



UNIVERSITÀ DI PERUGIA
Dipartimento di Matematica e Informatica



TESI MAGISTRALE IN INFORMATICA
CURRICULUM IN CYBERSECURITY

Un nuovo approccio alla steganografia di
rete basato sugli header HTTP

Relatore

Prof. Stefano Bistarelli
Prof. Francesco Santini

Candidato

Michele Ceccarelli

Anno Accademico 2021-2022

Ai miei genitori Sergio e Tiziana,
a mia sorella Elena
e ai sani valori e principi.

Indice

1	Introduzione	6
2	Background	8
2.1	Steganografia	8
2.2	Tipi di steganografia	9
2.3	Network steganography	10
2.4	Terminologia	10
2.5	Problema del prigioniero	12
2.6	Covert channel	14
2.7	Scenari di comunicazione	15
2.8	Applicazioni	17
2.9	Valutazione dei canali	19
2.10	Tassonomia	21
2.10.1	Proposta del 1985	22
2.10.2	Proposta del 2007	23
2.10.3	Proposta del 2015	27
2.10.4	Proposta del 2018	33
2.10.5	Proposta del 2022	38
2.11	Related work	41
2.11.1	Livello data link	41
2.11.2	Livello rete	43
2.11.3	Livello trasporto	55
2.11.4	Livello applicazione	59
2.11.5	Canali recenti	74

3 Strumenti disponibili	77
3.1 Tipi di licenze	77
3.2 Strumenti utili allo sviluppo di tool	78
3.2.1 CCEAP	79
3.2.2 Arpspoof	80
3.2.3 Scapy	81
3.2.4 Hping	82
3.2.5 Arp-sk	83
3.2.6 Arping	84
3.2.7 p0f	85
3.2.8 Nmap	86
3.2.9 Tcpdump e Libpcap	87
3.2.10 TShark	88
3.2.11 Wireshark	88
3.2.12 NeFiAS	89
3.2.13 ExifTool	90
3.2.14 GTFOBins	90
3.2.15 LOLBAS	91
3.2.16 Metasploit	91
3.2.17 Precisazioni	91
3.2.18 NELphase	92
3.3 Tool	93
3.3.1 Livello data link	93
3.3.2 ICMP	94
3.3.3 IGMP	98
3.3.4 IP	98
3.3.5 TCP	100
3.3.6 UDP	102
3.3.7 HTTP	103
3.3.8 DNS	108
3.3.9 VoIP	114
3.3.10 Altri tool	115
3.3.11 Pluggable Transport nella rete Tor	123

3.4	Attacchi che hanno usato steganografia di rete	125
4	Sviluppo di un Proof of Concept	126
4.1	Analisi dei protocolli più utilizzati nella steganografia di rete	126
4.2	Motivazione delle scelte progettuali	127
4.3	L'idea	128
4.3.1	Funzionamento	128
4.3.2	I campi di intestazione del protocollo HTTP	129
4.3.3	Struttura della richiesta inviata	130
4.3.4	Tecniche utilizzate nei vari campi	131
4.3.5	Strategie comunicative	138
4.4	L'implementazione	138
4.4.1	Client	139
4.4.2	Server	141
4.4.3	Librerie utilizzate	143
4.4.4	Distribuzione	145
5	Test	147
5.1	Architettura	147
5.2	Cattura del traffico	149
5.3	Analisi	150
6	Conclusioni e lavori futuri	152

Capitolo 1

Introduzione

La steganografia è uno strumento molto potente e sin dall'antichità è stato utilizzato dall'uomo per nascondere informazioni segrete all'interno di dati innocui. Nel corso degli anni gli ambiti di applicazione sono aumentati a dismisura sfruttando le nuove tecnologie sviluppate nel relativo periodo storico portando alla creazione di diversi tipi di steganografia.

Nel 2003 è stato introdotto il concetto di Network Steganography e da quel momento è diventata una delle tipologie più usate in ambito pratico, le cui implementazioni prendono il nome di *Canali Coperti*. Data la loro grande diffusione nel corso del tempo, dovuta anche ai molteplici ambiti di applicazione che variano dal terrorismo all'incremento della sicurezza, è sorta la necessità di fornire alla comunità scientifica una tassonomia. Per raggiungere quest'ultimo obiettivo è stato creato il progetto "Information Hiding Project"¹ che utilizza un approccio basato su modelli per classificare le varie tecniche. Altro aspetto correlato al grande numero di canali coperti è quello di avere la possibilità di paragonarne le prestazioni, di conseguenza sono stati proposti degli indici con il fine di registrare l'efficienza e l'efficacia delle tecniche.

Oltre ad un'analisi di tutti gli aspetti appena citati, in questa tesi riportiamo le tecniche proposte in letteratura in diversi ambiti ed anche le relative implementazioni pratiche. Insieme a quest'ultime abbiamo inserito anche le implementazioni di canali coperti disponibili in rete che sono non accompagnate da un articolo scientifico.

¹<https://patterns.ztt.hs-worms.de>

All'interno del documento è possibile trovare anche un'analisi dettagliata della nostra proposta di implementazione di un canale coperto. Scelte alcune caratteristiche per noi importanti abbiamo sviluppato un PoC composto da una parte client e una server che sfrutta le intestazioni HTTP per una comunicazione unidirezionale. Ne viene descritta l'idea su cui si basa, i dettagli implementativi e i risultati dei test che ne accertano l'efficacia.

In particolare questo documento è così organizzato:

- Capitolo 1: l'introduzione corrente;
- Capitolo 2: il background contenente le informazioni relative alla storia, ai tipi di steganografia, la terminologia utilizzata, i scenari di applicazione, i parametri di confronto, la tassonomia e le proposte presenti in letteratura;
- Capitolo 3: le implementazioni disponibili con sezioni dedicate alle tipologie di licenze, agli strumenti utili nello sviluppo di una soluzione e ai tool già presenti in rete;
- Capitolo 4: la descrizione del nostro PoC compresa la fase di scelte iniziale, l'idea e i dettagli implementativi;
- Capitolo 5: il processo di test sul traffico generato dal nostro prototipo;
- Capitolo 6: le conclusioni e gli sviluppi futuri che potrebbe avere il nostro studio.

Capitolo 2

Background

In questo capitolo verrà effettuata un'introduzione all'argomento della network steganography. Partendo dalla sua definizione verranno descritti i principali utilizzi, le tecniche pubblicate in letteratura fino ad oggi, i principali parametri di valutazione e le varie proposte di tassonomia susseguitesi nel corso del tempo.

2.1 Steganografia

Il termine “Steganografia” deriva dal greco ed è composto da due parole “στεγανός” e “γραφία”, che significano rispettivamente “nascosto” e “scrittura”, infatti già dalla sua etimologia si intuisce che questa disciplina ingloba tutte quelle tecniche il cui scopo è di rendere nascosta una comunicazione [3]. Da prestare attenzione al fatto che l'obiettivo di questi sistemi è quello di occultare l'esistenza stessa della comunicazione a possibili ascoltatori o terze persone, e non quello di rendere incomprensibile il messaggio, compito di cui se ne occupa la Crittografia.

Se Alice e Bob comunicano sfruttando tecniche steganografiche quindi, è perché non vogliono che Carl, un avversario in ascolto, sappia che si stiano scambiando informazioni, mentre se utilizzano la crittografia vorranno solo tenere nascosto a Carl il contenuto dei messaggi.

Le prime testimonianze dell'uso di tecniche steganografiche risalgono all'antica Grecia dove negli scritti si parla di tavolette di legno incise e poi ricoperte di cera, sulla quale era riportato il falso messaggio da recapitare. Altra testimonianza, non

meno famosa, è quella dello schiavo persiano il quale fu rasato, sulla testa fu tatuato il messaggio e una volta ricresciuti i capelli fu inviato a destinazione con il solo compito di tagliarseli nuovamente. Nel corso della storia quindi le tecniche steganografiche hanno giocato un ruolo fondamentale nelle comunicazioni segrete sviluppandosi di pari passo con le tecnologie proposte nel rispettivo periodo storico. Una crescita esponenziale di questo settore si è iniziata a registrare nel XX° secolo, a causa anche delle guerre mondiali, per poi raggiungere i suoi apici con l'avvento dei computer e di Internet.

2.2 Tipi di steganografia

Prima di scendere nel dettaglio, data la sua vastità, è opportuno presentare un quadro preciso dell'argomento di cui stiamo per trattare. Oggigiorno i principali scopi per cui viene impiegata la steganografia sono due:

- la protezione contro l'individuazione di dati segreti (data hiding);
- la protezione contro la rimozione di informazioni (document marking).

Ognuna di queste categorie si suddivide in altre due sottocategorie: la prima si differenzia in *steganografia iniettiva e generativa*, mentre la seconda in *watermarking e fingerprinting*. La differenza tra iniettiva e generativa è che nella prima il messaggio segreto viene “iniettato” all'interno di un traffico autentico, legittimo, mentre nella seconda il traffico, all'interno del quale si inserisce il messaggio segreto, viene generato da zero cercando di rispettare delle caratteristiche verosimili.

Trovandoci nell'era dell'elaborazione digitale si parla oramai di steganografia digitale, all'interno della quale, in base al supporto di copertura utilizzato, è possibile effettuare una classificazione delle varie tecniche [168]:

- Text steganography.
- Image steganography.
- Audio steganography.
- Video steganography.

- Network steganography.

Rispetto ad altri domini di steganografia, l’obiettivo della steganografia di rete non è archiviare ma trasferire i dati [117]. Dato lo scopo di questo studio nelle prossime sezioni andremo ad analizzare in modo più approfondito soltanto l’ultima di queste categorie per non riportare informazioni inutili alla comprensione dell’argomento e quindi appesantire in modo vano il documento.

2.3 Network steganography

Il concetto di steganografia di rete è stato proposto per la prima volta da Krzysztof Szczypliński nell’anno 2003 [173]. La sua introduzione ha contribuito a spostare in maniera importante l’attenzione su questo nuovo tipo di steganografia che fino a quel momento era stato poco esplorato in funzione di tecniche ben più note come la steganografia di immagini. Questo recente ramo è stato protagonista di un importante sviluppo negli ultimi decenni per quanto riguarda l’occultamento di comunicazioni, contribuendo nell’ambiente scientifico con un considerevole numero di nuovi metodi di steganografia di rete. Se prima i supporti steganografici erano rappresentati maggiormente da immagini e tracce audio, dall’affermazione della network steganography i principali vettori sono diventati i pacchetti di dati e i frame che viaggiano nelle reti di computer [19][202]. Questo nuovo settore è in continuo movimento e anche se si basa principalmente sulle reti di computer non è difficile trovare in letteratura tecniche specifiche che riguardano anche altri ambienti come quello industriale [213]. Jun O. Seo afferma che la steganografia di rete ha tre vantaggi rispetto alla steganografia dei media digitali [161]: ha una migliore anti-rilevabilità, il ciclo di vita delle informazioni segrete è più breve e la larghezza di banda è più flessibile e non è limitata alla dimensione del file di copertura.

2.4 Terminologia

Per descrivere queste tecniche sono stati usati molti termini nel corso degli anni e tra questi registriamo “canale segreto”, “occultamento di informazioni”, “steganografia”. Questa divergenza sui termini da utilizzare è dovuta a due fattori principali: nel

tempo sono state fornite definizioni che descrivono in maniera leggermente diversa queste tecniche e alcune espressioni, nei primi periodi dalla nascita di questo settore, non erano state ancora coniate [207]. Di seguito definiamo il significato di ogni termine utilizzato in questo documento:

- *Covert Channel (canale nascosto o segreto)*: si tratta dell'occultamento di informazioni nei protocolli di rete, attraverso il quale due peer riescono a scambiarsi in modo segreto dei dati.
- *Overt Channel (canale palese)*: un canale di comunicazione noto con il quale, attraverso traffico di rete legittimo, mittente e destinatario di scambiano informazioni.
- *Mittente nascosto (Secret Sender, SS)*: riserviamo l'aggettivo nascosto alle parti coinvolte nella comunicazione che si svolge attraverso il canale segreto, in questo caso particolare ci si riferisce a colui che invia i dati.
- *Destinatario nascosto (Secret Receiver, SR)*: è l'altra parte coinvolta nella comunicazione segreta, colui che riceve i dati o li estrapola grazie alla giusta interpretazione di alcune informazioni del traffico legittimo.
- *Mittente palese (Overt Sender, OS)*: come nel caso della comunicazione segreta, anche in quella palese vengono coinvolte due figure, e questa è colei che invia i dati nel canale palese.
- *Destinatario palese (Overt Receiver, OR)*: colui che riceve i dati dal canale palese, da sottolineare che SS e SR molto spesso non combaciano con OS e OR e quindi quest'ultimi non sono a conoscenza che la loro comunicazione viene sfruttata per altre operazioni da terze persone.
- *Traffico di copertura o legittimo*: la comunicazione tra OS e OR sfruttata per inserire al suo interno un canale nascosto.
- *Messaggio nascosto o segreto*: le informazioni segrete scambiate tra SS e SR attraverso il canale nascosto.
- *Steganografia*: tecnica generale di occultamento di informazioni in un contenuto.

- *Processo di incorporamento*: processo tramite il quale il mittente del canale nascosto inietta i dati segreti all'interno della trasmissione legittima.
- *Processo di estrazione*: processo svolto dal SR tramite il quale riesce ad estrarre le informazioni segrete dal canale palese. Queste possono essere rappresentate fisicamente da bit oppure ottenute tramite la giusta interpretazione di alcune caratteristiche del flusso di dati come il tempo di interarrivo tra pacchetti: un intervallo più alto di una certa soglia prestabilita viene valutato come “1” e viceversa.
- *Stegokey*: chiave segreta scambiata tra SS e SR per svolgere i processi di incorporamento e estrazione dei dati nascosti nel canale segreto [142].
- *Stegobit*: il bit dei dati segreti che viene inserito nel pacchetto preso in considerazione.

2.5 Problema del prigioniero

Quando si affronta il tema dei canali segreti è d'obbligo introdurre l'argomento con il problema del prigioniero, teorizzato da Simmons nel 1983 [167], in quanto rappresenta il modello de facto di questo tipo di comunicazione. Alice e Bob sono due persone incarcerate che vogliono progettare un piano di fuga, ma ogni loro dialogo è controllato dal direttore della prigione Carl, il quale, nel caso trovasse sospette alcune loro comunicazioni, spedirà i due in isolamento, rendendo di fatto impossibile la fuga. In queste condizioni quindi Alice e Bob devono escogitare un modo per scambiarsi messaggi apparentemente innocui agli occhi di Carl, ma contenenti informazioni segrete. Quest'ultime saranno inserite ed estratte dai messaggi legittimi, cioè quelli che leggerà anche Carl, tramite una qualche tecnica segreta che i due condividono e conoscono. In Figura 2.1 possiamo vedere una modellazione del problema del prigioniero.

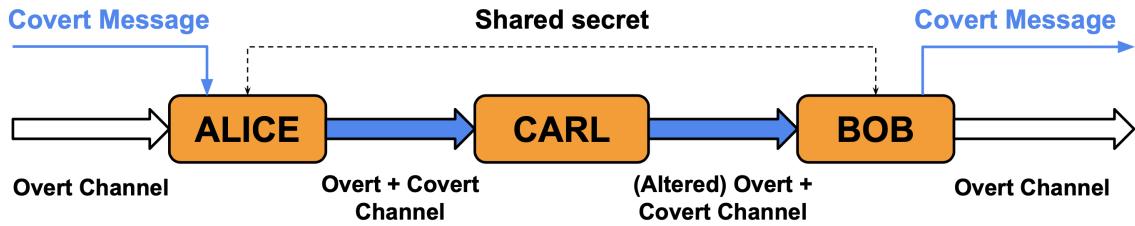


Figura 2.1: Problema del prigioniero

Una soluzione banale può essere quella utilizzata dai tedeschi nelle comunicazioni durante la seconda guerra mondiale: andando a selezionare solo le seconde lettere delle parole del messaggio si andava a formare il contenuto del messaggio nascosto. Un esempio è riportato in Figura 2.2.



Figura 2.2: Comunicazione durante la II° guerra mondiale contenente un messaggio segreto

Per quanto riguarda i possibili comportamenti di Carl facciamo riferimento allo studio di Craver del 1998 nel quale descrive i diversi tipi di avversari [37]:

- un avversario passivo può solo spiare il canale ma non può alterare nessun messaggio.
- un avversario attivo può modificare leggermente i messaggi, ma senza alterare la semantica.
- un avversario maligno può alterare i messaggi senza incappare in conseguenze (sono molto rari).

Successivamente questo scenario è stato esteso alle reti di computer da Hendel nel 1996 [67]. Alice e Bob utilizzano due computer per scambiarsi i messaggi, gestiscono quindi un canale di comunicazione palese al cui interno è contenuto un canale nascosto. I due condividono un segreto che gli permette di codificare e decodificare i

messaggi ma anche di utilizzare i giusti parametri per poter usare il canale nascosto. Rifacendoci allo studio di Craver possiamo immaginare Carl come colui che gestisce la rete e monitora il traffico su di essa, avendo la possibilità, qualora lo ritenesse opportuno, di alterare o interrompere il traffico per eliminare presunti canali nascosti. Nella realtà Alice e Bob possono essere tranquillamente la stessa persona, come nel caso di un malintenzionato che cerca di esfiltrare i dati riservati da un sistema di computer, oppure Bob può essere rappresentato da più persone nel caso in cui il canale nascosto consentisse la comunicazione multicast.

2.6 Covert channel

Il processo di comunicazione nascosta, di cui si tratta in questa tesi, nella pratica viene implementato attraverso i cosiddetti “canali nascosti”. Questa espressione è comparsa in letteratura nel 1973 grazie a Lampson [89], il quale ha definito questi canali “non destinati affatto al trasferimento di informazioni”.

Successivamente, come riportato nella sezione precedente, nel corso degli anni sono state proposte altre definizioni dell’argomento che si sono scostate da quella riportata, portando di fatto ad ampliare i confini e i termini con i quali ci si riferisce a questa materia. Nel 1985 è stato proprio il Dipartimento della Difesa degli Stati Uniti [184] a proporre una nuova definizione: secondo l’ente governativo americano i canali nascosti sono “qualsiasi canale di comunicazione che può essere sfruttato da un processo per trasferire le informazioni in modo da violare la politica di sicurezza del sistema”. Più tardi, nel 1993, la definizione avanzata da Gligor [64] riprende di fatto quella americana, diventando in effetti quella maggiormente riconosciuta nell’ambiente scientifico.

Ci preme sottolineare come sia possibile notare che nel corso degli anni si è arrivati ad avere una definizione sempre più completa e precisa di cosa rappresenti oggi un canale nascosto. Un canale segreto mira a non essere notato da osservatori di terze parti malintenzionati/non autorizzati, anche quando questi intercettano il traffico di rete palese.

2.7 Scenari di comunicazione

I possibili scenari di comunicazione che vedono coinvolti Alice e Bob, rispettivamente SS e SR, dipendono da diversi fattori. Prendendo in analisi gli elementi più importanti possiamo raggruppare tutte le possibilità in sei situazioni. Per poter procedere con la definizione di questi scenari ci basiamo prima di tutto sul fatto se SS e SR combacino o no con OS e OR, e quindi siano o meno degli intermediari, ma anche sul fatto che una volta effettuato il processo di estrazione il mittente del canale nascosto sia in grado di ripristinare il traffico originale, rendendo così il canale nascosto reversibile.

Nel primo caso analizzato Alice e Bob combaciano con il mittente e il destinatario del canale palese. In questa situazione Alice è in grado di manipolare il traffico di copertura per massimizzare la capacità e la furtività del canale nascosto (cioè inviare più dati possibili mantenendo un buon grado di non rilevabilità). Naturalmente non è possibile avere dei valori ottimi per entrambe le caratteristiche contemporaneamente perché, per esempio, se si vuole ottenere una buona furtività del canale questa è penalizzata da un'alta capacità; più dati vengono spostati in breve tempo e più facile sarà per i sistemi di monitoraggio rilevare la fuga di dati. Posizionando Bob prima del destinatario del canale palese si ottengono altre due possibili situazioni. Queste si differenziano dal fatto che Bob in un caso, dopo aver estratto le informazioni segrete, è in grado di eliminare tutte le modifiche effettuate da Alice per la creazione del canale nascosto (canale reversibile), mentre nell'altro no, permettendo di fatto al traffico modificato di arrivare al destinatario palese. I canali reversibili sono quindi tra i più furtivi mai analizzati perché l'unico sistema in grado di intercettarli è il firewall della rete dalla quale escono, in quanto il traffico che raggiunge il destinatario è stato riportato alla sua forma originale e quindi supererebbe anche il test di controllo basato sul confronto tra i pacchetti inviati e quelli ricevuti [123]. Quest'ultima contromisura risulta essere molto dispendiosa e quindi non applicata nella realtà.

Considerando Alice come un intermediario poi si ottengono gli altri tre scenari. L'unica differenza sostanziale con le precedenti opzioni sta nel fatto che Alice non vuole o non è in grado di creare un canale palese da sfruttare, perciò utilizza delle comunicazioni già esistenti per incorporare i dati nascosti e inviarli. La capacità del canale nascosto quindi dipende molto dalle caratteristiche del canale palese sfruttato. Date le analogie, per quanto riguarda Bob, le osservazioni effettuate in precedenza

sono valide anche in queste situazioni [67]. Quando ci riferiamo a intermediari o man-in-the-middle non bisogna solo prendere in considerazione situazioni dove il SS e il SR sono presenti su macchine diverse rispetto a quelle dei OS e OR, è vero che possono trovarsi su router e gateway attraversati dalla comunicazione, ma possono anche tranquillamente trovarsi a livelli inferiori dello stack del protocollo di rete [209]. Nel caso quindi in cui i due mittenti non coincidono ma si trovano sulla stessa macchina il SS necessita di diritti di amministratore per poter modificare i pacchetti in transito (ultimi 3 scenari della Figura 2.3), mentre se avesse i diritti di utente può procedere solo con la steganografia generativa (primi 3 scenari della Figura 2.3).

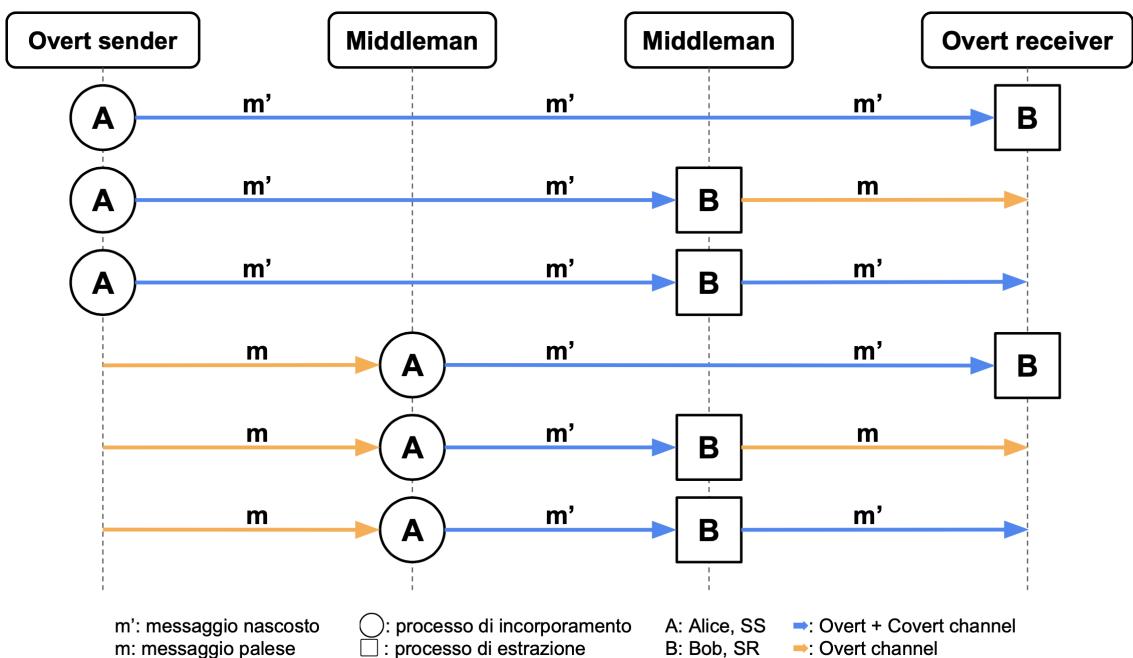


Figura 2.3: Possibili scenari

Nella Figura 2.3 sono riportate le possibili combinazioni di posizionamento dei due protagonisti della comunicazione segreta. Questo posizionamento è influenzato anche dall'effettivo utilizzo del canale segreto: se lo scopo è aggirare i sistemi di censura di un paese [43] molto probabilmente i mittenti/destinatari palesi combaceranno con quelli nascosti, mentre se viene utilizzato da un hacker questi probabilmente saranno intermediari. Specialmente in questo ultimo esempio è da considerare il fatto

che mittente e destinatario potrebbero essere rappresentati fisicamente dalla stessa persona.

2.8 Applicazioni

Nel pensiero comune la prima idea che viene associata all'espressione “canale coperto” è quella dell'esfiltrazione di dati, cioè l'utilizzo da parte di un malintenzionato per rubare dei dati da un sistema aggirando le misure di sicurezza. Certamente i canali segreti vengono utilizzati in modo inappropriato ma il loro grande sviluppo è dato anche da molti altri tipi di applicazioni che non sono dannosi. In letteratura possiamo trovare esempi di utilizzo di questi canali nascosti per qualsiasi ragione, ma la caratteristica che li accomuna tutti è una relazione contraddittoria tra due parti: agenzie governative contro organizzazioni criminali, hacker contro dipartimenti IT aziendali, cittadini dissenzienti contro i loro governi.

Come già scritto nella Sezione 2.1 queste tecniche mirano a nascondere l'esistenza stessa della comunicazione e non le informazioni trasmesse, infatti vengono utilizzate da agenzie governative, criminali, organizzazioni terroristiche e tutte quelle entità che hanno interesse a mantenere segrete le loro comunicazioni. L'uso della crittografia nasconde i dati scambiati ma permette comunque di osservare il traffico e ricostruire i modelli di comunicazione e i soggetti coinvolti. Spesso solo la prova che ci sia stata una comunicazione, anche se cifrata, è sufficiente per rilevare l'inizio di attività illecite o scoprire strutture organizzative. Riportiamo alcuni esempi di utilizzo in vari ambiti presenti in letteratura.

Terrorismo. In uno studio è stato ipotizzato che dei terroristi usassero dei canali segreti per coordinare le loro attività [108] mentre in un altro è stato dimostrato che i terroristi nel Regno Unito comunicassero con i propri contatti in Pakistan sfruttando una comunicazione riguardante l'ordinazione di merci per l'apertura di un nuovo negozio [136].

Spie Aziendali. Spie aziendali o hacker, una volta che hanno compromesso i sistemi informatici, sono obbligati a comunicare, con le macchine danneggiate, in modo silente per l'esfiltrazione di dati o l'esecuzione di comandi da remoto. Queste attività

generano traffico di rete che, se non fosse nascosto, sarebbe rilevato immediatamente dai sistemi di sicurezza che a loro volta avvertirebbero l'amministratore di rete. Oltre a persone esterne, potrebbero essere proprio gli impiegati stessi a voler utilizzare i canali nascosti per aggirare i firewall aziendali, o quelli del provider di servizi internet, per accedere a delle risorse in rete altrimenti irraggiungibili. Tra i possibili soggetti all'interno di un'azienda interessati all'utilizzo di queste tecniche non vanno esclusi gli amministratori di rete perché possono utilizzarli per nascondere le comunicazioni relative alla gestione della rete e degli honeypot [55][145].

Censura. Spostando l'attenzione a situazioni di maggiori dimensioni troviamo quei casi dove è il governo a imporre misure restrittive volte ad applicare una qualche forma di censura [93]. La censura di Internet è in genere praticata dai governi [198, 137] per tre ragioni principali: bloccare l'accesso dei cittadini a determinati servizi, interrompere strumenti come Tor¹ e identificare gli utenti coinvolti nell'elusione. Nonostante non ci siano dati completi e accurati sulle capacità tecniche dei censori le prove empiriche lasciano pensare che questi effettuino solo delle indagini sulla velocità di linea o qualcosa di simile. In particolare svolgono delle analisi statistiche, che richiedono una buona quantità di risorse, solo su parte dei flussi osservati e inoltre non memorizzano tutto il traffico per lungo tempo. Questo perché la maggior parte dei censori non vuole infastidire gli utenti regolari, che non sono coinvolti nell'elusione, interrompendo in modo netto la fruizione di servizi popolari. Nonostante quella cinese sia una delle più famose, questa censura non blocca l'utilizzo di GitHub [35] e alcuni servizi di Google, data la loro grande diffusione. A differenza della precedente ce ne sono alcune che, nonostante provochino il malcontento degli utenti, bloccano servizi anche molto usati e importanti: l'Etiopia blocca Skype [10], l'Iran a volte interrompe le comunicazioni SSL [24] e addirittura l'Egitto durante una rivolta ha tolto completamente Internet al popolo [87]. In questi casi i canali nascosti rappresentano una soluzione fondamentale per il trasporto sicuro di informazioni sia in entrata che in uscita [108, 151]. In paesi dove comunque non c'è una censura queste tecniche vengono altresì utilizzate, per esempio, dai giornalisti d'inchiesta che intendono raccogliere e scambiarsi dati e prove per mantenerli segreti fino alla loro pubblicazione.

¹Un software libero che permette la navigazione anonima sul web basato sul protocollo di rete onion routing. <https://www.torproject.org>.

Virus informatici e anonimizzazione. Inoltre i canali nascosti sono utilizzati da virus o worm per diffondersi inosservati oppure per scambiarsi informazioni utili al loro funzionamento [197]. In letteratura sono documentate più occasioni in cui è stato dimostrato che queste tecniche possono essere sfruttate per contrastare tecniche di anonimizzazione [128, 201, 22, 131, 130], ma anche per migliorare la sicurezza di protocolli come VoIP [112].

Incremento della sicurezza. Anche il processo di autenticazione può essere coinvolto da queste tecniche, infatti metodi come il “port knocking” e quello proposto da Mazurczyk utilizzano i canali segreti e la steganografia per inserire le informazioni di controllo, inclusi i dati di autenticazione, nei flussi di dati effettivi [42, 110, 111, 204]. Infine riportiamo che ci sono stati diversi ricercatori che hanno sviluppato tecniche di tracciamento dei pacchetti, molto utili per esempio in attacchi DoS per capire la provenienza e risalire agli aggressori, che sfruttano i canali nascosti [84, 147, 190, 76].

2.9 Valutazione dei canali

Come affronteremo nelle sezioni successive, sono stati presentati diversi tipi di canali nascosti, ed ognuno sfrutta tecniche e vulnerabilità diverse. Data quindi la grande varietà di proposte resta difficile poter valutare in modo oggettivo le prestazioni di ogni singola tecnica rispetto a un’altra. Per limitare il più possibile questo problema si è deciso di basarsi su alcuni attributi comuni a tutti. La maggioranza degli studi incentra il paragone tra canali coperti sugli indici di robustezza, capacità e furtività [209, 91, 171], ma in alcuni casi vengono presi in considerazione anche altri indici [25].

- *Robustezza:* la robustezza indica la capacità del canale di resistere a perturbazioni della rete. I canali segreti di rete spesso incontrano dispositivi di archiviazione e inoltro, firewall, server proxy o altri dispositivi simili e la sua abilità nel resistere a tali circostanze viene descritta da questo attributo. In alcuni studi viene utilizzata una semplice scala di valori (Alto, Medio, Basso) per poter comprendere meglio le potenzialità del metodo analizzato. I principali protocolli di rete su cui si basano le tecniche sono TCP e UDP, dove il primo premia l'affidabilità rispetto alla velocità mentre il secondo garantisce maggiori prestazioni

a discapito di alcune perdite di pacchetti. Alla luce di ciò, quando si utilizza UDP, è bene integrare all'interno del nostro covert channel dei meccanismi per migliorare l'affidabilità e l'integrità dei messaggi.

- *Capacità*: la seconda caratteristica presa in considerazione nelle valutazioni è la capacità di trasmissione, detta anche throughput. Questa capacità deriva dal rapporto dati/tempo ed è strettamente influenzata da due fattori. Essendo correlata inversamente alla furtività, terzo attributo analizzato, questa non può essere sfruttata al massimo perché più è grande la quantità di dati trasmessi su un canale coperto e maggiore è il rischio che questo venga scoperto. Inoltre, siccome vengono sfruttati flussi di comunicazione per l'instaurazione di questi canali nascosti, la capacità trasmissiva di quest'ultimi dipenderà molto anche dalla tipologia e quantità del traffico sfruttato. Questo porta a una valutazione generalmente di bit al secondo (b/s) o al massimo di kb al secondo (kb/s) in quanto si parla di pochi bit modificati per ogni pacchetto e delle volte neanche di tutti i pacchetti.
- *Furtività*: terzo attributo analizzato è la furtività che in alcuni articoli viene anche riportata come segretezza o indistinguibilità. Questa caratteristica determina la facilità con cui un canale nascosto può essere rilevato andando a confrontare il flusso con il traffico nascosto iniettato e il flusso legittimo. Un canale nascosto viene definito “rumoroso” se per comunicare sfrutta eccessivamente le risorse andando a danneggiare il corretto funzionamento di queste. Riuscire a comunicare mantenendo i vari parametri e comportamenti del traffico sfruttato nella norma ci rende “indistinguibili” da esso e quindi invisibili ai sistemi di monitoraggio. Come già introdotto col rapporto inverso tra capacità e furtività si può intuire che non è possibile massimizzare questi tre valori, ma nella creazione e implementazione di un canale nascosto va trovato il giusto equilibrio tra questi per poter raggiungere gli obiettivi prefissati.
- *Meccanismo di occultamento*: può essere analizzato il meccanismo di un canale nascosto, cioè come è stato costruito e cosa fa per “nascondersi” e trasportare le informazioni. Ogni tecnica ha un proprio meccanismo di occultamento e spesso è proprio questo a influenzare maggiormente le altre caratteristiche.

- *Tipo di canale*: un'altra proprietà molto importante è il tipo di un canale, ma dato il grande numero di tassonomie proposte, l'argomento verrà trattato più approfonditamente nella sezione 2.10. Ci limitiamo a dire che una classificazione grossolana prevede la distinzione tra canali di archiviazione, temporali e comportamentali [83].
- *Bit Error Rate*: se il confronto avviene tra canali di cui è stata presentata una effettiva implementazione, un indice indicativo delle potenzialità e del corretto funzionamento è dato da BER (Bit Error Rate), cioè il numero di bit decodificati dal destinatario nascosto nel modo sbagliato [103] .
- *Packet Raw Bit Rate*: un altro indice che descrive le prestazioni dei canali di archiviazione nascosti, e quindi può essere utilizzato per confrontare due o più implementazioni, è il PRBR (Packet Raw Bit Rate), cioè la quantità di informazioni che viene spedita, sotto forma di bit, per ogni pacchetto [114]. Nel caso ideale, se non vengono persi pacchetti, la capacità di alcuni canali di memorizzazione può essere espressa come $C = PRBR \times N$ bps, dove N è il numero di pacchetti inviati in un secondo.

2.10 Tassonomia

Un problema attuale collegato a queste tecniche steganografiche è quello della loro classificazione. Il loro sviluppo ha visto nascere metodi in grado di sfruttare vulnerabilità praticamente in ogni protocollo. Ricordando che una tassonomia è lo studio teorico della classificazione attraverso la definizione esatta dei principi, delle procedure e delle norme che la regolano, possiamo affermare che nel corso degli anni ne sono state proposte diverse riguardanti questo argomento. A volte quelle più recenti vanno ad aggiornare, estendere e migliorare quelle precedenti, ma ci sono casi dove le nuove proposte seguono tutt'altra strada. Alla luce di ciò, indipendentemente dalle categorie e dai metodi di classificazione utilizzati, abbiamo registrato un continuo aumento delle possibili classi. Questi canali coinvolgono diversi aspetti delle comunicazioni e proprio da questo deriva il grande numero di tassonomie presentate: è possibile basarsi sulle caratteristiche stesse del canale, sulle tecniche utilizzate, sui protocolli sfruttati e così via. Cercando di presentare un quadro generale a riguardo,

abbiamo deciso di procedere nel modo più cronologico possibile nel descrivere le varie proposte.

2.10.1 Proposta del 1985

A distanza di alcuni anni dalla loro introduzione in letteratura, nel 1985 [92] è stata definita la prima grande classificazione dei canali nascosti in due categorie: *canali di memorizzazione e canali di temporizzazione*. Questa differenziazione è ancora oggi la più utilizzata, infatti in molti articoli scientifici vengono utilizzate le sigle CSC (Covert Storage Channels) e CTC (Covert Timing Channels) per riferirsi a queste categorie. Le tecniche utilizzate per costruire un canale segreto sono molte ma tutte rientrano in una delle due classi: quelle che vanno a modificare bit di pacchetti, e quindi memorizzano le informazioni direttamente nel traffico, oppure quelle che modificano la tempistica o il comportamento del flusso in modo tale che il destinatario decodifica i dati segreti osservando e interpretando il traffico. Più recentemente a queste due categorie ne è stata aggiunta una terza: *i canali ibridi* [58]. Questi metodi combinano l'utilizzo sia di tecniche di memorizzazione che di temporizzazione.

Sulla scia di questa distinzione, all'interno di diversi studi ed indagini, ne sono state presentate altre che si basano su ulteriori caratteristiche del canale e del traffico. In letteratura è presente la suddivisione tra *canali con copertura prevedibile, variabile e casuale* [209]. Considerando il fatto che con il termine copertura ci si riferisce al traffico palese, possiamo affermare che la classificazione si basa sulle caratteristiche di quest'ultimo, cioè se è praticamente identico ad un flusso di comunicazione normale verrà definito prevedibile, mentre se presenta delle piccole variazioni si parlerà di copertura variabile. Il terzo caso si riferisce a quelle situazioni dove il traffico palese sfruttato per la creazione del canale è composto da un flusso di copertura pseudocasuale, e quindi non rispettando nessun modello comportamentale potrebbe essere più facile da rilevare dai sistemi di monitoraggio.

Cover nel 1991 [36] ha descritto la differenza tra *canali rumorosi e non*. I canali segreti sono comunque canali di comunicazione tra due soggetti e come tutti possono essere caratterizzati da rumore. Il rumore è l'insieme di segnali imprevisti e indesiderati che si sovrappongono al segnale utile. Il rumore provoca una perdita di informazioni o un'alterazione del messaggio che nella pratica si manifestano tramite

sostituzione, inversione e cancellazione di bit. Su un canale non rumoroso questi problemi non si verificano.

Analizzando il traffico di copertura da un altro punto di vista possiamo distinguere tra *canali passivi*, *semipassivi* e *attivi* [36]. In quelli passivi il SS dispone dei diritti di amministratore e sfrutta il traffico esistente tra utenti inconsapevoli mentre in quelli attivi ha solo diritti utente ed è in grado di generarlo, cioè sarà il mittente del canale palese e avrà il pieno controllo su l'intera comunicazione. Quelli semipassivi si hanno quando il SS genera il traffico palese manomettendo applicazioni reali.

Se invece spostiamo l'attenzione sui soggetti coinvolti e sul loro posizionamento nel canale comunicativo potremmo dividere le varie tecniche tra *canali diretti* e *indiretti*. Nel primo caso le informazioni, e quindi i dati segreti, fluiscono direttamente dal SS al SR, mentre nel secondo viene coinvolto un terzo utente ignaro. In questa circostanza possiamo considerare il flusso di informazioni diviso in due parti: la prima va dal mittente a un utente intermedio inconsapevole e la seconda da quest'ultimo al destinatario. Per comprendere meglio questa situazione potremmo immaginare il caso in cui un malintenzionato è riuscito ad entrare in una macchina all'interno di una rete e nel router subito fuori il firewall. Inviando traffico che rispetta apparentemente le politiche di sicurezza fuori dalla rete, può tranquillamente recuperare i dati segreti dal router. Come già sottolineato i vari utenti possono essere rappresentati non solo da persone fisiche ma anche da dispositivi e processi.

2.10.2 Proposta del 2007

Altro documento molto importante, tra i più citati sull'argomento, è l'indagine effettuata da Zander nel 2007 [207]. In questo documento, per la prima volta dall'introduzione dei canali segreti, si è cercato di creare una tassonomia completa di tutte le tecniche dal 1987 al 2007, riportando anche le relative contromisure. La strategia utilizzata per categorizzare è quella di dividere dapprima i canali di memorizzazione da quelli di temporizzazione, e poi analizzare in quale livello e come agiscono le varie tecniche. Seguendo l'ordine del documento riportiamo la classificazione proposta.

Unused Header Bit. Come prima classe analizzata troviamo quella delle tecniche steganografiche di rete che sfruttano i bit inutilizzati negli header dei vari protocolli.

Un esempio è il campo TOS (Type Of Service) del TCP che, nonostante abbia grandi potenzialità come specificare una priorità o un indirizzamento specifico del pacchetto, non è mai stato usato nella pratica. Oggi il campo è stato deprecato e sostituito con i 6 bit del campo DS (Differentiated Services), che è simile al TOS e indica come il router deve trattare i pacchetti, e i 2 bit del campo ECN (Explicit Congestion Notification), che consente ai router di origine di conoscere la congestione di un router prima che questo cominci effettivamente a far cadere i pacchetti. Un altro campo utilizzabile è Don't Fragment del protocollo IP che, se impostato a 1, non permette la divisione del pacchetto in unità di dimensione ridotta e ne richiede la ritrasmissione. Sono state presentate delle proposte che utilizzano valori non assegnati nel campo Code del protocollo ICMP. La struttura dei pacchetti TCP è osservabile nella Figura 2.14 mentre quella dei pacchetti IP nella Figura 2.9.

Header Extensions and Padding. Successivamente abbiamo i metodi che sfruttano l'estensione degli header e il padding. Ogni protocollo prevede degli header standard, che non possono mancare all'interno dei pacchetti, ma in alcuni casi ne sono previsti di aggiuntivi per aumentarne le funzionalità. Proprio questi, vista la loro natura, possono essere soggetti a un minor numero di clausole. Altre proposte che applicano un ragionamento simile sono quelle che sfruttano il padding dei pacchetti, che essendo solo bit riempimento, in quanto il frame deve avere una grandezza prefissata, può non essere controllato durante la trasmissione.

IP Identification and Fragment Offset. La terza categoria si basa sullo sfruttamento dei campi ID e Fragment Offset del protocollo IP. Il campo di intestazione ID viene utilizzato per riassemblare i pacchetti IP frammentati e l'unico requisito è che ogni ID identifichi univocamente un pacchetto IP per un certo periodo di tempo [143]. Il Fragment Offset invece viene utilizzato per determinare in quale sequenza i frammenti devono essere riassemblati. Nella Figura 2.9 sono riportati i vari header del protocollo IP.

TCP Initial Sequence Number Field. Altra tipologia definita nell'indagine riguarda i metodi che utilizzano i valori dei numeri di sequenza del TCP, in particolare il Numero di Sequenza Iniziale (ISN). Questi servono per coordinare i dati ricevuti e

trasmessi garantendo affidabilità alla comunicazione. L'ISN scelto deve far in modo che i successivi numeri di sequenza non si sovrappongano con quelli delle precedenti comunicazioni [144]. In Figura 2.14 osserviamo il suo posizionamento nella struttura del pacchetto.

Checksum Field. Tra i vari campi dell'intestazione che possono essere utilizzati per la creazione di canali segreti c'è anche il campo Checksum. Nonostante questo rappresenti una sorta di firma di tutto il pacchetto, garantendone l'integrità, è possibile modificare il suo valore e aggiungere un header di intestazione, tra quelli facoltativi, per rispettare ancora i vincoli e superare i sistemi di controllo [1].

Modulating the IP Time To Live Field. Un altro campo soggetto a modifiche al fine di inviare dati segreti è il campo Time To Live (TTL), anche lui osservabile in Figura 2.9 insieme agli altri header. Questo valore stabilisce per quanto tempo un pacchetto può "restare in vita", cioè fino a quando può circolare in rete senza essere scartato. Ci sono proposte che attraverso la modulazione del campo TTL riescono a trasmettere dati e informazioni, ma alcune non tenendo conto dei valori iniziali tipici risultano avere prestazioni peggiori in termini di non rilevabilità.

Modulating Address Field and Packet Length. Una strategia usata da diverse tecniche riguarda la modulazione dei campi degli indirizzi del mittente e del destinatario, i più importanti e più utilizzati sono quelli del protocollo IP, rappresentati nelle Figure 2.9 e 2.10 ma può essere utilizzata a tutti i livelli. Non si vanno a modificare direttamente i valori ma piuttosto intervenendo sulla loro quantità, ordine e altre caratteristiche è possibile includere dei dati. All'interno di questa categoria troviamo anche quelle tecniche che modulano la lunghezza dell'intestazione, dell'estensioni di intestazione e del messaggio per trasmettere dati segreti.

Modulating Timestamp Field. Alcune proposte basano la loro capacità di trasmissione di dati sulla modifica del campo Timestamp del protocollo TCP. Quasi tutte vanno ad intervenire sui bit meno significativi del campo (Least Significant Bit). In Figura 2.14 è riportata la struttura del pacchetto TCP.

Packet Rate/Timing. La maggior parte dei canali di temporizzazione è rappresentato da tutte quelle tecniche che intervengono sulla velocità dei pacchetti, cioè agiscono sul tempo di invio applicando dei ritardi. In questo tipo di canali possiamo distinguere quelli binari, che codificano solo due valori (0 e 1), da quelli multi-rate che ne riescono a codificare di più. Un problema che deve essere gestito nel caso si utilizzassero metodi del genere è la sincronizzazione tra mittente e destinatario.

Message Sequence Timing. Nel corso delle comunicazioni i due soggetti si inviano dei messaggi di controllo per regolare il loro scambio di informazioni [199]. Questi messaggi variano da protocollo a protocollo ed ognuno prevede un certo comportamento, un'anomalia quindi rispetto al funzionamento previsto può codificare un dato segreto. Queste tecniche intervengono sulla tempistica dei messaggi di controllo.

Packet Loss and Packet Sorting. C'è un'ulteriore categoria caratterizzata dalla perdita dei pacchetti o dal loro ordinamento. Sono state presentate tecniche che non vanno a eliminare fisicamente alcuni pacchetti della comunicazione, ma piuttosto saltano dei numeri di sequenza. Se mittente e destinatario condividono la giusta strategia di interpretazione di queste mancanze, allora è possibile sfruttarle per inviare dei dati segreti. Casi molto simili che rientrano in questa categoria sono quelli che invece di eliminare dei pacchetti li riordinano. Per esempio sequenze particolari di numeri di sequenza, di indirizzi di destinazione o altri campi possono rappresentare le informazioni segrete.

