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

1 Background

1.1 ICMP [26]

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 viene utilizzato per il monitoraggio dello stato di una rete e per la risoluzione dei problemi che avvengono in essa.

In un **Covert Channel ICMP** (Internet Control Message Protocol) verranno utilizzati i messaggi ICMP per nascondere i dati. L'implementazione del canale è possibile siccome è un protocollo che, dati i suoi utilizzi [Tabella 1], non può essere del tutto disabilitato. Molti firewall e dispositivi di sicurezza consentono il traffico ICMP. Tuttavia, sebbene sia essenziale per la diagnostica di rete, può essere comunque utilizzato in modo improprio per degli attacchi o per la ricognizione 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 1: Utilizzi del protocollo 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. Nell'intestazione, nel caso del protocolllo ICMP, il campo *protocol* avrà valore 1

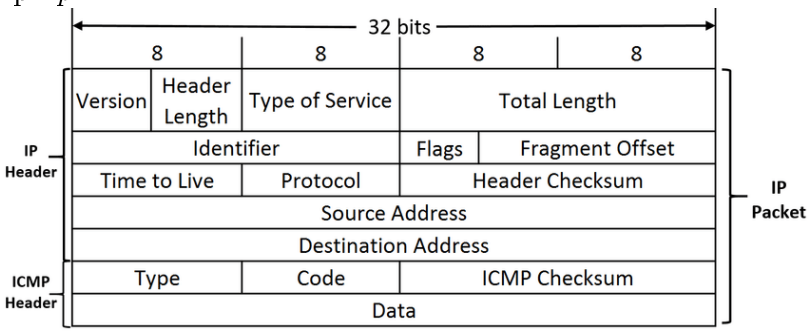


Figura 1: Struttura pacchetto ICMPv4/IPv4 [13]

- **Type**: Identifica la tipologia di messaggio. In base a questo campo verrà determinato il formato dei rimanenti dati.
- **Code**: Fornisce dettagli aggiuntivi sulla tipologia di messaggio.
- **Checksum**: Garantisce l'integrità dei dati.
- **Data**: Campo opzionale, può contenere ulteriori dati.

1.1.1 Tipologie di Messaggi ICMP [5] [4]

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.

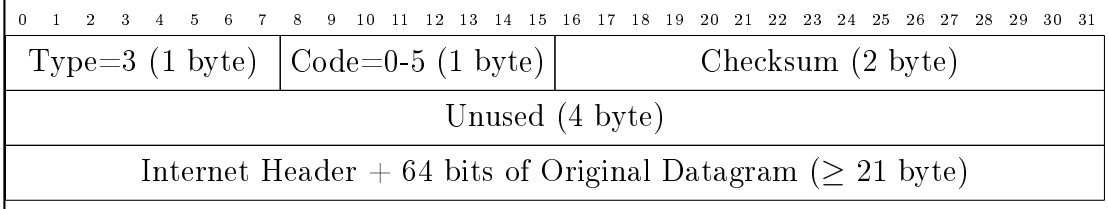
Destination Unreachable

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. Il codice impostato nel messaggio indicherà il motivo [Tabella 2]

Codice	**Descrizione**
Rete irraggiungibile	La rete di destinazione non è raggiungibile.
Host irraggiungibile	Il router può raggiungere la rete ma non l'host specifico. siccome non risponde o non è raggiungibile.
Protocollo non attivo	Il protocollo specificato nel pacchetto IP non è attivo.
Porta non attiva	La porta di destinazione non è attiva o nessun servizio è in ascolto.
Necessaria frammentazione	È necessaria la frammentazione del pacchetto ma il flag "Don't Fragment" (DF) è impostato.

Tabella 2: Destination Unreachable possibili codici

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.

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)		

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.

1.1.2 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. Il codice impostato nel messaggio indicherà il motivo [Tabella 3]

Evento	**Esempio**
TTL scaduto	Il TTL specificato inizialmente è troppo basso o è presente un loop nel routing.
Tempo per il riassemblaggio scaduto	Il tempo per riassemblare i frammenti scaduto.

Tabella 3: Time Exceeded possibili codici

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

In ICMPv4 il campo *Internet Header*: viene utilizzato dall'host per accoppiare il messaggio di errore al processo appropriato.

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 (1 byte)	Code=0-1 (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 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.

1.1.3 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)		

In ICMPv4 il campo *Internet Header*: viene utilizzato dall'host per accoppiare il messaggio di errore al processo appropriato.

Il puntatore identifica l'ottetto nell'intestazione del pacchetto originale in cui è stato rilevato l'errore.

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 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.

1.1.4 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 del tipo Source Quench dal gateway. Nel protocollo **ICMPv4** il pacchetto è strutturato in questo modo:

<table>
<tr><td colspan="32">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</td></tr>
<tr><td colspan="8">Type=4 (1 byte)</td><td colspan="8">Code=0 (1 byte)</td><td colspan="16">Checksum (2 byte)</td></tr>
<tr><td colspan="32">Unused (4 byte)</td></tr>
<tr><td colspan="32">Internet Header + 64 bits of Original Datagram (≥ 21 byte)</td></tr>
</table>

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.

1.1.5 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. Il codice impostato nel messaggio indicherà il motivo [Tabella 4]

Evento	**Esempio**
Route migliore	Il gateway riceve il pacchetto e dalla tabella di routing ottiene che il secondo gateway si trova sulla stessa rete.

Tabella 4: Redirect possibili codici

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.

1.1.6 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.

Il mittente non ha alcuna limitazione sulla quantità di dati inseribili nel campo del payload. Tuttavia una limitazione potrà essere data dalla massima capacità di trasporto dei collegamenti. 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 [7].

1.1.7 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.

1.1.8 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.

1.1.9 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.

1.2 Covert Channel

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 per la creazione di un canale nascosto. E per poter essere efficace, un Covert Channel deve possedere determinate caratteri-sitche [Tabella 5]. 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	Più dati il canale trasmette per un determinato intervallo di tempo, maggiore sarà il rischio che venga scoperto.
Uso delle risorse	Un uso delle risorse improprio o sproporzionato da parte del canale potrebbe andare in conflitto con le risorse legittime presenti nel sistema.

Rumore	L'uso dei servizi e/o delle risorse nel sistema potrebbe alterare il loro funzionamento. Ciò potrebbe attirare l'attenzione da parte degli amministratori.
Indistinguibilità	La trasmissione dei dati mantiene conforme e inalterato il funzionamento delle risorse utilizzate. L'obbietivo è quello di rendersi indistinguibili dalla risorsa autorizzata.

Tabella 5: Caratterisitche di un Covert Channel

1.2.1 Tipologie di Covert Channel [14] [6]

Timing Covert Channel

I covert channel di temporizzazione sono metodi di comunicazione che permettono ad un osservatore (un umano o un processo) di acquisire informazioni attraverso il cambiamento del tempo di risposta di una risorsa. Qualsiasi metodo che utilizza un orologio (o una misurazione del tempo) per segnalare il valore può implementarlo.

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.

Behavioural 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.

Hybrid Covert Channel

Un Hybrid Covert Channel combina più tecniche per aumentare la capacità di trasmissione dei dati nascosti e rendere più difficile la loro rilevazione.

2 Strumenti Utilizzati

Virtual Box [19]

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 [22]

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 rete basterà una variabile di tipo *AsyncSniffer*.

RITA [1]

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,

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.

ICMP Door [15]

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 [16]

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; Siccome non riuscirebbero a trovare alcun dato, a meno che non sappiano della tecnica utilizzata.

ICMP Tunnel [2]

Strumento che permette il tunneling del traffico IP. Tramite delle richieste e risposte ICMP Echo, incapsula il traffico e lo invia al server proxy. Quest'ultimo lo decapsulano e lo inoltra. 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).

