risposta adeguati, tra cui la raccolta di informazioni sull'attacco inviate al bersaglio o emesse da esso, la chiusura o l'isolamento di un sistema sotto attacco per proteggerlo da ulteriori danni, il tracciamento del percorso dell'attacco e la chiusura o l'isolamento del sistema dell'attaccante se l'attacco viene lanciato da un host interno;

**Ridurre il carico di rete** Piuttosto che trasferire i dati attraverso la rete per eseguire computazioni su questi, gli agenti mobili possono essere inviati direttamente alla macchina in cui risiedono i dati, portando il la computazione vicino ai dati invece di dover spostare i dati verso la computazione. Ciò consente di ridurre il carico sulla rete;

**Esecuzione autonoma e distribuita** Per grandi sistemi distribuiti, è essenziale la capacità del sistema di continuare a funzionare quando parti di esso vengono distrutte o vengono isolate. Gli agenti mobili possono esistere e funzionare in modo indipendente dalla piattaforma di creazione, rendendoli utili come componenti di IDS, poiché gli agenti che sopravvivono a un attacco potrebbero essere in grado di ricostituire le componenti danneggiate (ad esempio, clonandole) e ripristinare la funzionalità;

**Adattamento dinamico** I sistemi di agenti mobili sono in grado di rilevare il proprio ambiente e adattarsi ai cambiamenti, rendendoli particolarmente utili nella rilevazione

delle intrusioni. Gli agenti possono muoversi in modo autonomo per migliorare la loro posizione o evitare situazioni pericolose, clonarsi per garantire la ridondanza e l'esecuzione parallela, o unirsi ad altri agenti per ottenere assistenza. Inoltre, possono adattarsi in modo flessibile a situazioni sfavorevoli o favorevoli. Grazie alla loro esecuzione autonoma e asincrona, queste caratteristiche consentono di costruire sistemi robusti e in grado di tollerare i guasti;

**Indipendenza dalla piattaforma** I sistemi di agenti forniscono un ambiente di calcolo astratto per gli agenti, indipendente dall'hardware e dal software del computer su cui viene eseguito. Queste caratteristiche rendono questo ambiente un'opzione adatta per le applicazioni di gestione di rete in generale e di rilevazione delle intrusioni in particolare, permettendo agli agenti di muoversi relativamente liberamente all'interno di un dominio. Questo è particolarmente vantaggioso per i meccanismi di risposta, poiché quando viene rilevata un'intrusione, i rimedi possono essere applicati o avviati da quasi qualsiasi punto della rete. Inoltre, i meccanismi di rilevamento beneficiano della mobilità diffusa degli agenti, che ha il potenziale di acquisire e unire facilmente dati da diverse fonti di rete;

**Software upgrade** Per aggiornare il software su più host, di solito l'amministratore deve fermare il server, disinstallare la vecchia versione del software e reinstallare la nuova versione. Ciò comporta la necessità di fermare l'intero sistema software. Tuttavia, se ogni componente del software è gestito da un agente, è possibile semplicemente disabilitare l'agente vecchio e distribuire un nuovo agente con le funzionalità richieste, senza interrompere l'intero sistema. Questo approccio consente di fermare solo il componente basato su agente che necessita di aggiornamento, evitando di fermare l'intero sistema.

Anche se il nostro interesse è associare gli agenti mobili alla rilevazione delle intrusioni, è improbabile che la piena mobilità di tutti i componenti di un IDS sia efficace in pratica, a causa dell'eccessivo overhead associato. Pertanto, alcuni componenti dell'IDS vengono designati come agenti statici o rimangono statici una volta implementati. Ciò consente l'applicazione del paradigma dell'agente mobile, ma si basa sulla mobilità solo dove appropriato.

Gli agenti mobili non migliorano direttamente le tecniche di rilevamento, ma possono modellare l'applicazione di tali tecniche migliorandone l'efficienza e l'efficacia. Un'area di potenziale utilizzo è la riduzione dell'enorme quantità di dati di log distribuiti tra i nodi interni di un sistema IDS convenzionale. I mobile agent possono visitare i repository dei dati e analizzare i risultati, trasferendo il calcolo ai dati stessi. Questo approccio riduce il carico di rete e consente agli agenti specializzati di concentrarsi su classi specifiche di intrusioni.

Un altro utilizzo dei mobile agent è la minimizzazione della capacità di un attaccante di ingannare un IDS. Gli agenti possono replicarsi e risiedere su più piattaforme, eliminando tali tentativi. Passare da IDS basati su rete ad agenti di rilevamento basati su host che funzionano contemporaneamente riduce anche la possibilità di pacchetti persi e massimizza la possibilità di innescare una risposta rapida a una intrusione rilevata. Inoltre, avere componenti residenti sull'host fornisce l'unico modo per l'IDS di visualizzare i pacchetti in testo normale, nelle situazioni in cui l'host utilizza la crittografia a livello di rete.

I mobile agent possono agevolare l'implementazione di architetture IDS robuste e resistenti agli attacchi [65]. Gli agenti si spostano quando rilevano pericolo o attività sospette, si clonano per garantire la ridondanza o la sostituzione, operano autonomamente e in modo asincrono rispetto al luogo di creazione, collaborano e condividono conoscenze, e si auto-organizzano per ovviare ad anomalie. Inoltre, gli agenti si prestano alla diversità genetica, che aiuta a evitare attacchi mirati a eludere i meccanismi di rilevamento noti e stabili dell'IDS.

Il maggior potenziale dei mobile agent risiede nella risposta a un'incursione piuttosto che nel suo rilevamento. Poiché le risposte possono essere avviate da quasi ovunque nella rete, i mobile agent possono gestire gli attacchi in modo più ottimale rispetto a un IDS convenzionale.

Alcuni dei vantaggi nell'implementare un mobile agent per rispondere a intrusioni possono essere:

**Tracciatura di un attaccante** Gli attaccanti spesso si insinuano all'interno di una catena di molti host prima di attaccare l'obiettivo e generalmente falsificano l'indirizzo di origine. Per trovare l'attaccante l'IDS deve ispezionare la catena di host per trovare il responsabile dell'inoltro dei pacchetti considerati come maligni. Per eseguire una tale traccia l'IDS deve essere in grado di analizzare il traffico di rete di ogni segmento di rete e ogni host. In genere tale compito sarebbe troppo dispendioso ma è realizzabile con una base di agent abbastanza ampia;

**Risposta in host target** Quando un attacco viene rilevato è importante poter rispondere alla minaccia all'interno dell'host colpito. Ciò è effettuabile attraverso gli agent;

**Risposta in sorgente di attacco** Per limitare le azioni dell'attaccante, l'IDS può agire direttamente sul suo host, ma ciò richiede l'uso di agenti mobili, altrimenti l'accesso sarebbe limitato. Nonostante questa opzione abbia delle restrizioni, come la necessità che l'attacco provenga dall'interno del dominio di gestione, può comunque essere un'arma molto efficace per l'IDS;

**Raccolta di evidenze** Al momento, è difficile raccogliere automaticamente prove di attacchi provenienti da molteplici fonti. Ciò è dovuto alla necessità di avere il software adatto eseguito nel luogo e nel momento giusto. Tuttavia, gli agenti mobili possono

essere utilizzati per eseguire qualsiasi cosa in qualsiasi momento e ovunque, rendendo possibile la raccolta di prove da diverse piattaforme hardware, sistemi operativi e persino applicazioni come i server web;

**Isolamento di target e attaccante** Poiché le azioni per rispondere automaticamente al bersaglio e alla fonte possono fallire, è necessaria una risposta a livello di rete per limitare le azioni di un attaccante. Ci sono tre strategie generali: bloccare le comunicazioni del bersaglio, bloccare le comunicazioni dell'attaccante e bloccare le comunicazioni tra il bersaglio e l'attaccante. Grazie alla loro capacità di viaggiare in tutti gli elementi della rete per eseguire azioni correttive, gli agenti mobili sono in grado di attuare tali strategie.

Il principale ostacolo nell'utilizzo di agenti mobili per la rilevazione delle intrusioni è la sicurezza. L'uso di agenti mobili può introdurre vulnerabilità che possono essere sfruttate da un attaccante per propagare un attacco o sovvertire la rilevazione dell'IDS. Le minacce alla sicurezza per il paradigma di elaborazione degli agenti mobili possono essere classificate in quattro categorie:

**Da agente ad agente** Si verificano quando gli agenti sfruttano le debolezze di sicurezza o lanciano attacchi contro altri agenti che risiedono in stessa piattaforma;

**Da agente a piattaforma** Si verificano quando gli agenti sfruttano le debolezze di sicurezza o lanciano attacchi contro una piattaforma di agenti in cui risiedono;

**Da piattaforma ad agente** Si verificano quando le piattaforme di agenti compromettono la sicurezza degli agenti che risiedono lì;

**Da altri a piattaforma** si verificano quando entità esterne, inclusi agenti e piattaforme di agenti situati altrove sulla rete, minacciano la sicurezza di una piattaforma di agenti, inclusi il suo sistema operativo sottostante e il servizio di comunicazione di rete.

In linea di principio, queste minacce non sono diverse da quelle che affrontano altre applicazioni distribuite e possono essere gestite con tecniche di sicurezza convenzionali, come l'isolamento degli agenti dagli altri agenti e dalla piattaforma di agenti, la sicurezza del linguaggio e del codice, l'accesso controllato alle risorse, l'Audit da parte degli agenti e delle piattaforme e le comunicazioni autenticate e protette. Tuttavia, la categoria da piattaforma ad agente è particolarmente difficile da difendere in applicazioni che richiedono un movimento illimitato degli agenti su Internet.

Tuttavia, la rilevazione di intrusioni ha caratteristiche uniche rispetto alle tipiche applicazioni Internet. Coinvolge un dominio chiuso e un ristretto gruppo di utenti noti, il che consente di implementare difese adeguate. In generale, le applicazioni di gestione di rete presentano le stesse caratteristiche, il che rende possibile l'adozione di contromisure efficaci

Dato che l'obiettivo è garantire che l'utilizzo di agenti mobili nella rilevazione di intrusioni offra una protezione almeno pari, se non superiore, a quella fornita da un IDS convenzionale, il principio alla base di questo approccio risulta essere consentire agli agenti di muoversi liberamente tra host affidabili all'interno del loro dominio di sicurezza, facendo in modo che ogni piattaforma imponga le stesse politiche di sicurezza, concedendo o negando gli stessi privilegi che aveva sulla piattaforma precedente [62].

Per fare in modo che le componenti critiche del sistema di agenti mobili rimangano al sicuro, questi possono spostarsi verso le componenti più robuste dell'IDS, sfruttando quindi l'affidabilità del sistema di IDS.

La firma digitale è un meccanismo chiave di protezione. Tutti gli agenti di gestione devono avere un codice firmato da un amministratore e superare la convalida prima di essere eseguiti sulla piattaforma dell'agente. In questo modo, si impedisce a un attaccante di modificare il codice di un agente o di creare un agente falso. Combinando la firma digitale con la capacità delle piattaforme degli agenti di autenticarsi reciprocamente prima di consentire lo spostamento di un agente, diventa molto difficile attaccare direttamente il paradigma degli agenti mobili.