Frame Collision. Alcune tecniche presentate in letteratura coinvolgono direttamente il meccanismo Ethernet CSMA/CD (Carrier Sense Multiple Access / Collision Detection), cioè quel protocollo che permette ad un pc di utilizzare una rete Ethernet soltanto se nessun altro la elaboratore la sta già utilizzando. In particolare quando si verifica una collisione tra due pacchetti, questi vengono scartati, i due pc sceglieranno ognuno un tempo di ritardo casuale dopo il quale ritrasmetteranno il frame. Le proposte prevedono di bloccare intenzionalmente i pacchetti di un altro pc e poi trasmettere i dati segreti modulando i tempi di re-invio.

Ad-hoc Routing Protocol. Zander [207] introduce anche una categoria riguardante i protocolli di routing ad-hoc. Nel suo documento riporta tecniche che coinvolgono il DSR (Dynamic Source Routing) e il simile AODV (Ad-hoc On-demand Distance Vector), entrambi protocolli di routing per reti mobili.

Wireless LAN (WLAN). Altra tipologia di tecniche molto numerosa è quella dei canali nascosti all'interno delle reti WLAN. In letteratura è possibile trovare metodi di implementazione di canali nascosti praticamente per ogni versione dello standard 802.11.

Hypertext Transfer Protocol (HTTP). Anche il protocollo HTTP può essere sfruttato per creare dei meccanismi di comunicazione segreta. Le proposte sono molte ed è possibile trovare metodi che impiegano Javascript/HTML, parametri URL, valori dell'intestazione e cookie.

Domain Name System (DNS) Protocol. Uno dei protocolli più presenti in letteratura è senza dubbio il DNS (Domain Name System), il responsabile della risoluzione dei domini internet in indirizzi IP. La sua grande popolarità è data dal grande numero di proposte presentate nei primi anni e dalle diverse contromisure, basate su Machine Learning, che puntano proprio a limitare l'utilizzo di questo tipo di canali.

Other Application Protocol. Zander [207] infine raccoglie nell'ultima categoria tutte le proposte che interessano altri protocolli di comunicazione. All'interno di questa possiamo trovare, per esempio, canali nascosti implementati nelle comunicazioni VoIP, SSH o FTP. Probabilmente data la scarsità di tecniche presenti in quel periodo ha deciso di inserirle tutte nella stessa classe, mentre oggi se usassimo la sua strategia di classificazione, visto il grande utilizzo e le proposte presenti in letteratura, dovremmo sicuramente considerare una classe dedicata solamente alla tecnologia VoIP.

2.10.3 Proposta del 2015

Nonostante l'ottimo lavoro effettuato, la tassonomia proposta [207] presenta dei limiti: all'interno della categoria relativa al protocollo HTTP abbiamo metodi che usano

l'intestazione dei pacchetti a livello applicazione, ma questi potrebbero far parte anche della classe che comprende tecniche che sfruttano i bit di intestazione dei pacchetti. Anche per questo motivo nel corso degli anni si sono registrati diversi tentativi di proporre nuove tassonomie, ma quello di maggior rilievo risale al 2015 ed è stato scritto dallo stesso Zander in collaborazione con Wendzel, Fechner, Herdin [194].

In questo articolo si analizzano 109 tecniche di canali coperti che vengono classificate in 11 categorie, affermando poi che circa il 70% delle proposte appartiene solo a 4 classi: Add Redundancy, Value Modulation, Reserved/Unused e Random Value. La novità presentata sta nel fatto che viene applicato il Pattern Language alla classificazione dei canali segreti, e quindi tramite i modelli è possibile raggruppare i metodi simili. Attraverso la tecnica dei pattern si possono creare delle tassonomie basate su modelli in qualunque ambito, compreso quello della steganografia di rete. Il Pattern Language Markup Language (PLML), basato su XML, è il pattern language standard nel campo dell'interazione uomo-computer [52] ed è in grado di fornire una formalizzazione coerente con le descrizioni dei vari modelli.

Questa classificazione proposta, organizzata in modo gerarchico, risulta essere facilmente estensibile nel caso in cui in futuro vengano proposti nuovi metodi che sfruttano modelli non ancora analizzati, e questo perché si basa meno sugli aspetti tecnici, a differenza delle precedenti proposte, e più sul comportamento astratto comune delle tecniche dei canali nascosti. Gli autori hanno deciso di utilizzare i pattern perché il loro impiego garantisce un maggior numero di vantaggi [159]: sono semplici e facili da leggere per gli addetti ai lavori, facilitano la collaborazione tra più persone coinvolte allo stesso progetto e si basano su conoscenze consolidate. Ogni modello rappresenta una relazione tra un certo problema di progettazione e una soluzione in un dato contesto come è possibile osservare in Figura 2.4.

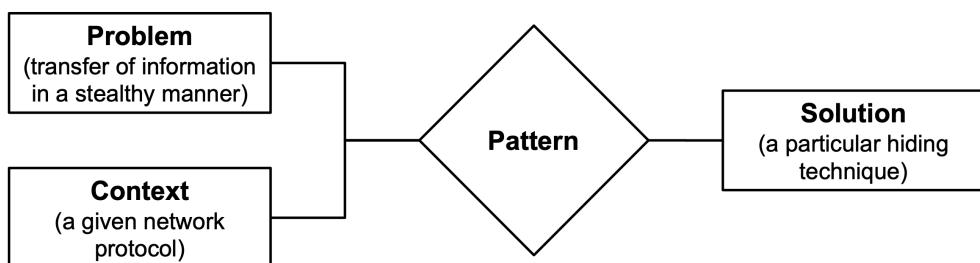


Figura 2.4: Un pattern mette in relazione problema, soluzione e contesto.

Il problema è rappresentato da una descrizione del problema da risolvere, la soluzione si riferisce a un'implementazione per risolvere il problema dato e il contesto descrive un insieme di casi in cui il modello può essere utilizzato. I tag usati per descrivere i vari modelli sono i seguenti:

- *pattern id*: identifica il modello all'interno di un catalogo;
- *name*: è importante assegnare un nominativo ad ogni modello per recuperarlo nel caso in cui diventa parte di un secondo modello;
- *alias*: contiene altri nomi, utile nel caso in cui deve recuperare lo stesso modello ma che è contenuto in più cataloghi con diversi nomi;
- *illustration*: scenario applicativo per il modello;
- *context*: specifica le situazioni in cui può essere applicato il modello;
- *solution*: descrive la soluzione a un problema per cui può essere applicato il modello;
- *evidence*: contiene ulteriori dettagli sul modello e sul design, in alcuni casi anche esempi di usi noti del modello;
- *literature*: elenco dei riferimenti alle pubblicazioni relative al modello;
- *implementation*: introduce implementazioni esistenti, frammenti di codice o di implementazioni.

Nella Tabella 2.1 riportiamo gli 11 pattern individuati da Wendzel e suoi collaboratori [194] descrivendo soltanto il tag Illustration e Context in quanto le proposte e le implementazioni verranno trattate nelle sezioni successive. Con PDU (Packet Data Unit) si intende l'unità di informazione, il singolo pacchetto quindi.

P1. Modulazione delle dimensioni:

Illustration: il canale nascosto utilizza le dimensioni di un elemento di intestazione o di una PDU per codificare il messaggio nascosto.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Modifica della struttura

P2. Sequenza:

Illustration: il canale nascosto altera la sequenza degli elementi header/PDU per codificare le informazioni nascoste.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Modifica della struttura

P2.a. Posizione:

Illustration: il canale nascosto altera la posizione di un dato elemento di intestazione/PDU per codificare le informazioni nascoste.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Modifica della struttura → Sequenza

P2.b. Numero di elementi:

Illustration: il canale nascosto codifica le informazioni nascoste in base al numero di elementi di intestazione/PDU trasferiti.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Modifica della struttura → Sequenza

P3. Aggiunta di ridondanza:

Illustration: il canale nascosto crea nuovo spazio all'interno di un determinato elemento di intestazione o all'interno di una PDU in cui nascondere i dati.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Modifica della struttura

P4. Corruzione/perdita della PDU:

Illustration: il canale nascosto genera PDU corrotte che contengono dati nascosti o utilizza attivamente la perdita di pacchetti per segnalare informazioni nascoste.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Modifica della struttura

P5. Valore casuale:

Illustration: il canale nascosto incorpora i dati nascosti in un elemento di intestazione contenente un valore "casuale".

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Conservazione della struttura → Modifica di un attributo

P6. Modulazione del valore:

Illustration: il canale nascosto seleziona uno degli n valori che un elemento di intestazione può contenere per codificare un messaggio nascosto.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Conservazione della struttura → Modifica di un attributo

P6.a. Contenitore:

Illustration: il canale nascosto utilizza la modifica delle maiuscole/minuscole delle lettere negli elementi di intestazione per codificare i dati nascosti.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Conservazione della struttura → Modifica di un attributo → Modulazione del valore

P6.b. Bit meno significativo (LSB):

Illustration: il canale nascosto utilizza i bit meno significativi degli elementi di intestazione per codificare i dati nascosti.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Conservazione della struttura → Modifica di un attributo → Modulazione del valore

P7. Riservato/non utilizzato:

Illustration: il canale nascosto ha codificato i dati nascosti in un elemento di intestazione/PDU riservato o inutilizzato.

Context: Canali di archiviazione nascosti di rete → Modifica del carico non utile → Conservazione della struttura → Modifica di un attributo

P8. Tempo di interarrivo:

Illustration: il canale nascosto altera gli intervalli di tempo tra le PDU di rete (tempi di arrivo) per codificare i dati nascosti.

Context: canali di temporizzazione nascosti di rete

P9. Tasso:

Illustration: il mittente del canale nascosto altera la velocità dei dati di un flusso di traffico da se stesso o da terzi al ricevitore del canale nascosto.

Alias: modello di produttività

Context: canali di temporizzazione nascosti di rete

P10. Ordine PDU:

Illustration: il canale nascosto codifica i dati utilizzando un ordine PDU sintetico per un determinato numero di PDU che scorrono tra il mittente nascosto e il ricevitore.

Context: canali di temporizzazione nascosti di rete

P11. Ritrasmissione:

Illustration: il canale nascosto ritrasmette le PDU inviate/ricevute in precedenza.

Context: canali di temporizzazione nascosti di rete

Tabella 2.1: Pattern proposti nel 2015

Oltre alla rappresentazione gerarchica è possibile classificare i modelli riguardanti i seguenti aspetti aggiuntivi:

- *semantica*: la semantica di una PDU viene modificata se il pattern altera gli elementi di intestazione in modo tale che provoca una diversa interpretazione della PDU. I modelli che mantengono la semantica dei dati in generale attirano meno attenzione.
- *sintassi*: chiamiamo una modifica della struttura della PDU una modifica della sintassi. Questa categorizzazione interessa solo i canali di archiviazione poiché i canali di temporizzazione non intervengono sui dati all'interno del pacchetto.
- *rumore*: tutti i danneggiamenti sotto forma di modifica di bit o perdita pacchetti. I canali di temporizzazione saranno classificati come rumorosi. Non viene preso in considerazione il rumore inserito dai guardiani attivi.

Il documento continua riportando la classe di appartenenza di ognuna delle 109 tecniche analizzate, dimostrando che il 70% circa è incluso in: P3 Aggiunta di ridondanza, P5 Valore casuale, P6 Modulazione del valore e P7 Riservato/Non utilizzato. Riprendendo le descrizioni riportate qui sopra possiamo affermare che la maggior parte delle proposte prevede di inserire i dati segreti in:

- elementi aggiunti di intestazione o di PDU;
- elementi di intestazione che prevedono un valore casuale;
- uno degli n valori che un elemento di intestazione può contenere;
- un elemento di intestazione riservato o non usato.

Successivamente viene descritta la Variazione di Pattern, affinché sia possibile sfruttare appieno le potenzialità del metodo proposto, e come questa classificazione debba influire anche sulle contromisure per rendere il loro sviluppo più facile.

2.10.4 Proposta del 2018

Il precedente lavoro ha rappresentato un ottimo punto di partenza per quanto concerne la classificazione dei canali segreti, infatti senza avere degli standard ufficiali da seguire tutte le successive proposte facevano riferimento a questa tassonomia. Dato il numero sempre crescente di tecniche e il forte sviluppo del settore gli autori hanno aggiornato più volte la tassonomia. I due aggiornamenti principali sono avvenuti uno l'anno successivo alla pubblicazione, nel 2016 [120], e uno nel 2018 [121].

Nel primo caso si è passati da 11 pattern (4 di timing e 7 di storage) a 14 pattern (8 di timing e 6 di storage), l'aumento è dovuto principalmente all'introduzione di un nuovo livello per i modelli temporali, dividendo quelli “agnosticci dal protocollo” da quelli “consapevoli del protocollo”. Inoltre è stato rimosso il pattern “P4 Corruzione/perdita della PDU” e sono stati aggiunti a quelli temporali il modello “Perdita artificiale dei messaggi/pacchetti” e “Collisione di frame”, oltre ad altre modifiche e rinominazioni di altri pattern. Nell'ultimo aggiornamento per la prima volta vengono trattati anche sottoargomenti come la “variazione dei pattern”, “combinazione dei pattern” e “hopping del pattern”. Successivamente viene fornita la descrizione aggiornata dei modelli che viene riportata di seguito:

- *Rate/throughout*: Il mittente del canale nascosto altera la velocità dei dati del traffico da se stesso o da terzi al ricevitore del canale nascosto.
- *Inter-packets time*: Il canale nascosto altera gli intervalli di tempo tra le PDU di rete (tempi di interarrivo) per codificare i dati nascosti.
- *Message timing*: I dati nascosti sono codificati nei tempi delle sequenze di messaggi, ad es. confermando ogni ennesimo pacchetto ricevuto o inviando comandi m volte.
- *Artificial loss*: Il canale nascosto segnala le informazioni nascoste tramite la perdita artificiale dei messaggi trasmessi (PDU).

- *Frame collision*: Il mittente provoca collisioni di frame artificiali per segnalare informazioni nascoste.
- *Temperature*: Il mittente influenza la temperatura della CPU di un nodo di terze parti, ad es. utilizzando il traffico a raffica. Questo influenza il clock skew² del nodo. Il clock skew può quindi essere interpretato dal ricevitore nascosto interagendo con il nodo.
- *Retransmission*: Un canale nascosto ritrasmette le PDU precedentemente inviate o ricevute.
- *Message ordering*: Il canale nascosto codifica i dati utilizzando un ordine PDU sintetico per un determinato numero di PDU che scorrono tra mittente e ricevitore nascosti.
- *Size modulation*: Il canale nascosto utilizza le dimensioni di un elemento di intestazione o di una PDU per codificare un messaggio nascosto.
- *Sequence modulation*: Il canale segreto altera la sequenza degli elementi header/PDU per codificare le informazioni nascoste. Questo modello si divide ulteriormente in: P2.a. Posizione e P2.b. Numero di modelli di elementi.
- *Add redundancy*: Il canale nascosto crea nuovo spazio all'interno di un dato elemento di intestazione o all'interno di una PDU in cui nascondere i dati.
- *Random value*: Il canale nascosto incorpora i dati nascosti in un elemento di intestazione contenente un valore (pseudo-)casuale.
- *Value modulation*: Il canale nascosto seleziona uno degli n valori che un elemento di intestazione può contenere per codificare un messaggio nascosto. Questo schema si divide ulteriormente in: P6.a. Schema del caso e P6.b. Pattern di bit meno significativi (LSB).
- *Reserved/unused*: Il canale nascosto codifica i dati nascosti in un elemento header/PDU riservato o inutilizzato.

²Il clock Skew è un fenomeno nei sistemi sincroni in cui lo stesso segnale di clock di origine arriva a componenti diversi in momenti diversi a causa di diversi motivi.

Gli autori continuando le loro ricerche nel settore si sono accorti che la tassonomia proposta presentava ancora dei limiti in relazione a certi aspetti: doveva incorporare ancora più dettagli, non teneva conto dei canali ibridi, non distingueva tra tecniche basate su uno o più pacchetti, non era adatta alle tecniche più avanzate presentate recentemente e non prendeva in considerazione modelli di modifica del payload, scelta fatta dagli autori fino a quel momento. Spostando la loro attenzione su questi temi nel 2018 hanno pubblicato un ulteriore aggiornamento della tassonomia.

Le principali novità riguardano: come l'approccio originale può essere esteso per includere i processi lato mittente e destinatario che influenzano sia il processo di creazione del modello che la categorizzazione delle tecniche segrete, l'introduzione di nuovi modelli applicabili alla modifica del payload, la descrizione dei canali nascosti distribuiti e come il concetto di modelli possa essere utilizzato per descriverli. Le estensioni proposte in questo documento come detto riguardano i canali nascosti all'interno del payload dei pacchetti e sono 4, tutti contenuti nella Tabella 2.2. Come fatto nella sezione 2.10.3 nella tabella riassuntiva riporteremo solo il tag illustration e context.

PS20. Modulazione della dimensione del campo del carico utile

Illustration: questo modello utilizza la dimensione del payload delle PDU/messaggi di un flusso per codificare il messaggio nascosto. Questo modello è una variante (figlio) del modello P1. Size Modulation.

Literature: PS1. Modulazione dimensionale

Context: Pattern di canali nascosti di rete → Pattern di canali di archiviazione nascosti
→ Modifica del carico utile → Agnostico dei dati utente

PS21. Corruzione dei dati dell'utente

Illustration: questo modello è correlato ai casi in cui i metodi steganografici non tengono conto del tipo di dati utente trasportati all'interno del payload e/o delle sue caratteristiche (modifica cieca). Può essere applicato a singole PDU o a più PDU (un flusso). Ciò accade in genere se parti (o interi) dati utente vengono sostituiti con bit segreti e quindi i dati utente vengono danneggiati/persi. Questo modello è simile al modello PDU Corruption.

Context: Pattern di canali nascosti di rete → Pattern di canali di archiviazione nascosti
→ Modifica del carico utile → Agnostico dei dati utente

PS30. Modifica ridondanza

Illustration: Questo modello viene utilizzato quando è possibile sfruttare la ridondanza dei dati utente trasformandoli in modo tale da ottenere uno spazio libero per i dati segreti (ad esempio mediante transcodifica). Questo modello è un po' simile al modello Aggiungi ridondanza definito in [22] ma può anche diminuire la ridondanza e viene applicato al carico utile anziché ai metadati.

Context: Pattern di canali nascosti di rete → Pattern di canali di archiviazione nascosti → Modifica del carico utile → Consapevole dei dati dell'utente

PS31. Modulazione del valore dei dati utente e Riservato/Non utilizzato

Illustration: le caratteristiche caratteristiche dei dati utente possono essere utilizzate per memorizzare informazioni segrete. Ciò include l'applicazione di metodi come la modifica LSB a campioni vocali o immagini digitali trasportate all'interno del campo del carico utile. Rispetto ai modelli precedenti si tratta di una modifica mirata. Questo modello è analogo alla combinazione dei modelli Value Modulation e Reserved/Unused, ma applicato al carico utile.

Context: Pattern di canali nascosti di rete → Pattern di canali di archiviazione nascosti → Modifica del carico utile → Consapevole dei dati dell'utente

Tabella 2.2: Nuovi modelli proposti nel 2018.

La descrizione completa di tutti i modelli proposti è contenuta nella Tabella 2.3.

Nome del pattern	Descrizione del pattern
PT1. Inter-packets Times	Il canale nascosto altera gli intervalli di tempo tra i messaggi di rete di un flusso (tempi di interarrivo) per codificare i dati nascosti.
PT2. Message Timing	I dati nascosti sono codificati nella tempistica delle sequenze di messaggi all'interno di un flusso, ad esempio riconoscendo ogni n-esimo messaggio ricevuto o inviando comandi m volte.
PT3. Rate/Throughput	Il mittente del canale nascosto altera la velocità di trasmissione dei dati di un flusso da se stesso o da una terza parte al ricevitore nascosto.
PT10. Artificial Loss	Il canale nascosto segnala le informazioni nascoste attraverso la perdita artificiale dei messaggi trasmessi da un flusso, ad esempio tramite frame-corruption o caduta di messaggi.

PT11. Message Ordering	Il canale nascosto codifica i dati utilizzando un ordine sintetico dei messaggi in un flusso.
PT12. Retransmission	Un canale nascosto ritrasmette i messaggi precedentemente inviati o ricevuti di un flusso.
PT13. Frame Collisions	Il mittente provoca collisioni artificiali di frame per segnalare informazioni nascoste.
PT14. Temperature	Il mittente influenza la temperatura hardware di un nodo terzo utilizzando il traffico di un flusso. Deve esistere una tecnica per la ricezione occulta per misurare la temperatura (indirettamente).
PS1. Size Modulation	Il canale nascosto utilizza la dimensione dei metadati del flusso (ad esempio, la dimensione della PDU o di un elemento dell'intestazione) per codificare messaggi nascosti.
PS2. Sequence Modulation	Il canale nascosto altera la sequenza dei metadati di flusso per codificare informazioni nascoste. Questo schema si divide ulteriormente in: P2.a. Posizione e P2.b. Numero di elementi.
PS3. Add Redundancy	Il canale occulto inserisce metadati ridondanti (ad esempio aggiungendo un'opzione IP non utilizzata) in cui i dati sono nascosti in un flusso. Si noti che, rispetto a PS1, i dati sono nascosti nella presenza dei dati ridondanti, non nella dimensione di una PDU o di un elemento dell'intestazione).
PS10. Random Value	Il canale nascosto inserisce i dati nascosti nei metadati del flusso che contengono un valore (pseudo-)casuale.
PS11. Value Modulation	Il canale nascosto seleziona uno degli n valori che l'elemento di metadati di un flusso può contenere per codificare un messaggio nascosto. Questo schema si divide ulteriormente in: P11.a. Pattern di caso e P11.b. Pattern di bit meno significativo (LSB).
PS12. Reserved/Unused	Il canale nascosto codifica i dati nascosti negli elementi di metadati riservati o non utilizzati di un flusso.
PS20. Payload Field Size Modulation	La dimensione del carico utile di un flusso viene utilizzata per codificare le informazioni nascoste (si tratta di un derivato di PS1, ma per il carico utile, poiché comporta la modifica del campo di lunghezza del carico utile di una PDU, ossia PS1).
PS21. User-data Corruption	Il canale di copertura esegue un inserimento (cieco) di dati nascosti nel payload di un flusso (simile a PT10).
PS30. Modify Redundancy	Il canale di copertura comprime il carico utile di un flusso e lo spazio libero risultante viene utilizzato per nascondere i dati.

PS31. User-data Value Modulation and Reserved/Unused	Il canale di copertura esegue una modifica del carico utile di un flusso in un modo che non è riflesso da PS30 e che non risulta in un'interpretazione significativamente modificata dei dati, ad esempio modificando i bit meno significativi delle immagini digitali o nascondendo i dati nei bit inutilizzati/riservati del carico utile.
--	--

Tabella 2.3: Modelli di covert channel proposti in [121].

2.10.5 Proposta del 2022

Steffen Wendzel, dopo aver collaborato con Zander [194], ha proseguito le sue ricerche nell’ambito della sicurezza di rete per approfondirne tutti i vari aspetti. Insieme ad altri esperti del settore, tra cui l’italiano Luca Caviglione, ha creato il progetto “Information Hiding Patterns Project”³ che si dedica esclusivamente alla ricerca e pubblicazione di articoli relativi a questo argomento [195]. Uno degli obiettivi su cui lavorano costantemente è quello di riuscire a pubblicare e fornire alla comunità scientifica una tassonomia completa, basata su modelli, in grado di adattarsi a tutti i domini conosciuti della steganografia, comprese le aree più piccole ed emergenti come la steganografia del filesystem e dei sistemi cyber-fisici. Dopo una prima pubblicazione nel 2021, riguardante principalmente la steganografia di rete, a luglio 2022 hanno proposto la loro tassonomia generale [196].

I principali contributi forniti dal loro studio sono: presentazione della tassonomia completa, descrizione di una terminologia unica per tutti i domini delle steganografia e, per facilitare una descrizione unificata dei metodi di occultamento, presentazione di un tutorial sull’applicazione della tassonomia proposta. Sorvolando i vari tecnicismi, riportiamo che l’aspetto centrale del loro studio è rappresentato dalla suddivisione dei pattern di incorporazione da quelli di rappresentazione: cioè come i dati vengono inseriti nell’oggetto di copertura e come poi vengono rappresentati effettivamente, informazione fondamentale per il SR per poterli estrarre. Sulla base di questa proprietà si afferma che i vari modelli di rappresentazione derivano da quelli di incorporamento, infatti come possiamo osservare in Figura 2.5 la classificazione proposta vale per entrambi .

³<https://patterns.ztt.hs-worms.de>.

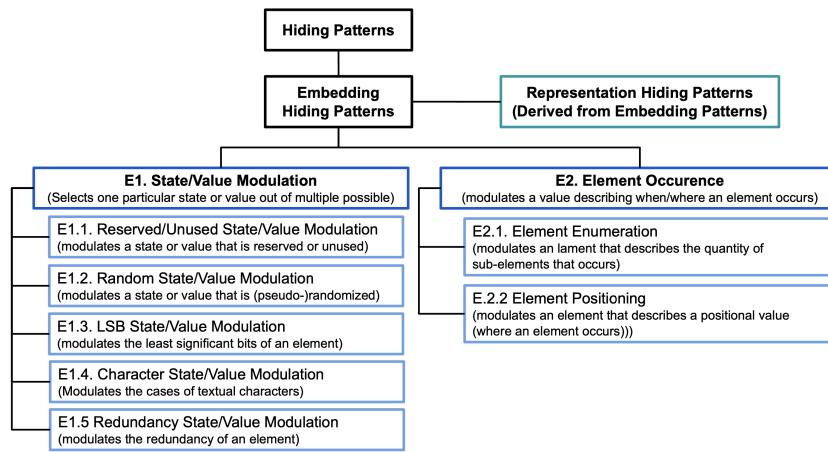


Figura 2.5: Classificazione aggiornata al 2022.

Altro aspetto su cui hanno lavorato è la semplificazione delle relazioni tra i vari termini, basando la descrizione dei vari contesti solo su “Elementi” e i relativi “Stati/Valori”, lasciando la possibilità che gli Elementi possano contenere anche altri sotto-elementi in modo ricorsivo. Questa organizzazione permette di descrivere in maniera precisa e flessibile qualsiasi situazione di utilizzo. Per ogni categoria di modelli viene presentata una sotto-tassonomia per ognuno dei domini della steganografia perché in questo articolo i vari oggetti utilizzati sono di cinque tipi:

- network steganography;
- text steganography;
- digital media steganography;
- cyber-physical system steganography;
- filesystem steganography.

Nella Figura 2.6 riportata di seguito è schematizzata la sotto-tassonomia relativa alla steganografia di rete in quanto è l’argomento principale di questo documento.

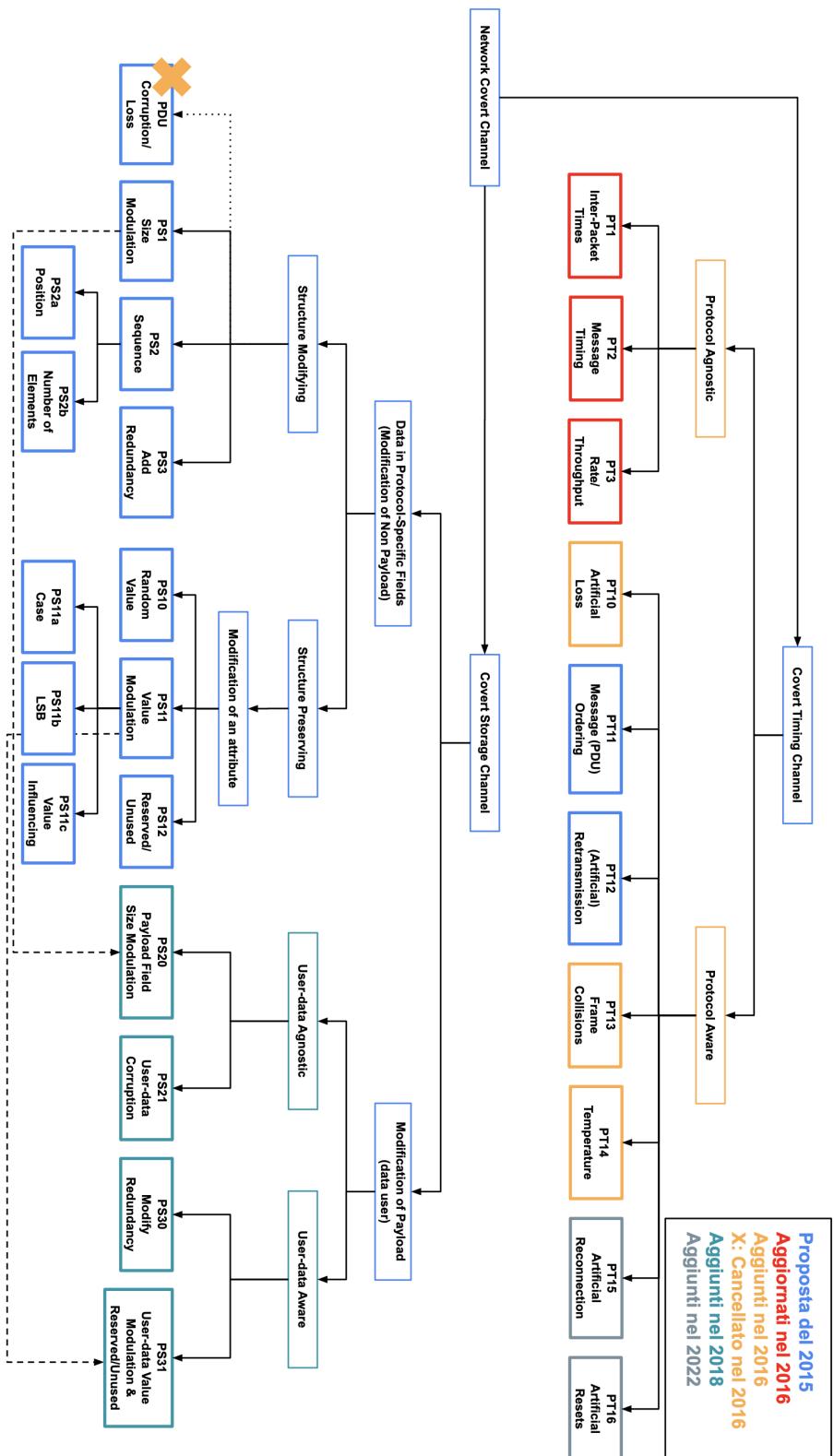


Figura 2.6: Schema di tutti i modelli di network steganography aggiornato alla classificazione del 2022 [196].

2.11 Related work

I canali nascosti sono stati introdotti nel 1973 e nel corso della storia questo settore è andato crescendo in modo esponenziale fino ad oggi. Come più volte ribadito, le proposte presenti in letteratura sono molte e non mancano articoli che in qualche modo cercano di raccoglierle tutte. Il compito però non è di certo semplice visto il susseguirsi di pubblicazioni e le continue scoperte nel settore. Tra queste raccolte, la più completa e recente è senza dubbio quella di Mileva nel 2014 [124] che, suddividendo in base al protocollo della pila TCP/IP utilizzato, riporta tutti i metodi presentati fino a quel momento. Seguendo la strategia usata in quel documento, riporteremo i canali nascosti già presenti nell'articolo integrandoli con quelli proposti in letteratura dopo il 2014. In Figura 2.7 si può osservare una rappresentazione del modello ISO/OSI e a quale livello intervengono i vari protocolli presi in considerazione.

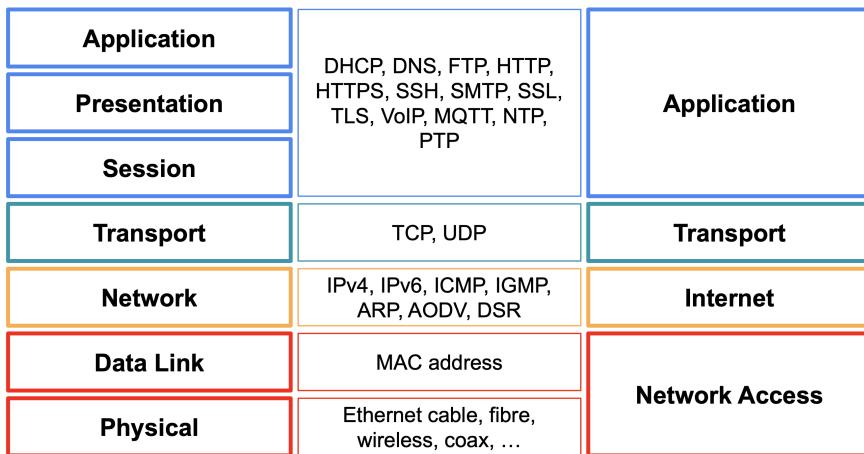


Figura 2.7: Modello ISO/OSI vs TCP/IP.

2.11.1 Livello data link

Il primo livello della pila ISO/OSI coinvolto da tecniche steganografiche è il Data Link. Wolf nel 1989 [199] ha presentato dei metodi per la creazione di canali nascosti all'interno degli standard IEEE 802.2, 3, 4 e 5. Szczypiorski [172] invece ha proposto il sistema steganografico HICCUPS (HIDDEN Communication system for CorrUPted networkS), il quale è rivolto alle reti a mezzo condiviso come le reti locali wireless e può raggiungere una capacità di 216kbit/s. Due nuovi modelli di canali nascosti

vengono proposti anche da Zhao [217] e questi sono dedicati nello specifico allo standard 802.11e. L'idea del 2011 di Gonçalves [65] invece prevede l'iniezione dei dati nel campo Protocol Version dell'intestazione MAC, osservabile in Figura 2.8, attraverso la creazione di frame CTS e ACK su misura. L'implementazione di questa proposta verrà descritta nella Sezione 3.3.1. Un'altra tecnica che coinvolge i protocolli MAC è quella di Li et al [94] la quale codifica le informazioni nelle decisioni di split degli algoritmi. In letteratura sono presenti diverse tecniche che puntano a sfruttare le comunicazioni WLAN per la creazione di canali nascosti e analizzandole si può affermare che usano molte strategie diverse tra loro. La proposta di Frikha et al [56] usa il controllo della sequenza, gli header o entrambi, l'idea di Holloway e Beyah [73] sfrutta il backoff casuale della funzione DCF per evitare le collisioni mentre Zillien e Wendzel [218] usano le riconnessioni. Le WLAN sono state protagoniste anche di una proposta recente: nel 2021 Sawicki et al [155] hanno presentato il metodo StegoFrameOrder (SFO) che garantisce la trasmissione di dati in modo silente utilizzando anche la funzione DCF. Nella Tabella 2.4 vengono riportati tutti i metodi sopra elencati insieme al loro PRBR, il Packet Raw Bit Rate illustrato nella Sezione 2.9.

Articolo	Anno	Metodo	Tipo	PRBR
Wolf [199]	1989	802.5	storage	4b
Wolf [199]	1989	802.5	storage	3b
Wolf [199]	1989	802.3 - length padding modulation	storage	<10b
Wolf [199]	1989	802.3 - padding	storage	0 - 43b
Wolf [199]	1989	802.4 - [63]	storage	~1b
Wolf [199]	1989	802.2	storage	2b
Szczypiorski [172]	2003	HICCUPS	storage	~60b
Li et al [94]	2005	Decision split	storage	$\log_2 \text{subset}/\text{split}$
Frikha et al [56]	2008	Sequence and Header	storage	1-3B
Gonçalves [65]	2011	Protocol Version field (MAC)	storage	6b
Holloway e Beyah [73]	2011	Backoff of DCF	storage	5b
Zhao [217]	2014	QoS Control - 802.11e	storage	8b
Zhao [217]	2014	QoS Control - 802.11e	storage	8b
Zillien e Wendzel [218]	2021	Reconnections	timing	2.5b/s
Zillien e Wendzel [218]	2021	Reconnections	timing	8.5b/s
Sawicki et al [155]	2021	StegoFrameOrder - DCF	timing	9.76b/s

Tabella 2.4: Canali nascosti nel livello Data Link.

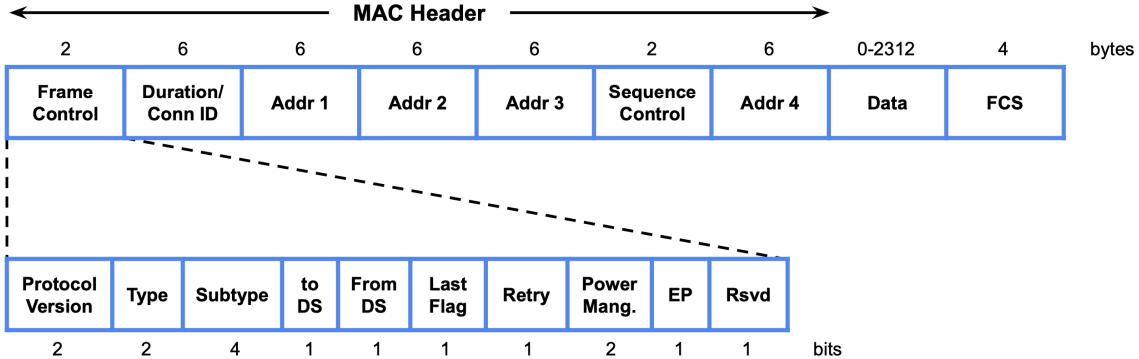


Figura 2.8: Header del protocollo MAC.

2.11.2 Livello rete

La prima tipologia di canali nascosti a livello rete del modello ISO/OSI è quella che interessa il protocollo IP. Questo protocollo è responsabile dell'incapsulamento dei pacchetti ricevuti dal livello Trasporto e della consegna di tali frame da una sorgente a una destinazione. E' disponibile in due versioni, IPv4 e IPv6, e per entrambe sono stati proposti dei metodi di occultamento. Nelle Figure 2.9 e 2.10 è possibile osservare la struttura dei pacchetti delle due versioni del protocollo.

IPv4.

Canali di archiviazione nascosti. Tra le varie proprietà da sfruttare per creare un canale nascosto ci sono i campi di intestazione ridondanti o che normalmente non vengono usati. Una delle prime pubblicazioni è stata fornita da Girling [63] e contiene ben tre proposte: una che sfrutta il campo Address, una la lunghezza del blocco e una il tempo di invio tra pacchetti successivi. Solo la tecnica relativa al campo indirizzo è stata ritenuta utile da dimostrare in una LAN che utilizza CSMA/CD e HDLC (High-Level Data Link Control). Rowland [151] sfrutta il campo ID inserendo il valore ASCII del carattere moltiplicato per 256. Ahsan et al [6, 5] utilizzano i campi relativi alla frammentazione perché, nei casi in cui non si verifica, alcuni valori come MF, DF e Fragment Offset restano pari a 0. Hanno dimostrato di arrivare

ad inviare 17 bit per pacchetto tramite la loro idea. Cauich et al [33] combinando l'utilizzo del campo ID e Fragment Offset propongono un canale con una capacità trasmissiva di 29 bit per frame. Mazurczyk e Szczypliński [115] tramite il processo di frammentazione IP propongono 4 diversi canali nascosti, sfruttando per esempio il numero di frammenti in cui viene diviso il pacchetto originale (0 pari, 1 dispari) oppure modulando i valori inseriti nel campo Fragment Offset. Un altro campo su cui si basano diverse proposte è il campo TTL, tramite il quale Qu et al [146] riescono ad inviare 1 bit per pacchetto. La loro proposta è stata poi migliorata da quella di Zander et al [209] che tiene conto anche del valore iniziale e quello normale che dovrebbe avere il campo.

IPv4 e IPv6 sono dotati di meccanismi per scoprire la Path MTU, cioè la massima unità di trasmissione accettata dal canale di comunicazione. Queste tecniche utilizzano dei pacchetti con il flag DF impostato e ICMP per le risposte. Mazurczyk et al [115, 116] sfruttano questa proprietà dei protocolli per inviare dati segreti, inoltre spiegano perché convenga l'utilizzano di RSTEG [118]. Abad [1] ha dimostrato che tramite un difetto del campo checksum si possono inviare dati nascosti, questi potranno essere recuperati dal destinatario con il valore originale TTL del pacchetto. Nel 2016 Sharma et al [164] hanno studiato un metodo per aumentare la capacità di dati trasmessi con un singolo pacchetto IP attraverso un canale segreto che sfrutta il numero di identificazione ID. Bedi et al [19], tramite il campo Overflow dell'opzione Timestamp del protocollo IP, creano un canale nascosto difficile da rilevare perché utilizzano dei valori legittimi per quel campo.

Trabelsi et al [181, 182] propongono un metodo che, tramite il comando Traceroute e le opzioni contenute nel campo Record dell'intestazione IP, riesce a raggiungere un PRBR di 320. Sempre Trabelsi, ma questa volta insieme a Jawhar [180], ha sviluppato un ulteriore canale per il trasferimento di dati segreti basato sul campo Record IP. Dabbagh [38] implementa una botnet e coordina un attacco DoS tramite i campi IP ID e TTL verificando se Snort sia in grado di rilevare le azioni malevoli. Abbas et al [2] si servono del campo di identificazione IP per implementare la loro idea, caratterizzata dall'algoritmo MPSO che sceglie i pacchetti migliori all'interno dei quali nascondere i dati. Tommasi et al [179] nel 2022 hanno presentato uno dei più recenti metodi di network steganography, chiamato COTIIP, basato su pacchetti IP incompleti. Il destinatario dalle informazioni sui pacchetti incompleti o mancanti

ripristina il messaggio segreto.

Un autore che ha contribuito in maniera importante alla letteratura in questo settore è Zander e nella sua pubblicazione [209] descrive ben 4 canali nascosti. Il primo è un miglioramento di un precedente canale di archiviazione nel campo TTL dei pacchetti IP mentre gli altri 3 sono nuovi, uno di questi è basato sul gap di invio tra pacchetti. Tra le tante pubblicazioni di Mazurczyk, quella del 2019 [123] propone un nuovo canale segreto reversibile basato sui flussi IPv4, una sua implementazione prototipo e la sua valutazione, dove riporta i risultati della reversibilità ma non del PRBR.

Riportiamo in questa sezione anche lo studio del 2008 di Trabelsi et al [183] con il quale hanno sviluppato tramite un canale nascosto che incorpora le informazioni nell'intestazione IP, una versione nascosta del protocollo FTP, Hidden File Transfer Protocol (HFTP), il quale offre anche un servizio di messaggistica breve nascosto (SMS). In Tabella 2.5 vengono riportati tutti i metodi citati e il loro PRBR.

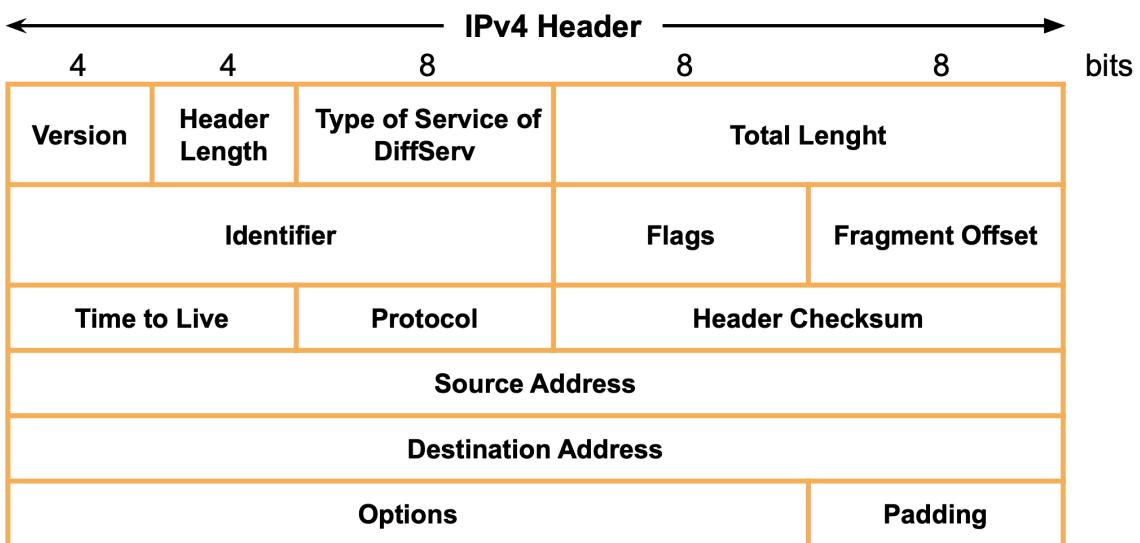


Figura 2.9: Header del protocollo IPv4.

Articolo	Anno	Metodo	PRBR
Girling [63]	1987	Address field	~1b
Rowland [151]	1997	ID field	16b
Abad [1]	2001	Header checksum	16b

Ahsan et al [5]	2002	DF field	1b
Ahsan [6]	2002	DF and ID fields	17b
Ahsan [6]	2002	Version, IHL, ID fields	8b
Qu et al [146]	2004	TTL	1b
Catich et al [33]	2005	ID and Fragment Offset fields	29b
Zander et al [206]	2006	TTL	1b
Trabelsi et al [181, 182]	2007	Record route options	320b
Mazurczyk et al [115]	2009	Predefined number of fragments	1b
Mazurczyk et al [115]	2009	Modulating values Fragment Offset	1b
Mazurczyk et al [115]	2009	Steganogram in fragment payload	1b
Mazurczyk et al [115]	2009	Probe messages in PMTUD	<1500b
Trabelsi and Jawhar [180]	2010	Covert FTP	34B
Zander [209]	2010	TTL	0.9b
Zander [209]	2010	Multiplayer	<15bps ⁴
Sharma et al [164]	2016	TCP-IP packet header	4B
Abbas et al [2]	2018	IP ID	8b
Dabbagh [38]	2018	TTL modulation	1b
Dabbagh [38]	2018	IP ID	2 ASCII (16b)
Dabbagh [38]	2018	Urgent Pointer	2 ASCII(16b)
Mazurczyk et al [123]	2019	Reversible	-
Bedi et al [19]	2020	Overflow field of Timestamp option IP	20b
Tommasi et al [179]	2022	Incomplete IP packets	1 ASCII(8b)

Tabella 2.5: Canali di memorizzazione nel protocollo IPv4.