3 Implementazione

Dopo aver analizzato gli strumenti che già hanno provato a sviluppare un covert channel tramite ICMP e dopo aver analizzato metodologie sfruttabili per poter nascondere i dati; procediamo nel svilupparne uno.

Per la definizione della struttura, si è preso spunto da **icmp tunnel** [2]. Lo schema generale di funzionamento è illustrato nella figura seguente [Figura 2].

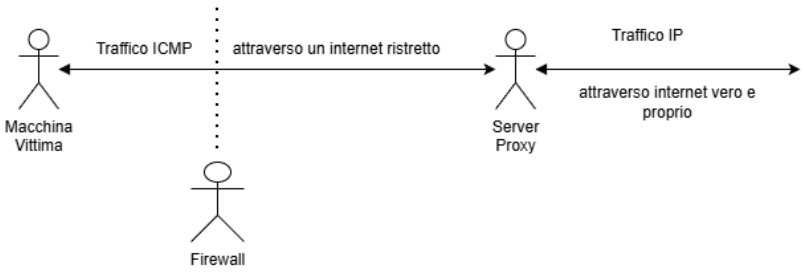


Figura 2: Architettura generale in **icmp tunnel** [2]

icmp tunnel reindirizza il traffico IP verso un interfaccia virtuale per poi reinoltrarlo al proxy tramite il protocollo ICMP [Figura 3]. Il proxy invece rimane in ascolto di questi pacchetti ICMP e li invia verso la destinazione finale. I pacchetti poi in entrata nel proxy, che devono essere inoltrati alla macchina vittima, sono poi incapsulati tramite il protocollo IP e poi inviati [Figura 4].

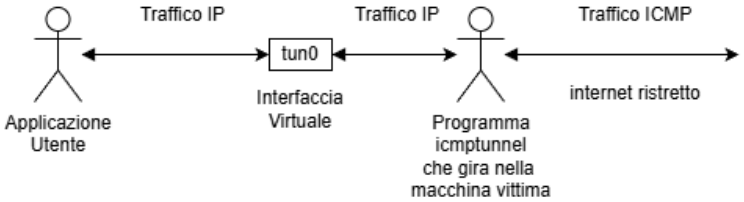


Figura 3: Architettura vitttima-proxy in **icmp tunnel** [2]

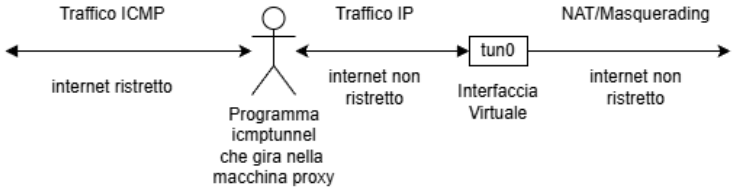


Figura 4: Architettura proxy-internet in **icmp tunnel** [2]

Alcuni aspetti da considerare per l'implementazione della comunicazione sono:

1. La presenza di più di un proxy intermediario fra l'attaccante e la vittima. Sebbene sia possibile usare un singolo proxy, l'utilizzo di più macchine proxy permette di nascondere meglio l'attacco. Con un singolo proxy, un sistema di difesa potrebbe notare che tutto il traffico ICMP viene inviato verso un'unica sorgente. Utilizzando più proxy, il traffico verrà distribuito fra più sorgenti diverse. Tuttavia molteplici proxy comportano un aumento nella complessità della comunicazione.

2. La quantità di dati che si vuole inviare. Se mai si volesse esfiltrare un file contenente una grande quantità di dati; questo potrebbe generare rumore e destare sospetti. Quindi si deve tenere conto di un **limite massimo** di dati da inviare in un intervallo di tempo ed in caso implementare un **periodo di riposo** prima di inviarne altri.

3. La trasmissione dei dati in chiaro permettera a chiunque di leggerne il contenuto. Un sistema di difesa che fà un ispezione approfondita dei pacchetti, potrebbe rilevare la comunicazione. Tuttavia anche mandarli cifrati potrebbe destare sospetti. Si devono qundi poter mandare i dati cifrati ma che sembrino in chiaro. Al momento non si è trovato un metodo efficace a parte l'inserimento di dati in forma numerica; anche se potrebbe essere trovato un pattern anche in questo caso.

4. Un ulteriore fattore sono le tipologie di messaggi che richiedano una risposta. Siccome ad ogni richiesta viene mandata una risposta (avente gli stessi dati ricevuti), si avranno due messaggi identici. Ciò potrebbe non risultare un problema se la dimensione del campo dei dati non sia eccessiva. Una possibile soluzione a questo problema è l'uso solamente dei messaggi di risposta. Tuttavia sarà anomalo che il numero delle risposte non combaci con quello delle richieste. Un ulteriore possibilità è quella, se possibile, di disabilitare l'invio da parte del sistema delle risposte e mandare una risposta che non ripeterà il contenuto della richiesta. In questo caso ad una richiesta combacierà una richiesta sebbene non conforme agli standard.

3.1 Struttura della comunicazione fra le entità

Le entita coinvolte sono:

- **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.

La comunicazione fra le entità è definita dall'immagine seguente [Figura 5].