Per aggirare il sistema degli agenti, un attaccante deve essere in grado di penetrare con successo nel sistema host che supporta la piattaforma degli agenti. Un attacco andato a buon fine su un sistema basato su agenti può compromettere gli agenti che stazioneranno sulla piattaforma dell'host infetto. L'attaccante quindi in seguito alla riuscita dell'attacco potrebbe terminare agenti, falsificare informazioni in loro possesso o modificare le loro informazioni di stato.

Il problema nel terminare un agent è che l'IDS potrebbe notare la sua assenza e quindi aumentare il livello di allerta e inviare agenti in risposta sul nodo ritenuto infetto. La falsificazione delle informazioni risulta quindi una miglior scelta, seppur potenzialmente complicata da implementare (potrebbe risultare difficile modificare le informazioni senza alterare il corretto funzionamento dell'agente o venire identificati dall'IDS).

Anche nel caso l'attaccante riesca nell'intento di prendere controllo di un agent, spesso molti sistemi offrono un modo per limitare le capacità di un agent attraverso la gestione dei privilegi. Sebbene non tutti gli agenti che eseguono attività di gestione necessitino di un alto livello di autorizzazione, alcuni agenti potrebbero dover essere eseguiti con privilegi amministrativi.

## 5.1 Wazuh

Wazuh [48] è una piattaforma open-source utilizzata per rilevamento, prevenzione e risposta di minacce. È in grado di proteggere i carichi di lavoro su ambienti on-premise, virtualizzati, containerizzati e basati su cloud.

Il sistema è composto da un server di management e da uno o più agent installati sugli *end-point* da monitorare. Il server colleziona e analizza i dati recuperati dagli agent.

### 5.1.1 Architettura e funzionamento

Wazuh è composto dei seguenti componenti:

**Indexer** Mantiene memorizzati e indicizzati gli *alert* generati;

**Server** Analizza i dati ricevuti dagli agent. Un singolo server può analizzare dati provenienti da decine di migliaia di agenti e scalare orizzontalmente all'occorrenza attraverso *clustering*;

**Dashboard** Interfaccia web utilizzata per visualizzare e analizzare dati. Può essere utilizzata anche per il management di configurazioni;

**Agent** Installati sui server da monitorare, sono statici ed eseguono operazioni o inviano dati al server. L'agente è composto da un demone e da più moduli operativi. Il processo demone si occupa della crittografia e dell'autenticazione necessarie per la comunicazione con il server, nonché dell'applicazione delle configurazioni remote e dell'esecuzione dei moduli. I moduli operativi svolgono le attività assegnate all'agente per il monitoraggio e la protezione della macchina su cui è installato.

L'agent invia continuamente eventi al server per l'analisi e la rilevazione di minacce. Per inoltrare i dati, l'agent stabilisce una connessione col servizio che gestisce le connessioni del server, che è in ascolto sulla porta 1514 di default. Il server, alla ricezione degli eventi, ne esegue il *decoding*. Con decoding si intende estrarre informazioni in formato chiave-valore attraverso l'uso di *decoder*. In seguito compara le informazioni così estratte dagli eventi con le regole interne. Le regole interne vengono utilizzate per generare alert in caso di match. Sia gli eventi che gli alert vengono memorizzati all'interno del server.

La comunicazione tra agente e server è criptata attraverso AES.

Il server a sua volta invia gli eventi e gli alert all'indexer attraverso Filebeat e con comunicazione criptata attraverso TLS. I dati vengono quindi indicizzati e la dashboard può essere utilizzata per reperire e visualizzare le informazioni indicizzate.

Wazuh è anche fornito di un API RESTful utilizzabile per visualizzare e modificare configurazioni e status di server e agent.

Alcune delle capacità principali di Wazuh sono:

**Intrusion detection** Gli agent effettuano una scansione dei sistemi monitorati alla ricerca di malware, rootkit[1] e anomalie sospette, possono individuare file nascosti, processi nascosti e ascoltatori di rete non legittimi, così come incongruenze nelle risposte alle

[1] tipologia di malware capace di nascondere la sua presenza agli strumenti di sistema.

chiamate di sistema. Oltre alle capacità degli agenti, il componente server utilizza un metodo di rilevamento delle intrusioni signature-based per identificare fattori di compromissione all'interno dei log analizzati;

**Analisi dei log** Gli agent sono in grado di leggere i log di applicazioni e del sistema monitorato, per poi inoltrarli in modo sicuro verso il server incaricato di analizzarli attraverso meccanismi basati su regole. Il server può ricevere log anche da dispositivi che non sono in grado di ospitare un agente, attraverso l'inoltro via Syslog. Attraverso le regole è possibile acquisire consapevolezza di errori di sistema, misconfigurazioni, attività malevole, violazioni di policy e altre problematiche a livello di sicurezza e operatività;

**File integrity monitoring** Wazuh è in grado di monitorare il file system, individuando modifiche nel contenuto, nelle autorizzazioni, nella proprietà e negli attributi dei file che richiedono un'attenzione particolare. Inoltre, è in grado di identificare automaticamente gli utenti e le applicazioni che creano o modificano i file. Questa funzionalità può essere combinata con l'analisi delle minacce per identificare eventuali attacchi o host compromessi;

**Vulnerability detection** Gli agenti possono estrapolare l'inventario dei software installati sulla macchina monitorata ed inviarlo al server, dove viene incrociato con i database di Common Vulnerabilities and Exposure[2] (CVE) costantemente aggiornati per identificare il software noto come vulnerabile;

**Configuration assessment** Wazuh è in grado di monitorare configurazioni di sistema o di applicazioni per assicurarsi che siano in linea con le linee guida e policy definite;

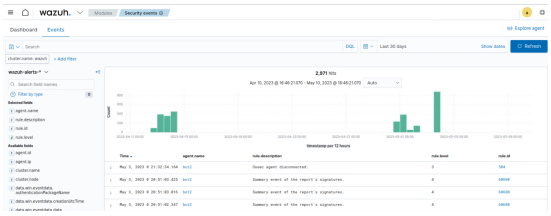
**Incident response** Permette di configurare risposte attive in caso determinati criteri sono rilevati, come bloccare l'accesso di una sorgente considerata malevola. Inoltre permette di eseguire comandi e query di sistema per eseguire attività forense o rispondere a minacce;

**Cloud security** Wazuh aiuta a monitorare le infrastrutture cloud a livello di API, utilizzando moduli di integrazione in grado di estrarre dati di sicurezza dai provider cloud più noti;

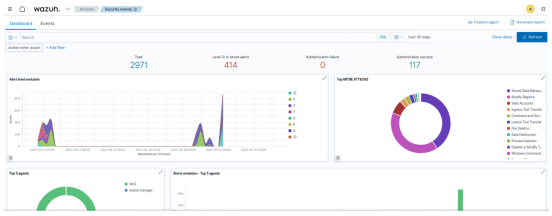
**Containers security** Wazuh offre visibilità sulla sicurezza dei container Docker, monitorando il loro comportamento e rilevando minacce, vulnerabilità ed anomalie. L'agente ha una integrazione nativa con l'engine Docker, consentendo di monitorare immagini, volumi, impostazioni di rete e container in esecuzione.

[2]identificatore per una particolare vulnerabilità o falla di sicurezza.

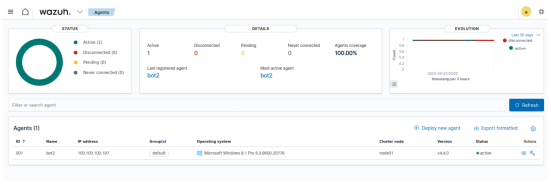
Le precedenti funzionalità sono configurabili attraverso file di configurazione XML globali, mantenuti in server centrale, e locali, conservati all'interno degli agenti. Le configurazioni degli agenti possono essere centralizzate e distribuite dal server. La generazione degli allarmi possono essere personalizzate aggiungendo regole e decoder al set predefinito.



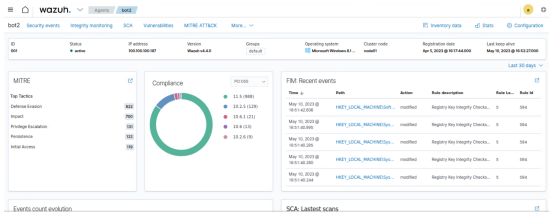
(a) visualizzazione eventi.



(b) Dashboard.



(c) schermata di gestione degli agent.



(d) overview di un agent specifico.

Figura 5.1: Alcune delle visualizzazioni offerte dall'interfaccia web.

### 5.1.2 Deploy e configurazione

L'obiettivo era l'installazione di un agent all'interno di una delle macchine virtuali destinate a diventare bot, al fine di poter studiare le tecnologie HIDS. Dato che la prossima botnet oggetto di studio è progettata per operare solo su macchine Windows, l'agent verrà installato sulla macchina virtuale con tale sistema operativo.

Sia Security Onion che Ossim, offrono la possibilità di ricevere log inviati da agenti Wazuh e mettono a disposizione dell'utente la possibilità di scaricare l'agent dal web server.

Inizialmente è stato eseguito il deploy dell'agent Wazuh sfruttando Security Onion come server. Per fare ciò, in seguito all'aggiunta di regole firewall e alla registrazione dell'host come agent, è stato scaricato il pacchetto di installazione dalla web gui di Security Onion. In seguito all'installazione dell'agent è stato notato che la versione del software non era aggiornata. Per evitare complicazioni in fase di testing e per poter sfruttare a pieno le capacità della versione più aggiornata è stato deciso di procede con l'installazione dell'agent fornito da Ossim.

Ossim offre la possibilità di installare automaticamente l'agent all'interno di Asset con sistema operativo Windows. Tuttavia tale approccio non è andato a buon fine e quindi si è proseguito con l'installazione manuale. Una volta registrato l'Asset come possessore di agent, tramite l'interfaccia web, è stato possibile eseguire il deploy dell'agent attraverso un semplice comando da terminale, fornito al momento della registrazione. Il comando in

questione era incaricato di scaricare, installare e configurare l'agent in modo che potesse comunicare col server.

Seppure la versione di Ossim fosse più aggiornata di quella di Security Onion, non era comunque l'ultima versione disponibile del software. Inoltre, entrambi gli IDS offrivano possibilità di visualizzazione e debug limitate in confronto al server Wazuh originale. Per i motivi elencati quindi, è stato deciso di proseguire con il deploy di un'ulteriore macchina che ospitasse il server Wazuh e che offrisse le versioni di agenti più aggiornate.

