Università degli Studi di Perugia
Dipartimento di Matematica e Informatica

A.D. 1308 unipg
DIPARTIMENTO
DI MATEMATICA E INFORMATICA

Tesi magistrale in Informatica

Sviluppo di un Covert Channel tramite il protocollo ICMP per l'esfiltrazione di dati

Relatore
Prof. Santini Franceso

Laureando
Mecarelli Marco

Anno Accademico 2024-2025

Indice

Listings

Elenco delle figure

1 Prima Parte

ICMP (Internet Control Message Protocol) è un protocollo che opera al livello di rete (livello 3 nel modello ISO/OSI). e permette la **segnalazione errori**, la **diagnostica di rete** e la **messaggistica di controllo**. Proprio per questo che viene utilizzato per il monitoraggio dello stato di una rete e per la risoluzione dei problemi che avvengono in essa. Data la sua necessita per la diagnostica di rete e la segnalazione degli errori, può essere utilizzato in modo improprio per mettere a segno degli attacchi o per studiare la rete (ricognizione della rete).

Nei seguenti capitoli verrà illustrato come può essere sfruttato per la creazione di un Covert Channel; un attacco che permette (in ambienti ritenuti sicuri) la capacità di comunicare e/o trasferire dati in maniera non autorizzata e non voluta. L'attacco opera al di fuori degli usuali meccanismi di comunicazioni e per questo risulta difficile da rilevare e/o identificare. Sia dagli amministratori che dai tipici strumenti di monitoraggio. Infine, siccome qualsiasi risorsa condivisa può essere utilizzata per la sua creazione, può esistere in qualunque sistema.

1.1 Strumenti Utilizzati

Virtual Box

Il codice sviluppato è stato testato in un ambiente Linux. Per poter far ciò sono state create, per ciascun entità necessaria, una macchina virtuale contenente Ubuntu. Alla fine si sono ottenute quattro macchine virtuali: una per l'attaccante e la vittima, mentre le altre due per i proxy. Si poteva usare anche un singolo proxy ma si voleva testare anche come i dati ricavati dalla vittima, e da inoltrare all'attaccante, venissero distribuiti ai proxy connessi a essa.

Scapy

Scapy è un framework per la manipolazione dei pacchetti scritto in Python che consente di falsificare molti tipi di pacchetti (http, tcp, ip, udp, icmp, ecc.) È in grado di creare o decodificare pacchetti di vari protocolli. Inoltre può inviarli in rete, catturarli, memorizzarli o leggerne i dati.

Svolge principalmente due funzioni: invia i pacchetti e riceve le risposte, consentendo all'utente di inviare, intercettare, analizzare e falsificare pacchetti di rete. Questa capacità consente la creazione di strumenti in grado di sondare, scansionare o attaccare le reti.

Nella libreria il meotdo **send** (o similari) permetteranno di inviare un definito pacchetto. Per definire un pacchetto basterà concatenare i livelli che dovranno essere presenti, e opportunatamente inizializzati; mentre per poter ascoltare il traffico di ree basterà una variabile di tipo *AsyncSniffer*.

RITA

RITA (Real Intelligence Threat Analytics) è uno strumento open source per la ricerca delle minacce di rete, progettato per identificare attività di comando e controllo (C2) dannose. Acquisisce i log di Zeek e utilizza l'analisi comportamentale per identificare sistemi potenzialmente compromessi. Per l'installazione si è seguita la seguente guida. Siccome si utilizza un computer Windows, i comandi sono stati esguiti tramite WSL (Windows Subsystem for Linux).

Le funzionalità principali sono: il rilevamento dei Beacon, rilevamento del tunneling DNS, rilevameno di connessioni che hanno comunicato per tempi lunghi, controllo dei feed per le Threat Intel (domini e host sospetti), valutazione per gravità delle connessioni, quanti host hanno comunicato con un determinato host, il primo incontro di un host,

ICMP Door

È stato studiato per compredere come potesse effettuare il tunneling dei dati. Oltre alla struttura delle entità, che successivamente verrà ridefinita, risulta interessante come il programma richiede degli argomenti dall'utente. Tramite la libreria *argparse* richiede all'utente l'interfaccia su cui ascoltare i dati e l'indirizzo di destinazione dei pacchetti. Inoltre usa il metodo *sniff* del framework Scapy per ascoltare il flusso dei dati mentre tramite *sr* invia i paccheti.

Sebbe il programma ci introduce a una possibile struttura del Covert Channel; gli si sono trovati dei difetti. Gli svantaggi sono che i dati vengono trasemssi non solo

nel campo data, del messaggio di tipologia ICMP Echo Reply, ma anche in chiaro. Inoltre il valore del campo identifier rimano invariato per tutta la sessione. Un sistema di sicurezza, se vedesse le molteplici risposte (che non combaciano con il numero di richieste) e leggesse i testi in chiaro, potrebbe identificare il canale nascosto. Di solito per ogni Echo Request corrisponde una singola Echo Reply in cui la risposta rimanda i dati ricevuti e il campo data di solito contiene frasi gia preimpostate e sempre costanti (e.g *'helloworld'*).

ICMP Exfil

Analizzato per vedere come un Covert Channel temporalizzato postesse funzionare. Riceve un dato, lo converte in binario e dopodichè avrà una lista di numeri binari. Per mandare il dato, invia un ping con un timeout pari a **binary_number+leway**. Il valore leway viene utilizzato per rallentare il numero di pacchetti inviati ed avere una connesisone maggiormente silenziosa. L'autore infine indica come miglioria, la crittografia dei dati per aggiungere del rumore, dell'entropia.

Ciò su cui si affida è che un osservatore, vedendo i pacchetti ICMP, li veda come vailidi; iccome non riuscirebbero a trovare alcun dato, a meno che non sappiano della tecnica utilizzata.

ICMP Tunnel

Strumento che permette il tunneling del traffico IP. Tramite delle richieste e risposte ICMP Echo, incapsula il traffico e lo invia al server proxy. Questi ultimo lo decapsulano e lo inoltrano. I pacchetti in entrata, che sarebbero diretti alla macchina vittima, sono poi incapsulati dal proxy e inviati. L'approccio è possibile siccome RFC-792, che indica le linee guida del protocollo ICMP, permette una quantità arbitraria di dati nei pacchetti ICMP Echo (sia richiesta che risposta).

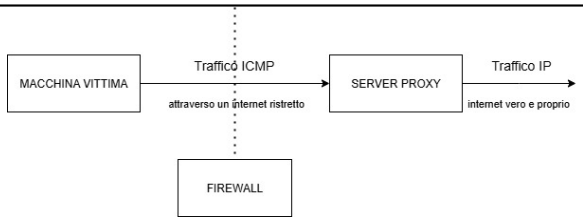


Figura 1: Architettura generale *icmp tunnel*

2 Seconda Parte

Un **Covert Channel** è un attacco che permette (in ambienti ritenuti sicuri) la capacità di comunicare e/o trasferire dati in maniera non autorizzata e non voluta. Solitamente operano al di fuori degli usuali meccanismi di comunicazioni sfruttando vulnerabilità o comportamenti non previsti nei sistemi. Ciò gli permette di non generare segnali di un uso improprio del sistema ed inoltre, nascondendosi all'interno dei normali processi del sistema, sono difficili da rilevare e/o identificare.

Qualsiasi risorsa condivisa può essere utilizzata come canale nascosto. Questo permetterà ai Covert Channel di esistere in qualsiasi sistema. E per questo la loro esistenza rappresenta un problema che spesso rimane non notato.

Un Covert Channel possiede determinate caratterisitche. L'**indistinguibilità** è la principale; è estremamente importante riuscire a trasmettere informazioni mantenendo conforme lo stato del sistema. L'obbiettivo è rendere il canale indistinguibile rispetto alle altre risorse presenti nel sistema così da risultare invisibili ai sistemi di monitoraggio.

Caratteristica	**Descrizione**
Furtività	Evitare di attirare le attenzioni sia degli amministratori che degli strumenti utilizzati per il rilevamento degli attacchi.
Capacità di trasmissione	Espressa in termini di **throughput** ($\frac{dati}{tempo}$). Più dati il canale trasmette per un determinato intervallo di tempo, maggiore sarà il rischio che venga scoperto siccome potrebbe rendere anomalo il funzionamento delle altre risorse.

Uso delle risorse	Un uso delle risorse improprio o sproporzionato aumenta il rischio di essere individuati. Siccome il canale potrebbe andare in conflitto con le risorse legittime presenti nel sistema.
Rumore	Sfruttando servizi e/o risorse già presenti nel sistema, si potrebbe alterare il loro funzionamento. L'alterazione del comportamento della risorsa o del servizio sfruttato potrebbe attirare l'attenzione da parte degli amministratori.
Indistinguibilità	Si ha la necessità di riuscire a trasmette i dati sempre mantenendo conforme e inalterato il funzionamento della risorsa utilizzata. L'obbietivo è quello di rendersi indistinguibili dalla risorsa autorizzata e di conseguenza invisibili ai sistemi di monitoraggio.

Tabella 1: Caratterisitche di un Covert Channel

Nei seguenti capitoli verrà illustrato come è stato implementato. Principalmente le risorse condivise, e manipolate, saranno i pacchetti ICMP e i dati verranno codificati all'interno dei campi presenti. Altre metodologie sfrutteranno gli intervalli di tempo fra un pacchetto e il successivo oppure la tipologia di messaggio ICMP ricevuto.

2.1 Implementazione

In un Covert Channel ICMP (Internet Control Message Protocol) verranno utilizzati messaggi ICMP per nascondere i dati all'interno dei campi che normalmente vengono ignorati o non monitorati. L'implementazione del canale è possibile siccome è un protocollo che, dati i suoi utilizzi, non può essere del tutto disabilitato. In sinergia con il protocollo IP, permette di informare il mittente sui problemi di rete, di aiutare nella risoluzione dei problemi di rete e di gestire la congestione della rete.

Utilizzi	**Descrizione**
Diagnostica della rete	Il protocollo fornisce metriche critiche per il monitoraggio continuo delle prestazioni della rete
Segnalazione di errori	Il protocollo rileva e segnala i problemi riscontrati durante la trasmissione dei dati tra i dispositivi sulla rete
Risoluzione dei problemi	Gli amministratori di rete possono ricevere avvisi in tempo reale e rispondere rapidamente ai problemi di rete grazie agli strumenti di monitoraggio basati su ICMP.
Ottenimento di informazioni	Può inviare messaggi senza la necessità di una connessione preventiva permettendo così di ottenere informazioni.