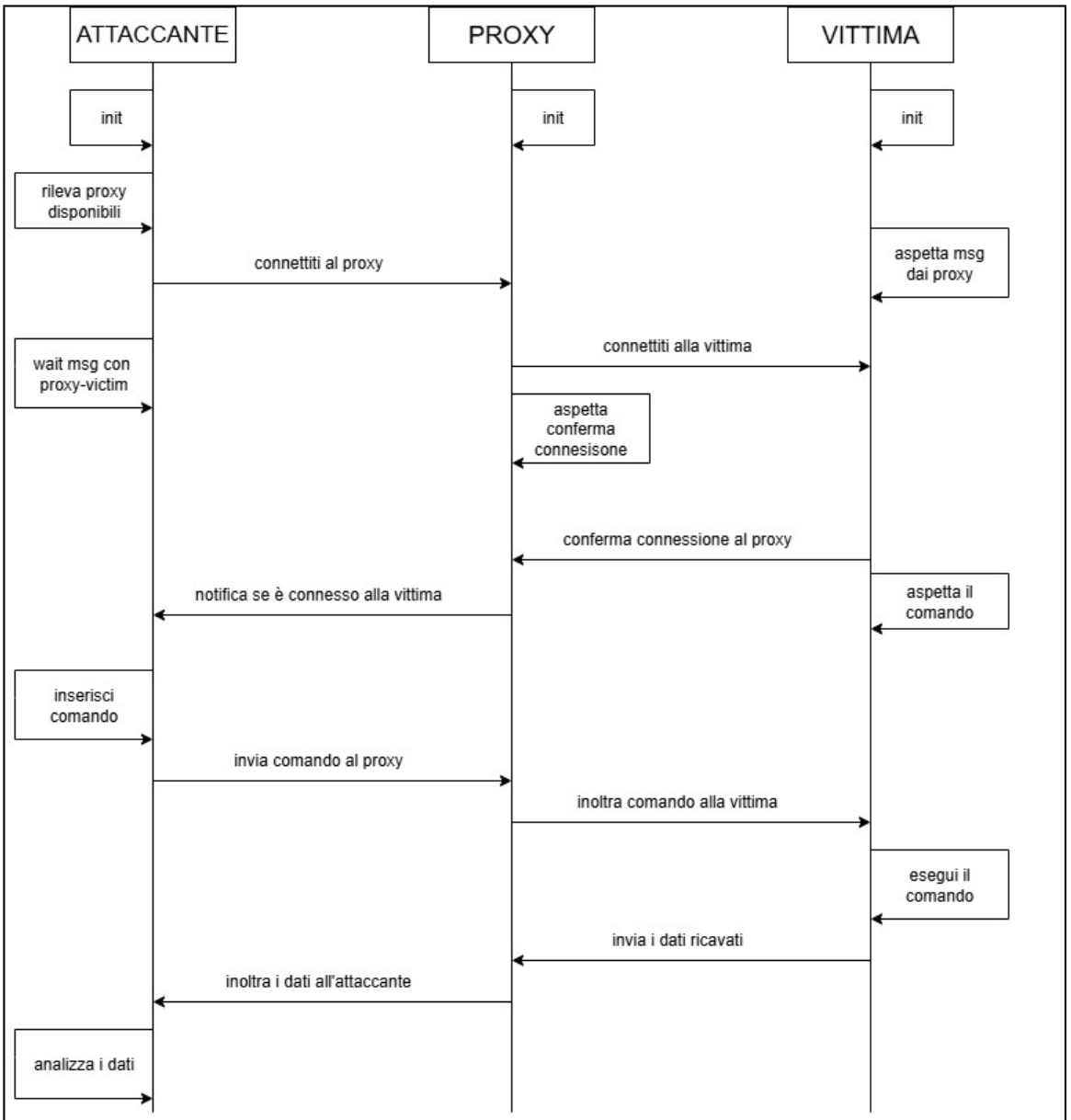


Figura 5: Flusso di comunicazione fra le entita

Il canale di comunicazione che il proxy e l'attaccante potranno avere dipende dall'affidabilità del proxy. Si potrà utilizare una comunicazione tramite ICMP (ovvero la stessa che il proxy avrà con la vittima) oppure si potrà usare il protocollo TCP per avere una comunicazione maggiormente stabile e affidabile.

Inoltre l'attaccante potrà usare uno o più proxy per comunicare con la vittima [Figura 6]. Il caso standard sarà quello in cui l'attaccante usa un singolo proxy. Tuttavià per

l'attaccante sarà possibile comunicare direttamente con la vittima o tramite ulteriori proxy.

Nel primo caso alcune funzioni presenti nell'entità proxy verranno eseguite dall'attaccante (e.g connettersi alla vittima). Nell'ultimo invece, l'attaccante dovrà prevedere un metodo per riunire in modo ordinato i messaggi ricevuti dai proxy.

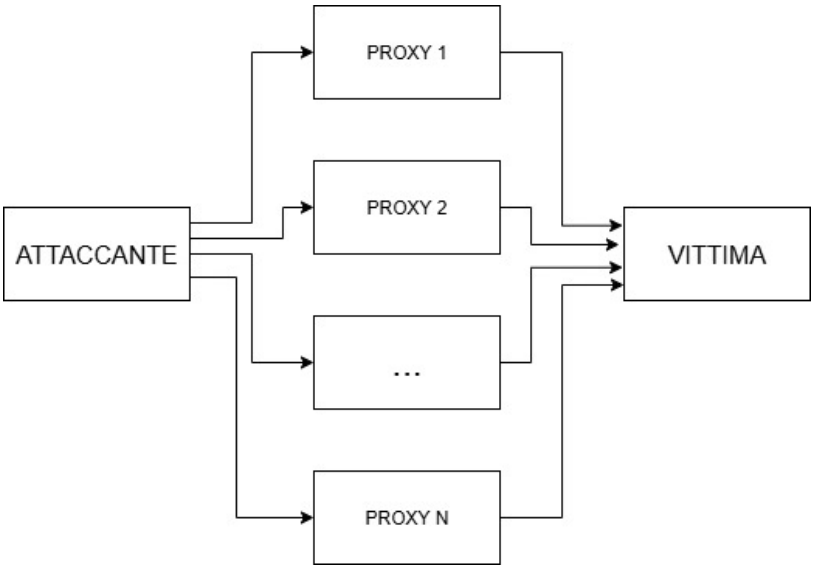


Figura 6: Struttura delle entità presenti e come dialogano

3.1.1 Struttura dell'attaccante

Tramite un parser rileve se nella linea di comando sono state inserite le opzioni necessarie all'inizializzazione [Tabella 7].

Opzione	**Utilizzo**
Path file	Percorso 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 che si vogliono usare.

Tabella 7: Opzioni richieste nel comando dell'attaccante

Per ogni proxy specificato nel file di configurazione, si verifica quali sono disponibili; ovvero quali proxy sono connessi sia all'attaccante che alla vititma. Per farlo si crea

un canale di comunicazione con ciascun proxy, o tramite ICMP o TCP/IP, e si invia un messaggio di connessione. In questo messaggio si specifica sia l'indirizzo IP della vittima che la metodologia di attacco scelta. Successivamente si rimarrà in attesa che il proxy confermi la connessione con la vittima.

Nel caso non si usassero dei sockets TCP/IP ma il protocollo ICMP, bisognerà definire un thread che si occupi di monitorare il traffico di rete per poter catturare i messaggi ICMP contenenti i dati inviati dai proxy. Quindi prima di inviare alcun comando (o dato), bisogna assicurarsi che il thread sia già partito altrimenti si potrebbero perdere delle informazioni.

Una volta ricevuti tutti i dati inoltrati dai proxy, l'attaccante dovrà riordinarli per ricavare il messaggio. Inoltre se si volesse mandare un altro comando, si dovranno reimpostare le variabili necessarie per farlo. Altrimenti si può decidere di interrompere la comunicazione e in quel caso i proxy ne verranno aggiornati.

3.1.2 Struttura del Proxy

Come per l'attaccante, il proxy richiede degli argomenti [Tabella 9].

Opzione	**Utilizzo**
Indirizzo IP	Rappresenta l'indirizzo IP dell'attaccante. Quando il proxy crea il socket e rimane in ascolto delle connessioni, accetterà solo quelle che combaciano con questo indirizzo; qualunque altra connessione verrà rifiutata. Questo viene fatto anche nel caso di un canale tramite ICMP.

Tabella 9: Opzioni richieste nel comando del proxy

Per poter comunicare con l'attaccante, il proxy definisce un socket in cui rimane in ascolto. Nel caso si utilizzi il protoccollo ICMP, monitorerà il flusso di rete per catturare i messaggi inviati dall'attaccante. Stabilita una comunicazione con l'attaccante, il proxy aspetterà un messaggio indicante l'indirizzo IP della vittima oltre alla metodologia di esfiltrazione scelta.

I messaggi che un proxy può ricevere dall'attacante sono:

1. il messaggio indicante il comando, che il proxy dovrà inoltrare alla vittima, o se invece deve aspettare solo i dati, perchè non ha ricevuto il comando.
2. quello che indica la volonta, da parte dell'attaccante, di terminare la comunicazione. In questo caso il proxy aggiornerà la vittima della cosa.

Per la connessione con la vittima, il proxy comunicherà tramite pacchetti ICMP. Prima di inviare alcun dato, si imposta un thread con lo scopo di analizzare il traffico e catturare i dati che la vittima ritornerà. Se questo non viene fatto dopo i pacchetti potrebbe andare persi. Una volta ricevuti tutti i dati dalla vittima, il proxy procederà ad inoltrarli all'attaccante.

3.1.3 Struttura Vittima

Come per l'attaccante e il proxy, la vittima richiede degli argomenti [Tabella 11].

Opzione	**Utilizzo**
Numero di Proxy	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. Una volta raggiunto questo numero, la vittima procederà con l'esecuzione del comando. Altrimenti, se il numero non viene raggiunto, allo scadere del timer si chiederà se si vuole procedere comunque.

Tabella 11: Opzioni richieste nel comando della vittima

La vittima rimane in attesa di connessioni da parte dei proxy. Ciò viene fatto monitornado il flusso di rete e filtrando i messaggi ICMP, destinati alla vittima, che rappresentano una richiesta di connessione. Se il messaggio catturato è valido, si risponde al mittente con un messaggio di conferma e il suo l'indirizzo IP viene inserito nella lista indicante i proxy connessi. Da questo messaggio la vittima ricaverà anche la metodologia di attacco scelta.