Canali di temporizzazione nascosti. Padlipsky et al [140] hanno suggerito l’idea di un canale nascosto di temporizzazione dove il bit segreto è rappresentato dal fatto che il mittente invii o no il pacchetto entro un intervallo di tempo. La capacità trasmissiva sarà pari a N bps, dove N è il numero di intervalli in cui è diviso un secondo. Cabuk et al [28] hanno ripreso l’idea presentando anche un prototipo funzionante del canale nascosto. Berk et al [21] sfruttano i tempi di interarrivo tra pacchetti per codificare delle informazioni. Tutti i ritardi vengono memorizzati e viene calcolata una media, se il ritardo del pacchetto ricevuto è superiore alla media sarà considerato

⁴Non è possibile ricavare il valore PRBR in quanto il metodo si basa sui spostamenti effettuati dal giocatore.

1 altrimenti 0. Per inserire questi ritardi è possibile utilizzare un Keyboard Jitterbug [163], un dispositivo che si posiziona tra la tastiera e il computer. Cabuk [27] ha presentato una versione migliorata di questo canale di temporizzazione utilizzando dei modelli di ritardo reali, li ha divisi in due gruppi per rappresentare i valori binari 1 e 0.

Servetto et al [162] creano un canale segreto saltando intenzionalmente dei numeri di sequenza nell'invio di un flusso di dati. Se si verifica una perdita in un certo intervallo di tempo questa rappresenta il bit 1 dei dati segreti, 0 altrimenti. Una strategia simile è stata usata in [115, 116], ma in questo caso si creano pacchetti fantasma sfruttando il campo Fragment Offset.

Ahsan e Kundur [5, 6] hanno dimostrato come, attraverso il riordino dei frame, sia possibile trasferire $\log_2 n!$ bit, con n pacchetti, se la rete è affidabile. La loro tecnica di traffic obfuscation si basa sul campo Sequence Number dell'header di autenticazione e sui campi Option Data Length e Option Data di IPsec. Ordinamento che è anche al centro della proposta di El-Atawy et al [48], infatti tramite il riordino dei pacchetti hanno sviluppato un metodo robusto e resistente a eventuali riordini esterni, dotato anche di un sistema di rilevamento/correzione degli errori. Metodo ripreso e migliorato in [49]. Galatenko et al [57], riordinando invece i pacchetti in base all'indirizzo di destinazione, hanno sviluppato e proposto un canale segreto statistico attraverso un server Proxy.

Gianvecchio et al [60], Sellke [160] e Zander et al [208] hanno presentato altri tre canali di temporizzazione. I primi due modellano e imitano le proprietà statistiche del ritardo tra pacchetti del traffico normale perciò risultano essere poco rilevabili, mentre il terzo presenta una migliore sincronizzazione ma una minore robustezza contro il jitter di rete. Allix [11] descrive un esempio di un CTC che vede il coinvolgimento di due macchine A e B connesse allo stesso server C. Se prima invia due pacchetti A e poi uno B può essere considerato come 1 binario, mentre il contrario come 0. Nella Tabella 2.6 sono elencate tutte le proposte di canali temporizzati appena citate.

Articolo	Anno	Metodo
Padlipsky et al [140]	1978	Send or not IP packet in interval
Servetto et al [162]	2001	Phantom packets
Ahsan et al [5], Ahsan [6]	2002	Packet sorting

Galatenko et al [57]	2005	Packet sorting with ordered dest. address
Shah et al [163]	2006	IPDs
Cabuk [27]	2006	IPDs
Allix [11]	2007	Sending more or less than x
Gianvecchio et al [60]	2008	IPDs
Sellke et al [160]	2009	IPDs
Mazurczyk et al [115]	2009	Different rates for fragments
Mazurczyk et al [115]	2009	Phantom fragments
Mazurczyk et al [115]	2009	Fragment sorting
El-Atawy et al [48]	2009	Packet sorting
Zander [209]	2010	IPDs
Zander [209]	2010	Temperature - Indirect
Zander et al [208]	2011	IPDs

Tabella 2.6: Canali di temporizzazione nel protocollo IPv4.

IPv6. Anche per il protocollo IPv6, nonostante ancora non abbia sostituito definitivamente la vecchia versione, sono state pubblicate diverse tecniche. La prima proposta è quella di Graf del 2003 [66] che suggerisce di utilizzare l'intestazione Destination Options per nascondere il messaggio. Dopo due anni Lucena et al [102] hanno pubblicato uno studio molto approfondito sulle tecniche di occultamento in IPv6, evidenziando addirittura più di 10 canali nascosti di archiviazione. Resta ad oggi la più importante pubblicazione di network steganography su IPv6. Per quasi tutte le intestazioni del protocollo, osservabili in Figura 2.10, hanno presentato un metodo per l'iniezione di dati segreti arrivando addirittura a proporre metodi con capacità superiori a 1MB, grazie anche alla lunghezza degli indirizzi. Hanno anche sottolineato il fatto che alcuni metodi potrebbero essere rilevati abbastanza facilmente perché il mittente deve calcolare l'ICV (Integrity Check Value) dopo aver inserito i dati e il destinatario deve intercettare i pacchetti prima che raggiungano la destinazione. Bedi e Dua [20] propongono una tecnica di steganografia di rete che utilizza i pacchetti IPv6 con 0 o più (fino a 4) intestazioni di estensione per nascondere i dati segreti. Può trasferire 5 o più bit di dati nascosti per pacchetto. Nel 2019 diversi canali proposti per IPv6 sono stati esaminati da Mazurczyk et al [122] e di 6 di que-

sti ne sono state valutate le prestazioni. Tutti le proposte e le loro prestazioni sono riportate in Tabella 2.7.

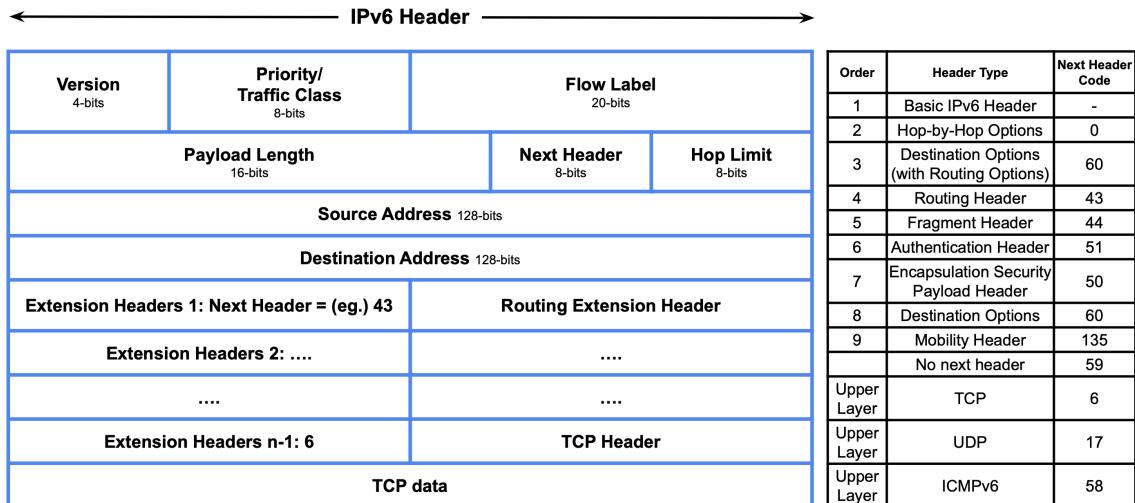


Figura 2.10: Header del protocollo IPv6.

Articolo	Anno	Metodo	PRBR
Graf [66]	2003	Destionation Options Header	max field length
Lucena et al [102]	2005	traffic class	8b
Lucena et al [102]	2005	Flow Label	20b
Lucena et al [102]	2005	Source Address	128b
Lucena et al [102]	2005	Hop Limit	-
Lucena et al [102]	2005	Payload Lenght	$(2^{16}-L_P)B$
Lucena et al [102]	2005	Hop-by-Hop options header	up to 2038B
Lucena et al [102]	2005	Reserved in Routing header	4B
Lucena et al [102]	2005	address in routing header	up to 2048B
Lucena et al [102]	2005	Destination options header	up to 2038B
Lucena et al [102]	2005	False padding in the Destination options header	up to 256B
Lucena et al [102]	2005	Fragment Header	8+2b
Lucena et al [102]	2005	Next Header field	8b
Lucena et al [102]	2005	Fake fragment	up to 64KB
Lucena et al [102]	2005	Reserved in Authentication header	2B
Lucena et al [102]	2005	Fake Authentication header	up to 1022B

Lucena et al [102]	2005	Fake ESP header	up to 1022B
Lucena et al [102]	2005	False padding value in ESP header	up to 255B
Mazurczyk et al [115, 116]	2009	PLMTUD using RSTEG	up to MTU
Mazurczyk et al [122]	2019	Traffic Class	2b
Mazurczyk et al [122]	2019	Flow Label	2b
Mazurczyk et al [122]	2019	Payload Length	up to MTU
Mazurczyk et al [122]	2019	Next Header	size of the fake headers injected
Mazurczyk et al [122]	2019	Hop Limit	1b
Mazurczyk et al [122]	2019	Source Address	128b
Bedi and Dua [20]	2020	Extension header	>5b

Tabella 2.7: Canali di archiviazione nel protocollo IPv6.

ICMP. Al livello internet il protocollo che si occupa della trasmissione di messaggi di errori (e altre informazioni) è ICMP, la struttura del suo pacchetto è riportata in Figura 2.11 e anche questo è stato al centro di diverse proposte. Il progetto Loki [39, 40] dimostra come sia possibile e facile implementare un canale nascosto tra un client e un server sfruttando i messaggi Echo Request e Echo Reply. Ci sono altri metodi di tunnel IP-over-ICMP e tra questi troviamo ICMPTX⁵, Skeeve [210] e ICMP-chat [129]. Le loro performance dipendono dal sistema operativo e dal contesto in cui operano ma possono raggiungere un PRBR superiore a 56B. Riportiamo che questi canali sono stati esaminati ulteriormente in [170] per testare la loro resistenza ai sistemi di monitoraggio moderni. Stødle⁶ ha sviluppato un sistema basato sul comando ping per creare un canale affidabile per le connessioni TCP sfruttando i pacchetti ICMP Echo Request e Echo Reply. Tra le ultime proposte registrate troviamo quella di Murphy [133] che introduce lo strumento V00d00n3t, basato su IPv6 e ICMPv6, e quella di Ray e Mishra [148] che dimostra di trasmettere grandi quantità di dati in modo nascosto attraverso i pacchetti ICMP Echo Request. Infine Lu et al

⁵ICMPTX, John Plaxco, 1998, https://codeberg.org/jakkarth/icmptx/src/branch/main/tun_dev.c.

⁶ptunnel, Stødle, 2003, <https://github.com/f1vefour/ptunnel>.

[100] nel 2022 sfruttano Echo Reply all'interno dell'ambiente Linux per codificare le informazioni. La Tabella 2.8 riporta le varie proposte e le loro potenziali prestazioni.

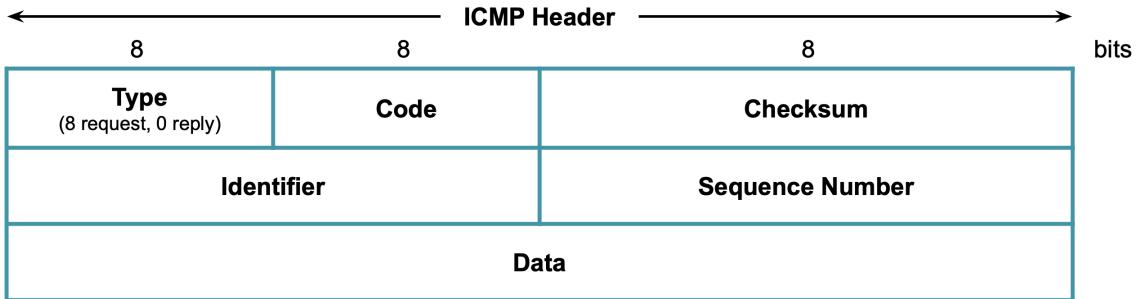


Figura 2.11: Header del protocollo ICMP.

Articolo	Anno	Metodo	PRBR
Loki [39, 40]	1996	Payload Echo Request/Reply	up to 24B, 56B or more
Ahsan [5]	2002	Reserved field	32
ICMP-chat [129]	2003	Payload Echo Request/Reply	up to 24B, 56B or more
Skeeve [210]	2004	Payload Echo Request/Reply	up to 24B, 56B or more
ICMPTX	2005	Payload Echo Request/Reply	up to 24B, 56B or more
Ping tunnel	2005	Payload Echo Request/Reply	up to 24B, 56B or more
V00d00n3t [133]	2006	Payload Echo Request/Reply	-
Ray e Mishra [148]	2008	ID	1B
Lu et al [100]	2022	Echo Reply	2B

Tabella 2.8: Canali di archiviazione nel protocollo ICMP.

IGMP. Il protocollo IGMP (Internet Group Management Protocol), il cui pacchetto-tipo può essere osservato nella Figura 2.12, stabilisce l'appartenenza a gruppi multi-cast su reti IPv4. Dopo Ahsan [5], dove nella sua tesi ha presentato il primo canale segreto che sfrutta questo protocollo, ne sono stati pubblicati altri due: quello di

Scott [158] basato sul campo Reserved della versione 3 e quello sulla versione 4 la cui implementazione prende il nome di B0CK [186]. In Tabella 2.9 l'elenco dei tre metodi e delle loro prestazioni.

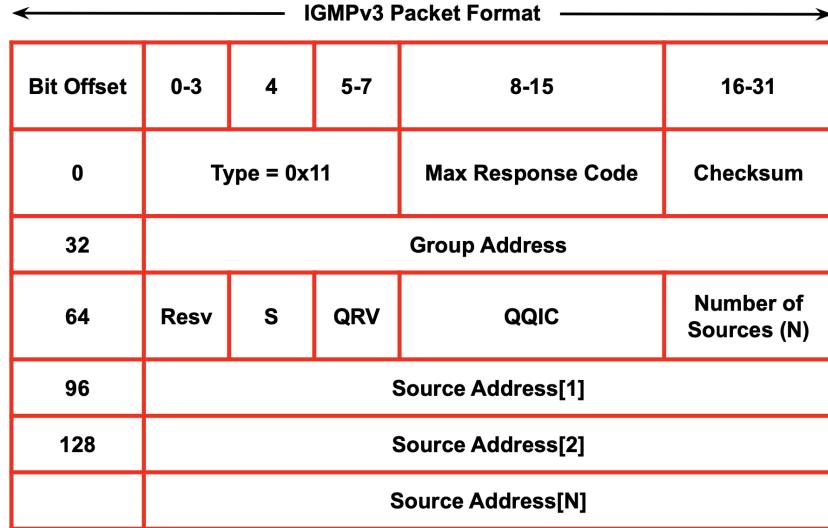


Figura 2.12: Header del protocollo IGMPv3.

Articolo	Anno	Metodo	PRBR
B0CK [186]	2000	IGMP header	up to 8B
Ahsan [5]	2002	IGMP header	8-16b
Scott [158]	2008	IGMP header	8-16b

Tabella 2.9: Canali nascosti nel protocollo IGMP.

ARP. In letteratura si registrano anche proposte che sfruttano il protocollo ARP (Address Resolution Protocol). Nel 2010 è stata pubblicata la tecnica di Ji et al [82] mentre nel 2021 quella di Dua et al [44]. Quest'ultima per creare una comunicazione segreta utilizza i messaggi di richiesta di trasmissione ARP di una LAN. La tecnica usa un valore seed comune già noto sia al mittente che al destinatario per codificare i dati segreti e riesce a inviare 7 bit per pacchetto. In Figura 2.13 è rappresentata la

struttura del pacchetto mentre nella Tabella 2.10 sono elencati i due metodi e il loro PRBR.

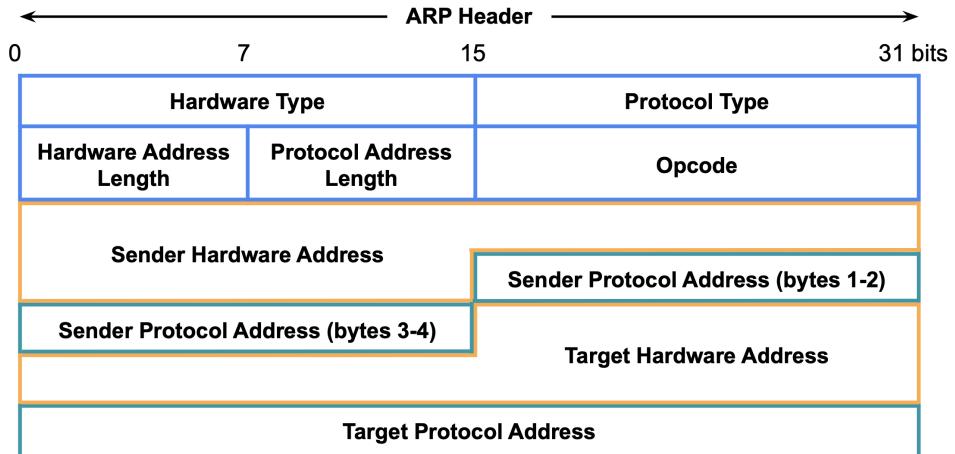


Figura 2.13: Header del protocollo ARP.

Articolo	Anno	Metodo	PRBR
Ji et al [82]	2010	Target IP	7b
Dua et al [44]	2021	ARP Broadcast Request packet	7b

Tabella 2.10: Canali nascosti nel protocollo ARP.

DSR e AODV. Il protocollo Dynamic Source Routing (DSR) è un protocollo di routing per reti MANET che è stato sviluppato per ridurre il consumo di banda. Marone [109] nella sua pubblicazione include diversi meccanismi al fine di sfruttare questo protocollo per trasmettere dati in modo silente. Simile al protocollo precedente c'è Ad-hoc On-demand Distance Vector (AODV) ed anche questo è al centro delle 4 proposte di canali segreti effettuate da Li e Ephremides [95]. In Tabella 2.11 sono riportate anche le prestazioni potenziali delle due proposte.

Articolo	Anno	Metodo	PRBR
Marone [109]	2003	Header DSR Request	<5b
Li and Ephremides [95]	2004	AODV	up to <1b

Tabella 2.11: Canali nascosti nei protocolli DSR e AODV.

BAS. I BAS (Building Automation System) sono dei sistemi di automazione di edifici, possono coinvolgere per esempio riscaldamento, ventilazione e aria condizionata. Wendzel et al [193] hanno scoperto 3 canali segreti sfruttando il protocollo BACnet: i primi due sono di archiviazione e sono basati sul tipo del messaggio e sui parametri (Don't Fragment) mentre l'ultimo è di temporizzazione. In Tabella 2.12 sono riportate le prestazioni dei primi due canali in quanto nel terzo il valore è variabile.

Articolo	Anno	Metodo	PRBR
Wendzel et al [193]	2012	Type message	$\log_2 n$ b
Wendzel et al [193]	2012	Parameters	IP-TCP ⁷
Wendzel et al [193]	2012	Timing	-

Tabella 2.12: Canali nascosti nei protocolli BAS.

Canali protocollari. Nel 2008 Wendzel et al [191] hanno proposto una categoria di canali nascosti da loro denominata “canali protocollari”, cioè invece di inviare dei pacchetti modificati si concordano due protocolli, quando viene spedito un pacchetto del primo rappresenterà 1 binario e uno del secondo protocollo sarà lo 0. Il vantaggio è rappresentato dal fatto che si possono inviare pacchetti autentici che non risultano anomali ai sistemi di sicurezza. Come riportato in Tabella 2.13 ogni pacchetto rappresenta/trasporta un solo bit.

⁷Vengono usati i parametri dei protocolli IP e TCP, quindi fare riferimento alle rispettive Tabelle 2.5 e 2.14.

Articolo	Anno	Metodo	PRBR
Wendzel et al [191]	2008	Type protocol	1b

Tabella 2.13: Canali protocollari.

2.11.3 Livello trasporto

Nel modello ISO/OSI, subito sopra il livello rete troviamo il livello trasporto che è caratterizzato principalmente dai protocolli TCP e UDP. Il primo punta a una maggiore affidabilità e offre una connessione tra i due host, mentre il secondo a discapito di perdite di pacchetti garantisce prestazioni maggiori. La maggior parte delle proposte si basa su TCP. Le strutture delle due tipologie di pacchetti sono riportate rispettivamente in Figura 2.14 e Figura 2.15.

TCP. Rowland [151] implementa il suo canale segreto (*Covert_TCP*) sfruttando i campi ISN e Acknowledge Sequence Number dell'intestazione IP, tecnica ripresa e aggiornata da Rutkowska [152] andando a sfruttare il traffico esistente senza crearne di nuovo. Anche la proposta di Murdoch e Lewis si basa sul campo ISN ma è sviluppato per OpenBSD e Linux [132]. Ahsan attraverso le 29 combinazioni valide di 6 bit di flag ha proposto diversi canali di memorizzazione nascosti. Nel caso in cui il bit URG non fosse impostato è possibile creare un canale segreto [5, 6] con una capacità di 16 bit per pacchetto. Allix [11] invece usa il campo Reserved per inviare 4 bit di dati segreti in ogni pacchetto.

Giffin et al [62] modificando il bit meno significativo del campo Timestamp del TCP invia un bit per pacchetto, la loro tecnica risulta essere molto efficace e difficile da individuare su reti di computer lente. Chakinala et al [34] riprendendo lo schema proposto in [5, 6] creano un nuovo canale segreto temporizzato riordinando i segmenti TCP e utilizzando il campo Numero di Sequenza (ISN). Luo et al [106, 107] hanno implementato due canali di memorizzazione nascosti, *Covert Clack* e *ACKLeaks*, modificando il campo Acknowledge Sequence Number e inserendo i dati nascosti nei pacchetti puri TCP ACK. *ACKLeaks* può sfruttare le connessioni già esistenti e risulta essere resistente contro il rilevamento basato sui contenuti.

Cloak [104] definisce una nuova classe di canali temporizzati che inviano i dati segreti attraverso una distribuzione unica di N pacchetti su X connessioni, vengono anche definiti 10 metodi di codifica/decodifica. TCP-Script [105] è un metodo di canale temporizzato che usa i burst di dati del TCP. Ne viene presentata anche un'implementazione che risulta essere robusta al jitter di rete, alla perdita e al rior-dino dei pacchetti, ma è caratterizzata da una bassa capacità ed è facile distinguere il suo flusso da quello legittimo. Mazurczyk et al [118, 119] applicano il metodo RSTEG al protocollo TCP tramite i meccanismi di ritrasmissione RTO (Retransmis-sion Timeout), FR/R (Fast/Retransmit) e SACK (Selective ACK). L'idea su cui si basa questa strategia è quella di non riconoscere intenzionalmente un pacchetto TCP e forzarne la ritrasmissione, il segmento ritrasmesso conterrà dei dati steganografati al posto del payload. Mohamed Azran [127] nella sua tesi descrive la creazione di un canale nascosto inserito nel cookie TFO del protocollo TCP, i dati da trasmettere vengono rappresentati attraverso la modulazione del valore del campo.

Il mondo dell'industria è entrato nell'era 4.0 grazie all'avvento dei sistemi au-tomatizzati che comunicano attraverso protocolli, come Modbus, che si basano su TCP. Alcaraz et al [9] testano due strategie per effettuare un attacco nell'industria 4.0 e Lamshöft et al [90] dopo aver discusso di 18 modelli noti applicabili alle reti ICS (Industrial Control System), ne implementano due e ne analizzano le prestazio-ni. Sempre Lamshöft [91], questa volta dopo aver analizzato canali nascosti noti in TCP e DNS, ha pubblicato la sua strategia di nascondere informazioni attraverso le scansioni di porte TCP. Hussain et al [80] hanno progettato e testato una stra-tegia, tramite il simulatore ns2⁸, risultata avere una capacità più alta rispetto ai modelli precedentemente proposti. Un riassunto delle proposte citate lo troviamo in Tabella 2.14.

⁸<https://www.nsnam.org>.

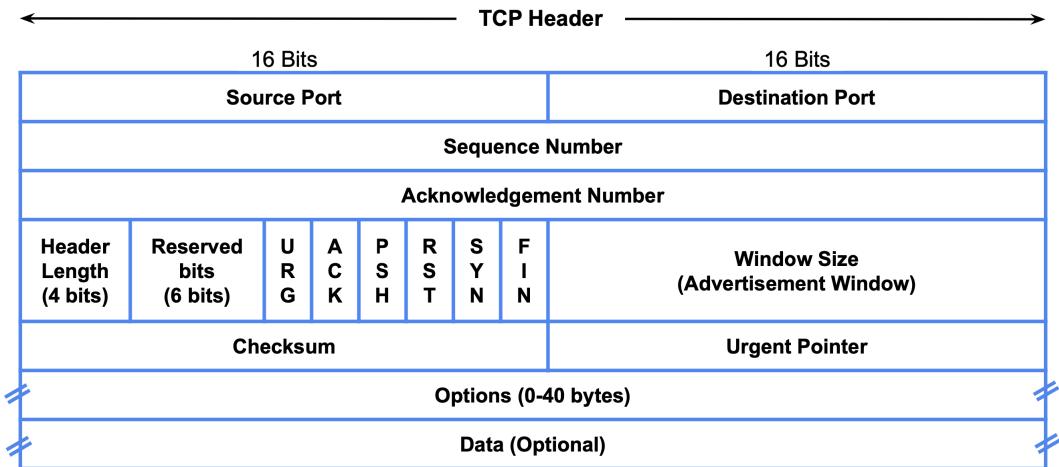


Figura 2.14: Header del protocollo TCP.

Articolo	Anno	Metodo	Tipo	PRBR
Covert_TCP [191]	1997	ISN and ACK	storage	64b
Abad [1]	2001	Header checksum	storage	16b
Ahsan [5] Hintz [71]	2002	Urgent Pointer	storage	16b
Giffin et al [62]	2002	TCP timestamps	timing	1b
NUSHU [152]	2004	ISN	storage	32b
Lantra et al [132]	2005	ISN	storage	32b
Chakinala et al [34]	2006	Segment reordering	timing	$\log_2 n! b$
Allix [11]	2007	Reserved	storage	4b
Cloak [104]	2007	X TCP Flows	timing	-
TCP-SCript [105]	2008	TCP burst	timing	-
CLACK [106]	2009	ACK Sequence Number	storage	32b
RSTEG [118, 119]	2010	retransmission	storage	max IP payload length
ACKLeaks [107]	2011	TCP ACK packets	storage	1 or more bit
Hussain et al [80]	2011	payload and retransmission	storage	84b
Azran [127]	2019	TFO TCP	storage	64b
Alcaraz [9]	2019	Modbus-TCP	storage	1b
Lamshöft et al [90]	2020	Modbus-TCP IPDs	timing	1b
Lamshöft et al [90]	2020	Modbus-TCP Artificial loss	timing	1b
Lamshöft et al [90]	2020	Modbus-TCP Reserved- unused	storage	1b

Lamshöft et al [90]	2020	Modbus-TCP User-data storage modulation and Rserverd- unused	7b
Lamshöft et al [91]	2022	port scans storage	768b

Tabella 2.14: Canali nascosti nel protocollo TCP.

UDP. Anche nel protocollo UDP (User Datagram Protocol) tra le intestazioni troviamo il campo Checksum. Questo campo è osservabile in Figura 2.15 insieme agli altri che compongono la struttura del pacchetto UDP, a differenza di altri però il checksum è opzionale. La sua presenza o assenza può essere considerata come 1 o 0 in binario, e su questa idea si basa la proposta di Fisk et al [53]. Questo tipo di canale avrà un PRBR pari a 1. Il metodo di Abad [1] può essere applicato anche al campo Checksum del protocollo UDP oltre a quello del TCP. Thyer et al [177], attraverso la loro pubblicazione, hanno dimostrato che è possibile inviare fino a 6B in un pacchetto UDP attraverso i campi: indirizzo sorgente, lunghezza e checksum. Come negli altri paragrafi la Tabella 2.15 riassume le caratteristiche delle tecniche citate.

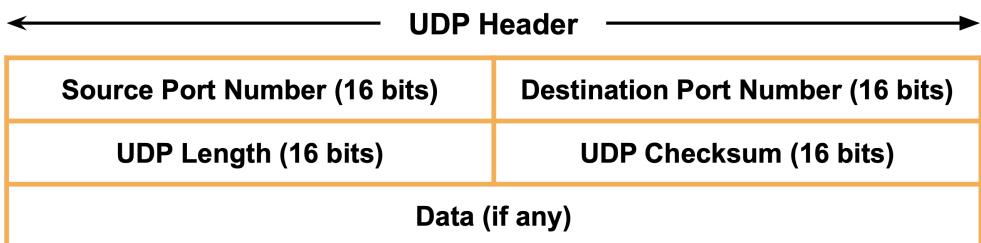


Figura 2.15: Header del protocollo UDP.

Articolo	Anno	Metodo	Tipo	PRBR
Abad [1]	2001	Header checksum	storage	16b
Fisk et al [53]	2002	Presence/absence of checksum	storage	1b
Thyer [177]	2008	Source addr., Length, Checksum	storage	up to 6B

Tabella 2.15: Canali nascosti nel protocollo UDP.

2.11.4 Livello applicazione

Anche i protocolli a livello applicazione, il più in alto nella pila ISO/OSI, sono stati utilizzati per proporre molti canali nascosti. I protocolli che troviamo a questo livello possono seguire un modello Client-Server oppure uno Peer-to-Peer in cui gli utenti si scambiano le informazioni tra loro in modo cooperativo.

HTTP. L’Hypertext Transfer Protocol è un protocollo a livello applicativo usato come principale sistema per la trasmissione d’informazioni sul web, la struttura di due pacchetti di richiesta e risposta può essere visualizzata in Figura 2.16. Quasi tutte le organizzazioni permettono la navigazione su internet tramite esso, nonostante sia disponibile la sua versione basata su TLS (HTTPS) che garantisce un maggiore sicurezza. Dyatlov et al [46] hanno presentato dei canali di memorizzazione che sfruttano l’intestazione e/o il corpo della richiesta/risposta HTTP. Non è possibile fornire una valutazione precisa delle prestazioni di queste tecniche perché la dimensione delle intestazioni accettate varia in base alla versione del server web.

Lo strumento Reverse WWW Shell [68] dimostra come sia possibile inviare comandi e output degli stessi, in modo nascosto, tramite HTTP. Invece Bowyer [23] utilizza questi canali nascosti per comunicare con i trojan dietro i firewall, codificando i messaggi nei parametri URL o dopo le richieste GET. Bauer [18] suggerisce come creare una rete anonima di overlay sfruttando le comunicazioni regolari degli utenti sul web, attraverso reindirizzamenti, cookie, intestazioni, elementi HTML e contenuti attivi.

Kwecka [88] inserisce i dati segreti nelle intestazioni HTTP sfruttando il fatto che il protocollo tratta le varie quantità di spazio bianco come carattere singolo. Per esempio la tabulazione può essere interpretata come 1 binario uno spazio normale come lo 0. Essendo le intestazioni insensibili alle maiuscole o minuscole si può anche utilizzare la diversa capitalizzazione delle lettere per trasferire dati segreti. Una tecnica invece che si basa sulla lunghezza dei pacchetti HTTP è stata ideata da Ji et al [81] nel 2009. Le prestazioni registrate sono pari a 50 bytes: 20 bytes TCP Header, 20 bytes IP Header e 18 bytes Ethernet Header. Alman [12] descrive come sia pos-

sibile, sfruttando una debolezza nel metodo CONNECT, instaurare una connessione attraverso un server proxy. Van Horenbeeck [185], con il suo strumento Wondjina⁹, è in grado di permettere a un client di controllare se la copia che ha in cache è ancora valida, il tutto utilizzando i tag HTTP Entity. Idee simili sono state utilizzate in [45] insieme ad approcci LSB sui campi Data e Ultima Modifica.

Tramite il ritardo (1) o meno (0) della risposte del server web, Eßer e Freiling [50] creano un canale di temporizzazione nascosto, mentre Castro [32] suggerisce di utilizzare i cookie per creare canali nascosti. Citiamo in questo paragrafo anche il framework Infranet [51] che usa canali nascosti per aggirare la censura, rispondendo con le informazioni richieste steganografate in immagini innocue. Tra le idee più recenti troviamo quella di Castiglione et al [31] che suggeriscono un nuovo protocollo steganografico che usa i messaggi di “controllo” HTTP, applicabile ai dispositivi che comunicano in un ambiente MEC (Mobile Edge Computing). Nel 2018 Yao Shen et al [166] hanno ideato due canali nascosti basati sul comportamento (LiHB, HBCC), cioè incorporano le informazioni nelle diverse distribuzioni del flusso, senza modificare il contenuto del pacchetto. In Tabella 2.16 sono riportati tutti i metodi citati.

⁹<https://github.com/maartenvhb/wondjina>

HTTP/1.1 200 OK	
Access-Control-Allow-Origin: *	Response Headers
Connection: Keep-Alive	Representation Headers
Content-Encoding: gzip	Generals Headers
Content-Type: text/html; charset=utf-8	
Date: Wed, 10 Aug 2016 13:17:18 GMT	
Etag: "d9b3b803e9a0dc6f22e2f20a3e90f69c41f6b71b"	
Keep-Alive: timeout=5, max=999	
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT	
Server: Apache	
Set-Cookie: csrtoken=...	
Transfer-Encoding: chunked	
Vary: Cookie, Accept-Encoding	
X-Frame-Options: DENY	
(body)	
POST / HTTP/1.1	
Host: localhost:8000	Response Headers
User-Agent: Mozilla/5.0 (Macintosh;...);... Firefox/51.0	Representation Headers
Accept: text/html, application/xhtml+xml,...,*/*;q=0.8	Generals Headers
Accept-Language: en-US,en;q=0.5	
Accept-Encoding: gzip, deflate	
Connection: Keep-Alive	
Upgrade-Insecure-Requests: 1	
Content-Type: multipart/form-data; boundary=-12656974	
Content-Length: 345	
-12656974	
(more data)	

Figura 2.16: Header del protocollo HTTP

Articolo	Anno	Metodo	Tipo	PRBR
Reverse WWW Shell [68]	1999	HTTP request/response	storage	-
Corkscrew ¹⁰	2001	HTTP request/response	storage	-
Bowyer [23]	2002	URL parameters / body of GET request	storage	-
Dyatlov et al [46]	2003	HTTP request/response	storage	-
Muted Posthorn [18]	2003	Redirection	storage	up to 1024B
Muted Posthorn [18]	2003	Refer	storage	up to 1024B
Muted Posthorn [18]	2003	Set-Cookie	storage	up to 4096B
Muted Posthorn [18]	2003	HTML elements and Acrive content	storage	-
Alman [12]	2003	CONNECT method	storage	-
Eßer et al [50]	2005	Delaying or not response	timing	1b

¹⁰Pat Padgett, corkscrew, 2003, <https://github.com/patpadgett/corkscrew>

HTTunnel ¹¹	2005	-	storage	-
Kwecka [88]	2006	consequent linear white space characters	storage	up to 8190B
Wondjina [185]	2006	Entity tags	storage	-
Wondjina [185]	2006	Content MD5 header	storage	128b
Castro [32]	2006	cookies	storage	-
Ji et al [81]	2009	Length packets	storage	50B
Duncan et al [45]	2010	Location header	storage	-
Duncan et al [45]	2010	Date and Last-Modified	storage	-
Shen et al [166]	2018	HBCC	timing	500b/s
Castiglione et al [31]	2021	HTTP "control" message	timing	-

Tabella 2.16: Canali nascosti nel protocollo HTTP.

FTP. File Transfer Protocol è un protocollo a livello applicativo con il quale due host si scambiano dati, si basa su TCP ed è caratterizzato da un'architettura client-server. In Figura 2.17 è rappresentata la struttura del classico pacchetto FTP. Zou et al [219] hanno proposto due canali nascosti all'interno delle comunicazioni FTP. Il primo sfrutta il numero di volte in cui si invia un certo comando, mentre il secondo usa il numero di NOOP, intervallati da ABOR, inviati nei periodi di inattività. Anche Cabuk et al [29], sfruttando i comandi del protocollo FTP, riescono a definire delle sequenze tramite le quali codificano i dati segreti da inviare. In Tabella 2.17 sono riportati i metodi ma senza prestazioni in quanto non vengono modificati effettivamente i bit all'interno di pacchetti ma piuttosto assomigliano più a canali di temporizzazione.



Figura 2.17: Header del protocollo FTP.

¹¹Patrick LeBoutillier, htunnel, 2005, <https://metacpan.org/pod/Apache::HTTunnel>

Articolo	Anno	Metodo
Zou et al [219]	2005	n° times of command
Zou et al [219]	2005	n° NOOP and ABOR
Cabuk et al [29]	2008	sequences of commands

Tabella 2.17: Canali nascosti nel protocollo FTP.

DHCP. Il protocollo a livello Internet che assegna in modo dinamico gli indirizzi IP ai vari dispositivi è il DHCP (Dynamic Host Configuration Protocol). In Figura 2.18 osserviamo la struttura dei suoi pacchetti. Rios et al [150] hanno proposto ben 5 tecniche per nascondere dei dati all'interno dei pacchetti di protocollo, tra queste troviamo lo strumento HIDE_DHCP dedicato al protocollo DHCP. In Tabella 2.18 è riportato il valore di Byte trasmessi per ogni transazione completa e non per ogni pacchetto.

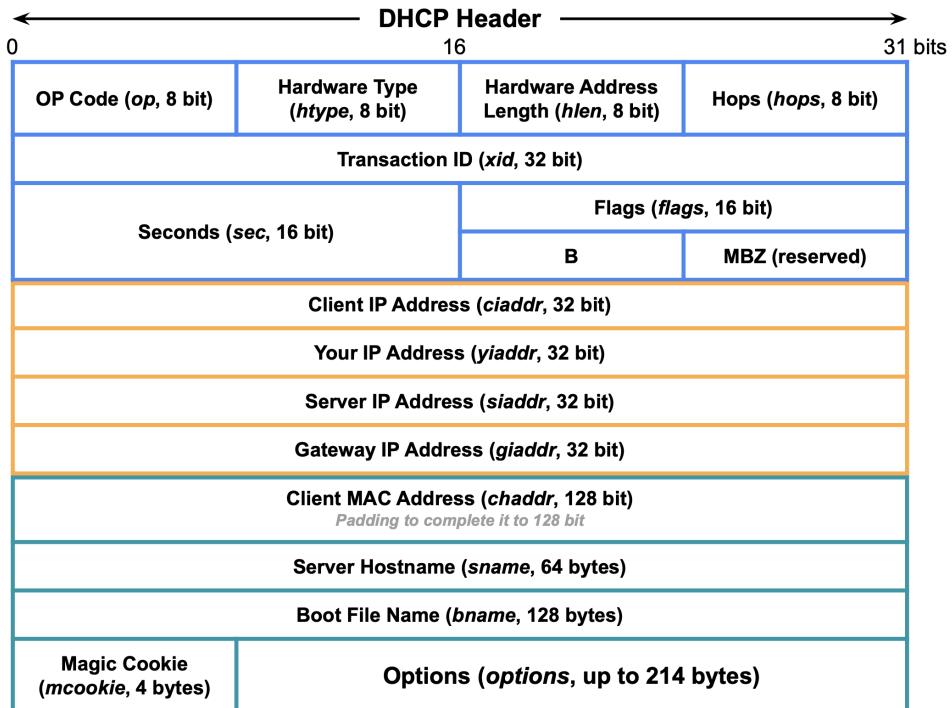


Figura 2.18: Header del protocollo DHCP.

Articolo	Anno	Metodo	bit/transaction
Rios et al [150]	2012	HIDE_DHCP	4-380B

Tabella 2.18: Canali nascosti nel protocollo DHCP.

SSL e TLS. Il Secure Socket Layer e il suo successore Transport Layer Security sono entrambi protocolli crittografici che criptano i dati e autenticano una connessione durante il trasferimento di dati. La Figura 2.19 rappresenta il meccanismo di cifratura e le intestazione del protocollo SSL. In [14, 4] è descritto un canale nascosto ibrido che sfrutta il canale di copertura delle reti in TCP e il canale subliminale in SSL/TLS. Heinz et al [69] esaminano 7 metodi di creazione di canali nascosti nel protocollo TLS, implementandone 3 al fine di studiarne le prestazioni e la fattibilità. Le prestazioni dei tre metodi sono riportate nella Tabella 2.19 ma non sono state presentate sotto forma di PRBR ma di *ratio*, cioè come rapporto tra bit segreti inviati rispetto a quelli palesi. Sottolineiamo che oggi il protocollo SSL è deprecato e le tecniche quindi non utilizzabili, ma sono state riportate per completezza sull'argomento.

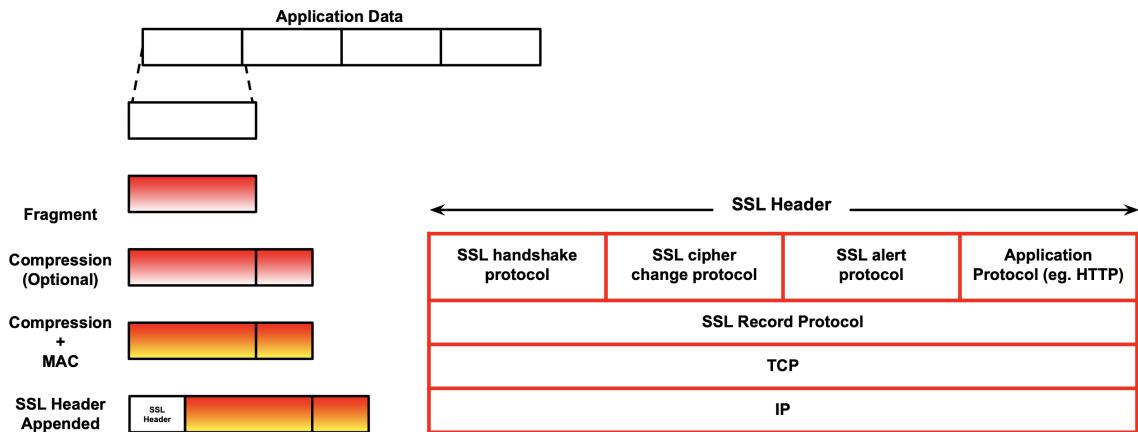


Figura 2.19: Header del protocollo SSL.

Articolo	Anno	Metodo	Ratio
Anjan et al [14, 4]	2010	SSL	-
Heinz et al [69]	2020	Alteration of Chipers and Compression	16B on 64B (21%)
Heinz et al [69]	2020	Record ContentType	1B on 555B (0.15%)
Heinz et al [69]	2020	Record-Length Encoding	1B on 616B (0.16%)

Tabella 2.19: Canali nascosti nel protocollo SSL/TLS.

DNS. Il protocollo Domain Name System è il responsabile della traduzione dei domini internet in indirizzi IP e risulta molto adatto per il tunneling di altri protocolli. In Figura 2.20 possiamo osservare la struttura dei suoi pacchetti. La maggior parte delle tecniche proposte utilizza i record NS, CNAME, TXT e NULL. Quest’ultimo al momento delle pubblicazioni era sperimentale mentre oggi è deprecato. Siccome quasi tutti i metodi presenti in letteratura che creano dei canali nascosti sfruttando DNS sono accompagnati da dalle implementazioni in vari linguaggi di programmazione, questi verranno trattati nel capitolo seguente. Tra le varie idee proposte registriamo un canale di temporizzazione che utilizza la cache negativa del DNS¹² e quella di Hoffmann [72] che usa il campo TTL. Nel 2019 Anagnostopoulos et al [13], tramite un proof of concept, hanno dimostrato la possibilità di incorporare dei dati segreti all’interno del campo DNSKEY che normalmente viene usata per fornire la chiave pubblica di una zona di dominio abilitata a DNSSEC. Tutte le tecniche sono comunque riportate in Tabella 2.20.

¹²DNS Covert channels and Bouncing Techniques, 2005, http://seclists.org/fulldisclosure/2005/Jul/att-452/p63_dns_worm_covert_channel.txt.

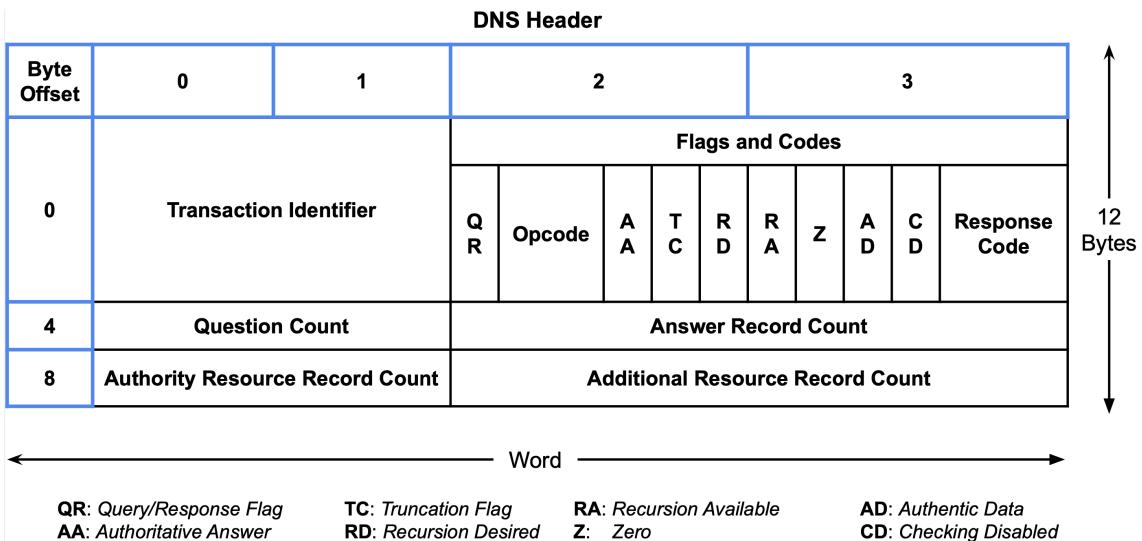


Figura 2.20: Header del protocollo DNS.

Articolo	Anno	Metodo	Tipo	PRBR
NSTX ¹³	2002	TXT records	storage	up to 255B
DNSCat ¹⁴	2004	CNAME records	storage	up to 255B
OzymanDNS ¹⁵	2004	TXT records	storage	up to 255B
Anonymous	2005	Negative caching	timing	1b
DNS2TCP ¹⁶	2008	TXT record	storage	up to 255B
TUNS [138]	2009	CNAME racords	storage	up to 255B
Iodine ¹⁷	2010	Download/upload different types of records	storage	Down. up to 1200B, Up. up to 255B
Hoffman [72]	2012	TTL of DNS packets	storage	2B
Anagnostopoulos et al [13]	2019	DNSKEY	storage	<2048b

Tabella 2.20: Canali nascosti nel protocollo DNS.

