Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

DESIGN E SVILUPPO DI UN FRAMEWORK SCALABILE E ISOLATO PER L'INDAGINE AUTOMATIZZATA SUI RANSOMWARE

Relatore:
Chiar.mo Prof.
SAVERIO GIALLORENZO

Correlatore:
Dott.
SIMONE MELLONI

Presentata da:
TOMMASO
COMPAGNUCCI

III Sessione
Anno Accademico 2022/2023

Indice

Capitolo 1

Introduzione

Le frodi informatiche sono in forte crescita. Un uso sempre più sfrenato degli strumenti digitali e una sensibilità ancora non sviluppata da parte degli utenti del Web nella gestione dei propri dati online, ha spalancato le porte a coloro che, indebitamente, vogliono trarre guadagni. Le frodi informatiche possono presentarsi in varie forme: una delle più affermate è il phishing. Questa tecnica consiste nell'invio di comunicazioni da parte di un malintenzionato che si finge un ente riconosciuto, richiedendo dati personali all'utente. Un tipo di frode informatica che sta proliferando negli ultimi anni è rappresentata dai ransomware. I ransomware sono software malevoli che, se eseguiti in un sistema informatico, ne limitano l'accesso da parte degli utenti, richiedendo un riscatto (ransom) al fine di rimuovere tale limitazione. In particolare, con la diffusione delle cryptovalute, si stanno affermando i cryptoransomware. Questi software criptano il contenuto di un sistema informatico e richiedono il pagamento su un portafoglio di criptovalute per la decriptazione. Negli ultimi 10 anni abbiamo assistito a diversi attacchi ransomware su larga scala. Un esempio è WannaCry: nel 2017 ha iniziato a diffondersi in tutto il mondo infettando oltre 23000 computer in 150 paesi, con richieste di riscatto in BitCoin in 28 lingue differenti. Europol lo ha definito come il più grande attacco ransomware di sempre.[Wik24]

Lo studio dei ransomware e del loro comportamento e lo sviluppo di strumenti che possano contrastarli è divenuto un argomento fortemente discusso nella comunità scientifica. Un esempio è Ranflood[BGM+24], il cui scopo è sopprimere l'azione dei ransomware tramite la tecnica del data flooding[BGM+23]. SAFARI (Scalable Air-gapped Framework for Automated Ransomware Investigation), il framework di seguito presentato, ha come obiettivo quello di aiutare gli esperti di sicurezza nello studio dei ransomware e nello sviluppo e nel testing di software che possano rilevare e contrastare la loro azione. Creando un ambiente isolato, al fine di non danneggiare

i beni informatici e automatizzando le operazioni da eseguire, SAFARI permette di testare i ransomware ed eventuali strumenti di contrasto in un ambiente virtualizzato. SAFARI sfrutta strumenti di Infrastructure-as-Code per creare un ambiente virtuale e strumenti di automazione della configurazione per eseguire i test.

Nel corso di questo elaborato vengono illustrati i principi sottostanti a SAFARI, il suo utilizzo e un caso di studio che ne valida l'utilizzo.

Nel Capitolo 2 vengono presentati i concetti e le tecnologie sfruttate da SAFARI per raggiungere il suo scopo. Nel Capitolo 3 vengono illustrati i principi che hanno guidato la creazione di SAFARI e come, utilizzando le tecnologie precedentemente presentate, è possibile applicare tali principi. Nel Capitolo 4 viene spiegato come utilizzare SAFARI, entrando nel dettaglio di ogni singola fase. Nel Capitolo 5, viene presentato il caso di studio, dove Ranflood viene integrato in SAFARI, al fine di parametrizzare il suo funzionamento, permettendo una più rapida fase di testing, rispetto a quella già effettuata manualmente.

Con grande orgoglio da parte mia, SAFARI è entrato a far parte del progetto open-source "Flooding-Against-Ransomware" di cui fa parte Ranflood. Il codice di SAFARI è consultabile al seguente link `https://github.com/Flooding-against-Ransomware/SAFARI`

Capitolo 2

Concetti e Tecnologie Necessarie

In questo capitolo, si presentano alcuni concetti e tecnologie su cui si basa SAFARI.

2.1 Virtualizzazione

In informatica, il termine virtualizzazione si riferisce alla possibilità di astrarre le componenti hardware degli elaboratori al fine di renderle disponibili al software in forma di risorsa virtuale[Wik23]

La virtualizzazione permette lo sfruttamento dell'hardware di una macchina da parte di più computer virtuali, detti macchine virtuali (VM). Le macchine virtuali possono eseguire sistemi operativi differenti fra loro e ognuna è indipendente dall'altra.

2.1.1 Hypervisor

Per attuare il processo di virtualizzazione si fa uso di un software specializzato, chiamato hypervisor. L'hypervisor si occupa della creazione e della gestione delle macchine virtuali, coordinando l'accesso alle risorse hardware sottostanti.

Esistono due tipi di hypervisor:

- Tipo 1: anche detti bare-metal, vengono direttamente installati sull'hardware.
- Tipo 2: vengono eseguiti sopra un sistema operativo già esistente.

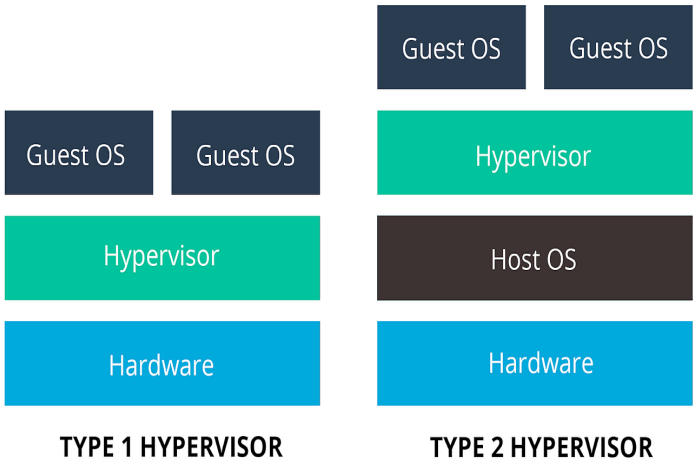

Figura 2.1: Tipi di hypervisor.

L'hypervisor è l'elemento centrale di una piattaforma di virtualizzazione, che invece può includere altre funzionalità come la migrazione delle macchine virtuali, il bilanciamento del carico o la gestione delle reti virtuali.

2.1.2 Vantaggi della Virtualizzazione

I vantaggi della virtualizzazione risiedono principalmente nella capacità di ottimizzare le risorse hardware della macchina su cui è installato l'hypervisor e l'isolamento offerto ad ogni macchina creata.

Tramite la virtualizzazione, possiamo eseguire più sistemi operativi sulla stessa macchina, evitando di avere più elaboratori fisici che eseguono una sola applicazione ognuno. Questa caratteristica permette una riduzione drastica dei costi hardware, legati all'acquisto e alla manutenzione.

Nel nostro specifico caso, la virtualizzazione è fondamentale perchè ci permette di creare una sandbox, ovvero un ambiente isolato in cui il lancio di programmi dannosi non andrà ad intaccare l'ambiente che coordina l'esecuzione dei test ma solo la risorsa virtuale creata.

L'uso di macchine virtuali permette di raggiungere un alto livello di isolamento non solo tra i processi software in esecuzione sullo stesso hardware ma anche tra il sistema operativo ospite e l'hardware stesso.[PZH13] Ciò è reso possibile dall'hypervisor, che pone un livello logico tra la macchina virtuale e l'hardware sottostante.

L'ambiente virtuale permette inoltre di replicare un sistema con determinate caratteristiche e contenuti più e più volte. Tramite il sistema degli snapshot e dei template, possiamo creare una struttura predefinita che può essere clonata, risultando in una vera e propria macchina virtuale con le caratteristiche definite nel modello di partenza. Eseguendo programmi pericolosi sulla macchina virtuale non perderemo le informazioni contenute nello snapshot o nel template.

La scalabilità è un'altra caratteristica della virtualizzazione che la rende potente. In un ambiente fisico, se si volessero incrementare le risorse, sarebbe necessario acquistare nuovo hardware, risultando in un costo economico, che andrebbe poi configurato, risultando in un costo temporale. La virtualizzazione permette di creare nuove risorse usando lo stesso hardware. L'unico limite esistente è quello dell'hardware su cui si esegue l'hypervisor.

2.2 Infrastruttura e Provisioning

Nel mondo dell'informatica, per infrastruttura si intende l'insieme di componenti hardware e software che costituiscono un ambiente informatico. Tra i componenti hardware troviamo computer desktop, dispositivi di archiviazione, router. Tra i componenti software possiamo avere sistemi operativi e web server.

L'infrastruttura che si vuole costruire per il lavoro presentato in questo elaborato è un'infrastruttura virtuale: l'utilizzatore del tool non avrà a sua disposizione macchine fisiche ma virtuali, il cui processo di creazione viene automatizzato grazie a strumenti ad hoc.