Come già osservato nella sezione inerente l'architettura di Wazuh, esso è composto di più componenti: Indexer, Server, Dashboard. Dato che il deploy di Wazuh servirà solo a scopi educativi si è optato per un'architettura stand-alone, in cui tutte le componenti coesistono in stessa macchina. Per facilitare l'installazione della piattaforma HIDS, è stato utilizzato ancora una volta Docker per la containerizzazione dei servizi. In particolare è stato sfruttato il file di configurazione Docker Compose offerto da Wazuh per l'esecuzione congiunta e coordinata dei servizi necessari.

| | CPU (N. core) | RAM (GB) | DISK (GB) |
|---|---|---|---|
| Minimi | 4 | 8 | 50 |
| Utilizzati | 8 | 16 | 50 |

Tabella 5.1: Requisiti minimi e risorse effettivamente assegnate al Wazuh server.

Una volta che il server è diventato operativo, si è proseguito con l'installazione dell'agent, sfruttando il relativo pacchetto d'installazione scaricabile dalla pagina web ufficiale di Wazuh.

In seguito all'installazione dell'agente, è stato installato Sysmon cui log verranno inviati attraverso l'agent verso il server per permettere analisi più approfondite. Sysmon [66] (System Monitor) è un tool appartenente alla suite Sysinternals creata da Microsoft. Esso permette di monitorare e notificare al Windows Event Log System[3] attività di sistema come la creazione di processi, instaurazione di connessioni di rete, modifiche a file e molto altro. Questo è stato inizializzato con un file di configurazione [67] che permettesse una buona base di tracciatura di eventi, in particolare legati al fattore sicurezza del sistema. L'agente Wazuh è stato istruito di collezionare i log di Sysmon che venivano inseriti all'interno del registro eventi di Windows ed in seguito di inviarli al server per poter permettere un analisi più approfondita del sistema. Nella sezione di codice 5.1, è possibile osservare l'aggiunta al file di configurazione Wazuh che permette all'agent di tenere traccia delle attività di Sysmon.

```
<localfile>
    <location>Microsoft_Windows_Sysmon/Operational</location>
    <lof_format>eventchannel</lof_format>
</localfile>
```

[3]sistema di memorizzazione e visualizzazione di log all'interno del sistema operativo Windows.

Listing 5.1: Aggiunta al file di configurazione dell'agent.

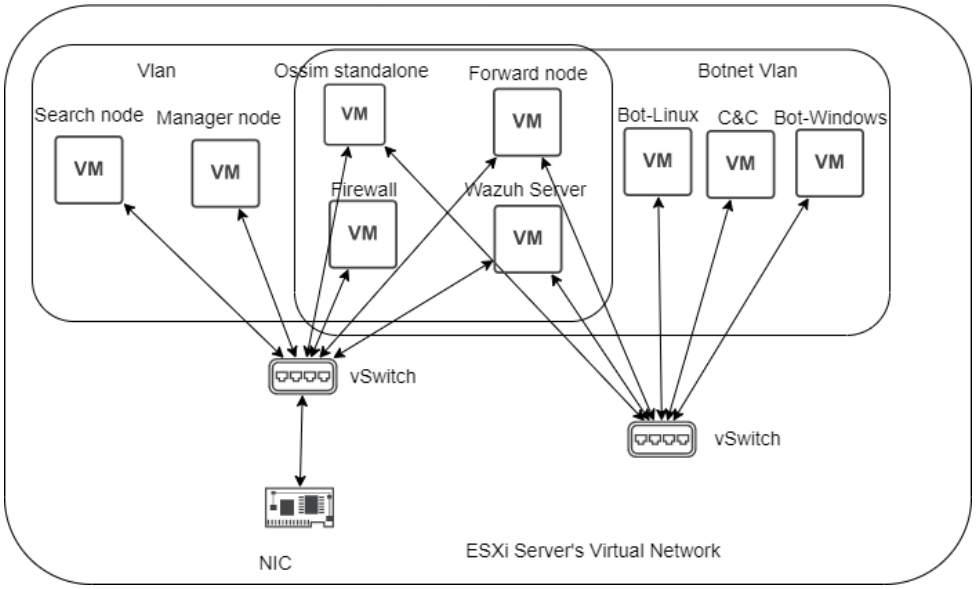



Figura 5.2: deploy del server Wazuh all'interno dell'infrastruttura.

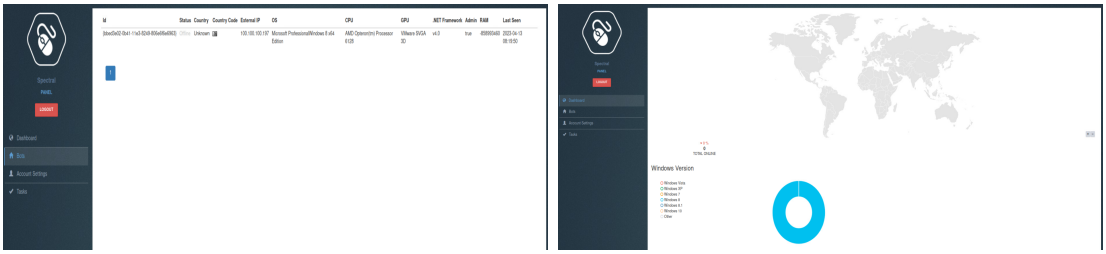
# 6 Studio Botnet 2 - UBoat

UBoat [68] rappresenta un progetto open-source di botnet basato su HTTP, volto a fornire una dimostrazione concreta (*proof of concept*) di una classica botnet centralizzata, con finalità prettamente educative. È scritta in linguaggio C++ senza alcuna dipendenza esterna e permette ridondanza di comunicazione col C&C attraverso un server di *fall-back*.

Tra le funzionalità operazionali built-in di questa botnet vi sono:

- Persistenza all'interno delle macchine infette per prevenire la perdita del controllo su di esse;
- Possibilità di DDoS basato su traffico TCP;
- Esecuzione di comandi e processi da remoto;
- Update della botnet;
- Download ed esecuzione di altri malware;
- Capacità di keylogging;
- Estrapolazione di screenshot da macchina infetta;

Il server di comando mette a disposizione, previa autenticazione, un'interfaccia web che consente di gestire e visualizzare i bot connessi ad esso.



(a) (b)

Figura 6.1: Visualizzazione delle informazioni dei bot.

In Figura 6.1a si vede l'interfaccia di visualizzazione dei bot che hanno avuto interazioni col server mentre in Figura 6.1b è mostrata la pagine web attraverso cui si può controllare la posizione geografica dei bot.

I bot possono essere selezionati per poter eseguire operazioni su di essi, la Figura 6.2 mostra l'interfaccia di comando dei bot attraverso la quale è possibile inoltrare task operazionali.

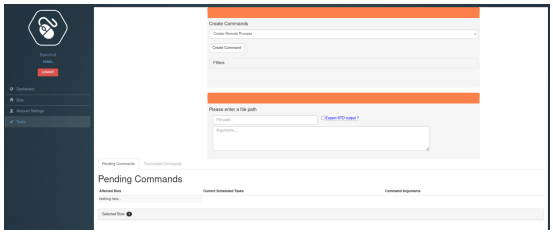



Figura 6.2: Interfaccia di controllo dei bot.

## 6.1 Dettagli implementativi

### 6.1.1 Server

Il C&C è composto di un web server, cui tecnologia è a discrezione dell'utente, e di MariaDB, un Database Management System (DBMS) relazionale. Mantiene le informazioni dei bot, i comandi inseriti ed altre informazioni all'interno del database. Il web server e la relativa interfaccia web sono raggiungibili attraverso la porta 8080 dell'host che ospita il web server.

Una volta che il server è operativo, aspetta che un bot si metta in contatto con esso. Il bot comunica con il server attraverso comandi di *Join* o di *Poll* attraverso richieste HTTP. Il primo viene utilizzato per iniziare la comunicazione col server. Alla ricezione del comando di Join quindi il server controlla che il bot sia presente all'interno del database e nel caso non lo fosse viene aggiunto. Il comando di Poll invece, viene utilizzato per richiedere comandi al server. Questa peculiarità definisce la botnet come pull-based. Il server riceve anche comandi contenenti l'esito dei task inoltrati. All'invio di un comando e alla ricezione di un risultato, il server aggiunge tale informazioni al database.

Il server inoltre, mantiene un ulteriore processo in background che esegue codice PHP. Questo si occupa di gestire l'upload degli screenshot, inviati dai bot attraverso richieste HTTP POST.

### 6.1.2 Payload

Il payload usa librerie di Windows per svolgere i propri task e quindi è eseguibile solo su tale sistema operativo.

All'avvio dell'esecuzione del payload, il Bot crea un thread che ha lo scopo di creare persistenza all'interno della macchina vittima. Per prima cosa crea una directory all'interno del sistema e vi posiziona all'interno una copia dell'eseguibile. Successivamente inserisce

un record all'interno del registro di sistema Windows. Il registro di sistema di Windows è un database gerarchico che archivia le informazioni, le impostazioni e le configurazioni necessarie per il funzionamento del sistema operativo Microsoft Windows. Il record in questione usa la chiave RunOnce all'interno del registro per far sì che il payload venga eseguito ad ogni accesso dell'utente. Infine il thread entra in un loop che si preoccupa di controllare la presenza della copia del payload e della chiave di registro, e nel caso reinserirli.

Una volta creato il thread, esegue l'inizializzazione delle *socket* e degli handler dei comandi che riceverà dal server.

In seguito invia un comando di *Join* al server attraverso HTTP, in cui all'interno inserisce le informazioni della macchina infetta ed in particolare:

- Version di Windows installata;
- Modello di CPU;
- Versione di framework .NET[1] installata;
- Quantità di RAM;
- Se l'utente ha diritti di amministratore;
- Identificativo Hardware (HWID) associato al dispositivo.

In fine entra in un loop che si occupa di gestire la comunicazione col server e tutte le fasi operazionali di cui è incaricato. Entrando nei dettagli del loop, invia periodicamente un comando di Poll a cui master risponde all'occorrenza con il prossimo comando da eseguire. Quando il comando viene ricevuto, ne esegue il parsing e nel caso sia un comando riconosciuto esegue l'handler ad esso associato (Figura 6.3). Infine attende una quantità di tempo preimpostata prima di ripetere il ciclo e quindi di rieffettuare il polling verso il server.

Un comando è composto di tre campi: identificativo, tipo e dati. Il campo dei dati è utilizzato per passare parametri aggiuntivi al bot o per l'inoltro di risultati di operazioni al server.

Nella sezione di codice 6.1 si può osservare un esempio di handler dei comandi. L'handler in esame è responsabile di inoltrare uno screenshot al server. In primo luogo analizza i dati aggiuntivi passati come parametro per estrapolare la porta a cui è associato il processo incaricato di gestire gli upload. In seguito viene eseguito lo screenshot e inoltrato al server sulla porta specificata. Infine invia l'esito del task al server con un ulteriore comando.