¹³savannah.nongnu.org, NSTX, 2002, <http://savannah.nongnu.org/projects/nstx/>.

¹⁴T. Pietraszek, DNSCat, 2004, <http://tadek.pietraszek.org/projects/DNScat/>.

¹⁵D. Kaminsky, OzymanDNS, 2004, <http://dankaminsky.com/2004/07/29/51/>.

¹⁶O. Dembour, C. Collignon, DNS2TCP, 2008, <http://hsc.fr/ressources/outils/dns2tcp/>.

¹⁷Kryo, iodine, 2010, <http://code.kryo.se/iodine/>.

Applicazioni in tempo reale. Un'altra area in forte sviluppo riguarda i canali nascosti in applicazioni in tempo reale come VoIP, giochi multigiocatore on-line, streaming A/V, etc. In queste applicazioni è plausibile che l'audio e/o il video vengano trasmessi separatamente, tramite il Real-time Transfer Protocol (RTP), il cui pacchetto è rappresentato in Figura 2.21, e il Real-time Transport Control Protocol, la parte che monitora la qualità del servizio. Una parte di RTP lavora a livello di trasporto su UDP e una parte a livello applicazione, inoltre prima di inviare i dati avviene una fase di segnalazione basata su protocolli come il *Session Initiation Protocol* (SIP)¹⁸ e il *Session Description Protocol* (SDP)¹⁹. Si registrano proposte di utilizzo di canali nascosti che sfruttano questo tipo di comunicazione non per esfiltrare dati ma per fornire un miglioramento del servizio [110, 111, 154]. La proposta di Liang et al [205] punta sulla modulazione della lunghezza dei pacchetti del traffico VoIP per inviare i dati segreti. Sempre Liang et al [96] ha proposto un canale di temporizzazione basato sulla riorganizzazione di pacchetti, che li divide in delimitatori e portanti. Gli autori hanno testato la loro proposta su tre tipi di traffico VoIP risultando essere robusto e invisibile ai test BER e KS. Con l'avvento delle reti mobili e del miglioramento delle loro prestazioni grazie alla versione 5G, le comunicazioni VoIP si sono evolute in VoLTE. Questa evoluzione però non ha bloccato l'inserimento di canali nascosti all'interno del protocollo. Uno degli autori più attivi nel settore è senza dubbio Zhang grazie alle sue pubblicazioni [212, 216, 214, 175, 215]. Nelle sue proposte va a sfruttare periodi di silenzio, tempo di invio tra pacchetti voce e audio e modulazione del timestamp oltre a presentare metodi per l'ottimizzazione dei canali di temporizzazione [211, 79]. La Tabella 2.21 riassume tutti i canali appena citati utilizzando per i canali di temporizzazione il valore bps.

Articolo	Anno	Metodo	Tipo	PRBR
Liang et al [96]	2018	packet-reordering	timing	<17.98b/s
Zhang et al [214]	2018	Adjusting Silence Periods	timing	varies
Zhang et al [175]	2018	Packet dropout	timing	1-4.1b/s
Zhang et al [212]	2019	Timestamp-regulation	storage	1-4b
Zhang et al [216]	2019	packet-reordering	timing	85.6b/s

¹⁸<https://www.rfc-editor.org/rfc/rfc3261>.

¹⁹<https://www.rfc-editor.org/rfc/rfc4566>.

Liang et al [205]	2022	Packets length	storage	varies
-------------------	------	----------------	---------	--------

Tabella 2.21: Canali nascosti nella tecnologia VoIP.

RTP e RTCP. Mazurczyk e Szczygielski [112, 114] descrivono come tramite i campi Padding, Extension Sequence Number e Timestamp del protocollo RTP possano essere usati per creare dei canali nascosti. Tra le varie tecniche riportano anche il metodo di Giffin [61], cioè modificare il bit meno significativo del campo Timestamp, perché è possibile applicarlo sia a RTP che al campo NTP Timestamp di RTCP. Le prestazioni migliori (fino a 160 bit per pacchetti) però si ottengono sfruttando i blocchi di report in Receiver Report (RR) e Sender Report (SR) in RTCP. Tra le proposte ce ne sono alcune che coinvolgono i meccanismi di sicurezza di Secure RTP e RTCP e una che sfrutta il ritardo intenzionale di pacchetti. All'interno di quest'ultimi vengono inseriti i dati segreti perché il ricevitore a conoscenza del meccanismo non li scarterà a priori ma ne estrarrà il contenuto. Bai et al [15] usano il campo interarrival jitter dell'header del RTCP, dopo aver analizzato le proprietà del campo nella rete corrente ne modulano il valore seguendo i dati ottenuti in precedenza. Lizhi et al [99] suggeriscono un canale di temporizzazione segreto che cerca di limitare l'invio di pacchetti RTCP per una maggiore impercettibilità e robustezza. L'idea principale è che se lo stegobit corrente è uguale al precedente viene inviato un pacchetto RTP, altrimenti vengono inviati sia un pacchetto RTP che uno RTCP. Forbes [54] nella sua tesi ha proposto e implementato un canale nascosto nel protocollo RTP andando a modulare il valore del timestamp, raggiungendo una capacità teorica di 350bps, mentre Gao et al [59] utilizzano il campo data. In Tabella 2.22 sono riportate tutte le proposte e le loro prestazioni in PRBR dove possibile, altrimenti la loro velocità in KB/s.

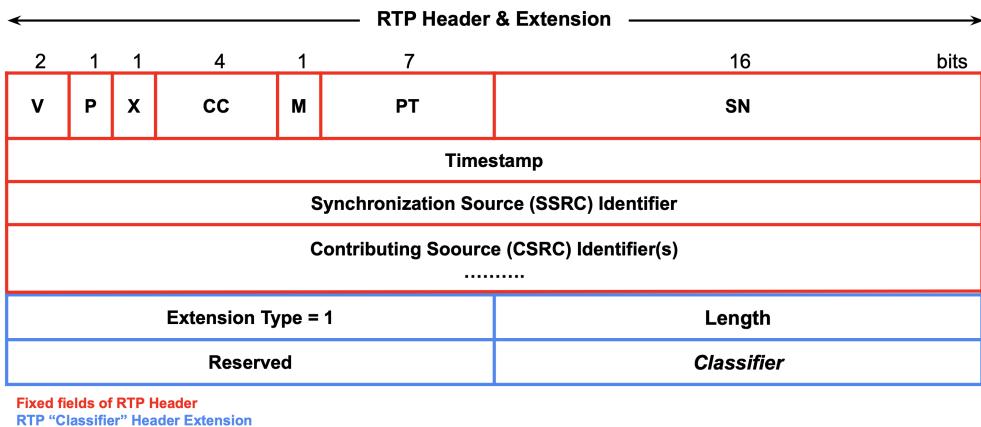


Figura 2.21: Header del protocollo RTP.

Articolo	Anno	Metodo	Tipo	PRBR
Giffin [61]	2002	LSB timestamp	storage	1b
Mazurczyk et al [114]	2008	Padding field	storage	8b
Mazurczyk et al [114]	2008	Extension header	storage	varies
Mazurczyk et al [114]	2008	Sequence Number field	storage	16b
Mazurczyk et al [114]	2008	Timestamp field	storage	32b
Mazurczyk et al [114]	2008	Timestamp field	timing	1b
Mazurczyk et al [114]	2008	NTP Timestamp in RTCP	timing	1b
Mazurczyk et al [114]	2008	Report blocks in RR and SR in RTCP	storage	up to 160b
Mazurczyk et al [114]	2008	authentication tag	storage	80b
LACK [114]	2008	intentionally delayed packets	timing	-
Bai et al [15]	2008	interarrival jitter in RTCP	storage	32b
Forbes [54]	2009	Modulation value of timestamp	storage	7b
Lizhi et al [99]	2012	Send RTP or RTCP packets	storage	32b
Gao et al [59]	2019	Data field	storage	45KB/s

Tabella 2.22: Canali nascosti nei protocolli RTP e RTCP.

SIP e SDP. Mazurczyk e Szczypiorski [113] suggeriscono di creare canali nascosti di archiviazione sfruttando molti campi di intestazione del protocollo SIP: From, Via,

Call-ID, CSeq, Max-Forwards e altri. Invece per il protocollo SDP le loro proposte interessano i campi v, o, s, t e k. La struttura-tipo di un messaggio SIP/SDP è riportata in Figura 2.22. Le ultime due proposte si basano sull’ordinamento degli header dei protocolli SIP e SDP, ad esempio il campo Call-ID dopo CSeq è interpretabile come 1, e sull’uso delle maiuscole in quanto i campi non sono sensibili. Nel loro studio suggeriscono anche di creare canali segreti sfruttando i meccanismi di sicurezza per fornire autenticazione e riservatezza. Una possibilità può essere quella di incorporare il contenuto SDP nel messaggio SIP INVITE e firmarlo utilizzando S/MIME. Unendo i vari metodi raggiungono una capacità del canale nascosto pari a 2.36Kb.

<pre> INVITE sip:53248717@79.14.212.52 SIP/2.0 Record-Route <sip:212.97.59.76:5061;lr=on;ftag=aslc5cblf6;rpp=np> Via: SIP/2.0/UDP 212.97.59.76:5061;bran=zhG4bKcae7.09b3e0b7.1 Via: SIP/2.0/UDP 193.227.104.23:5060;bran=zhG4K046660df;rport=5060 From: "+393470763341" <sip:+393470763341@sip.messagenet.it>;tag=aslc5cblf6 To: <sip:01042064015@212.97.59.76:5061> Contact: <sip:+393470763351@193.227.104.23> Call-ID: 355fee321ea539b629cb93c32d66088@sip.messagenet.it CSeq: 102 INVITE User-Agent: victor Max-Forwards: 69 Date: Mon, 28 Nov 2011 23:57:50 GMT Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO Supported: replaces Content-Type: application/sdp Content-Length: 381 </pre>	Headers
<pre> v=0 o=root 4369 4369 IN IP4 193.227.104.23 s=session c=IN IP4 193.227.104.23 t=0 m=audio 35302 RTP/AVP 18 3 97 8 0 101 a=rtpmap:18 G729/8000 a=fmtp:18 annexb=no a=rtpmap:3 GSM/8000 a=rtpmap:97 ILBC/8000 a=fmtp:97 mode=30 a=rtpmap:8 PCMA/8000 a=rtpmap:0 PCMU/8000 a=rtpmap:101 telephone-event/8000 a=fmtp:101 0-16 a=silenceSupp:off ---- a=ptime:20 a=sendrecv </pre>	Message Body

Figura 2.22: Header e body del protocollo SIP/SDP.

SSH. Secure Shell è un protocollo crittografico client-server destinato allo scambio sicuro di dati, l’esecuzione di comandi da remoto, la gestione dei dati e altri scopi, il cui pacchetto è rappresentato in Figura 2.23. Lucena et al [101] propongono di utilizzare il campo MAC per trasportare fino a 160 bit per pacchetto, oppure di intercettare il

traffico e inserire fino a 20B di dati prima del payload SSH crittografato, caratterizzati da un numero “magico” di 4 byte, così che il destinatario intuisce la presenza ed estragga i dati segreti. Perkins [141] suggerisce di usare il campo Random Padding per spedire fino a 255B di traffico nascosto per pacchetto. Tutte le varie proposte sono riassunte nella Tabella 2.23.

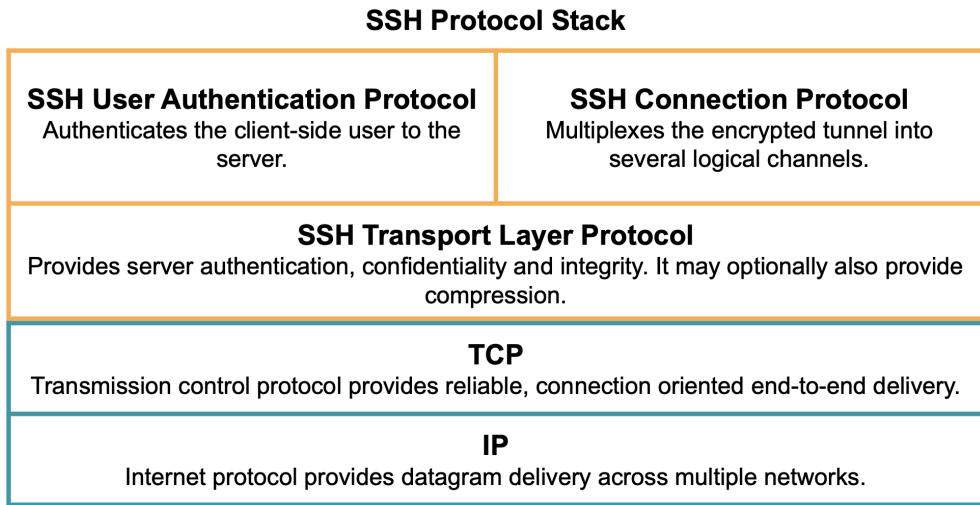


Figura 2.23: Header del protocollo SSH.

Articolo	Anno	Metodo	Tipo	PRBR
Lucena et al [101]	2005	MAC field	storage	up to 160b
Lucena et al [101]	2005	Beginning of payload	storage	up to 20B
Perkins [141]	2005	Random padding field	storage	up to 255B

Tabella 2.23: Canali nascosti nel protocollo SSH.

MQTT. Message Queue Telemetry Transport è un protocollo di messaggistica leggero progettato per le situazioni dove la banda è limitata, usato spesso in ambienti industriali e con dispositivi IoT. Velinov et al [187] nel loro studio propongono ben 7 canali segreti diretti e 6 indiretti basati proprio su MQTT. Nella Tabella 2.24 sono riportate le prestazioni potenziali dei vari canali in bit al secondo, ma per calcolare il valore preciso si necessita della variabile t , cioè il tempo tra due pacchetti, e questo

varia da rete a rete. Per ovviare a ciò si lascia il numero massimo di bit trasportabili dal metodo diviso la variabile t .

Metodo	Tipo	PRBR
PUBLISH: Application Message	direct	up to $524280/t$ bps
CONNECT: Client ID	direct	up to $184/t$ bps
CONNECT: User Name and/or Password	direct	up to $1048560/t$ bps
CONNECT: Keep Alive	direct	up to $16/t$ bps
Packet ID	direct	up to $16/t$ bps
PUBLISH: Topic Name	direct	up to $524280/t$ bps
SUBSCRIBE/UNSUBSCRIBE: Topic Filters	direct	up to $524280/t$ bps
PUBLISH: Topic Name and SUBSCRIBE: Topic Filters	indirect	up to $524280/t$ bps
Topic ordering and updates presence/absence	indirect	up to $n(\text{topics})/t$ bps
Persistent sessions	indirect	up to $1/(2t)$ bps
Presence/absence of the Retained message	indirect	up to $1/(2t)$ bps
Topic ordering and presence/absence of the Retained messages	indirect	up to $n(\text{topics})/(2t)$ bps
Information specific to the broker	indirect	up to $1/t$ bps

Tabella 2.24: Canali nascosti nel protocollo MQTT da Velinov et al [187] nel 2019.

NTP e PTP. Il Network Time Protocol e il Precision Time Protocol sono protocolli con il compito di mantenere la sincronizzazione temporale tra i vari dispositivi nella rete, il cui pacchetto è rappresentato in Figura 2.24. Lamshöft et al [91], dopo aver effettuato un’analisi approfondita, hanno identificato 49 canali nascosti basati su questi protocolli. Ne hanno implementati due risultando essere praticamente irrilevabili. Lavoro che interessa sempre il PTP è quello di Smith-Donovan et al [169] tramite il quale dimostrano la fattibilità di un attacco tramite un canale nascosto. Le caratteristiche principali dei tre metodi sono riportate in Tabella 2.25

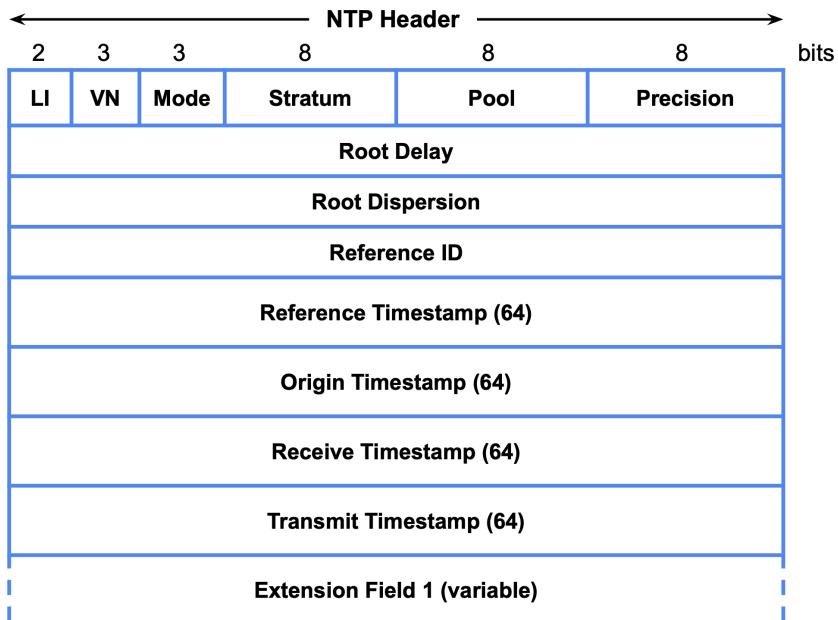


Figura 2.24: Header del protocollo NTP.

Articolo	Anno	Metodo	Tipo	PRBR
Lamshöft et al [91]	2022	User-data Value Modulation	storage	32b
Lamshöft et al [91]	2022	Value Modulation	storage	7b
Smith-Donovan et al [169]	2022	Unused bytes	storage	7B

Tabella 2.25: Canali nascosti nei protocolli NTP e PTP.

Canali di temporizzazione. Nel corso degli anni sono stati presentati approcci riguardanti i canali di temporizzazione che possono essere applicati non solo a uno ma a più protocolli. Uno di questi è quello di Yao et al [203] che studia la distribuzione di velocità di invio dei pacchetti e classifica i canali in deterministici e non deterministici. Tan et al [176] analizzano l'applicabilità dei canali di temporizzazione classici alle reti 4G/5G e presentano un modello di canale di temporizzazione per l'IoT. Houmansadr [75] nel 2011 ha creato CoCo, un canale nascosto che codifica il messaggio segreto tramite la modulazione di ritardi tra pacchetti nei flussi di rete. Le tre proposte sono riportate in Tabella 2.26.

Articolo	Anno	Metodo
Yao et al [203]	2009	Sending or not in a predetermined short period
Houmansadr [75]	2011	IPDs
Tan et al [176]	2018	IPDs

Tabella 2.26: Canali di temporizzazione nascosti.

2.11.5 Canali recenti

Analizzando la letteratura riguardante le proposte di canali nascosti, alcune delle tecniche pubblicate negli ultimi anni risultano più difficili da collocare in un livello del modello ISO/OSI, a causa della loro complessità e del coinvolgimento di più protocolli. Keller e Wendzel [86] propongono di nascondere dei dati attraverso le catene di hash mentre Heßeling e Jörg Keller [70] suggeriscono di modulare il valore di un numero in virgola mobile per trasferire dati segreti. Hash che sono coinvolti anche nello studio di Yuanzhang et al [205] tramite il quale propongono un robusto canale di temporizzazione basato sulla perdita di pacchetti. Anche tutto ciò che permette ai telefoni cellulari di fornire i vari servizi può essere usato al fine di creare canali nascosti. Un esempio è la proposta di Narteni et al [135] con il protocollo di messaggistica *Short Message Service* (SMS).

Tra le tecniche più innovative nel campo della steganografia di rete troviamo quelle che prevedono l'utilizzo di più canali nascosti, che lavorano in parallelo, per esfiltrare i dati. Akil et al [8] hanno proposto dei protocolli in grado di utilizzare questa tipologia di approcci. Dakhane e Narawade [41] utilizzano il campo di intestazione del numero di sequenza TCP (32bit) e il campo di identificazione IP (16bit) per garantire una larghezza di banda totale pari a 48 bit per pacchetto per il loro canale segreto. Schmidt [157] e Kalmbach et al [85] propongono due canali simili, uno pensato per piattaforme AMD e l'altro Intel. La strategia utilizzata è quella di modulare le informazioni sulla frequenza massima della CPU. Applicando il carico su più core della CPU, la frequenza di tutti i core viene ridotta. Questa caduta di frequenza può essere misurata da un ricevitore, consentendo la trasmissione di messaggi.

Una delle tecnologie più recenti è senza dubbio la blockchain, il cui utilizzo è esploso inizialmente grazie alle criptovalute. Questa tecnologia oggi è utilizzata in moltissimi ambiti e le più recenti tendenze dei canali nascosti vedono proprio un suo

impiego. Tian et al [178] ha presentato DLchain che si basa sulla sostituzione delle etichette fisse con quelle dinamiche. Anche la criptovaluta *Ethereum* è soggetta a canali nascosti, Shaoyuan Liu et al [98] utilizzano il campo Value di una transazione per trasferire dati. Tra i protocolli più giovani nel campo dell'IoT c'è *LoRa*, una tecnologia che utilizza bande di radiofrequenza sub-gigahertz libere come 433 MHz. Nonostante la sua recente introduzione è stato pubblicato uno strumento da parte di Hou e Zheng [74], CloackLoRa. che lavorando sulla parte al livello fisico del protocollo può inviare informazioni segrete anche a 250 metri.

Negli ultimi anni si è assistito a un aumento delle tipologie dei dispositivi in rete, tra le quali troviamo le auto. Questi strumenti vanno a formare le *Vehicular ad hoc network* (VANET), reti che applicano i principi delle MANET (*Mobile ad hoc network*) ma ai veicoli. Nel 2018 [174] è stato presentato il primo canale nascosto ibrido che sfrutta questa tecnologia. Reti MANET che anche loro non sono immuni da steganografia di rete, una dimostrazione la da Edwards [47] che tramite la temporizzazione del messaggio HELLO nel protocollo *OLSR* riesce a inviare dati segreti. Abbiamo diversi protocolli di routing per reti ad-hoc, tra cui troviamo ExOR dove all'interno del quale è stato dimostrata la possibilità di creare un canale segreto [7]. Tra le tecnologie di messaggistica più utilizzate c'è sicuramente Whatsapp che grazie alle sue caratteristiche offre molte opportunità per trasferire file in modo segreto. Mustafayeva et al [134] infatti studiano l'argomento e analizzano le varie opportunità di inserire dati nascosti in file grafici di diversi formati. Anche i nuovi standard di telefonia mobile LTE possono essere soggetti a questo tipo di tecniche, sia Liu et al [97] che Guangliang Xu et al [200] nel 2018 hanno presentato la loro idea. La seconda pubblicazione contiene la descrizione di un canale di temporizzazione e uno ibrido con la parte di archiviazione contenuta a livello MAC. La tecnologia *Software-Defined Networking* rappresenta un nuovo approccio nel cloud computing che facilita l'amministrazione e la configurazione delle apparecchiature. In letteratura è presente una proposta di canale segreto, Macchiatto [153], rivolta proprio a questa tecnologia. Riportiamo anche le proposte di Oakley et al [139] e di Mileva et al [125]. Nel primo caso il traffico viene convertito in traffico UDP, viene introdotto FTE (*Format Transforming Encryption*) per assomigliare a un altro protocollo e poi vengono temporizzati, mentre nel secondo caso i canali proposti utilizzano il *Constrained Application Protocol* (CoAP) [165]. In letteratura è possibile anche trovare uno

studio di una implementazione di un canale nascosto tramite il metodo StegBlocks che dimostra come avvengono le comunicazioni silenti di un malware [16].

Uno dei nuovi concetti introdotti è chiamato “dead drop” il quale non prevede un invio diretto dei dati, ma questi vengono memorizzati nella cache ARP di un host ignaro e poi estrapolate dal destinatario tramite SNMP (*Simple Network Management Protocol*) [156]. Infine tra le proposte più recenti e interessanti riportiamo quella di Castiglione et al [30] la quale prevede la creazione di e-mail spam con i dati steganografati nelle intestazioni. In Tabella 2.27 sono riportati i metodi citati e dove possibile anche un parametro riguardo le loro prestazioni, altrimenti è riportato il simbolo “-”.

Articolo	Anno	Metodo	Tipo	Performance
Castiglione et al [30]	2011	Header e-mail	storage	50B
Edwards [47]	2012	MANET	timing	~2b/s
Akhtari et al [7]	2017	MORE	storage	218-231b/s
Taheri et al [174]	2018	VANET	hybrid	4-6Kb/connection
Liu et al [97]	2018	LTE-A	storage	1600Kb/s
Guangliang Xu et al [200]	2018	LTE-A	hybrid	929b/s
Mileva et al [125]	2018	CoAP	storage	16-2040b/packet
Bak et al [16]	2018	StegBlocks	storage	1.85B/s
Akil et al [8]	2019	Multi-Covert Channel	storage	-
Schmidt [157]	2019	AMD Precision Boost 2	timing	1.9b/s
Dakhane et al [41]	2020	Number Sequence TCP + ID IP	storage	48b/packet
Kalmbach et al [85]	2020	Intel Turbo Boost	timing	12-61b/s
DLchain [178]	2020	dynamic label	storage	-
Mustafayeva et al [134]	2020	Whatsapp	storage	-
Shaoyuan et al [98]	2020	Value field	storage	-
Hou et al [74]	2020	CloakLoRa	physical	250meters
Oakley et al [139]	2020	TCP and FTE	storage	182b/s
Keller et al [86]	2021	OTP based on Hash Chain	storage	-
Yuanzhang et al [205]	2021	packet-dropping	timing	-
Narteni et al [135]	2021	SMS	storage	1200B
Macchiato [153]	2021	SDN	storage	5.9b/s
Heßeling et al [70]	2022	Sensor data transmission	storage	12b/packet

Tabella 2.27: Canali nascosti recenti.

Capitolo 3

Strumenti disponibili

In questo capitolo riportiamo alcuni strumenti che possono rivelarsi utili nello sviluppo di tool e successivamente analizzeremo gli strumenti pubblicati nel corso degli inerenti alla network steganography e in particolare all'implementazione di canali coperti.

3.1 Tipi di licenze

Prima di analizzare i vari strumenti in rete riteniamo opportuno affrontare l'argomento delle licenze software. Ogni proposta è accompagnata da una licenza cioè un contratto secondo il quale il proprietario dei diritti di sfruttamento economico ne stabilisce i limiti di circolazione, di utilizzo e di cessazione. La principale differenza si ha tra licenze per il software proprietario (closed source) e licenze per il software libero (open source). La prima categoria si divide principalmente in tre sottocategorie:

- EULA (End-User License Agreement): il contratto tra il fornitore di un programma software e l'utente finale (Es. Contratto di termini di servizio);
- shareware: i programmi sotto questo tipo di licenza possono essere ridistribuiti e possono essere utilizzati in modo gratuito per un periodo di prova, al termine del quale l'utente è obbligato all'acquisto di una licenza;
- freeware: l'utilizzo del software viene concesso a titolo gratuito come la ridistribuzione, ma ne viene vietato l'uso commerciale.

Per quanto riguarda le licenze per il software libero invece la principale differenza è data dalla clausola relativa al copyleft. Questa clausola rappresenta l'obbligo per i fruitori dell'opera, nel caso vogliano distribuire l'opera modificata, a farlo sotto lo stesso regime giuridico e generalmente sotto la stessa licenza. In questo modo le libertà associate al prodotto vengono sempre garantite ad ogni rilascio. Il copyleft è considerato più o meno forte a seconda del modo in cui si propaga nelle opere derivate e questo porta a quattro sottocategorie:

- Non copyleft: licenze BSD, MIT, PHP, Apache e WTFPL.
- Copyleft debole: GNU Lesser General Public License (LGPL), Mozilla Public License (MPL), Eclipse Public License (EPL).
- Copyleft forte: GNU General Public License (GPL).
- Network copyleft: GNU Affero General Public License (AGPL), European Union Public Licence (EUPL).

L'ultima categoria è la più restrittiva perché obbliga a rendere disponibile il codice sorgente del programma a tutti gli utenti che raggiungessero da remoto il server presso il quale il software è eseguito come servizio (Software as a service). Riassumendo il discorso sulle licenze open source possiamo affermare che BSD, MIT e Apache sono molto simili e quello che ci spinge a sceglierne una piuttosto che l'altra è solo un discorso di compatibilità perché, per esempio, MIT permette di includere progetti con altre licenze mentre BSD no. Oltre a queste, la GPL è la più restrittiva, e qualora si usasse uno strumento GPL insieme ad altri con diverse licenze, il prodotto finale dovrà avere questa licenza.

3.2 Strumenti utili allo sviluppo di tool

In questa sezione andremo ad analizzare i vari strumenti disponibili in grado di implementare un canale nascosto per le comunicazioni. Prima però di concentrarci sui vari tool presenti in rete esaminiamo alcuni strumenti che risultano utili, e a volte indispensabili, proprio per lo sviluppo di queste implementazioni.

3.2.1 CCEAP

Il primo strumento riportato si tratta di un protocollo caratterizzato dalla licenza GPL-3.0, il Covert Channel Educational Analysis Protocol (CCEAP). Questo protocollo di rete è stato ideato da Wendzel et al [192] con lo specifico scopo di insegnare i canali nascosti a professionisti e studenti. Il CCEAP risulta essere volontariamente vulnerabile contro diverse tecniche di occultamento in modo tale da poter essere utilizzato per le varie dimostrazioni. Il protocollo in questione ha una struttura semplice ed autoesplicativa ed è caratterizzato da una implementazione di circa 1000 righe per renderlo particolarmente accessibile agli studenti. Date le sue caratteristiche, questo protocollo può rivelarsi utile come generatore di traffico per effettuare i primi test di un tool. La sua implementazione si divide in due componenti, client e server entrambi scritti in C, dove la prima invia i pacchetti alla seconda. Il server mostrerà le informazioni su ogni pacchetto ricevuto, come ad esempio il tempo di interarrivo con il precedente, come riportato in Figura 3.2. In Figura 3.1 è rappresentata la struttura del pacchetto inviato con l'intestazione delle opzioni. Nella cartella scaricabile da GitHub sono presenti anche delle implementazioni di tecniche classiche di canali segreti, come la codifica delle informazioni nel campo Sequence Number.

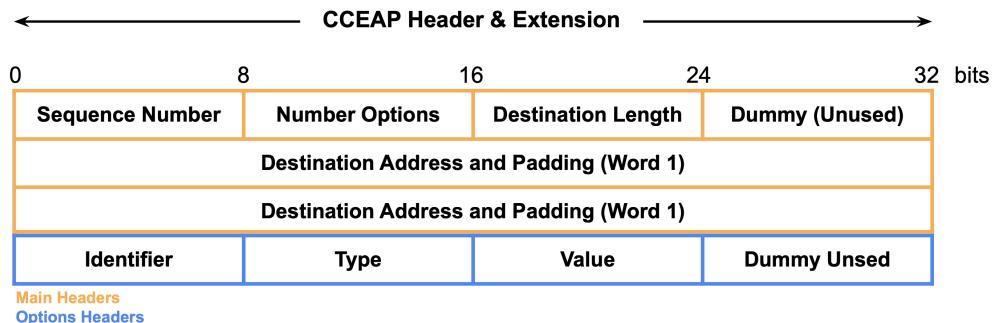


Figura 3.1: Struttura del pacchetto CCEAP.

```

CCEAP - Covert Channel Educational Analysis Protocol (Server)
=> version: 0.6.2, written by: Steffen Wendzel, https://www.wendzel.de
received data (12 bytes):
> time diff to prev pkt: 0.000
> sequence number: 1
> destination length: 0
> dummy value: 42
> destination + padding: XXXXXXXX
> number of options: 0
received data (12 bytes):
> time diff to prev pkt: 1.000
> sequence number: 2
> destination length: 0
> dummy value: 42
> destination + padding: XXXXXXXX
> number of options: 0
received data (12 bytes):
> time diff to prev pkt: 1.000
> sequence number: 3
> destination length: 0
> dummy value: 42
> destination + padding: XXXXXXXX
> number of options: 0
received data (12 bytes):
> time diff to prev pkt: 1.000
> sequence number: 4
> destination length: 0
> dummy value: 42
> destination + padding: XXXXXXXX
> number of options: 0

```

Figura 3.2: Output del processo server al ricevimento di ogni pacchetto.

3.2.2 Arpspoof

Arpspoof è uno strumento pubblicato da smikims su GitHub¹ con licenza GPL-3.0. Questo codice è stato sviluppato e poi pubblicato a scopo educativo, infatti tramite esso è possibile compiere un attacco di ARP spoofing contro un dispositivo presente sulla nostra stessa rete. Il file arpspoof.c invia due richieste ARP, una al gateway e una alla vittima così da raccogliere i loro indirizzi MAC. Una volta a conoscenza di queste informazioni il programma è in grado di costruire una risposta ARP per la vittima in cui viene comunicato l'indirizzo del gateway predefinito diverso da quello reale. Il codice fornisce anche l'opzione di effettuare un ping verso tutti i dispositivi nella rete senza inviare loro il pacchetto fasullo. Nella Figura 3.3 è riportata la sezione "Help" dello strumento.

¹smikims, arpspoof, 2014, <https://github.com/smikims/arpspoof>.

```
michele@michele-System-Product-Name:~/tools/arpspoof-master$ ./arpspoof -h
./arpspoof: invalid option -- 'h'
Usage: ./arpspoof [OPTION [ARG]]... VICTIM_IP
-t, --interface Network interface to use, given as the argument. If this option is not used, the interface defaults to the first running non
-loopback interface.
-r, --repeat Resends the packet continuously with a delay given in seconds by the argument. A delay of zero means only one packet is sent.
-a, --attacker-ip Choose another machine (other than the one running this program) as the one to be disguised.
-g, --gateway-ip Spoof to an IP (given as an argument) other than the default gateway.
-v, --verbose Prints out extra info about the machines involved.
```

Figura 3.3: Sezione "Help" dello strumento arpspoof.

3.2.3 Scapy

Scapy è un programma/libreria basato su Python, caratterizzato dalla licenza GPL-2.0² dedicato alla manipolazione dei pacchetti. Questo strumento mette a disposizione dell'utente la possibilità di inviare, sniffare, sezionare e falsificare i pacchetti di rete, funzionalità che possono essere sfruttate anche per sondare, scansionare o attaccare le reti. Il punto di forza di Scapy è la sua flessibilità e il grande numero di pacchetti dei vari protocolli di rete che è in grado di gestire. Date le sue potenzialità può essere utilizzato in sostituzione di altri programmi come hping, arpspoof, arp-sk, arping, p0f e persino alcune parti di Nmap, tcpdump e tshark (tutti descritti nella Sezione 3.2), inoltre è stato dimostrato comportarsi molto bene anche per compiti più difficili come inviare frame non validi, iniezione di frame 802.11 personalizzati e combinazione di tecniche (VLAN hopping + ARP cache poisoning, decodifica VoIP su canale criptato WEP,...)[26]. Scapy funziona nativamente su Linux, Windows, OSX e sulla maggior parte degli UNIX con libpcap, la stessa base di codice viene eseguita sia su Python2 che su Python3. Nella Figura 3.4 è riportato il messaggio all'avvio del tool.

²secdev, Scapy, 2022, <https://scapy.net>.

```
michele@michele-System-Product-Name:~/tools/scapy$ ./run_scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

          aSPY//YASa
          apyyyyCY/////////YCa
          sY////////YSpcs  scpCY//Pp      | Welcome to Scapy
          ayp ayyyyyyyySCP//Pp           syY//C
          AYAsAYYYYYYYYY//Ps           cY//S
          pCCCCCY//p                 cSSps y//Y
          SPPPP//a                  pP///AC//Y
          A//A                     cyP///C
          P///Ac                   sC///a
          P///YCpc                A//A
          scccccp///pSP///p           p//Y
          sY/////////y caa           S//P
          cayCyayP//Ya             pY/Ya
          sY/PsY///YCc              aC//Yp
          sc  sccaCY//PCypaapyCP//YSs
          spCPY//////YPSps
          ccaacs

                                         | https://github.com/secdev/scapy
                                         | Have fun!
                                         | We are in France, we say Skappee.
                                         | OK? Merci.
                                         | -- Sebastien Chabal
                                         |
                                         | using IPython 7.13.0
>>> 
```

Figura 3.4: Interfaccia all'avvio di Scapy.

3.2.4 Hping

Hping3 è un generatore e analizzatore di pacchetti per il protocollo TCP/IP, scritto da Salvatore Sanfilippo (noto anche come *antirez*) e pubblicato con licenza GPL-2.0³. Il software si basa sullo stesso concetto del comando Unix ping ma fa uso anche di protocolli diversi dall'ICMP, come TCP e UDP. Inoltre permette anche di gestire la costruzione a piacere del pacchetto IP (TTL, DF, ID, MTU, TOS e varie IP options) permettendo quindi anche l'invio di pacchetti IP semplici, che al loro interno sono privi di un protocollo di livello 4. Hping3 viene utilizzato come strumento di verifica di sicurezza e di test di firewall e reti. Con l'ultima versione Hping3 fornisce la possibilità di preparare degli script usando il linguaggio Tcl, inoltre implementa un motore per la descrizione di pacchetti TCP/IP in formato leggibile per facilitare il programmatore. In Figura 3.5 la schermata "Help" a terminale dello strumento Hping3.

³Salvatore Sanfilippo, hping, 2003, <https://www.kali.org/tools/hping3/>.

```

michele@michele-System-Product-Name:~/tools/scapy$ hping3 -h
usage: hping3 host [options]
-h --help      show this help
-v --version   show version
-c --count     packet count
-i --interval wait (UX for X microseconds, for example -i u1000)
--fast         alias for -i u10000 (10 packets for second)
--faster        alias for -i u1000 (100 packets for second)
--flood         sent packets as fast as possible. Don't show replies.
-n --numeric   numeric output
-q --quiet     quiet
-I --interface interface name (otherwise default routing interface)
-V --verbose    verbose mode
-D --debug     debugging info
-z --bind      bind ctrl+z to ttl          (default to dst port)
-Z --unbind    unbind ctrl+z
--beep        beep for every matching packet received
Mode
default mode  TCP
-o --rawip     RAW IP mode
-1 --icmp     ICMP mode
-2 --udp      UDP mode
-8 --scan      SCAN mode.
-9 --listen    Example: hping --scan 1-30,70-90 -S www.target.host
listen mode
IP
-a --spoof    spoof source address
--rand-dest   random destination address mode. see the man.
--rand-source random source address mode. see the man.
-t --ttl      ttl (default 64)
-N --id       id (default random)
-W --winid    use win* id byte ordering
-r --rel      relativize id field        (to estimate host traffic)
-f --frag     split packets in more frag. (may pass weak acl)
-x --morefrag set more fragments flag
-y --dontfrag set don't fragment flag
-g --fragoff  set the fragment offset
-m --mtu      set virtual mtu, implies --frag if packet size > mtu

```

Figura 3.5: Sezione "Help" dello strumento Hping3.

3.2.5 Arp-sk

Arp-sk è uno strumento progettato per manipolare le tabelle ARP di tutti i tipi di apparecchiature⁴ attraverso l'invio di pacchetti appropriati. Per raggiungere l'obiettivo bisogna manipolare nel corretto modo i 7 parametri più importanti di un messaggio ARP in una rete Ethernet/IP:

- il livello Ethernet fornisce 2 indirizzi (src e dst),
- il livello ARP contiene il codice del messaggio (richiesta o risposta) e le coppie (eth, ip) per la sorgente e la destinazione, come riportato in Figura 2.13.

Tramite questo tool è possibile effettuare MAC spoofing, ARP spoofing, ARP cache poisoning, sniffing, proxying e hijacking, escaping firewall (spoofing), man in the middle e DoS. In Figura 3.6 la schermata iniziale del tool.

⁴Frédéric Raynal, Èric Detoisien, Cédric Blancher, arpsk, 2004, <https://web.archive.org/web/20180223202629/sid.rstack.org/arp-sk/>.

```

michele@michele-System-Product-Name: ~/tools/arp-sk/src$ arp-sk version 0.0.16 (Thu Dec 1 16:53:02 CET 2022)
Author: Frederic Raynal <pappy@security-labs.org>

Usage: arp-sk
-w --who-has      send a ARP Who-has
-r --reply        send a ARP Reply
-p --arping       (bad) RARP emulation (NOT YET IMPLEMENTED)
-m --arpmtm       Man in the Middle (NOT YET IMPLEMENTED)

-d --dst          dst in link layer (<hostname|hostip|MAC>)
-s --src          dst in link layer (<hostname|hostip|MAC>)
--rand-hwa        set random addresses in link header
--rand-hwa-dst   set random dst in link header
--rand-hwa-src   set random src in link header

-D --arp-dst     dst in ARP message (<hostname|hostip|[:MAC]>)
-S --arp-src     dst in ARP message (<hostname|hostip|[:MAC]>)
--rand-arp       set random addresses in ARP message
--rand-arp-dst   set random dst addresses in ARP message
--rand-arp-src   set random src addresses in ARP message
--rand-arp-hwa-dst set random dst MAC address in ARP message
--rand-arp-log-dst set random dst IP address in ARP message
--rand-arp-hwa-src set random src MAC address in ARP message
--rand-arp-log-src set random src IP address in ARP message

-l --interface   specify interface (eth0)
-c --count       # of packets to send (infinity)
-T --time        wait the specified number of seconds between sending \
each packet (or X micro seconds with -T ux)
--rand-time     randomize the sending period of the packets
--beep          beeps for each packet sent
-n --network    broadcast address to use for icmp-timestamp
--use-ts        an icmp-timestamp is send to resolve MAC to IP
-N --call-dns   force address resolution in outputs (default is off)
-V --version    print version and exit
-h --help        this help :)

michele@michele-System-Product-Name:~/tools/arp-sk/src$ 

```

Figura 3.6: Schermata all'avvio del tool arp-sk.

3.2.6 Arping

Arping è uno strumento software per scoprire e sondare gli host su una rete di computer pubblicato con licenza GPL-2.0⁵. Arping sonda gli host sul collegamento di rete esaminato inviando frame di livello link con il metodo di richiesta Address Resolution Protocol (ARP). I pacchetti sono inviati ai vari host che sono identificati dall'indirizzo MAC dell'interfaccia di rete. Questo tool può utilizzare ARP per risolvere un indirizzo IP fornito dall'utente. La funzione di arping risulta molto simile a quella dell'utilità ping che sonda la rete attraverso ICMP. In Figura 3.7 la sezione "Help" della versione scritta da Thomas Habets.

```

michele@michele-System-Product-Name:~/tools/arp-sk/src$ arping -h
ARPing 2.20, by Thomas Habets <thomas@habets.se>
usage: arping [ -0aAbdDeFpPqrRuUv ] [ -w <sec> ] [ -W <sec> ] [ -S <host/ip> ]
           [ -T <host/ip> ] [ -s <MAC> ] [ -t <MAC> ] [ -c <count> ]
           [ -C <count> ] [ -i <interface> ] [ -m <type> ] [ -g <group> ]
           [ -V <vlan> ] [ -Q <priority> ] <host/ip/MAC | -B>
For complete usage info, use --help or check the manpage.
michele@michele-System-Product-Name:~/tools/arp-sk/src$ 

```

Figura 3.7: Schermata help del tool arping.

⁵Thomas Habets, arping, 2007, <https://github.com/ThomasHabets/arping>.

3.2.7 p0f

P0f, acronimo di Passive OS fingerprint, è un software open source con licenza GPL-3.0, che attraverso la tecnica del fingerprinting passivo riesce a determinare il sistema operativo di un host remoto⁶. Nella pratica analizza il traffico presente in rete che interessa l'host indicato e basandosi sulle caratterizzazioni presenti nei pacchetti (TTL, DF, TOS, ...) stabilisce quale sistema operativo è presente sulla macchina designata. In Figura 3.8 la schermata "Help" del tool.

```
michele@michele-System-Product-Name:~/tools$ p0f -h
--- p0f 3.09b by Michal Zalewski <lcamtuf@coredump.cx> ---

p0f: invalid option -- 'h'
Usage: p0f [ ...options... ] [ 'filter rule' ]

Network interface options:

-i iface  - listen on the specified network interface
-r file   - read offline pcap data from a given file
-p        - put the listening interface in promiscuous mode
-L        - list all available interfaces

Operating mode and output settings:

-f file   - read fingerprint database from 'file' (/etc/p0f/p0f.fp)
-o file   - write information to the specified log file
-s name   - answer to API queries at a named unix socket
-u user   - switch to the specified unprivileged account and chroot
-d        - fork into background (requires -o or -s)

Performance-related options:

-S limit  - limit number of parallel API connections (20)
-t c,h    - set connection / host cache age limits (30s,120m)
-m c,h    - cap the number of active connections / hosts (1000,10000)

Optional filter expressions (man tcpdump) can be specified in the command
line to prevent p0f from looking at incidental network traffic.

Problems? You can reach the author at <lcamtuf@coredump.cx>.

michele@michele-System-Product-Name:~/tools$ █
```

Figura 3.8: Schermata help del tool p0f.

⁶Michał Zalewski, p0f, 2000, <https://www.kali.org/tools/p0f/>.

3.2.8 Nmap

Nmap ("Network Mapper") è un'utility gratuita e open source con licenza GPL-2.0⁷, rivolta alla scoperta delle reti e la verifica della sicurezza. Questo software utilizza i pacchetti IP grezzi in modo da determinare quali host sono disponibili nella rete, quali servizi offrono questi host, quali sistemi operativi stanno eseguendo e altre decine di caratteristiche. Inizialmente fu progettato per scansionare rapidamente grandi reti ma nonostante ciò funziona molto bene anche su singoli host. Oggi la suite Nmap è disponibile per tutti i principali sistemi operativi (Windows, Linux, MacOS) e comprende:

- il classico eseguibile a riga di comando,
- un'interfaccia grafica avanzata e un visualizzatore di risultati (Zenmap⁸), visibile in Figura 3.9,
- il reindirizzamento e il debug dei dati (Ncat⁹),
- un'utility per il confronto dei risultati delle scansioni,
- uno strumento per la generazione dei pacchetti e l'analisi delle risposte (Nping¹⁰).

⁷Gordon Lyon, nmap, 1997, <https://nmap.org>.

⁸<https://nmap.org/zenmap/>.

⁹<https://nmap.org/ncat/>.

¹⁰<https://nmap.org/nping/>.