Un termine spesso accostato a infrastruttura è provisioning: con questo termine si intende il processo tramite cui una certa infrastruttura viene resa disponibile agli utenti.

Il provisioning può presentarsi in due forme: statico, ovvero si conoscono a priori le risorse necessarie all'utente, oppure dinamico, anche detto on-demand, che permette all'utente di richiedere nuove risorse a run-time.

Negli ultimi anni, grazie allo sviluppo di soluzioni di cloud computing, sono stati creati strumenti che permettono l'automatizzazione della fase di provisioning dell'infrastruttura mediante la scrittura di codice.

2.3 Infrastructure-as-Code

Il processo di provisioning di un'infrastruttura svolto in maniera manuale risulta essere molto dispendioso. Se pensiamo ad un semplice ambiente formato da pochi

PC, è necessario installare un determinato sistema operativo su ogni macchina manualmente, impostare le configurazioni di rete, installare gli applicativi necessari. Oltre ad essere dispendioso da un punto di vista temporale, la probabilità di andare incontro ad errori è molto alta, a causa dell'intervento manuale.

La creazione di strumenti Infrastructure-as-Code ha completamente rivoluzionato il modo di fare provisioning.

Infrastructure-as-Code (IaC) automatizza il processo di provisioning passando da un approccio manuale ad uno basato sulla scrittura di codice di alto livello.

Nel mondo dello IaC vengono usati due paradigmi di programmazione per l'esecuzione delle attività di configurazione dell'infrastruttura:

- imperativo: seguendo questo approccio, si deve specificare al tool IaC i passi da compiere per arrivare ad una determinata infrastruttura;
- dichiarativo: si specifica la configurazione desiderata e il tool IaC si impegna a crearla.

2.3.1 Terraform

Terraform[Has14] è uno strumento Infrastructure-as-Code che sfrutta linguaggio dichiarativo, precisamente linguaggio HCL (HashiCorp Configuration Language). Le componenti principali di Terraform sono i provider e le risorse:

- Provider: plugin specializzati nella comunicazione con i servizi on premise o cloud;
- Risorse: istanze che devono essere create. Possono essere macchine virtuali, container, database.

In un file TF si definiscono le risorse che devono essere create, specificandone le caratteristiche hardware e software. Il provider scelto si occupa quindi di istruire la piattaforma on premise o cloud circa le risorse da creare, accedendo alle API esposte dalla piattaforma stessa.

2.4 Configurazione Remota

Per configurazione remota si fa riferimento al processo tramite cui si svolgono dei compiti su una macchina senza una diretta interazione con essa. I compiti da svolgere possono essere di varia natura: aggiornamento del sistema operativo, installazione di pacchetti software, impostazione del firewall, trasferimento di file.

Un protocollo di fondamentale importanza nella gestione remota è SSH (Secure Shell). Esso permette la comunicazione tra due nodi della stessa rete tramite interfaccia a riga di comando. Quando si deve configurare una macchina, ci si connette via SSH e si impartiscono i comandi necessari.

Tuttavia, se si ha un numero considerevole di macchine, risulta scomodo dover ripetere il procedimento più volte. Per questo motivo sono nati strumenti di automazione. Con questi tool si scrive una sola volta la configurazione che tutte le macchine che vogliamo configurare devono rispettare.

2.4.1 Ansible

Ansible[eA12] è uno strumento che permette di automatizzare il processo di configurazione di una macchina. Viene definito agent-less in quanto non necessita l'installazione di software sulla macchina da configurare, ma sfrutta il protcollo SSH o Windows Remote Management in caso di macchine Windows.

In Ansible distinguiamo due attori: macchina controller e macchina host. La macchina controller esegue comandi sulla macchina host.

I comandi vengono specificati nei playbook, dei file definiti in formato YAML. All'interno dei playbook, tramite l'uso di moduli Ansible, ovvero programmi atti a svolgere un particolare compito, vengono riportati i comandi che il controller deve eseguire sull'host.

La lista delle macchine host da configurare viene chiamato inventario. All'interno dell'inventario si specificano gli indirizzi IP degli host. Ansible permette anche la definizione di gruppi, ovvero insiemi di macchine identificabili da un nome. Nei playbook è poi possibile eseguire comandi su un gruppo piuttosto che su un altro.

2.5 Air-Gap

Il termine air-gap si riferisce ad una misura di sicurezza nell'ambito delle reti tramite cui una rete viene isolata dall'ambiente esterno.

Si tratta di un approccio che mira a creare una separazione fisica o logica completa tra una rete o un sistema informatico e qualsiasi altra forma di connettività esterna, inclusa Internet e altre reti. Ci sono due principali modi in cui una rete air-gapped può essere implementata:

- Isolamento Fisico: in questo caso, i cavi di rete, gli apparati e i collegamenti fisici che fanno comunicare i dispositivi e i server non sono "fisicamente" connessi

con il mondo esterno. Questo tipo di isolamento garantisce una separazione fisica completa, rendendo estremamente difficile l'accesso non autorizzato;

- Isolamento Logico: l'isolamento logico implica l'uso di dispositivi di sicurezza come firewall o router per creare una barriera virtuale tra la rete air-gapped e altre reti. Questo metodo consente un certo grado di connessione tra i sistemi, ma attraverso un filtro software/hardware rigoroso che blocca l'accesso non autorizzato[Bro23].

Capitolo 3

Architettura

In questo capitolo vengono presentati i principi sottostanti a SAFARI e gli strumenti che hanno permesso l'applicazione di tali principi.

3.1 Principi di SAFARI

L'obiettivo di SAFARI è l'analisi dei ransomware e del loro comportamento. L'esecuzione di un ransomware è un'operazione pericolosa, che può mettere a rischio il funzionamento di un elaboratore. Ciò comporta che per testarne il comportamento è necessario un ambiente isolato, che non abbia connessioni con l'esterno. Per configurare questa situazione, SAFARI fa uso della virtualizzazione e del concetto di air-gap. Come detto in sezione 2.1.2, la virtualizzazione permette di creare un livello di isolamento tra il sistema operativo ospite e l'hardware sottostante e anche tra i processi software in esecuzione sull'hypervisor. L'isolamento orizzontale, ovvero quello tra i vari processi, fa si che il malware in esecuzione su una macchina virtuale non vada ad infettare le altre. L'isolamento verticale, ovvero quello tra il sistema operativo e l'hardware, permette l'esecuzione del ransomware senza rischi per l'ambiente che coordina le macchine virtuali. L'air-gap, invece, fa si che il programma malevolo non si propaghi nella rete, rendendo la macchina isolata da un punto di vista della connettività.

L'uso di macchine virtuali, combinato con strumenti IaC e di automazione dei task, fa godere SAFARI della caratteristica della scalabilità, ovvero quell'abilità di un sistema di adattarsi alle variazioni del carico di lavoro. La scalabilità attraverso la virtualizzazione può essere raggiunta in due modi: aggiungendo o diminuendo il numero di macchine virtuali in esecuzione; o incrementando o decrementando le prestazioni dell'hardware delle macchine virtuali. SAFARI sfrutta la prima moda-

lità. Gli strumenti IaC e di automazione dei task, inoltre, permettono di gestire un'infrastruttura più o meno estesa, applicando piccole modifiche.

Tali strumenti, oltre a far raggiungere un alto grado di scalabilità, permettono a SAFARI di essere un framework automatizzato. L'infrastruttura come codice permette di definire, tramite linguaggi ad alto livello, l'infrastruttura che si vuole creare. Non è quindi necessario creare le macchine virtuali manualmente nell'ambiente di virtualizzazione. Strumenti come Ansible, invece, fanno si che determinati compiti, come l'esecuzione del ransomware, non richiedano l'intervento manuale ma, tramite codice, vengano eseguiti in maniera automatizzata.

Un'ultima caratteristica di SAFARI è la sua indipendenza dall'ambiente di virtualizzazione e dai sistemi operativi montati sulle macchine virtuali. L'indipendenza dalla piattaforma di virtualizzazione viene raggiunta grazie a Terraform, progettato per interfacciarsi con una moltitudine di piattaforme di virtualizzazione, sia on-premise che cloud. La possibilità di gestire macchine con sistemi operativi differenti deriva dalla caratteristica di Ansible di essere agent-less e non richiedere l'installazione di alcun componente software sui nodi controllati e di sfruttare sia il protocollo SSH che Windows Remote Management, utile per istruire macchine Windows.

3.2 Funzionamento ad Alto Livello

Illustrati i principi che guidano SAFARI, viene presentato il funzionamento ad alto livello.