Definiti i proxy con cui si comunicherà, si attenderà che inoltrino il comando dell'attaccante. Una volta ricevuto, verrà eseguito e i risultati ricavati (sia quelli legati all'output che quelli legati ad eventuali errori) vengono inviati all'attaccante tramite i proxy disponibili. Nel caso siano presenti molteplici proxy connessi, si definirà una lista indicante i dati che ciscuno di essi dovrà ricevere. L'ultimo messaggio che verrà inviato a ciascuno dei proxy sarà quello che indica che tutti i dati sono stati mandati.

3.2 Implementazione del Timing Covert Channel

Per poter temporizzare i dati da inviare, c'è bisogno sia di un metodo per la codifica e decodifica dei tempi. Dovrà quindi essere presente una funzione che è responsabile di mappare un dato binario a un intervallo di tempo.

Per la **codifica**, si calcolerà dal dato il delay associato; che verrà usato per indicare il tempo da aspetare prima di inviare un nuovo pacchetto. Per esempio al dato binario 1001 potrebbe essere associato il tempo 3; quindi il programma, prima di mandare un nuovo pacchetto, aspetterà tre secondi.

Invece per la **decodifica**, il destinatario rimane in attesa dei pacchetti che gli vengono inviati. Ogni volta calcolerà l'intervallo di tempo fra un pacchetto e il successivo, e da quello ricaverà il dato che gli è associato. Per esempio se l'intervallo di tempo risultasse di 6 secondi, il destinatario controllerà a chi è associato tale tempo e ricaverà il dato binario 1101.

Per l'invio dei dati, si è preso spunto dall'approccio implementato in ICMP Exfill [16] nel quale il delay viene determinato dal valore del byte che si vuole esfiltrare. Ciò è possibile siccome un carattere di un testo viene memorizzato tramite una sequenza di bit la quale potrà essere interpretata anche come un numero intero.

3.2.1 Funzione di codifica

In una **prima versione** della funzione si associa ad ogni punto una distanza di sicurezza, coì da evitare che due codici combacino verso lo stesso tempo di delay [Figura 7]. Infatti durante la trasmissione dela pacchetto possono avvenire dei ritardi non costanti per tutti i datagram inviati [25][8]. La variazione del valore di questo parametro

dipenderà dall'ubicazione geografica dell'entità mittente e di quella destinataria, oltre alla reattività delle due macchine.

La mediana di questo valore [17] è di solito 122 ms (o 0.122 s) con un massimo di 266 ms (o 0.266 s) [1].

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

Il significato dei parametri è descritto nella seguente tabella [Tabella 12].

Parametro	**Descrizione**
Tempo di base	Il minimo di secondi che si dovrà aspettare per ciascun possibile dato. È necessario per evitare di causare un overflow di pacchetti verso la macchina che riceve i dati.
Indice	È l'indice associato alla codifica del dato. Il suo valore và da 0 sino a 255 siccome si prende un byte alla volta.
Distanza fra i tempi	Distanza minima di secondi che tutte le codifiche dovranno avere fra di loro.

Tabella 12: Parametri della prima funzione (Timing Channel)

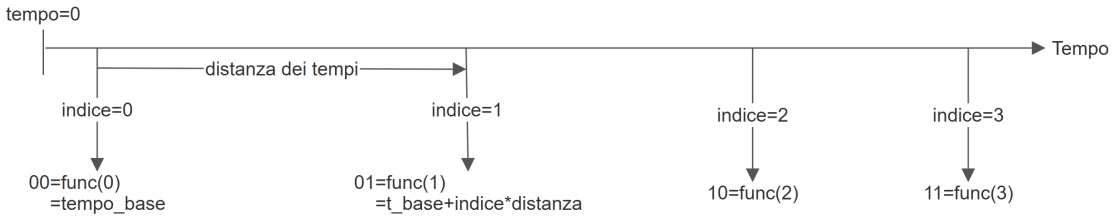


Figura 7: Assegnazione degli intervalli con la distanza

Lo svantaggio di questo approccio sono i tempi che si aspettano. Nel miglior caso si aspetterà solo il tempo di base nel caso peggiore invece 255 volte la distanza scelta.

[1] i dati presi in analisi sono quelli relativi all'Italia

Cercando di avere una stima più precisa, si sa che i caratteri stampabili vanno da 32 a 126 [27]. Da ciò si ricava che in media si aspetterà 79 [2] volte la distanza scelta. Supp che il tempo di base sia di 3 secondi e che la distanza fra i tempi sia di 0.6 secondi (600 ms); in media si aspetteranno circa 51 secondi.

È quindi richiesta un **seconda implementazione** per evitare tempi di attesa eccessivamente lunghi per l'invio dei dati.

Siccome il collo di bottiglia risultava il valore che lo stesso byte aveva, si è pensato di ridurre i dati trasmissibili riducendo i valori ASCII usati nella trasmisisone; in particoalre si sarebbero inviati solo i caratteri stampabili. La scelta è stata ritenuta valida siccome nella codifica ASCII i principali caratteri stampabili vanno da 32 sino a 126 [Tabella 13]. Tuttavia ciò, oltre ad aggiungere overhead inutile, avrebbe tagliato fuori alcuni caratteri speciali come il tab o il carattere di nuova linea. Inoltre il metodo sarebbe stato vincolato alla codifica ASCII. Quindi se i dati fossero stati codificati tramite un altro metodo, si sarebbero potuti tagliare dati importanti.

Range	**Descrizione**
Control Character [0-31]	Codici di controllo non stampabili e che vengono utilizzati per controllare le periferiche.
Printable character [32-127]	Rappresentano lettere, cifre, segni di punteggiatura e vari simboli.
Extended ASCII Codes [128-255]	Non fanno parte dell'ASCII standard ma alla sua versione estesa. I caratteri presenti dipendono dalla codifica utilizzata.

Tabella 13: Struttura della tabella ASCII

Quindi si è definita una funzione che cercerà di normalizzare l'intervallo di tempo, associato al dato, fra un valore minimo e uno massimo [11] [24] [23]. Tramite questa funzione il range dei possibili delay è stato ristretto a due valori precisi e definiti che permetterà di inviare un maggiore numero di informazioni in un tempo minore

[2] $\text{tempo base} + \frac{32+126}{2} * \text{distanza}$

[Figura 8].

Il lato negativo è una maggiore possibilità di errore. In questo caso bisongerà impostare correttamente il valore minimo e massimo affinche i dati vengano decodificati in maniera errata. Minore è la loro differenza maggiore sarà la probabilità che i ritardi influenzino il risultato.

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

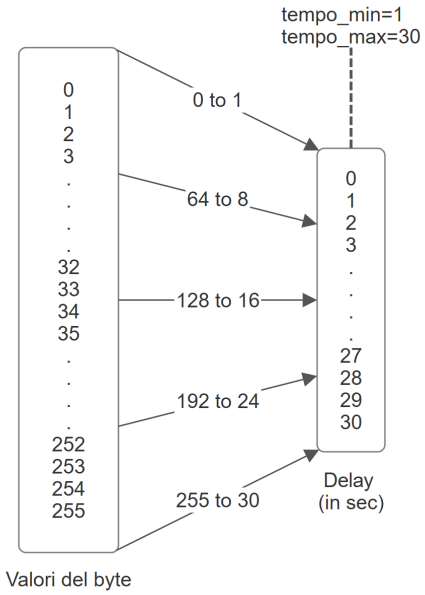


Figura 8: Normalizzazione degli intervalli (Timing Channel) [18]

3.2.2 Randomizzazione dei tempi di invio dei pacchetti

Siccome si mandano i pacchetti con tempi costanti si sviluppa il Covert Channel inserendo del rumore. Il tempo di delay associato a un byte non sarà più costante ma spazierà fra un range di valori, nel quale potrà variare [Figura 9]. Ciò è necessario siccome gli IDS riescono ad individuare gli schemi temporali, quindi poter avere uno schema che sembri randomico, permetterà di non essere rilevati.

