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*').

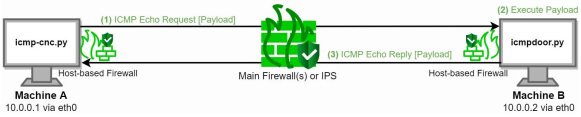


Figura 1

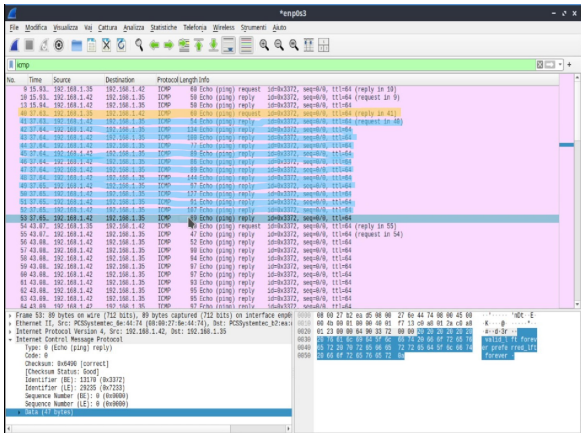


Figura 2: Traffico relatico al comando *ls -s*

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).

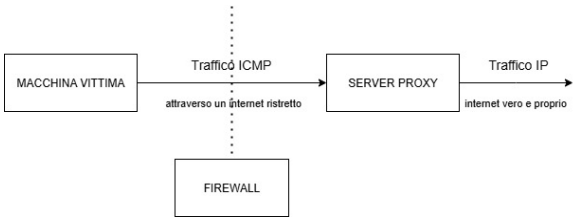


Figura 3: Architettura generale *icmp tunnel*

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. La loro esistenza quindi rappresenterà un problema che spesso rimane non notato dagli amministratori o dai tipici strumenti di monitoraggio. Da notare inoltre che qualsiasi risorsa condivisa può essere utilizzata come canale nascosto. Questo permetterà ai Covert Channel di esistere in qualsiasi sistema.

Siccome un covert channel deve poter aggirare i controlli in maniera nascosta; avrà bisogno di alcune caratteristiche. tuttavia, uno dei maggiori problemi nell'implementazione di un canale nascosto è l'uso eccessivo delle risorse. La necessità cardine, è quella di riuscire a trasmettere informazioni mantenendo conforme lo stato del sistema; così da rendere il canale **indistinguibile** rispetto alle altre risorse presenti nel sistema. dalla risorsa sfruttata e di conseguenza invisibili ai sistemi di monitoraggio. In generale il canale è incorporato in operazioni di sistema legittime per poter mascherare la trasmissione dei dati.

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

2.1 Implementazione

In un Covert Channel ICMP verranno utilizzati messaggi ICMP (di solito richieste e risposte Eco) per nascondere i dati all'interno di campi che normalmente vengono ignorati o non monitorati. Il canale sarà possibile siccome il protocollo ICMP (Internet Control Message Protocol) consente agli attaccanti di trasferire dati aggirando le politiche di sicurezza ed eludendo il rilevamento. Ciò è possibile siccome è un protocollo che viene utilizzato, in sinergia con il protocollo IP, per la diagnostica della rete, per la segnalazione di errori, per ottenere informazioni tramite la messagistica di controllo e per la risoluzione dei problemi nelle reti. Permette quindi di informare il mittente sui problemi di rete , aiutare nella risoluzione dei problemi di rete e gestisce la congestione della rete oltre gli aggiornamenti di routing (in alcuni casi)

Sebbene sa essenziale per la diagnostica di rete e la segnalazione degli errori, può essere comunque utilizzato in modo improprio per degli attacchi o per la ricognizione della rete (network reconnaissance). Questi messaggi verranno usati per creare una comunicazione fra una vittima e il suo attaccante, in cui potrebbero essere presenti in caso dei proxy intermedi.

Nei seguenti capitoli verrà illustrato come può essere sfruttato per la creazione di un Covert Channel. Principalmente le risorse condivise, e manipolate, saranno i pacchetti ICMP e i dati verranno codificati all'interno dei campi presenti. Altre codifiche, che non includono l'insserimento di onfromazioni nei campi, saranno quelle relative alla distanza di tempo oppure alla tipologia di messaggio ricevuto. Nel primo caso in base alla distanz adi tempo fra un pacchetto e l'altro, viene codificato l'informazione. Nel secondo caso la codifica dei dati verrà associata alla particlare tipologia di messaggio ricevuto.

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**: Opzionale, può contenere parte del pacchetto IP originale che ha causato l'errore.