I momenti salienti che caratterizzano il funzionamento di SAFARI sono:

1. Creazione di un template di macchina virtuale
2. Provisioning dell'infrastruttura a partire dal template
3. Trasferimento del ransomware nelle VM
4. Esecuzione dei comandi da remoto per avviare il ransomware
5. Spegnimento delle VM

SAFARI consente anche l'esecuzione di un programma di contrasto all'azione del ransomware. Oltre allo studio dei ransomware, SAFARI permette, quindi, di valutare l'efficacia di tali strumenti. La caratteristica della scalabilità assume maggiore importanza, permettendo di lanciare in parallelo più test con parametri, come ad esempio il tempo che intercorre tra l'avvio del ransomware e quello dello strumento di contrasto, diversi tra le varie esecuzioni.

3.2.1 Creazione del Template

Un template è un modello predefinito di macchina virtuale. Ha determinate caratteristiche hardware e software, ereditate dalle VM che vengono clonate da esso.

L'uso di template viene incoraggiato da Terraform in fase di provisioning dell'infrastruttura, a scapito della creazione di macchine virtuali a partire da immagini ISO o da altre macchine virtuali già in funzione.

Inoltre, l'uso di un template ci assicura che il contenuto delle VM sia sempre lo stesso prima del lancio del ransomware. Questa caratteristica ci permette di svolgere con più facilità l'analisi del disco e ci permette di fare confrontare i risultati ottenuti nelle varie esecuzioni dei test.

3.2.2 Provisioning dell'Infrastruttura

SAFARI sfrutta Terraform per il provisioning dell'infrastruttura.

In questa fase è necessario definire le risorse che si vogliono creare e il provider Terraform che si vuole utilizzare.

La definizione delle risorse deve essere coerente con le specifiche del template precedentemente generato. Quindi, devono essere riportate con esattezza le caratteristiche hardware e software del template.

La scelta del provider dipende dalla piattaforma di virtualizzazione in uso. Ci sono provider per quasi ogni piattaforma, sia on-premise che cloud.

Il provisioning che si vuole attuare è di tipo statico. Le risorse vengono allocate a priori e non è possibile modificarne la quantità o la qualità a run-time.

Terminata l'esecuzione di Terraform, si hanno a disposizione le macchine virtuali avviate, pronte a ricevere comandi da Ansible.

3.2.3 Automazione dei Task

Una volta avviate le macchine, con Ansible è possibile eseguire comandi da remoto in maniera totalmente automatizzata.

Per farlo, è necessario definire due tipi di file: l'inventario e i playbook. L'inventario è composto dagli indirizzi IP delle macchine su cui si vogliono eseguire determinati comandi. La definizione dell'inventario varia a seconda del fatto che alle VM siano stati assegnati indirizzi IP statici o dinamici. Nel capitolo seguente, si vedrà come definire l'inventario in caso di indirizzi IP dinamici. Un costrutto molto potente che

può essere introdotto nell'inventario è quello dei gruppi, ovvero insiemi di macchine etichettate da un nome.

Il playbook, invece, è il file YAML in cui si specificano i comandi da eseguire. Ansible mette a disposizione un grande numero di moduli, ovvero piccoli programmi che eseguono determinati compiti. Nei playbook è possibile istruire le macchine con comandi diversi, sfruttando i gruppi definiti nell'inventario. Ad esempio, si potrebbe lanciare un determinato ransomware su un gruppo di macchine e un altro su altre macchine.

Per quanto riguarda il playbook, i comandi da eseguire sono i seguenti:

1. Trasferimento nelle VM del ransomware
2. Disabilitazione delle schede di rete
3. Avvio del ransomware
4. Avvio dell'eventuale programma di contrasto
5. Spegnimento delle VM

Come detto nella sezione 2.4.1, Ansible comunica con gli host via SSH o Windows Remote Management, quindi sfruttando la rete. Nel momento in cui vengono disabilitate le schede di rete delle VM, Ansible non è più in grado di comunicare con esse. Per risolvere questo problema, vengono definiti due playbook. Uno esterno, si occupa di trasferire il ransomware nelle VM e di avviare il playbook interno, che, invece, disabilita le schede di rete, ed esegue il ransomware e l'eventuale strumento di contrasto. In ultima istanza, trascorso un determinato periodo di tempo, il playbook esterno, comunicando con la macchina su cui è in esecuzione l'hypervisor per la virtualizzazione, spegne le VM.

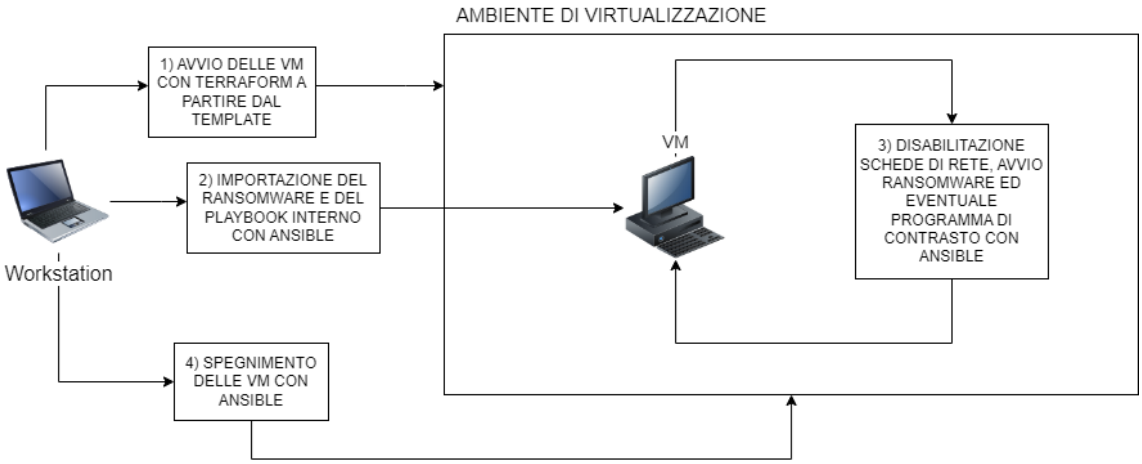

Figura 3.1: Architettura.

Capitolo 4

Utilizzo di SAFARI

In questo capitolo vengono riportati i passi da eseguire per utilizzare SAFARI.

4.1 Creazione della VM

Il primo passo è la creazione di un template in un ambiente di virtualizzazione. Per farlo è necessario prima di tutto scegliere un'immagine ISO di un sistema operativo. Come già detto, SAFARI è indipendente dal sistema operativo in uso nelle VM ed è quindi possibile sceglierne uno qualsiasi. Vedremo successivamente le eventuali operazioni da compiere in caso di sistemi operativi Windows.

Scelto il sistema operativo, tramite la piattaforma di virtualizzazione è necessario impostare le caratteristiche hardware della VM e montare su di essa l'immagine ISO scelta. L'ambiente di virtualizzazione può essere liberamente scelto. Nei prossimi paragrafi si vedranno i parametri da modificare in base alla piattaforma scelta.

4.2 Preparazione della VM

Creata la VM, è necessaria la sua preparazione. Il passaggio fondamentale di questa fase è la generazione di files che simulino l'uso da parte di un vero utente. Tale requisito deriva dalla capacità dei ransomware di comprendere l'ambiente in cui si trovano. Se il ransomware si rende conto di trovarsi in un ambiente creato ad hoc per la sua analisi, potrebbe non comportarsi come farebbe normalmente durante un vero attacco. Questa capacità nasce dalla volontà degli sviluppatori di malware di non permettere agli esperti di sicurezza di studiare il programma in laboratorio.

Per la generazione di file all'interno della macchina si possono seguire due approcci[BGM+23]:

- Uso della macchina per un certo periodo;
- Creazione artificiale dei file.

Nel caso in cui si voglia eseguire uno strumento di contrasto, è necessario caricare gli eseguibili sulla macchina.

Se per la macchina virtuale si è scelto un sistema operativo Windows, bisogna compiere delle operazioni aggiuntive, al fine di renderla compatibile con l'uso di Ansible. Seguendo la documentazione di Ansible[eA], per controllare una macchina Windows dobbiamo accertarci che in essa:

- sia installata una versione di Powershell maggiore della 5.1
- sia installata una versione del framework .NET maggiore della 4.0
- sia configurato il servizio Windows Remote Management (WinRM)

Per verificare la versione di Powerhsell, digitare in una finestra Powershell il comando

```
$PSVersionTable
```

e controllare la voce "PSVersion".

Per verificare la versione del framework .NET, è possibile usare il modulo Powershell "DotNetVersionLister"[JS23]. Per installare il modulo, digitare in una finestra Powerhsell il comando

```
Install-Module -Name DotNetVersionLister
```

e una volta installato, eseguire

```
Get-STDotNetVersion
```

Oltre alla configurazione necessaria per essere un host Ansible, si deve preparare la VM al fine di agire da controller Ansible. Un controller Ansible, tuttavia, deve essere un sistema Unix. Per soddisfare questo requisito, come consigliato dalla documentazione di Ansible, è possibile installare il sottosistema di Windows per Linux (WSL). Questa funzionalità simula un ambiente Linux in Windows. Per attivarlo, recarsi nel menu "Attiva o disattiva funzionalità di Windows", scorrere fino a "Sottosistema Windows per Linux" e spuntare la relativa casella. Fatto ciò, tramite il Microsoft Store, è necessario scaricare una distribuzione Linux. Finito il download, digitare nel prompt dei comandi "wsl". Tale comando fa partire l'installazione della distribuzione Linux, al termine della quale viene chiesto di impostare uno username e una password.