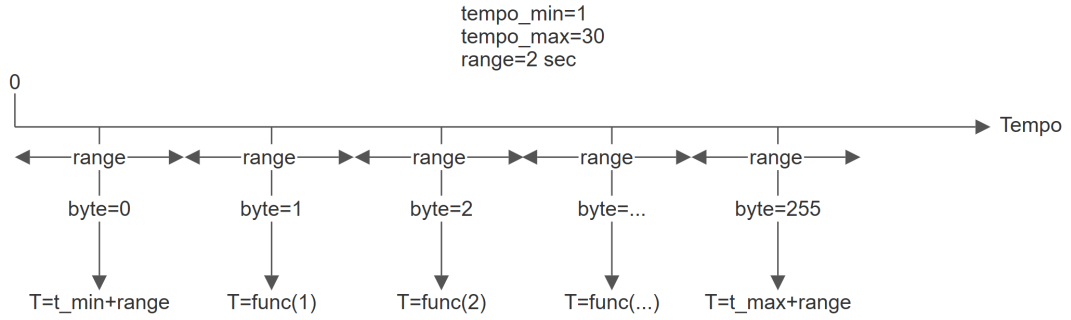


Figura 9: Randomizzazione dei tempi con il rumore (Timing Channel)

Per potersi scambiare i dati, il mittente e il destinatario devono potersi sincronizzare sulla quantità di rumore che si è aggiunto. Le strade possibili sono due:

1. Ogni entità crea un numero randomico e si dovranno sincronizzare sul seed usato oltre al numero di estrazioni effettuate. Se i valori estratti non combaciassero; le entità non riusciranno a scambiarsi i dati correttamente.

2. Il mittente indica il rumore generato nel pacchetto stesso (tramite uno dei campi o il payload). Il ricevente otterrà il valore generato dalla risorsa condivisa. Tuttavia in questo caso sarebbe maggiormente comodo inserire il dato direttamente nel campo.

3. Il tempo non trasporterà alcun dato negli intervalli di tempo. Ma in questo caso non si tratterà più di un Timing Covert Channel.

3.2.3 Requisiti per la comunicazione

Siccome il destinatario rimane in ascolto e, all'arrivo di un pacchetto, ricava il dato associato all'intervallo di tempo; avrà bisogno di un messaggio che indichi quando la trasmissione inizia. Il mittente dovrà quindi mandare un pacchetto per indicare la cosa prima di iniziare la comunicazione.

Questo datagram potrà essere un pacchetto contenente un valore specifico oppure di un tipo specifico.

3.3 Implementazione del Storage Covert Channel

In uno Storage Covert Channel, la risorsa condivisa sarà il pacchetto ICMP inviato e i dati saranno scritti nei suoi campi. Il destinatario, una volta ricevuto, potrà poi leggere i valori codificati al loro interno.

Nelle tabelle [Tabella 15][Tabella 17] sono indicati quali tipologie di messaggi verranno sfruttati oltre a quali campi sono stati utilizzati e quanti byte pesa un singolo pacchetto. Nel caso di tipologie come i messaggi Echo Request/Reply, si avranno delle varianti sia nella quantità di dati esfiltrabili sia sulla quantità di byte che vengono trasmessi per singolo datagram. Ciò è dovuto alla presenza del campo *data* che permette di inserire una quantità "illimitata" di dati.

Tipologia	**Byte per pacchetto**	**Campi Utilizzati**
Destination Unreachable	20+8+21 →min 49 byte	unused(4 bytes) header+64 bits (2bytes)
Time Exceeded	20+8+21 →min 49 byte	unused(4 byte) header+64 bits(2 byte)
Parameter Problem	20+8+21 →min 49 byte	pointer(1byte) unused(3byte) header+64 bits(2byte) 4-8
Source Quench	20+8+21 →min 49 byte	unused(4byte) header+64 bits(2byte) 3-8
Redirect Message	20+8+21=min 49 byte	header+64 bits(2byte) 3-4
Echo Request/Reply	20+8+32 →min 60 byte	identifier(2byte) data($\geq$ 32) 2
Timestamp Request/Reply	20+8+12+32 →min 72 byte	identifier(2byte) timestamp(4byte*3) data($\geq$ 32) 5
Information Request/Reply	20+8=28 byte	identifier(2byte) 2

Tabella 15: Tipologie di messaggi ICMPv4

Tipologia	Byte per pacchetto	Campi Sfruttatti
Destination Unreachable	40+8+41 →min 89 byte	unused(4 bytes) invoking packet(2bytes) 4-8
Time Exceeded	40+8+41 →min 89 byte	unused(4 bytes) invoking packet(2bytes) 4-8
Parameter Problem	40+8+41 →min 89 byte	pointer(4byte) invoking packet(2byte) 8
Echo Request/Reply	40+8+32 →min 80 byte	identifier(2byte) data($\geq$ 32) 2
Packet Too Big	40+8+41 →min 89 byte	mtu(4byte) invoking packet(2byte) 8

Tabella 17: Tipologie di messaggi ICMPv6

3.3.1 Come i dati vengono inseriti nei campi utilizzati

Campo unused

Dalle specifiche RFC 792[5] (e RFC 4434 [4]) deve essere 0 quindi Scapy, quando manderà il pacchetto, azzererà qualsiasi valore inserito al suo interno [Figura 10].

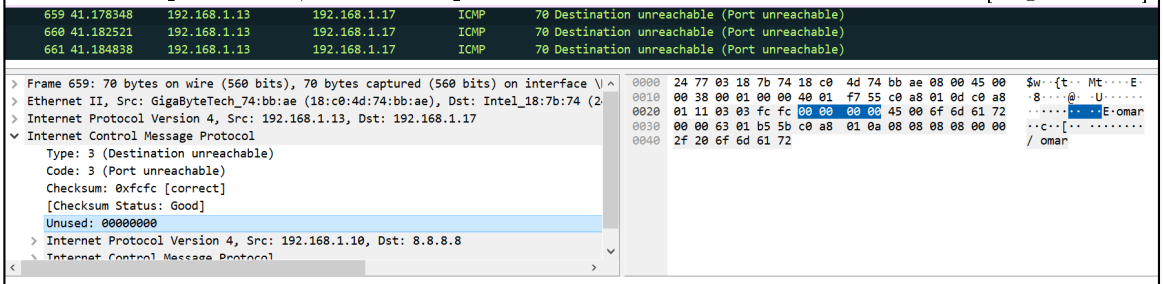


Figura 10: Messaggio Destination Unreachable con il campo *unused* azzerato

Tramite degli accorgimenti è possibile inserire dei dati all'interno del campo [Figura 11]. Tramite la libreria *struct* [20], si può riscrivere, il pacchetto evitando di utilizzare il livello ICMP di Scapy; così che al suo invio i dati siano presenti.

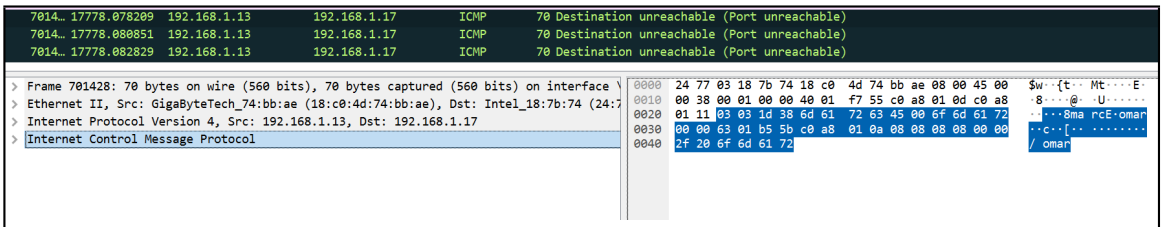


Figura 11: Messaggio Destination Unreachable che usa il campo *unused*

Siccome il suo utilizzo rende il messaggio non conforme allo standard RFC [5][4]; un IDS che implementi la Deep Packet Inspection rileverà l'anomali. Si potrà quindi scegliere se inserire dati nel campo o no [Figuara 12].