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

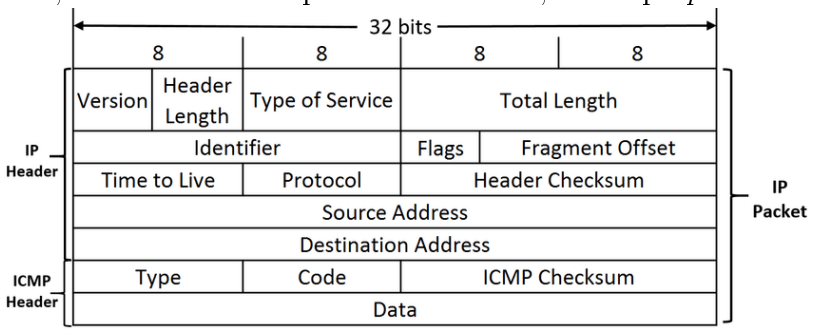


Figura 4: Struttura pacchetto ICMPv4/IPv4

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. Di seguito la struttura delle varie tipolgie.

2.2.1 Destination Unreachable

Un messaggio *Destination Unreachable*, viene inviato quando la rete specificata (nel campo di destinazione) è irraggiungibile. E quindi il pacchetto non può essere recapitato. Altri possibili casi per cui verrà inviato è se l'host è irraggiungibile o se il protocollo indicato o la porta di destinazione specificati non sono attivi. Il messaggio non verrà (e non dovrà essere) generato se un pacchetto viene scartato a causa della congestione del traffico. Inoltre un nodo che riceve un messaggio *Destination Unreachable* deve notificare l'evento al processo di livello superiore (se il processo in

questione può essere identificato).

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=3 (1B)	Code=0-5 (1B)	Checksum (2B)
Unused (4B)		
Internet Header + 64 bits of Original Datagram ($\geq$ 21B)		

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.

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

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.

2.2.2 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 scarterà il datagramma e notificherà l'host sorgente della cosa. Ciò indica un loop nel routing o un valore iniziale della quantità di hop possibili troppo basso. Altri casi possibili in cui questo messaggio può avvenire è quando un host che riassembla un datagramma frammentato , non riesce a completare il riassemblaggio a causa di frammenti mancanti entro il proprio limite di tempo.

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=11 (1B)	Code=0-1 (1B)	Checksum (2B)
Unused (4B)	
Internet Header + 64 bits of Original Datagram (≥ 21B)	

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.

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=3 (1B)	Code=0-1 (1B)	Checksum (2B)
Unused (4B)	
As much of invoking packet as possible without	
the ICMPv6 packet exceeding the minimum IPv6 MTU (≥ 0B)	

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.

2.2.3 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 potrà notificare l'host sorgente della cosa indicando il tipo e la posizione del problema. IL messaggio viene inviato solo se l'errore ha causato lo scarto del pacchetto.

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

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 (1B)	Code=0 (1B)	Checksum (2B)
Pointer (1B)	Unused (3B)
Internet Header + 64 bits of Original Datagram (≥ 21B)	

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.

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

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.

2.2.4 Source Quench

Questa tipologia viene usata quando il gateway gateway scarta un pacchetto. In questo caso invierà un messaggio di Source Quench all'host mittente. Una motivazione per cui un gateway può scartare un pacchetto è se non ha lo spazio necessario nel buffer per mantenere in coda i pacchetti; che dovranno essere inoltrati alla rete successiva, la quale farà parte della rotta per la rete di destinazione. Inoltre un host di destinazione può inviare un messaggio di questo tipo, anche nel caso in cui i datagrammi arrivino troppo velocemente per poter essere elaborati. Ciò indicherà una richiesta di ridurre la velocità di invio dei pacchetti nel traffico.

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

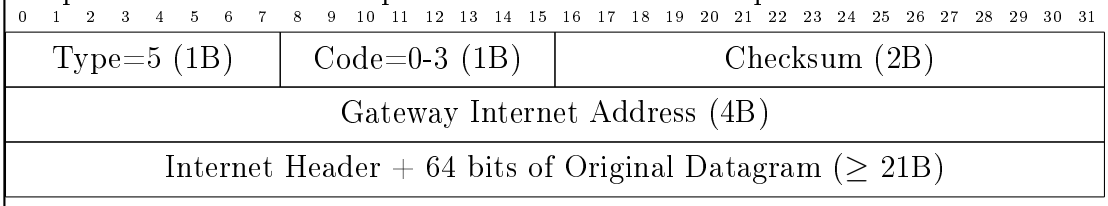
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Type=4 (1B)	Code=0 (1B)	Checksum (2B)
Unused (4B)		
Internet Header + 64 bits of Original Datagram ($\geq$ 21B)		

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.

2.2.5 Redirect

La tipologia *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. 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.

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



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).

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.

2.2.6 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. Questo perchè chi risponde dovrà restituire gli stessi valori ricevuti nel messaggio integralmente e senza modifiche. Il mittente può inserire nel campo del payload quanti dati desidera, a patto che risultino minori di 65.000 bytes. Questo perchè nell'intestazione IP, il campo relativo alla lunghezza totale del pacchetto è composto da 16 bit. Sempre nel messaggio, i campi identificatore e numero di sequenza possono essere utilizzati dal mittente per facilitare l'abbinamento delle risposte con le richieste.

NB Una limitazione inferiore sui dati potrà essere definità dalla massima capacità

di trasporto dei collegamenti. Nel caso la dimensione del messaggio la superasse, il pacchetto dovrà essere frammentato. In media questo valore si attesta sui 1400 bytes.