Installata la distribuzione Linux, è possibile installarvi Ansible. L'installazione può avvenire tramite il package manager "pip" o repository come "apt".

Un ultimo passaggio utile è l'installazione di un servizio SSH, se non già presente nella macchina. Il servizio SSH è utile per ottenere, tramite Terraform, gli indirizzi IP da riportare nell'inventario nel caso in cui non siano stati assegnati indirizzi statici alle VM.

Completati questi passaggi, è possibile convertire la macchina virtuale in un template.

4.3 Avvio delle VM

Creato il template, si passa alla fase di provisioning e di avvio delle macchine.

Il primo file da definire per usare Terraform è quello in cui si specificano la versione di Terraform e il provider che si vogliono usare e le credenziali con cui il provider può autenticarsi per accedere alle API della piattaforma di virtualizzazione scelta. Questo file deve avere un'estensione ".tf".

```
terraform {
    required_version = "versione_terraform"
    required_providers {
        piattaforma = {
            source = "nome_provider"
            version = "versione_provider"
        }
    }
}


provider "piattaforma" {

    pm_api_url = url_api
    pm_api_token_id = token_id
    pm_api_token_secret = token_secret

}
```

In questo caso, è stato usato come metodo di autenticazione un token API. Piattaforme di virtualizzazione diverse prevedono modalità diverse di autenticazione.

Definito tale file, è necessario eseguire da riga di comando il comando

```
terraform init
```

Questo comando installa il provider specificato e inizializza la directory di lavoro.

La seconda fase prevede la definizione delle VM che si vogliono creare. In un altro file, con estensione ".tf", in un blocco "resource", si devono specificare quante macchine virtuali si vogliono avviare e le loro caratteristiche. Creando tali macchine partendo da un template con specifiche hardware e software già definite, è necessario riportare esattamente questi valori.

A seconda della piattaforma di virtualizzazione e quindi del provider in uso, il numero e il nome dei parametri può cambiare. Nel prossimo capitolo, si vedrà come definire il blocco "resource" per lanciare delle VM in Proxmox.

All'interno del blocco "resource" è inoltre possibile definire dei provisioners. I provisioners sono costrutti che permettono di modellare un determinato comportamente sulle VM create o sulla macchina su cui viene eseguito Terraform. Esistono 3 tipi di provisioners:

1. file: permette di importare un file dalla macchina su cui è in esecuzione Terraform alla risorsa creata;
2. local-exec: permette di compiere un'azione sulla macchina su cui è in esecuzione Terraform;
3. remote-exec: permette di compiere un'azione sulla risorsa creata.

Se nel blocco "resource" si è impostato un indirizzo IP dinamico per le VM, è utile usare un provisioner di tipo local-exec per accedere a tali indirizzi , al fine di riportarli nell'inventario Ansible.

```
provisioner "local-exec" {
  command = "echo ${self.ssh_host} >> vm_ip_address.txt"
}
```

Questo provisioner stampa in un file "vm_ip_address.txt" la variabile "self.ssh_host". L'oggetto "self", utilizzato all'interno del blocco "resource", permette di accedere a determinate caratteristiche delle macchine virtuali che si vogliono creare. Il parametro "ssh_host" corrisponde all'indirizzo IP delle macchine. Il contenuto del file creato viene poi usato per aggiungere dinamicamente gli host all'inventario Ansible. Si può ora comprendere il motivo per il quale, in fase di preparazione della VM, è stato installato un servizio SSH.

Definite le risorse che si vogliono creare, con il comando

```
terraform plan
```

viene creato un piano di esecuzione. Nel piano di esecuzione vengono specificate le risorse che verranno create insieme alle loro caratteristiche.

Per applicare quanto riportato nel piano di esecuzione ed avviare le macchine virtuali, eseguire il comando

```
terraform apply
```

4.4 Esecuzione dei Task

Per l'esecuzione del ransomware e dell'eventuale programma di contrasto, come detto in sezione 3.2.3, è necessario definire due playbook: il primo, chiamato anche esterno, importa nelle VM i file necessari e avvia il secondo playbook, anche detto interno, che invece disabilita le schede di rete, avvia il ransomware e lo strumento di contrasto. Infine, il primo playbook fa si che le VM vengano spente. La definizione dei due playbook comporta la definizione di due inventari. Il primo inventario deve contenere gli indirizzi IP delle macchine virtuali e della macchina su cui è in esecuzione l'hypervisor. Il secondo contiene al suo interno l'indirizzo dell'interfaccia di loopback, cosicchè le macchine virtuali possano istruirsi da sole.

Nella definizione del primo inventario è utile l'uso di gruppi: è possibile inserire tutte le VM in un gruppo e la macchina che le coordina in un altro. Se si vogliono lanciare ransomware diversi nelle varie macchine virtuali o si vogliono specificare parametri diversi relativi allo strumento di contrasto, può risultare utile definire più gruppi, ognuno per uno specifico caso.

L'inventario viene definito in un file INI. Per definire i gruppi e le macchine che ne fanno parte si usa la seguente notazione:

```
[gruppo1]
192.168.1.x
192.168.1.y

[gruppo2]
192.168.1.u
192.168.1.z
```

Nel caso in cui, nella fase di provisioning, siano stati assegnati alle VM degli indirizzi statici, è possibile inserire direttamente nell'inventario tali indirizzi. Se invece sono stati assegnati indirizzi dinamici, questi possono essere aggiunti dinamicamente tramite il modulo "ansible.builtin.add_host", richiamabile nel primo playbook.

Oltre a specificare gli indirizzi IP delle macchine, è necessario specificare nell'inventario anche il protocollo di connessione, SSH o WinRM per Windows, e le credenziali tramite cui Ansible si può autenticare presso l'host. Per specificare queste variabili, nell'inventario si aggiungono le seguenti righe:

```
[all:vars]
ansible_user = admin
ansible_password = admin

[gruppo1:vars]
ansible_connection = ssh

[gruppo2:vars]
ansible_connection = winrm
ansible_port= 5985
ansible_winrm_transport= basic
```

Con questa configurazione, ad esempio, si sta specificando che tutti gli host hanno come username e password "admin". Inoltre, si specifica che Ansible deve comunicare via SSH con le macchine appartenenti al "gruppo1" e via WinRM con le macchine del "gruppo2".

Quando ci si connette via WinRM, oltre alle credenziali per l'autenticazione è necessario specificare anche altre variabili quali la porta su cui è in ascolto il listener di WinRM e il tipo di autenticazione. Nell'esempio precedente, la porta impostata è la 5985 e il tipo di autenticazione è il "basic". Se i task vengono eseguiti su macchine che si trovano all'interno di una rete non sicura è preferibile usare altri tipi di autenticazione, ad esempio il Kerberos.

Definiti gli inventari, si passa alla creazione dei playbook. I playbook sono definiti in formato YAML. Il primo playbook, nel caso in cui non siano stati assegnati indirizzi IP statici alle VM, si occupa, prima di tutto, di aggiungere gli indirizzi IP delle macchine virtuali all'inventario

```
- hosts: localhost
  tasks:

    - name: Aggiungi VM all'inventario
      ansible.builtin.add_host:
        name: "{{ item }}"
        groups: gruppoVM
      loop: "{{ new_ip }}"
```

Come si può notare, nei playbook, il primo aspetto da specificare sono gli host su cui vogliamo che determinate istruzioni siano eseguite. In questo caso, l'aggiunta degli indirizzi IP all'inventario è un compito che deve essere svolto sulla macchina su cui si esegue Ansible e per questo viene specificato come destinatario "localhost". Specificati gli host su cui eseguire i compiti, si passa alla sezione "tasks", al cui interno vengono definiti, tramite l'uso di moduli Ansible, le operazioni da svolgere. Per ogni task va definito un nome e il modulo che si vuole usare, con i relativi attributi. Tramite il modulo "ansible.builtin.add_host" è possibile specificare, sotto la voce "name", l'indirizzo che deve essere aggiunto all'inventario e, eventualmente, sotto la voce "groups" i gruppi a cui si vuole aggiungere l'indirizzo specificato. Nell'esempio riportato, in "name" è presente la variabile "item" e, dopo "groups", è presente una voce "loop" a cui viene assegnata la variabile "new_ip".

In Ansible è possibile definire il playbook e, in un altro file YAML, le variabili con i relativi valori che verranno usate nel playbook. In questo modo, se cambiano determinate caratteristiche dell'infrastruttura ma non le operazioni da eseguire, è possibile lasciare inalterato il playbook, andando a modificare solamente il file contenente le variabili.

