

Politecnico di Torino

Politecnico di Torino

Master of Science in Computer Engineering

A.a. 2025/2026

"Forensically-aware" steganographic covert channels

Relatori:
Andrea Atzeni

Candidati:
Cristina Demontis

Abstract

AI based detectors for steganographic covert channels achieve high accuracy but operate as "black boxes", limiting their admissibility as forensic evidence. This thesis addresses this gap by shifting the classification perspective from the specific carrier medium to the nature of the forensic trace it produces, emphasizing the balance between detectability and explainability.

Investigating both network and memory domains, the study evaluates threats across multiple system layers. While memory based channels, such as those exploiting cache behavior, are addressed theoretically through a four level analytical pipeline, a practical proof of concept was implemented for the Silent History Protocol (SHP) over ARP and DNS traffic using cross layer correlation.

Experimental results demonstrate the effectiveness of combining network anomalies with host level traces through a structured macro (run level) and micro (window level) analysis. Specifically, in the SHP over ARP scenario three macro level metrics, the percentage of ARP active time windows, binary entropy, and the presence of channel like host processes, remained statistically significant after Benjamini–Hochberg correction. This shows that persistent network activity combined with host level evidence reliably distinguishes the covert channel from baseline traffic.

However, anticipating a forensically aware attacker's evasion strategies (e.g., parameter obfuscation or decoy traffic) highlights the current limitations of non blind experimental setups and implementation specific rules. Therefore, no single indicator can reliably expose advanced covert channels; establishing transparent, court admissible proof requires an integrated, multi source approach.

Acknowledgements

Table of Contents

List of Figures VII

Glossary VIII

1 Research Context and Objectives 1
- 1.1 Motivation . . . 1
- 1.2 Objectives of the Thesis . . . 2
- 1.3 Thesis Organization . . . 2

2 Introduction and Steganography Fundamentals 4
- 2.1 From Antiquity to Digital Era . . . 4
  - 2.1.1 Ancient Methods: 5th Century BC, Roman Era . . . 4
  - 2.1.2 20th Century: WWII, birth of digital steganography . . . 4
  - 2.1.3 Covert Channels: Simmons (1984), Szczypiorski (2003) . . . 5
  - 2.1.4 Today's Challenges . . . 6
- 2.2 Basic Concepts and Metrics . . . 7
  - 2.2.1 What is Steganography? . . . 7
  - 2.2.2 Steganography vs Steganalysis . . . 8
  - 2.2.3 Evaluation Metrics . . . 8
- 2.3 Network Covert Channels . . . 10
  - 2.3.1 Storage Channels . . . 10
  - 2.3.2 Timing Channels . . . 11
- 2.4 Memory Based Covert Channels . . . 13
  - 2.4.1 DRAM Based Channels . . . 13
  - 2.4.2 Cache Based Channels . . . 14
  - 2.4.3 Memory Structure and Shared Memory Covert Channels . . . 16
- 2.5 Limitation Resource Based Classification . . . 17
- 2.6 Generalized Taxonomy . . . 18
- 2.7 Conclusion Remarks . . . 21

3 Forensic Classification and Detection Methodologies 23
3.1 Paradigm Shift: from "carrier medium" to "trace nature" 23
3.2 Digital Forensic Investigation Framework 24
3.3 Taxonomy of Traces 29
3.3.1 Signature based Traces 29
3.3.2 Statistical Traces 31
3.3.3 Latent/Heuristic Traces 33
3.4 Detectability vs Explainability 37
3.5 Information Fusion: Multi Source Evidence and Cross Layer Correlation 38
3.6 Conclusion Remarks 39

4 Proof of Concept 40
4.1 Presentation of studied Covert Channels 40
4.1.1 Silent History Channel over ARP and DNS 40
4.1.2 Write Back Based Covert Channel 41
4.2 Generalized Framework 43
4.2.1 Silent History Protocol Channel Framework 43
Component 1: Shared Entropy Source 43
Component 2: Feature Extraction Function 43
Component 3: Signaling Channel 44
Component 4: Reference Mechanism 44
SHP Parametrization 44
4.2.2 Write Back Channel Framework 44
Component 1: Shared Hardware Resources 45
Component 2: Modulation Function 45
Component 3: Measurement Function 45
Component 4: Synchronization Protocol 46
Hardware Channel Parameters 46
4.3 SHP over ARP: Implementation and Forensic Analysis 47
4.3.1 Experimental Environment 47
4.3.2 Generic Framework: SHP over ARP scenario 48
4.3.3 Implementation 50
4.3.4 Multiple Source Forensic Pipeline 54
Operational Implementation of the Forensic Analysis 54
Macro Analysis 58
Micro Analysis 65
4.4 SHP over DNS: Implementation and Forensic Analysis 70
4.4.1 Generic Framework: SHP over DNS scenario 70
Experimental Environment 75
4.4.2 Multi Source Forensic Pipeline 76