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 (1B)	Code=0 (1B)	Checksum (2B)
Identifier (2B)		Sequence Number (2B)
Data ... ($\geq$ 0B)		

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 (1B)	Code=0 (1B)	Checksum (2B)
Identifier (2B)		Sequence Number (2B)
Data ... ($\geq$ 0B)		

2.2.7 Timestamp Request / Timestamp Reply

Viene usata 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. 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.

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 (1B)	Code=0 (1B)	Checksum (2B)
Identifier (2B)		Sequence Number (2B)
Originate Timestamp (4B)		
Receive Timestamp (4B)		
Transmit Timestamp (4B)		
Data ... ($\geq$ 0B)		

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

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

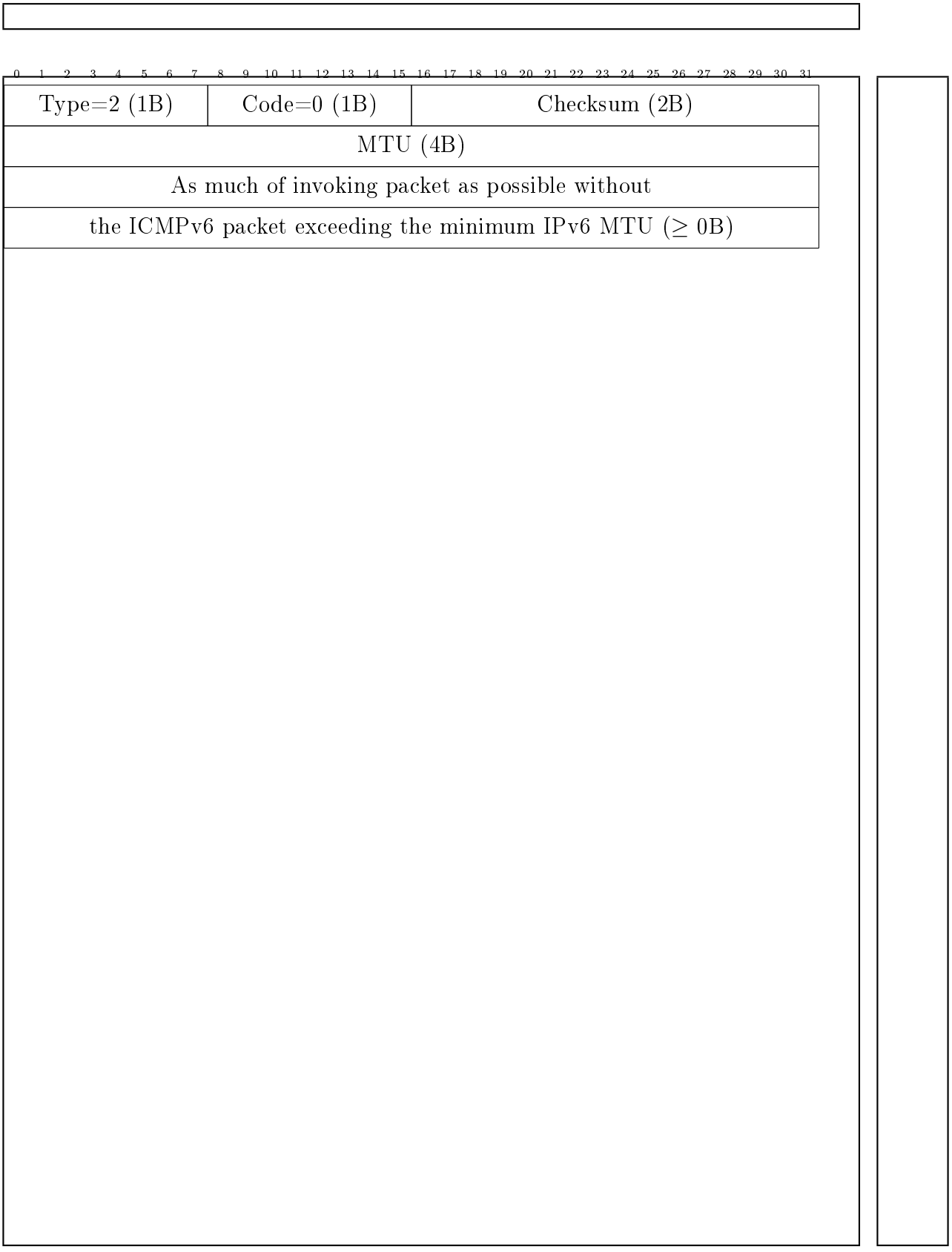
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Type=15 (1B)	Code=0 (1B)	Checksum (2B)
Identifier (2B)		Sequence Number (2B)

2.2.9 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). 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.

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 (1B)
Code=0 (1B)
Checksum (2B)
MTU (4B)
As much of invoking packet as possible without
the ICMPv6 packet exceeding the minimum IPv6 MTU (≥ 0B)

2.3 Implementazione di 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 implementarli sarà quindi presente un metodo di codifica, usato dal mittente, che ricaverà il tempo associato ad un dato e poi aspetterà tale intervallo di tempo. Per esempio se al dato binario 1001 fosse associato il tempo 3, il programma prima leggerà il dato e poi aspetterà tre secondi.