Il costrutto "loop" permette di ripetere una determinata operazioni per tutti i valori presenti nella variabile che gli viene assegnata. La variabile "new_ip", infatti, è una lista, contenente tutti gli indirizzi IP delle VM.

```
    new_ip:
  - 192.168.1.x
  - 192.168.1.y
```

Questi valori sono quelli che Terraform ha stampato nel file "vm_ip_address.txt" e che devono essere riportati in "new_ip".

Per ogni valore della variabile "new_ip", Ansible richiamerà il modulo "ansible.builtin.add_host" e aggiungerà i valori all'inventario.

Aggiunte le VM all'inventario, è possibile eseguire operazioni su di esse.

```
- hosts: gruppoVM
  tasks:

    - name: Trasferisci ransomware compresso
      ansible.builtin.copy:
        src: "{{ local_working_directory }}{{ ransomware_name }}"
        dest: "{{ remote_user_directory }}{{ ransomware_name }}"
```

La prima operazione da compiere è il trasferimento del file zip contenente il ransomware. Con il modulo "ansible.builtin.copy" o, nel caso in cui gli host siano macchine Windows, con il modulo "ansible.windows.win_copy", è possibile specificare nella voce "src" il path del file presente nella macchina su cui è in esecuzione Ansible e nella voce "dest" il path di destinazione del file. I path sorgente e destinazione vengono per comodità spezzati in due in quanto la prima parte di ognuno dei due path sarà poi riutilizzata. Anche in questo caso, i path vengono specificati tramite l'uso di variabili. Si rende necessario quindi impostarne il valore nel file contenente tutte le variabili. La variabile "ransomware_name" contiene il nome della cartella compressa al cui interno è presente il ransomware. Il modulo Ansible che permette di copiare file dal controller agli host viene poi utilizzato anche per trasferire il secondo playbook, il secondo inventario e il file contenente le variabili.

L'ultima operazione da eseguire sulle VM è l'avvio del playbook appena trasferito. Per avviare un playbook Ansible, si digita in una shell il comando

```
ansible-playbook -i /path/inventario /path/playbook
```

a cui si può eventualmente aggiungere il parametro

```
--extra-vars "@variabili.yaml"
```

che permette di passare come argomento un file YAML contenente le variabili. L'operazione di avvio del secondo playbook sfrutta il modulo Ansible "ansible.builtin.command" o, in caso di hosts Windows, "ansible.windows.win_command".

```
- name: Avvia playbook interno
  ansible.builtin.command: ansible-playbook
  -i {{ remote_user_directory }}{{ internal_inventory_name }}
  {{ remote_user_directory }}{{ internal_playbook_name }}
  --extra-vars "@{{ remote_user_directory }}{{ ansible_variables_path }}"
  async: 3600
  poll: 0
```

Con l'opzione "-i" si specifica il path dell'inventario.

In questo caso, si introducono due nuovi costrutti: "async" e "poll". L'esecuzione delle operazioni in Ansible è di default sequenziale. Quindi, un determinato compito non viene svolto se quello precedente non è terminato. Il primo playbook, tuttavia, non ha modo di verificare l'esecuzione del secondo, in quanto la prima operazione svolta dal secondo playbook è la disabilitazione delle schede di rete. Da quel momento in poi, il primo playbook non ha più modo di verificare lo stato di avanzamento del secondo.

I costrutti “async” e “poll” permettono di passare da un’esecuzione sequenziale ad una asincrona e concorrente. Impostando il valore “poll” pari a zero, il playbook continuerà nella sua esecuzione, senza aspettare che l’operazione sia completata.

Le ultime operazioni svolte dal primo playbook hanno come host la macchina su cui è in esecuzione l’hypervisor.

```
- hosts: hypervisor
  tasks:

  - name: Attendi
    ansible.builtin.pause:
      seconds: "{{ attesa_spegnimento }}"

  - name: spegni vm
    ansible.builtin.command: qm shutdown {{ item }}
    loop: "{{ vm_id }}"
```

Tramite il modulo “ansible.builtin.pause” è possibile mettere in pausa l’esecuzione del playbook per un certo periodo di tempo. Questo periodo permette al secondo playbook di svolgere le sue operazioni e quindi al ransomware ed eventualmente allo strumento di contrasto di agire sulla VM.

L’ultima operazione spegne le VM. Il comando “qm shutdown” è specifico di Proxmox e va quindi modificato in base alla piattaforma di virtualizzazione utilizzata. Anche in questo caso, si fa uso del costrutto “loop” che permette di ciclare su tutti i valori di “vm_id”, una lista che contiene gli id delle VM lanciate.

Vengono analizzate ora le operazioni compiute dal secondo playbook, avviato dal primo.

```
- hosts: windows
  tasks:

  - name: Stacca scheda di rete
    community.windows.win_net_adapter_feature:
      interface: '*'
      state: disabled
      component_id:
        - ms_tcpip
        - ms_tcpip6
        - ms_implat
        - ms_lltdio
```

```
    - ms_rspndr
    - ms_lldp
    - ms_msclient
    - ms_pacer
```

La prima operazione del secondo playbook disabilita le schede di rete. Per sistemi Windows esiste un modulo Ansible specifico. Per altri tipi di sistemi, invece, è necessario usare il modulo "ansible.builtin.command" ed eseguire i comandi specifici del sistema operativo che disabilitano le schede di rete. Con questo comando si applica il concetto di air-gap e quindi di isolamento della macchina dalla rete.

I passi successivi riguardano l'estrazione del ransomware dalla cartella compressa e l'esecuzione dello stesso. Per sistemi Windows è possibile usare il modulo "community.windows.win_unzip", specificando il path dell'archivio, il path di destinazione, l'eventuale password che protegge l'archivio e la possibilità di eliminarlo una volta terminata l'estrazione. Per altri sistemi, è possibile usare il modulo "ansible.builtin.unarchive". Per Windows, si ha:

```
- name: Decomprimi ransomware
  community.windows.win_unzip:
    src: "{{ remote_user_directory }}{{ ransomware_name }}"
    dest: "{{ unzipped_ransomware_destination_path }}"
    password: pass
    delete_archive: true
```

Per altri sistemi:

```
- name: Decomprimi ransomware
  ansible.builtin.unarchive:
    src: "{{ remote_user_directory }}{{ ransomware_name }}"
    dest: "{{ unzipped_ransomware_destination_path }}"
    remote_src: true
  extra_opts:
  - "-P"
  - "pass"
```

Usando il modulo "ansible.builtin.unarchive", è possibile specificare la password usando l'argomento "extra_opts". Inoltre, è necessario specificare, tramite il parametro "remote_src", che la cartella compressa si trova già nella VM.

L'ultima operazione da svolgere è l'avvio del ransomware. Anche in questo caso si usa il modulo "ansible.builtin.command" o "ansible.windows.win_command" e si esegue il contenuto della cartella in cui è stato estratto il ransomware. Dato che

l'esecuzione del ransomware non ha una fine, si usano i costrutti "async" e "poll" precedentemente visti.

Se si volesse eseguire uno strumento di contrasto, come fatto nel Capitolo 5, è necessario inserire un'altra operazione che, tramite il modulo Ansible "ansible.builtin.command" o "ansible.windows.win_command", avvii l'eseguibile del programma.

Per avviare il primo playbook, quindi, recarsi nella directory in cui sono presenti i playbook e digitare in una shell il comando:

```
ansible-playbook -i /path/primo/inventario
/path/primo/playbook --extra-vars "@ansible_variables"
```

Nel caso in cui la macchina da cui si avvia Ansible sia fornita di un sistema Windows, anteporre a "ansible-playbook" il comando "wsl". Ciò farà si che Ansible sia avviato nel sottositema di Windows per Linux.

Capitolo 5

Caso di Studio

In questo capitolo, viene presentato un caso di studio in cui si utilizza Ranflood [BGM+24] per contrastare un ransomware, estraendo i dati di esecuzione tramite SAFARI.

5.1 Ranflood

L'obiettivo del caso di studio in questione è il testing di Ranflood. Ranflood è uno strumento che punta a contrastare l'azione dei ransomware, specialmente quelli della famiglia "crypto", che agiscono criptando il contenuto di una macchina, tramite la tecnica del data flooding. Per data flooding si intende il processo di generazione di un'enorme quantità di file in determinate aree della macchina. Tali files possono essere casuali o copie di files già presenti nel sistema, a seconda della strategia messa in atto. Ranflood prevede infatti 3 strategie:

1. random: vengono generati files casuali;
2. on-the-fly: i files generati sono copie di quelli già presenti nel sistema;
3. shadow: strategia simile alla on-the-fly ma basata sulla presenza di backup dei file dell'utente.

Durante la fase di testing, a causa delle prestazioni limitate della mia strumentazione, solo la strategia "random" viene analizzata. Le altre strategie richiedono infatti più risorse, in quanto basano il loro funzionamento sulla copia di files.