L'inoltro di un comando è preceduto da una fase di offuscamento di seguito descritta:

[1]ambiente di esecuzione realtime della piattaforma .NET [69].

```
1  void ScreenshotParser(int commandId, int commandType, char* data)
2  {
3    int len;
4    char** splitResults = SplitString(data, "@", &len, false);
5    unsigned short port = (unsigned short)strtoul(splitResults[1], NULL, 0);
6
7    int length;
8    char* scr = CaptureScreenshot(&length);
9
10   bool result = SendScreenshot(splitResults[0], port, scr, length,
11             GetBotId());
12   FreeSplitStringBuffer(splitResults, len);
13
14   char* command = CreateCommand(commandId, commandType,
15             result ? "Success" : "Fail", result ? 7 : 4);
16   FreeScreenshot(scr);
17   char* response = SendCommandWithDecodedResponse(command);
18   FreeDecodedResponse(response);
19 }
```

Listing 6.1: Esempio di handler di comandi.

1. Generazione di una chiave di 32 Byte generata con algoritmo pseudocasuale;
2. Il comando, compreso dei tre campi precedentemente descritti, viene combinato con la chiave attraverso XOR.
3. Il comando e la chiave vengono codificati in Url *charset*.
4. La chiave viene inserita negli header della richiesta POST mentre il comando nel *body*.

## 6.2 Deploy, testing e debug

Il deploy del server è stato eseguito attraverso containerizzazione del web server e del DBMS. In particolare è stato utilizzato Docker Compose per eseguire contemporaneamente entrambi i container attraverso un unico file di configurazione ed un unico comando. Questo (6.2) illustra le porte e le directory da mappare, le variabili d'ambiente da utilizzare per la configurazione di MariaDB, nonché altre informazioni pertinenti, come la rete a cui associare i servizi e informazioni di risoluzione dei container. L'handler delle richieste di upload veniva eseguito dall'interno del container all'occorrenza, eseguendo uno script PHP in background.

Dopo aver configurato l'indirizzo del server C&C nel codice sorgente del payload e installato gli strumenti necessari, come il Software Developer Kit (SDK) e i Build Tools, il payload è stato compilato ed eseguito in Visual Studio. Questo ha permesso di utilizzare il *debugger* integrato dell'IDE per analizzare il funzionamento del bot.

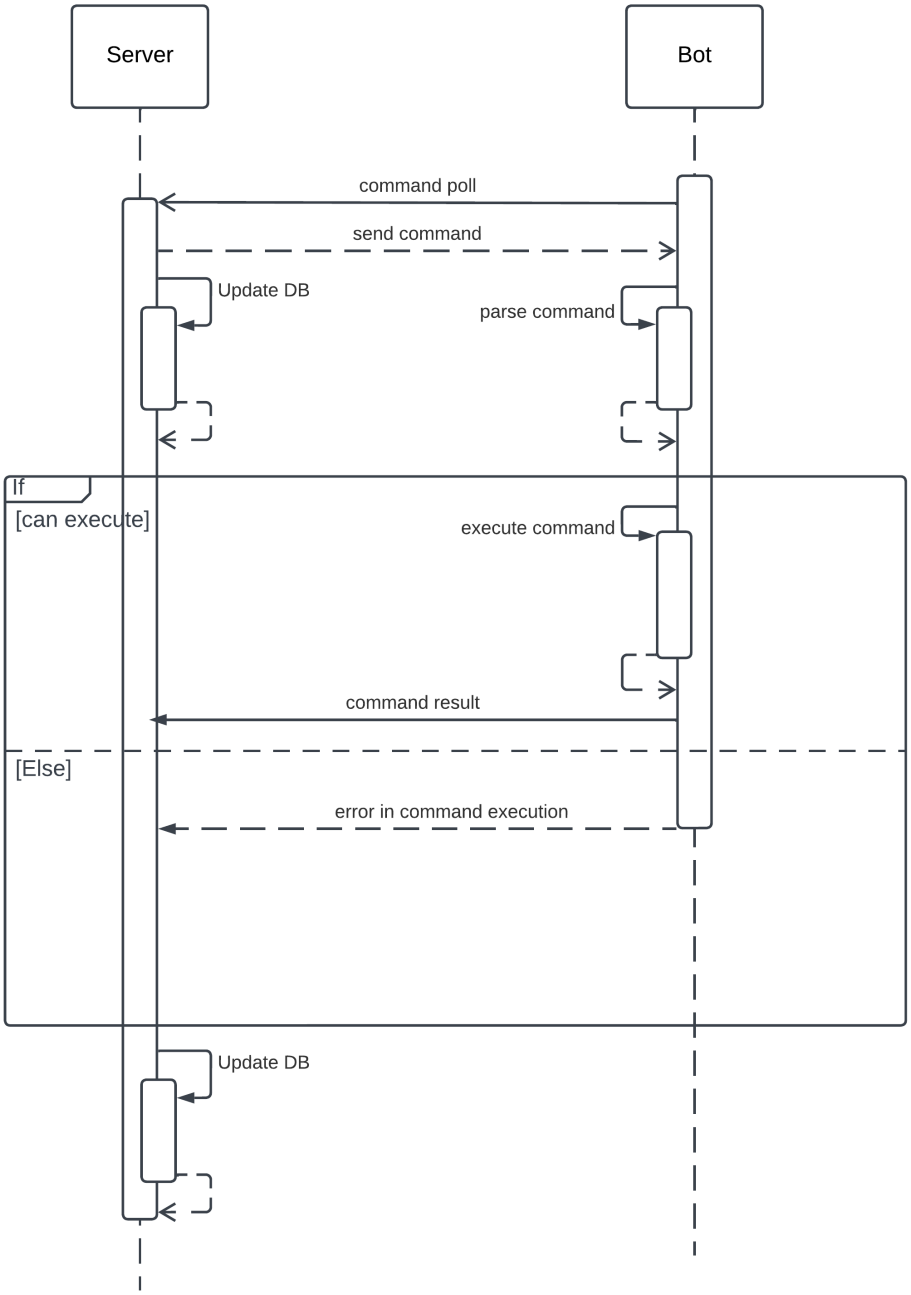



Figura 6.3: esecuzione di un comando richiesto al C&C.

Prima di procedere con il test della botnet, sono state apportate alcune modifiche al fine di risolvere alcuni dei bug individuati, i quali verranno di seguito descritti:

- Le richieste HTTP del server non erano indirizzate verso il corretto percorso di rete (path);
- Quando il bot inviava il messaggio di Join al server, le informazioni inserite al suo interno erano posizionate in un ordine differente dall'ordine in cui venivano elaborate dal server. Il server quindi, nel procedere all'inserimento in database dei dati ricevuti, generava un'eccezione, poiché i tipi dei dati inseriti non combaciavano con i tipi dei dati della *query* di inserimento;
- Il percorso in cui venivano salvati i risultati del Keylogger, era formattato erroneamente;
- Una delle directory in cui venivano salvate delle informazioni non era presente al momento dell'esecuzione e non avendo abbastanza permessi per crearla ne conseguiva il malfunzionamento;
- L'inoltro dello screenshot è diviso in due scambi di dati tra server e bot, ma era coordinato male. In particolare vi era un problema di sincronizzazione per via della ricezione non bloccante dei messaggi;
- La funzionalità di persistenza, che prevede l'inserimento di un record nel registro di sistema, presentava un bug che causava l'inserimento continuo della chiave nel registro stesso.

Una volta sistemati questi bug è stato possibile testare le varie funzionalità operative della botnet:

- Esecuzione di processo da remoto con possibilità di aggiungere parametri. La possibilità di aggiungere parametri permette anche di eseguire script, ad esempio, eseguendo un interprete di comandi e passandogli comandi interpretabili;
- Funzionalità DDoS, seppure l'implementazione non sia molto efficiente. Essa infatti consiste di un loop in cui vengono aperte connessioni TCP verso il target per inviare una serie di dati. Per provvedere alla scarsa efficienza della funzionalità built-in, è stato testato un tool di DDoS reperito in Internet, che è stato eseguito per mezzo della funzionalità di esecuzione di processi;
- Keylogging, ovvero la capacità di poter estrapolare dal bot, in forma di dati, le digitazioni da tastiera dell'utente;
- Download ed esecuzione di eseguibile. Un dettaglio importante che verrà utilizzato nella fase di rilevazione, è che il bot usa l'*esplora risorse* di Windows per eseguire i file scaricati;

- La funzionalità di update del bot è molto simile al *download and execute* già visto. Il bot esegue il download del nuovo payload, lo esegue ed in seguito termina la propria esecuzione per lasciare spazio alla versione aggiornata;

- Capacità di inviare una richiesta HTTP ad un sito web target per controllare che sia operativo. Ritorna esito positivo solo in caso la risposta HTTP abbia codice 200.

```
version: '3'

services:
  webserver:
    image: uboat-webserver
    build: .
    ports:
      - "80:80"
      - "25019:25019"
    volumes:
      - ./public:/var/www/html
      - ./src:/var/www/src
      - ./logs:/var/www/src/logs
      - ./screenshots:/var/www/src/screenshots
    links:
      - mysql
    networks:
      - vlan

  mysql:
    image: mariadb:10.1
    ports:
      - "3306:3306"
    volumes:
      - ./container/mysql/var/lib/mysql:/var/lib/mysql
      - ./container/sqlFiles:/home/sqlFiles
    environment:
      - MYSQL_DATABASE=uboat
      - MYSQL_USER=uboat
      - MYSQL_PASSWORD=uboat
      - MYSQL_ROOT_PASSWORD=root
    networks:
      - vlan

networks:
  vlan:
```

Listing 6.2: File di configurazione docker compose.

## 6.3 Rilevazioni con IDS

### 6.3.1 Network based IDS

L'invio periodico delle richieste di Poll da parte del bot non indicano niente di sospetto se non che le richieste sono destinate a indirizzi IP e non a hostname, cosa abbastanza insolita in un contesto normale.

Se l'indirizzo IP del server fosse noto come un indirizzo IP malevolo e fosse inserito all'interno di liste di IP considerati pericolosi, gli IDS potrebbero rilevare il traffico di rete sfruttando queste liste.

La funzionalità di download and execute del server viene rilevata dagli IDS. In particolare è stato segnalato il download di un eseguibile con pochi header, tipico di file Stager, e che il download è avvenuto verso un indirizzo IP non specificandone l'hostname. Inoltre rileva anche la presenza della keyword 'isDebuggerPresent', tipicamente utilizzata all'interno di malware per applicare restrizioni di anti-debugging. In caso di download di file malevolo noto come un malware, Strelka si preoccuperebbe di segnalarne l'identificazione

La funzionalità di update porta agli stessi risultati in quanto il funzionamento è quasi identico.

Il controllo dello stato di un sito web non ha ovviamente rilevato nulla in quanto è basato su una semplice richiesta HTTP, ed essendo questo un protocollo comune in rete.

L'esecuzione di un processo da remoto, l'estrapolazione di screenshot e il keylogging non sono stati segnalati, in quanto il loro comportamento malevolo avviene principalmente sull'host piuttosto che sulla rete.

L'attacco DDoS non ha portato a segnalazioni.