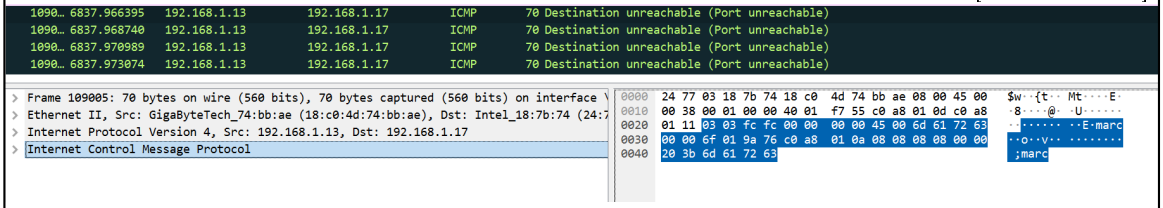


Figura 12: Messaggio Destination Unreachable che non usa il campo *unused*

Campo Header+64 bits

Nel campo si dovranno inserire l'header IP più 8 byte. Nel nostro caso si è usato IP/ICMP e siccome il campo rappresenta un datagram già spedito, si sfruttano un maggior numero di campi (in particolare quelli del protocollo IP [Tabella 19]).

Header IPv4

Campo	**Byte**	**Utilizzo**
Time to live	1 byte	Tempo di vita del pacchetto
Total length	2 byte	Dimensione dell'intero pacchetto
Identification	2 byte	Identifica i frammenti di un pacchetto IP

Header ICMPv4

Campo	**Byte**	**Utilizzo**

Identifier	2 byte	Identificativo del messaggio di richiesta invocato
Sequence	2 byte	Numero di sequenza del messaggio di richiesta invocato

Tabella 19: Struttura del campo Header+64 bits

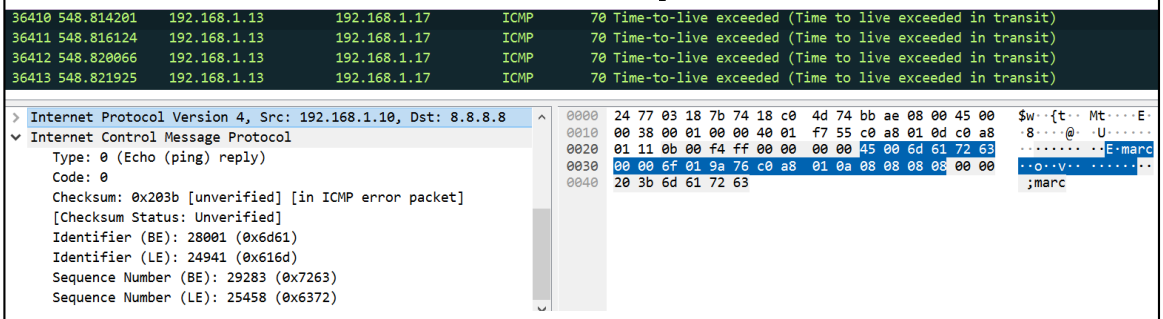


Figura 13: Messaggio Time Exceeded e il campo *Header+64 bit*

Campo 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.

Header IPv6

Campo	**Byte**	**Utilizzo**
Payload Length	2 byte	Tempo di vita del pacchetto
Hop Limit	1 byte	Decrementato di 1 per ogni nodo che inoltra il pacchetto

Header ICMPv6

Campo	**Byte**	**Utilizzo**

Identifier	2 byte	Identificativo del messaggio di richiesta invocato
Sequence	2 byte	Numero di sequenza del messaggio di richiesta invocato

Tabella 21: Struttura del campo Invoking Packet

Campo Pointer

Indica l'ottetto del messaggio in cui è presente l'errore. Tuttavia può contenere dei dati non correlati ad esso; in questo caso, verranno inseriti tanti dati quanto la sua dimensione in byte.

Campo Identifier e Sequenza

Il campo *Identifier* definisce l'identificativo delle richieste e può assumere un qualsiasi valore. Il campo *Sequenza*, dalle specifiche RFC 792, viene incrementato ad ogni richiesta inviata con lo stesso Identifier. Se il valore cambiasse troppo spesso, a differenza del campo *Identifier*, la cosa potrebbe risultare sospetta.

Quindi sebbene può essere utilizzato per inserire le informazioni, si userà quasi esclusivamente il campo Identifier.

Campo Data

In questo campo il mittente può inserire quanti dati preferisce. Ma per sicurezza la dimensione sarà fra i 32 ed i 64 byte.

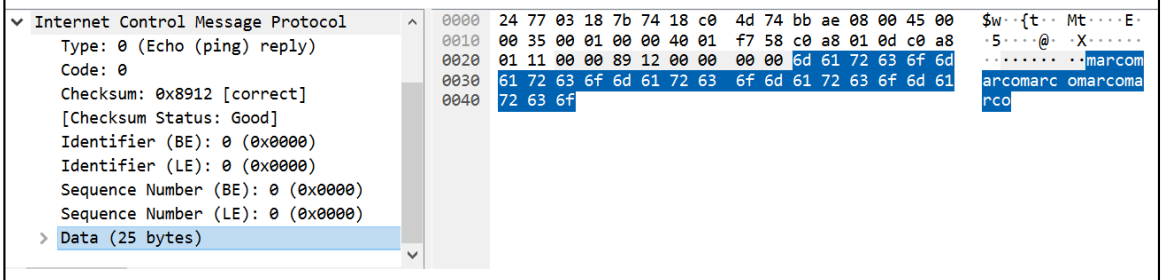


Figura 14: Messaggio Echo Reply e il campo *Data*

Campo Timestamp

Contiene i millisecondi dalla mezzanotte UT e ciascun campo ha un ordine temporale specifico; che dovrà rimanere congruente quando si inseriscono i dati. In ciascun campo timestamp verrà inserito un byte nella parte dei millisecondi [Figura 1].

```
Dati nasocsti:  b'r'     114
Dati nasocsti:  b'c'     99
Dati nasocsti:  b'o'     111

Timestamp origin prima:  2025-11-07 23:13:36.146254+00:00
Timestamp origin dopo:  2025-11-07 23:13:36.114000+00:00
Tempo campo origin:  83616114
```

Listing 1: Output per la creazione di un messaggio Timestamp

Non è possibile inserire informazioni in ulteriori parti del timestamp siccome hanno dei valori ben definiti, pocihè rappresentando le ore, i minuti ed i secondi [Figura 15]. Si potrebbero inserire i dati sottoforma di dati temporali; ma questo ci espone al rischio di avere tempi errati (e.g ricevo il messaggio prima che arrivi).

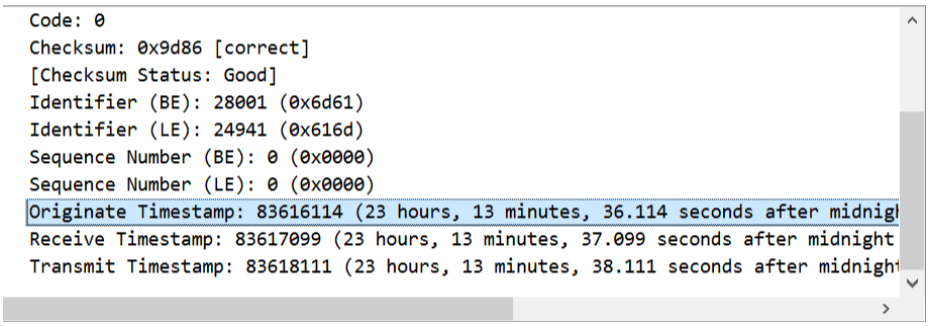


Figura 15: Messaggio Echo Reply e il campo *Data*

Campo MTU

Nel campo viene indicata la capacità massima del collegamento; siccome il suo valore è variabile, e quindi non c'è un valore prestabilito, potrà essere usato per inserire dei dati.