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

Listing 1: Pseudocodice di un Timing Covert Channel

Invece per ricevere le informazioni, il destinatario rimane in attesa dei pacchetti che gli verranno inviati. Successivamente ricaverà l'intervallo di tempo fra il pacchetto precedente e quello corrente e poi ricaverà il valore associato. Per esempio se l'intervallo di tempo risultassedi 6 secondi, il destinatario controllerà a chi è associato tale tempo (in questo caso diciamo il dato binario 1101).

```
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 di un Timing Covert Channel

2.3.1 Covert Channel Temporale tramite pacchetti ICMP Echo

Siccome deve essere presente una maniera per codificare e decodificare i tempi associati ai dati, si è definita in un primo momento la seguente funzione:

$$\text{tempo\_base} + \text{index} * (2 * \text{distanza\_tempi}) \tag{1}$$

Il **tempo di base** indica il minimo dei secondi da aspettare per ogni tempo possibile. e l'indice indicherà l'indicizzazione associata alla codifica del dato. Invece la distanza è la differenza minima di tempo che tutti i tempi calcoalti dovranno avere; il suo scopo è quello di non avere due tempi che si sovrappongano, siccome in quei casi il dato potrebbe essere decodificato erroneamente.

Quindi, siccome si sta lavorando a livello di bit, dal dato estrarremo i singoli bit che poi manderemo, codificandoli tramite l'intervallo di tempo associato. Nel frattempo il destinatario rimarrà in ascolto e all'arrivo di un pacchetto calcolerà l'intervallo di tempo trascorso e ricaverà il dato associato ad esso.

NB all'inizio si manderà un pacchetto per indicare al destinatario l'inizio della comunicazione. Questo serve e impostare il tempo di inizio; se non lo si determina, il destinatario all'inizio non riuscirà a calcoalre l'intervallo di tempo fra il pacchetto corrente e quello precedente. E quindi non si riuscirà a decodificare il primo bit.

2.3.2 Timing Channel a 8 bit

Uno sviluppo effettuato è stato l'amplamento dei bit trasportabili. Siccome la quantità di dati risulta non congrua alla quantità di tempo speso; si è sviluppato l'approccio presente in ICMP Exfill. In esso viene mandato l'intero byte e il tempo da aspettare è determinato da esso. Il byte rappresentante il carattere, verrà quindi letto come un numero intero ed usato per determinar il tempo di delay. Successivamente si è cercato di ridurre la quantità di tempo necessaria per mandare i dati. La nuova fun-

zione cercerà di normalizzare il tempo associato al dato fra un valore minimo e uno massimo.

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

Questo permetterà di inviare un magigore numero di informazioni in un tempo minore. Il lato negativo è una maggiore possibilità di errore: se il computer che riceve i dati subisce un delay, la possibilità di decodificare il carattere sbagliato è maggiore rispetto ai metodi che implementavano una distanz adi sicurezza.

2.3.3 Timing Channel con tempi casuali

La versione a 8 bit è stata sviluppata ulteriormente inserendo del rumore al tempo di delay che si aspetta fra ogni pacchetto. Quindi invece di aspettare un tempo costante per lo stesso carattere; si definisce un range di tempo associato in cui il carattere può variare. Per poter sincronizzare il mittente e il destinatario, su quanto tempo si è aggiunto o sottratto a quello di base, si sono sviluppate due strade. La prima è creando un numero randomico e sincronizzare le due entità tramite un seed condiviso. Il lato negativo è che se i numeri non combaciano, perchè un'entità ha effettuato un numero maggiore o minore di estrazioni, il dato associato non potrà essere ricavato in modo appropriato. Il secondo approccio è di mettere il delay randomico aggiunto nel pacchetto stesso o direttamente nel payload o tramite uno dei campi, per esempio il campo identifier.

Siccome gli IDS riescono ad individuare gli schemi temporali, si è definito un range così da poter mandare tempi randomici e non costanti. Ora la codifica di un byte, non è più definito da un singolo, costante intervallo di tempo, ma da un range di intervalli temporali. Quindi se si volesse mandare il dato c, non si aspetteranno più 7 secondi, ma un tempo randomico appartenente al range range $[5, 9]$ (supponendo un range di 2 secondi). Inoltre si è pensato di introdurre anche un tempo (in minuti) che definirà quanto aspettare una volta che si è mandato una certa quantità di byte. Ciò potrà essere usato se lo scambio di informazioni risultasse prolungato; un numero elevato e continuo di pacchetti potrebbe risultare anomalo. 5Inoltre nel livello *IPv6* bisognerà definire lo ScopeId dell'indirizzo di destinazione;

2.4 Storage Covert Channel