Ranflood è composto da due programmi: un programma demone, sempre attivo, e uno client, che fa richieste al demone.

Il testing prevede l'esecuzione di un ransomware e il successivo avvio di Ranflood. In particolare, si vuole studiare l'efficacia di Ranflood al variare del periodo che intercorre tra l'avvio del malware e quello del flooding.

5.2 Strumentazione Utilizzata

Per il caso di studio viene usata come piattaforma di virtualizzazione Proxmox. Proxmox è una distribuzione Linux basata su Debian. Fornisce un'interfaccia web per la gestione delle VM e le API REST, utili per connetervi strumenti di terze parti. La scelta di Proxmox deriva da due fattori. Il primo è la volontà di usare strumenti open-source. Il secondo risiede nella possibilità di avere supporto online. Navigando in vari forum online, ho notato che, tra gli strumenti open-source per la virtualizzazione, Proxmomx è uno dei più discussi.

Proxmox è stato installato su una macchina con le seguenti caratteristiche hardware:

- processore Intel i5 di settima generazione a 3 GhZ;
- memoria RAM da 8 GB;
- SSD da 256 GB;
- HDD da 256 GB.

La macchina virtuale utilizzata come template è una macchina Windows 10 che già viene usata per il testing di Ranflood. Al suo interno sono presenti numerosi files di varia natura e gli eseguibili Ranflood.

Il ransomware scelto è il WannaCry. WannaCry fu responsabile di un attacco informatico su larga scala nel 2017. Il ransomware attacca sistemi Windows criptandone il contenuto e richiedendo all'utente un riscatto per il loro sblocco.

5.3 Uso di Terraform

Dopo aver svolto le operazioni preliminari presentate in sezione 4.2, è necessario scegliere il provider Terraform compatibile con Proxmox e successivamente definire le risorse che si vogliono creare. Uno dei provider più usati per Proxmox è Telmate [tel24]. Il primo passo è definire il file "provider.tf", al cui interno vengono definite la versione di Terraform e il provider scelto e le credenziali per l'autenticazione del provider presso Proxmox, al fine di accedere alle API.

```
terraform {
    required_version = ">= 1.6.1"
```

```
    required_providers {
        proxmox = {
            source = "telmate/proxmox"
            version = "2.9.14"
        }
    }
}


provider "proxmox" {

    pm_api_url = var.proxmox_api_url
    pm_api_token_id = var.proxmox_api_token_id
    pm_api_token_secret = var.proxmox_api_token_secret

}
```

Per l'autenticazione del provider uso il token API. Questo è generabile tramite l'interfaccia Web di Proxmox nel menu "Datacenter". Per farlo è necessario specificare l'utente Proxmox per il quale si vuole generare il token, il nome che si vuole assegnare al token, la scadenza ed un eventuale commento. Indicati tali parametri, Proxmox mostra il valore segreto del token.

Come si può notare dal codice sopra riportato, nei file di Terraform inerenti il provider e le risorse non ho riportato direttamente i valori, che invece vengono indicati in un altro file con estensione ".auto.tfvars".

Per la definizione delle risorse è necessario seguire la documentazione di Telmate, in quanto i parametri delle VM che è possibile specificare dipendono dal provider in uso.

```
resource "proxmox_vm_qemu" "windows_clone" {

    count = var.vms-count

    name = "${var.vm_name}-${count.index}"
    target_node = var.target_node
    vmid = sum([var.vm_id, count.index])
    clone = var.template_clone
    bios = var.vm_bios
    ipconfig0 = var.vm_ip
    full_clone = true
```

```
    sockets = var.vm_sockets
    cores = var.vm_cores
    memory = var.vm_memory

    scsihw = var.vm_scsi

    disk {
        type = var.vm_disk_type
        storage = var.vm_disk_storage
        size = var.vm_disk_size
    }

    network {
        model = var.network_card_model
        bridge = var.network_bridge
    }

    provisioner "local-exec" {
      command = "echo ${self.ssh_host} >> vm_ip_address.txt"
    }
}
```

Per creare macchine virtuali in Proxmox, bisogna impostare il tipo di risorsa a "proxmox_vm_qemu". In questo caso, "windows_clone" è il nome che viene assegnato alla risorsa.

Tramite il parametro "count" è possibile specificare quante istanze si voglio creare. Con "name" si imposta il nome che si vuole assegnare alla VM in Proxmox. Impostando il nome, ho concatenato alla variabile "var.vm_name", il valore "count.index", così da evitare problemi dovuti alla creazione di macchine virtuali con lo stesso nome. Il valore "count.index" assume i valori nell'intervallo [0,count-1].

Il parametro "target_node" si riferisce al nome del nodo Proxmox su cui si vogliono creare le risorse. Con "vm_id" si indica l'id che si vuole assegnare alla VM. Per assicurarmi che, nel caso in cui vengano create piu macchine, non ci siano VM con lo stesso id, ho usato la funzione "sum" i cui addendi sono "var.vm_id", a cui è stato assegnato un valore di base e "count.index".

Con "clone" si specifica il nome del template che si vuole clonare per creare le VM. La voce "bios" permette di indicare il tipo di BIOS che si vuole montare sulla macchina. Seguendo quanto specificato dalla VM Windows, ho indicato il valore

"SeaBIOS". Con "ipconfig" si può specificare se alle macchine si vogliono assegnare indirizzi statici o dinamici. Se alla variabile "var.vm_ip" viene assegnato un valore pari a "ip=dhcp", allora sarà compito del server DHCP attribuire gli indirizzi IP alle VM. Inserendo un valore specifico, alla macchina sarà assegnato tale valore. Dal momento in cui nella mia rete domestica non ho un insieme di indirizzi IP a priori liberi, ho preferito la via degli indirizzi dinamici.

Il parametro "full_clone" permette di scegliere tra un clone completo e un clone linkato. Quando si crea una VM a partire da un template in Proxmox, viene data la possibilità di scegliere tra queste due opzioni. Un clone completo è una copia esatta del template ed è indipendente da esso. Tuttavia, richiede maggior tempo per la creazione e maggior spazio in memoria. Un clone linkato, invece, non può funzionare senza l'accesso al template di partenza[VE18]. Per ragioni di sicurezza e isolamento, è consigliabile creare un clone completo.

Si passa poi alla definizione del numero di sockets e di cores, si specifica la quantità di memoria RAM da assegnare alle VM e il tipo di controller SCSI.

Si definiscono poi il tipo di dispositivo di archiviazione, la sua dimensione e in quale volume Proxmox deve essere memorizzato. Come tipo di disco è possibile scegliere tra "ide", "scsi", "virtio" e "sata". In questo caso ho scelto "ide", come definito dalla VM. Per il parametro "model" ho impostato il valore "e1000" mentre per "bridge" il valore "vmbr0", il bridge di rete predefinito di Proxmox.

Infine, è stato definito il provisioner "local-exec", utile per conoscere gli indirizzi IP delle macchine create che vengono poi usati per l'inventario Ansible.

5.4 Uso di Ansible

Avviate le VM, si passa all'esecuzione delle operazioni su di esse. Oltre ai compiti da svolgere sulle macchine presentati in sezione 4.4, si vuole illustrare come, tramite Ansible, è possibile parametrizzare il funzionamento di Ranflood.

L'inventario relativo al primo playbook Ansible si presenta così

```
[windows]


[windows:vars]
ansible_user=username
ansible_password=password
ansible_port=5985
ansible_connection=winrm
```

```
ansible_winrm_transport=basic
ansible_winrm_server_cert_validation=ignore

[proxmox]
192.168.1.x

[proxmox:vars]
ansible_user=proxmox_user
ansible_password=proxmox_password
```

Il gruppo “windows” appare vuoto, in quanto viene riempito a run-time. Tuttavia, vengono già impostate le sue variabili. Trattandosi di macchine derivanti da un template Windows, si usa come protocollo di comunicazione WinRM.

Il gruppo “proxmox” fa invece riferimento alla macchina su cui è in esecuzione l’hypervisor. In fase di installazione di Proxmox, viene assegnato alla macchina un indirizzo IP statico, che non cambia nel tempo. Per questo motivo, viene già indicato l’indirizzo IP. Trattandosi di un sistema Debian, è possibile collegarsi via SSH. Per l’autenticazione vengono usati username e password del nodo.

Data la mia strumentazione, non ho possibilità di lanciare più macchine virtuali contemporaneamente e, di conseguenza, posso eseguire solo un esperimento alla volta. Per questo motivo definisco un solo gruppo di VM, il gruppo “windows”. Avendo avuto una macchina su cui è in esecuzione l’hypervisor più potente, sarebbe stato possibile lanciare più macchine nello stesso momento in cui, ad esempio, su alcune veniva eseguito Ranflood in modalità shadow e altre Ranflood in modalità random.