Tabella 2: Utilizzi del protocollo ICMP

2.2 Struttura di un pacchetto ICMP

I messaggi ICMP vengono inviati utilizzando l'intestazione IP di base. In essi i primi venti byte indicano l'intestazione IP mentre il primo ottetto, della porzione dati del

datagramma, riguarda l'intestazione ICMP. I campi relativa al protocollo ICMP sono i seguenti:

- **Tipo**: Identifica il tipo di messaggio (ad esempio, Echo Request, Destinazione irraggiungibile). In base al valore di questo campo verrà determinato il formato dei rimanenti dati.
- **Codice**: Fornisce dettagli aggiuntivi sul tipo di messaggio.
- **Checksum**: complemento a 16 bit relativo alla somma del messaggio ICMP. Garantisce l'integrità dei dati.
- **Dati**: Campo opzionale, può contenere la parte del pacchetto IP originale che ha causato il messaggio o altre tipologie di dati.

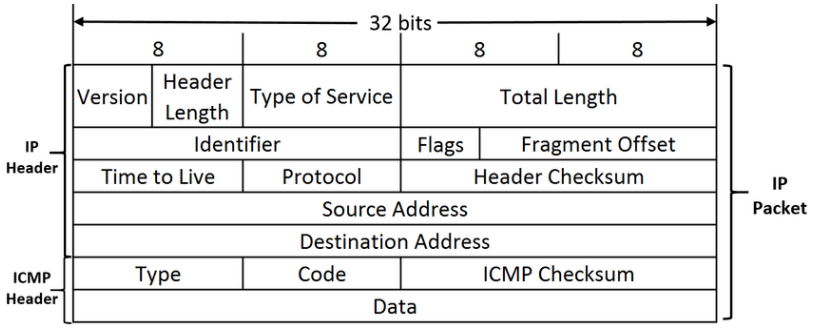


Figura 2: Struttura pacchetto ICMPv4/IPv4

Nell'intestazione, nel caso si usi il protocolllo ICMP, il campo *protocol* avrà valore 1

I messaggi presenti in ICMP sono classificati o come messaggi di errore o come messaggi informativi. I primi segnalano problemi nella comunicazione di rete mentre i secondi vengono utilizzati per scopi diagnostici e di controllo.

2.3 Timing Covert Channel

Sfruttano gli intervalli di tempo o l'ordine degli eventi per codificare le informazioni (e.g. ritardi fra i pacchetti di rete,...). Qualsiasi metodo che utilizza un orologio (o una misurazione del tempo) per segnalare il valore può implementarlo.

Per la **codifica**, si ricava il delay associato al dato e successivamente si apetta l'intervallo di tempo ricavato.

```
with open("path","r") as read_file:
    for data in read_file.read():
        delay=getDelay(data)
        wait(delay)
```

Listing 1: Pseudocodice per la codifica (Timing Covert Channel)

Invece per la **decodifica**; il destinatario calcolerà l'intervallo di tempo fra un pacchetto e l'altro, e da quello ricaverà il dato che gli è associato.

```
tempoPrecedente=None
def wait_packet(pacchetto):
    if not tempoPrecedente:
        tempoCorrente=pacchetto.time
        tempoPrecedente=pacchetto.time
        return
    tempoCorrente=pacchetto.time
    deltaTempo=tempoCorrente-tempoPrecedente
    data=getData(deltaTempo)
    tempoPrecedente= tempoCorrente
```

Listing 2: Pseudocodice per la decodifica (Timing Covert Channel)

2.3.1 Prima implementazione

Per poter codificare e decodificare i tempi da associare ai dati, in un primo momento si è definita la seguente funzione:

$$\text{tempo\_base} + \text{indice} * \text{distanza\_tempi} \tag{1}$$

Parametro	**Descrizione**
Tempo di base	Il minimo di secondi che si dovrà aspettare per ciascun possibile dato.
Distanza fra i tempi	Distanza minima di tempo che tutte le codifiche dovranno avere fra di loro. Lo scopo è evitare che i tempi relativi a due codifiche si sovrappongano.
Indice	Indica l'indice associato alla codifica del dato. Il suo valore và da 0 sino al masismo numero di codifiche possibili per i dati.

Tabella 3: Parametri della prima funzione (Timing Channel)

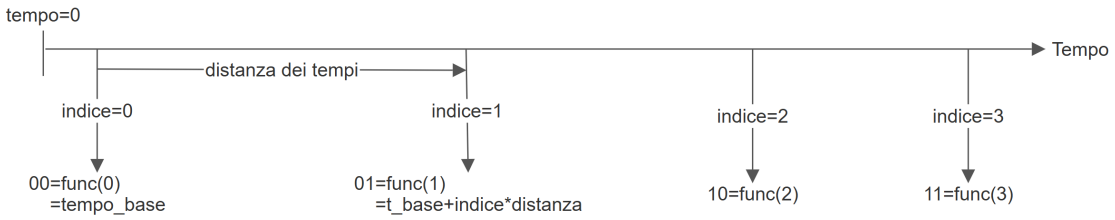


Figura 3: Funzione 1: Assegnazione degli intervalli con la distanza

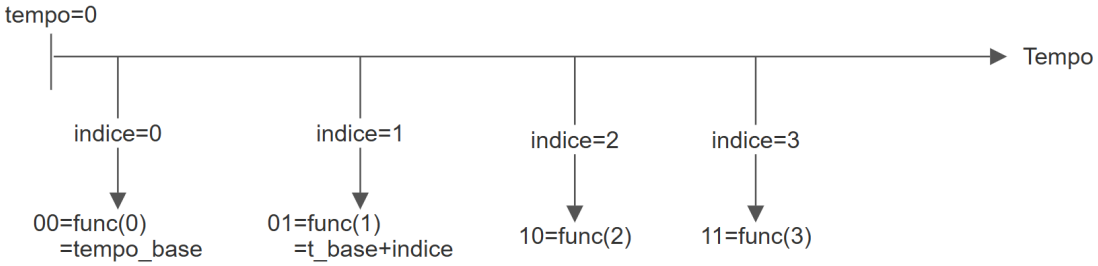


Figura 4: Funzione 1: Assegnazione degli intervalli senza la distanza

Se mai nel sistema avvenissero dei ritardi data la latenza della rete unn intervallo di tempo potrebbe codificare il dato sbagliato. Infatti il tempo in cui la destinazione riceverà il messaggio è dato dal delay per codificare il dato più il ritardo di rete. In generale il valore di questo parametro dipenderà dall'ubicazione geografica dell'entità mittente e di quella destinataria, oltre alla reattività delle due macchine.

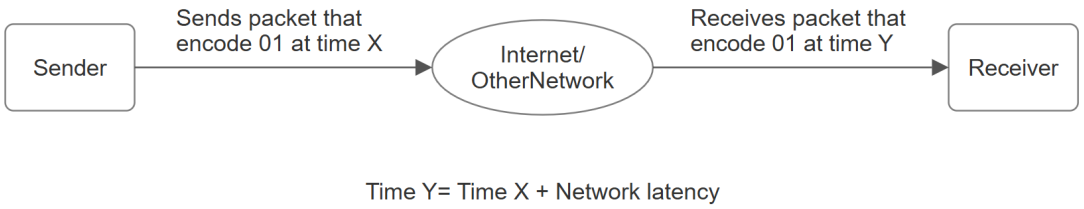


Figura 5: Schema latenza nel sistema

2.3.2 Seconda implementazione

Uno secondo sviluppo effettuato è stato per l'ampliamento dei dati trasportabili. Nella prima versione i tempi di attesa per inviare i dati risultavano eccessivamente lunghi. Oer migliorare questo aspetto, si è preso spunto dall'approccio implementato in ICMP Exfill: in esso il delay da aspettare viene determinato dal valore del byte che si vuole inviare. ciò è possibile siccome in memoria rappresenta una sequenza di bit che può essere interpretata come un numero intero. L'intervallo di tempo verrà quindi ricavato dall'intero associato al byte da codificare.

Tuttavia una semplice conversione risulta inefficace siccome il tempo minimo di at-

tessa risultava zero mentre il delay masismo in alcunicasi potrà essere 255 secondi. Inizialmente si è optato per sottrarre al numero ricavato un certo valore, questa sceltaè stata dettata dal fatto che i caratteri stampabili nella codifica ASCII hanno un range ben definito. Tuttavia ciò avrebbe tagliato fuori alcuni caratteri speciali come il tab o il carattere che indica una nuova linea. Inoltre avrebbe vincolato l'approccio solo alla codifica ASCII e quindi avrebbe portato a un metodo per codifica dei caratteri presenti.

Range	**Descrizione**
Control Character [0-31]	Codici di controllo non stampabili e che vengono utilizzati per controllare periferiche.
Printable character [32-127]	Rappresentano lettere, cifre, segni di punteggiatura e vari simboli.
Extended ASCII Codes [128-255]	I valori fra 128 a 255 non fanno parte dell'ASCII standard; appartengono alla sua versione estesa. I caratteri presenti dipendono dalla codifica utilizzata. La più comune è ISO 8859-1 (chiamata anche ISO Latin-1).

Tabella 4: Struttura della tabella ASCII

Quindi per poter ridurre la quantità di tempo necessaria per mandare i dati, si è definita una nuova funzione che cercerà di normalizzare l'intervallo di tempo, associato al dato, fra un valore minimo e uno massimo. In questo caso bisongerà impostare correttamente il valore minimo e quello massimo. Minore è la loro differenza maggiore sarà la probabilità che i dati vengano codificati in maniera errata.

$$\text{min\_delay} + (\text{byte}/255) * (\text{max\_delay} - \text{min\_delay}) \tag{2}$$

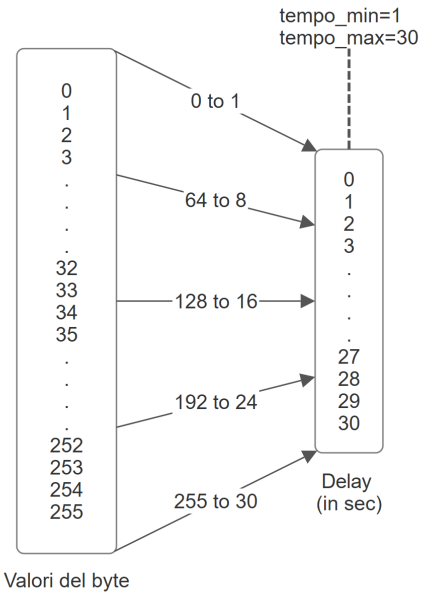


Figura 6: Funzione 2: Assegnazione degli intervalli

2.3.3 Implementazione finale

L'implementazione precedente è stata sviluppata ulteriormente inserendo del rumore al tempo di delay così che per ogni possibile intervallo, non sarà più presente un tempo costante fra un pacchetto e il successivo. Siccome la quantità di rumore aggiunta dovrà risultare casuale, si è definito un range di tempo nel quale può variare.

Gli IDS riescono ad individuare gli schemi temporali, quindi definire un range permetterà di non essere rilevati. Ora la codifica di un byte, non è più definita da un singolo, costante e prevedibile intervallo di tempo, ma da un range di intervalli temporali che continuamente cambiano.