Lo scambio di comandi tra bot e server non è segnalato come malevolo in quanto non erano presenti regole di matching mirate alla rilevazione di UBoat, oltre al fatto che i comandi all'interno del corpo del messaggio sono offuscati. Un modo per poter rilevare la presenza di UBoat potrebbe essere sfruttare le capacità di LUA scripting di Suricata per deoffuscare il messaggio, in congiunta con l'inserimento di regole specifiche basate sul matching del formato dei messaggi di UBoat.

In aggiunta alle funzionalità della botnet è stata sfrutta la capacità di instaurare una reverse-shell sfruttando le capacità built-in, per analizzare se gli IDS l'avrebbero segnalata. In Figura 6.4a si può osservare il comando composto[2] utilizzato per creare una reverse shell sfruttando Nmap [53]. Il server apre una porta e rimane in ascolto mentre il bot, attraverso Nmap, esegue un comando e ne redirige l'input e l'output verso la porta aperta del server.

[2]formulato per testare anche la capacità di eseguire più comandi in un solo messaggio.

(a) Comando per la creazione di una reverse-shell.


(b) Shell collegata al bot.

Figura 6.4

A scopo di test, Nmap è stato installato sulla macchina vittima, in un contesto reale si potrebbe sfruttare la capacità di composizione dei comandi (osservabile in Figura 6.4a) per eseguirne prima il download e poi l'esecuzione.

In Figura 6.4b si può osservare il successo effettivo dell'operazione di reverse-shell, che tuttavia, l'IDS non ha rilevato.

### 6.3.2 Host based IDS

Combinando Sysmon con Wazuh è stato possibile sfruttare a pieno le funzionalità di HIDS che offre come Agent. Attraverso il set di regole di Wazuh, molto vasto e completo, è stato possibile identificare azioni potenzialmente malevole come la creazione di un record all'interno del registro di sistema o la creazione di file in zone delicate. Per sfruttare a pieno le capacità di rilevazione di Wazuh, e testare a pieno le capacità di rilevazione signature-based, sono state create delle regole Wazuh atte ad identificare UBoat. Queste regole sono basate sulle principali caratteristiche di UBoat descritte in precedenza, e su altri dettagli implementativi.

Attraverso l'aggiunta delle seguenti configurazioni a quelle dell'agent sulla macchina target è stato possibile monitorare le connessioni aperte e directory specifiche. In caso di cambiamenti, come aggiunta o rimozione di file, all'interno di una delle directory monitorate, verrà generato un evento.

```
<ossec_config>
    <syscheck>
        <disabled>no</disabled>
        <directories realtime="yes">path a directory
            da monitorare</directories>
        ...
    </syscheck>
    <localfile>
        <log_format>full_command</log_format>
        <command>netstat -n</command>
        <alias>netstat command</alias>
        <frequency>60</frequency>
        ...
    </localfile>
    ...
```

```
</ossec_config>
```

Di seguito analizzeremo le regole utilizzate per identificare le varie funzionalità operative della botnet. Le regole 100013 e 100014 sono utilizzate per identificare connessioni verso l'indirizzo IP della botnet, sfruttando la lista delle connessioni aperte inviate dall'agent. Le regole 100015 e 100016 sono state realizzate per cancellare il rumore creato dall'esecuzione del comando utilizzato per identificare le connessioni di rete sulla macchina monitorata.

```
<group name="ossec">
    <rule id="100013" level="0">
        <if_group>ossec</if_group>
        <match>netstat</match>
        <description>netstat command execution</description>
    </rule>
    <rule id="100014" level="12">
        <if_sid>100013</if_sid>
        <match>100.100.100.151</match>
        <description>Botnet - connection to botnet server</description>
    </rule>
    <rule id="100015" level="0">
        <if_sid>92052</if_sid>
        <field name="win.eventdata.parentImage">wazuh-agent.exe</field>
        <description>agent check command execution</description>
    </rule>
    <rule id="100016" level="0">
        <if_sid>92032</if_sid>
        <field name="win.eventdata.parentImage">cmd.exe</field>
        <field name="win.eventdata.image">NETSTAT.EXE</field>
        <description>agent check command execution</description>
    </rule>
</group>
```

La regola 100007 sfrutta la capacità di monitorare directory designate precedentemente descritta, e genera un allarme nel caso nella particolare directory sia inserito un file eseguibile denominato 'filename.exe'. In questo caso si sfrutta l'hardcoding del nome del file per identificare la botnet.

```
<group name="syscheck">
    <rule id="100007" level="12">
        <if_sid>554</if_sid>
        <regex>foldername\\\\filename.exe\.*added</regex>
        <description>Botnet - malware file creation for persistance
            </description>
    </rule>
</group>
```

Di seguito verranno le analizzate le regole che sono state realizzate sfruttando le funzionalità di generazione eventi di Sysmon. Le regole 100002 e 100003 sono utilizzate per

allertare che è stata inserita una determinata chiave di registro in una specifica posizione del registro di sistema.

```
<rule id="100002" level="0">
    <if_group>sysmon</if_group>
    <match>Registry value set</match>
    <description>Registry value set</description>
    <options>no_log</options>
</rule>
<rule id="100003" level="12">
    <if_sid>100002</if_sid>
    <field name="win.eventdata.targetObject">\\\\RunOnce\\
        \\RegKey</field>
    <field name="win.eventdata.details">\\\\AppData\\\\Roaming\\
        \\FolderName\\\\FileName.exe</field>
    <description>Botnet - reg key insertion</description>
</rule>
```

La regola 100017 fa riferimento all'esecuzione del payload ormai persistenze all'interno del computer della vittima. Grazie al meccanismo precedentemente analizzato, sfrutta la chiave di registro 'RunOnce' per avviare l'esecuzione del bot all'accesso dell'utente, e questa regola si basa su questa azione.

```
<rule id="100017" level="12">
    <if_group>sysmon</if_group>
    <match>Process create</match>
    <field name="win.eventdata.image">FileName.exe</field>
    <description>Botnet - botnet execution</description>
</rule>
```

La regola 100018 ha fine analogo a quello della regola 100014, ovvero la rilevazione di una connessione col C&C noto. Tuttavia mentre la prima fa affidamento sul risultato dell'esecuzione di un comando sull'agent, la seconda sfrutta gli eventi di Sysmon per generare un allarme.

```
<rule id="100018" level="12">
    <if_group>sysmon</if_group>
    <match>Network connection detected</match>
    <field name="win.eventdata.destinationIp">100.100.100.151</field>
    <description>Botnet - connection to botnet server initiated
        </description>
</rule>
```

La regola 100019 sfrutta la rilevazione della regola 100018 in combinazione con la porta nota dell'handler degli upload per segnalare l'estrapolazione di screenshot.

```
<rule id="100019" level="12">
    <if_sid>100018</if_sid>
    <field name="win.eventdata.destinationPort">25019</field>
```

```
    <description>Botnet - screenshot exfiltration</description>
</rule>
```

Le regole 100020 e 100021 sono utilizzate in combinazione per rilevare uno spike del numero di connessioni in uscita dall'host monitorato e segnalare quindi l'origine di un possibile attacco DoS.

```
<rule id="100020" level="5">
    <if_group>sysmon</if_group>
    <field name="win.eventdata.sourceIp">100.100.100.197</field>
    <field name="win.eventdata.destinationIp" negate="yes">
        100.100.100.151</field>
    <description>connection initiated from host</description>
    <options>no_log</options>
</rule>
<rule id="100021" level="12" frequency="200" timeframe="10">
    <if_matched_sid>100020</if_matched_sid>
    <description>Possible DoS attack originated from host</description>
</rule>
```

Le regole 100022 e 100023 mirano ad identificare il download e l'esecuzione attraverso l'explorer manager, di file malevoli. Ancora una volta si sfrutta la conoscenza del sorgente del malware e il pattern matching con parole chiave.

```
<rule id="100022" level="12">
    <if_sid>92213</if_sid>
    <field name="win.eventdata.targetFilename">BOTNET</field>
    <description>Botnet - Downloaded executable</description>
</rule>
<rule id="100023" level="12">
    <if_sid>61640</if_sid>
    <field name="win.eventdata.image">explorer.exe</field>
    <field name="win.eventdata.commandLine">BOTNET</field>
    <description>Botnet - execution of downloaded file</description>
</rule>
```

Concludiamo lo studio di UBoat con una carrellata delle segnalazioni effettuate da Wazuh in seguito all'aggiunta delle regole e configurazioni presentate.

| | Time ↓ | Agent | Agent name | Technique(s) | Tactic(s) | Description | Level | Rule ID |
|---|---|---|---|---|---|---|---|---|
| › | Apr 12, 2023 @ 16:03:44.143 | 001 | bot2 | | | Botnet - connection to botnet server | 12 | 100014 |
| › | Apr 12, 2023 @ 16:02:56.706 | 001 | bot2 | | | Botnet - reg key insertion | 12 | 100003 |
| › | Apr 12, 2023 @ 16:02:56.299 | 001 | bot2 | T1565.001 | Impact | Integrity checksum changed. | 7 | 550 |
| › | Apr 12, 2023 @ 16:02:56.286 | 001 | bot2 | | | Botnet - malware file creation for persistance | 12 | 100007 |

Figura 6.5: Segnalazioni all'esecuzione del payload.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| › | Apr 12, 2023 @ 22:00:00.167 | 001 | bot2 | | | Botnet - execution of downloaded file | 12 | 100023 |
| › | Apr 12, 2023 @ 22:00:00.167 | 001 | bot2 | T1055 | Defense Evasion, Privilege Escalation | Sysmon - Suspicious Process - explorer.exe | 12 | 61640 |
| › | Apr 12, 2023 @ 22:00:00.158 | 001 | bot2 | | | Botnet - Downloaded executable | 12 | 100022 |

Figura 6.6: Segnalazioni in seguito alla funzionalità di Download and Execute.

| | | | | | | |
|---|---|---|---|---|---|---|
| › | Apr 12, 2023 @ 21:55:27.413 | 001 | bot2 | Botnet - screenshot exfiltration | 12 | 100019 |
| › | Apr 12, 2023 @ 21:55:27.367 | 001 | bot2 | Botnet - connection to botnet server initiated | 12 | 100018 |

Figura 6.7: Segnalazioni in seguito all'estrapolazione di screenshot.

| | | | | | | |
|---|---|---|---|---|---|---|
| › | Apr 12, 2023 @ 21:53:30.928 | 001 | bot2 | Possible DoS attack originated from host | 12 | 100021 |

Figura 6.8: Segnalazioni in seguito all'attacco DoS.

| | | | | | | |
|---|---|---|---|---|---|---|
| › | Apr 12, 2023 @ 16:06:57.459 | 001 | bot2 | Botnet - botnet execution | 12 | 100017 |

Figura 6.9: Segnalazioni in merito l'esecuzione all'accesso dell'utente.

# 7 Analisi e altri possibili approcci fingerprint based

Nei capitoli precedenti è stato osservato come l'utilizzo di un agente e delle relative funzionalità di HIDS possano essere utilizzate per aumentare il panorama di caratteristiche analizzabili di un sistema e incrementarne di conseguenza la sicurezza.