Implicano la scrittura di dati su un'area di memoria condivisa accessibile da entrambi i processi. I veicoli saranno tutte quelle risorse che consentono la scrittura (diretta o indiretta) da parte di un processo e la lettura (diretta o indiretta) da parte di un altro. Quindi in uno Storage Covert channel, un processo scrive su una risorsa condivisa mentre un altro processo legge da essa. In questo caso come risosrsa condivisa verranno utilizzate le tipologie di messaggi presenti nel protocollo ICMP. Il mittente codificherà i dati nei campi presenti e il destinatario, una volta ricevuto il pacchetto, li decodificherà. Nelle tabelle sono stati indicati quali tipologie di messaggi verranno sfruttati e, per ogni tipologia di messaggio ICMP, quali campi sono stati utilizzati oltre alla quantità di byte inseribili in un singolo messaggio.

Nel caso dei messaggi Echo Request e Echo Reply, si avranno diverse marianti. Questo perchè il messaggio può trasportare un payload al suo interno. Le varianti implicheranno quindi l'utliizzazione o meno di questo campo. Nel caso lo si utilizzi il payload o conterrà 32 byte di informazione oppure 56 byte; il primo caso verrà usato perchè è il valore di defualt usato nei sistemi Windows il secondo è invece il sistema usato nei sistemi Linux. Anche il TTL varierà, il suo valore sarà 128 nei sistemi windows mentre 64 nei sistemi Linux.

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 3: 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 5: Tipologie di messaggi ICMPv6

2.5 Behavioral Covert Channel

I canali nascosti comportamentali operano estraendo le informazioni osservando il comportamento del sistema durante l'esecuzione di operazioni, piuttosto che attraverso l'accesso diretto ai dati. Operano quindi trasmettendo dati in base all'avvenimento di diversi eventi. In questo caso l'evento che si osserverà è l'arrivo di una tipologia di messaggio ICMP. Siccome oltre alla tipologia si potrà variare il codice inviato; la quantità di messaggi inviabili risutla 7. Ciò permetterà di codificare 4 bit alla volta (siccome $\log_2 16 = 4$) e la tipologia che risulterà inutilizzata, verrà sfruttata per indicare l'inizio e la fine della comunicaizone.

In questo caso l'attaccante itererà il dato byte per byte ed ogni volta lo separerà in due parti uguali mascherando il suo valore binario. Ricavati i primi quattro bit e gli ultimi quattro bit, determinerà quali tipologie di messaggi inviare.

Il destinatario invece si metterà in ascolto nel traffico di rete di un messaggio *Information Request* proveniente dall'attaccante. Una volta ricevuto, memorizzerà le tipologie di messaggi ICMP provenienti dalla sorgente e ogni volta ricava il byte associato alla coppia di messaggi ottenuta. Infine quando rileverà ancora un pacchetto *Information Request* smetterà di monitorare la rete.

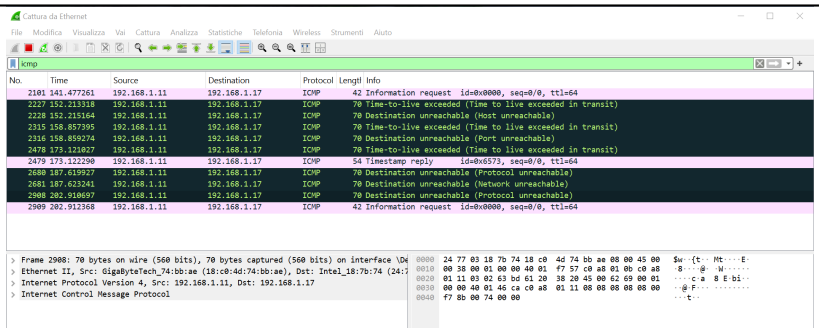


Figura 5: Flusso di un Covert Channel Comportamentale

2.6 Hybrid Covert Channel

Dopo aver analizzato, studiato ed implementato varie tipologie di Covert Channel; procediamo nello sviluppare una loro versione ibrida. Questa versione invieerà un messaggio dopo un determinato intervallo di tempo; ricavato dalla codifica del byte presente nei dati. Dopodichè, ricaverà un ulteriore byte e da esso ricaverà la coppia di messaggi che determinano la sua codifica. In ciascuno di questi messaggi verranno nascoste delle porzioni di dati aggiuntive.

Messaggio ICMP	**Dati trasportabili**
Information Request (15-16)	2 byte
Timestamp Request (13-14)	5 byte
Echo Request (8-0)	2 byte
Redirect (5)	4 byte
Source Quench (4)	8 byte
Parameter Problem (12)	7 byte
Time Exceeded (11)	6 byte
Destination Unreachable (3)	8 byte

Tabella 6: Quantità di byte trasportabili da un messaggio

Quindi il mittente avrà una sequenza di dati da inviare. Da essa ricaverà il primo byte e ricaverà il suo intervallo di tempo. Successivamente prenderà un ultriore byte e lo dividerà in due parti ognuna di 4 bit. ciascuna parte identificherà una tipologia di messaggio. In base ai messaggi ircavati al loro interno codificherà tanti dati quanto la loro capacità.

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

    byte_2=dato[index]
    primi_4bit=(byte_2 & 0b11110000)>>4
    ultimi_4bit=byte_2 & 0b00001111

    tipologia_msg1=getTipologia(primi_4bit)
    capacita_msg=getCapacita(tipologia_msg1)
    data_msg=data[index:index+capacita_msg]
    packet=getPacket(tipologia_msg1, data_msg)
    index+=capacita_msg
    send(packet)

    tipologia_msg2=getTipologia(primi_4bit)
    capacita_msg=getCapacita(tipologia_msg2)
    data_msg=data[index:index+capacita_msg]
    packet=getPacket(tipologia_msg2, data_msg)
    index+=capacita_msg
    send(packet)
```