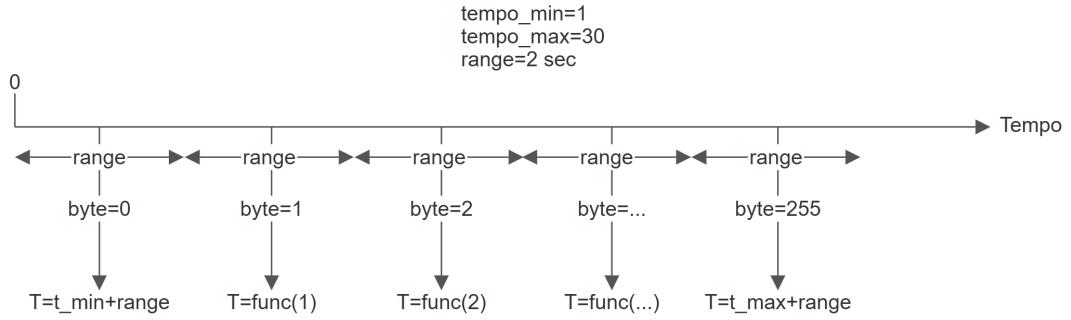


Figura 7: Schema assegnazione dei tempi con rumore

Il mittente e il destinatario devono potersi sincronizzare sulla quantità di rumore si è aggiunto; due strade possibili sono:

- Nella prima ogni entità crea un numero randomico e bisognerà sincronizzare le due entità sulla quantità e sul valore dei numeri estratti. Se i numeri non combaciano, il dato ricavato risulterà non corretto.
- Il secondo approccio è di mettere il rumore generato nel pacchetto stesso o direttamente nel payload. Il ricevente otterrà il valore dalla risorsa condivisa.

2.4 Storage Covert Channel

Il canale viene creato scrivendo dei dati su un'area di memoria condivisa accessibile da tutte entità presenti. I possibili veicoli saranno tutte quelle risorse che consentiranno la scrittura (diretta o indiretta) da parte di un processo e la lettura (diretta o indiretta) da parte di un altro.

Nel nostro caso la risorsa condivisa sarà il pacchetto ICMP inviato. Il mittente scriverà i dati nei campi del messaggio così che il destinatario, una volta ricevuto, potrà poi leggere i valori codificati al loro interno.

Tipologia	**Byte trasmessi**	**Campi Utilizzati**	**Descrizione**

Destination Unreachable	3-8	unused, header+64 bits	Una destinazione (rete, porta,...) risulta non disponibile
Source Quench	3-8	unused, header+64 bits	Un gateway notifica il buffer di memoria per i pacchetti pieno (indica la congestione nella rete).
Redirect Message	3-4	header+64 bits	Suggerisce un reindirizzamento del pacchetto verso un percorso migliore.
Time Exceeded	3-8	unused, header+64 bits	Il gateway notifica che il TTL del pacchetto ricevuto risulta zero.
Parameter Problem	4-8	pointer, unused, header+64 bits	Il gateway rileva dei problemi nei campi dell'intestazione
Echo Request	2	identifier, data	Usato per inviare un dato a un destinatario e ricevere una risposta indietro.
Echo Reply	2	identifier, data	Replica i dati ricevuti nella richiesta rimandandoli al mittente
Timestamp Request	5	identifier,timestamp, data	Usato per inviare una serie di timestamp a un destinatario e ricevere una risposta indietro.

Timestamp Reply	5	identifier,timestamp, data	Replica i timestamp ricevuti nella richiesta rimandandoli al mittente
Information Request	2	identifier	Permete di scoprire se l'host si trova nella stessa rete di chi ha risposto. Nel mandare il pacchetto lascia il campo *destinazione* vuoto.
Information Reply	2	identifier	Risponde alla richiesta con tutti i campi compilati correttamente. In particolare quello relativo al proprio indirizzo IP.

Tabella 6: Tipologie di messaggi ICMPv4

Tipologia	**Codici**	**Campi Sfruttatti**	**Uso**
Destination Unreachable	4-8	unused, invoking packet	Una destinazione (rete, porta,...) risulta non disponibile
Packet Too Big	8	mtu, invoking packet	Un router notifica l'impossibilità nell'inoltrare un pacchetto (indica la congestione nella rete)

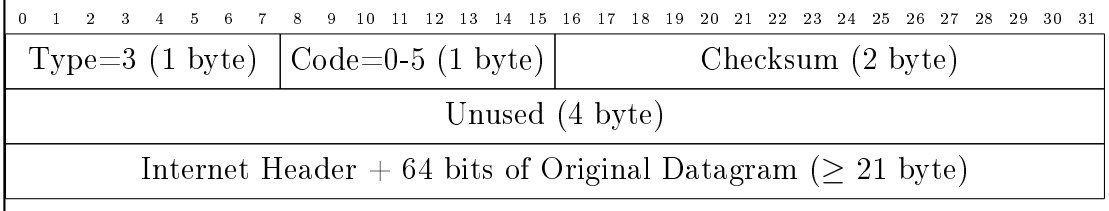
Time Exceeded	4-8	unused, invoking packet	Il gateway notifica che il TTL del pacchetto ricevuto risulta zero.
Parameter Problem	8	pointer, invoking packet	Il gateway rileva dei problemi nei campi dell'intestazione
Echo Request	2	identifier, data	Usato per inviare un dato a un destinatario e ricevere una risposta indietro.
Echo Reply	2	identifier, data	Replica i dati ricevuti nella richiesta rimandandoli al mittente

Tabella 8: Tipologie di messaggi ICMPv6

2.4.1 Destination Unreachable

Viene inviato quando il pacchetto non può essere recapitato alla destinazione specificata nell'header IP. Di solito perchè il percorso definito non può essere seguito. Il messaggio non verrà (e non dovrà essere) generato se un pacchetto viene scartato a causa della congestione del traffico.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:



Nel protocollo **ICMPv6** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Type=1 (1 byte)	Code=0-6 (1 byte)	Checksum (2 byte)
Unused (4 byte)		
As much of invoking packet as possible without		
the ICMPv6 packet exceeding the minimum IPv6 MTU ($\geq$ 0 byte)		

In ICMPv4 il campo *Internet Header*: viene utilizzato dall'host per accoppiare il messaggio di errore al processo appropriato. Se un protocollo di livello superiore utilizza numeri di porta, si presume che siano nei primi 64 bit dei dati del datagramma originale.

In ICMPv6 il campo *Invoking Packet* indica quanta parte del pacchetto (che ha attivato l'errore ICMPv6) debba essere inclusa. Il tutto senza eccedere il *IPv6 MTU* il cui valore di default equivale a 1280 bytes.

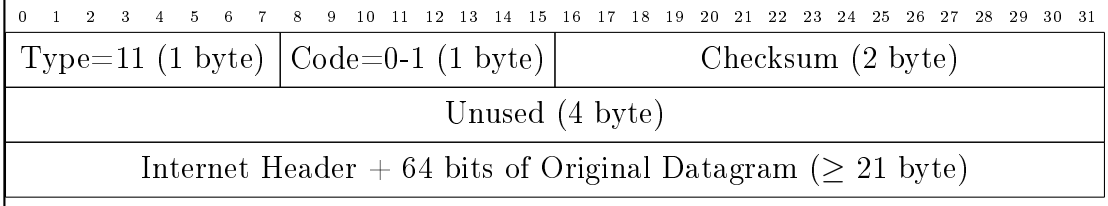
Campo	**Utilizzo**
Unused	Dalle specifiche RFC 792 (e RFC 4434) dovrebbe essere 0. Tuttavia è comunque utilizzabile.
Header+64 bits (ICMPv4)	Nel campo si dovranno inserire l'header IP più 8 byte. Nel nostro caso si è usato ICMP/IP ma può essere utilizzato anche TCP/IP o altri protocolli purchè lunghezza passata, non sfori i 28 bytes.-
Invoking Packet (ICMPv6)	Nel campo, il pacchetto usatoper indicare l'errore, sarà quello con il protocollo IPV6 e ICMPv6. Tuttavia, in questo caso si dovrà stare attenti a non superare la *IPv6 MTU*, ma ciò non succederà siccome l'intestazione IPv6 sarà di 40 byte emntre l'intestazione ICMPv6 sarà di 8 byte.

Tabella 9: Campi usati in Destination Unreachable

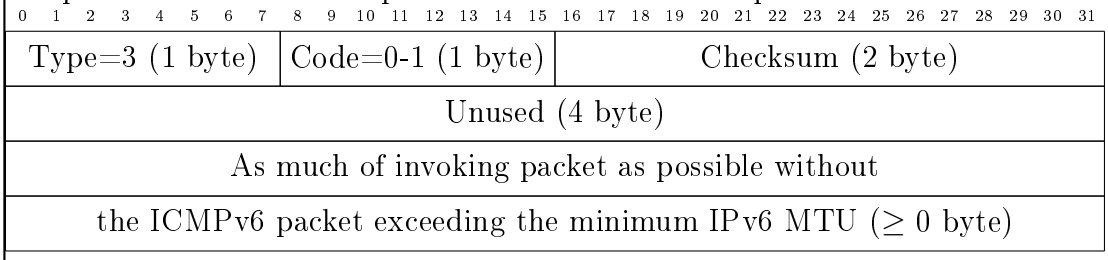
2.4.2 Time Exceeded

Questa tipologia di messaggio viene usata quando il gateway che elabora un pacchetto trova che il suo TTL (tempo di vita) è zero. In questi casi il gateway dovrà scartare il datagramma e notificare l'host sorgente della cosa.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:



Nel protocollo **ICMPv6** il pacchetto è strutturato in questo modo:



In ICMPv4 il campo *Internet Header*: viene utilizzato dall'host per accoppiare il messaggio di errore al processo appropriato. Se un protocollo di livello superiore utilizza numeri di porta, si presume che siano nei primi 64 bit dei dati del datagramma originale.

In ICMPv6 il campo *Invoking Packet* indica quanta parte del pacchetto (che ha attivato l'errore ICMPv6) debba essere inclusa. Il tutto senza eccedere il *IPv6 MTU* il cui valore di default equivale a 1280 bytes.

Campo	**Utilizzo**
Unused	Dalle specifiche RFC 792 (e RFC 4434) dovrebbe essere 0. Tuttavia è comunque utilizzabile.

Header+64 bits (ICMPv4)	Nel campo si dovranno inserire l'header IP più 8 byte. Nel nostro caso si è usato ICMP/IP ma può essere utilizzato anche TCP/IP o altri protocolli purchè lunghezza passata, non sfori i 28 bytes.-
Invoking Packet (ICMPv6)	Nel campo, il pacchetto usatoper indicare l'errore, sarà quello con il protocollo IPV6 e ICMPv6. Tuttavia, in questo caso si dovrà stare attenti a non superare la *IPv6 MTU*, ma ciò non succederà siccome l'intestazione IPv6 sarà di 40 byte emntre l'intestazione ICMPv6 sarà di 8 byte.