3.4 Implementazione del Behavioural Covert Channel

In questo caso l'evento che si osserverà, e che indicherà il dato trasmesso, è la tipologia ed il codice del messaggio ICMP arrivato.

Seguendo questo approccio, la quantità totale di eventi possibili risulta 17 [Figura 16]. Quindi ogni messaggio permetterà di codificare 4 bit alla volta (siccome $\log_2 16 = 4$). Siccome un evento (tipologia,codice) rimane inutilizzato; verrà sfruttato per indicare l'inizio e la terminazione dell'esfiltrazione dei dati.

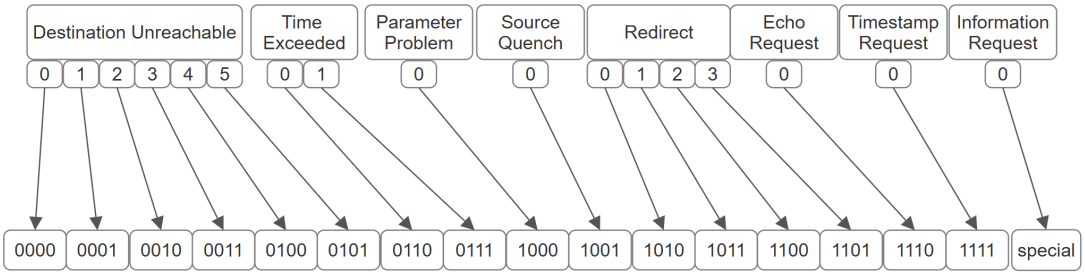


Figura 16: Schema Hybrid Channel [21]

Siccome un singolo evento può trasmettere solo 4 bit, ogni byte che si vuole inviare dovrà essere diviso in due parti. La divisione viene fatta mascherando il byte [Figura 17]. Questi primi quattro bit ricavati, così come gli ultimi quattro bit ricavati, determineranno la tipologia (oltre al codice) del messaggio che verrà inviato.

Figura 17: Divisione del byte

Il destinatario, per ricavare i dati, monitora il traffico di rete e controllerà le tipologie di messaggi ICMP ricevuti; per ognuno di essi, prende la coppia (Tipologia, Codice) e da questa ricaverà i quatttro bit associati.

3.4.1 Tipologie di messaggi deprecati [12] [10] [9]

Un problema di questa possibile implementazione, sono le tipologie deprecate dei messaggi. Questi messaggi potranno comunque essere inviati ma le linee guide indicano

ai router di ignorarli e all'host utente di non mandarli.

L'uso di tipologie deprecate renderà il canale meno legittimo e più facile da individuare. Traffico di questo tipo inoltre sarà visto con sospetto e non è assicurata la ricezione dei messaggi.

Source Quench

Il messaggio ICMP Source Quench (tipo 4, codice 0) era concepito come meccanismo per il controllo della congestione. Tuttavia, data la sua inefficacia per la congestione, la generazione di questi messaggi da parte dei router è stata formalmente deprecata dal 1995. Per questo la maggior parte delle implementazioni TCP ignorano silenziosamente i messaggi ICMP Source Quench. Infatti TCP implementa i propri meccanismi di controllo della congestione (che non dipendono dai messaggi ICMP Source Quench).

Quindi, siccome la reazione a questi messaggi nei protocolli di trasporto non è mai stata formalmente deprecata, la IETF (Internet Engineering Task Force) ha aggiornato le linee guida riguardo il messaggio Source Quench in questo modo:

- Un host **NON DEVE inviare** dei messaggi di questo tipo. Se viene ricevuto un messaggio di questo tipo, **PUÒ ignorarlo** (silenziosamente).
- Un router **DEVE ignorare** tutti i messaggi ICMP Source Quench ricevuti. Inoltre **NON DOVRÀ inviare** messaggi ICMP Source Quench in risposta a una congestione.
- Se un protocollo di trasporto riceve un messaggio Source Quench, **DEVE essere ignorato/scartato** silenziosamente.
- Host, gateway e firewall DEVONO scartare silenziosamente i pacchetti ICMP Source Quench ricevuti e **DOVREBBERO registrare** la cosa come un errore di sicurezza con almeno i seguenti dettagli (indirizzo IP sorgente, indirizzo IP di destinazione, tipo di messaggio ICMP, data/ora in cui il pacchetto è stato visualizzato).

Nell'Internet attuale, non ci sono ragioni per cui un host debba generare o reagire a un messaggio ICMP Source Quench (o perchè un router debba reagire). L'unico motivo

per cui un utente ignori questo requisito, è per poter consumare le risorse di rete. I messagg ICMP Source Quench potrebbero essere infatti sfruttati per eseguire attacchi "blind throughput-reduction". Le linee guida indicano di ignorare silenziosamente questi messaggi; questo eliminerà il vettore di attacco.

Inoltre, siccome la generazione e la reazione ai messaggi ICMP Source Quench è stata deprecata, le applicazioni non dovranno aspettarsi di ricevere questi messaggi.

Information Request/Reply

È stato deprecato siccome meccanismi come il DHCP [3], lo hanno sostituito per la configurazione dell'host

3.5 Implementazione di un Hybrid Covert Channel

Un Covert Channel ibrido è stato implementato combinando le tre tipologie di Covert Channel usate precedentemente:

- *Timing Channel*: un byte indicherà il delay fra un pacchetto e l'altro.
- *Behavioural Channel*: da un byte verranno ricavate le due tipologie di messaggi da mandare.
- *Storage Channel*: in ciascun pacchetto verranno inseriti tanti dati quanto la capacità di trasmissione disponibile.

La comunicazione viene iniziata tramite un messaggio *Information Request* e chiusa sempre da un messaggio dello stesso tipo.

Il destinatario monitora il flusso dei dati nella rete per rilevare l'intervallo di tempo con cui arrivano le coppie di messaggi. Da questo si ricaverà il byte relativo al tempo. Dalla coppia di messaggi ricava i quattro bit associati a ciascun messaggio. Dall'unione dei bit ricaverà il byte relativo alla tipologia di messaggi inviati. Ora non resta che ricavare i dati che sono stati nasconsti nei campi dei messaggi ICMP ricevuti.

4 Test e Risultati

4.1 Test dei Covert Channel con RITA

Per potr verificare se RITA fosse in grado di rilevare i Covert Channel implementati, si sono effettuati diversi test. RITA ha analizzato i log di Zeek generati usando la sua configurazione di default.

Nel **primo test** si è esfiltrato un file di testo contenente una quantità di dati pari a **37 KB**. Inoltre fra l'invio di un pacchetto e quello successivo si ha un **delay** in media di **2 secondi**.

Dai risultati [Tabella 22] si ricava che alla maggior parte dei Covert Channel viene assegnata una gravità bassa e un tasso di beaconing dello 0%. Ciò significa che l'esfiltrazione non ha destato sospetti, sebbene lo scambio di dati sia stato comunque rilevato. I motivi della gravità bassa sono dati dalla qunatità bassa di dati esfiltrati mentre il livello di beaconing è derivato dal fatto che la richiesta dei dati (e il loro invio) è stata fatta poche volte.

Fra questi test, un eccezzione è il *Covert Channel Ibrido* nel quale risulta un tasso di beacon del 85.4% e un alto tasso di gravità.

Un ulteriore eccezzione è il *Covert Channel Echo* che utilizza i campi del messaggio ed il payload. Non risulta alcun dato perchè il tempo è stato troppo breve.