Per quanto riguarda il primo playbook, il primo passo da compiere è l’aggiunta dell’indirizzo IP della VM all’inventario

```
- hosts: localhost
  tasks:

    - name: Aggiungi VM all'inventario
      ansible.builtin.add_host:
        name: "{{ item }}"
        groups: windows
      loop: "{{ new_ip }}"
```

Nel file “ansible_variables.yml”, dove sono definite tutte le variabili usate nei playbook, viene definita la variabile “new_ip”, una lista di indirizzi IP. Nel mio specifico

caso, la lista è formata da un solo indirizzo. Tale indirizzo viene copiato e incollato dal file "vm_ip_address.txt" generato dal provisioner Terraform.

Fatto ciò, tramite il modulo "ansible.windows.win_copy", vengono trasferiti nella VM la cartella compressa contenente il ransomware, il secondo playbook, il secondo inventario e il file contenente le variabili in formato YAML.

```
- hosts: windows
  tasks:

    - name: Trasferisci ransomware zippato
      ansible.windows.win_copy:
        src: "{{ local_working_directory }}{{ ransomware_name }}"
        dest: "{{ remote_windows_user_directory }}{{ ransomware_name }}"

    - name: Trasferisci playbook interno
      ansible.windows.win_copy:
        src: "{{ local_working_directory }}{{ internal_playbook_name }}"
        dest: "{{ remote_windows_user_directory }}{{ internal_playbook_name }}"

    - name: Trasferisci inventario Ansible
      ansible.windows.win_copy:
        src: "{{ local_working_directory }}{{ internal_inventory_name }}"
        dest: "{{ remote_windows_user_directory }}{{ internal_inventory_name }}"

    - name: Trasferisci file variabili ansible
      ansible.windows.win_copy:
        src: "{{ local_working_directory }}{{ ansible_variables_path }}"
        dest: "{{ remote_windows_user_directory }}{{ ansible_variables_path }}"
```

La variabile "remote_windows_user_directory" assume il valore "C:\Users", mentre "local_working_directory" fa riferimento alla cartella della macchina da cui viene avviato Ansible in cui sono presenti i playbook. L'ultima operazione da compiere sul gruppo "windows" è l'avvio del playbook appena trasferito.

```
      - name: Avvia playbook interno
    ansible.windows.win_command: wsl ansible-playbook
    -i {{ remote_linux_user_directory }}{{ internal_inventory_name }}
    {{ remote_linux_user_directory }}{{ internal_playbook_name }}
    --extra-vars
    "@{{ remote_linux_user_directory }}{{ ansible_variables_path }}"
    async: 3600
```

```
    poll: 0
```

La VM monta un sistema Windows ma con l'esecuzione di questo comando diventa un controller Ansible. Per questo motivo, il comando "ansible-playbook" viene preceduto dal comando "wsl", in modo tale che il playbook venga avviato all'interno del WSL. Inoltre, non si fa più riferimento alla variabile "remote_windows_user_directory" ma alla variabile "remote_linux_user_directory", che assume il valore "/mnt/c/Users", che corrisponde alla cartella "Users" di Windows ma acceduta dal WSL.

Le ultime operazioni del primo playbook vengono eseguite sul nodo Proxmox.

```
- hosts: proxmox
  tasks:

  - name: Attendi
    ansible.builtin.pause:
      seconds: "{{ attesa_spegnimento }}"

  - name: spegni vm
    ansible.builtin.command: qm shutdown {{ item }}
    loop: "{{ vm_id }}"
```

Con il modulo "ansible.builtin.pause" si blocca l'esecuzione del playbook per un determinato periodo di tempo, al fine di permettere l'esecuzione del ransomware e di Ranflood. In questo caso, tale periodo viene fissato a 300 secondi. Trascorso questo periodo di tempo, viene spenta la VM. La variabile "vm_id" è una lista, i cui valori devono corrispondere agli id che si sono assegnati alle VM tramite Terraform.

Il playbook interno, come illustrato in sezione 4.4, si occupa di creare l'air-gap, estrarre il ransomware dalla cartella compressa ed avviarlo. Viene presentato ora come avviare Ranflood e come parametrizzarne il comportamento. Prima di avviare il flooding, è necessario avviare il demone Ranflood.

```
  - name: Avvia ranflood daemon
    ansible.windows.win_command:
      argv:
        - "{{ path_demone_programma_contrasto }}"
        - "{{ path_settings_ini }}"
    async: 1800
    poll: 0
```

La variabile "path_demone_programma_contrasto" assume come valore il path dell'eseguibile del demone Ranflood. La variabile "path_settins_ini" fa riferimento al

file “settings.ini”. Per avviare il demone Ranflood, è necessario passare come parametro un file al cui interno si specificano alcune informazioni tra cui l’indirizzo e la porta su cui il demone è in ascolto per le richieste del client. Il costrutto “argv” del modulo “ansible.windows.win_command” permette di concatenare gli elementi che vengono specificati al suo interno.

Nel caso in cui si voglia avviare Ranflood in modalità shadow o on-the-fly, è necessario, prima del flooding, acquisire uno snapshot. Queste due modalità sono basate su copia e hanno quindi bisogno di sapere quali sono i file dell’utente, così da evitare di copiare i file generati dal ransomware. Per farlo, si utilizza uno snapshot, ovvero una lista di file contenuti in una certa directory con il loro relativo checksum. In fase di flooding, Ranflood copierà solo i file il cui checksum corrisponde con quello salvato nello snapshot.

```
- name: Snapshot ranflood
  ansible.windows.win_command:
    argv:
      - "{{ path_programma_contrasto }}"
      - snapshot
      - take
      - "{{ strategia_programma_contrasto }}"
      - "{{ directory_target_programma_constrasto }}"
  when: "'on-the-fly' in strategia_programma_constrasto
  or 'shadow_copy' in strategia_programma_constrasto"
```

Per effettuare uno snapshot si usa il comando “snapshot take” di Ranflood, seguito dalla strategia, shadow o on-the-fly, e la directory sulla quale si vuole eseguire lo snapshot. In questo caso, viene usato il costrutto “when” di Ansible, che permette di eseguire una determinata operazione solo se la variabile “strategia_programma_-contrasto” assume il valore shadow o on-the-fly. Se la variabile assume il valore random, questa operazione non viene presa in considerazione da Ansible.

Dopo aver estratto e avviato il ransomware, si introduce un’attesa prima di eseguire Ranflood. Il caso di studio, infatti, vuole valutare l’efficacia di Ranflood al variare del tempo intercorso tra l’avvio del ransomware e di quest ultimo.

```
- name: Aspetta ranflood client
  ansible.builtin.pause:
    seconds: "{{ attesa_programma_contrasto }}"
```

Nello specifico, si vuole valutare l’efficacia di Ranflood avviandolo dopo 15, 30 o 60 secondi dall’avvio del ransomware.

L’ultima operazione svolta, riguarda l’avvio del flooding.

```
- name: Avvia ranflood client
  ansible.windows.win_command:
    argv:
      - "{{ path_programma_contrasto }}"
      - flood
      - start
      - "{{ strategia_programma_contrasto }}"
      - "{{ directory_target_programma_contrasto }}"
```

L'inventario del secondo playbook appare così:

```
[windows]
127.0.0.1

[windows:vars]
ansible_user=username
ansible_password=password
ansible_port=5985
ansible_connection=winrm
ansible_winrm_transport=basic
ansible_winrm_server_cert_validation=ignore
```

L'host è rappresentato dall'interfaccia di loopback, in quanto la macchina deve istruirsi da sola. La comunicazione deve comunque avvenire via WinRM, dato che Ansible viene avviato nel WSL, ma le operazioni sono da svolgere in ambiente Windows.

5.5 Analisi dei Risultati

5.5.1 Filechecker e Profiler

Eseguiti i test, si passa alla fase di analisi dei risultati. Per conoscere l'efficacia di Ranflood e le differenze tra le varie esecuzioni date dai differenti ritardi di attivazione viene usato uno strumento sviluppato dal team di Ranflood, Filechecker. Questo tool si basa sul confronto dei checksum dei files presenti nella macchina prima e dopo l'esecuzione del ransomware e di Ranflood.

Tramite il comando "save" del Filechecker, passando come parametro il path di una directory, viene generata una lista di checksum dei files presenti in tale directory. Una volta eseguito il ransomware e Ranflood, con il comando "check", a cui si passano come argomenti il file creato in precedenza con il comando "save" e il path

della stessa directory, viene generato un file contenente la lista dei checksum e dei path dei files che non sono stati intaccati dal ransomware.

Dopo aver generato il file contenente i checksum, un altro programma, il Profiler, genera un file JSON con la struttura ad albero delle cartelle e dei files. Per ogni file, il Profiler attribuisce uno stato:

- pristine: se il file non è stato toccato dal malware;
- replica: se il file è stato criptato dal virus, ma abbiamo una copia effettuata da Ranflood (nel caso in cui si siano scelte le modalità shadow o on-the-fly);
- lost: se il file è stato criptato dal virus e non abbiamo una replica.