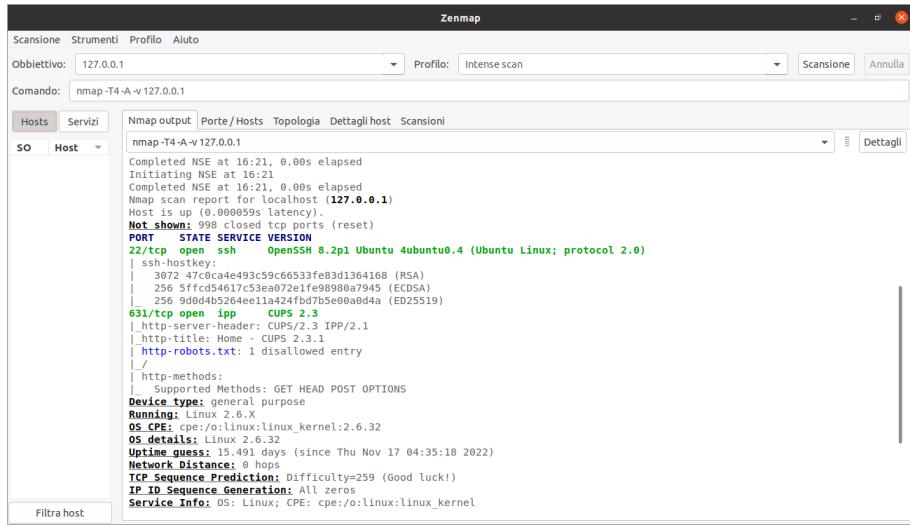


Figura 3.9: Interfaccia grafica Zenmap.

3.2.9 Tcpdump e Libpcap

Uno degli analizzatori di pacchetti più potenti è tcpdump, che si basa sulla libreria portatile C/C++ libpcap, rivolta alla cattura del traffico di rete. Questo strumenti, che possono essere utilizzati anche per monitorare la rete, sono sotto licenza BSD 3-clause e nel corso degli anni sono stati migliorati grazie ai contributi della community¹¹. Tcpdump è disponibile come pacchetto nativo il che lo rende facile da installare e aggiornare, per provare delle versioni più recenti è comunque possibile compilarlo direttamente dal codice sorgente. Date le sue caratteristiche compila e funziona su molte piattaforme tra cui Linux, MacOS, FreeBSD e Windows, ma per quest'ultimo è necessario WinPcap o Npcap e Visual Studio con CMake. Come detto in precedenza tcpdump basa il suo funzionamento sulla libreria libpcap che necessita di essere recuperata e compilata precedentemente. Questa libreria è nata per fornire una API unica e indipendente dal sistema dato che tutti i fornitori di sistemi forniscono un'interfaccia diversa per la cattura dei pacchetti.

¹¹Lawrence Berkeley National Laboratory, tcpdump & libpcap, 1991, <https://www.tcpdump.org>.

3.2.10 TShark

TShark è un analizzatore di protocolli di rete diffuso tramite licenza GPL-2.0¹². Tramite questo programma è possibile catturare dei pacchetti da una rete o di leggere i dati da un file di cattura precedentemente salvato a patto che sia nel formato .pcapng, lo stesso utilizzato da Wireshark. Senza alcuna opzione impostata, TShark funziona come tcpdump: utilizza la libreria pcap per catturare il traffico dalla prima interfaccia di rete disponibile e visualizza una riga di riepilogo sullo standard output per ogni pacchetto ricevuto. La cattura dei pacchetti avviene tramite libreria pcap alla quale deve essere passato un filtro e i pacchetti che non soddisfano quest'ultimo vengono scartati. Ad oggi lo strumento TShark fa parte del famoso software Wireshark descritto nella Sezione 3.2.11.

3.2.11 Wireshark

Wireshark¹³ è l'analizzatore di protocolli di rete più importante e diffuso al mondo ed è caratterizzato da una licenza GPL-2.0. Questo programma permette di vedere ciò che accade sulla rete a livello microscopico ed è lo standard de facto (e spesso de jure) in molte aziende commerciali e no-profit, agenzie governative e istituzioni educative. Lo sviluppo di Wireshark prospera grazie al contributo volontario di esperti di rete di tutto il mondo ed è la continuazione di un progetto iniziato da Gerald Combs nel 1998. Nel corso del tempo è stato oggetto di importanti ottimizzazioni e miglioramenti fino a includere altri software per offrire tramite un unico software molteplici strumenti. Ad oggi tra le principali funzionalità offerte da Wireshark troviamo:

- Ispezione approfondita di centinaia di protocolli (in continuo aumento).
- Acquisizione del traffico online e successiva analisi offline.
- Browser dei pacchetti a tre pannelli.
- Multipiattaforma: Windows, Linux, macOS, Solaris, FreeBSD, NetBSD, etc.
- Ricca analisi VoIP.

¹²The Wireshark team, TShark, Dump and analyze network traffic, 2008, <https://www.wireshark.org/docs/man-pages/tshark.html>.

¹³The Wireshark team, Wireshark, 1998, <https://www.wireshark.org>.

- Lettura/Scrittura di diversi formati di file di acquisizione: tcpdump (libpcap), Pcap NG, Catapult DCT2000, etc.
- Supporto della decodifica per molti protocolli, tra cui IPsec, ISAKMP, Kerberos, etc.

Uno strumento del genere risulta fondamentale nei test di sviluppo di un tool di implementazione di canali nascosti per simulare l'analisi della rete da parte di un possibile avversario. In Figura 3.10 è riportato uno screenshot durante un'acquisizione dalla rete con Wireshark.

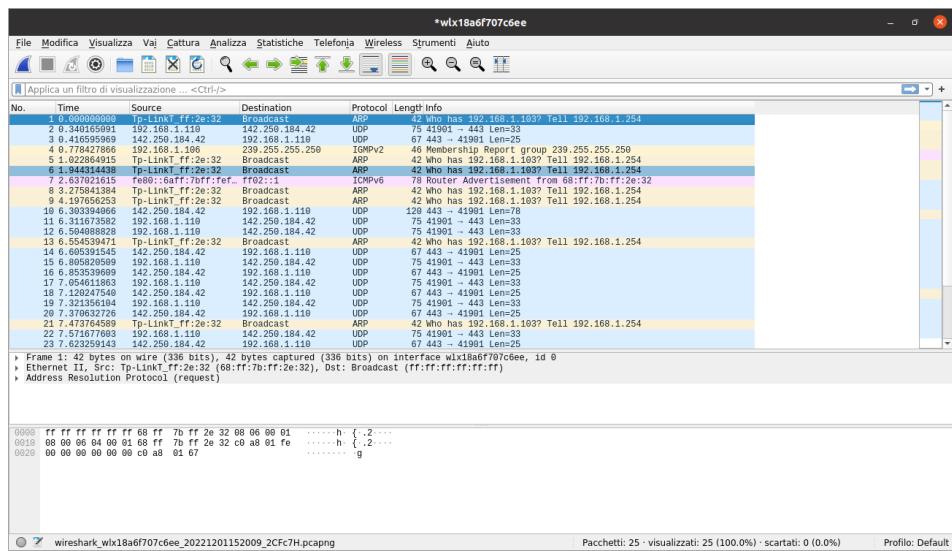


Figura 3.10: Interfaccia grafica di Wireshark.

3.2.12 NeFias

Questo strumento dotato di licenza GPL-3.0, è stato sviluppato per network anomaly detection e netowork forensics, risultando adatto alla rilevazione di canali occulti (steganografia di rete). Scritto inizialmente da Steffen Wendzel¹⁴ vuole rappresentare un banco di prova il più possibile accessibile e facile da usare. L'obiettivo dell'autore era quello di avere uno strumento molto leggero, che non dipendesse da linguaggi come Java e Python, che fosse banale da distribuire e che funzionasse su linux standard, per

¹⁴Steffen Wendzel, NeFias, 2017, <https://github.com/cdpxe/nefias/>.

questo ha deciso di svilupparlo in bash. NeFiAS richiede due tipi di nodi: un nodo master e almeno un nodo slave. Dato che entrambi teoricamente potrebbero essere installati sulla stessa macchina è possibile anche l'installazione su un nodo fisico. Il nodo master è responsabile dell'esecuzione dei lavori per cui è lui che legge il traffico, estrae metadati, divide i dati e li carica nei nodi slave, i quali hanno il compito di svolgere le operazioni. Le prestazioni di questo sistema dipendono dal numero dei nodi slave piuttosto che da una implementazione efficiente. La comunicazione tra i nodi si basa su protocollo OpenSSH. Questo strumento risulta molto utile nei test finali dello sviluppo di un tool per verificare se la tecnica implementata può essere rilevata da tecniche di anomaly detection.

3.2.13 ExifTool

ExifTool¹⁵ è una libreria Perl indipendente dalla piattaforma e un'applicazione a riga di comando per leggere, scrivere e modificare le metainformazioni in un'ampia gamma di file. ExifTool supporta molti formati di metadati diversi, tra cui EXIF, GPS, IPTC, XMP, JFIF, GeoTIFF, ICC Profile, Photoshop IRB, FlashPix, AFCP e ID3, Lyrics3, nonché le note del produttore di molte fotocamere digitali di Canon, Casio, DJI, FLIR, FujiFilm, GE, GoPro, HP, JVC/Victor, Kodak, Leaf, Minolta/Konica-Minolta, Motorola, Nikon, Nintendo, Olympus/Epson, Panasonic/Leica, Pentax/Asahi, Phase One, Reconyx, Ricoh, Samsung, Sanyo, Sigma/Foveon e Sony.

3.2.14 GTFOBins

GTFOBins¹⁶ è una raccolta di binari Unix che possono essere utilizzati per aggirare le restrizioni di sicurezza in sistemi mal configurati. Le funzioni contenute nel progetto sono tutte legittime ma possono essere utilizzate per ottenere una reverse shell, un innalzamento di privilegi o il trasferimento di file. Anche se non è incentrato sulla steganografia di rete, alcuni di questi binari e l'utilizzo proposto possono rappresentare una parte di tool di più grandi dimensioni.

¹⁵Phil Harvey, ExifTool, 2003, <https://exiftool.org>.

¹⁶Emilio Pinna e Andrea Cardaci, GTFOBins, 2018, <https://gtfobins.github.io>.

3.2.15 LOLBAS

Lo stesso progetto ma rivolto alla piattaforma Windows si chiama LOLBAS¹⁷. Si tratta di una raccolta di binari, script e librerie e relativi utilizzi per svolgere operazioni non previste. Sono riportati anche tutti gli ID delle relative tecniche registrate nel MITRE ATT&CK¹⁸, una raccolta disponibile gratuitamente a livello globale di tecniche e tattiche utilizzate dai criminali informatici.

3.2.16 Metasploit

Riteniamo giusto riportare in questa sezione anche il famoso framework Metasploit¹⁹. Questo progetto raccoglie informazioni su vulnerabilità e semplifica le operazioni di penetration testing. Il suo utilizzo per l'exploiting di un sistema prevede cinque fasi: scegliere e codificare un exploit (codice che penetra nel sistema, ne sono archiviati più di 2000), verificare se il sistema è suscettibile all'exploit, scegliere il payload, scegliere la codifica per eludere gli IPS ed eseguire l'exploit. Questo tool risulta essere tra i più utilizzati per lo sviluppo di exploit data la grande raccolta di payload che mette a disposizione. Quelli che permettono di controllare lo schermo, navigare su internet, caricare e scaricare file sulla macchina vittima sono raccolti in Meterpreter²⁰.

3.2.17 Precisazioni

Naturalmente questa non può essere considerata una lista completa degli strumenti utili allo sviluppo di programmi di steganografia di rete. Infatti nella Tabella 3.1 non andremo a riportare proposte come Binutils²¹, una raccolta di binari che comprende "strings" che restituisce le stringhe presenti in un file. Questi strumenti non sono strettamente legati alla steganografia di rete ma per effettuare piccoli test e verifiche durante lo sviluppo potrebbero rivelarsi adatti. Nella Tabella 3.1 sono riportati tutti i dati relativi ai tool sopra citati.

¹⁷LOLBAS-Project, Living Off The Land Binaries and Scripts (and now also Libraries), 2018, <https://lolbas-project.github.io>.

¹⁸<https://attack.mitre.org>.

¹⁹Rapid7, Strategic Cyber LLC, Metasploit, 2003, <https://www.metasploit.com>.

²⁰<https://docs.metasploit.com/docs/using-metasploit/advanced/meterpreter/>.

²¹<https://www.gnu.org/software/binutils/>.

3.2.18 NELphase

Alcuni canali segreti sono in grado di eseguire una fase di apprendimento dell’ambiente di rete chiamata fase NEL. Una volta che hanno ottenuto le informazioni necessarie riguardo a firewall, normalizzatori di traffico e guardiani attivi, sono in grado di decidere quale sia il miglior modo per scambiare segretamente dei dati. È durante questa fase che i due peer comunicanti adattano la loro strategia in base all’ambiente dove operano. NELphase è la prima implementazione pubblica, scritta in C, di una fase NEL basata su `scapy` e `libpcap`. Prevede anche un canale di feedback nel caso in cui il canale nascosto fosse bloccato durante i test.

Nome	Tipologia	Licenza	Linguaggio	Descrizione
CCEAP [192]	Protocollo	GPL-3.0	C	Generatore di traffico vulnerabile
arpspoof	Software	GPL-3.0	C	Arp spoofing
Scapy	Software/Libreria	GPL-2.0	Python	Manipolazione di pacchetti
Hping3	Software	GPL-2.0	C	Generatore e analizzatore di pacchetti TCP/IP
arp-sk	Software	GPL-2.0	C	Manipolare le tabelle ARP
Arping	Software	GPL-2.0	C	Scopre e sonda host in rete
p0f	Software	GPL-3.0	C	Determina OS di un host remoto
nmap	Software	GPL-2.0	Python, C, C++, Lua, Java	Scoperta della rete e verifica della sicurezza
tcpdump e libpcap	Software	BSD 3-clause	C/C++	Analizzatore di pacchetti e debugger della rete
Tshark	Software	GPL-2.0	C/C++	Analizzatore di protocolli
Wireshark	Software	GPL-2.0	C/C++	Analizzatore di protocolli
NeFiAS	Software	GPL-3.0	BASH	Rilevatore di canali segreti
ExifTool	Software/Libreria	GPL-3.0	Perl	Modifica metadati
GTFOBins	Binari	GPL-3.0	Vari	Raccolta di binari utilizzabili per altri scopi su Unix
LOLBAS	Binari/Script/ Librerie	GPL-3.0	Vari	Raccolta di strumenti utilizzabili per altri scopi su Windows
Metasploit	Framework	BSD 3-clause	Ruby	Framework per penetration test

Tabella 3.1: Strumenti utili nello sviluppo di software per l’implementazione di canali nascosti.

3.3 Tool

Dopo aver analizzato tutti quegli strumenti che risultano utili nelle fasi di sviluppo e test di software in grado di implementare canali di comunicazione coperti, riportiamo di seguito i migliori programmi già disponibili in rete che forniscono questa possibilità.

3.3.1 Livello data link

In questo livello sono state registrate 3 implementazioni le cui caratteristiche sono riassunte in Tabella 3.2.

CCW. Nel 2011 Ricardo Goncalves ha presentato all'interno della sua tesi lo strumento CCW [65], in grado di implementare un canale nascosto all'interno delle reti WLAN descritte dallo standard IEEE 802.11. Questo strumento utilizza il campo Protocol Version nell'intestazione MAC per trasferire i dati, questo perché è in grado di creare dei frame Clear To Send (CTS) e Acknowledgement (ACK) modificati. Il prototipo è disponibile con licenza GPL-3.0, è stato scritto in Python ed è stato testato su ambiente Linux.

WiFi_CCC. Il tool WiFi_CCC²² disponibile su GitHub fornisce un servizio di chat implementato in modo nascosto attraverso la modifica della struttura dei pacchetti nelle reti 802.11. Una volta collegata una scheda WiFi in modalità monitor che supporti l'iniezione del traffico basta lanciare l'eseguibile. All'avvio chiederà un nickname e il nome segreto della stanza, ed è proprio su quest'ultimo che poi si baseranno le configurazioni. Ognuno a conoscenza di questo nome può accedere e interagire con gli altri, inoltre ogni utente sarà anche un ripetitore per aumentare il segnale tra i nodi. Il tool fornisce la possibilità di inviare file o immagini a chiunque. Il tutto avviene tramite pacchetti 802.11 malformati che di solito vengono scartati silenziosamente dalle schede WiFi standard.

²²yadox666, WiFi_CCC, 2017, https://github.com/yadox666/WiFi_CCC.

GhostTunnel. GhostTunnel²³ è un metodo di trasmissione nascosto disponibile su GitHub che prevede il rilascio dell’agente attraverso un dispositivo Human Interface Devices (HID). Questo tool utilizza 802.11 Probe Request Frames e Beacon Frames per comunicare e non ha bisogno di stabilire una connessione wifi perché inserisce i dati nelle richieste di probe e beacon. Il server e l’agente sono implementati in C/C++ e il secondo non ha bisogno di privilegi elevati. Il server invece è previsto che sia installato su una piattaforma Linux e per eseguirlo sono necessarie una o due schede wifi usb che supportino la modalità monitor e l’iniezione di pacchetti. Ne è disponibile anche una versione scritta in GO²⁴.

Nome	Licenza	Linguaggio	Descrizione				Rilascio	Ultima Modifica
CCW [65]	GPL-2.0	Python	Canale coperto 802.11	nelle reti	2011	2011		
WiFi_CCC	-	Python	Chat tramite 802.11 malformati	pacchetti	2016	2017		
GhostTunnel	-	C/C++	Comunicazione attraverso richieste di probe e beacon		2018	2019		

Tabella 3.2: Tool per implementare canali coperti nel livello Data Link.

3.3.2 ICMP

Il protocollo ICMP è uno dei più utilizzati dalle implementazioni di covert channel in quanto offre un payload facilmente sfruttabile e poco analizzato. In Tabella 3.3 sono riassunti i tool analizzati di seguito.

Loki. In tutte le reti e sottoreti basate su TCP/IP è presente del traffico ICMP. L’Internet Control Message Protocol è un protocollo senza connessione utilizzato per trasmettere messaggi di errore e altre informazioni verso indirizzi unicast. Questo

²³PegasuLab, GhostTunnel, 2018, <https://github.com/PegasusLab/GhostTunnel>.

²⁴AmyangXYZ, GhostTunnel - GO, 2018, <https://github.com/AmyangXYZ/GhostTunnel-Go>.

protocollo ha un formato di pacchetto standard riconosciuto da ogni router IP e viene utilizzato universalmente per la gestione, il test e la misurazione della rete (infatti molto spesso il traffico ICMP viene considerato benigno dai firewall). Il pacchetto prevede che i primi 4 byte dell'intestazione siano uguali per ogni messaggio, mentre gli altri varino a seconda del tipo del messaggio. Esistono 15 tipi di messaggi ICMP tra cui troviamo il tipo 0 che indica un ECHO_REPLY (risposta) e il tipo 8 che rappresenta un ECHO (domanda). Nel traffico ICMP sono contenuti i messaggi di ping, cioè vengono inviate delle ECHO verso un host per verificare che sia effettivamente raggiungibile. Nei pacchetti ECHO c'è la possibilità di includere una sezione dati che spesso viene utilizzata per salvare le informazioni di temporizzazione per determinare i tempi di andata e ritorno, ma nonostante questo non c'è nessun controllo da parte di alcun dispositivo sul contenuto dei dati. Questa sezione quindi viene usata dal software Loki per iniettare i dati e implementare il canale nascosto [39, 40]. Il progetto in questione quindi sfrutta la sezione dati nei pacchetti ECHO e ECHO_REPLY inserendoci dati arbitrari. Loki può essere utilizzato come backdoor in un sistema fornendo un metodo nascosto per far eseguire i comandi, per esfiltrare dati o come metodo segreto di comunicazione utente e macchina o utente e utente. Tramite questo canale è possibile inserire segretamente dati all'interno di traffico legittimo.

icmptunnel. In rete è possibile trovare diversi strumenti che hanno un comportamento molto simile a icmptunnel²⁵. Infatti come è possibile intuire dal nome, permette di creare dei tunnel di comunicazione tramite il protocollo ICMP. In particolare è in grado di gestire un flusso IP all'interno dei pacchetti richiesta e risposta della funzione ping. A differenza degli altri strumenti disponibili, icmptunnel fornisce un protocollo più affidabile e un meccanismo di tunneling attraverso firewall stateful e NAT.

ICMP Backdoor. Nel novembre del 2000 il team CodeZero ha pubblicato la sua implementazione ICMP Backdoor nel hack-zine Confidence Remains High (CRH)²⁶. Il codice si compone di due parti, icmpd.c e icmp.c, rispettivamente un server e un client, caratterizzati in ogni loro parte da un errore di battitura forse per filtrare

²⁵jamesbarlow, icmptunnel, 2016, <https://github.com/jamesbarlow/icmptunnel>.

²⁶BiT, ICMP Backdoor, 1998, <http://web.textfiles.com/ezines/CRH/crh009.txt>.

banali di tentativi di riutilizzo del loro codice. A differenza del programma Loki vengono utilizzati solo pacchetti di risposta al ping, questi risultano essere molto brevi a causa della mancanza di padding e a volte mal categorizzati dai vari software: Snoop li vede come pacchetti IP sconosciuti mentre tcpdump li considera ICMP.

007Shell. Questo progetto è stato pubblicato dal team S0ftpr0ject che proveniva dall'underground dell'hacking italiano²⁷. Questo gruppo di ragazzi pubblicavano l'hack-zine aperiodica Butchered From Inside (BFI) con l'obiettivo di avvicinare gli utenti al mondo linux e per sensibilizzarli sulla protezione dei dati e sulla loro trasmissione. Nel numero 4 del dicembre 1998 veniva incluso questo strumento autodefinito "Covert Shell Tunneling", con ringraziamenti in particolare a Loki [39, 40] e al suo autore deamon9. Questa distribuzione è suddivisa in una libreria di tunneling riutilizzabile e un programma che funge sia da client e che da server. I dati vengono organizzati a multipli di 64 bit in modo da sembrare dei pacchetti di risposta al ping, dato che l'intero funzionamento si basa su questo tipo di pacchetti. Per accedere al socket ICMP sono necessari i permessi di root e la comunicazione avviene in chiaro.

VSTT. Sempre all'interno della cartella GitHub di Wendzel è possibile trovare un tool da lui scritto durante il suo percorso di studi, Very Strange Tunneling Tool (VSTT)²⁸. Questo semplice programma scritto in C permette di implementare comunicazioni TCP attraverso protocolli diversi come ICMP e POP3. Il codice C è diffuso con licenza GPL-3.0.

Ptunnel. Un altro protocollo molto utilizzato dalle implementazioni di canali coperti disponibili è ICMP. Nel programma Ptunnel²⁹ i pacchetti di ECHO_REQUEST e ECHO_REPLY, più conosciuti come richieste e risposte ping, vengono utilizzati per nascondere delle informazioni al fine di implementare una connessione TCP. Questo programma agisce come un proxy e può gestire socket e identificazione protetta, è sviluppato in C e diffuso con licenza BSD.

²⁷s0ftpr0ject, 007shell, 1998, <http://www.s0ftpj.org/en/site.html>.

²⁸Steffen Wendzel, VSTT - Very Strange Tunneling Tool, 2019, <https://github.com/cdpxe/NetworkCovertChannels/tree/master/vstt>.

²⁹Stødle, ptunnel, 2003, <https://github.com/f1vefour/ptunnel>.

ICMPTX. Altra proposta che sfrutta sempre il protocollo ICMP è ICMPTX³⁰. Questo tool, se eseguito con privilegi di root, permette di creare dei link di rete virtuali tra due pc encapsulando i dati nei pacchetti ICMP. Questo codice era incluso nei progetti Vtun e itunnel, il primo verrà descritto nella Sezione 3.3.4 e il secondo nella Sezione 3.3.6

Itun. Tra i vari strumenti per implementare covert channel attraverso il protocollo ICMP troviamo anche Itun³¹. Come altre proposte sopra riportate anche questa sfrutta i pacchetti ping per iniettare dati e aggirare le misure di sicurezza di una rete. È sviluppato in C e diffuso con licenza GPL-2.0 da Sergey Chvalyuk.

HANS. Ulteriore tool che sfrutta il protocollo ICMP per creare dei tunnel IPv4 è HANS³². Come molte altre proposte simili sfrutta le richieste e le risposte ping per iniettare dati e creare la comunicazione silente. Il codice sorgente è disponibile su GitHub, scritto in C e C++, ed è allegata a una licenza GPL-2.0. Si ispira allo strumento ICMPTX ma presenta delle caratteristiche aggiuntive: maggior affidabilità, meccanismo di login, client multipli e semplicità di configurazione. Inoltre è disponibile come client per macOS, Windows, iPhone, iPad mentre per Linux anche come server.

Nome	Licenza	Linguaggio	Descrizione	Rilascio	Ultima Modifica
Loki [39, 40]	GPL-3.0	C	Backdoor, esfiltrare dati e comunicazione segreta	1997	1997
icmptunnel	MIT	C	Comunicazione IP sopra ICMP (ping)	2016	2016
ICMP Backdoor	-	C	Comunicazione attraverso ICMP	1998	1998
007Shell	-	C	Covert shell nelle risposte ping	1998	1998
VSTT	GPL-3.0	C	Tunnel TCP su ICMP/POP3	2006	2006

³⁰ICMPTX, John Plaxco, 1998, https://codeberg.org/jakkarth/icmptx/src/branch/main/tun_dev.c.

³¹Sergey Chvalyuk, Itun, 2005, <https://github.com/grubberr/itun>.

³²friedrich, HANS, 2013, <https://github.com/friedrich/hans>.

Ptunnel	BSD	C	Tunnel TCP sopra ICMP	2003	2003
ICMPTX	GPL-3.0	C	Tunnel sopra ICMP	1998	1998
Itun	GPL-2.0	C	Tunnel sopra ICMP	2005	2005
HANS	GPL-3.0	C/C++	Ispirato a ICMPTX ma con miglioramenti	2013	2022

Tabella 3.3: Tool per implementare canali coperti nel protocollo ICMP.

3.3.3 IGMP

B0CK. Nel dicembre 1999 s0ftpr0ject ha presentato una variante di covert shell ICMP scritta interamente in italiano che si basa su IGMP, denominata B0CK [186]. L'autore afferma che questa rappresenta un miglioramento rispetto a Loki e 007Shell, descritti nelle Sezioni 3.3.2 e 3.3.2, in quanto sempre più misure di sicurezza vanno ad analizzare il traffico ICMP e le risposte a richieste ping non effettuate sono molto sospette. Viene quindi scelto il protocollo IGMP in quanto sottoposto a un numero minore di controlli, ogni riga di output, dal lato client e server, viene suddivisa in sezione di 4 byte e il codice ASCII corrispondente a ciascun carattere viene utilizzato come indirizzo di origine falsificato. Un difetto di questo meccanismo è che va a creare pacchetti IP, con il campo protocol impostato su IGMP, imbottiti di zeri che li rendono classificabili in pacchetti IGMP di tipo 0, cioè obsoleti. Il tool è scritto in C, è diffuso con licenza GPL-2.0 e non risulta essere stato aggiornato dalla sua data di rilascio.

3.3.4 IP

Nonostante per il protocollo IP fossero presenti molte proposte in letteratura, nella pratica queste si sono rivelate essere molto minori. In Tabella 3.4 sono riportate le informazioni principali delle 4 implementazioni analizzate di seguito.

Covert_tcp. Il programma `covert_tcp` è una semplice utility sviluppata solo per sistemi Linux ed è stata testata con kernel versione 2.0³³. Questo software utilizza dei socket grezzi per costruire pacchetti contraffatti e incapsulare i dati estratti da un file fornito a riga di comando. La velocità di trasmissione è ridotta a un pacchetto al secondo per garantire che i pacchetti arrivino nell'ordine giusto dato che le caratteristiche di affidabilità del TCP/IP non possono essere utilizzate, nella pratica si tratta di UDP. Il programma inserisce i dati nei campi ID IP, Initial Sequence Number TCP e Sequence Number TCP, senza sfruttarne l'intera lunghezza, perché sono campi obbligatori e quindi hanno meno probabilità di essere alterati/eliminati dai meccanismi di filtraggio e riassembaggio. Alla base del funzionamento c'è una codifica dei valori ASCII 0-255 nei campi sopra riportati che fornisce la possibilità quindi di inviare dati tra host tramite pacchetti che sembrano essere richieste di connessione iniziali, flussi di dati stabiliti o altri passaggi intermedi. Come riportato quindi sono disponibili tre metodi di iniezione di dati, uno per ogni campo utilizzato, i quali senza un inserimento di una componente casuale rappresenterebbero dei cifrari a sostituzione con tutti i vari rischi che ne comporta.

IPv6teal. Come affrontato nella sezione precedente, sono state proposte diverse tecniche sulla possibilità di esfiltrare dati tramite il protocollo IPv6, però poi nella pratica ne sono state implementate un numero minore. Tra queste c'è `IPv6teal`³⁴, un tool scritto totalmente in Python che sfrutta il campo Flow Label per raggiungere l'obiettivo, dato che la modifica del valore di questo campo non compromette la trasmissione del pacchetto. Il tool si compone di due moduli, `exfiltrate.py` e `receiver.py`, e trasmette 20 bit per ogni segmento. Naturalmente entrambe le macchine devono supportare e avere un indirizzo IPv6 e la libreria Scapy installata, la quale è stata descritta nella Sezione 3.2.3.

NETsteg. Tra le tecniche descritte nel capitolo precedente c'è quella riguardante il campo TTL delle intestazioni IP. `NETsteg`³⁵ è un tool per effettuare un trasferimento occulto di informazioni tramite i bit più significativi (MSB) del campo TTL. Il pro-

³³zaheercena, `covert_tcp`, 2016, <https://github.com/zaheercena/Covert-TCP-IP-Protocol>.

³⁴christophetd, `IPv6teal`, 2019, <https://github.com/christophetd/IPv6teal>.

³⁵chrispetrou, `NETsteg`, 2019, <https://github.com/chrispetrou/NETsteg>.

getto si compone di due moduli, per l'invio e la ricezione di dati, scritti in Python e diffusi con licenza GPL-3.0, quello dedicato alla ricezione produce un file .pcap e restituisce i dati già decifrati.

Vtun. Per gli ambienti Linux, FreeBSD e Solaris è disponibile il tool Vtun³⁶. Dato che comprendeva il progetto ICMPXT, descritto nella Sezione 3.3.2, è facile intuire che la finalità di utilizzo di quest'ultimo sia molto simile: fornisce il metodo per la creazione di tunnel virtuali su reti TCP/IP. A differenza di altre soluzioni permette anche di modellare, comprimere e crittografare il traffico nascosto che viene spedito su questi tunnel. Vtun è altamente configurabile e non richiede la modifica di alcuna parte del kernel.

Nome	Licenza	Linguaggio	Descrizione	Rilascio	Ultima Modifica
covert_tcp	MIT	C	Invia dati inserendoli in campi TCP/IP	3	2016
IPv6teal	-	Python	Covert channel tramite campo Flow Label di IPv6	2019	2019
NETsteg	GPL-3.0	Python	Esfiltrazione dati tramite campo TTL IP	2019	2019
Vtun	Copyright	C	Tunnel sopra TCP/IP	1999	2016

Tabella 3.4: Tool per implementare canali coperti nel protocollo IP.

3.3.5 TCP

Anche per il TCP il numero di implementazioni non segue quello delle proposte, infatti di seguito ne vengono riportate solo 3. In Tabella 3.5 il riassunto delle caratteristiche principali dei tool analizzati.

³⁶Maxim Krasnyansky, Vtun, 1999, <https://vtun.sourceforge.net>.

BO2K. Back Orifice 2000, spesso abbreviato in BO2K, è un programma per computer progettato per l'amministrazione remota dei sistemi³⁷. Questo software è stato scritto da un membro del gruppo hacker Cult of the Dead Cow (cDc) ed è stato presentato il 10 luglio 1999 al DEF CON 7. Questa versione rappresenta un importante aggiornamento rispetto al suo predecessore Back Orifice, perché ora supporta anche i SO Windows NT, 2000, XP oltre al 95 e 98. La sua ultima versione stabile rilasciata risale al 2007 ed essendo diffuso come software libero è stato possibile portare le sue funzionalità anche su sistemi Linux. Date alcune sue caratteristiche come l'associazione con cDc, la presentazione al DEF CON 7 e l'installazione e il funzionamento silenzioso, l'hanno portato ad essere considerato come malware da parte di tutti i vari antivirus. In particolare il programma si compone di due moduli, server e client, dove tramite il secondo si andava a controllare il primo, che era in esecuzione presso una macchina remota. Le operazioni che permette sono molte e variano tra di loro: si va dall'esecuzione di qualsiasi applicazione al riavvio, dal blocco al recupero della password dallo screen saver. BO2K utilizza i protocolli di rete TCP e UDP e viene eseguito sulla porta 31337, e la procedura di installazione prevede che sia prima eseguito il modulo server che è rappresentato da un file di appena 122KB. Nonostante le controversie legate a questo software viene considerato da tutti una pietra miliare nella sicurezza informatica e un eccellente esempio di sviluppo di software.

AckCmd. Un ulteriore strumento che permette di implementare una covert shell è AckCmd, scritto da Arne Vidstrom per i sistemi Windows 2000 [188]. Nonostante anche le shell tradizionali remote comunichino tramite TCP, questo tool utilizza solo le segmenti TCP ACK, anziché i TCP SYN tipicamente analizzati. Utilizza la porta 80 sul client e la 1054 sul server per aumentare la probabilità di attraversamento del firewall, tuttavia i pacchetti non sono formattati correttamente per l'HTTP, il programma è visibile nell'elenco delle attività e i dati viaggiano in chiaro.

Syn-file. Syn-file³⁸ è un tool disponibile su GitHub interamente scritto in C, per l'esfiltrazione dei dati. Come illustrato nel capitolo precedente è possibile utilizzare

³⁷Cult of the Dead Cow, BO2K ORIFICE 2000, 2007, <http://www.bo2k.com/category/back-orifice/>.

³⁸defensahacker, syn-file, 2017, <https://github.com/defensahacker/syn-file>.

l'intestazione TCP infatti questa tecnica sfrutta proprio i numeri di sequenza sincronizzati dal campo TCP SYN. Si compone da un modulo da eseguire sulla macchina vittima e uno che monitora il traffico di rete per decodificare le informazioni.

Nome	Licenza	Linguaggio	Descrizione	Rilascio	Ultima Modifica
BO2K ORIFICE 2000	GPL-2.0	C++	Amministrazione remota su Windows	2007	2007
AckCmd [188]	-	.exe	Covert shell tramite TCP ACK	2000	2000
syn-file	-	C	Esfiltrazione su campo TCP SYN	2017	2017

Tabella 3.5: Tool per implementare canali coperti nel protocollo TCP.

3.3.6 UDP

deamonshell. Questo tool è stato presentato nella prima pubblicazione di Unix Hacking Tools nel 1999 da Van Hauser del THC³⁹ e da allora non sono stati rilasciati altri aggiornamenti. Nell'archivio erano presenti due versioni scritte in Perl. Il programma in questione si mette in ascolto su una porta arbitraria, effettua l'autenticazione tramite password e lancia una shell Unix standard. La connessione al client viene effettuata tramite strumenti come netcat⁴⁰ e le informazioni trasmesse all'interno della sezione dati dei pacchetti TCP e UDP, a seconda della versione utilizzata. A differenza del programma Loki, descritto nella Sezione 3.3.2, non richiede la compromissione di root, inoltre la versione UDP è in grado di sfruttare qualsiasi canale UDP legittimo garantendo maggiore clandestinità. Il tutto viene fornito in 44 righe di codice Perl, che al momento della scrittura di questa tesi non è stato possibile recuperare. Caratteristica negativa del tool è che le transazioni di shell avvengono in

³⁹THC, deamonshell, 1999, <https://www.thc.org>.

⁴⁰<https://sectools.org/tool/netcat/>.

chiaro, quindi è molto facile rilevarle. Nell'anno successivo è stato rilasciato lo strumento Itunnel, da parte del gruppo austriaco *teso*, che ne portava dei miglioramenti, ma su questo non è stato possibile reperire informazioni.

3.3.7 HTTP

In controtendenza agli protocolli affrontati, le implementazioni che riguardano HTTP sono molte, probabilmente in numero maggiore alle proposte registrate in letteratura. Quello che si è registrato che alcuni tool non trattano di steganografia di rete pura, ma effettuano un tunnel di altri protocolli all'interno di traffico HTTP. In Tabella 3.6 il riassunto degli strumenti analizzati di seguito.

Rwwwshell. Van Hauser ha pubblicato l'articolo "Placing Backdoors Through Firewalls" dove illustra delle tecniche per posizionare delle backdoor per la fase post-exploitation e presenta un proof-of-concept scritto in Perl [68]. Questo prototipo chiamato rwwwshell è in grado di funzionare con qualsiasi firewall che permette la navigazione in internet perché va a creare un processo figlio sulla macchina vittima ogni giorno a una certa ora che invierà un segnale al server di proprietà dell'hacker. Le richieste avranno la forma di una normale comunicazione web ma all'interno di questa saranno inseriti i comandi da eseguire e vari risultati, perché nel frattempo il processo figlio eseguirà una shell locale sulla macchina. Il software è composto da un unico script scritto in Perl.

Http tunnel. Il programma pubblicato da larsbrinkhoff su GitHub nel 2010, chiamato htptunnel⁴¹, permette di creare un percorso virtuale bidirezionale, basato sulle richieste HTTP, lungo il quale due macchine possono scambiarsi dei dati. Questa implementazione garantisce il funzionamento anche in presenza di un proxy e quindi tramite essa è possibile magari connettersi tramite telnet⁴² o PPP⁴³ a computer fuori dalla rete aziendale. L'intero programma è scritto in C è diffuso con licenza GPL-2.0.

⁴¹larsbrinkhoff, htptunnel, 2010, <https://github.com/larsbrinkhoff/htptunnel>.

⁴²<https://www.rfc-editor.org/rfc/rfc854>.

⁴³<https://www.rfc-editor.org/rfc/rfc1661>.

Cirripede. Nel 2011 Houmansadr et al [77] hanno presentato Cirripede, il loro strumento contro la censura. Questo tool viene utilizzato per la comunicazione non osservabile, intercetta le connessioni dei client verso siti dall’aspetto innocente e le reindirizza verso la vera destinazione richiesta dal client. È stato progettato per essere installato all’interno degli ISP dato che le sue caratteristiche comportano un minimo sovraccarico e non interrompono il traffico esistente.

Wondjina. Questo strumento è stato sviluppato da Van Horenbeeck nel 2015 come proof-of-concept⁴⁴ che il tunneling HTTP su header Etags/If-None-Match fosse possibile. Doveva effettuare un penetration test in cui il team doveva spostare un file all’esterno di una rete filtrata e il tunneling DNS non era possibile. L’unica soluzione percorribile era quella dei proxy HTTP perché potevano effettuare delle ricerche esterne. Non era possibile procedere con il classico tunneling HTTP perché le stringhe di richiesta erano registrate, mentre le intestazioni no.

corkscrew. Pat Padgett è l’autore dello strumento corkscrew, uno strumento per effettuare tunneling di SSH attraverso i proxy HTTP⁴⁵. Il tool è stato compilato per diversi sistemi operativi (Solaris, Linux, Win32, macOS, etc) e fatto funzionare con diversi proxy HTTP: Gauntlet⁴⁶, CacheFlow⁴⁷, JunkBuster⁴⁸ e Apache mod_proxy⁴⁹. Viene distribuito tramite GitHub ed è scritto in C. Nella pagina di presentazione vengono riportati tutti i passaggi della procedura di compilazione e installazione, compreso come implementare la funzione di autenticazione.

Httunnel. Le soluzioni che prevedono lo sfruttamento del protocollo HTTP per trasmettere dati in modo silente sulla rete sono diverse e tra queste troviamo HT-Tunnel⁵⁰. Questo strumento è stato sviluppato da Patrick LeBoutillier nel 2005 e viene caricato nel server web Apache attraverso il modulo *mod_perl*. Questo tool

⁴⁴Marteen Van Horenbeeck, wondjina, 2015, <https://github.com/maartenvhb/wondjina>.

⁴⁵Pat Padgett, corkscrew, 2003, <https://github.com/patpadgett/corkscrew>.

⁴⁶https://techpubs.jurassic.nl/manuals/0620/admin/Gauntlet_AG/sgi_html/ch01.html.

⁴⁷<https://www.giac.org/paper/gsna/85/auditing-cacheflow-proxy-solution-auditors-perspective/103112>.

⁴⁸<http://internet.junkbuster.com>.

⁴⁹https://httpd.apache.org/docs/2.4/mod/mod_proxy.html.

⁵⁰Patrick LeBoutillier, httunnel, 2005, <https://metacpan.org/pod/Apache::HTTunnel>.

funziona solo su piattaforme di tipo Unix, in particolare quelle che supportano le API sendmsg e recvmsg. Tramite Httunnel è possibile quindi trasportare qualsiasi protocollo (TCP o UDP) su HTTP.

Infranet. Una delle cause principali che porta allo sviluppo di tecniche di network steganography è senza dubbio l’elusione della censura, e tra le proposte riportiamo anche Infranet [51]. Questo sistema consente di recuperare dati bloccati tramite server Web cooperanti in nella rete internet globale. Come altre soluzioni anche Infranet basa il suo tunnel di comunicazione sulla modulazione del traffico assomigli a navigazione web innocua. I client inviano messaggi occulti ai server associando un significato alla sequenza delle richieste HTTP, mentre i server restituiscono i dati censurati nelle immagini semplici e pure tecniche steganografiche.

Firepass. Firepass⁵¹ è in grado di encapsulare flussi di dati all’interno di richieste HTTP POST per aggirare le restrizioni dei firewall. Quindi è possibile creare dei flussi nascosti TCP o UDP attraverso i due moduli, client e server, scritti in Perl che compongono il programma. La parte server viene eseguita su server httpd esterno dalla rete come programma CGI, al quale il client invierà direttamente le richieste oppure attraverso un server proxy. Per ovviare al problema della tante richieste HTTP in poco tempo, che potrebbero allertare i sistemi di monitoraggio, è possibile configurare dei ritardi durante le comunicazioni.

Hcovert. Hcovert⁵² è uno strumento pubblicato nel 2005, con licenza GPL-2.0, che fornisce servizi di steganografia di rete. In particolare è in grado di implementare un canale coperto attraverso le richieste GET del protcollo HTTP. Per inviare il messaggio verso il server web si usano le richieste GET mentre per recuperare il messaggio devono essere analizzati i registri. Questo tool garantisce la possibilità di inviare i messaggi in entrambe le direzioni e quindi di implementare una comunicazione bidirezionale.

⁵¹Alex Dyatlov, Firepass, 2003, <https://packetstormsecurity.com/files/download/31233/firepass-1.0.2a.tar.gz>.

⁵²druid-cau, hcovert - HTTP Payload Covert Channel, 2005, <https://sourceforge.net/projects/hcovert/>.

StegoProxy. StegoProxy⁵³ è una libreria Java e un'applicazione per creare connessioni proxy staganografate su Internet. Sulla macchina che invia i dati viene eseguito il modulo ProxyClient mentre ProxyServer sul server esterno e connesso a un server proxy reale come Squid⁵⁴. Quindi le informazioni dal browser, o qualsiasi altra applicazione, arrivano al client che le spedisce al server di Stegoproxy tramite un canale crittografato, infine il server, dopo averle decodificate, le inoltrerà al proxy reale. Il programma è disponibile su GitHub con licenza BSD-2 clause.

Tunna. Tunna⁵⁵ è uno strumento che permette di effettuare tunneling di comunicazioni TCP tramite protocollo HTTP. L'obiettivo dello sviluppatore è quello di avere una webshell su un computer all'interno di una rete protetta da firewall dove solo il traffico internet è permesso. Il programma si compone di tre moduli: il webserver e il proxy scritti in Python e la webshell fornita in formato .php, .aspx e .jsp. Sarà compito del proxy ricevere le richieste/risposte e impacchettarle nel corpo delle richieste POST HTTP.

NativePayload HTTP. Damon Mohammadbagher ha pubblicato su GitHub due script per effettuare esfiltrazione dati tramite richieste HTTP nominati NativePayload_HTTP⁵⁶. La sua tecnica si divide in due moduli, server e client, il primo è scritto totalmente in Shell mentre il secondo in C# e Shell. Presenta in modo dettagliato, supportate anche da video e screenshot, tre tecniche differenti per trasferire le informazioni: tramite variabile ID e Values, tramite il campo di intestazione HTTP "Refer" o tramite i Cookie.

Chisel. Tra i più recenti tool nella steganografia di rete c'è Chisel⁵⁷, questo programma implementa un tunnel TCP/UDP veloce trasportato tramite HTTP. A differenza di altre soluzioni è formato da un singolo eseguibile che include sia il client che il server, e inoltre fornisce una protezione basata su SSH alla comunicazione. Può

⁵³LabunskyA, StegoProxy, 2016, <https://github.com/LabunskyA/StegoProxy>.

⁵⁴<http://www.squid-cache.org>.

⁵⁵Nikos Vassakis, Tunna, 2014, <https://github.com/SECFORCE/Tunna>.

⁵⁶Damon Mohammadbagher, NativePayload_HTTP, 2019, https://github.com/DamonMohammadbagher/NativePayload_HTTP.

⁵⁷Jaime Pillora, Chisel, 2020, <https://github.com/jpillora/chisel>.

essere utilizzato per attraversare i firewall ma anche per fornire un endpoint sicuro alla rete. È scritto totalmente in Go e diffuso con licenza MIT.

Nome	Licenza	Linguaggio	Descrizione	Rilascio	Ultima Modifica
rwwwshell [68]	-	Perl	Cover shell tramite traffico internet	1999	1999
httptunnel	GPL-2.0	C	Tunnel tramite richieste HTTP	2010	2022
Cirripede [77]	-	-	Strumento anti-censura	2011	2011
Wondjina	GPL-3.0	Perl	Tunneling su intestazioni HTTP	2015	2015
corkscrew	GPL-2.0	C	Tunneling delle connessioni TCP attraverso i proxy HTTP	2003	2018
HTTunnel	GPL-2.0	Perl	Tunneling sopra HTTP	2005	2007
Infranet [51]	-	C++	Modulo apache per elusione alla censura	2002	2002
Firepass	GPL-2.0	Perl	Tunnel TCP/UDP nelle richieste HTTP POST	2003	2003
hcovert	GPL-2.0	C	Covert channel tramite richieste HTTP GET	2005	2014
StegoProxy	BSD-2 clause	Java	Connessioni proxy steganografate in Internet	2016	2021
Tunna	GPL-3.0	Python/ PHP/...	Tunneling TCP su HTTP	2014	2021
NativePayload HTTP	-	Shell/C#	Script per esfiltrazione dati tramite HTTP	2019	2019
Chisel	MIT	Go	Tunnel TCP/UDP tramite HTTP con SSH	2020	2022

Tabella 3.6: Tool per implementare canali coperti nel protocollo HTTP.