Dalle analisi condotte sulle due botnet, emerge che sono stati utilizzati principalmente software IDS che utilizzino strumenti di rilevazione basati su tecniche di pattern matching, prediligendo quindi l'approccio fingerprint based. Come osservato nel Capitolo 3, gli approcci fingerprint based funzionano molto bene per malware e attività malevole per cui esista già una firma identificativa al momento della rilevazione. Nello studio effettuato su UBoat (Capitolo 6) si può osservare come le regole create per l'identificazione della botnet siano basate su caratteristiche peculiari del software analizzato. Piccole variazioni o l'offuscamento all'interno del codice del payload, potrebbero rendere parte delle regole di rilevazione create inefficienti verso la nuova versione del software malevolo. Ad esempio, se il botmaster distribuisse una nuova versione del payload, che durante la fase operativa di instaurazione della persistenza inserisce una copia dell'eseguibile nel sistema con un nome o una posizione differenti da quelle specificate, si potrebbero perdere gli allarmi relativi a tale evento. Complicazioni aggiuntive come la cifratura del traffico di rete e tecniche di *detection evasion* possono rendere ancora più ardua la rilevazione.

Va considerato anche che le attività degli host monitorati potrebbero generare molta attività di log rendendo le rilevazioni molto rumorose e spesso difficili da analizzare. Inoltre come già precedentemente osservato, tale approccio non può essere applicato in caso di malware zero-day. Per cercare di ovviare a queste problematiche i ricercatori e vendor di sistemi di sicurezza informatici, cercano di tenere i database di fingerprint il più aggiornati possibile oltre a ricercare nuove tecniche principalmente basati su rilevazioni di anomalie.

Tra le tecniche e le possibilità di rilevazione fingerprint based analizzate, che sarebbe possibile integrare facilmente all'interno dell'infrastruttura di testing sviluppata, vi sono:

- Yara rules;
- Integrazione con virus engine;
- API calls.

Yara [70] è un tool utilizzato per ricercare e classificare malware. Attraverso Yara è possibile creare descrizioni, anche dette regole, basate su pattern testuali o binari. Le YARA rule sono scritte in una sintassi specifica che permette di definire una serie di criteri di ricerca per individuare un determinato oggetto o comportamento. Ogni regola YARA è composta da tre sezioni principali:

**Meta** Contiene il nome della regola e altre informazioni, come l'autore, la data di creazione e la descrizione della regola stessa.

**Strings** Contiene i pattern che vengono utilizzati all'interno della sezione Condition.

**Condition** La parte centrale della regola che definisce il criterio di ricerca da applicare. La condizione è composta di un'espressione booleana e di stringhe su cui eseguire il match.

Un esempio di Yara rule è il seguente [71]:

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Listing 7.1: Esempio di Yara rule.

All'analisi di un file, la regola in esempio, andrà alla ricerca dei pattern binari e testuali elencati nella sezione Strings e nel caso almeno uno venisse trovato la regola darebbe esito positivo e il file sarebbe etichettato come *silent_banker*. Le Yara rule sono utilizzate all'interno di Strelka in Security Onion per la ricerca di file malevoli all'interno del traffico di rete. Questo approccio può essere importato all'interno dell'host da monitorare. Attraverso Wazuh, per esempio, è possibile integrare Yara con il modulo di File Integrity Monitoring dell'agent. Così facendo, ogni volta che viene modificato un file monitorato Yara effettuerà una scansione su di esso alla ricerca di file malevoli sfruttando le Yara rule.

Per aumentare ulteriormente la sicurezza degli end-point è possibile integrare sistemi antivirus o simili, sfruttando le capacità di inoltro dei log degli agent o di altri software.

Wazuh ad esempio, all'interno della propria documentazione cita ClamAV [72] come possibile alternativa open source di software antivirus multi piattaforma, basato su fingerprint. Esso è in grado di supportare diversi tipi di signature per analizzare molteplici formati di file, compresi archivi ed email.

Su un sistema Windows può essere utile sfruttare i log di Windows Defender[1] per avere visibilità delle infezioni rilevate da esso o per analizzare i risultati di possibili scansioni effettuate.

Un'ultima menzione è la possibilità di integrazione del modulo di File Integrity Monitoring degli agent Wazuh con il servizio VirusTotal [73]. VirusTotal è un sistema che sfrutta più di 70 antivirus/scanner e molteplici liste di domini/IP malevoli, per eseguire l'analisi di file online direttamente dal portale web del servizio. Esso offre inoltre un API utilizzabile per sfruttare il servizio programmaticamente. Grazie all'integrazione con Wazuh, quando l'agente segnala un cambiamento, una creazione o una cancellazione di un file, il server utilizza l'API di VirusTotal per inoltrare l'hash del file e ricevere i risultati dell'analisi da esso effettuata.

In conclusione, questi sono solo alcuni dei rilevanti esempi di integrazione che è possibile effettuare per aumentare ulteriormente l'analisi su un sistema. Il vasto mercato dei sistemi di sicurezza lascia un'ampia scelta all'amministrazione della sicurezza, permettendo di personalizzare ulteriormente il livello di protezione del sistema.

Per quanto riguarda le API call invece, si è voluto testare uno dei meccanismi utilizzato da molti Endpoint Detection and Response (EDR) e antivirus, per il monitoraggio di processi in real-time alla ricerca di comportamenti malevoli. La tecnica in questione è chiamata API hooking.

L'API hooking consente di monitorare e intercettare le chiamate ad una o più API call di un sistema operativo o di un'applicazione, per modificare o estendere il loro comportamento. In pratica, l'API hooking consente di intercettare la comunicazione tra un'applicazione e il sistema operativo, permettendo di monitorare e modificare il comportamento dell'applicazione o del sistema operativo stesso, ad esempio per rilevare e prevenire attacchi informatici. Gli sistemi di sicurezza sfruttano questa tecnica per intercettare API call comunemente utilizzate da malware per osservarne il comportamento alla ricerca di pattern sospetti.

L'idea di fondo era sfruttare questa tecnica per monitorare le chiamate di sistema di un software alla ricerca di pattern nell'ordine o insieme di chiamate effettuate.

Entrando nei dettagli, un possibile esempio di hooking di chiamata di funzione in un processo a 32 bit può essere il seguente [74], [75]:

1. Si estrae l'indirizzo in memoria della funzione di cui si vuole eseguire l'hook;

2. Vengono salvati i primi 5 byte della funzione;

[1] antivirus installato di default all'interno dei sistemi operativi Windows.

3. Si crea la funzione a cui passerà il controllo quando la funzione di cui si vuole eseguire l'hook viene chiamata, che chiameremo *HookedFunction*. Prima che questa finisca la propria esecuzione, riscriverà i byte precedentemente salvati al loro posto, per poi eseguire la funzione originale;

4. Si estrae l'indirizzo in memoria della HookedFunction;

5. Si sovrascrivono i primi 5 byte della funzione originale con gli opcode di jump assoluto o relativo all'indirizzo della HookedFunction.

Questo è il modo più banale di eseguire tale operazione, in quanto ad ogni esecuzione della funzione di cui si esegue l'hook i 5 byte di partenza vengono sovrascritti più volte. Un implementazione più efficiente può essere ottenuta sfruttando i cosiddetti *trampolini*. Un trampolino è una funzione che esegue quanto segue [75]:

1. Esegue i byte originali precedentemente salvati della funzione di cui è eseguito l'hook;

2. Salta all'interno del codice della funzione originaria, dopo l'istruzione di jump inserita manualmente.

Tuttavia tale operazione richiede la gestione del caso in cui la sovrascrittura dei 5 byte iniziali dovessero rompere la sintassi della restante porzione di codice.

Comunque in entrambi i casi, quando la funzione da monitorare viene eseguita, il controllo passerà alla funzione da noi definita per poi tornare ad eseguire la funzione originale.

Le cose sono però più complicate in processi a 64 bit, dato che le funzioni potrebbero essere posizionate tra loro troppo distanti per far sì che siano raggiungibili con un jump relativo a 32 bit. Dato che non esistono jump relativi a 64 bit, si potrebbe pensare di usare un jump assoluto ad un indirizzo di memoria preciso a 64 bit. Tuttavia tale salto richiede almeno 13 byte di opcode, e quindi tale approccio non sarebbe utilizzabile per funzioni che abbiano meno di 13 byte da sovrascrivere.

Per risolvere questo problema si può eseguire un salto relativo a 32 bit, che richiede 5 byte di opcode, ad una porzione di codice allocata in vicinanza (così che sia raggiungibile con un salto relativo) e quindi eseguire il jump assoluto all'indirizzo della HookedFunction.

Wazuh offre la possibilità di monitoraggio di API call su Linux sfruttando l'integrazione con l'Audit System del sistema operativo. Tuttavia, dato che la botnet su cui si sarebbe voluto effettuare il test (UBoat) può eseguire solo su Windows è stato ricercato un software alternativo che servisse allo scopo.

Inizialmente è stato utilizzato WinAPIOverride [76], un tool di monitoraggio e hooking di API call per processi a 32 e 64 bit. Questo offre anche un tool per l'estrazione di tutte le API call effettuate da un software, così che risulti più facile estrarne una possibile fingerprint per i nostri scopi. A basso livello esso utilizza un Kernel driver rendendo quindi

possibile intercettare ogni chiamata di sistema senza dover iniettare codice in applicativo. Specificando quali API call si desiderasse monitorare e caricando un DLL contenenti le strutture dati con le funzioni sui cui eseguire l'hook e le funzioni a cui ridirigerle, WinAPIOverride si preoccupava di eseguire l'hooking. All'interno della sezione di codice 7.2 è possibile vedere un esempio di struttura dati utilizzata in cui viene specificata l'API call sul quale eseguire l'hook, in questo caso Sleep della libreria kernel32.dll, e la funzione a cui reindirizzare il flusso di esecuzione con relativi parametri.

```
STRUCT_FAKE_API_WITH_USERPARAM pArrayAfterAPICall[] =
{
 {_T("kernel32.dll"),_T("Sleep"),(FARPROC)MyFunction,
              StackSizeOf(DWORD)  ,0,0},
 {_T(""),_T(""),NULL,0,0,0}
};
...
```

Listing 7.2: esempio di struttura dati per hook in WinAPIOverride.

Seppure il programma funzionasse correttamente, la licenza gratuita del tool non permette l'utilizzo su macchina virtuale. Per questo motivo è stato deciso di implementare un piccolo prototipo d'esempio senza l'ausilio di strumenti di terze parti, che sarebbe quindi potuto essere testato su macchina virtuale. Il prototipo è stato tenuto il più semplice possibile in quanto il fine ultimo era il consolidamento di quanto appreso riguardo la tecnica di hooking. Lo scopo del prototipo è intercettare le API call di un processo in esecuzione e di inviare un log al Windows Event Log System per notificarne l'accaduto, così che l'agente Wazuh potesse inviare tale log al Server per ulteriore analisi e intrusion detection.