Tabella 10: Campi usati in Time Exceeded

2.4.3 Parameter Problem

Viene usata quando il gateway che elabora un pacchetto trova un problema con i parametri dell'intestazione in modo tale da non poter completare l'elaborazione del datagramma. In questo caso dovrà scartare il datagramma e notificare la cosa all'host indicando il tipo e la posizione del problema. Il messaggio viene inviato solo se l'errore ha causato lo scarto del pacchetto.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Type=12 (1 byte)	Code=0 (1 byte)	Checksum (2 byte)
Pointer (1 byte)	Unused (3 byte)	
Internet Header + 64 bits of Original Datagram ($\geq$ 21 byte)		

Nel protocollo **ICMPv6** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Type=4 (1 byte)	Code=0-2 (1 byte)	Checksum (2 byte)
Pointer (4 byte)		
As much of invoking packet as possible without		
the ICMPv6 packet exceeding the minimum IPv6 MTU ($\geq$ 0 byte)		

In ICMPv4 il campo *Internet Header*: viene utilizzato dall'host per accoppiare il messaggio di errore al processo appropriato. Se un protocollo di livello superiore utilizza numeri di porta, si presume che siano nei primi 64 bit dei dati del datagramma originale.

In ICMPv6 il campo *Invoking Packet* indica quanta parte del pacchetto (che ha attivato l'errore ICMPv6) debba essere inclusa. Il tutto senza eccedere il *IPv6 MTU* il cui valore di default equivale a 1280 bytes.

Il puntatore identifica l'ottetto nell'intestazione del pacchetto originale in cui è stato rilevato l'errore (può trovarsi nel mezzo di un'opzione).

Campo	**Utilizzo**
Unused (ICMPv4)	Dalle specifiche RFC 792 dovrebbe essere 0. Tuttavia è comunque utilizzabile.
Pointer	Indica l'ottetto in cui è presente l'errore. Tuttavia potrebbe contenere dei dati non correlati ad esso.
Header+64 bits (ICMPv4)	Nel campo si dovranno inserire l'header IP più 8 byte. Nel nostro caso si è usato ICMP/IP ma può essere utilizzato anche TCP/IP o altri protocolli purchè lunghezza passata, non sfori i 28 bytes.-

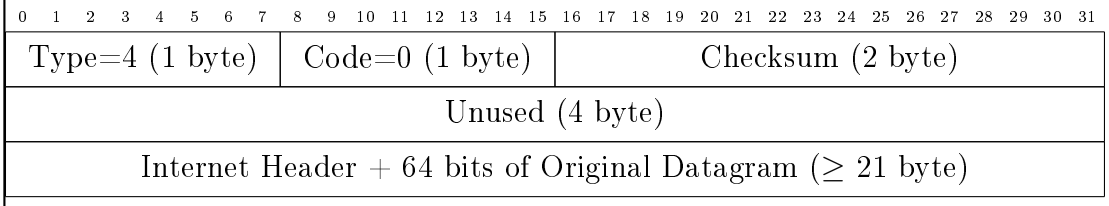
Invoking Packet (ICMPv6)	Nel campo, il pacchetto usatoper indicare l'errore, sarà quello con il protocollo IPV6 e ICMPv6. Tuttavia, in questo caso si dovrà stare attenti a non superare la *IPv6 MTU*, ma ciò non succederà siccome l'intestazione IPv6 sarà di 40 byte emntre l'intestazione ICMPv6 sarà di 8 byte.

Tabella 11: Campi usati in Parameter Problem

2.4.4 Source Quench

Questa tipologia viene usata quando il gateway vuole richiedere di ridurre la velocità di invio dei pacchetti. Questo perchè il gateway ha scartato un pacchetto ma non a causa di un errore. Al suo ricevimento, l'host sorgente dovrà ridurre la velocità sino a quando non riceverà più messaggi deltipo source Quench dal gateway.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:



Il campo *Internet Header*: viene utilizzato dall'host per accoppiare il messaggio di errore al processo appropriato. Se un protocollo di livello superiore utilizza numeri di porta, si presume che siano nei primi 64 bit dei dati del datagramma originale.

Campo	**Utilizzo**
Unused (ICMPv4)	Dalle specifiche RFC 792 dovrebbe essere 0. Tuttavia è comunque utilizzabile.

Header+64 bits (ICMPv4)	Nel campo si dovranno inserire l'header IP più 8 byte. Nel nostro caso si è usato ICMP/IP ma può essere utilizzato anche TCP/IP o altri protocolli purchè lunghezza passata, non sfori i 28 bytes.-

Tabella 12: Campi usati in Source Quench

2.4.5 Redirect

Indica un messaggio di reindirizzamento a un host. Il gateway manda questo tipo di messaggio se, dopo aver controllato la sua tabella di routing, trova che esiste un gateway migliore che si trova sulla sua stessa rete. Questo secondo gateway rappresenterà un percorso migliore per la destinazione.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Type=5 (1 byte)	Code=0-3 (1 byte)	Checksum (2 byte)
Gateway Internet Address (4 byte)		
Internet Header + 64 bits of Original Datagram ($\geq$ 21B)		

Nel campo *Gateway Internet Address* verrà indicato l'indirizzo del nuovo gateway a cui dovrà essere inviato il traffico per la rete di destinazione (specificata nel campo di destinazione del datagram originale). Si potrebbe pensare di utilizzare il campo ma un gateway o la vittima per necessità potrebbero leggere i dati e scoprire che non sono conformi.

Il campo *Internet Header* viene utilizzato dall'host per accoppiare il messaggio di errore al processo appropriato. Se un protocollo di livello superiore utilizza numeri di porta, si presume che siano nei primi 64 bit dei dati del datagramma originale.

Se nell'itestazione IP è presente l'opzione *IP Source Route*, il messaggio di reindirzzamento non verrà inviato anche se è presente un percorso migliore.

Campo	Utilizzo
Header+64 bits (ICMPv4)	Nel campo si dovranno inserire l'header IP più 8 byte. Nel nostro caso si è usato ICMP/IP ma può essere utilizzato anche TCP/IP o altri protocolli purchè lunghezza passata, non sfori i 28 bytes.-

Tabella 13: Campi usati in Redirect

2.4.6 Echo Request / Echo Reply

Un messaggio *Echo*, viene usato per ricevere indietro una risposta da un host. Si inviano dei dati tramite una Echo Request, e questi'ultimi dovranno essere restituiti in un messaggio di risposta integralmente e senza modifiche.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Type=8 (1 byte)	Code=0 (1 byte)	Checksum (2 byte)
Identifier (2 byte)		Sequence Number (2 byte)
Data ... ($\geq$ 0 byte)		

Nel protocollo **ICMPv6** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Type=128 (1 byte)	Code=0 (1 byte)	Checksum (2 byte)
Identifier (2 byte)		Sequence Number (2 byte)
Data ... ($\geq$ 0B)		

Nel messaggio, i campi identificatore e numero di sequenza possono essere utilizzati dal mittente per facilitare l'abbinamento delle risposte con le richieste.

Inoltre il mittente non ha alcuna limitazione sulla quantità di dati inseribili nei messaggi *Echo* (finchè la dimensione risulti minore di 65 KB). Tuttavia una limitazione potrà essere definita dalla massima capacità di trasporto dei collegamenti. Questo

perchè nel caso la dimensione del messaggio la superasse; il pacchetto dovrà essere frammentato per poter essere spedito. In media il valore si attesta sui 1400 bytes.

Campo	**Utilizzo**
Identifier	Definisce l'identificativo delle richieste e può assumere un qualsiasi valore.
Sequenza	Dalle specifiche RFC 792 (e RFC 4443) viene incrementato ad ogni richiesta inviata. Se il valore cambiasse in maniera non strettamente crescente, la cosa potrebbe risultare sospetta. Tuttavia può essere utilizzato.
Data	In questo campo il mittente può inserire quanti dati preferisce. Ma per sicurezza la dimensione sarà strettamente uguale o inferiore a 32 bytes.

Tabella 14: Campi usati in Echo Reques/Reply

2.4.7 Timestamp Request / Timestamp Reply

Viene usato per ricevere indietro una risposta da un host. I dati ricevuti nel messaggio di richiesta, vengono restituiti in quello di risposta insieme a dei timestamp aggiuntivi. Il timestamp è pari a 32 bit e indica i millisecondi che sono passati dalla mezzanotte UT.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Type=13 (1 byte)	Code=0 (1 byte)	Checksum (2 byte)
Identifier (2 byte)		Sequence Number (2 byte)
Originate Timestamp (4 byte)		
Receive Timestamp (4 byte)		
Transmit Timestamp (4 byte)		
Data ... ($\geq$ 0 byte)		

L'identificatore e il numero di sequenza possono essere utilizzati dal mittente del pacchetto per facilitare l'abbinamento delle risposte con le richieste. Mentre i campi relativi ai timestamp indicheranno rispettivamente: il tempo in cui il mittente ha toccato il messaggio per l'ultima volta prima di inviarlo, il tempo in cui il destinatario ha toccato per la prima volta il messaggio (alla ricezione) e il tempo in cui il destinatario ha toccato il messaggio per l'ultima volta prima di inviarlo.

Campo	**Utilizzo**
Identifier	Definisce l'identificativo delle richieste e può assumere un qualsiasi valore.
Sequenza	Dalle specifiche RFC 792 viene incrementato ad ogni richiesta inviata. Se il valore cambiasse in maniera non strettamente crescente, la cosa potrebbe risultare sospetta. Tuttavia può essere utilizzato.
campi Timestamp	Contiene i milliseocndi dalla mezzanotte UT e ciascun campo ha un ordine temporale specifico; che dovrà rimanere congruente quando si inseriscono i dati. In ciascunodi essi verrà inserito un byte nella parte indicante i millisecondi.
Data	In questo campo il mittente può inserire quanti dati preferisce. Ma per sicurezza la dimensione sarà strettamente uguale o inferiore a 32 bytes.

Tabella 15: Campi usati in Timestamp Reques/Reply

2.4.8 Information Request / Information Reply

La tipologia *Information* viene usata per consentire di scoprire il numero della rete in cui un host si trova. Serve quindi per capire se si trova nella stesse rete dell'host che risponde. Sebbene il messaggio può essere inviato con la destinazione nell'intestazione IP pari a zero (ciò significa "questa" rete); l'intestazione IP presente nel messaggio

di risposta dovrà essere inviata con gli indirizzi IP completamente specificati.

Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Type=15 (1 byte)	Code=0 (1 byte)	Checksum (2 byte)
Identifier (2 byte)		Sequence Number (2 byte)

L'identificatore e il numero di sequenza possono essere utilizzati dal mittente del pacchetto per facilitare l'abbinamento delle risposte con le richieste.

Campo	**Utilizzo**
Identifier	Definisce l'identificativo delle richieste e può assumere un qualsiasi valore.
Sequenza	Dalle specifiche RFC 792 viene incrementato ad ogni richiesta inviata. Se il valore cambiasse in maniera non strettamente crescente, la cosa potrebbe risultare sospetta. Tuttavia può essere utilizzato.

Tabella 16: Campi usati in Information Reques/Reply

2.4.9 Packet Too Big

Un messaggio *Packet Too Big* viene generato da un router in risposta a un pacchetto che non può inoltrare perché è più grande dell'MTU del collegamento in uscita. Un nodo che riceve un messaggio *ICMPv6 Packet Too Big* deve notificare la cosa al processo di livello superiore (se il processo in questione può essere identificato).

Nel protocollo **ICMPv6** il pacchetto è strutturato in questo modo:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Type=2 (1 byte)	Code=0 (1 byte)	Checksum (2 byte)
MTU (4 byte)		
As much of invoking packet as possible without		
the ICMPv6 packet exceeding the minimum IPv6 MTU ($\geq$ 0 byte)		

Il campo *MTU* indica la massima unità di trasmissione del collegamento nel salto successivo. Mentre il campo *Invoking Packet* indica quanta parte del pacchetto (che ha attivato l'errore ICMPv6) debba essere inclusa. Il tutto senza eccedere il *IPv6 MTU* il cui valore di default equivale a 1280 bytes.

Campo	**Utilizzo**
MTU	Rappresenta la capacità massima del collegamento; siccome il suo valore è variabile, potrà essere usato per inserire in esso dei dati.
Invoking Packet	Nel campo, il pacchetto usato per indicare l'errore, sarà quello con il protocollo IPV6 e ICMPv6. Tuttavia, in questo caso si dovrà stare attenti a non superare la *IPv6 MTU*, ma ciò non succederà siccome l'intestazione IPv6 sarà di 40 byte mentre l'intestazione ICMPv6 sarà di 8 byte.

Tabella 17: Campi usati in Packet Too Big

2.5 Behavioral Covert Channel

Le informazioni vengono codificate modificando il comportamento del sitema. Quindi i dati vengono ricavati osservando il comportamento del sistema piuttosto che attraverso l'accesso diretto ai dati.

In questo caso l'evento che si osserverà è la tipologia ed il codice del messaggio ICMP che è arrivato. Di condesguenza la quantità di eventi inviabili risulta 17. E quindoi

ogni messaggio permetterà di codificare 4 bit alla volta (siccome $\log_2 16 = 4$). La tipologia inutilizzata, verrà sfruttata per indicare l'inizio e la fine della comunicaizone.

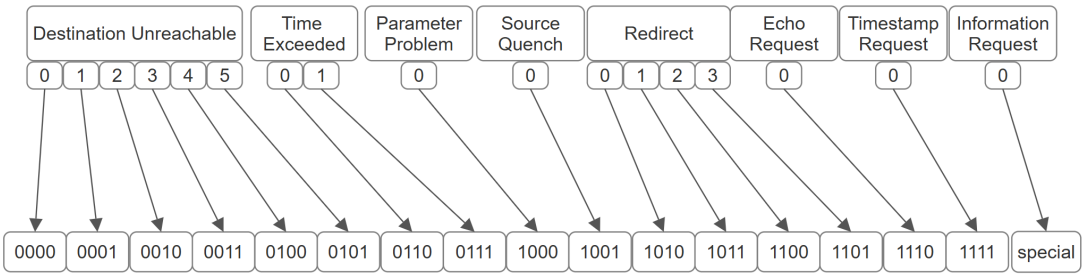


Figura 8: Schema Hybrid Channel

In questo caso si maschererà il byte così da dividerlo in due parti. Ogni 4 bit determinano la tipologie e il codice che il messaggio da inviare dovrà avere.

```
primi_4bit=(numero& 0b11110000)>>4
ultimi_4bit=numero& 0b00001111

bit_codice={}
lista_codici=[
    30,31,32,33,34,35 #destination unreachable
    ,110,111 #time exceeded
    ,120 #parameter problem
    ,40 #source quench
    ,50,51,52,53 #redirect
    ,80#, 0 #echo request/reply
    ,130#, 140 #timestamp request/reply
    ,150#, 160 #info request/reply
]
for index in range(0,len(lista_codici)):
    bit_codice.update({index:lista_codici[index]})
```

```
#messaggio=[codice ,tipologia]
primo_messaggio=[
        bit_codice[primi_4bit]%10
       ,bit_codice[primi_4bit]//10
]
secondo_messaggio=[
        bit_codice[ultimi_4bit]%10
       ,bit_codice[ultimi_4bit]//10
]
```

Tabella 18: Estrazione dal byte il Codice e Tipologia per il messaggio

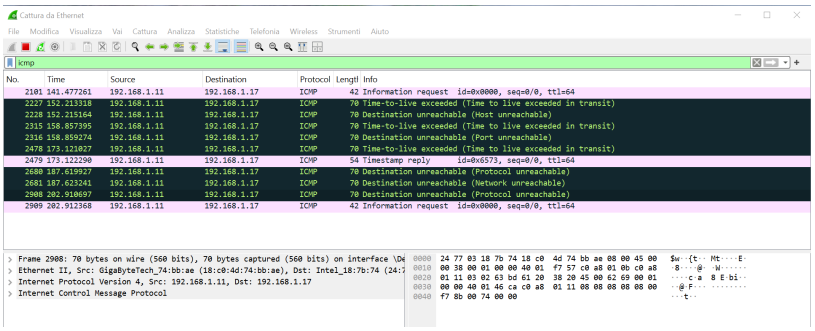


Figura 9: Flusso di un Covert Channel Comportamentale

Il destinatario monitora il traffico di rete e controllerà le tipologie di messaggi ICMP provenienti. Da ogni coppia (Tipologia, Codice) ricaverà quatttro bit.

```
previous_type=prev_pkt[ICMP].type*10+prev_pkt[ICMP].code
current_type=curr_pkt[ICMP].type*10+curr_pkt[ICMP].code

codice_bit={}
lista_codici=[
    30,31,32,33,34,35 #destination unreachable
    ,110,111 #time exceeded
    ,120 #parameter problem
    ,40 #source quench
    ,50,51,52,53 #redirect
```

```
    ,80#, 0 #echo request/reply
    ,130#, 140 #timestamp request/reply
    ,150#, 160 #info request/reply
]
for index in range(0,len(lista_codici)):
    codice_bit.update({lista_codici[index]:index})

previous_bit=codice_bit[previous_type]
current_bit=codice_bit[current_type]
byte=(previous_bit<<4)+current_bit
```

Tabella 19: Estrazione dai messaggi il byte del dato

2.6 Hybrid Covert Channel

Partendo dalle versioni precedenti dei Covert Channel sviluppati; si può implementare una versione che le combina.

- Da un byte verrà ricavato il delay fra un pacchetto e l'altro. (Timing Channel)
- Da un byte verranno ricavate le due tipologie di messaggi da mandare. (Behavioural Channel)
- In ciascun pacchetto verranno inseriti tanti dati quanto la capacità di trasmissione del messaggio. (Storage Channel)

```
dato="Dato da inviare"
index=0
while index < len(dato)
    byte_1=dato[index]
    index+=1
    delay=getDelay(byte_1)
    wait(delay)

    byte_2=dato[index]
```

```
        index+=1
        primi_4bit=(byte_2 & 0b11110000)>>4
        ultimi_4bit=byte_2 & 0b00001111

        for bit in [primi_4bit, ultimi_4bit]:
            tipologia_msg=getTipologia(bit)
            capacita_msg=getCapacita(tipologia_msg)
            data_msg=data[index:index+capacita_msg]
            packet=getPacket(tipologia_msg, data_msg)
            index+=capacita_msg
            send(packet)
```

Listing 3: Pseudocodice per l'invio dei dati con Hybrid Covert Channel

La comunicazione viene iniziata tramite un messaggio *Information Request* e chiusa sempre da un messaggio dello stesso tipo. Una volta aperta, si ricaverà l'intervallo di tempo con cui arrivano le coppie di messaggi e da questo si ricaverà un byte. Per ogni messaggio presente nella coppia, vengono ricavati i quattro bit associati alla tipologia eal codice del pacchetto in questione. Una volto uniti tutti i bit, in base all'ordine di arrivo dei due pacchetti, si oterrà un byte. Infine si ricaveranno i dati che sono stati nascosti nei campi del messaggio ICMP.

```
isSending=False
tempo_precedente=None
tempo_corrente=None
received_packet=0
tipologia_prima=None

data=[]

def wait_packet(pacchetto):
    if pacchetto.tipologia =="Information Request":
        if not isSending:
```

```
            print("Primo Information Request")
            isSending= not isSending
            tempo_precedente=pacchetto.time
            tempo_corrente=pacchetto.time
        else:
            print("Secondo Information Request")
            exit()
    if not isSending:
        return
    received_packet=(received_packet+1)%2
    if received_packet==1:
        print("Primo pacchetto")
        tipologia_prima=pacchetto.tipologia
        tempo_corrente=pacchetto.time
        delta=tempo_corrente-tempo_precedente
        tempo_precedente=tempo_corrente
        data.append(getTimeData(delta))
    elif received_packet==0:
        print("Secondo pacchetto")
        tipologia_ora=pacchetto.tipologia
        data.append(
            getBehaveData(tipologia_ora, tipologia_prima)
        )
    data.append(getStorageData(pacchetto))
```