3.3.8 DNS

I tool che utilizzano il protocollo DNS per implementare un canale coperto sono molti perché, in analogia al protocollo ICMP, è presente un payload facilmente sfruttabile e poco analizzato. In Tabella 3.7 sono riportate le caratteristiche fondamentali degli strumenti analizzati di seguito.

Dns tunnel. Inizialmente questo strumento doveva essere presentato sul magazine Phrack⁵⁸, ma siccome gli autori non ricevettero risposta fu pubblicato su un forum⁵⁹. Questo tool si basa sulla tecnica del tunneling DNS ma attraverso bastion host, cioè un host che isola la rete interna dalla rete pubblica creando uno scudo che la protegge. L'autore specifica che fu scritto per effettuare test sulla rete di casa ma anche che potrebbe essere usato contro dispositivi di cui non si ha il controllo. Per il corretto funzionamento è necessario disporre dei privilegi di root su un host esterno con un indirizzo IP statico e un dominio da delegare al server. I client esegue delle ricerche DNS per un host del dominio delegato, se il server vuole connettersi risponde con un indirizzo IP "chiave". Il client avvia quindi una shell e invia l'output sotto forma di query DNS al server. Il server legge la sequenza dei caratteri e restituisce le informazioni al client sotto forma di indirizzi IP di risposta alle query DNS.

NSTX. NSTX è un tool sviluppato per Linux in grado di creare un tunnel IP bidirezionale attraverso richieste DNS valide⁶⁰. Il flusso di informazioni è ottimizzato dal client al server, ma presenta buone performance anche nel verso opposto. Il lato negativo di questo tool è che è stato presentato negli '90, molti anni fa, e non risulta sufficientemente aggiornato.

Iodine. Uno degli strumenti più conosciuti e supportati nell'insieme dei tunnel su protocollo DNS è iodine⁶¹. Questo strumento permette di eseguire un tunnel di dati IPv4 attraverso un server DNS, pensato per le situazioni dove il traffico internet è protetto da firewall ma le query DNS sono consentite. Funziona su Linux, macOS,

⁵⁸<http://www.phrack.org>.

⁵⁹Oskar Pearson, Dns tunnel - through bastion hosts, 1998, <https://seclists.org/bugtraq/1998/Apr/79>.

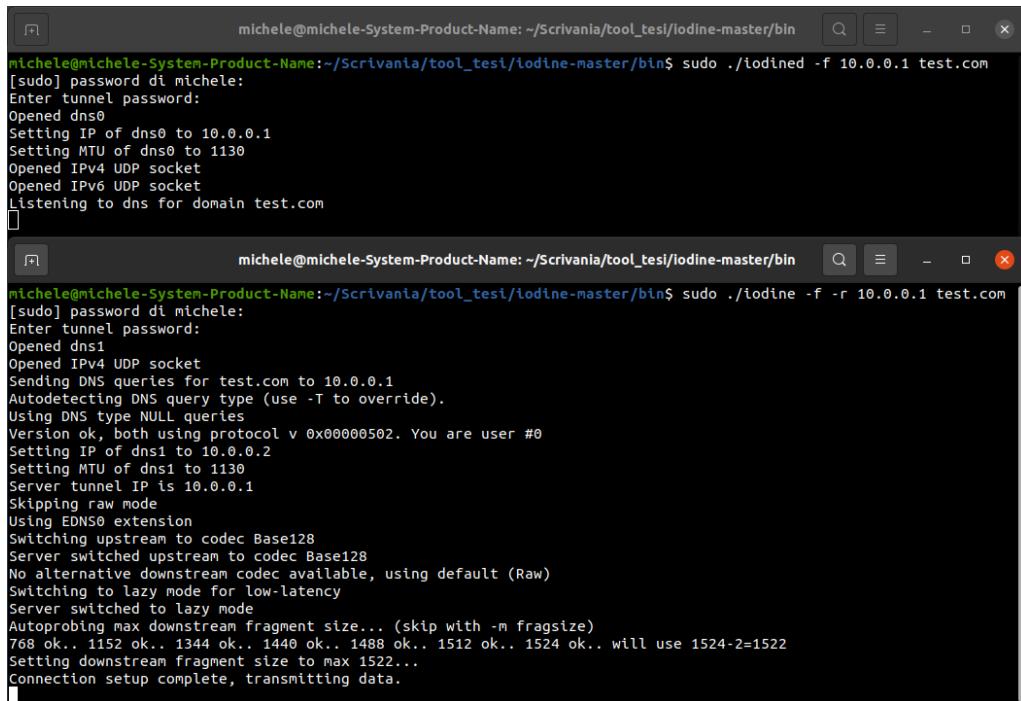
⁶⁰savannah.nongnu.org, NSTX, 2002, <http://savannah.nongnu.org/projects/nstx/>.

⁶¹Kryo, iodine, 2010, <http://code.kryo.se/iodine/>.

FreeBSD, NetBSD, OpenBSD e Windows e necessita di un dispositivo TUN/TAP. Quest'ultimi sono driver che permettono la creazione di periferiche di rete virtuali, i pacchetti spediti da e verso questi dispositivi sono gestiti da programmi software, a differenza delle comuni periferiche che sono controllate direttamente dalla scheda di rete. La larghezza di banda fornita è asimmetrica con upstream limitato e il downstream fino a 1Mb/s. Rispetto alle altre implementazioni iodine fornisce:

- prestazioni maggiori perché utilizza il tipo NULL e ogni risposta può contenere oltre 1KB di payload compressi,
- portabilità perché funziona su diversi SO,
- sicurezza perché utilizza un meccanismo di login challenge-response protetta da un hash MD5,
- più semplicità perché molte impostazioni vengono gestite in automatico.

In Figura 3.11 è possibile osservare l'output di un test.



The screenshot shows two terminal windows side-by-side. Both windows have a dark background and white text. The top window shows the initial configuration steps for a tunnel to 'test.com': setting the IP of 'dns0' to 10.0.0.1, opening UDP sockets for IPv4 and IPv6, and starting a DNS listener. The bottom window shows the tool entering raw mode, sending DNS queries to the server, and establishing a connection setup. It also mentions switching upstream to 'codec Base128' and setting downstream fragment sizes.

```
michele@michele-System-Product-Name:~/Scrivania/tool_tesi/iodine-master/bin$ sudo ./iodined -f 10.0.0.1 test.com
[sudo] password di michele:
Enter tunnel password:
Opened dns0
Setting IP of dns0 to 10.0.0.1
Setting MTU of dns0 to 1130
Opened IPv4 UDP socket
Opened IPv6 UDP socket
Listening to dns for domain test.com

michele@michele-System-Product-Name:~/Scrivania/tool_tesi/iodine-master/bin$ sudo ./iodine -f -r 10.0.0.1 test.com
[sudo] password di michele:
Enter tunnel password:
Opened dns1
Opened IPv4 UDP socket
Sending DNS queries for test.com to 10.0.0.1
Autodetecting DNS query type (use -T to override).
Using DNS type NULL queries
Version ok, both using protocol v 0x00000502. You are user #0
Setting IP of dns1 to 10.0.0.2
Setting MTU of dns1 to 1130
Server tunnel IP is 10.0.0.1
Skipping raw mode
Using EDNS0 extension
Switching upstream to codec Base128
Server switched upstream to codec Base128
No alternative downstream codec available, using default (Raw)
Switching to lazy mode for low-latency
Server switched to lazy mode
Autoprobing max downstream fragment size... (skip with -m fragsize)
768 ok.. 1152 ok.. 1344 ok.. 1440 ok.. 1488 ok.. 1512 ok.. 1524 ok.. will use 1524-2=1522
Setting downstream fragment size to max 1522...
Connection setup complete, transmitting data.
```

Figura 3.11: Interfaccia del tool Iodine.

DNScat. Altro strumento noto e ben sviluppato è DNSCat⁶². La seconda versione è presente su GitHub e permette di creare un canale criptato di comando e controllo (C&C) attraverso il protocollo DNS. Come altri strumenti si compone della parte client e di quella server. Gli autori hanno deciso di utilizzare il linguaggio C e utilizzare meno dipendenze possibili affinché sia funzionante ovunque. Il client invia delle richieste DNS al server locale che le inoltrerà al server autoritario di cui, presumibilmente, si ha il controllo. La parte server è sviluppata in Ruby e quando si esegue, come il client, vanno specificati i domini che deve ascoltare. Questo strumento, a differenza di altri, è stato nativamente sviluppato proprio per operazioni di comando e controllo, ma nonostante ciò permette comunque di svolgere altre operazioni dato che può eseguire il tunnel di qualsiasi dato. In Figura 3.12 l'output dei processi di client e server al loro avvio.

```
michele@michele-System-Product-Name:~/Scrivania/tool_tesi/dnscat2-master/server$ sudo ruby ./dnscat2.rb
New window created: 0
New window created: crypto-debug
Welcome to dnscat2! Some documentation may be out of date.

michele@michele-System-Product-Name:~/Scrivania/tool_tesi/dnscat2-master/client$ ./dnscat2
Starting DNS driver without a domain! This will only work if you
are directly connecting to the dnscat2 server.

You'll need to use --dns server=<server> if you aren't.

** WARNING!
*
* It looks like you're running dnscat2 with the system DNS server,
* and no domain name!
* That's cool, I'm not going to stop you, but the odds are really,
* really high that this won't work. You either need to provide a
* domain to use DNS resolution (requires an authoritative server):
*
* dnscat mydomain.com
*
* Or you have to provide a server to connect directly to:
*
* dnscat --dns=server=1.2.3.4,port=53
*
* I'm going to let this keep running, but once again, this likely
* isn't what you want!
*
** WARNING!

Creating DNS driver:
domain = (null)
host   = 0.0.0.0
port   = 53
type   = TXT,CNAME,MX
server = 127.0.0.53
```

Figura 3.12: Interfaccia del tool DNScat.

⁶²T. Pietraszek, DNSCat, 2004, <http://tadek.pietraszek.org/projects/DNSCat/>.

OzymanDNS. Nel 2004 Dan Kaminsky ha pubblicato OzymanDNS, un tool che permette di effettuare tunnel di traffico tramite il protocollo DNS⁶³. L’argomento non era nuovo ma nel corso degli anni questa proposta ha rappresentato le fondamenta sulle quali si sono basate molte altre pubblicazioni. Il programma è suddiviso in 4 file scritti in Perl, due dei quali servono per il caricamento/scaricamento dei file utilizzando il DNS. La parte client è un semplice script che codifica e trasferisce tutto ciò che riceve su STDIN alla destinazione tramite richieste DNS, mentre le risposte le scrive su STDOUT. Singolarmente questo modulo non è molto utile ma è stato pensato per essere usato insieme a SSH.

Dns2tcp. Tra gli strumenti che sfruttano DNS per implementare un canale coperto, c’è dns2tcp⁶⁴. L’idea che c’è dietro è la stessa di altre proposte sopra citate, sfruttare le richieste DNS per incapsulare una sessione TCP. Questo incapsulamento genera dei pacchetti più piccoli rispetto agli altri IP-over-DNS migliorando il throughput. Caratteristica che differenzia questa proposta è che il client non richiede i privilegi di root per essere eseguito.

TUNS. Tuns, un semplice tunnel IP su DNS, è un prototipo di implementazione che viene fornito nell’interesse della riproducibilità dei risultati scientifici [138], in modo che le valutazioni effettuate possano essere riprodotte da altri. La sue caratteristiche principali sono semplicità e efficienza dato che è stato ottimizzato per una bassa latenza. È stato sviluppato in Ruby e la parte client necessita dei privilegi di root per essere eseguita.

Heyoka. Tra le varie proposte riguardanti il DNS c’è Heyoka⁶⁵, un proof-of-concept che permette l’esfiltrazione dati attraverso le richieste DNS. Il suo obiettivo è quello di garantire alte prestazioni e una buona segretezza, il progetto non è più in sviluppo ma è disponibile con licenza GPL-2.0. Lo strumento è scritto interamente in C è quindi funziona in modo nativo su ogni macchina senza bisogno di interpreti installati, inoltre è in grado di falsificare l’indirizzo di origine rendendo più difficile individuare il punto finale del tunnel.

⁶³D. Kaminsky, OzymanDNS, 2004, <http://dankaminsky.com/2004/07/29/51/>.

⁶⁴O. Dembour, C. Collignon, DNS2TCP, 2008, <http://hsc.fr/ressources/outils/dns2tcp/>.

⁶⁵nico icesurfe, Heyoka, 2009, <https://heyoka.sourceforge.net>.

Netcross. Sempre su sourceforge.net è disponibile il tool Netcross⁶⁶ che fornisce una possibilità di implementare un canale coperto IP all'interno delle richieste/risposte DNS. L'implementazione viene descritta come "piuttosto instabile" e destinata a scopi di ricerca. Questo tool è dotato dei moduli di trasporto: TUN/TAP su DNS, reindirizzamento TCP su DNS e proxy HTTP su DNS. È stato pubblicato nel 2006, scritto in C++ e diffuso con licenza GPL-2.0.

DNSExfiltrator. Un altro strumento nato con lo scopo di effettuare test per l'e-sfiltrazione di dati è DNSExfiltrator⁶⁷. Come la maggior parte dei tool basati sul protocollo DNS anche questo è basato sulle richieste/risposte e si compone di due moduli: server e client. Il lato server è composto da un unico script Python che agisce come serve DNS e riceverà i dati esfiltrati, mentre per la parte client (lato vittima) sono presenti tre varianti. È disponibile come uno script C# che può essere compilato con csc.exe per fornire un eseguibile gestito da Windows, uno script PowerShell e uno script JScript. Tutti forniscono le stesse funzionalità e per far funzionare il tutto è necessario possedere un nome di dominio e impostare il record DNS (NS) per quel dominio in modo che punti al server dove viene eseguito il nostro file.

Coyote. Coyote⁶⁸ è un programma stand-alone scritto in C# per la fase di post exploitation per mantenere l'accesso alla macchina Windows compromessa durante gli interventi dei red team. Bypassa la whitelist utilizzando InstallUyil.exe (MITRE ATT&CK ID:T1218.004⁶⁹) e successivamente recupera i comandi crittografati tramite un tunnel DNS (MITRE ATT&CK ID:T1071.004⁷⁰). Questo è un esempio di come queste tecniche possano essere impiegate per diversi obiettivi e in operazioni molto più complesse.

PacketWhisper. Tramite PacketWhisper⁷¹ è possibile trasferire furtivamente dei dati tramite richieste DNS e steganografia su testo, il tutto senza avere, a differenza

⁶⁶primiano, netcross, 2006, <https://sourceforge.net/projects/netcross/>.

⁶⁷Arno0x, DNSExfiltrator, 2017, <https://github.com/Arno0x/DNSExfiltrator>.

⁶⁸TartarusLabs, Coyote, 2022, <https://github.com/TartarusLabs/Coyote>.

⁶⁹<https://attack.mitre.org/techniques/T1218/004/>.

⁷⁰<https://attack.mitre.org/techniques/T1071/004/>.

⁷¹Joe Gervais, PacketWhisper, 2018, <https://github.com/TryCatchHCF/PacketWhisper>.

di altre proposte, server di nomi o domini controllati. Questo tool converte qualsiasi tipo di file in un elenco di nomi di dominio qualificati tramite Cloakify⁷², un insieme di strumenti per la steganografia testuale. Le richieste DNS saranno rivolte proprio a questi nomi di indirizzo ma tramite questa tecnica le risposte non vengono prese in considerazione, infatti per estrarre i dati basterà catturare il traffico di rete e passarlo a PacketWhisper, in automatico analizzerà le richieste DNS effettuate e restituirà il file esfiltrato.

Nome	Licenza	Linguaggio	Descrizione	Rilascio	Ultima Modifica
DNS Tunnel NSTX	- GPL-2.0	Perl C	DNS Tunneling Tunnel bidirezionale su DNS per sistemi Linux	1998 2002	1998 2002
iodine	ISC	C	Tunnel sopra DNS	2010	2022
DNScat	BSD 3-clause	Ruby	Tunnel crittografato su DNS	2004	2022
OzymanDNS	-	Perl	Tunnel sopra DNS	2004	2004
dns2tcp	GPL-2.0	C	Tunnel sopra DNS	2008	2017
TUNS [138]	GPL-3.0	Ruby	Tunnel sopra DNS	2009	2009
Heyoka	GPL-2.0	C	Esfiltrazione dati tramite richieste DNS	2009	2009
netcross	GPL-2.0	C++	Covert channel su richieste/-risposte DNS	2006	2015
DNSExfiltrator	-	Python/ C#/ JScript	Covert channel su richieste/-risposte DNS	2017	2018
Coyote	GPL-3.0	C#	Tunnel DNS per fase di post-exploitation	2022	2022
PacketWhisper	MIT	Python	Esfiltrazione dati tramite DNS e Cloakify	2018	2018

Tabella 3.7: Tool per implementare canali coperti nel protocollo DNS.

⁷²TryCatchHCF, Cloakify, 2014, <https://github.com/TryCatchHCF/Cloakify>.

3.3.9 VoIP

Questa particolare tecnologia presenta grandi potenzialità per l'implementazione di canali coperti, come dimostrano le diverse proposte analizzate nella Sezione 2.11.4. Tuttavia, dato il numero esiguo di tool, questa soluzione nella pratica non sembra essere particolarmente apprezzata. In Tabella 3.8 il riassunto delle 4 proposte analizzate.

Skypemorph, Freewave e CensorSpoof. Oltre ai classici protocolli di comunicazione a livello 3 e 4, sono state presentate delle soluzioni che sfruttano tecnologie che operano a livelli più alti, come il VoIP. Tra i strumenti più famosi che si basano sul VoIP per trasmettere i dati in modo silente ci sono SkypeMorph, FreeWave e CensorSpoof. Il primo cerca di risolvere il problema dell'inosservabilità dei ponti Tor dato che i grandi censori sono in grado di utilizzare la Deep Packet Inspection per rilevare le connessioni TLS effettuate dai ponti [126]. Inizialmente il client SkypeMorph utilizza l'infrastruttura Skype per individuare i proxy e condurre la configurazione della sessione, per esempio scambierà le chiavi tramite la funzionalità di chat, e successivamente avvia una videochiamata direttamente al nodo. A questo punto il nodo ignora la chiamata e si mette in ascolto sulla porta stabilita, il client smetterà di utilizzare Skype e tenta semplicemente di imitarlo inviando dati crittografati in un flusso dalle caratteristiche simili a quelle di una videochiamata Skype. FreeWave [78] si occupa di inosservabilità e sbloccabilità, e sebbene sia simile nel concetto a SkypeMorph, fa un ulteriore passo in avanti nell'imitazione delle chiamate. Questo sistema infatti prevede l'invio effettivo del traffico IP tramite voce utilizzando un modem virtuale. I dati inviati da e verso il proxy vengono modulati in suoni tramite un modem software che a sua volta li trasmette a una scheda audio virtuale, inoltrandoli al server proxy tramite la chiamata Skype. Il terzo strumento disponibile per creare canali nascosti sfruttando VoIP è CensorSpoof [189]. L'osservazione chiave su cui si basa questo tool è che il traffico internet è altamente asimmetrico: una piccola quantità di dati viene inviata (richiesta) e un grande volume viene ricevuto (risposta). Nell'architettura di CensorSpoof la richiesta viene effettuata tramite un canale a bassa larghezza di banda, ad esempio un messaggio di posta elettronica, mentre i dati tramite protocollo SIP. L'utente contatterà un ID SIP, ma il proxy CensorSpoof non risponderà con

il suo IP, ma con un IP di un host casuale su internet che da una scansione nmap⁷³ risulta essere online e avere la porta SIP aperta. Così facendo il client non scopre mai l'indirizzo del proxy perché nei pacchetti che riceve è contenuto l'IP di un'altra macchina. Il client invierà dei dati indesiderati all'host fittizio, che scarterà in quanto non c'è una comunicazione attiva tra i due, per simulare una chiamata VoIP.

Deltashaper. Questo proof-of-concept sfrutta il canale video criptato delle applicazioni di videoconferenza più diffusi, come Skype, come vettore per comunicazioni TCP/IP segrete [17]. Questo sistema offre una interfaccia data-link e supporta applicazioni TCP/IP che tollerano collegamenti a basso throughput/alta latenza. I test effettuati dimostrano che è possibile eseguire protocolli standard come FTP, SMTP o HTTP sui flussi Skype.

Nome	Licenza	Linguaggio	Descrizione	Rilascio	Ultima Modifica
SkypeMorph [126]	-	C/C++	Covert channel su VoIP	2012	2012
FreeWave [78]	-	-	Covert channel su VoIP	2012	2012
CensorSpoof [189]	-	-	Covert channel su VoIP	2012	2012
Deltashaper [17]	MIT	C++	Canale video di videoconferenza come vettore per comunicazioni TCP/IP	2017	2017

Tabella 3.8: Tool per implementare canali coperti tramite tecnologia VoIP.

3.3.10 Altri tool

In quest'ultima sottosezione vengono riportati tutti quei tool che operano in maniera ibrida tra i vari livelli, o che comunque non è possibile posizionarli in un livello successivo. Inoltre alcuni presentano delle tecniche innovative e forniscono la scelta di quale tecnica utilizzare. In Tabella 3.9 sono riportate le caratteristiche fondamentali dei tool di seguito analizzati.

⁷³<https://nmap.org>.

Decoy routing. Una tecnica presentata più di recente, che permette sempre di aggirare dei sistemi di censura, è chiamata Decoy Routing. Questo sistema viene implementato direttamente sui router e non sugli host, i cui indirizzi IP possono essere facilmente bloccati. I router sono più difficili da filtrare perché i pacchetti non contengono indirizzi di router e inoltre se ben posizionati possono fungere da ponte rea un client e milioni destinazioni. Con questa proposta un router funge da (o fornisce accesso a) un server proxy. Il router così modificato si chiama decoy router. Il client si connette a una qualsiasi destinazione che comprende il decoy router nel percorso e una volta connesso segnala in modo occulto, attraverso il traffico TCP/IP, su quale proxy esca deve reindirizzare il traffico il decoy router. Il proxy esca fungerà poi come un proxy standard per il client. Una implementazione che sfrutta le e-mail come canale coperto è stata pubblicata su GitHub⁷⁴.

Marionette. Tra i vari strumenti disponibili per eludere sistemi di censura nel 2014 è stato presentato Marionette⁷⁵. Questo tool rappresenta il primo sistema programmabile di offuscamento del traffico di rete in grado di controllare simultaneamente le caratteristiche del traffico criptato a vari livelli, compresi i formati dei testi cifrati, la semantica dei protocolli statici e le proprietà statistiche. Il comportamento del sistema si basa su un linguaggio specifico che consente all'utente di regolare facilmente la strategia di offuscamento per soddisfare le esigenze dell'ambiente in cui si opera. Marionette quindi è in grado di emulare molti sistemi di offuscamento esistenti e consente di lavorare su una varietà di protocolli e a diversi livelli. Oltre a un'estrema flessibilità e controllo sulle caratteristiche del traffico, Marionette è anche in grado di raggiungere un throughput fino a 6,7 Mbps quando genera traffico di copertura conforme a RFC.

CCgen. Ad aprile del 2022 è stato presentato CCgen⁷⁶, un framework per l'iniezione di canali nascosti comuni nei traffici di rete e di trasporto. Questo strumento è stato sviluppato in Python ed è liberamente disponibile con licenza GPL-3.0, fornisce si la possibilità di operare nelle comunicazioni in real-time oppure operare su file di cattura

⁷⁴rkc007, DecoyRKC, 2019, <https://github.com/rkc007/DecoyRKC>.

⁷⁵marionette-tg, Marionette, 2014, <https://github.com/marionette-tg/marionette>.

⁷⁶CN-TU, CCgen, 2022, https://github.com/CN-TU/py_CCgen.

precedentemente effettuati. Le performance di questo strumento è stata testate con Suricata, un sistema open source di prevenzione e di rilevamento delle intrusioni, il quale conferma che la maggioranza dei canali rimane inosservata. Il programma si compone di un modulo centrale e da altri moduli secondari che ne aumentano le funzionalità e le possibili applicazioni. Nella Figura 3.13 è possibile visualizzare l’interfaccia grafica dopo aver effettuato un test di una iniezione tramite modifica degli indirizzi di destinazione.

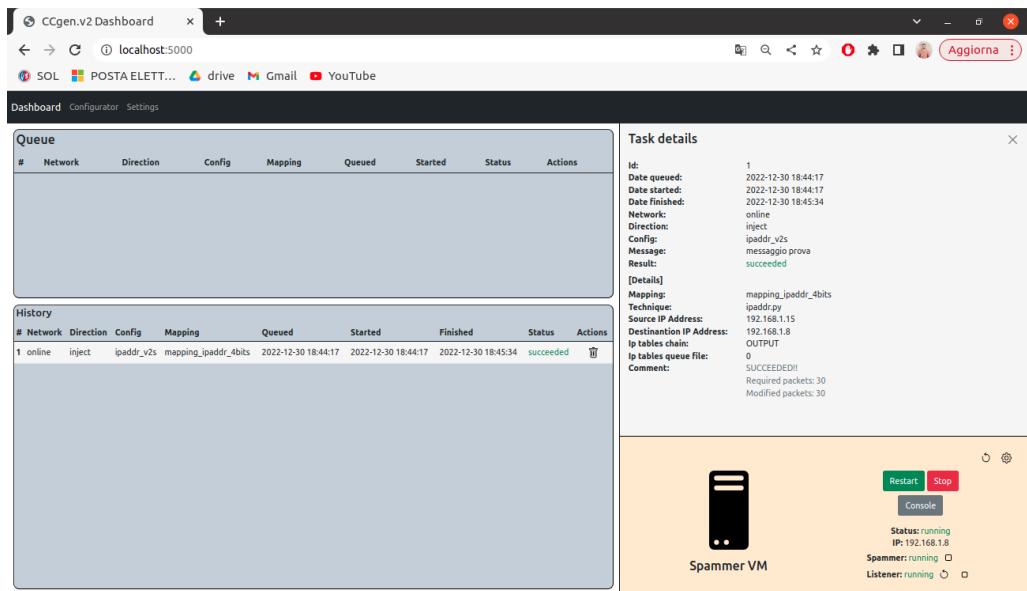


Figura 3.13: Interfaccia del tool CCgen.

WiFi reconnection - CoAP. Nel 2020 Wendzel e il suo team hanno presentato due canali nascosti basati sulla riconnessione dei dispositivi mobili ad una rete wifi [218]. Entrambi i prototipi sono sviluppati in Python e pubblicati con licenza GPL-3.0 su GitHub. Per quanto riguarda la loro composizione entrambi prevedono due moduli, quello del sender e quello del receiver. I due moduli vengono eseguiti su due host collegati ad una rete wifi, il sender forza la disconnessione di alcuni dispositivi mobili, obbligati poi a riconnettersi, mentre il receiver monitora queste cadute. In questo caso si tratta di canali indiretti in quanto il mittente e il destinatario dei dati segreti non comunicano direttamente tra loro.

linuxptp-CC. Aron J. Smith-Donovan nel 2022, tramite la sua tesi [169], ha dimostrato come l'implementazione del Precision Time Protocol in Linux presenti delle vulnerabilità. Il suo lavoro rappresenta un fork della versione originale in cui vengono modificate le funzioni "hdr _pre _send" e "hdr _post _recv" così da permettere l'invio di 7B per ogni messaggio. Infatti riesce a inviare 7 caratteri ASCII, alcuni dei quali però i bit vanno divisi in due gruppi da 4 bit, sfruttando le intestazioni del pacchetto non utilizzate. Il progetto è sviluppato in C e diffuso con licenza GPL-2.0.

PCT. In rete è possibile anche trovare un prototipo dei cosiddetti Canali Protocolari [191]. Con questa tecnica non vengono inseriti dei bit all'interno delle intestazioni dei pacchetti, ma lo 0/1 sono rappresentati da due protocolli. Nella pratica se si vuole inviare 0 verrà inviato un pacchetto autentico di un protocollo, nel caso dell'uno si cambia protocollo. Il codice del prototipo, scritto in C e Perl, è disponibile su GitHub con licenza GPL-3.0, questo è stato testato su ambiente Linux.

PHCC Tra le proposte fatte da Wendzel troviamo anche il Protocol Hopping Covert Channel Tool (PHCCT)⁷⁷. Questo prototipo molto semplice e pubblicato su GitHub prevede l'instaurazione di diversi canali di comunicazione tra il sender e il receiver basati su protocolli e tecniche di network steganography diverse. Nel caso in cui il canale in uso venisse bloccato gli altri sono ancora attivi e si sposta la comunicazione su uno di questi.

CCTT. Covert Channel Tunneling Tool (CCTT)⁷⁸ è lo strumento scritto in C da Simon Castro e scaricabile da *packetstormsecurity.com*. L'autore dopo aver utilizzato diversi strumenti ha deciso di raccogliere le varie funzionalità di cui necessitava in un'unica soluzione. Secondo l'autore il server doveva essere in grado di gestire più client, dare accesso alla shell ai client e sia server che client dovevano operare in modalità proxy: il client CCTT accetta le connessioni dai client applicativi, le inoltra al server CCTT che a sua volta invia i dati ai server applicativi.

⁷⁷Steffen Wendzel, PHCCT - Protocol Hopping Covert Channel Tool, 2020, <https://github.com/cdpxe/NetworkCovertChannels/tree/master/phcct>.

⁷⁸Simon Castro, CCTT, 2003, <https://packetstormsecurity.com/files/download/31227/cctt-0.1.7.tar.gz>.

SteganRTP. Nel 2007 è stato presentato SteganRTP⁷⁹, scaricabile dal sito *sourceforge.net*, un tool che sfrutta una sessione RTP come traffico di copertura per inviare dati in modo nascosto. Lo strumento per poter funzionare ha bisogno di sapere quali sono i due endpoint della sessione e li classificherà come estremità vicina e lontana, in modo da determinare la direzione del flusso di traffico e decidere su quale inserire le informazioni. SteganRTP utilizza la libreria *libfindrtp* per monitorare le segnalazioni VoIP e la negoziazione dei parametri, una sessione già attiva può essere comunque usata ma i parametri a quel punto vanno passati a riga di comando. Questo tool dispone di una interfaccia grafica composta da quattro finestre: quella di comando nella parte inferiore, quella principale al centro e quelle di ingresso e di uscita nella parte superiore.

Timeshifter. Un'altra tipologia di canali nascosti riportata nel Capitolo 2 è quella dei canali temporali, cioè l'insieme di quelle tecniche che intervengono sui tempi di interarrivo dei pacchetti per codificare le informazioni. Queste proposte, rispetto agli storage covert channel, presentano una difficoltà in più nell'implementazione che è la sincronizzazione tra mittente e destinatario, per questo sono meno diffuse. Timeshifter⁸⁰ è un tool scritto in C al quale vanno passate le informazioni da inviare e due valori numerici: tutti i ritardi tra pacchetti minori o uguali al primo saranno considerati come 0, mentre il secondo numero rappresenta il delay da aggiungere ai pacchetti che codificano 1.

MalCert. Tra le tecnologie sfruttate per effettuare delle comunicazioni occulte ci sono i certificati x.509. Questi certificati sono documenti digitali che associano in modo sicuro coppie di chiavi crittografiche a identità come siti Web, individui o organizzazioni. Il proof-of-concept MalCert⁸¹ dimostra come sia possibile utilizzare le estensioni x509 per la trasmissione e ricezione di dati. Il tool è presente su GitHub, i moduli client e server sono scritti totalmente in Go ed è associato a una pubblicazione [149].

⁷⁹druid-cau, SteganRTP - RTP Covert Channel, 2007, <https://sourceforge.net/projects/steganrtp/>.

⁸⁰anfractuousity, timeshifter, 2015, <https://github.com/anfractuousity/timeshifter>.

⁸¹fideliscyber, MalCert, 2017, <https://github.com/fideliscyber/x509>.

CPU load side channel. L’esfiltrazione dati non avviene necessariamente tra reti di computer diverse, a volte per raggiungere l’obiettivo occorre trasmettere informazioni tra processi diversi che girano sulla stessa macchina. Questo proof-of-concept⁸² dimostra come sia possibile scambiare dati tra processi tramite un registro di sistema nell’Apple M1. Non è da escludere che questa tecnica possa essere utilizzata anche tra macchine virtuali che girano sullo stesso host. Il metodo si basa sulla modulazione del protocollo di accesso multiplo a un canale condiviso di comunicazione, chiamato CDMA. È una delle tecniche più recenti e ben funzionante però ha una velocità di trasmissione di circa 0.06 bit al secondo se la comunicazione avviene tra ambienti virtuali differenti.

Chmod-stego. I metodi per trasmettere informazioni in modo nascosto sono molti e ci sono prototipi in qualsiasi ambito. Chmod-stego⁸³ è un PoC tramite il quale due entità si scambiano informazioni attraverso i privilegi assegnati ai file in Linux, cioè vengono modificate le triple RWX, che indicano i permessi di lettura, scrittura ed esecuzione, dei file. Si compone di due script, mittente e destinatario, e prevede solo una comunicazione unidirezionale. Le due parti condividono un minimo protocollo di trasferimento e un meccanismo di auto-sincronizzazione. La capacità trasmisiva di questo approccio può essere considerevole data la grande quantità di file presenti in un computer.

Rook. Il tool rook⁸⁴ è sviluppato a moduli ed ha lo scopo di fornire uno strumento di comunicazione occulto. Grazie al suo modulo che funge da parser è in grado di analizzare i pacchetti e trovare le sezioni di bit che, anche se alterate, non portano alla classificazione di quel pacchetto come non valido. L’autore ha progettato questo strumento principalmente per il traffico dei giochi di sparatutto in prima persona, ma il suo approccio può essere utilizzato per qualsiasi applicazione.

⁸²pavel-kirienko, cpu-load-side-channel, 2021, <https://github.com/pavel-kirienko/cpu-load-side-channel>.

⁸³operatorequals, chmod-steg, 2016, <https://github.com/operatorequals/chmod-steg>.

⁸⁴plvines, rook, 2016, <https://github.com/plvines/rook>.

Canale segreto SNMP - Chat. Sviluppato inizialmente per scopi accademici, su GitHub è stato pubblicato un tool⁸⁵ che fornisce le funzionalità di chat attraverso il traffico SNMP. Si compone di unico modulo scritto in Python al quale vanno passati gli indirizzi del mittente e del destinatario.

DET. DET (extensible) Data Exfiltration Toolkit⁸⁶ è uno strumento scritto in Python disponibile su GitHub per effettuare esfiltrazione dati. Questo è un prototipo che implementa tecniche per esfiltrare dati attraverso uno o più canali contemporaneamente, è stato sviluppato con lo scopo di testare nuovi sistemi di Network Monitoring e Data Leakage Prevention (DLP). Dispone di un modulo al quale è possibile aggiungere diversi plugin per le varie tecniche e servizi che si vogliono sfruttare, per ognuno di questi è necessaria una configurazione che va inserita nel file .json, uno per la parte client e uno per la parte server.

DarkFinger. Una possibile misura di sicurezza che applicano i sistemi operativi è quella di una whitelist di comandi permessi, tra questi ci saranno quasi sicuramente i comandi nativi del sistema operativo stesso. L'utility CertUtil.exe è stato bloccato da Windows Defender in quanto era un programma nativo che permetteva di scaricare file, ma nel 2020 è stato proposto DarkFinger⁸⁷. Questa tecnica rappresenta un utilizzo alternativo del comando nativo Finger TCP/IP di Windows che lo porta a svolgere la funzione di downloader e canale di comando e controllo. Questo comando non è bloccato da Windows Defender.

PowerExfil. Nella cartella GitHub PowerExfil⁸⁸ sono contenuti 4 script PowerShell per l'esfiltrazione di dati. In particolare uno è rivolto all'esfiltrazione dei numeri di carta di credito, tramite stringhe codificate in base64, verso un server DNS personalizzato, mentre il secondo esfiltra l'intero contenuto di un file. Gli ultimi due permettono l'invio di un intero file o di un comando tramite e-mail a un indirizzo qualsiasi oppure tramite richieste HTTP POST a server web personalizzato.

⁸⁵bernardi-sena-sgrinzi, SNMP Covert Channel - Chat, 2017, <https://github.com/bernardi-sena-sgrinzi/covert-channel>.

⁸⁶PaulSec, DET (extensible) Data Exfiltration Toolkit, 2016, <https://github.com/PaulSec/DET>.

⁸⁷hyp3rlinx, DarkFinger, 2020, <https://github.com/hyp3rlinx/DarkFinger-C2>.

⁸⁸1N3, PowerExfil, 2020, <https://github.com/1N3/PowerExfil>.

NaishoDeNusumu. Il tool Naisho⁸⁹ è stato sviluppato per facilitare le operazioni di penetration test da parte del red team. È scritto interamente in Python e permette l’esfiltrazione dati attraverso canali segreti, al fine di evitare sistemi DLP, IDS e altri strumenti di rilevamento. Non è correlato di una buona documentazione e da 8 anni non risulta aggiornato.

Nome	Licenza	Linguaggio	Descrizione	Rilascio	Ultima Modifica
DecoyRKC	MIT	C/C++/ Python	Decoy routing con email come canale coperto	2019	2019
Marionette	Apache 2.0	Python	Strumento di offuscamento programmabile	2014	2015
CCgen	GPL-3.0	Python	Framework per l’iniezione di canali nascosti	2022	2022
WiFi Reconnection - CoAP [218]	GPL-3.0	Python/C++	Canale indiretto in una rete wifi	2021	2021
linuxptp-CC [169]	GPL-2.0	C	Vulnerabilità in PTP su Linux	2022	2022
PCT [191]	GPL-3.0	C/Perl	Canale protocollare	2008	2008
PHCCT	GPL-3.0	C	Switch di canale coperto	2020	2020
CCTT	GPL-2.0	C	Implementazione di diversi canali coperti	2003	2003
SteganRTP timeshifter	GPL-2.0 -	C	Covert channel sopra RTP	2007	2015
		C	Canale coperto nei ritardi dei pacchetti	2015	2018
MalCert	MIT	Go	Comunicazione occulta attraverso le estensioni dei certificati x.509	2017	2018
cpu-load-side-channel	MIT	C++	Trasmissione dati tra processi	2021	2021
chmod-steg	-	Python	Trasmissione dati tramite triple RWX	2016	2017
rook	GPL-3.0	Python	Comunicazione occulta nel traffico di sparututto in prima persona	2016	2016
SNMP CC - Chat	-	Python	Chat su traffico SNMP	2017	2018

⁸⁹ Adam Crompton, NaishoDeNusumu, 2014, <https://github.com/3nc0d3r/NaishoDeNusumu>.

DET	MIT	Python	Tecniche di esfiltrazione per testare DLP	2016	2019
DarkFinger	MIT	Python	Utilizzo del comando finger come canale C2 e di esfiltrazione	2020	2022
PowerExfil	-	PowerShell	Script powershell per esfiltrazione dati	2020	2020
Naisho	-	Python	Framework per esfiltrazione dati tramite canali coperti	2014	2014

Tabella 3.9: Tool per implementare canali coperti disponibili in rete.

3.3.11 Pluggable Transport nella rete Tor

Il progetto *TOR (The Onion Router)*⁹⁰ inizialmente è nato per fornire l'anonimato ai suoi utenti mentre navigano in rete, ma col passare degli anni la sua struttura si è fatta più complessa e alcuni censori hanno implementato delle contromisure a questa tecnologia. Per ovviare a questi problemi sono stati aggiunti diversi servizi, come i Pluggable Transport che forniscono un livello in più di trasformazione attorno al traffico del protocollo OR. Una caratteristica di queste soluzioni è che possono essere applicate anche al normale traffico TCP e non solo all'interno della rete TOR. Tra i PT più famosi che cercano di mascherare il traffico TOR affinché non venga rilevato come tale troviamo:

- obfs2
- obfs3
- obfs4
- Meek
- ScrambleSuite
- FTE
- Snowflake

⁹⁰<https://www.torproject.org>

- Stegotorus

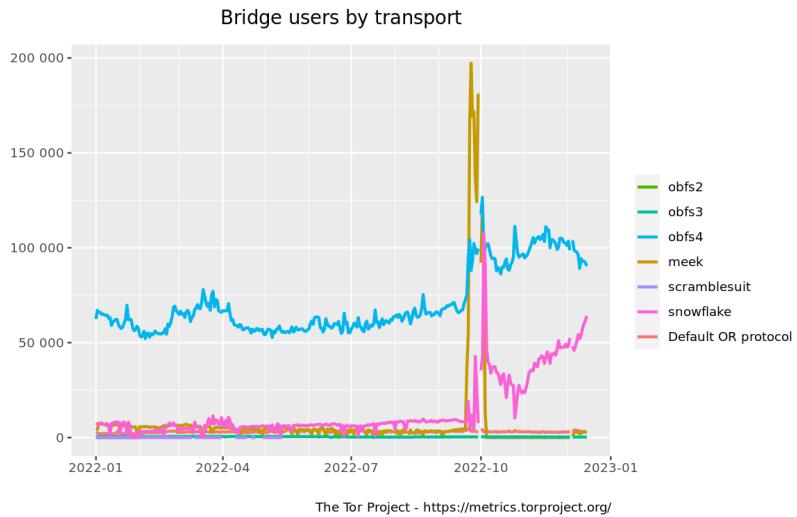


Figura 3.14: Numero di utenti connessi via bridge.

Come osservabile in Figura 3.14 nel 2022 i pluggable transport disponibili sono obfs4, meek e snowflake.

obfs4. Obfs4 rappresenta l'ultima versione di obfsproxy, un PT che introduce un ulteriore strato di crittografia tra il client e il bridge. L'obiettivo di questa ulteriore cifratura è quello di far sembrare il traffico TOR come byte casuali.

meek. Questo PT invece fa assomigliare il traffico TOR a una connessione a un sito web HTTPS. A differenza degli altri non si connette direttamente con il bridge ma prima si collega con un vero server web HTTPS (nel cloud Amazon o in Microsoft Azure) dal quale poi si interfacerà con il bridge vero e proprio. Il censore non potrà bloccare i server web così utilizzati perché questi forniscono anche altri servizi utili. Meek risulta essere uno tra i PT più pesanti e difficili da gestire.

snowflake. Snowflake è un miglioramento di Flashproxy e rappresenta probabilmente il Pluggable Transport più facile da implementare e da utilizzare, dato che può semplicemente bastare mantenere attiva una particolare finestra nel browser. Questa

tecnica utilizza la tecnologia WebRTC, la stessa utilizzata da Skype, Zoom, e altri fornitori di connessioni peer-to-peer nel web. Snowflake prevede la partecipazione di 3 soggetti: i volontari che eseguono i proxy snowflake, gli utenti Tor e il broker. Una volta che viene avviato un nuovo proxy questo contatterà il broker comunicando la sua disponibilità, anche l'utente contatterà prima il broker chiedendo un proxy con il quale interfacciarsi per accedere alla rete Tor. Il broker non fa altro che comunicare l'indirizzo IP di un proxy all'utente.

3.4 Attacchi che hanno usato steganografia di rete

Questo tipo di steganografia è fondamentale per implementare tecniche di attacco e di esfiltrazione dati, infatti in alcuni incidenti registrati è stato possibile analizzare nel dettaglio le soluzioni adottate dagli attaccanti. Qui di seguito un piccolo elenco con alcuni attacchi verificatisi realmente che hanno utilizzato le tecniche affrontate in questo documento:

- Sunburst: i dati sono nascosti nelle conversazioni HTTP e i comandi sono estratti tramite la scansione regexp dei corpi delle risposte,
- Okrum and Ketrican: le comunicazioni C&C sono nascoste nel traffico HTTP, ad esempio nelle intestazioni Set-Cookie e Cookie delle richieste,
- DarkHydrus: utilizza un tunnel DNS per trasferire le informazioni, una tecnica simile a quella registrata con Morto e Feederbot,
- ChChes: utilizza le intestazioni dei Cookie del protocollo HTTP per le comunicazioni C&C,
- NanoLocker: nasconde i dati nei pacchetti ICMP,
- FAKEM RAT: le comunicazioni C&C sono camuffate in Yahoo! Messenger e MSN Messenger oltre che nel protocollo HTTP (non si tratta strettamente di steganografia di rete),
- infine in alcuni cyberattacchi è stato registrato l'utilizzo di Backdoor.Win32.Denis che nasconde i dati all'interno di un tunnel DNS per le comunicazioni C&C.

Capitolo 4

Sviluppo di un Proof of Concept

In questo capitolo, dopo aver analizzato la letteratura relativa alla network steganography e le soluzioni disponibili per implementare alcune tecniche, presentiamo il nostro strumento. Prima di tutto illustreremo i motivi per i quali sono state effettuate alcune scelte piuttosto che altre, successivamente l'idea alla base della nostra applicazione ed infine i dettagli implementativi.

4.1 Analisi dei protocolli più utilizzati nella steganografia di rete

Alla luce di quanto riportato nei capitoli precedenti è possibile avere un quadro ben delineato di ciò che ruota attorno alla steganografia di rete. Possiamo affermare che sono state pubblicate proposte per ogni protocollo di rete, dal livello Applicazione fino al livello Data Link, ma soltanto una parte di queste poi hanno visto un riscontro con un'implementazione pratica. I protocolli protagonisti, per i quali è possibile trovare più proposte e implementazioni sono IP, TCP, UDP, ICMP, DNS e HTTP. Il punto di forza dei primi tre è rappresentato dal fatto che la maggior parte del traffico di rete viene generato ai livelli superiori nella pila ISO/OSI e quindi avere un'implementazione che opera più a basso livello è in grado di essere applicata a una maggior quantità di pacchetti. Lo svantaggio è dato dal fatto che per operare a questi livelli sono necessari i diritti di root, il ché potrebbe ridurre i possibili ambiti di applicazione.

Per quanto riguarda ICMP e DNS, questi sono i protocolli più utilizzati nella pratica in quanto mettono a disposizione un payload facilmente sfruttabile e la possibilità di avviare le comunicazioni senza i diritti di root. Il lato negativo è dato dal fatto che la loro grande diffusione ha portato a un proliferare di misure di monitoraggio facili da implementare e molto efficienti, grazie all'intelligenza artificiale, proprio contro ICMP e DNS.