Macro Analysis 77
Micro Analysis 82
4.5 Comparing SHP over ARP and SHP over DNS: evidence and forensic analysis 88
4.6 Evasion and countermeasures: the attacker's perspective 90
Network level evasion: randomization and decoy traffic . . . 90
Host level evasion: obfuscation and decoupling 90
Cross layer evasion: breaking the causal correlation 91
4.7 Cache Based Covert Channels: A Theoretical Study 92
The Four Layer Forensic Framework 93

5 Limitations and Future Works 96
5.1 Limitations of the Analysis and the Experimental Design 96
Risk of circularity and optimization bias 96
The ground truth problem and the need for blind scenarios . 97
Observer effect and limits of the allowlist approach 98
Statistical and sample size limitations 98
5.2 Conclusions 99

Bibliography 101

List of Figures

2.1 A Revised Taxonomy of Steganography Embedding Patterns. Reproduced from Wendzel et al. [17, p. 8], licensed under CC BY 4.0. 21

Glossary

LSB Least Significant Bit

EOF End Of File

HICCUPS Hidden Communication system for CorrUPted networkS

WLAN Wireless Local Area Network

MAC Media Access Control

C&C Command and Control

GAN Generative Adversarial Network

TCP Transmission Control Protocol

VoIP Voice over IP

CS Covert Sender
CR Covert Receiver

DHCP Dynamic Host Configuration Protocol

HTTP Hypertext Transfer Protocol

FTP File Transfer Protocol

PDU Protocol Data Unit

IPv4 Internet Protocol version 4

TTL Time To Live

DNS Domain Name System

NAT Network Address Translation

DRAM The Dynamic Random-Access Memory

CPU Central Processing Unit

WB Write Back

NVM Non Volatile Memory

PIM Processing In Memory

RFC Request For Comments

NTFS New technology file system

ADS Alternative Data Streams

POV Pairs of Values

ML Machine Learning

DL Deep Learning

CNN Convolutional Neural Networks

RNN Recurrent Neural Networks

SHP Silent History Protocol

ARP Address Resolution Protocol

HCC History Covert Channels

POI Packet Of Interest

ISD Inter Signal Delay
IPv6 Internet Protocol version 6
IPD Inter Packet Delay
ISPN Inter Signal Packet Number

ECC Error Correction Code

RAM Random Access Memory

AI Artificial Intelligence

MD5 Message Digest 5 (cryptographic hash function)

SHA-256 Secure Hash Algorithm 256 (cryptographic hash function)

ICMP Internet Control Message Protocol

DCT Discrete Cosine Transform

RS Regular/Singular (steganalysis)

IoT Internet of Things

OS Operating System

DLL Dynamic Link Library

BPF Berkeley Packet Filter

UDP User Datagram Protocol

BH Benjamini–Hochberg

PID Process ID

LAN Local Area Network

CTF Capture the flag

LRU Least Recently Used

CDF Cumulative Distribution Function

FFT Fast Fourier Transform

ACK Acknowledgment (Number)

Chapter 1

Research Context and Objectives

1.1 Motivation

Modern systems provide several shared resources that can be abused to exchange information through unintended communication paths. These paths, known as covert channels, can operate over network protocols, host processes, memory hierarchies, or hardware components. Because the transmitted information is designed to remain hidden within legitimate activity, covert channels can bypass traditional security controls and complicate digital forensic investigations.

Traditional detection methods often focus on the specific carrier medium, such as a protocol, a file, or a memory component. However, different covert channels may produce similar observable traces, while similar techniques may require different detection methods when implemented on different media. For this reason, this thesis adopts a trace oriented forensic perspective. The main objective is to identify measurable deviations from expected system behavior and to correlate evidence from multiple layers.