Listing 4: Pseudocodice per ricezione dei dati con Hybrid Covert Channel

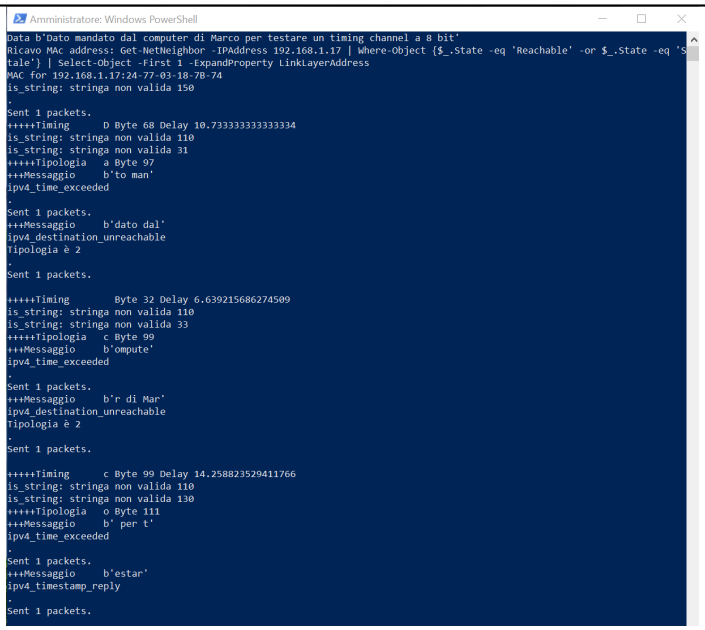


Figura 10: Invio dei dait tramite Hybrid Covert Channel

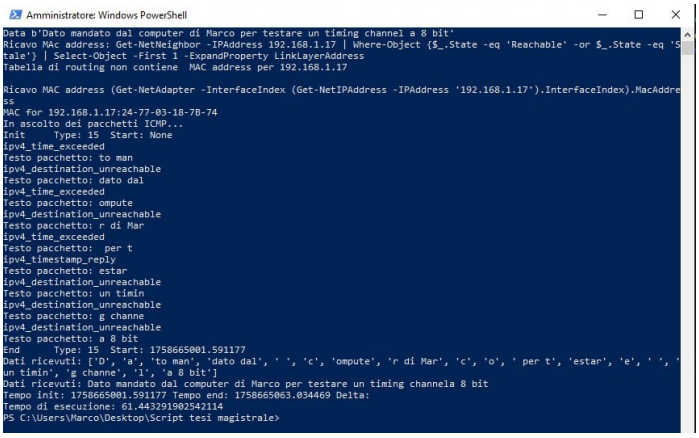


Figura 11: Ricezione dei dait tramite Hybrid Covert Channel

3 Struttura della comunicazione fra le entità

Le entita coinvolte sono l'attaccante, il proxy e la vittima.

Entità	**Descrizione**

Attaccante	Carica un file di configurazione nel quale viene definito l'indirizzo IP della vittima, il metodo di attacco e i proxy utilizzabili per farlo. Inoltre inserirà il comando che la vittima dovrà eseguire o il dato che gli si vuole mandare. Nel caso l'attaccante utilizzasse dei proxy, aspettera da essi i dati che la vittima ha restituito.
Proxy	Ha bisongo di sapere qual'è l'indirizzo IP dell'attaccante. Deve potersi connettere ad esso ed ottenere l'indirizzo IP della vittima e il comando da inoltrare. Una volta stabilita una connesisone sia con l'attaccante che con la vittima; si occuperà di inoltrare i dati che le due entità si inviano.
Vittima	Aspetta le connessioni o dall'attaccante o dai proxy se vengonoutilizzati. Dopodichè aspetterà un comando (o un dato). Nel primo caso lo eseguirà e invierà i dati ricavati dall'esecuzione.

Tabella 20: Entita presenti nell'attacco

Il flusso di come avviene la comunicazione fra le entità è definito dal diagramma seguente. Il canale di comunicazione che il proxy e l'attaccante protrano avere dipende dall'affidabilità del proxy. Nel caso si reputi il proxy insicuro, si potrà utilizare una comunicazione tramite ICMP (ovvero la stessa che il proxy avrà con la vittima) altrimenti si potra usare il protocollo TCP; che permetterà una comunicazione maggiormente stabile.

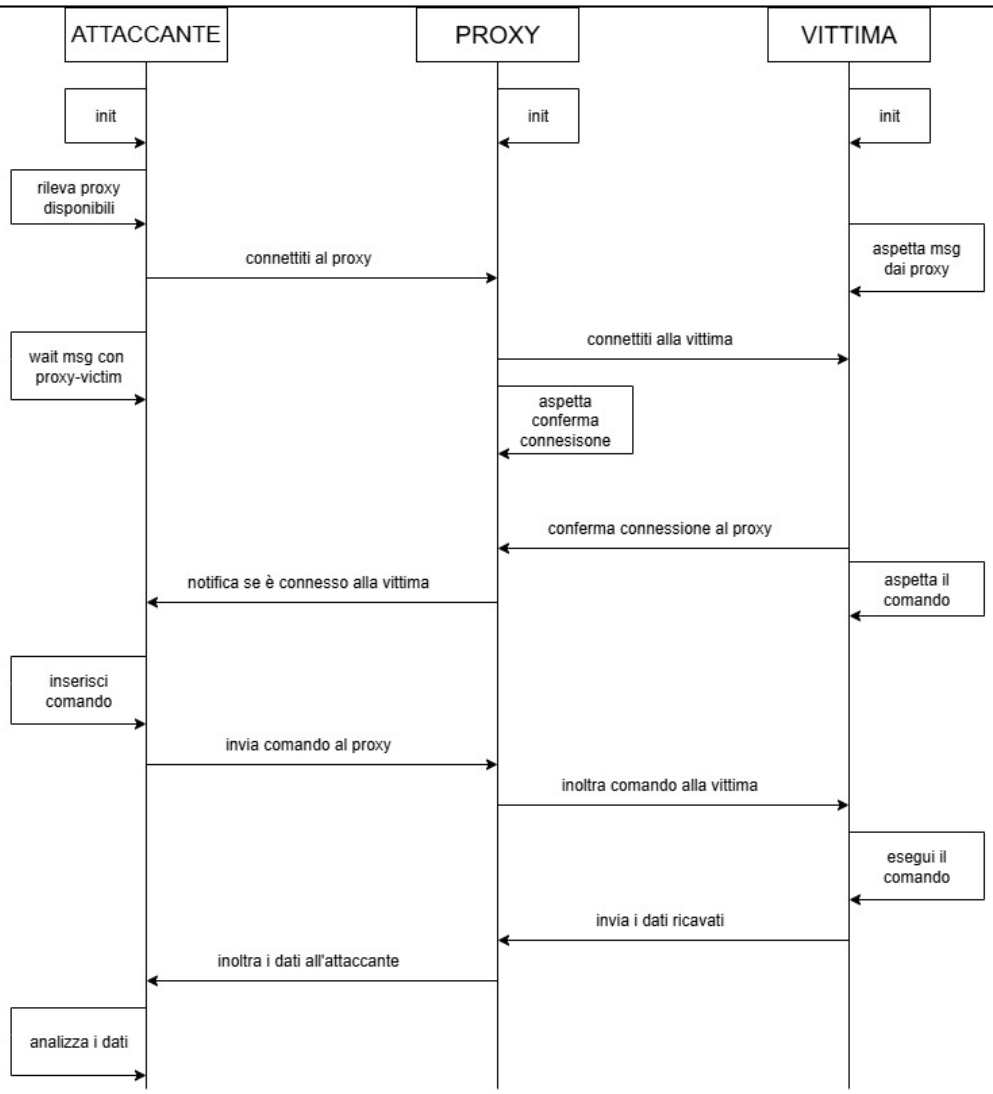


Figura 12: Diagramma del flusso di comunicazione fra le entita

Per la comunicazione l'attaccante potrà usare uno o più proxy per comunicare con la vittima. Il caso standard sarà quello rappresentato nella figura tuttavià sarà possibile, per l'attaccante, comunicare direttamente con la vittima. In questo caso l'entità proxy non verrà utilizzata ed alcune funzioni presenti in essa verranno eseguite dall'attaccante (e.g connettersi alla vittima) e quindi l'entità proxy potrà essere vista come l'entità attaccante.

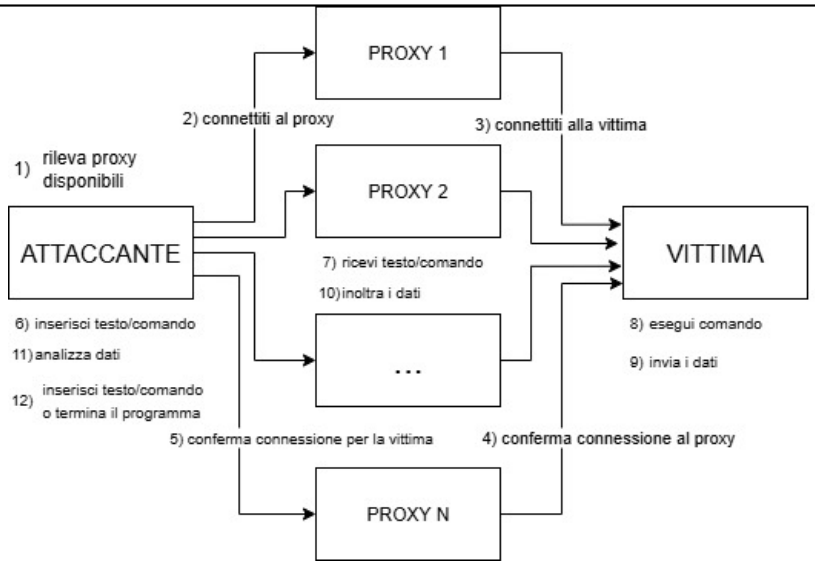


Figura 13: Struttura delle entità presenti e come dialogano

3.1 Struttura dell'attaccante

Tramite un parser rilevare se nella linea di comando, è stato inserito un path per il file di configuraizone da caricare. In questo file JSON viene specificato l'indirizzo IP della vittima, la metodologia di attacco e la lista dei proxy a cui ci si vuole connettere.

Dopodichè per ogni proxy specificato verificare quali sono disponibili; un proxy si può ritenere disponibile se è connesso sia all'attaccante che alla vititma. Quindi si connette al socket in cui il proxy è in ascolto e gli invierà un messaggio in cui specifica l'indirizzo IP della vittima e la metodologia di attacco scelta. Successivamente rimarrà inattesa che il proxy confermi la connessione con la vittima.

Prima di inviare il comando, l'attaccante definirà un thread per ogni proxy. Il codice eseguito nei thread si occuperà di ricevere i dati che il proxy inoltrerà. Dopodichè si inserirà e si invierà il comando alla vittima. Una volta ricevuti tutti i dati, inoltrati dai proxy, li si riordina così da ricavare il messaggio. Finito ciò si reimposteranno le variabili necessarie e se voluto il ciclo continua (tramite l'inserimento di un altro comando). Altrimenti si può decidere di interrompere la comunicazione e in quel caso i proxy ne verranno aggiornati.

3.2 Struttura del Proxy

Il proxy come argomento richiede l'indirizzo IP dell'attaccante; e dopo averlo ricavato tramite il parser, ricaverà il proprio indirizzo IP. Dopodichè definisce un socket in cui rimarra in ascolto per la connessione dell'attaccante. Per esserne sicuro confronta l'indirizzo di chi si è connesso con quello passato e, in aggiunta, controlla se il messaggio ricevuto successivamente corrisponde ad un messaggio di conferma. Nel caso non fosse il socket viene chiuso e si ritorna in ascolto; altrimenti dal messaggio si ricava l'indirizzo Ip della vittima e la metodologia di attacco. Dopodichè, tramite il socket definito, invierà all'attaccante il messagigo di conferma della connessione. In questo messaggio indicherà l'indirizzo della vititma ricevuto oltre al proprio indirizzo IP.