Covert Channel	**Analisi di RITA**		
	Beacon	Tempo connessione	Gravità
Information Reply	0.0%	1h 26m 32s	Low
Timestamp Reply	0.0%	1h 25m 53s	Low
Redirect	0.0%	1h 25m 28s	Low
Source Quench	0.0%	1h 26m 28s	Low
Parameter Problem	0.0%	1h 23m 51s	Low
Time Exceeded	0.0%	1h 24m 46s	Low
Destination Unreachable	0.0%	1h 23m 51s	Low
Echo solo campi	0.0%	1h 25m 23s	Low

Echo solo payload	0.0%	1h 9m 17s	Low
Echo campi+payload	no items		
Ibrido	85.4%	15h 7m 40s	High

Tabella 22: Invio di 37 KB una volta sola

Nel **secondo test** si si inviano due file. Il primo contiene un quantità di dati pari a **37 KB** mentre il secondo contiene **216 KB**. Il test è stato fatto siccome il precedente mandava solo un singolo file; quando normalmente un attaccante cercerebbe di esfiltrarne un numero maggiore. In quest testi ci si è serviti solo dei Covert Channel ICMP Echo; principalmente perchè ripsetto al test precedente davano risutlati migliori rispetto agli altri. Inoltre le Echo Request/Reply sono i messaggi ICMP più comuni e utilizzati.

I risultati [Tabella 23] sono che il Covert Channel che utilizza solo i campi per l'esfiltrazione ha dei dati di esecuzione elevati rispetto agli altri. Di conseguenza RITA gli assegnata una gravità alta sebbene il tasso di beaconing è relativamente contenuto. Invece il Covert Channel che sfrutta solamente il payload ha una gravita media ma dei tempi di comunicazioni simili a quelli precedenti. Infine il Covert Channel che utilizza sia i campi che il payload risulta il migliore: ha una gravità media tuttavia il suo tempo di esecuzione e notevolemnte minore rispetto a quello delli altri due. Tuttavia fra i tre, ha il tasso di beaconinmg maggiore.

Covert Channel	**Analisi di RITA**		
	Beacon	Tempo connessione	Gravità
Echo solo campi	12.20%	9h 30m 34s	High
Echo solo payload	10.7%	7h 42m 49s	Medium
Echo campi+payload	19.6%	4h 30m 44s	Medium

Tabella 23: Invio di 37KB e poi 216KB

Nel **terzo test** si sono mandati **1.5 MB** di dati senza delay, **3 MB** di dati senza delay e entrambi i dati precedenti senza delay (**4.5 MB**). Il Covert Channel utilizzato è quello che utilizza l'ICMP Echo sia con i campi che con il payload.

Il risultato [Tabella 24] è che non vi è alcuna voce nel database di RITA riguardante lo scambio di messaggi. Ciò può risultare positivo, siccome RITA non ha rilevato anomalie apparenti, tuttavia mandare una notevole quantità di dati in un canale viene notata.

Covert Channel	**Analisi di RITA**		
	Beacon	Tempo connessione	Gravità
1.5 MB no delay	No items		
3 MB no delay	No items		
1.5+3 MB no delay	No items		

Tabella 24: Invio di grandi dati

Nel **quarto test** si inviano i dati, la cui quantità risulterà di **1.5 MB**, tramite dei delay. Nel primo caso si dividono i dati in blocchi da al massimo 16KB, e poi si aspettano 3 minuti prima di inviare un altro blocco. Nel secondo caso invece si manda l'intero documento e poi si aspetta un ora pima si mandare ulteriori documenti.

I risultati [Tabella 25] sono che nel primo caso RITA rileva un tasso di beaconing del 97.7% ed una gravità alta. Ciò è dovuto al fatto che i dati sono stati mandati in blocchi regolari e con un delay quasi costante. Nel secondo caso invece RITA non rileva nulla, sebbene si sia mandata una grande quantità di dati, probabilmente perchè il tempo di attesa è stato abbastanza lungo.

Covert Channel	**Analisi di RITA**		
	Beacon	Tempo connessione	Gravità
1.5 MB +3 min delay	97.70%	47s	High

1.5 MB +wait 1 hour	9.10%	3m 0s	None

Tabella 25: Invio di grandi dati con del delay

Riferimenti bibliografici

[1] Active Countermeasures. Rita, 2025. URL https://github.com/activecm/rita.

[2] DhavalKapil. icmptunnel, 2025. URL https://github.com/DhavalKapil/icmptunnel.

[3] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997. URL https://www.rfc-editor.org/info/rfc2131.

[4] RFC Editor. Rfc 4443, March 2006. URL https://www.rfc-editor.org/rfc/rfc4443.

[5] RFC Editor. Rfc 792, September 1981. URL https://www.rfc-editor.org/rfc/rfc792.

[6] Muawia Elsadig. Network covert channels. In *from the Edited Volume: Steganography - The Art of Hiding Information, Joceli Mayer*, Submitted: 09 March 2024 Reviewed: 11 March 2024 Published: 03 April 2024.

[7] Gowthamraj F. Understanding the icmp protocol with wireshark in real time, Jan 21, 2022. URL https://learningnetwork.cisco.com/s/article/Understanding-the-ICMP-Protocol-with-Wireshark-in-Real-Time.

[8] GeeksforGeeks. Delays in computer network, 28 Dec, 2024. URL https://www.geeksforgeeks.org/computer-networks/delays-in-computer-network/.

[9] Fernando Gont. Deprecation of ICMP Source Quench Messages. RFC 6633, May 2012. URL https://www.rfc-editor.org/info/rfc6633.

[10] Fernando Gont and Carlos Pignataro. Formally Deprecating Some ICMPv4 Message Types. RFC 6918, April 2013. URL https://www.rfc-editor.org/info/rfc6918.

[11] Google. Dati numerici: normalizzazione, 2025-07-10 UTC. URL https://developers.google.com/machine-learning/crash-course/

numerical-data/normalization?hl=it#summary_of_normalization_techniques.

[12] IANA. Internet control message protocol (icmp) parameters, 2025-04-29. URL https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml.

[13] Md Nazmul Islam, Vinay Patil, and Sandip Kundu. Determining proximal geolocation of iot edge devices via covert channel. pages 196–202, 03 2017. doi: 10.1109/ISQED.2017.7918316.

[14] Science Direct; Mazurczyk Wojciech; Tan Yu'an; Guri Mordechai; J.M. Keller. Covert channel. URL https://www.sciencedirect.com/topics/computer-science/covert-channel.

[15] krabelize. icmpdoor, 2025. URL https://github.com/krabelize/icmpdoor.

[16] martinoj2009. Icmpexfill, 2025. URL https://github.com/krabelize/icmpdoor.

[17] Microsoft. Azure network round-trip latency statistics, 08/18/2025. URL https://learn.microsoft.com/en-us/azure/networking/azure-network-latency?tabs=Americas%2CWestUS.

[18] WallStreet Mojo. Normalization formula, Unknown. URL https://www.wallstreetmojo.com/normalization-formula/.

[19] Oracle Corporation. Virtualbox official website, 2025. URL https://www.oracle.com/virtualization/virtualbox/.

[20] Python. struct — interpret bytes as packed binary data, Nov 07, 2025. URL https://docs.python.org/3/library/struct.html.

[21] F. Santini.

[22] SecDev. Scapy, 2025. URL https://github.com/secdev/scapy.

[23] StackExchange. Map real numbers into [0:255] using fixed limits interval, Aug 2, 2012. URL https://gamedev.stackexchange.com/questions/33441/how-to-convert-a-number-from-one-min-max-set-to-another-min-max-set.

[24] StackExchange. Map real numbers into [0:255] using fixed limits interval, Aug 28, 2015. URL https://math.stackexchange.com/questions/1412371/map-real-numbers-into-0255-using-fixed-limits-interval.

[25] Wikipedia. Network delay, 18 October 2025. URL https://en.wikipedia.org/wiki/Network_delay.

[26] Wikipedia. Internet control message protocol, 2025. URL https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[27] Wikipedia. List of unicode characters, 27 September 2025. URL https://en.wikipedia.org/wiki/List_of_Unicode_characters.