Listing 3: Pseudocodice Hybrid Covert Channel

Il destinatario invece monitora il flusso dei dati nella rete per rilevare un messaggio *Information Request*. Dopodichè ricaverà l'intervallo di tempo con cui arriverà la coppia di messaggi; da esso ricaverà un byte. Arrivata la coppia di messaggi e ricavata la loro tipologia, oterrà un ulteriore byte definito dalla codifica della coppia di messaggi. Infine ricaverà i dati nasocsti nei due messaggi. Il ricevente continuerà in

questo modo finchè non riceverà un secondo messaggio di tipo *Information Request*; in quel caso la connessione è chiusa e si può terminare di monitoprare il traffico di rete.

```
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
        data.append(getTimeData(delta))
        tempo_precedente=tempo_corrente
    elif received_packet==0:
        print("Secondo pacchetto")
```

```
        tipologia_ora=pacchetto.tipologia
        data.append(
            getTypeData(tipologia_ora, tipologia_prima)
        )
    data.append(getPktData(pacchetto))
```

Listing 4: Pseudocodice Hybrid Covert Channel

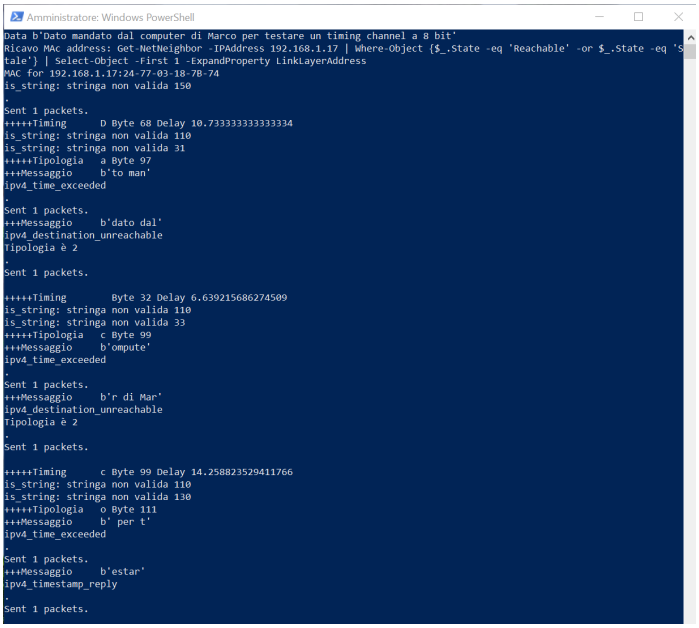


Figura 6: Invio dei dait tramite Hybrid Covert Channel

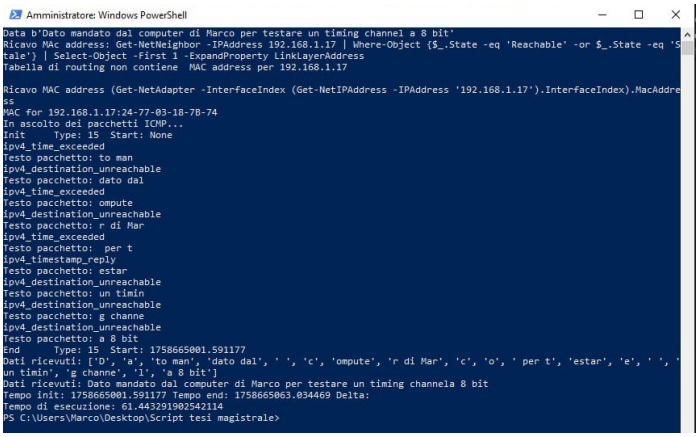


Figura 7: Ricezione dei dait tramite Hybrid Covert Channel

3 Struttura della comunicazione fra le entità

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

Il primo avrà bisogno di un file di configurazione nel quale viene definito l'indirizzo IP della vittima, il metodo di attacco e i proxy utilizzabili per farlo. Definisce il comando che la vittima deve eseguiree dovo averlo inoltrato, tramite i proxy, aspetta i dati relativi ad esso.

Il proxy ha bisongo di sapere qual'è l'indirizzo IP dell'attaccante. Questo perchè deve potersi connettere ad esso ed ottenere oltre all'indirizzo IP della vittima anche il comando da inviare. Stabilisce una connesisone sia con l'attaccante che con la vittima e si occupa princiaplemnte di inoltrare i dati che le due entità si inviano.

L'ultima entità, la vittima, aspetta le connessioni da uno o più proxy e una volta ricevuto il comanda, rimanda i dati ricavati dalla sua esecuzione.