Senza il supporto del tool era necessario trovare un modo per poter eseguire il codice, necessario per svolgere le attività di hooking, all'interno del processo da monitorare. Per evitare di dover scrivere un Kernel driver da zero in modo simile a WinAPIOverride, è stato deciso di utilizzare una tecnica chiamata DLL injection, cui fine è l'esecuzione di codice arbitrario all'interno di un processo. Vi sono molti modi per effettuare ciò. La versione di tecnica utilizzata consiste nel forzare un processo ad eseguire un thread, che è incaricato di caricare all'interno del contesto del processo una DLL [77]. Il caricamento di una DLL comporta come detto, tra le cose, l'esecuzione nel contesto del processo del codice contenuto all'interno del *DllMain* della libreria.

Ricapitolando quindi, inserendo il codice responsabile dell'hooking all'interno della DLL è possibile eseguirlo attraverso DLL injection all'interno del processo così monitorato.

Entrando nei dettagli, l'implementazione si compone dei seguenti punti:

1. Si ottiene l'handle del processo che si desidera monitorare;

2. Si alloca memoria all'interno del processo;

3. Si scrive all'interno della memoria riservata il path al DLL da iniettare;

4. Si ottiene l'indirizzo in memoria della funzione di libreria utilizzata per caricare librerie;

5. Si crea un thread all'interno del processo cui scopo della routine è l'esecuzione della funzione di caricamento di librerie precedentemente procurata a cui viene passato il path della DLL da iniettare come parametro.

```
int main()
{
    HANDLE proc = GetProcessByName("iniettami3.exe");
    if (proc == NULL) {
        return -1;
    }
    wchar_t dll[] = TEXT("C:\\Users\\manue\\Desktop\\evilDLL.dll");
    LPVOID mem = VirtualAllocEx(proc, NULL, sizeof dll, MEM_COMMIT, PAGE_READWRITE);

    WriteProcessMemory(proc, mem, (LPVOID)dll, sizeof dll, NULL);

    PTHREAD_START_ROUTINE threatStartRoutineAddress = (PTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandle(TEXT("Kernel32")),
            "LoadLibraryW");

    CreateRemoteThread(proc, NULL, 0, threatStartRoutineAddress, mem, 0,NULL);
    CloseHandle(proc);
}
```

Listing 7.3: Porzione di codice del DLL injector.

In seguito è stato implementato il DLL sfruttando i dettagli in merito alla tecnica di hooking sopra descritti. In particolare si è optato per la versione a 64 bit senza trampolino.

```
int log()
{
    HANDLE event_log = RegisterEventSource(NULL, L"mylogname");
    LPCWSTR str = TEXT("event log messagg");
    ReportEvent(event_log, EVENTLOG_SUCCESS, 0, 0, NULL, 1, 0, &str, NULL);
    std::cout << "logged";
    return 0;
}

void HookedSleep(DWORD dwMilliseconds) {
    std::cout << "niceu";
    log();
    memcpy(originalSleep, originalBytes, 5);
    Sleep(dwMilliseconds);
    memcpy(originalSleep, jmpInstruction, sizeof(jmpInstruction));
    return;
}

void InstallHook(void* func2hook, void* payloadFunction)
{
    void* relayFuncMemory = AllocatePageNearAddress(func2hook);
    WriteAbsoluteJump64(relayFuncMemory, payloadFunction);
    DWORD oldProtect;
    VirtualProtect(func2hook, 1024, PAGE_EXECUTE_READWRITE, &oldProtect);
    const uint64_t relAddr = (uint64_t)relayFuncMemory - ((uint64_t)func2hook + sizeof(jmpInstruction));
    memcpy(jmpInstruction + 1, &relAddr, 4);
    memcpy(func2hook, jmpInstruction, sizeof(jmpInstruction));
}

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        originalSleep = GetProcAddress(GetModuleHandleW(TEXT("Kernel32")), "Sleep");
        ReadProcessMemory(GetCurrentProcess(), originalSleep, originalBytes, 5, NULL);
        InstallHook(originalSleep, HookedSleep);
```

```
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Listing 7.4: Porzione di codice della DLL (evilDLL.dll).