Per ogni file è inoltre presente una chiave "replicas" contenente i path delle repliche generate da Ranflood.

Successivamente, uno script prende in pasto il JSON generato dal Profiler e conta il numero di files distribuiti per stato.

Eseguendo un certo numero di test con gli stessi parametri (strategia e ritardo di attivazione di Ranflood), viene poi calcolata la media del numero di files che si trovano in un determinato stato. Infine, a partire da queste medie, uno script Python genera i grafici.

5.5.2 Generazione dei checksum

Per generare i checksum viene utilizzata una VM Ubuntu. Il ransomware WannaCry, infatti, è stato progettato per essere eseguito in Windows e, quindi, montare un disco infettato in un sistema Linux risulta essere meno pericoloso.

Per montare un disco appartenente ad una VM in un'altra, in Proxmox, è necessario, tramite interfaccia Web, recarsi nel menu "Hardware" della VM da cui si vuole estrarre il disco. Cliccando poi sul disco d'interesse, "Azioni Disco" e "Riassegna proprietario", è possibile selezionare la VM sulla quale montare tale dispositivo. Fatto ciò, dopo aver avviato la VM su cui è stato montato il disco, con il comando

```
sudo mount /dev/nome_del_disco /mnt/nome_cartella
```

è possibile accedere al contenuto del file system nella directory "nome_cartella".

Per prima cosa, viene montata sulla macchina virtuale Ubuntu un disco di una VM Windows su cui non è stato eseguito né il ransomware né Ranflood. Con il comando

```
java -jar filechecker.jar save /path/checksum /path/da/analizzare
```

viene generato il file “checksum” contenente i path e i checksum dei files contenuti in “/path/da/analizzare”. Nel caso di studio ci si è concetrati sulla directory “C:\Users\IEUser\”, ovvero la cartella dell’utente della VM Windows.

Successivamente, vengono lanciate le VM su cui si eseguono il ransomware e Ranflood e, montati i relativi dischi sulla macchina Ubuntu, si esegue il comando

```
java -jar filechecker.jar check
/path/checksum /path/report /path/da/analizzare
```

che genera il file “report”, contenente i path e i checksum dei files non criptati.

5.5.3 Risultati ottenuti

I risultati ottenuti dai test differiscono notevolmente da quelli riscontatri dai test effettuatu in passato[BGM+23]. Ciò, molto probabilmente, è dovuto alla differenza nel tempo che il ransomware ha avuto a disposizione per criptare i file e dal diverso hardware su cui sono stati fatti i test. Ad esempio, negli studi effettuati, la macchina restava in esecuzione per 10 minuti mentre per il caso di studio dopo 5 minuti la macchina viene spenta.

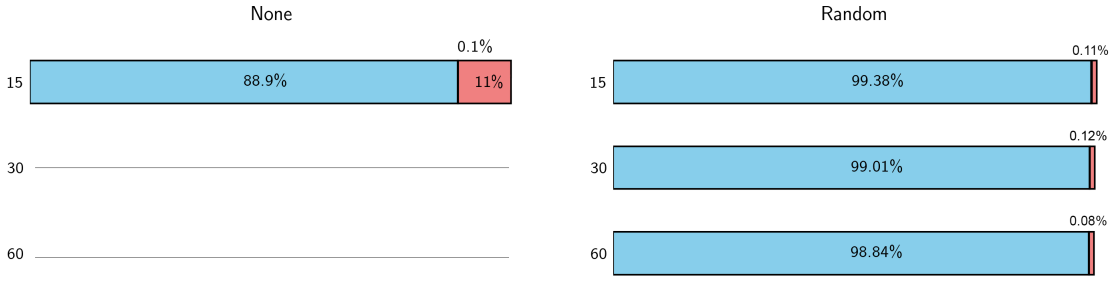

Figura 5.1: Risultati ottenuti

La colonna “None” mostra il comportamento del ransomware senza che Ranflood sia stato eseguito. La barra blu rappresenta i file che non sono stati criptati mentre quella rossa i file andati persi.

La colonna “Random”, invece, mostra i risultati ottenuti eseguendo Ranflood in modalità random. Si può notare come, per tempi di attivazione più brevi ci sia un minor numero di file criptati.

Le percentuali riportate sopra le barre rappresentano i file recuperati tramite copie. Ciò appare strano in quanto la modalità random non si basa sull’uso di copie ma sulla generazione di file casuali. Tuttavia, la ragione può essere ricercata nel funzionamento del WannaCry. Prima di criptare un file, tale ransomware potrebbe creare

una copia in cartelle note, come ad esempio “Documents”, lo cripta e poi cancella l’originale. Le copie trovate potrebbero essere quelle “rimaste in transito” in fase di spegnimento della macchina. Questo conferma l’ipotesi che SAFARI può essere utilizzato anche solo per analizzare il comportamento dei ransomware.

Nonostante la differenza con gli studi precendentemente condotti, il comportamento del ransomware e di Ranflood corrisponde alle aspettative.

Capitolo 6

Conclusioni e Possibili Sviluppi

In questa tesi è stato presentato SAFARI, uno strumento in grado di aiutare gli esperti di sicurezza nello studio dei ransomware e dei relativi attacchi. Il suo scopo è permettere di condurre studi sicuri, tramite l'isolamento delle macchine e più veloci, grazie a caratteristiche quali la scalabilità e l'automazione. SAFARI non vuole essere un semplice insieme di scripts che combinati svolgono delle operazioni, ma un design che può essere adattato in base alle specifiche esigenze. La sua importanza non risiede tanto nell'implementazione degli script, ma nei principi che hanno guidato la loro creazione.

Guardando a sviluppi futuri, uno di questi è l'integrazione con il programma Filechecker, così da avere un un'unica filiera completamente automatizza che parte dal provisioning e arriva all'analisi dei risultati.

Inoltre, potrebbe essere automatizzata anche la fase di creazione del template della macchina virtuale da clonare. Strumenti come Packer possono essere utili per raggiungere questo scopo. Packer consente di automatizzare il processo di creazione di immagini di macchine virtuali. Tramite linguaggio HCL, lo stesso usato da Terraform, si possono definire caratteristiche delle VM, eliminando il processo di preparazione delle macchine.

Uno sviluppo interessante potrebbe essere quello della creazione di una rete di VM tra loro interconnesse. L'obiettivo sarebbe quello di valutare l'impatto del ransomware "in movimento". Si potrebbe simulare l'invio di un file ritenuto sicuro, ma in realtà pericoloso, da un nodo all'altro. Si passerebbe, quindi, da un computer air-gapped ad una rete air-gapped.

Riferimenti bibliografici

[BGM+23] Davide Berardi, Saverio Giallorenzo, Andrea Melis, Simone Melloni, Loris Onori, and Marco Prandini. Data flooding against ransomware: Concepts and implementations. *Computers Security*, page 103295, 2023.

[BGM+24] Davide Berardi, Saverio Giallorenzo, Andrea Melis, Simone Melloni, and Marco Prandini. Ranflood: A mitigation tool based on the principles of data flooding against ransomware. *SoftwareX*, 25:101605, 2024.

[Bro23] Massimiliano Brolli. Cos'è una rete air gap?, 2023.

[eA] Red Hat e Ansible. Ansible documentation.

[eA12] Red Hat e Ansible. Ansible, 2012.

[Has14] HashiCorp. Terraform, 2014.

[JS23] JensGJ Joakim Svendsen, Jonathan Moore. Dotnetversionlister, 2023.

[PZH13] Michael Pearce, Sherali Zeadally, and Ray Hunt. Virtualization: Issues, security threats, and solutions. *ACM Comput. Surv.*, 45(2), mar 2013.

[tel24] terraform-provider-proxmox, 2024.

[VE18] Proxmox VE. Vm templates and clones — proxmox ve,, 2018. [Online; accessed 25-February-2024].

[Wik23] Wikipedia. Virtualizzazione — wikipedia, l'enciclopedia libera, 2023. [Online; in data 20-febbraio-2024].

[Wik24] Wikipedia. Ransomware — wikipedia, l'enciclopedia libera, 2024. [Online; in data 29-febbraio-2024].

Ringraziamenti

Ringrazio il Professor Giallorenzo e il Dottor Melloni per avermi seguito nei periodi di tirocinio e stesura della tesi con grande disponibilità e per aver fatto sì che SAFARI entrasse a far parte del progetto “Flooding-against-Ransomware”. Spero possa essere utile alla causa.

Ringrazio i miei genitori, per avermi permesso di studiare in questa città, che molto mi ha fatto crescere.

Ringrazio Monica, la mia cura.

Ringrazio i miei amici di sempre, in quanto mie radici.

Ringrazio Rocchino e Stefano, veri amici e consiglieri.

Ringrazio tutti i miei coinquilini, vecchi e nuovi, per avermi fatto ridere, fatto piangere, e insegnato a parlare dei problemi, invece di tenerli dentro.