Il flusso di come avviene la comunicazione fra le entità è definito dal diagramma.

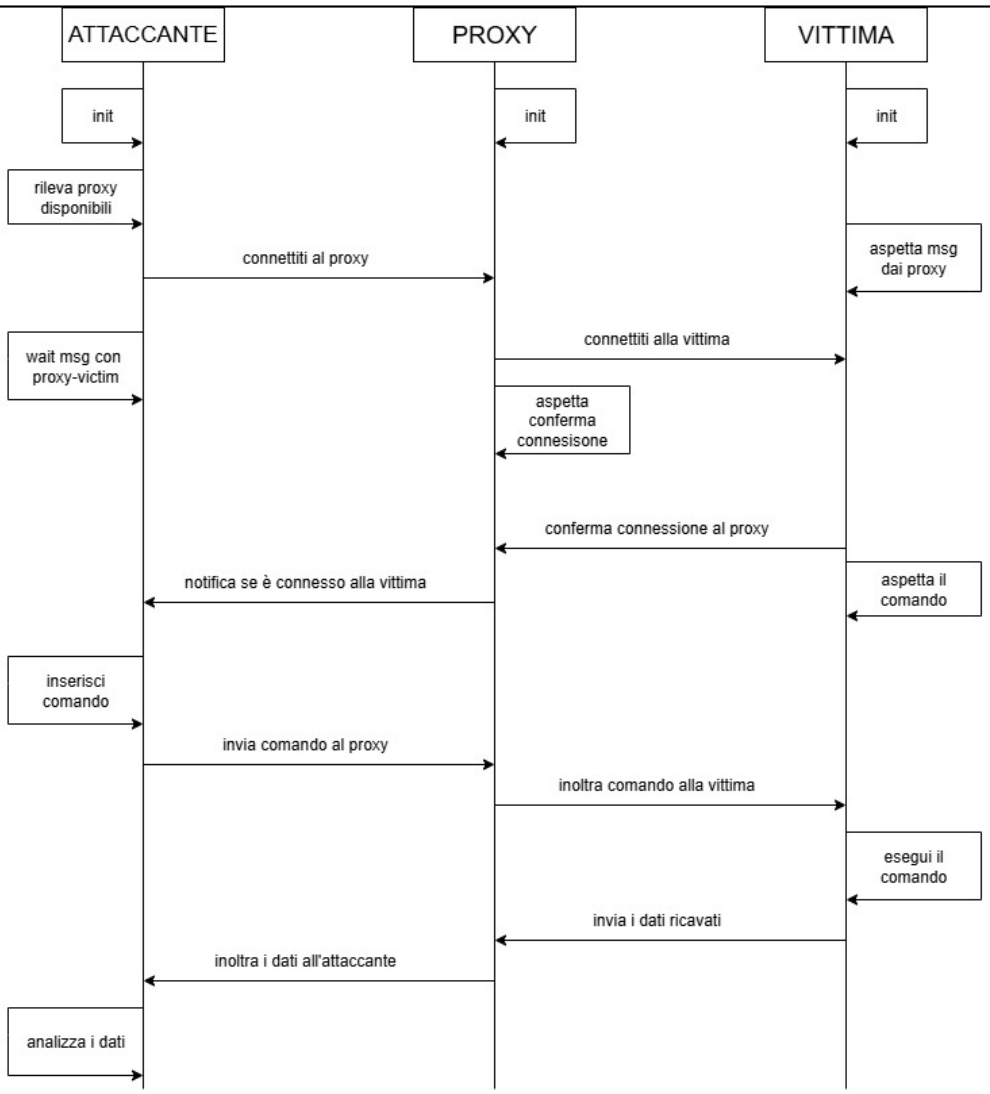


Figura 8: Diagramma del flusso di comunicazione fra le entita

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.

Per la comunicazione l'attaccante potrà usare uno o più proxy per comunicare con la vittima.

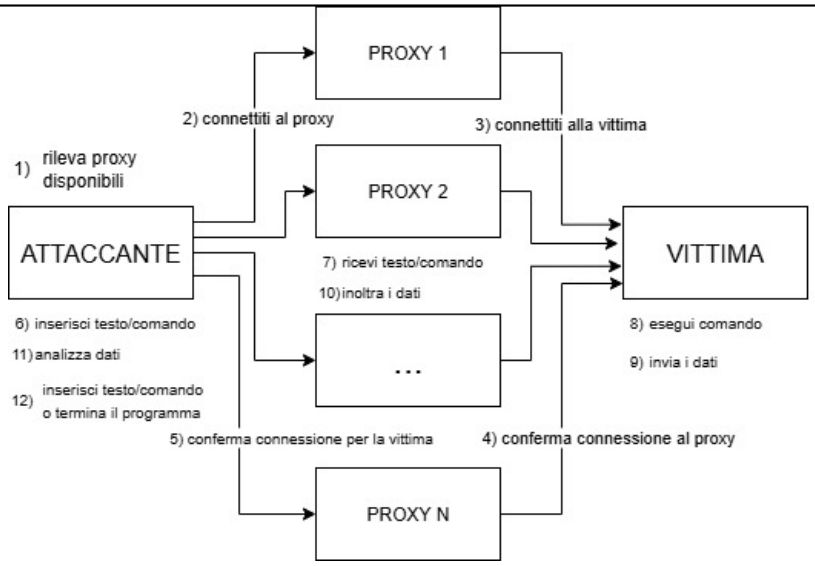


Figura 9: Struttura delle entità presenti e come dialogano

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.