```
int main()
{
    while (true) {
        std::cout << "Hello World!\n";
        Sleep(2000);
    }
}
```

Listing 7.5: semplice programma console su cui eseguire il test (iniettami3.exe).

In Figura 7.1 si può osservare come la DLL sia effettivamente stata caricata all'interno del processo da monitorare. Mentre in Figura 7.2 si può notare come il log creato sia stato notificato correttamente al sistema Windows. In fine in Figura 7.3 si può notare come il comportamento del programma si stato alterato in seguito all'hook.

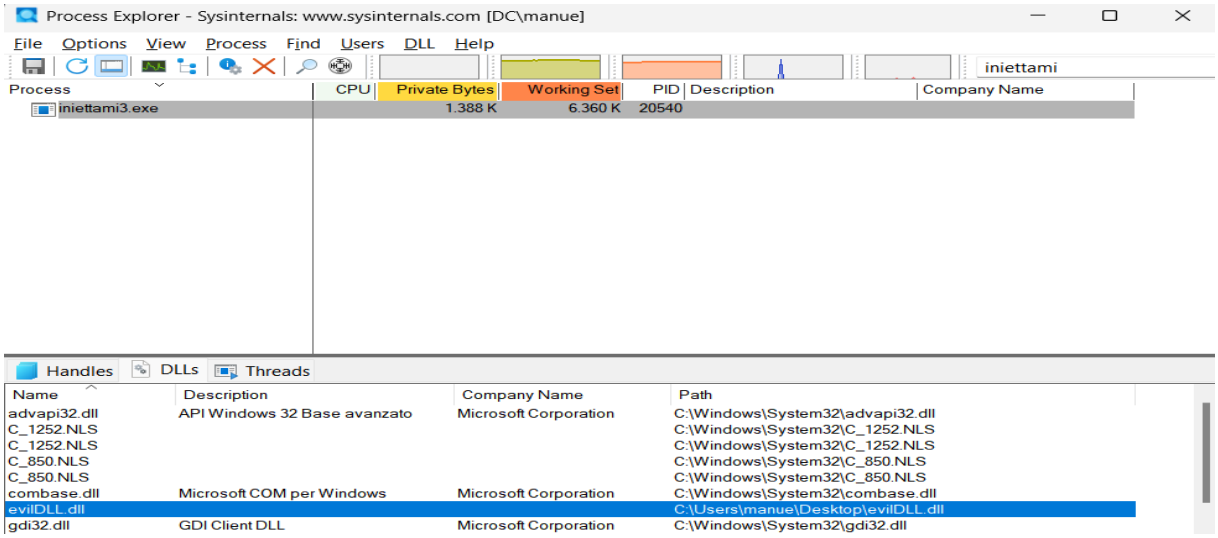



Figura 7.1: DLL all'interno del processo monitorato.


Figura 7.2: generazione di evento in Windows Event Log System.

Estendendo l'idea di fondo del prototipo si potrebbe iniettare una DLL all'interno di ogni nuovo processo alla ricerca di pattern nelle API call, ed eventualmente generare dei log all'interno del Windows Event Log System così da poterli analizzare attraverso Wazuh.

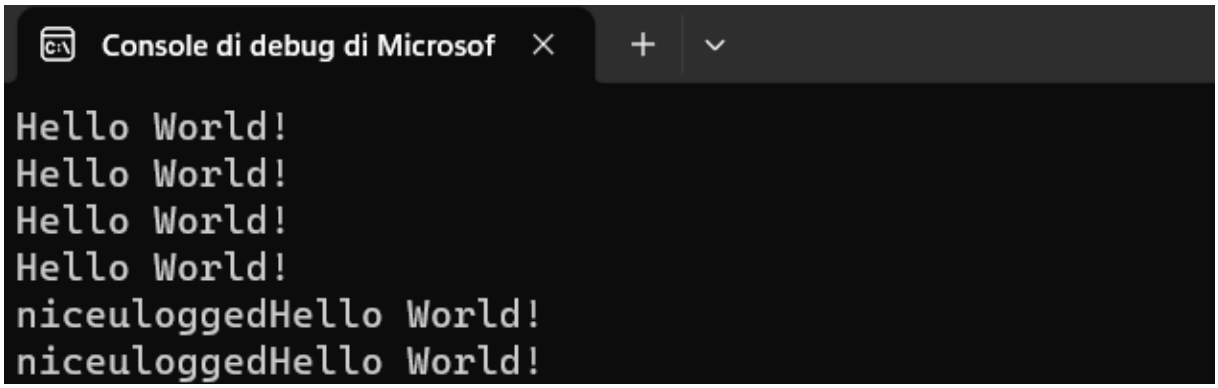



Figura 7.3: console del processo prima e dopo l'hook.

Un'applicazione del concetto di monitoraggio delle system call molto interessante è stata effettuata da Creech e Gideon [78], che hanno sfruttato tale concetto per creare un HIDS che rilevasse anomalie nelle chiamate effettuate.

# 8 Conclusioni

Il lavoro svolto ha riguardato lo studio delle botnet, sia dal punto di vista teorico che pratico. Sono state analizzate diverse implementazioni di botnet open source per comprenderne il funzionamento effettivo e testare la loro rilevazione attraverso NIDS e HIDS come Security Onion, Ossim e Wazuh. Grazie all'uso di agent installati sulle macchine da monitorare e strumenti di rilevazione di rete, è stato possibile identificare correttamente la presenza di bot all'interno dell'infrastruttura, confermando l'efficacia dei comuni strumenti di rilevazione basati su pattern matching di fingerprint.

Tuttavia, i test hanno anche evidenziato le attuali debolezze degli approcci fingerprint based, come le complicanze di rilevazione dovute a tecniche di detection evasion, sfruttate dalle moderne botnet in circolazione, o l'impossibilità di rilevare malware o attività zero day che sono prive di firme identificative. Questo ha portato alla considerazione di approcci di rilevazione delle botnet basati su anomalie, che rappresentano oggi la principale classe di tecniche di rilevazione di botnet studiate dai ricercatori.

All'interno del lavoro svolto, sebbene non siano stati inseriti all'interno del documento, vi è anche lo studio di strumenti di automazione come Ansible [79] e Terraform [80] che possono essere utilizzati per automatizzare il processo di deploy e la configurazione dell'infrastruttura di testing per possibili estensioni future.

In conclusione, il lavoro svolto ha permesso di comprendere meglio il funzionamento delle botnet e di valutare l'efficacia degli strumenti di rilevazione utilizzati. Tuttavia, data l'evoluzione continua delle tecniche di evasion delle botnet, è importante continuare a studiare e sviluppare nuovi approcci di rilevazione per garantire la sicurezza delle infrastrutture IT.

# Bibliografia


[1] C. .-. A. I. per la Sicurezza Informatica, «Rapporto CLUSIT 2023 sulla sicurezza ICT in Italia», 2023. indirizzo: https://clusit.it/pubblicazioni/.

[2] F-secure, *Ramnit*. indirizzo: https://www.f-secure.com/v-descs/virus_w32_ramnit.shtml.

[3] «Qbot/QakBot Malware», 2020. indirizzo: https://www.cisa.gov/sites/default/files/2023-02/202010221030_qakbot_tlpwhite.pdf.

[4] *Il malware Zeus trojan*. indirizzo: https://www.kaspersky.it/resource-center/threats/zeus-trojan-malware.

[5] G. Tubin. «Ramnit Evolution: From Worm to Financial Malware». (2011), indirizzo: https://securityintelligence.com/ramnit-evolution-from-worm-to-financial-malware/.

[6] f-secure, *Preemptive Blocklist and More Downadup Numbers*. indirizzo: https://archive.f-secure.com/weblog/archives/00001582.html.

[7] B. K. Jena. indirizzo: https://www.simplilearn.com/ice9/free_resources_article_thumb/Botnet_2.png.

[8] G. Vormayr, T. Zseby e J. Fabini, «Botnet communication patterns», *IEEE Communications Surveys & Tutorials*, vol. 19, n. 4, pp. 2768–2796, 2017.

[9] S. T. Vuong e M. S. Alam, «Advanced methods for botnet intrusion detection systems», in *Intrusion Detection Systems*, IntechOpen, 2011.

[10] D. Hoang e V. Hạnh, «An Enhanced Model for DGA Botnet Detection Using Supervised Machine Learning», mar. 2021.

[11] *Internet Relay Chat Protocol*, RFC 1459, mag. 1993. DOI: 10.17487/RFC1459. indirizzo: https://www.rfc-editor.org/info/rfc1459.

[12] *Server Message Block Protocol (SMB)*. indirizzo: https://wiki.wireshark.org/SMB.md.

[13] *WASTE web page*. indirizzo: https://waste.sourceforge.net/.

[14] P. Maymounkov e D. Mazieres, «Kademlia: A peer-to-peer information system based on the xor metric», in *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers*, Springer, 2002, pp. 53–65.

[15] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal e N. Borisov, «Stegobot: a covert social network botnet», in *Information Hiding: 13th International Conference, IH 2011, Prague, Czech Republic, May 18-20, 2011, Revised Selected Papers 13*, Springer, 2011, pp. 299–313.

[16] *VMware ESXi main page*. indirizzo: `https://www.vmware.com/it/products/esxi-and-esx.html`.

[17] A. Desai, R. Oza, P. Sharma e B. Patel, «Hypervisor: A survey on concepts and taxonomy», *International Journal of Innovative Technology and Exploring Engineering*, vol. 2, n. 3, pp. 222–225, 2013.

[18] S. J.Bigelow, *A beginner's guide to hosted and bare-metal virtualization*. indirizzo: `https://www.techtarget.com/searchitoperations/tip/A-beginners-guide-to-hosted-and-bare-metal-virtualization`.

[19] VMware, *VMware virtual networking concepts*. indirizzo: `https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/virtual_networking_concepts.pdf`.

[20] S.-d. Krit e E. Haimoud, «Overview of firewalls: Types and policies: Managing windows embedded firewall programmatically», in *2017 International Conference on Engineering & MIS (ICEMIS)*, IEEE, 2017, pp. 1–7.

[21] *ClearOS 7 User Guide*. indirizzo: `https://documentation.clearos.com/index:userguide7`.

[22] *Netfilter main page*. indirizzo: `https://www.netfilter.org/index.html`.

[23] J. Ellingwood, *A Deep Dive into Iptables and Netfilter Architecture*. indirizzo: `https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture`.

[24] B. Kang, K. Anh e H. Choo, *Implementation of fast handover for proxy mobile IPv6: Resolving out-of-order packets*, 2017. DOI: `10.1371/journal.pone.0182375`. indirizzo: `https://doi.org/10.1371/journal.pone.0182375.g005`.

[25] *Nftables main page*. indirizzo: `https://www.netfilter.org/projects/nftables/index.html`.

[26] *Firewalld main page*. indirizzo: `https://firewalld.org/`.

[27] A. Mouat, *Using docker: developing and deploying software with containers*. " O'Reilly Media, Inc.", 2015.

[28] *Docker main page*. indirizzo: `https://docker.com`.

[29] A. Khraisat, I. Gondal, P. Vamplew e J. Kamruzzaman, «Survey of intrusion detection systems: techniques, datasets and challenges», *Cybersecurity*, vol. 2, n. 1, pp. 1–22, 2019.

[30] J. Svoboda, I. Ghafir, V. Prenosil et al., «Network monitoring approaches: An overview», *Int J Adv Comput Netw Secur*, vol. 5, n. 2, pp. 88–93, 2015.

[31] *Security Onion main page*. indirizzo: https://securityonionsolutions.com/software.

[32] *Security Onion Documentation*. indirizzo: https://docs.securityonion.net/en/2.3/index.html.

[33] *Security Onion Image: example of usage*. indirizzo: https://docs.securityonion.net/en/2.3/_images/network-horiz.png.

[34] *Suricata main page*. indirizzo: https://suricata.io.

[35] *LUA scripting language*. indirizzo: https://www.lua.org/about.html.

[36] *zeek main page*. indirizzo: https://zeek.org/.

[37] *Stenographer main page*. indirizzo: https://github.com/google/stenographer.

[38] *Strelka main page*. indirizzo: https://github.com/target/strelka.

[39] *Sensoroni main page*. indirizzo: https://github.com/sensoroni/sensoroni.

[40] *Elastic Stack main page*. indirizzo: https://www.elastic.co/elastic-stack/.

[41] *ElastAlert main page*. indirizzo: https://elastalert2.readthedocs.io/en/latest/elastalert.html#overview.

[42] *Curator main page*. indirizzo: https://www.elastic.co/guide/en/elasticsearch/client/curator/current/about.html#about.

[43] *Redis main page*. indirizzo: https://redis.io/.

[44] *Grafana main page*. indirizzo: https://grafana.com/.

[45] MITRE, *ATT&CK main page*. indirizzo: https://attack.mitre.org/.

[46] *ATT&CK Navigator main page*. indirizzo: https://github.com/mitre-attack/attack-navigator.

[47] *Osquery main page*. indirizzo: https://osquery.io/.

[48] *Wazuh main page*. indirizzo: https://wazuh.com/.

[49] *Filebeat main page*. indirizzo: https://datatracker.ietf.org/doc/html/rfc3164.

[50] *Syslog RFC 3164*. indirizzo: https://datatracker.ietf.org/doc/html/rfc3164.

[51] *Security Onion distributed architecture image*. indirizzo: https://docs.securityonion.net/en/2.3/_images/distributed.png.

[52] *Security Onion: Hardware Requirements*. indirizzo: https://docs.securityonion.net/en/2.3/hardware.html.

[53] *Nmap main page*. indirizzo: https://nmap.org/.

[54] *Salt Project main page*. indirizzo: https://saltproject.io/.

[55] *ZeroMQ main page*. indirizzo: https://zeromq.org/.

[56] *YAML main page*. indirizzo: https://yaml.org/.

[57] *Jinja main page*. indirizzo: https://jinja.palletsprojects.com/en/3.1.x/.

[58] *Ossim main page*. indirizzo: https://cybersecurity.att.com/products/ossim.

[59] *Byob main page*. indirizzo: https://github.com/malwaredllc/byob.

[60] *Flask main page*. indirizzo: https://flask.palletsprojects.com/en/2.3.x/.

[61] *Sqlite main page*. indirizzo: https://sqlite.org/.

[62] W. A. Jansen, «Intrusion detection with mobile agents», *Computer Communications*, vol. 25, n. 15, pp. 1392–1401, 2002.

[63] D. B. Lange e M. Oshima, «Mobile agents with Java: the Aglet API», *World Wide Web*, vol. 1, n. 3, pp. 111–121, 1998.

[64] J. M. Smith, «A survey of process migration mechanisms», *ACM SIGOPS Operating Systems Review*, vol. 22, n. 3, pp. 28–40, 1988.

[65] P. Mell, D. Marks e M. McLarnon, «A denial-of-service resistant intrusion detection architecture», *Computer Networks*, vol. 34, n. 4, pp. 641–658, 2000.

[66] *Sysmon main page*. indirizzo: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon.

[67] SwiftOnSecurity, *sysmon-config | A Sysmon configuration file for everybody to fork*. indirizzo: https://github.com/SwiftOnSecurity/sysmon-config.

[68] *UBoat main page*. indirizzo: https://github.com/UBoat-Botnet/UBoat.

[69] *.NET framework main page*. indirizzo: https://dotnet.microsoft.com/en-us/download/dotnet-framework.

[70] *Yara main page*. indirizzo: https://github.com/VirusTotal/yara.

[71] P. Arntz, *Explained: YARA rules*. indirizzo: https://www.malwarebytes.com/blog/news/2017/09/explained-yara-rules.

[72] *ClamAV main page*. indirizzo: https://www.clamav.net/.

[73] *How it works - VirusTotal*. indirizzo: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works.

[74] *Windows API hooking*. indirizzo: https://www.ired.team/offensive-security/code-injection-process-injection/how-to-hook-windows-api-using-c++.

[75] K. Halladay, *X64 Function Hooking by Example*. indirizzo: http://kylehalladay.com/blog/2020/11/13/Hooking-By-Example.html.

[76] J. Potier, *WinApiOverride main page*. indirizzo: http://jacquelin.potier.free.fr/winapioverride32/.

[77] *DLL injection*. indirizzo: `https://www.ired.team/offensive-security/code-injection-process-injection/dll-injection`.

[78] G. Creech, «Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks», tesi di dott., UNSW Sydney, 2014.

[79] *Ansible main page*. indirizzo: `https://docs.ansible.com/`.

[80] *Terraform main page*.

[81] A. Compagno, M. Conti, D. Lain, G. Lovisotto e L. V. Mancini, «Boten ELISA: A novel approach for botnet C&C in online social networks», in *2015 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2015, pp. 74–82.

[82] *Visual Studio main page*. indirizzo: `https://visualstudio.microsoft.com/it/`.

# Ringraziamenti

Vorrei ringraziare innanzitutto il relatore, il Dott. Gabriele D'Angelo, e al correlatore, il Dott. Ciro Barbone, per la loro straordinaria disponibilità e professionalità che mi hanno consentito di vivere un'esperienza di tesi davvero stimolante.

Un ringraziamento speciale alla mia famiglia, per tutto quello che fanno per me, per il loro affetto e appoggio incondizionato.

Ringrazio di cuore Veronika, che in questo ultimo anno mi è sempre stata vicina, supportandomi e spronandomi anche durante le nottate di studio più matte e disperate.

Ringrazio Cicio, Lino, Vago e Salmon per essere ed essere stati al mio fianco.

Infine, desidero esprimere la mia gratitudine verso i miei amici, coloro che sono sempre al mio fianco e quelli che, nonostante il tempo e la distanza, so che ci saranno sempre.