NB nel caso il proxy e l'attaccante combaciassero, in questo caso non verrà definito alcun socket ma, come per l'attaccante, si ricaveranno i dati necessari dal file di configurazione.

Per la connessione con la vittima, il proxy comunicherà tramite pacchetti ICMP; quindi, prima di inviare alcun dato, imposta un thread il cui scopo è analizzare il traffico e catturare il pacchetto che la vittima manderà per confermare la connesisone. Se questo viene fatto dopo, il pacchetto potrebbe andare e perso. Dopo aver fatto partire il thread, si procederà codificando l'attacco scelta nel campo identifier del messaggio ICMP Echo Request mentre nel payload verrà ionserito il messaggio che richiede la connessione. Infine si aspetta che il thread termini e ritorni lo stato della connessione con la vititma. Dopodichè si procede ad aggiornare l'attaccante sul risultato della connesisone che si ha con la vittima e, nel caso il risultato sia negativo il programma viene terminato (siccome non potrà essere utilizzato per l'inoltro delle informazioni). Se invece si è stabilita una connessione con la vittima, si rimarra in attesa di messaggi da aprte dell'attaccante.

A questo punto i messaggi che un proxy può ricevere dall'attacante sono tre: nel primo caso, il pacchetto indicherà il comando che il proxy dovrà inoltrare alla vittima. La seconda tipologia di messaggio invece indica al proxy che non ha ricevuto il comando e che quindi può iniziare subito ad aspettare i dati dalla vittima. L'ultimo

invece indica la volonta, da parte dell'attaccante, di terminare la comunicazione. In questo caso il proxy aggiornerà la vittima della cosa.

Siccome lo scambio di messaggi avviene fra l'attaccante e il proxy, l'invio e la ricezione avverrà tramite il socket definito all'inizio della comunicazione. Tuttavia questo non varrà per l'inoltro del comando; infatti la comunicazione avverrà fra la vittima e il proxy e per questo si invieranno (e riceveranno) i dati in base alla tipologia di attacco definita. Inoltre non essendo presente alcun tipo di socket fra le due entità, il thread che si occuperà di ricevere i dati dovrà partire prima di inviare il comando.

Infine quando il thread avrà ricevuto i dati dalla vittima, si procederà ad inoltrarli all'attaccante così come li si è ricevuti.

NB nel caso il proxy e l'attaccante combaciassero, in questo caso non si aspetterà alcun comando lo lo si richiederà in input. Inoltre i dati non verranno inoltrati.

3.3 Struttura Vittima

Quando si eseguirà il programma, dovranno essere definitiil numero di proxy necessarri. Ciò indicherà il numero minimo di proxy necessari che serviranno per l'esecuzione dell'attacco. Successivamente si ricaverà il prorpio indirizzo IP e si andrà in attessa dei proxy che si vorranno connettere.

Questo viene fatto rimanendo in attessa e monitornado il flusso di rete, filtrando i messaggi ICMP destinati alla vittima e al cui interno sia presente un messaggio di richeista di connessione. Se viene ricavato un messaggio del genere, si risponde con un messaggio di conferma della connessione e l'indirizzo IP del mittente viene inserito nella lista dei proxy connesisone. Da questo messaggio la vittima ricaverà la metodologia di attacco scelta. La vititma smetterà di monitorare il traffico finche o non verrà raggiunto il nuemro minimo di proxy o finche il tmier non scade. In ogni caso la lista dei proxy connessi verrà controllata per verificare se ci sia alemo un proxy connesso. Se così non fosse (e il numero risulti zero) termina immediatamente la connessione altrimenti se il numero è inferiore a quello richiesto, chiese se si vuole procedere ciò nonostante.

Avendo la tipologia di attacco con cui il comando verrà inviato, la vittima si metterà in ascolto; in attesa che uno dei proxy lo mandi. Una volta ricevuto, aprirà una shell

dove poterlo eseguire e ne ricaverà i dati. I dati ricavati non saranno solo quelli legato all'output ma anche quelli legati ad eventuali errori che possono essere accaduti durante l'esecuzione. Questi dati saranno poi inoltrati ai proxy.

Per ogni porxy connesso, si definirà una lista indicante i dati che dovrà ricevere. Il modo in cui verranno distribuiti è sequenziale. Un esmepio è quando, in un gioco di carte, si distribuisce a tutti i giocatori una singola carta fino ad esaurimento del mazzo, piuttosto che distribuire ad ogni giro X carte a ciascun giocatore. Quindi il primo proxy riceverà la prima porzione, il seocndo proxy la seconda e così via fino a quando non si è ritornati al proxy di partenza e i dati da mandare non siano terminati. L'ulyimo messaggio che verrà inviato ai proxy sarà quello in cui si indica che tutti i dati sono stati mandati.

4 Terza Parte

5 Come funziona RITA

Una costante assoluta su cui RITA fà affidamento per rilevare i malware, è che dovranno chiamare "casa". Da questo presupposto; analizzando il traffico di rete, rilevare le chiamate C2 indipendentemente dalla piattaforma.

La persistenza è l'attributo chiave che si ricerca quando si analizza sistemi compromessi. RITA cerca gli indicatori principali relativi a questa persistenza. Viene fatto acquisendo i log di connessione di Zeek e tramite l'utilizzo dell'analisi comportamentale identifica i sistemi potenzialmente compromessi.

Durante la fase di analisi, suddivide questi indicatori principali di persistenza in moduli separati, per poi visualizzare i risultati in base a un indicatore specifico. La gravità è determinata da quattro fattori.

- Analisi del timestamp: coerenza nell'intervallo di tempo tra le sessioni.
- Analisi della dimensione dei dati: consistenza nella dimensione tra le sessioni.
- Analisi dell'istogramma: frequenza delle sessioni nel tempo.
- Analisi della durata: persistenza del canale all'interno dell'intervallo di tempo.

Modulo	**Descrizione**
Beacons	Sono impulsi di comunicazioni fra due host. Mentre alcuni sono innocui, un sistema compromesso li userà per esfiltrare dati o ricevere istruzioni. RITA identifica ed etichetta quattro differenti tipi di beacon.

Strobes	Le coppie (host interno, host esterno) che attivano una nuova connessione (una o più volte al secondo) vengono chiamate Strobe. Poiché si tratta di segnali basati sulla frequenza della comunicazione, non vengono valutati e vengono invece presentati come un elenco ordinato in base al numero di connessioni.
Connessioni lunghe	Consentono a un sistema compromesso di ricevere comandi ed esfiltrare dati senza dover effettuare costantemente il check-in con il server. Questi tipi di sessioni creano meno voci nei log; ciò le rende difficili da rilevare.
Threat Intel	I feed di Threat Intelligence contengono informazioni su host potenzialmente dannosi. Si basano su informazioni di attacco accumulate da varie fonti. I risultati mostrano l'elenco delle corrispondenze dannose suddivise in tre categorie: nomi host, IP contattati in uscita e IP esterni che hanno avviato una connessione.

Tabella 21: Moduli principali di RITA

Se si trova un risultato sospetto, si possono utilizzare i log di Zeek per raccogliere ulteriori indizi sugli host o sulle connessioni. Inoltre la maggior parte dei moduli ha soglie che possono essere modificate tramite il file di configurazione. I moduli beacon consentono anche di personalizzare il peso di ogni sottopunteggio sul punteggio finale.

5.1 Configurazione di RITA

Prima di iniizare a testare RITA, è necessario configurarlo. Questo passaggio è fondamentale per il corretto funzionamento del sistema. Senza una configurazione appropriata, il sitema non avrebbe rilevato l'attaccante in modo efficace.

La configurazione di RITA avviene tramite il file di configurazione **config.hjson** che si trova nella cartella **/etc/rita/**.

5.1.1 Configurazione del campo *filtering*

Il campo imposta i filtri che RITA utilizzerà durante lìanalisi dei log di Zeek.

Sottocampo	**Descrizione**
internal_subnets	Specifica gli indirizzi IP che si considereranno interni alla rete. Siccome l'attaccante e la vittima sono all'interno della stesse rete; nel campo è stato inserito solamente l'indirizzo IP della vititma.
always_included_subnets	Specifica le reti che RITA dovrà sempre includere nell'analisi. Siccome come rete interna si è specificato solo la vittima, il campo verrà lasciato al suo valore di default.
filter_external_to_internal	Se impostato a true, fa sì che RITA ignori tutte le comunicazioni che avvengono da un host esterno ad uno interno. In questo caso si è impostato a false, in modo che RITA analizzi anche queste comunicazioni.

Tabella 22: Sottocampi di *filtering*

```
"filtering": {
    "internal_subnets": [
        "192.168.1.3"
        #"192.168.1.11"
```

```
    ],
    "always_included_subnets": [],
    //Ignores external host to internal host communication
    "filter_external_to_internal":false
},
```

Listing 5: Configurazione del campo *filtering*

5.1.2 Configurazione del campo *scoring*

In questo campo si può impostare come RITA assegnerà i punteggi ad ogni tipologia di comunicazione sospetta.

Per testare i risultati delle configurazioni, sono stati effettuati due test principali: il primo test prevede una comunicazione reale tra l'attaccante e la vittima tramite il Covert Channel ICMP che tuttavia risulta essere troppo breve. Il secondo test invece prevede un ping continuo tra i due host in cui si simula uno scambio di messaggi.

Test	**Descrizione**
ICMP Test	Comunicazione reale fra l'attacante e la vittima tramite il Covert Channel ICMP. La comunicazione avviene con l'attaccante che chiede alla vititma il contenuto di un file; una volta che la vittima ha inviato il contenuto del file, l'attaccante procede a richiedere un nuovo file. I file di testo hanno sempre una dimensione crescente.
600 ping	Si ha la **vittima** che pinga ogni 2 secondi l'attaccante per un totale di 600 ping. Un volta finito, l'**attaccante** invece pinga la vittima ogni 2 secondi per un totale di 200 ping. Questo scambio di messaggi è stato ripetuto in totale 4 volte e simula uno scambio di messaggi tra i due host.

Tabella 23: Test effettuati con RITA

Prima configurazione

La configurazione iniziale del campo *scoring* è quella di default. Ovvero quella che RITA utilizza appena installato.

Comunicazione Vittima to Attaccante			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test	None	16%	19m 10s
600 ping	None	42.90%	1h 22m 2s

Tabella 24: Risultati dei test effettuati con RITA

Comunicazione Attaccante to Vittima			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test	None	22%	1m 9s
600 ping	None	41.60%	34m 10s

Tabella 25: Risultati dei test effettuati con RITA

Seconda configurazione

Il sottocampo **beacon** definisce i parametri che RITA utilizzerà per definire se una connessione è un beacon e in caso il punteggio daassegnargli. Al suo interno si possono settare diversi parametri che influenzano il calcolo del punteggio finale da associare alla connessione. Il valore del punteggio è dato dalla media pesata di 4 sottocampi; il valore di default per ognuno di essi è 0.25, tuttavia possono essere modificati in base alle esigenze; a patto che la somma di tutti i pesi sia alla fine uguale a 1.