Infine abbiamo il protocollo a livello Applicazione legato alla navigazione web, l'HTTP. Oggigiorno la maggioranza del traffico nelle reti di computer è rappresentata dal traffico internet a causa di diversi fattori, come per esempio il fatto che molti servizi vengono forniti tramite il cloud. Data la grande quantità è difficile trovare sistemi di monitoraggio che analizzino a fondo i pacchetti HTTP, piuttosto effettuano un controllo sulle caratteristiche di una comunicazione. Per esempio se in una comunicazione è il client a inviare più dati rispetto al server questo potrebbe portare ad un'anomalia. Anche le comunicazioni attraverso questo protocollo possono essere avviate senza i diritti di root ma la maggior parte delle implementazioni presenti non si tratta di steganografia pura di rete, piuttosto di tecniche di tunneling di altri protocolli che vengono inglobati all'interno di traffico HTTPS apparentemente innocuo. Viene utilizzata questa strategia perché l'uso di HTTPS fornisce una protezione ulteriore alla comunicazione, in quanto il pacchetto viene cifrato e il contenuto quindi risulta nascosto.

4.2 Motivazione delle scelte progettuali

Dati alla mano, a questo punto abbiamo scelto le caratteristiche principali del nostro prototipo. Il nostro obiettivo era creare un proof of concept che fosse applicabile sulla maggior parte dei computer e che sia in grado di bypassare i principali sistemi di monitoraggio. Sicuramente il non possesso dei diritti di root per noi ha rappresentato una clausola importante che ci ha spinto a scegliere un protocollo a livello Applicazione. Dato che la maggior parte dei computer ha accesso a Internet e sono utilizzati da utenti che effettuano ricerche sul web quotidianamente si è scelto di basare il nostro prototipo sul protocollo HTTP.

A questo punto si è dovuto scegliere tra un canale bidirezionale o unidirezionale. Generalmente i canali unidirezionali sono usati per effettuare esfiltrazione dati

mentre quelli bidirezionali sono di comando e controllo oppure sono usati per aggirare la censura. La nostra decisione è stata quella di concentrarci su una tecnica più vicino all'esfiltrazione dati, e quindi a un canale unidirezionale, questo sempre per renderla utilizzabile in un maggior numero di situazioni. La nostra applicazione risulta comunque molto flessibile e facilmente ampliabile con le funzioni di comunicazione nella direzione opposta, così da ottenere un ipotetico canale bidirezionale.

L'ultimo aspetto ha riguardato la strategia di incorporazione dei dati nei pacchetti HTTPS. Considerando l'argomento principale di questa tesi, la steganografia di rete, e la scarsità di applicazioni pure per HTTP, abbiamo deciso di progettare una soluzione che fosse in grado di utilizzare i vari campi di intestazione di una richiesta HTTP per inglobare le informazioni.

4.3 L'idea

Analizzando le varie proposte pubblicate in letteratura quella che abbiamo ritenuto più utile per il nostro lavoro è quella di Castiglione et al [30] riguardo le e-mail. Partendo dalla loro idea di inserire i dati nei campi "Message-ID" e "Content-Type" abbiamo ampliato la tecnica agli header delle richieste HTTP, adattando la tecnica di rappresentare i dati alla natura dei vari campi.

4.3.1 Funzionamento

Il nostro prototipo si divide in due parti principali: il `sender.py` e il `receiver.py`. Il primo è rappresentato da uno script Python che riceve in input come parametri l'url, verso le quali inviare le richieste, e il file o directory da steganografare all'interno delle intestazioni. Il funzionamento è schematizzato nella Figura 4.1.

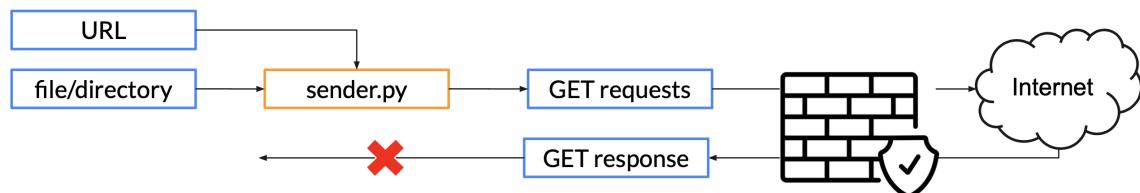


Figura 4.1: Funzionamento di `sender.py`.

Invece il receiver, come il sender, è scritto in Python ed è a tutti gli effetti un server web in ascolto su una determinata porta. Questa porta, e il nome dei file da produrre vengono presi come parametri, altrimenti userà quelli impostati di default che sono rispettivamente la porta 8000 e il nome *file_output*. I file estratti dalle richieste verranno salvati nella cartella *file_received* rinominati con al stringa passata come parametro seguita da un numero crescente che parte da 0. Il receiver ad ogni richiesta risponderà con una pagina web casuale in modo da concludere la comunicazione e non far generare anomalie, dopodiché analizzerà gli header delle richieste per recuperare i dati steganografati e ricostruire il file inviato. Il suo funzionamento è schematizzato nella Figura 4.2. Precisiamo che il nostro prototipo è in grado di inviare ogni tipo di file dato che lavora a livello di bit.

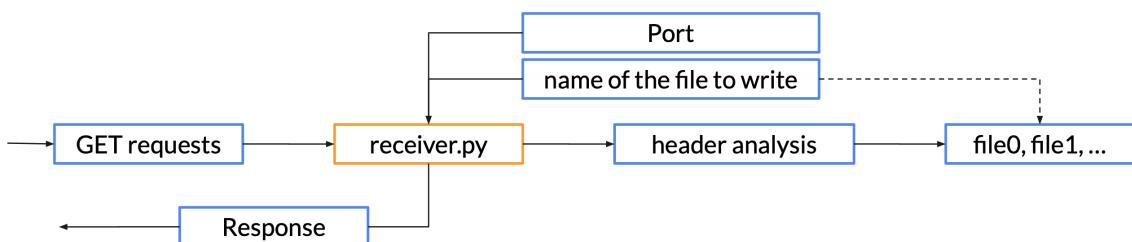


Figura 4.2: Funzionamento di receiver.py.

4.3.2 I campi di intestazione del protocollo HTTP

Ogni richiesta HTTP è composta da due parti: gli header¹ e il body. Nella prima sono contenute tutte le informazioni per elaborare i dati contenuti nella seconda. Tramite questi campi, nelle richieste, il client va a comunicare al server quali parametri è in grado di processare e come preferirebbe ricevere le risorse richieste. Per riportare alcuni esempi può comunicare che è in grado di elaborare risorse compresse oppure no, che preferisce ricevere una risorsa in una lingua piuttosto che in un'altra o che vorrebbe la versione di una risorsa risalente a una qualche data. Alcuni di questi campi sono indispensabili per creare una richiesta effettivamente funzionante, mentre altri vengono usati solo in condizioni particolari. Inoltre è da sottolineare che il client comunica queste preferenze ma il server è libero di ignorarle o, nel caso in cui non sia

¹<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>.

in grado di rispettarle, di rispondere secondo le sue preferenze. Di questi campi ne esistono alcuni standard che possono essere inclusi nelle richieste e nelle risposte ma c'è anche la possibilità di inserirne alcuni personalizzati. Tra questi troviamo quelli utilizzati per alcune impostazioni riguardanti i proxy, le misure di sicurezza oppure alcuni impostati direttamente dai server per ottimizzare le comunicazioni.

Campi standard e non. I campi standard possono essere divisi in due categorie: quelli per le richieste e quelli per le risposte. Tramite quelli delle richieste il client è in grado di comunicare quali manipolazioni, caratteri, codifiche e lingue è in grado di elaborare per la risorsa. Nel caso fosse necessario, tramite alcuni di questi, colui che effettua la richiesta può fornire alcune sue informazioni al server come la data, l'user agent o quelle relative all'autenticazione. Un altro insieme di header invece permette di richiedere versioni o porzioni specifiche di alcune risorse, oppure richiederne l'invio soltanto se questa ha subito degli aggiornamenti. Qualora il client dovesse inviare dei dati utilizzerà quegli header contenenti le specifiche dei dati come la lunghezza, il tipo, la codifica o l'hash. Questi ultimi campi sono gli stessi utilizzati nelle risposte dal server quando trasmette i dati richiesti, i quali naturalmente saranno accompagnati da altri campi più specifici che faciliteranno maggiormente l'elaborazione della risorsa richiesta. Su quelli standard previsti per le risposte inoltre c'è "Set-Cookie" che serve per comunicare quali cookie deve impostare il client per le comunicazioni successive. Su quelli non standard troviamo un insieme di header molto più specifici per la risorsa inviata, come per esempio "X-Content-Duration" che contiene la durata dell'audio o del video in secondi. Dati i vari sistemi che una richiesta deve attraversare per raggiungere il server e viceversa è probabile che alcuni parametri vengano modificati, controllati o aggiunti dai proxy, per esempio, per ottimizzarne il recapito.

4.3.3 Struttura della richiesta inviata

I campi utilizzabili per creare le nostre richieste sono molti se si considerano anche quelli non standard, inoltre alcuni implicano la presenza e l'assenza di altri. Altro aspetto per noi molto importante era la quantità di bit che riuscivamo a iniettare in un singolo campo oltre naturalmente a evitare quei campi che sarebbero potuti essere soggetto di modifiche durante il percorso. Tutti questi vincoli ci hanno portato

a costruire le nostre richieste come se richiedessimo parte di una versione passata di una risorsa tramite il metodo GET. Di seguito andremo ad elencare i campi che compongono la richiesta:

- Accept: contiene i tipi MIME che sono accettati per la risposta;
- Accept-Encoding: contiene i tipi codifica accettati della risposta;
- Accept-Language: contiene le lingue preferite in cui ricevere la risposta;
- Accept-Datetime: contiene la data della risorsa richiesta;
- From: contiene l'email di chi ha effettuato la richiesta;
- If-Match: contiene almeno un valore identificativo di una risorsa e l'azione viene eseguita se l'entità inviata dal client corrisponde a quelle presente sul server;
- Range: contiene gli intervalli di byte richiesti di una particolare risorsa;
- TE: contiene le codifiche di trasferimento che l'agente è disposto ad accettare;
- User-Agent: la stringa identificativa dell'user agent che effettua la richiesta.

4.3.4 Tecniche utilizzate nei vari campi

Nei paragrafi successivi andremo a descrivere la tecnica utilizzata per ogni campo per rappresentare le varie stringhe di bit. Principalmente andremo a lavorare con potenze di 2 per facilitare la rappresentazione delle possibili stringhe binarie. Facendo riferimento alla tassonomia descritta nella Sezione 2.10.5 possiamo considerare la tecnica proposta in questa tesi nella categoria PS11 perché si tratta di Modulazione del valore e la struttura viene preservata.

Accept. In questo campo è previsto che vengano inseriti i tipi MIME che sono accettati per la risposta, ma questi possono essere svariati, perciò abbiamo deciso di utilizzare soltanto quelli più comuni per creare richieste che generino meno anomalie possibili. Nel nostro caso andremo a lavorare con 16 valori possibili, quindi riusciremo a steganografare sequenze di 4 bit all'interno del campo Accept. Nella Tabella 4.1

sono i riportati i valori scelti e le rispettive stringhe binarie rappresentate.

Valore	Bit
text/plain	0000
text/html	0001
text/css	0010
text/javascript	0011
image/gif	0100
image/png	0101
image/jpeg	0110
image/webp	0111
video/mpeg	1000
video/webm	1001
video/ogg	1010
video/mp4	1011
application/octet-stream	1100
application/javascript	1101
application/xml	1110
application/pdf	1111

Tabella 4.1: Modulazione dei valori nel campo Accept.

Accept-Encoding. Nel campo Accept-Encoding vengono inseriti i tipi di codifica che sono accettati per la risposta. Anche in questo campo è possibile avere un valore o una lista di essi, e tra le tante possibilità, ragionando sempre per potenze di 2, abbiamo selezionato le 8 soluzioni più comuni che si incontrano nelle richieste web. Nella Tabella sono riportati i valori scelti e le rispettive stringhe di bit rappresentate. Usando otto valori riusciamo a steganografare stringhe di 3 bit.

Valore	Bit
gzip	000
compress	001
deflate	010
identity	011
gzip, compress	100
gzip, deflate	101
gzip, identity	110
gzip, br	111

Tabella 4.2: Modulazione dei valori nel campo Accept-Encoding.

Accept-Language. Tramite questo campo il client comunica al server in quale lingua o lingue preferisce ricevere la risorsa richiesta. Quindi è possibile inserire uno o più valori, inoltre tramite il valore ":q=" si può stabilire anche la preferenza in presenza di più valori. In questo caso abbiamo utilizzato 3 lingue: italiano, perché immaginiamo un'applicazione nel nostro paese, e inglese americano e britannico, le due lingue più diffuse al mondo. Oltre a utilizzare la presenza o meno di una delle lingue prese in considerazione, sfruttiamo anche il valore di preferenza "q" per ampliare l'insieme di dati rappresentabili. Il massimo numero di bit steganografati con Accept-Language è 3, ma a differenza degli altri campi questo viene utilizzato anche per sequenze di bit di lunghezza minore. È stato ritenuto necessario inserire questo accorgimento per garantire che qualsiasi sia la lunghezza del blocco di informazioni da inviare, questo abbia una combinazione che gli permetta di inviare tutti i bit che lo compongono. Nella Tabella 4.3 sono riportati i valori scelti e le rispettive stringhe di bit rappresentate.

Valore	Bit
it	0
it, *	1
it, en-US	00
it, en-US:q=0.8	01

it:q=0.9, en-US	10
it:q=0.9, en-US:q=0.8	11
it, en-US, en-GR	000
it, en-US, en-GR:q=0.7	001
it, en-US:q=0.8, en-GR	010
it, en-US:q=0.8, en-GR:q=0.7	011
it:q=0.9, en-US, en-GR	100
it:q=0.9, en-US, en-GR:q=0.7	101
it:q=0.9, en-US:q=0.8, en-GR	110
it:q=0.9, en-US:q=0.8, en-GR:q=0.7	111

Tabella 4.3: Modulazione dei valori nel campo Accept-Language.

Accept-Datetime. In questo campo è inserita la data della versione della risorsa richiesta. Le date vengono scritte seguendo lo standard "<nome-del-giorno>, <giorno> <mese> <anno> <ora>:<minuti>:<secondi> <fuso-orario>". Dato questo formato di data si è deciso di sfruttare tutti i campi tranne il primo e l'ultimo: il primo perché dipende dagli altri e l'ultimo perché se le richieste partono dalla stessa macchina è impensabile che differiscano per il fuso orario.

Come fatto anche per gli altri campi, per ogni dato si è considerata la potenza di 2 più alta minore del numero di possibili valori, e a quel numero di elementi sono state associate delle stringhe binarie. Riportando degli esempi iniziamo dal giorno. Il mese di febbraio è il più corto con 28 giorni e la potenza di 2 più grande minore di 28 è 16 (2^4), perciò usando i numeri da 1 a 16 possiamo steganografare stringhe binarie di 4 bit. Lo stesso identico ragionamento è stato fatto per ore, minuti e secondi. Per quanto riguarda i mesi, essendo 12, sono stati presi in considerazione soltanto i primi 8 (2^3), mentre per gli anni si va dal 1991 al 2022, così da avere 32 (2^5) valori utilizzabili ed evitando date antecedenti all'avvento di internet. Il fuso orario come detto viene usato lo stesso, in particolare quello italiano CET, e il giorno della settimana viene ricavato una volta create la data. Naturalmente nella fase di decodifica questi due valori verranno ignorati. Considerando quindi stringhe di lunghezza 4 per il giorno e l'ora, 3 per il mese, 5 per l'anno, i minuti e i secondi, all'interno di una data è possibili

steganografare 26 bit. Nella Tabella 4.4 vengono riportati i valori della stringa binaria nulla, della stringa binaria composta da tutti "1" e di una casuale.

Valore	Bit
Thu, 01 Jan 1991 00:00:00 CET	00000000000000000000000000000000
.....
Wed, 05 Feb 2003 12:24:13 CET	01000010110011001100001101
.....
Thu, 16 Aug 2022 16:32:32 CET	111111111111111111111111111111

Tabella 4.4: Modulazione dei valori nel campo Accept-Datetime.

From. Il campo From ha la funzionalità di contenere un recapito di chi ha effettuato la richiesta, questo perché nel caso ci fossero dei problemi il server avrebbe un recapito da contattare per risolverli. Lo standard prevede che questo contatto sia un classico indirizzo e-mail.

Alla luce di ciò abbiamo deciso di scaricare i database dei nomi DNS più utilizzati negli ultimi anni in America², unirli e creare un dizionario di 32768 nomi, così che ogni elemento rappresenti una stringa binaria di 15 bit ($2^{15}=32768$). Invece per quanto riguarda il dominio delle e-mail ne abbiamo scelti 8 tra i più utilizzati e associato stringhe binarie di 3 bit. I domini scelti e i relativi bit associati sono riportati in Tabella 4.5.

Valore	Bit
gmail.com	000
outlook.com	001
yahoo.com	010
proton.me	011
virgilio.it	100
libero.it	101
email.it	110
mail.com	111

Tabella 4.5: Modulazione dei valori dei domini del campo From.

²<https://www.ssa.gov/oact/babynames/limits.html>.

If-Match. Questo campo viene utilizzato principalmente insieme al metodo POST per verificare se la risorsa richiesta sia stata modificata oppure no, perché al suo interno è previsto un identificativo, o una lista di essi, e la richiesta verrà evasa solo se uno di questi combacia con uno salvato sul server.

Ci sono diverse strategie per calcolare un identificativo di una risorsa ma la più utilizzata è quella che utilizza l'hash. In questo caso abbiamo ipotizzato che venga utilizzato lo SHA-256 per calcolare il digest, e che ne vengono inseriti due nella richiesta. Lo SHA-256 produce un digest di 256 bit perciò abbiamo semplicemente tradotto 256 bit, del messaggio segreto da inviare, in valori esadecimali e poi inseriti nel campo. Inserendo due valori steganografiamo 512 bit all'interno di questo campo. Naturalmente se la quantità di dati rimasti da inviare fosse minore di 256 bit, questo campo e il relativo campo Range, descritto nel paragrafo successivo, non vengono utilizzati.

Range. Come riportato nel paragrafo precedente questo campo sarà presente nella richiesta inviata solo se presente anche il campo *If-Match*, in quanto nella maggior parte delle occasioni vengono utilizzati insieme, proprio per richiedere (o modificare) una parte di una certa risorsa. In questo campo vengono inseriti i due ipotetici intervalli di byte interessati.

Una risorsa può essere formata da molti byte per cui abbiamo ritenuto plausibile introdurre due intervalli lavorando con valori tra 0 e 1023 (2^{10} valori). Così facendo ad ogni numero viene associata una stringa binaria di 10 bit, quindi delimitando ogni intervallo con due valori, riusciamo a steganografare all'interno di questo campo 40 bit. Prima di tutto viene ricavato il primo valore dai 10 bit, poi il secondo dagli altri 10, a quest'ultimo valore viene aggiunto il primo, così da non avere un intervallo non logico dove la fine è minore dell'inizio. Stessa strategia viene usata per il calcolo del secondo intervallo. Nella Tabella 4.6 è riportata l'associazione del primo, dell'ultimo e di un valore casuale con la rispettiva stringa binaria.

Valore	Bit
0	0000000000
.....
673	1010100001
.....
1023	1111111111

Tabella 4.6: Modulazione dei valori nel campo Range.

TE. Questo campo standard viene inserito nelle richieste affinché il client comunichi al server quale codfiche di trasferimento è disposto ad accettare. Nella versione 2 e 3 del protocollo HTTP, questo campo viene accettato solo se è impostato sul valore *trailers*³. Essendo un campo fondamentale nelle richieste, e quindi sempre presente, è stato deciso di utilizzare anche questo per steganografare delle informazioni. Dati i pochi valori possibili abbiamo optato per sfruttare i 4 più utilizzati e associare loro stringhe binarie di 2 bit. Nella Tabella 4.7 è riportata l'associazione tra valore e bit rappresentati.

Valore	Bit
compress	00
deflate	01
gzip	10
trailers	00

Tabella 4.7: Modulazione dei valori nel campo TE.

User-Agent. Il campo *User-Agent* è obbligatorio nelle comunicazioni e sta ad indicare la versione di client che effettua la richiesta. Questa informazione è molto utile al server per comprendere con quale software si sta interfacciando perché da questa può

³<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/TE>.

ricavare quali dati è in grado di elaborare, e quindi implementare delle ottimizzazioni oltre a quelle contenute negli altri campi.

Riguardo a questo campo abbiamo deciso di non procedere con una modulazione, ma di inserire in ogni pacchetto la stringa identificativa di una versione di Mozilla in quanto essere il browser più diffuso tra le varie piattaforme. Vedere uscire richieste dalla stessa macchina con diversi User-Agent, e in breve tempo, avrebbe sicuramente sollevato alcune anomalie.

4.3.5 Strategie comunicative

Per allertare il server che si sta per iniziare l'invio di un file viene effettuata una richiesta con il campo *Accept* impostato sul valore *application/zip*, mentre al termine dell'invio, la richiesta finale avrà il valore *application/rtf*. Naturalmente questi due valori non sono utilizzati nella modulazione del campo *Accept* e sono riservati per questo scopo.

Le richieste inviate possono essere molteplici, anticipiamo che dai nostri test per inviare un'immagine di circa 100KB sono state necessarie più di 1300 richieste, motivo per cui abbiamo ritenuto necessario inserire dei ritardi tra una e l'altra per evitare di sollevare anomalie.

Dalla parte del server, data la facilità di implementare controlli sulla grandezza delle risposte e la loro ripetitività, abbiamo deciso di mettergli a disposizione un insieme di pagine html, che lui utilizzerà in modo casuale nelle risposte.

4.4 L'implementazione

In questa sezione andremo a descrivere le scelte tecniche relative all'implementazione del nostro prototipo. Essendo composto da due eseguibili, verrà primo illustrato quello relativo al client e poi quello del server. Entrambi sono stati scritti nel linguaggio Python, ritenuto più versatile e dal quale è possibile ottenere eseguibili per ogni sistema operativo.

4.4.1 Client

L'eseguibile del client è rinominato *sender.py*, in quanto il suo compito principale è quello di steganografare, e quindi possiamo affermare anche di esfiltrare, un file attraverso le richieste HTTP.

Operazioni preliminari. L'esecuzione inizia dalla funzione *main* la quale inizialmente avvia delle operazioni preliminari quali: la stampa del banner del tool a terminale e la creazione dei dizionari usati nei campi Range e From. Per quest'ultimo in particolare è necessario che all'interno della stessa cartella sia presente il file di testo *db_names.txt* contenente i nomi di persona più utilizzati in America, uno per riga. Per effettuare i nostri test abbiamo prima scaricato i vari file relativi agli anni dal 1980 al 2021⁴ e successivamente li abbiamo uniti gestendo i duplicati e ignorando alcune informazioni come sesso e numero di persone. Queste operazioni sono state automatizzate tramite un altro script Python.

Parametri. Per terminare vengono analizzati i parametri passati a riga di comando. Ricordiamo che questo eseguibile necessita dell'URL e del percorso del file o directory da inviare, senza i quali stampa il messaggio di Help e forza l'uscita. Se tutte e due le informazioni sono passate nel modo corretto le salva in due variabili globali e procede con l'esecuzione.

L'inizio. Arrivati a questo punto avvia l'invio vero e proprio dei dati, come descritto nella Sezione 4.3.5, la comunicazione inizia con una richiesta il cui valore del campo *Accept* è impostato su *application/zip* e gli altri campi non inseriti o se necessari vengono lasciati i valori di default.

L'invio. Successivamente viene avviata la funzione *send_files()* che si occupa di leggere il path, stabilire se è una directory o un file, creare le richieste e inviarle. Il file verrà aperto in modalità lettura binaria, ne verrà letta la dimensione e, sapendo che il massimo di byte steganografati all'interno della richiesta è 75, verrà stampata a terminale una stima del numero di richieste necessarie per inviarlo. Questa operazioni,

⁴<https://www.ssa.gov/oact/babynames/limits.html>.

nel caso si passasse una directory, vengono eseguite per ogni file, anche quelli nascosti, al suo interno.

Dopodiché si procede con la lettura dei blocchi di bit del file, per ognuno dei quali, dopo avergliene aggiunti 8 all'inizio che ne indicano la lunghezza, vengono create le rispettive richieste. A questo punto il processo si basa sulla stringa binaria ottenuta dal file che viene utilizzata da ogni funzione per inserire il rispettivo campo, con il giusto valore, all'interno della richiesta da inviare. Per ogni campo è presente un dizionario dove la chiave è la stringa binaria e il valore associato è l'informazione che verrà scritta nella richiesta.

Il primo analizzato è il campo *Accept*, il quale è in grado di steganografare 4 bit, perciò vengono letti i primi 4 bit del blocco, si ricava il valore associato nel dizionario e si inserisce il campo nella richiesta in fase di costruzione. Lo stesso identico procedimento viene utilizzato nel campo successivo *Accept-Encoding* e nell'ultimo *TE*.

Per il valore da inserire in *Accept-Language* viene effettuato un controllo ulteriore sulla quantità dei bit da inviare, perché questo campo ha la possibilità di steganografare fino a 3 bit. Si è deciso di avere almeno una intestazione con questa possibilità per gestire file di qualsiasi lunghezza, in quanto anche se l'ultima stringa binaria non rientra perfettamente in un campo, viene generata un'altra richiesta dove è presente solo questo campo così da inviare il bit, o i due bit, rimanenti.

Per la data da inserire in *Accept-Datetime*, prima vengono analizzati i bit e ricavati i valori che andranno a comporre la data, poi viene ricavato il giorno della settimana e inserito il fuso orario. Il procedimento visto per l'intestazione *Accept* viene eseguito per: il giorno, il mese, l'anno, l'ora, i minuti e i secondi, ad ognuno dei quali è associato il rispettivo dizionario.

Procedimento molto simile è quello per il campo *From* il quale prevede due fasi: i primi 15 bit andranno a determinare il nome mentre gli altri 3 il dominio. Infine quindi si avranno mail composte da *nome@dominio.com*.

L'intestazione *If-Match* deve contenere dei valori esadecimali, verosimili ad output dell'algoritmo SHA-256, perciò vengono letti 256 bit, considerati come intero e poi convertiti in esadecimale.

Per il campo *Range* non vengono utilizzati dizionari, vengono presi in considerazione i 40 bit, se presenti, altrimenti 20, e tradotti nei relativi interi. A questo punto per creare degli intervalli consistenti al secondo valore viene sommato il valore

del primo, al terzo quello del secondo e così via. In questa maniera non si hanno situazioni dove la fine è minore dell'inizio dell'intervallo.

Al termine viene inviata la richiesta appena creata ma nel caso la lunghezza non sia compatibile con la stringa binaria letta dal file, alcuni bit non verranno inviati e si procederà con la creazione di una nuova richiesta in modo ciclico finché l'intero blocco non sia stato completamente inviato. Dopodiché si passa a leggere il successivo blocco ma per come è stato progettato il funzionamento questo comportamento si dovrebbe avere solo nell'ultimo step. Ad ogni passaggio verrà aggiornato a video il numero di richiesta attualmente inviata così che l'utente possa avere una stima della percentuale di completamento raggiunta. Nella Figura 4.3 è possibile osservare una delle varie richieste che viene inviata durante un trasferimento di un file.

```
GET https://localhost:8000/
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36 Edg/109.0.1518.70
Accept-Encoding: gzip, deflate
Accept: image/gif
Connection: keep-alive
Accept-Language: it;q=0.9, en-US, en-GR;q=0.7
Accept-Datetime: Sat, 10 Jan 2015 05:23:22 CET
From: brendon@outlook.com
If-Match: 189DDD5BD99A189D59185BDB9D989CDA5BDD989B999CDBD8991CDBD89D9A991C,
D8589D9E9CDAC2989D9A9ADCD89D9AD91E82989D9A991ADCDD999AD91CC2991C
Range: 879-1272, 2142-2426
TE: trailers

Sendind request n° 6 of the approximately 8 expected
```

Figura 4.3: Esempio di richiesta inviata dal sender.

La fine. Alla fine, per comunicare al server il termine delle operazioni di esfiltrazione, come descritto nella Sezione 4.3.5, viene inviata la richiesta con il campo *Accept* impostato su *application/rtf*. Successivamente si stampa a video il totale delle richieste usate effettivamente nell'intero processo, il cui numero era stato gestito in una variabile aggiornata ad ogni invio di una richiesta.

4.4.2 Server

L'eseguibile del server è nominato *receiver.py* ed è a tutti gli effetti un server web. Questo si metterà in ascolto su una porta, passata come parametro a riga di coman-

do, e dopo aver analizzato le richieste in entrata, ricostruirà il file steganografato rinominato in base al secondo parametro passatogli.

Operazioni preliminari. Anche l'esecuzione del server si basa su una funzione *main* la cui prima operazione è quella di stampare il banner a terminale. Quest'ultimo è una semplice scritta "HTTP_Server" in formato ASCII art. Subito dopo, nella stessa maniera del client, si va a creare i dizionari relativi ai nomi di persona e ai range, dato che gli altri sono già inseriti nel codice. La differenza con quelli utilizzati dal client è che in questo caso la chiave è l'informazione che troveremo nella richiesta mentre il valore associato è rappresentato dalla stringa binaria.

L'inizio. A questo punto viene prima istanziato e poi avviato il server web tramite la classe *HTTPServer* e il relativo metodo *serve_forever()*. Per effettuare queste operazioni si è utilizzata la libreria *requests*⁵. Se tutto è andato a buon fine stamperà a video la porta su cui è in ascolto e il nome del file che produrrà. Questo sarà il segnale che è pronto per ricevere richieste e in caso decodificare le informazioni contenute negli header. Come già più volte riportato, le operazioni di estrazioni del file inizieranno dopo la ricezione di una richiesta con il valore del campo *Accept* impostato su *application/zip*. L'utente verrà informato tramite un messaggio stampato nel terminale che è stata effettivamente ricevuta e che è stato creato il file di destinazione dove scrivere i dati decodificati.

L'estrazione. Basandosi solo sulle richieste GET, in questo prototipo abbiamo definito soltanto il metodo *do_GET()*. Ad ogni richiesta di questo tipo prima di tutto verranno salvati gli header in una variabile, poi si procede nello scegliere la pagina html da inviare come risposta ed infine si analizzano gli header per verificare se contengano delle informazioni.

Abbiamo cercato di replicare il funzionamento di un vero server web che risponde fornendo pagine html, di diversa grandezza, senza soffermarsi sul loro contenuto effettivo. Dal sito Kaggle abbiamo scaricato un database di pagine internet⁶ destinato al test di alcune reti neurali, le abbiamo rinominate da 0 a 340 e inserite nella cartella

⁵<https://pypi.org/project/requests/>.

⁶<https://www.kaggle.com/datasets/uciml/identifying-interesting-web-pages>.

archive/pages che si troverà nella stessa posizione del file *receiver.py*. Ad ogni richiesta il server produrrà la risposta con codice 200 e nel body inserirà in modo casuale una di queste pagine.

Infine analizza gli header salvati in precedenza in cerca di informazioni. Effettuando il procedimento inverso rispetto al client, legge i dati delle richieste e tramite i dizionari di cui dispone ne ricava le stringhe binarie. Queste vengono salvate in modo consecutivo in una variabile. Al termine dell’analisi di ogni richiesta si analizza il primo byte della stringa binaria costruita, in quanto rappresenterà la lunghezza del blocco letto e inviato, e se il totale degli altri bit è superiore al valore letto in precedenza si procede con la scrittura di quei byte nel file di destinazione. Durante questa fase viene aggiornato nel terminale il numero di richieste ricevute.

La fine. Analogamente alla fase iniziale, quando si riceve la richiesta con il campo *Accept* impostato su *application/rtf*, che rappresenta la fine di invio dei dati, il server procederà con l’eventuale scrittura degli ultimi byte nel file e comunicherà, sempre tramite un messaggio nel terminale, che le operazioni si sono concluse e che il file è pronto.

4.4.3 Librerie utilizzate

Nei prossimi due paragrafi riporteremo le librerie Python utilizzate nello sviluppo dei nostri eseguibili e il motivo della loro applicazione. *Requests*, *urllib3* e *pandas* non essendo librerie standard vanno installate precedentemente nell’ambiente in cui si andranno ad effettuare i test.

Client Per sviluppare *sender.py*, cioè la parte client che invia le richieste, sono state importate le seguenti librerie:

- *requests*: usata per creare e inviare le richieste HTTP. Questa libreria può essere usata sia per HTTP che per HTTPS;
- *sys*: per effettuare la chiusura del programma nel caso in cui i parametri passati non siano corretti;

- *getopt*: questa fornisce una serie di metodi per la gestione dei parametri passati a riga di comando;
- *pandas*: fornisce una grande quantità di metodi per la gestione di dati e variabili, in questo caso particolare è stata utilizzata per ricavare un oggetto *date_time* da una stringa, per calcolarne il giorno della settimana;
- *time e random*: per inserire dei ritardi casuali tra una richiesta e l'altra con il metodo *sleep*;
- *os*: per leggere le informazioni del file da inviare e calcolare una stima del numero di richieste necessarie per inviarlo;
- *urllib3*: per disabilitare i warning nel caso si utilizzasse il protocollo HTTPS. Questo perché non procediamo con la verifica del certificato fornito dal server e ciò solleva degli avvertimenti.

Server Invece per quanto riguarda *receiver.py*, cioè il server web in ascolto e in grado di decodificare le informazioni, abbiamo utilizzato:

- *os*: per implementare le operazioni riguardanti la creazione della cartella dove inserire i file estratti;
- *sys*: anche in questo caso viene usata per effettuare la chiusura del programma nel caso in cui i parametri passati non siano corretti;
- *HTTPserver e BaseHTTPRequestHandler*: per sfruttare tutte quelle funzionalità riguardanti l'implementazione e la gestione del server web;
- *random*: per generare un numero casuale che rappresenterà la pagina html da usare nella risposta;
- *getopt*: come per il client questa libreria viene utilizzata per la gestione dei parametri passati a riga di comando;
- *ssl*: per implementare le funzionalità relativa al protocollo HTTPS e alla gestione dei certificati.

4.4.4 Distribuzione

Per raggiungere il nostro obiettivo di creare un prodotto applicabile sulla maggior parte dei sistemi abbiamo eseguito una ulteriore fase di sviluppo. Oltre ad aver scelto Python⁷, per la sua versatilità, abbiamo deciso di procedere nel creare due eseguibili *.exe* così da poter essere lanciati su macchine Windows. Omettiamo i procedimenti per la creazione di eseguibili per altre piattaforme perché sono identici al seguente, l'unica differenza è che deve esser eseguito sullo stesso sistema operativo per il quale si vuole creare l'eseguibile.

Per svolgere questa operazione abbiamo utilizzato lo strumento *pyinstaller*⁸. Questo programma dedicato a Python va scaricato tramite il comando *pip install pyinstaller* e una volta chiamato, *pyinstaller myscript.py*, andrà a produrre due cartelle, *build* e *dist* e il file *myscript.spec*. Se utilizzato senza parametri nella cartella *dist* troveremo l'eseguibile e le librerie utilizzate in formato *.dll*, mentre noi andremo a sfruttare l'opzione -F (OneFile) così da avere un unico eseguibile con all'interno tutto il necessario per il funzionamento.

Nelle Figure 4.4 e 4.5 è rappresentata la struttura delle cartelle da noi distribuite e come è possibile osservare abbiamo ignorato i file e cartelle creati da *pyinstaller* prendendo in considerazione solo il prodotto finale. Per logica ovviamente abbiamo separato l'ambiente server da quello client. Nella cartella del receiver troviamo:

- *archive*: cartella contenente le pagine web da utilizzare come risposte,
- *cert*: cartella contenente i certificati per SSL,
- *db_names.txt*: file contenente i nomi per creare il dizionario utilizzato nel campo From,
- *file_received*: la cartella che si creerà per memorizzare i file ricevuti,
- *receiver.py*: il codice sorgente,
- *receiver.exe*: l'eseguibile.

Invece la cartella del sender è così composta:

⁷Sviluppato con l'ultima versione stabile 3.10.10.

⁸<https://pyinstaller.org/en/stable/>.

- *db_names.txt*: file contenente i nomi per creare il dizionario utilizzato nel campo From,
- *message.txt*: file di testo utilizzato per effettuare i test,
- *test_send*: cartella utilizzata per effettuare i test,
- *sender.py*: il codice sorgente,
- *sender.exe*: l'eseguibile.



Figura 4.4: Struttura della cartella del server.

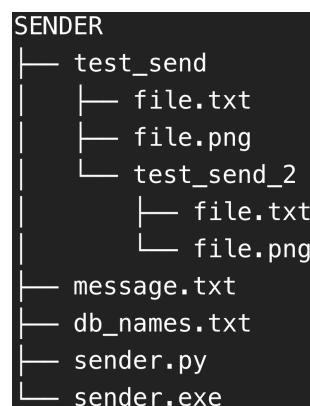


Figura 4.5: Struttura della cartella del client.

Capitolo 5

Test

Dopo aver concluso lo sviluppo del prototipo abbiamo ritenuto fondamentale effettuare alcuni test per confermare l'effettivo raggiungimento del nostro obiettivo. Tra le varie possibilità abbiamo deciso di procedere con l'analisi di un file *.pcap* contenente diverse tipologie di traffico, tra cui quello generato dai nostri eseguibili. Nelle seguenti sezioni descriveremo gli strumenti, le tecniche e le procedure utilizzate per condurre i test sul nostro PoC.

5.1 Architettura

Prima di tutto abbiamo iniziato a creare la nostra infrastruttura all'interno della quale generare e catturare il traffico incriminato. Per semplicità tutte le attività sono state svolte su una macchina con sistema operativo Ubuntu 20.04.5 LTS 64 bit e con le seguenti componenti:

- processore: Intel Core i3-10100 3.60GHz;
- memoria: 8GB 3000MHz DDR4 CL15;
- scheda madre: ASUS Prime H410M-E LGA1200 micro-ATX;
- storage: SSD interno 256GB Rocket NVMe PCIe M.2 2280.

All'interno di questa è stato installato il software VirtualBox 6.1.38_Ubuntu r153438 perché le due macchine comunicanti saranno virtuali.

Procedendo con l'analisi di un file *.pcap* non abbiamo riservato particolare attenzione al sistema operativo delle due macchine, anche perché da Python si possono ottenere eseguibili per qualsiasi piattaforma e una volta avviati il traffico di rete generato sarà il medesimo nella maggior parte dei casi. Alla luce di ciò abbiamo installato Xubuntu, basato su Ubuntu 22.10, nelle due macchine in quanto rappresenta un'implementazione che sfrutta poche risorse fisiche e quindi permette un utilizzo senza ritardi o cadute dei due sistemi.

Su una macchina verrà avviato *receiver.py*, la parte server, e sull'altra *sender.py*, la parte client del prototipo. La prima sarà caratterizzata dall'indirizzo IP 192.168.1.113 mentre la seconda 192.168.1.110. Queste informazioni sono visibili negli screen riportati nelle Figure 5.1 e 5.2.

```
michele@sender:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
              ether 02:42:2e:c0:ea:f7 txqueuelen 0 (Ethernet)
                    RX packets 0 bytes 0 (0.0 B)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 0 bytes 0 (0.0 B)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.110 netmask 255.255.255.0 broadcast 192.168.1.255
              inet6 fe80::35e:32bf:ec79:63db prefixlen 64 scopeid 0x20<link>
                    ether 08:00:27:0f:45:72 txqueuelen 1000 (Ethernet)
                    RX packets 28046 bytes 40770287 (40.7 MB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 9938 bytes 808252 (808.2 KB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
              inet6 ::1 prefixlen 128 scopeid 0x10<host>
                    loop txqueuelen 1000 (Loopback locale)
                    RX packets 265 bytes 27990 (27.9 KB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 265 bytes 27990 (27.9 KB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 5.1: Output del comando *ifconfig* lanciato sulla macchina client.

```
michele@receiver:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
              ether 02:42:3c:93:26:9d txqueuelen 0 (Ethernet)
                    RX packets 0 bytes 0 (0.0 B)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 0 bytes 0 (0.0 B)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.113 netmask 255.255.255.0 broadcast 192.168.1.255
              inet6 fe80::cf65:2d1e:ebdd:b90a prefixlen 64 scopeid 0x20<link>
                    ether 08:00:27:3a:81:d2 txqueuelen 1000 (Ethernet)
                    RX packets 101 bytes 42186 (42.1 KB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 106 bytes 15287 (15.2 KB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
              inet6 ::1 prefixlen 128 scopeid 0x10<host>
                    loop txqueuelen 1000 (Loopback locale)
                    RX packets 181 bytes 15469 (15.4 KB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 181 bytes 15469 (15.4 KB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 5.2: Output del comando *ifconfig* lanciato sulla macchina server.

5.2 Cattura del traffico

Una volta che l’architettura è stata preparata abbiamo installato il software per catturare il traffico, e data la sua grande versatilità abbiamo scelto Wireshark¹. Dato che si lavora con delle macchine virtuali, sempre per una questione di praticità, abbiamo deciso di posizionarlo sul pc principale (quello con sistema operativo Ubuntu 20.04) in quanto è in grado comunque di intercettare e analizzare i pacchetti scambiati tra i due sistemi.

È stata installata la versione 3.2.3 e avviata con diritti di root in ascolto su tutte le interfacce. Successivamente è stato avviato il server con il comando *python3 receiver.py -p 443 -f file_output* e inviato un file dal client tramite il comando *python3*

¹<https://www.wireshark.org>.

`sender.py -u https://192.168.1.113:443 -f message.txt`. Oltre a questa operazione, sempre con la cattura ancora attiva, è stata effettuata un'altra serie di operazioni classiche come navigazione internet, videochiamate e ping. Il file trasmesso era di tipo `.txt` con una dimensione di 459byte che ha prodotto in totale 9 richieste. Infine il traffico catturato è stato salvato con il formato `.pcap`.

5.3 Analisi

Il software tramite il quale analizzare il traffico catturato è stato scelto in base alla versatilità e alla diffusione, pertanto la scelta è ricaduta su l'ultima versione disponibile di Suricata², ovvero la 6.0.10. Questo è un software di analisi di rete open source ad alte prestazioni e un sistema di rilevamento di minacce che viene utilizzato nella maggior parte delle organizzazioni private e pubbliche. Questo programma basa i propri controlli sull'applicazione di un determinato insieme di regole, le quali possono trovarsi organizzate in pacchetti già pronti sulla rete oppure scritte in maniera personalizzata e poi aggiunte. Un punto di forza questo software è che risulta compatibile anche con i pacchetti di regole Snort³, il più diffuso Intrusion Prevention System al mondo. Alcuni suoi pacchetti per essere scaricati richiedono un pagamento quindi abbiamo condotto i nostri test con le versioni free più recenti disponibili per Snort 2.9 e Snort 3.0. In particolare nel software Suricata sono stati aggiunti i pacchetti `snortrules-snapshot-2983.tar.gz`, `snortrules-snapshot-31470.tar.gz` e `snort3-community-rules.tar.gz` in questo ordine, così che se un sottoinsieme di regole fosse presente in più pacchetti questo fosse presente con la sua ultima versione.

Una volta aggiunte le regole a Suricata, lo abbiamo avviato sfruttando la sua funzione di analisi di file `.pcap`, il quale produrrà nella stessa posizione altri 3 file di log contenenti dati e risultati dell'analisi effettuata: `fast.log`, `stats.log` e `suricata.log`.

In particolare, all'interno del file `fast.log`, sono contenuti i warning generati accompagnati dal livello di priorità e da una breve descrizione. La classifica di priorità usata da Suricata va da 1, gli alert più critici, a 3, i meno rilevanti. Nel nostro caso sono stati registrati solo allarmi di livello 3 a causa del valore di checksum non valido e altri inerenti il protocollo IGMP. Effettuando una ricerca sulla possibile

²<https://suricata.io>.

³<https://www.snort.org>.

causa abbiamo riscontrato che è un caso che si ripete con frequenza e che dipende dal software utilizzato per implementare le macchine virtuali oppure dalla scheda di rete. Dato che all'interno del traffico analizzato non c'erano solo quei pacchetti specifici ma erano presenti anche comunicazioni all'esterno di altri tipi, possiamo ricondurre i problemi al software VirtualBox. Nella Figura 5.3 sono presenti le righe riguardanti gli alert sollevati dal checksum non valido, in particolare alle righe 1,2,3,4 e 6. Sottolineiamo che solo su 5 pacchetti è stata riscontrata questa anomalia, a fronte dei 9 necessari per inviare il *.txt* utilizzato nel test.

```

1 03/13/2023-17:09:18.698224 [**] [1:2200074:2] SURICATA TCPv4 invalid checksum [**] [Classification: Generic Protocol Command Decode]
[Priority: 3] {TCP} 192.168.1.113:443 -> 192.168.1.110:43510
2 03/13/2023-17:09:18.671349 [**] [1:2200074:2] SURICATA TCPv4 invalid checksum [**] [Classification: Generic Protocol Command Decode]
[Priority: 3] {TCP} 192.168.1.113:443 -> 192.168.1.110:43488
3 03/13/2023-17:09:18.705164 [**] [1:2200074:2] SURICATA TCPv4 invalid checksum [**] [Classification: Generic Protocol Command Decode]
[Priority: 3] {TCP} 192.168.1.113:443 -> 192.168.1.110:43512
4 03/13/2023-17:09:18.719485 [**] [1:2200074:2] SURICATA TCPv4 invalid checksum [**] [Classification: Generic Protocol Command Decode]
[Priority: 3] {TCP} 192.168.1.113:443 -> 192.168.1.110:43534
5 03/13/2023-17:09:25.527461 [**] [1:51035:1] POLICY-OTHER IP option strict source routing attempt [**] [Classification: Generic Protocol
Command Decode] [Priority: 3] {IGMP} 192.168.1.254:0 -> 224.0.0.1:0
6 03/13/2023-17:09:18.712558 [**] [1:2200074:2] SURICATA TCPv4 invalid checksum [**] [Classification: Generic Protocol Command Decode]
[Priority: 3] {TCP} 192.168.1.113:443 -> 192.168.1.110:43526

```

Figura 5.3: Alert sollevati riguardanti il checksum non valido.