3.1 Struttura dell'attaccante

Quando si esegue il programma, tramite linea di comando, devono essere passati degli argomenti; in particolare verrà utilizzato un parser per rilevare se è stato inserito un path per il file di configuraizone da analizzare. In questo file JSON si specificherà l'indirizzo IP della vittima, la metodologia di attacco e la lista dei proxy a cui ci si vuole connettere. Un ulteriore dato che si ricaverà è l'indirizzo IP dell'host stesso.

Dopodichè ci si connette ai proxy definiti nella lista precedente per verificare quali sono disponibili. Un proxy si può ritenere disponibile se è connesso sia all'attaccante che alla vititma. Quindi si creeerà un socket per ogni proxy connettendosi all'indirizzo e alla porta in cui si sono messi in ascolto e tramite esso, si invia un messaggio di conferma. Questo messaggio specificherà l'indirizzo IP della vittima e la metodologia di attacco scelta. Successivamente rimarrà inattesa del messaggio in cui il conferma la connessione con l'attaccante. Il contentuto sarà l'indirizzo IP della vittima ricevuto

oltre che all'indirizzo IP del proxy stesso. Infine rimane in attesa che il proxy invii un aggiornamento sul suo stato di coneessione con la vittima. Se il risultato delle seguenti operazioni risulterà positivo, il proxy verrà considerato disponibile altrimenti verrà consdiderato non disponibile e di conseguenza scartato (questo significherà che verrà rimosso dalla lista dei proxy connessi).

Prima di inviare il comando, l'attaccante prima definirà un thread per ogni proxy. Il codice eseguito nei thread si occuperà di ricevere i dati che il proxy inoltrerà. Fatti partire i thread; si inserirà e poi si invierà tramite un proxy, il comando alla vittima. Ai restanti proxy verrà notificato di ascoltare direttamente i dati. Dopodichè si aspetta che i thread ricevano i dati e una volta ricevuti 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

4.1 Test

Per testare se RITA fosse in grado di rilevare i Covert Channel implementati; tramite Wireshark si è monitorato il traffico e il risultato è stato salvato in dei file pcap. Tuttavia RITA non supporta questo formato, ma solo i log di Zeek. Quindi tramite Zeek si sono estratti dai file pcap i log necessari che sono stati poi analizzati da RITA.

Per testare se i pacchetti venissero rilevati, si sono inviate tre tipologie di dati: il primo è un testo corto e lineare volto a simulare un comando, il secondo è un testo di lunghezza variabile, mentre il terzo è invece un testo la cui lunghezza incrementa ogni volta. L'odine di grandezza è dei Kilobytes; questo è stato scelto per non appesantire troppo il carico di rete ed evitare lunghi tempi di attesa.

I risultati sono divisi per la tipologia di messaggio inviato e la quantità di dati inviati. All'interno di ogni cella si indicherà il livello di gravità che RITA ha associato alla comunicazione.

Ed il risultato è insoddisfacente. Il riusltato è che RITA non rileva le connessioni ICMP. Sebbene si siano provate diverse tipologie di messaggio e diverse tipologie di messaggi RITA non rileva niente. All'inizio il problema erano i log di Zeek che non moritoravano le connesisoni ICMP; ma dopo aver dockerizzato Zeek così da avere l'ultima versione, il test è risultato Indefinibile. E in questa seconda prova zeek nei file di log dava una cossesisone ICMP.

Si sono quindi provate vari livelli di connesioni, dalle più leggere a quelle che inviavano un costante flusso di dati. Per testare il caso di una comunicazione pesante, si è creato apposta un file contenente 2 MB di dati. Il risultato è ancora Indefinito sicocme i log di Zeek mostrano queste connessione sebbene i database creati da RITA risultino negativi. Data questa ambiguità il riusltato non può essere definito ne positivo ne negativo.

Tipologia Messaggio	**Quantita di byte totali inviati**		
	33 bytes	2,5 KB	20 KB
Information Reply	Non definito	Non definito	Non definito
Timestamp Reply	Non definito	Non definito	Non definito
Redirect	Non definito	Non definito	Non definito
Echo	Non definito	Non definito	Non definito
Source Quench	Non definito	Non definito	Non definito
Parameter Problem	Non definito	Non definito	Non definito
Time Exceeded	Non definito	Non definito	Non definito
Destination Unreachable	Non definito	Non definito	Non definito
Timing Channel 1bit	Non definito	Non definito	Non definito
Timing Channel 2bit	Non definito	Non definito	Non definito

Tabella 7: Quali comunicazioni RITA ha rilevato