Qui9ndi. siccome i test effettuati con la configurazione di base non hanno generato alcun avvertimento sulla presenza di un anomalia, si è deciso di modificare le soglie per il calcolo del punteggio finale.

Siccome in questo tipo di attacco, la dimensione dei pacchetti rimangono costanti; i dati scambiati non sono così rilevanti. Si è deciso quindi di diminuire il peso del campo **datasize_score_weight** a **0.15**. Così come anche quello di **duration_score_weight**, che è stato diminuito a **0.1**.

Invece **timestamp_score_weight** è stato aumentato a **0.4**. Mentre il campo **histogram_score_weight** avrà un peso pari a **0.35**.

Sottocampo	**Descrizione**
timestamp_score_weight	Parametro utilizzato per influenzare il modo in cui i timestamp vengono considerati nel punteggio delle potenziali minacce. Questo peso regola l'importanza dei pattern basati sul tempo, nell'analisi complessiva delle minacce. Pattern basati sul tempo sono spesso indicativi di attività dannose come le comunicazioni di Comando e Controllo (C2).
datasize_score_weight	Utilizzato per regolare il peso o l'importanza della dimensione dei dati nel calcolo dei punteggi per l'analisi delle minacce. Modificandolo, è possibile enfatizzare o ridurre il ruolo della dimensione dei dati nel punteggio complessivo delle minacce.
duration_score_weight	Parametro utilizzato per regolare il peso del punteggio di durata della connessione nel calcolo dei punteggi. Modificandolo, è possibile influenzare l'impatto della durata delle connessioni sul punteggio complessivo. Un peso maggiore: aumenta l'importanza della durata della connessione nel calcolo del punteggio beacon. Un peso minore: riduce l'impatto della durata della connessione, dando maggiore importanza ad altri fattori come l'intervallo o il volume dei dati.

histogram_score_weight	Il punteggio dell'istogramma misura la regolarità nei tempi di comunicazione tra due host utilizzando un istogramma dei tempi di interarrivo (ritardi tra pacchetti o flussi). Da questo calcola quanto sia "grande" o "ripetitiva" la distribuzione: se una coppia di host scambia pacchetti a intervalli molto regolari (ad esempio ogni 2 secondi), l'istogramma presenterà dei picchi significativi e il punteggio dell'istogramma sarà alto. Se invece la comunicazione avviene a intervalli casuali o variabili, l'istogramma è piatto, assegnando un punteggio basso. Una varianza inferiore, ceh aumenta il punteggio dell'istogramma, suggerisce che l'host stia effettuando del beaconing a un sistema remoto secondo una pianificazione prevedibile, una caratteristica comune nel traffico di comando e controllo.

Tabella 26: Sottocampi di *beacon* relativi al punteggio

```
"timestamp_score_weight": 0.4,
"datasize_score_weight": 0.15,
"duration_score_weight": 0.1,
"histogram_score_weight": 0.35,
```

Listing 6: Seconda configurazione dei campi in *beacon*

Comunicazione Vittima to Attaccante			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test 2	None	19.10%	19m 10s
600 ping 2	None	43.60%	1h 22m 2s

Tabella 27: Risultati dei test effettuati con RITA

Comunicazione Attaccante to Vittima			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test 2	None	22.7%	1m 9s
600 ping 2	None	41.50%	34m 10s

Tabella 28: Risultati dei test effettuati con RITA

Terza configurazione

Dato il miglioramennto nella seconda configurazione si procede nel definire i parametri mancanti, oltre che nel raffinare ulteriormente i parametri già impostati.

Si è provato ad abbassare il numero minimo di connessioni univoche richieste per analizzare una connessione come beacon. Tuttavia se il parametro risulta minore di 4 RITA non convalid a la configurazione.

Si raffinano poi il peso da assegnare agli score per enfatizzare la regolarità dei tempo e gli istogrammi. Infatti gli attacchi Covert Channel ICMP tipicamente inviano pacchetti altamente regolari con dimensioni e strutture di payload quasi identiche.

Si raffina poi la sensibilità alla durata. I beacon ICMP spesso si verificano in raffiche più brevi di 6 ore, quindi si è abbassata la soglia entro cui iniziare a calcolare il punteggio. Così da permettere di calcoalre il punteggio anche delle connessioni più brevi. Infine si è abbassata anche la soglia ideale per ottenere il punteggio massimo ($duration\_consistency\_ideal\_hours\_seen$).

Si è poi raffinato anche la sensibilità dell'istogramma. Si è deciso di rendere la rilevazione della modalità più sensibile (con bucket più stretti si avrà meno tolleranza) così che da distinguere chiarmanete gli intervalli ICMP.

Infine si sono abbassati leggermente i punteggi delle soglie così che anche le comunicazioni ICMP più brevi appaiano nell'analisi almeno nelle categorie "bassa/media".

Sottocampo	Descrizione
unique_connection_threshold	Il campo indica il minimo di connessioni necessarie affinche RITA consideri la connessione come una beacon. Due host con un numero di connessioni inferiore a questo non verranno analizzati.

Tabella 29: Sottocampi di *beacon* relativi al numero minimo di connessioni

Sottocampo	Descrizione
duration_min_hours_seen	Il numero di ore visualizzate in una rappresentazione delle connessioni di un beacon deve essere superiore a questa soglia affinché venga calcolato un punteggio di durata complessivo.
duration_consistency _ideal_hours_seen	Questo è il numero minimo di ore visualizzate in una rappresentazione grafica delle connessioni di un beacon affinché il sottopunteggio di coerenza della durata raggiunga il 100%.
histogram_mode_sensitivity	La variabile controlla la dimensione del bucket per il raggruppamento delle connessioni. Questo valore è espresso come percentuale del numero massimo di connessioni. Aumentando questa variabile, l'algoritmo diventa più tollerante alle variazioni.
histogram_bimodal _outlier_removal	Questo è il numero di bucket che possono essere considerati valori anomali ed esclusi dal calcolo.
histogram_bimodal _min_hours_seen	Questo è il numero minimo di ore visualizzate in una rappresentazione delle connessioni di un beacon prima che venga utilizzato il punteggio del sottopunteggio bimodale.

Tabella 30: Sottocampi di *beacon*

```
"unique_connection_threshold": 4,

// Timing regularity = strongest signal
"timestamp_score_weight": 0.45,
// Repeated interval/size histogram shapes
"histogram_score_weight": 0.30,
// Consistent ICMP payload size
"datasize_score_weight": 0.20,
// De-emphasize total connection time
"duration_score_weight": 0.05,

"duration_min_hours_seen": 2,
"duration_consistency_ideal_hours_seen": 6,

//Tighter clustering
"histogram_mode_sensitivity": 0.03,
"histogram_bimodal_outlier_removal": 1,
// Match duration window
"histogram_bimodal_min_hours_seen": 6,

"score_thresholds": {
    "base": 40,
    "low": 65,
    "medium": 85,
    "high": 100
}
```

Listing 7: Terza configurazione dei campi in *beacon*

Comunicazione Vittima to Attaccante			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test 2	None	24.2%	19m 10s

600 ping 2	None	56%	1h 22m 2s

Tabella 31: Risultati dei test effettuati con RITA

Comunicazione Attaccante to Vittima			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test 2	None	29.6%	1m 9s
600 ping 2	None	54.4%	34m 10s

Tabella 32: Risultati dei test effettuati con RITA

Quarta configurazione

In questa configurazione si vanno a modificare i parametri esterni al sottocampo *beacon*, ma sempre all'interno del campo *scoring*. Tuttavia non si sono raggiunte modifiche sostanziali nei risultati finali.

Sottocampo	**Descrizione**
long_connection_score_thresholds	controls how RITA buckets long-lived connections into base/low/medium/high tiers by duration. Lowering these makes shorter connections count as "long" (so the "long connection" detector will flag them sooner). This can help if your ICMP tests are shorter than the defaults and you want them assessed as notable long connections.

c2_score_thresholds	applies to DNS / domain-subdomain analysis for C2 detection. Not directly relevant to ICMP, but lowering these will make small-scale domain churn look more suspicious (useful if you want more sensitivity to metadata-based C2).
strobe_impact & threat_intel_impact	already set to "high". Any flow that RITA flags as a strobe (fast repeated connections) or that matches threat intel will be escalated as high — keep these at high to ensure flagged ICMP flows are prioritized.

Tabella 33: Sottocampi di *beacon*

```
"long_connection_score_thresholds": {
    "base": 1800, // 30 min
    "low": 7200     // 2 hours
    "medium": 14400, // 4 hours
    "high": 28800, // 8 hours
},
```

Listing 8: Terza configurazione dei campi in *beacon*

Comunicazione Vittima to Attaccante			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test 2	None	24.2%	19m 10s
600 ping 2	None	56%	1h 22m 2s

Tabella 34: Risultati dei test effettuati con RITA

Comunicazione Attaccante to Vittima			
Test	**Risultato**	**Beacon**	**Connessione**
ICMP test 2	None	29.6%	1m 9s
600 ping 2	None	54.4%	34m 10s

Tabella 35: Risultati dei test effettuati con RITA

5.1.3 Risultati della configurazione del campo *scoring*

Comunicazione Vittima to Attaccante			
Test	**Risultato**	**Beacon**	**Connessione**
Prima configurazione			
ICMP test	None	16%	19m 10s
600 ping	None	42.90%	1h 22m 2s
Seconda configurazione			
ICMP test 2	None	19.10%	19m 10s
600 ping 2	None	43.60%	1h 22m 2s
Terza configurazione			
ICMP test 2	None	24.2%	19m 10s
600 ping 2	None	56%	1h 22m 2s
Quarta configurazione			
ICMP test 2	None	24.2%	19m 10s
600 ping 2	None	56%	1h 22m 2s

Tabella 36: Risultati dei test effettuati con RITA

Comunicazione Attaccante to Vittima			
Test	**Risultato**	**Beacon**	**Connessione**

Prima configurazione			
ICMP test	None	22%	1m 9s
600 ping	None	41.60%	34m 10s
Seconda configurazione			
ICMP test 2	None	22.7%	1m 9s
600 ping 2	None	41.50%	34m 10s
Terza configurazione			
ICMP test 2	None	29.6%	1m 9s
600 ping 2	None	54.4%	34m 10s
Quarta configurazione			
ICMP test 2	None	29.6%	1m 9s
600 ping 2	None	54.4%	34m 10s

Tabella 37: Risultati dei test effettuati con RITA