Capitolo 6

Conclusioni e lavori futuri

In questa tesi abbiamo affrontato un'analisi completa sulla steganografia di rete, partendo dalla descrizione del concetto e di tutte le caratteristiche a lui correlate, si è arrivati ad esaminare nel dettaglio le varie proposte fino a quelle più recenti. La network steganography rappresenta la frontiera più moderna della steganografia ed anche quella caratterizzata da più ambiti di applicazione; il tutto è dimostrato dai numerosi articoli scientifici pubblicati di recente a riguardo.

Non sempre le proposte sono accompagnate da una implementazione pratica e viceversa, ma nonostante ciò, abbiamo dedicato un capitolo ai canali coperti realizzati, già disponibili in rete. In aggiunta a ciò abbiamo ritenuto necessario anche riservare delle sezioni alle tipologie di licenze, agli strumenti utili nello sviluppo e agli attacchi informatici realmente avvenuti che hanno utilizzato queste tecniche.

Nell'ultima parte di questo documento è stata descritta la nostra proposta per implementare un canale coperto. Nei Capitoli 4 e 5 sono riportate le motivazioni che ci hanno spinto a prendere determinate decisioni durante lo sviluppo e tutti i dettagli implementativi. Il nostro PoC si basa su richieste HTTP/HTTPS, nelle quali i valori delle intestazioni sono modulati affinché al loro interno siano steganografate le informazioni di nostro interesse. Il traffico generato è stato analizzato risultando ben formattato e successivamente è stato sottoposto ad un IDS all'interno del quale non ha sollevato alcun alert di alta priorità, apparendo quindi come traffico innocuo.

Il primo aspetto che teniamo a sottolineare di questo prototipo è il ruolo svolto nella dimostrazione dell'applicabilità pratica della nostra idea. Attraverso questo

canale coperto è possibile creare un comunicazione unidirezionale da client a server che non viene rilevata dai principali sistemi di monitoraggio della rete.

Il secondo aspetto, ma di certo non meno importante, è rappresentato dall'ampia possibilità di sviluppo. Utilizzare la stessa strategia comunicativa nelle risposte del server porterebbe alla creazione di un canale bidirezionale, applicabile anche per scopi di *command and control*. Ovvio che la modulazione dei valori da noi proposta è solo il punto di partenza e può essere soggetta a diversi processi di ottimizzazione, che si ripercuoterebbero di conseguenza anche sulle prestazioni del canale coperto. Inoltre per ottenere uno strumento più affidabile è necessaria una fase di test più approfondita con altri analizzatori di traffico come RITA¹. Aspetto non affrontato, ma fondamentale per lo sviluppo di questa tecnica, è lo studio del suo comportamento in presenza di proxy lungo il percorso dei pacchetti. Una sua ottimizzazione anche sotto questo aspetto la renderebbe realmente applicabile ad un vero server web che fornisce servizi e pagine web, garantendo così il trasferimento di una grande quantità di dati nascosti con una sequenza e quantità di richieste veramente molto simili a quelle che si registrerebbero in una normale navigazione. Immaginando questo scenario i possibili limiti di applicazione si ampliano a dismisura e la difficoltà di rilevamento cresce, portando alla necessità di uno sviluppo degli attuali sistemi di prevenzione che probabilmente solo con l'ausilio dell'intelligenza artificiale saranno in grado di prevenire tecniche come questa.

¹<https://github.com/activecm/rita>

Bibliografia

- [1] Christopher Abad. Ip checksum covert channels and selected hash collision. *USA, University of California*, 2001.
- [2] Sana Adnan Abbas. Internet protocol (ip) steganography using modified particle swarm optimization (mpso) algorithm. *Diyala J Pure Sci*, 14:220–236, 2018.
- [3] Osamah Ibrahem Abdullaziz, Vik Tor Goh, Huo-Chong Ling, and KokSheik Wong. Network packet payload parity based steganography. In *2013 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (CSUDET)*, pages 56–59. IEEE, 2013.
- [4] Jibi Abraham, Mamatha Jadhav V, et al. Design of transport layer based hybrid covert channel detection engine. *arXiv preprint arXiv:1101.0104*, 2010.
- [5] Kamran Ahsan. Covert channel analysis and data hiding in tcp/ip. *MA Sc. thesis, Dept. of Electrical and Computer Engineering, University of Toronto*, 2002.
- [6] Kamran Ahsan and Deepa Kundur. Practical data hiding in tcp/ip. In *Proc. Workshop on Multimedia Security at ACM Multimedia*, volume 2, pages 1–8. ACM Press New York, 2002.
- [7] Sara Akhtari, Neda Moghim, and Mojtaba Mahdavi. Binary middleman covert channel in exor protocol. In *2017 Iranian conference on electrical engineering (ICEE)*, pages 1469–1474. IEEE, 2017.

- [8] Mahdi Akil, Luigi V Mancini, and Daniele Venturi. Multi-covert channel attack in the cloud. In *2019 Sixth International Conference on Software Defined Systems (SDS)*, pages 160–165. IEEE, 2019.
- [9] Cristina Alcaraz, Giuseppe Bernieri, Federica Pascucci, Javier Lopez, and Roberto Setola. Covert channels-based stealth attacks in industry 4.0. *IEEE Systems Journal*, 13(4):3980–3988, 2019.
- [10] AllAfrica. Ethiopia: Govt denies banning skype and other internet communication services. *AllAfrica*, 2012. URL <https://allafrica.com/stories/201206250202.html>.
- [11] Pierre Allix. Covert channels analysis in tcp/ip networks. *IFIPS School of Engineering, University of Paris-Sud XI, Orsay, France*, 14:15, 2007.
- [12] Daniel Alman. Http tunnels though proxies. *SANS Institute*, 2003.
- [13] Marios Anagnostopoulos and John André Seem. Another step in the ladder of dns-based covert channels: Hiding ill-disposed information in dnskey rrs. *Information*, 10(9):284, 2019.
- [14] Koundinya Anjan and Jibi Abraham. Behavioral analysis of transport layer based hybrid covert channel. In *International Conference on Network Security and Applications*, pages 83–92. Springer, 2010.
- [15] Linda Yunlu Bai, Yongfeng Huang, Guannan Hou, and Bo Xiao. Covert channels based on jitter field of the rtcp header. In *2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 1388–1391. IEEE, 2008.
- [16] Patryk Bąk, Jędrzej Bieniasz, Michał Krzemieński, and Krzysztof Szczygielski. Application of perfectly undetectable network steganography method for malware hidden communication. In *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*, pages 34–38. IEEE, 2018.
- [17] Diogo Barradas, Nuno Santos, and Luís ET Rodrigues. Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams. *Proc. Priv. Enhancing Technol.*, 2017(4):5–22, 2017.

- [18] Matthias Bauer. New covert channels in http: adding unwitting web browsers to anonymity sets. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 72–78, 2003.
- [19] Punam Bedi and Arti Dua. Network steganography using the overflow field of timestamp option in an ipv4 packet. *Procedia Computer Science*, 171:1810–1818, 2020.
- [20] Punam Bedi and Arti Dua. Network steganography using extension headers in ipv6. In *International Conference on Information, Communication and Computing Technology*, pages 98–110. Springer, 2020.
- [21] Vincent Berk, Annarita Giani, and George Cybenko. Detection of covert channel encoding in network packet delays. *Computer Science Technical Report*, 2005.
- [22] John Bethencourt, Jason Franklin, and Mary K Vernon. Mapping internet sensors with probe response attacks. In *USENIX security symposium*, pages 193–208, 2005.
- [23] L Bowyer. Firewall bypass via protocol steganography. *Network Penetration*, 2002.
- [24] Jon Brodkin. Iran reportedly blocking encrypted internet traffic. *arstechnica*, 2012. URL <https://arstechnica.com/tech-policy/2012/02/iran-reportedly-blocking-encrypted-internet-traffic/>.
- [25] Erik Brown, Bo Yuan, Daryl Johnson, and Peter Lutz. Covert channels in the http network protocol: Channel characterization and detecting man-in-the-middle attacks. *Journal of Information Warfare*, 9(3):26–38, 2010.
- [26] Bryan Burns, Dave Killion, Nicolas Beauchesne, Eric Moret, Julien Sobrier, Michael Lynn, Eric Markham, Chris Iezzoni, Philippe Biondi, Jennifer Stisa Granick, et al. *Security power tools*. " O'Reilly Media, Inc.", 2007.
- [27] Serdar Cabuk. *Network covert channels: Design, analysis, detection, and elimination*. PhD thesis, Purdue University, 2006.

- [28] Serdar Cabuk, Carla E Brodley, and Clay Shields. Ip covert timing channels: design and detection. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 178–187, 2004.
- [29] Serdar Cabuk, Carla E Brodley, and Clay Shields. Ip covert channel detection. *ACM Transactions on Information and System Security (TISSEC)*, 12(4):1–29, 2009.
- [30] Aniello Castiglione, Alfredo De Santis, Ugo Fiore, and Francesco Palmieri. Email-based covert channels for asynchronous message steganography. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 503–508. IEEE, 2011.
- [31] Aniello Castiglione, Michele Nappi, Fabio Narducci, and Chiara Pero. Fostering secure cross-layer collaborative communications by means of covert channels in mec environments. *Computer Communications*, 169:211–219, 2021.
- [32] S. Castro. Gray world team: Cooking channels. *hakin9 Magazine*, 2006.
- [33] Enrique Cauich, Roberto Gómez Cárdenas, and Ryouske Watanabe. Data hiding in identification and offset ip fields. In *International Symposium and School on Advances in Distributed Systems*, pages 118–125. Springer, 2005.
- [34] RC Chakinala, Abishek Kumarasubramanian, R Manokaran, Guevara Noubir, C Pandu Rangan, and Ravi Sundaram. Steganographic communication in ordered channels. In *International Workshop on Information Hiding*, pages 42–57. Springer, 2006.
- [35] Thomas Claburn. China’s github censorship dilemma. *InformationWeek*, 2013. URL <https://www.informationweek.com/social/china-s-github-censorship-dilemma>.
- [36] Thomas M Cover and Joy A Thomas. Wiley series in telecommunications. In *Elements of information theory*. John Wiley & Sons New York, 1991.
- [37] Scott Craver. On public-key steganography in the presence of an active warden. In *International Workshop on Information Hiding*, pages 355–368. Springer, 1998.

- [38] Mehiar Dabbagh. Covert channels in botnets, 2018.
- [39] daemon9. Project loki. *Phrack Magazine* 49, 1996.
- [40] daemon9. Project loki. *Phrack Magazine* 51, 1997.
- [41] Dhananjay M Dakhane and Vaibhav E Narawade. Reference model storage covert channel for secure communications. In *Advanced Computing Technologies and Applications*, pages 489–496. Springer, 2020.
- [42] Rennie Degraaf, John Aycock, and Michael Jacobson. Improved port knocking with strong authentication. In *21st Annual Computer Security Applications Conference (ACSA'05)*, pages 10–pp. IEEE, 2005.
- [43] Lucas Dixon, Thomas Ristenpart, and Thomas Shrimpton. Network traffic obfuscation and automated internet censorship. *IEEE Security & Privacy*, 14(6):43–53, 2016.
- [44] Arti Dua, Vinita Jindal, and Punam Bedi. Covert communication using address resolution protocol broadcast request messages. In *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 1–6. IEEE, 2021.
- [45] Robert Duncan and Jean Everson Martina. Steganographic message broadcasting using web protocols. In *proceedings of: Simposio Brasilerio de Seguranca (SBSeg 2010), Fortaleza, Brasil*, pages 61–70, 2010.
- [46] Alex Dyatlov and Simon Castro. Exploitation of data streams authorized by a network access control system for arbitrary data transfers: tunneling and covert channels over the http protocol. *Zugriff am unter http://dl.packetstormsecurity.net/papers/protocols/covert_paper.txt*, 2003.
- [47] Jonathan Edwards. *Covert channels in ad hoc networking: an analysis using the optimized link state routing protocol*. PhD thesis, Carleton University, 2012.
- [48] Adel El-Atawy and Ehab Al-Shaer. Building covert channels over the packet reordering phenomenon. In *IEEE INFOCOM 2009*, pages 2186–2194. IEEE, 2009.

- [49] Adel El-Atawy, Qi Duan, and Ehab Al-Shaer. A novel class of robust covert channels using out-of-order packets. *IEEE Transactions on Dependable and Secure Computing*, 14(2):116–129, 2015.
- [50] Hans-Georg Eßer and Felix C Freiling. Kapazittsmessung eines verdeckten zeitkanals ber http. Technical report, Technical Report TR-2005-10, Universitt Mannheim, 2005.
- [51] Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing web censorship and surveillance. In *11th USENIX Security Symposium (USENIX Security 02)*, 2002.
- [52] Sally Fincher, Janet Finlay, Sharon Greene, Lauretta Jones, Paul Matchen, John Thomas, and Pedro J Molina. Perspectives on hci patterns: concepts and tools. In *CHI'03 extended abstracts on Human factors in computing systems*, pages 1044–1045, 2003.
- [53] Gina Fisk, Mike Fisk, Christos Papadopoulos, and Joshua Neil. Eliminating steganography in internet traffic with active wardens. In *International workshop on information hiding*, pages 18–35. Springer, 2002.
- [54] Christopher R Forbes. *A new covert channel over RTP*. Rochester Institute of Technology, 2009.
- [55] Dario V Forte, Cristiano Maruti, Michele R Vetturi, and Michele Zambelli. Sec-syslog: An approach to secure logging based on covert channels. In *First International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'05)*, pages 248–263. IEEE, 2005.
- [56] Lilia Frikha, Zouheir Trabelsi, and Wassim El-Hajj. Implementation of a covert channel in the 802.11 header. In *2008 International Wireless Communications and Mobile Computing Conference*, pages 594–599. IEEE, 2008.
- [57] Alexei Galatenko, Alexander Grusho, Alexander Kniazev, and Elena Timonina. Statistical covert channels through proxy server. In *International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 424–429. Springer, 2005.

- [58] Abdughalil Ganivev, Obid Mavlonov, Baxtiyor Turdibekov, et al. Improving data hiding methods in network steganography based on packet header manipulation. In *2021 International Conference on Information Science and Communications Technologies (ICISCT)*, pages 1–5. IEEE, 2021.
- [59] Jinbao Gao, Yuanzhang Li, Hongwei Jiang, Lu Liu, and Xiaosong Zhang. An rtp extension for reliable user-data transmission over voip traffic. In *International Symposium on Security and Privacy in Social Networks and Big Data*, pages 74–86. Springer, 2019.
- [60] Steven Gianvecchio, Haining Wang, Duminda Wijesekera, and Sushil Jajodia. Model-based covert timing channels: Automated modeling and evasion. In *International Workshop on Recent Advances in Intrusion Detection*, pages 211–230. Springer, 2008.
- [61] John Giffin, Rachel Greenstadt, Peter Litwack, and Richard Tibbetts. Covert messaging through tcp timestamps. In *International Workshop on Privacy Enhancing Technologies*, pages 194–208. Springer, 2002.
- [62] John Giffin, Rachel Greenstadt, Peter Litwack, and Richard Tibbetts. Covert messaging through tcp timestamps. In Roger Dingledine and Paul Syverson, editors, *Privacy Enhancing Technologies*, pages 194–208, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36467-2.
- [63] C. Gray Girling. Covert channels in lan's. *IEEE Transactions on software engineering*, 13(2):292, 1987.
- [64] Virgil Gligor. Covert channel analysis of trusted systems. a guide to understanding. Technical report, NAVAL COASTAL SYSTEMS CENTER PANAMA CITY FL, 1993.
- [65] Ricardo André Santana Gonçalves. *A MAC layer covert channel in 802.11 networks*. PhD thesis, Monterey, California: Naval Postgraduate School, 2011.
- [66] T. Graf. Messaging over ipv6 destination options. Technical report, Swiss Unix User Group, 2003.

- [67] Theodore G Handel and Maxwell T Sandford. Hiding data in the osi network model. In *International Workshop on Information Hiding*, pages 23–38. Springer, 1996.
- [68] Van Hauser. Placing backdoors through firewalls. *WindowsSecurity.com, May*, 1999.
- [69] Corinna Heinz, Wojciech Mazurczyk, and Luca Caviglione. Covert channels in transport layer security. In *Proceedings of the European Interdisciplinary Cybersecurity Conference*, pages 1–6, 2020.
- [70] Carina Hefseling and Jörg Keller. Pareto-optimal covert channels in sensor data transmission. In *Proceedings of the 2022 European Interdisciplinary Cybersecurity Conference*, pages 79–84, 2022.
- [71] A Hintz. Covert channels in tcp and ip headers. *Presentation at DEFCON*, 10:16, 2002.
- [72] Christopher Hoffman, Daryl Johnson, Bo Yuan, and Peter Lutz. A covert channel in ttl field of dns packets. *International Conference on Security and Management*, 2012.
- [73] Russell Holloway and Raheem Beyah. Covert dcf: A dcf-based covert timing channel in 802.11 networks. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 570–579. IEEE, 2011.
- [74] Ningning Hou, Xianjin Xia, and Yuanqing Zheng. Cloaklora: A covert channel over lora phy. *IEEE/ACM Transactions on Networking*, 2022.
- [75] Amir Houmansadr and Nikita Borisov. Coco: coding-based covert timing channels for network flows. In *International Workshop on Information Hiding*, pages 314–328. Springer, 2011.
- [76] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. Rainbow: A robust and invisible non-blind watermark for network flows. In *NDSS*, volume 47, pages 406–422. Citeseer, 2009.

- [77] Amir Houmansadr, Giang TK Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 187–200, 2011.
- [78] Amir Houmansadr, Thomas J. Riedl, Nikita Borisov, and Andrew C. Singer. IP over voice-over-ip for censorship circumvention. *CoRR*, abs/1207.2683, 2012. URL <http://arxiv.org/abs/1207.2683>.
- [79] Tianwen Huang, Lejun Zhang, Xiaoyan Hu, and Xiaoying Lei. A data validation method based on ip covert channel packet ordering. In *2018 14th International Conference on Computational Intelligence and Security (CIS)*, pages 223–227. IEEE, 2018.
- [80] Mehdi Hussain and M Hussain. A high bandwidth covert channel in network protocol. In *2011 International Conference on Information and Communication Technologies*, pages 1–6. IEEE, 2011.
- [81] Liping Ji, Wenhao Jiang, Benyang Dai, and Xiamu Niu. A novel covert channel based on length of messages. In *2009 International Symposium on Information Engineering and Electronic Commerce*, pages 551–554. IEEE, 2009.
- [82] Liping Ji, Yu Fan, and Chuan Ma. Covert channel for local area network. In *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*, pages 316–319. IEEE, 2010.
- [83] Daryl Johnson, Peter Lutz, and Bo Yuan. Behavior-based covert channel in cyberspace. In *Intelligent decision making systems*, pages 311–318. World Scientific, 2010.
- [84] Emanuele Jones, Olivier Le Moigne, and J-M Robert. Ip traceback solutions based on time to live covert channel. In *Proceedings. 2004 12th IEEE International Conference on Networks (ICON 2004)(IEEE Cat. No. 04EX955)*, volume 2, pages 451–457. IEEE, 2004.

- [85] Manuel Kalmbach, Mathias Gotschlag, Tim Schmidt, and Frank Bellosa. Turbocc: A practical frequency-based covert channel with intel turbo boost. *arXiv preprint arXiv:2007.07046*, 2020.
- [86] Jörg Keller and Steffen Wendzel. Reversible and plausibly deniable covert channels in one-time passwords based on hash chains. *Applied Sciences*, 11(2):731, 2021.
- [87] Sahar Khamis and Katherine Vaughn. Cyberactivism in the egyptian revolution: How civic engagement and citizen journalism tilted the balance. *Arab Media and Society*, 14(3):1–25, 2011.
- [88] Zbigniew Kwecka. Application layer covert channel analysis and detection. *Undergraduate Project Dissertation, Napier University*, 2006.
- [89] Butler W Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.
- [90] Kevin Lamshöft and Jana Dittmann. Assessment of hidden channel attacks: Targetting modbus/tcp. *IFAC-PapersOnLine*, 53(2):11100–11107, 2020.
- [91] Kevin Lamshöft, Jonas Hielscher, Christian Krätzer, and Jana Dittmann. The threat of covert channels in network time synchronisation protocols. *Journal of Cyber Security and Mobility*, pages 165–204, 2022.
- [92] Donald C Latham. Department of defense trusted computer system evaluation criteria. *Department of Defense*, 198, 1986.
- [93] Christopher S Leberknight, Mung Chiang, Harold Vincent Poor, and Felix Wong. A taxonomy of internet censorship and anti-censorship. In *Fifth International Conference on Fun with Algorithms*, pages 52–64, 2010.
- [94] Song Li and Anthony Ephremides. A covert channel in mac protocols based on splitting algorithms. In *IEEE Wireless Communications and Networking Conference, 2005*, volume 2, pages 1168–1173. IEEE, 2005.

- [95] Song Li and Anthony Epliremides. A network layer covert channel in ad-hoc wireless networks. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 88–96. IEEE, 2004.
- [96] Chen Liang, Xianmin Wang, Xiaosong Zhang, Yu Zhang, Kashif Sharif, and Yu-an Tan. A payload-dependent packet rearranging covert channel for mobile voip traffic. *Information Sciences*, 465:162–173, 2018.
- [97] Jianhua Liu, Wuji Chen, and Yanxiang Wen. A robust and flexible covert channel in lte-a system. In *Journal of Physics: Conference Series*, volume 1087, page 062027. IOP Publishing, 2018.
- [98] Shaoyuan Liu, Zhi Fang, Feng Gao, Bakh Koussainov, Zijian Zhang, Jiamou Liu, and Liehuang Zhu. Whispers on ethereum: Blockchain-based covert data embedding schemes. In *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, pages 171–179, 2020.
- [99] Ying Lizhi, Huang Yongfeng, Yuan Jian, and Yunlu Bai Linda. A novel covert timing channel based on rtp/rtcp. *Chinese Journal of Electronics*, 21(4):711–714, 2012.
- [100] Jie Lu, Yong Ding, Zhenyu Li, and Chunhui Wang. A timestamp-based covert data transmission method in industrial control system. In *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)*, pages 526–532. IEEE, 2022.
- [101] Norka B Lucena, James Pease, Payman Yadollahpour, and Steve J Chapin. Syntax and semantics-preserving application-layer protocol steganography. In *International Workshop on Information Hiding*, pages 164–179. Springer, 2004.
- [102] Norka B Lucena, Grzegorz Lewandowski, and Steve J Chapin. Covert channels in ipv6. In *International Workshop on Privacy Enhancing Technologies*, pages 147–166. Springer, 2005.
- [103] Walter Lucia and Amr Youssef. Covert channels in stochastic cyber-physical systems. *IET Cyber-Physical Systems: Theory & Applications*, 6(4):228–237,

2021. doi: <https://doi.org/10.1049/cps2.12020>. URL <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cps2.12020>.
- [104] Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. Cloak: A ten-fold way for reliable covert communications. In *European Symposium on Research in Computer Security*, pages 283–298. Springer, 2007.
 - [105] Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. Tcp covert timing channels: Design and detection. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 420–429. IEEE, 2008.
 - [106] Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. Clack: A network covert channel based on partial acknowledgment encoding. In *2009 IEEE International Conference on Communications*, pages 1–5. IEEE, 2009.
 - [107] Xiapu Luo, Peng Zhou, Edmond WW Chan, Rocky KC Chang, and Wenke Lee. A combinatorial approach to network covert communications with applications in web leaks. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 474–485. IEEE, 2011.
 - [108] Kevin Maney. Bin laden’s messages could be hiding in plain sight. *USA Today*, 19, 2001.
 - [109] Michael Marone. Adaptation and performance of covert channels in dynamic source routing. *tech. rep., Computer Science Department, Yale University*, 2003.
 - [110] Wojciech Mazurczyk and Zbigniew Kotulski. New security and control protocol for voip based on steganography and digital watermarking. *arXiv preprint cs/0602042*, 2006.
 - [111] Wojciech Mazurczyk and Zbigniew Kotulski. New voip traffic security scheme with digital watermarking. In *International Conference on Computer Safety, Reliability, and Security*, pages 170–181. Springer, 2006.

- [112] Wojciech Mazurczyk and Zbigniew Kotulski. Covert channel for improving voip security. In *Advances in Information Processing and Protection*, pages 271–280. Springer, 2007.
- [113] Wojciech Mazurczyk and Krzysztof Szczypiorski. Covert channels in sip for voip signalling. In *International Conference on Global e-Security*, pages 65–72. Springer, 2008.
- [114] Wojciech Mazurczyk and Krzysztof Szczypiorski. Steganography of voip streams. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 1001–1018. Springer, 2008.
- [115] Wojciech Mazurczyk and Krzysztof Szczypiorski. Steganography in handling oversized ip packets. In *2009 International Conference on Multimedia Information Networking and Security*, volume 1, pages 559–564. IEEE, 2009.
- [116] Wojciech Mazurczyk and Krzysztof Szczypiorski. Evaluation of steganographic methods for oversized ip packets. *Telecommunication Systems*, 49(2):207–217, 2012.
- [117] Wojciech Mazurczyk and Steffen Wendzel. Information hiding: Challenges for forensic experts. *Commun. ACM*, 61(1):86–94, dec 2017. ISSN 0001-0782. doi: 10.1145/3158416. URL <https://doi.org/10.1145/3158416>.
- [118] Wojciech Mazurczyk, Milosz Smolarczyk, and Krzysztof Szczypiorski. Retransmission steganography applied. In *2010 International Conference on Multimedia Information Networking and Security*, pages 846–850. IEEE, 2010.
- [119] Wojciech Mazurczyk, Miłosz Smolarczyk, and Krzysztof Szczypiorski. On information hiding in retransmissions. *Telecommunication Systems*, 52(2):1113–1121, 2013.
- [120] Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski. *Information hiding in communication networks: fundamentals, mechanisms, applications, and countermeasures*. John Wiley & Sons, 2016.

- [121] Wojciech Mazurczyk, Steffen Wendzel, and Krzysztof Cabaj. Towards deriving insights into data hiding methods using pattern-based approach. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–10, 2018.
- [122] Wojciech Mazurczyk, Krystian Powójski, and Luca Caviglione. Ipv6 covert channels in the wild. In *Proceedings of the third central european cybersecurity conference*, pages 1–6, 2019.
- [123] Wojciech Mazurczyk, Przemysław Szary, Steffen Wendzel, and Luca Caviglione. Towards reversible storage network covert channels. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–8, 2019.
- [124] Aleksandra Mileva and Boris Panajotov. Covert channels in tcp/ip protocol stack-extended version. *Open Computer Science*, 4(2):45–66, 2014.
- [125] Aleksandra Mileva, Aleksandar Velinov, and Done Stojanov. New covert channels in internet of things. *The 12th International Conference on Emerging Security Information, Systems and Technologies*, 2018.
- [126] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS ’12, page 97–108, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450316514. doi: 10.1145/2382196.2382210. URL <https://doi.org/10.1145/2382196.2382210>.
- [127] Aziz Mohamed Azran. *Applying covert channel in TCP Fast Open (TFO)/Mohamed Azran Aziz*. PhD thesis, University of Malaya, 2019.
- [128] Ira S Moskowitz, Richard E Newman, and Paul F Syverson. Quasi-anonymous channels. Technical report, NAVAL RESEARCH LAB WASHINGTON DC CENTER FOR HIGH ASSURANCE COMPUTING SYSTEMS . . . , 2003.
- [129] M. Muench. Icmp-chat. *GitHub*, 2003.

- [130] Steven J Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 27–36, 2006.
- [131] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195. IEEE, 2005.
- [132] Steven J. Murdoch and Stephen Lewis. Embedding covert channels into tc-p/ip. In Mauro Barni, Jordi Herrera-Joancomartí, Stefan Katzenbeisser, and Fernando Pérez-González, editors, *Information Hiding*, pages 247–261, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31481-3.
- [133] RP Murphy and CEH CISSP. V00d00n3t - ipv6/icmpv6 covert channels. *Las Vegas: Defcon*, 27:29, 2006.
- [134] Esmira Mustafayeva, Gunay Huseynova, and Vagif Gasimov. Implementing covert channels to transfer hidden information over whatsapp on mobile phones. *American Journal of Engineering and Applied Sciences*, 6(2):32–35, 2020.
- [135] Sara Narteni, Ivan Vaccari, Maurizio Mongelli, Maurizio Aiello, and Enrico Cambiaso. On the feasibility of covert channels through short-message-service. In *ITASEC*, pages 23–34, 2021.
- [136] BBC News. Airlines bomb plot: The e-mails. *BBC*, September 2009. URL http://news.bbc.co.uk/2/hi/uk_news/8193501.stm.
- [137] Fox News. Joining china and iran, australia to filter internet. *Fox*, 2015. URL <https://www.foxnews.com/tech/joining-china-and-iran-australia-to-filter-internet>.
- [138] Lucas Nussbaum, Pierre Neyron, and Olivier Richard. On robust covert channels inside dns. In *IFIP International Information Security Conference*, pages 51–62. Springer, 2009.
- [139] Jonathan Oakley, Lu Yu, Xingsi Zhong, Ganesh Kumar Venayagamoorthy, and Richard Brooks. Protocol proxy: An fte-based covert channel. *Computers & Security*, 92:101777, 2020.

- [140] Michael A Padlipsky, David W Snow, and Paul A Karger. Limitations of end-to-end encryption in secure computer networks. Technical report, Mitre Corp Bedford MA, 1978.
- [141] Michael Perkins. *Hiding out in plaintext: covert messaging with bitwise summations*. PhD thesis, Iowa State University, 2005.
- [142] Birgit Pfitzmann. Information hiding terminology. In *Information Hiding: First International Workshop Proceedings, 1996*. Springer, 1996.
- [143] J. Postel. Internet protocol - darpa internet programm, protocol specification. RFC 0791, IETF, September 1981. URL <https://www.ietf.org/rfc/rfc791.txt>.
- [144] J. Postel. Transmission Control Protocol - DARPA Internet Program, Protocol Specification. RFC 0793, IETF, September 1981. URL <https://www.rfc-editor.org/info/rfc793>.
- [145] The Honeynet Project. Know your enemy: Sebek - a kernel based data capture tool. *Technical report*, 2003. URL <https://www.cs.jhu.edu/~rubin/courses/sp04/sebek.pdf>.
- [146] Haipeng Qu, Purui Su, and Dengguo Feng. A typical noisy covert channel in the ip protocol. In *38th Annual 2004 International Carnahan Conference on Security Technology, 2004.*, pages 189–192. IEEE, 2004.
- [147] Huyu Qu, Qiang Cheng, and Ece Yaprak. Using covert channel to resist dos attacks in wlan. In *ICWN*, pages 38–44, 2005.
- [148] Baishakhi Ray and Shivakant Mishra. A protocol for building secure and reliable covert channel. In *2008 Sixth Annual Conference on Privacy, Security and Trust*, pages 246–253. IEEE, 2008.
- [149] Jason Reaves. Covert channel by abusing x509 extensions. *viXra.org*, 2018.
- [150] Ruben Rios, Jose A Onieva, and Javier Lopez. Hide_dhcp: Covert communications through network configuration messages. In *IFIP International Information Security Conference*, pages 162–173. Springer, 2012.

- [151] Craig H Rowland. Covert channels in the tcp/ip protocol suite. *Munksgaard International Publishers Ltd., Copenhagen*, 1997.
- [152] Joanna Rutkowska. The implementation of passive covert channels in the linux kernel. In *Proc. Chaos Communication Congress*, 2004.
- [153] Amir Sabzi, Liron Schiff, Kashyap Thimmaraju, Andreas Blenk, and Stefan Schmid. Macchiato: Importing cache side channels to sdns. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, pages 8–14, 2021.
- [154] Jens Saenger, Wojciech Mazurczyk, Jörg Keller, and Luca Caviglione. Voip network covert channels to enhance privacy and information sharing. *Future Generation Computer Systems*, 111:96–106, 2020.
- [155] Krzysztof Sawicki, Grzegorz Biesczad, and Zbigniew Piotrowski. Stegoframeorder—mac layer covert network channel for wireless ieee 802.11 networks. *Sensors*, 21(18):6268, 2021.
- [156] Tobias Schmidbauer, Steffen Wendzel, Aleksandra Mileva, and Wojciech Mazurczyk. Introducing dead drops to network steganography using arp-caches and snmp-walks. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–10, 2019.
- [157] Tim Schmidt. Covert channel based on amd precision boost 2. *Karl*, 2019.
- [158] Carlos Scott. Network covert channels: Review of current state and analysis of viability of the use of x. 509 certificates for covert communications. *Technical Report: Department of Mathematics-University of London*, 2008.
- [159] Ahmed Seffah. The evolution of design patterns in hci: from pattern languages to pattern-oriented design. In *Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems*, pages 4–9, 2010.
- [160] Sarah H Sellke, C-C Wang, Saurabh Bagchi, and Ness Shroff. Tcp/ip timing channels: Theory to implementation. In *IEEE INFOCOM 2009*, pages 2204–2212. IEEE, 2009.

- [161] Jun O Seo, Sathiamoorthy Manoharan, and Aniket Mahanti. A discussion and review of network steganography. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 384–391. IEEE, 2016.
- [162] Sergio D Servetto and Martin Vetterli. Communication using phantoms: covert channels in the internet. In *Proceedings. 2001 IEEE International Symposium on Information Theory (IEEE Cat. No. 01CH37252)*, page 229. IEEE, 2001.
- [163] Gaurav Shah, Andres Molina, Matt Blaze, et al. Keyboards and covert channels. In *USENIX Security Symposium*, volume 15, page 64, 2006.
- [164] Kartik Sharma and Akshay Sharma. High bandwidth covert channel using tcp-ip packet header. In *International Conference on Electronics and Communication Systems*, 02 2016.
- [165] Z. Shelby, K. Hartke, and C. Bormann. Rfc 7252: The constrained application protocol (coap). Technical report, RFC Editor, USA, 2014.
- [166] Yao Shen, Wei Yang, and Liusheng Huang. Concealed in web surfing: Behavior-based covert channels in http. *Journal of Network and Computer Applications*, 101:83–95, 2018.
- [167] Gustavus J Simmons. The prisoners’ problem and the subliminal channel. In *Advances in Cryptology*, pages 51–67. Springer, 1984.
- [168] Harjit Singh. Analysis of different types of steganography. *International journal of scientific research in science, engineering and technology*, 2:578–582, 2016.
- [169] Aron J Smith-Donovan. Passing time and syncing secrets: Demonstrating covert channel vulnerabilities in precision time protocol (ptp). *Mathematics, Statistics, and Computer Science Honors Projects*, 2022.
- [170] Kristian Stokes, Bo Yuan, Daryl Johnson, and Peter Lutz. Icmp covert channel resiliency. In *Technological Developments in Networking, Education and Automation*, pages 503–506. Springer, 2010.

- [171] Przemysław Szary, Wojciech Mazurczyk, Steffen Wendzel, and Luca Caviglione. Design and performance evaluation of reversible network covert channels. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–8, 2020.
- [172] Krzysztof Szczypiorski. Hiccups: Hidden communication system for corrupted networks. In *International Multi-Conference on Advanced Computer Systems*, pages 31–40, 2003.
- [173] Krzysztof Szczypiorski. Steganography in tcp/ip networks. state of the art and a proposal of a new system-hiccups. *Warsaw University of Technology, Poland Institute of Telecommunications, Warsaw, Poland*, 2003.
- [174] Samira Taheri, Mojtaba Mahdavi, and Neda Moghim. A dynamic timing-storage covert channel in vehicular ad hoc networks. *Telecommunication Systems*, 69(4):415–429, 2018.
- [175] Yu-an Tan, Xinting Xu, Chen Liang, Xiaosong Zhang, Quanxin Zhang, and Yuanzhang Li. An end-to-end covert channel via packet dropout for mobile networks. *International Journal of Distributed Sensor Networks*, 14(5):1550147718779568, 2018.
- [176] Yu-an Tan, Xiaosong Zhang, Kashif Sharif, Chen Liang, Quanxin Zhang, and Yuanzhang Li. Covert timing channels for iot over mobile networks. *IEEE Wireless Communications*, 25(6):38–44, 2018.
- [177] Jonathan S Thyer. Covert data storage channel using ip packet headers. *SANS institute*, 2008.
- [178] Jing Tian, Gaopeng Gou, Chang Liu, Yige Chen, Gang Xiong, and Zhen Li. Dlchain: A covert channel over blockchain based on dynamic labels. In *International Conference on Information and Communications Security*, pages 814–830. Springer, 2019.
- [179] Franco Tommasi, Christian Catalano, Alessandro Caniglia, and Ivan Taurino. Cotiip: a new covert channel based on incomplete ip packets. In *2022 7th*

International Conference on Smart and Sustainable Technologies (SpliTech), pages 1–7. IEEE, 2022.

- [180] Zouheir Trabelsi and Imad Jawhar. Covert file transfer protocol based on the ip record route option. *Journal of Information Assurance and Security*, 5(1):64–73, 2010.
- [181] Zouheir Trabelsi, Hesham El-Sayed, Lilia Frikha, and Tamer Rabie. Traceroute based ip channel for sending hidden short messages. In *International Workshop on Security*, pages 421–436. Springer, 2006.
- [182] Zouheir Trabelsi, Hesham El-Sayed, Lilia Frikha, and Tamer Rabie. A novel covert channel based on the ip header record route option. *International journal of advanced media and communication*, 1(4):328–350, 2007.
- [183] Zouheir Trabelsi, Wassim El-Hajj, and Safuat Hamdy. Implementation of an icmp-based covert channel for file and message transfer. In *2008 15th IEEE International Conference on Electronics, Circuits and Systems*, pages 894–897. IEEE, 2008.
- [184] Computer Security Center (US). *Computer Security Requirements: Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*. Dod Computer Security Center, 1985.
- [185] Maarten Van Horenbeeck. Deception on the network: thinking differently about covert channels. *Australian Information Warfare and Security Conference*, 2006.
- [186] vecna. B0ck. *Butchered From Inside*, 2000.
- [187] Aleksandar Velinov, Aleksandra Mileva, Steffen Wendzel, and Wojciech Mazurczyk. Covert channels in the mqtt-based internet of things. *IEEE Access*, 7:161899–161915, 2019.
- [188] Arne Vidstrom. Ack tunneling trojans. Covert shell through TCP ACK, 2000.
- [189] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. Censorspoof: Asymmetric communication with IP spoofing

- for censorship-resistant web browsing. *CoRR*, abs/1203.1673, 2012. URL <http://arxiv.org/abs/1203.1673>.
- [190] Zhenghong Wang, Jing Deng, and Ruby B Lee. Mutual anonymous communications: a new covert channel based on splitting tree mac. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 2531–2535. IEEE, 2007.
 - [191] Steffen Wendzel. Protocol channels as a new design alternative of covert channels. Technical report, CERN, 2008.
 - [192] Steffen Wendzel and Wojciech Mazurczyk. Poster: An educational network protocol for covert channel analysis using patterns. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1739–1741, 2016.
 - [193] Steffen Wendzel, Benjamin Kahler, and Thomas Rist. Covert channels and their prevention in building automation protocols: A prototype exemplified using bacnet. In *2012 IEEE International Conference on Green Computing and Communications*, pages 731–736. IEEE, 2012.
 - [194] Steffen Wendzel, Sebastian Zander, Bernhard Fechner, and Christian Herdin. Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys (CSUR)*, 47(3):1–26, 2015.
 - [195] Steffen Wendzel, Luca Caviglione, Wojciech Mazurczyk, Aleksandra Mileva, Jana Dittmann, Christian Krätzer, Kevin Lamshöft, Claus Vielhauer, Laura Hartmann, Jörg Keller, Tom Neubert, and Sebastian Zillien. Information hiding patterns project. <https://patterns.ztt.hs-worms.de>, 2021. Accessed: 2022-11-07.
 - [196] Steffen Wendzel, Luca Caviglione, Wojciech Mazurczyk, Aleksandra Mileva, Jana Dittmann, Christian Krätzer, Kevin Lamshöft, Claus Vielhauer, Laura Hartmann, Jörg Keller, et al. A generic taxonomy for steganography methods. *TechRxiv*, 2022.

- [197] Steve R White. Covert distributed processing with computer viruses. In *Conference on the Theory and Application of Cryptology*, pages 616–619. Springer, 1989.
- [198] Philipp Winter and Stefan Lindskog. *How the great firewall of china is blocking tor*. USENIX-The Advanced Computing Systems Association, 2012.
- [199] Manfred Wolf. Covert channels in lan protocols. In *Local Area Network Security Workshop*, pages 89–101. Springer, 1989.
- [200] Guangliang Xu, Wei Yang, and Liusheng Huang. Hybrid covert channel in lte-a: modeling and analysis. *Journal of Network and Computer Applications*, 111:117–126, 2018.
- [201] Jun Xu, Jinliang Fan, Mostafa H Ammar, and Sue B Moon. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 280–289. IEEE, 2002.
- [202] Pengfei Xue, Hanlin Liu, Jingsong Hu, and Ronggui Hu. A multi-layer steganographic method based on audio time domain segmented and network steganography. In *AIP Conference Proceedings*, volume 1967, page 020046. AIP Publishing LLC, 2018.
- [203] Lihong Yao, Xiaochao Zi, Li Pan, and Jianhua Li. A study of on/off timing channel based on packet delay distribution. *Computers & Security*, 28(8):785–794, 2009.
- [204] Xuhang Ying, Giuseppe Bernieri, Mauro Conti, and Radha Poovendran. Tacan: Transmitter authentication through covert channels in controller area networks. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 23–34, 2019.
- [205] Li Yuanzhang, Liu Junli, Xu Xinting, Zhang Xiaosong, Zhang Li, and Zhang Quanxin. A robust packet-dropping covert channel for mobile intelligent terminals. *International Journal of Intelligent Systems*, 2022.

- [206] Sebastian Zander, Grenville Armitage, and Philip Branch. Covert channels in the ip time to live field. *Swinburne University of Technology*, 01 2007.
- [207] Sebastian Zander, Grenville Armitage, and Philip Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57, 2007.
- [208] Sebastian Zander, Grenville Armitage, and Philip Branch. Stealthier inter-packet timing covert channels. In *International Conference on Research in Networking*, pages 458–470. Springer, 2011.
- [209] Sebastian Zander et al. Performance of selected noisy covert channels and their countermeasures in ip networks. *Centre for Advanced Internet Architectures Faculty of Information and Communication Technologies*, 2010.
- [210] L. Zelenchuk. Skeeve - icmp bounce tunnel. *GitHub*, 2004.
- [211] Lejun Zhang, Tianwen Huang, Xiaoyan Hu, Zhijie Zhang, Weizheng Wang, Donghai Guan, Chunhui Zhao, and Seokhoon Kim. A distributed covert channel of the packet ordering enhancement model based on data compression. *CMC-Computers, Materials & Continua*, 64(3):2013–2030, 2020.
- [212] Quanxin Zhang, Mengyan Zhu, Chen Liang, Kunqing Wang, Kai Yang, and Yuanzhang Li. A timestamp-regulating volte covert channel against statistical analysis. *Mobile Networks and Applications*, 26(4):1493–1502, 2021.
- [213] Xiao-Guang Zhang, Guang-Hong Yang, and Xiu-Xiu Ren. Network steganography based security framework for cyber-physical systems. *Information Sciences*, 609:963–983, 2022.
- [214] Xiaosong Zhang, Yu-An Tan, Chen Liang, Yuanzhang Li, and Jin Li. A covert channel over volte via adjusting silence periods. *IEEE Access*, 6:9292–9302, 2018.
- [215] Xiaosong Zhang, Linhong Guo, Yuan Xue, and Quanxin Zhang. A two-way volte covert channel with feedback adaptive to mobile network environment. *IEEE Access*, 7:122214–122223, 2019.

- [216] Xiaosong Zhang, Liehuang Zhu, Xianmin Wang, Changyou Zhang, Hongfei Zhu, and Yu-an Tan. A packet-reordering covert channel over volte voice and video traffics. *Journal of Network and Computer Applications*, 126:29–38, 2019.
- [217] Hong Zhao. Covert channels in 802.11 e wireless networks. In *2014 Wireless Telecommunications Symposium*, pages 1–5. IEEE, 2014.
- [218] Sebastian Zillien and Steffen Wendzel. Reconnection-based covert channels in wireless networks. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 118–133. Springer, 2021.
- [219] Xin-guang Zou, Qiong Li, Sheng-He Sun, and Xiamu Niu. The research on information hiding based on command sequence of ftp protocol. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 1079–1085. Springer, 2005.

Ringraziamenti

Al termine di questo progetto durato 8 mesi non potevo non dedicare una sezione alle persone che si sono rivelate fondamentali per me nella riuscita dello stesso.

Il primo e più sentito ringraziamento va ai relatori Bistarelli e Santini che mi hanno dato la possibilità di poter approfondire un tema così attuale e stimolante come quello della steganografia di rete. Durante questo periodo mi hanno seguito in maniera maniacale aiutandomi nel migliorare ogni singolo dettaglio di questa tesi.

È stato un progetto al quale ho dedicato una grande quantità di energie e ci sono stati momenti che mi hanno messo a dura prova, ma l'ho portato a termine soprattutto grazie alle persone che mi sono state vicino. Impossibile non partire dai miei genitori, Sergio e Tiziana, che mi hanno dato la possibilità di intraprendere questo percorso facendo di tutto affinché lo portassi a termine. In particolare però voglio ringraziare mia sorella Elena, sempre pronta a sostenermi in qualsiasi situazione. La stessa gratitudine è riservata ai miei zii e alle mie cugine.

Un grazie di cuore ad Alex, Francesca e Mattia che sono state le persone che più di tutti mi hanno aiutato nei momenti di poca lucidità con la loro disponibilità, sensibilità e schiettezza.

In un percorso di studi così tecnico e specifico le conoscenze sono fondamentali per raggiungere gli obiettivi. Per questo motivo sarò sempre grato ai miei compagni di università Federico, Giorgio, Sara, Gabriele e Riccardo per aver condiviso con me le loro esperienze.

Essenziali sono state le amicizie che mi hanno regalato attimi di spensieratezza e leggerezza imprescindibili per mantenere i buoni ritmi di lavoro tenuti. Per questo dedico un ringraziamento speciale al mio Club, ad ogni singolo membro per avermi fatto vivere momenti che ricorderò per sempre con il sorriso.

Ho ricevuto più di quanto sono riuscito a donare in questo periodo, e mi sento in debito con tutte le persone a me care. Ho apprezzato tantissimo ogni vostro gesto, anche il più banale, per questa ragione la mia gratitudine è rivolta anche ai miei amici birraioli, ceraioli, interisti e volontari.

Voglio quindi ringraziare tutte quelle persone che in un modo o nell'altro fanno parte della mia vita e con le quali ho condiviso questo percorso, ed infine...