A further challenge concerns forensic explainability. Machine learning methods may provide high detection performance, but their decisions can be difficult to interpret. In contrast, statistical analysis, process monitoring, and causal correlation produce evidence that can be more clearly explained by a forensic analyst. This thesis therefore emphasizes interpretable and multi source analysis.

1.2 Objectives of the Thesis

The main objective of this thesis is to study the forensic traces produced by steganographic covert channels and to define a practical methodology for their detection and interpretation.

The specific objectives are:

- **Background analysis:** describe steganography, covert channels, network and memory based communication, and their main evaluation criteria.
- **Trace based classification:** classify covert channel evidence according to its observable nature, distinguishing deterministic, statistical, and heuristic traces.
- **Forensic methodology:** define an analysis process based on evidence acquisition, preservation, statistical analysis, host monitoring, and cross layer correlation.
- **Proof of concept implementation:** implement and analyze the SHP covert channel over ARP and DNS traffic.
- **Multi source investigation:** compare a baseline scenario with an active covert channel scenario using network and host level evidence.
- **Correlation analysis:** evaluate whether ARP anomalies, process activity, loaded modules, and packet capture behavior occur together within the same time windows.
- **Memory channel framework:** develop a theoretical forensic pipeline for cache based covert channels using latency distributions, periodicity analysis, and host context information.

The thesis does not aim to provide a definitive detector for every possible covert channel. Instead, it aims to show how interpretable, multi layer evidence can support the identification and forensic assessment of suspicious covert communication.

1.3 Thesis Organization

This thesis is organized as follows:

- **Chapter 2 – Introduction and Steganography Fundamentals:** introduces the evolution of steganography, the main concepts of covert communication, evaluation metrics, and the differences between network and memory based channels.

- **Chapter 3 – Forensic Classification and Detection Methodologies:** presents the trace oriented classification, the main acquisition and preservation requirements, existing detection techniques, and the role of information fusion and cross layer correlation.
- **Chapter 4 – Proof of Concept:** describes the studied covert channels, including SHP over ARP, SHP over DNS and write back cache channels. It also presents the generalized channel models and the implemented forensic pipelines.
- **Chapter 5 – Limitations and Future Works:** summarizes the main findings, discusses the limitations of the proposed analysis, and identifies future research directions for automated detection, memory channel experimentation, and large scale forensic validation.

Terminology Criterion Finally, a specific criterion has been applied regarding technical terminology. Concepts whose immediate understanding is essential to follow the flow of the discourse are explained directly in footnotes, allowing the reader to get their meaning without interruption. Conversely, acronyms and broader technical terms are gathered in the Glossary at the beginning of the document.

Chapter 2

Introduction and Steganography Fundamentals

2.1 From Antiquity to Digital Era

2.1.1 Ancient Methods: 5th Century BC, Roman Era

The earliest forms of information hiding date back to ancient Greece, with the story of Histiaeus. He was the ruler of Miletus, but once he was captured at Susa by the Persian court, he wanted to rise up against them. To inform Aristagoras, his son in law and governor of his country, he had a secret message tattooed on his most trusted slave with the meaning of "Rise up against Persians", waited for the hair to regrow, and then sent him to Miletus [1].

This was the first form of steganography, and it arose from the need to hide the existence of the message itself.

Throughout the centuries, many different types of steganography emerged: for example, Herodotus describes messages carved into wood and covered with wax, or later, in Roman times, sympathetic inks, mostly based on milk or fruit juice, became widespread, becoming visible only with heat or chemical reagents [1].

2.1.2 20th Century: WWII, birth of digital steganography

Interest in steganography definitely grew during World War II and the Cold War: microdots reduced entire pages to microscopic dots embedded in innocent texts;

there was the return of invisible inks, and the innovative idea of Hedy Lamarr and George Antheil. They thought about a way to not easily find and intercept the signals used to guide torpedoes. The idea was to disperse the signal over a wide frequency bandwidth: in this way, the command was spread over multiple frequencies, and it was too difficult to block it [1].

The concept of embedding information in different frequencies was adopted in the following decades in the fields of digital image and audio steganography.

With the advent of computers (1970s-1990s), the first digital techniques emerged, such as the LSB, a technique used in image steganography that consists of replacing the least significant bit of each pixel with a bit of the secret message that you want to hide: the color change is imperceptible to the human eye. The same method is used for audio steganography: the added noise is imperceptible to the human ear [1].

Besides concealment through multimedia, steganography was developed also in file system based methods, that respectively exploit slack space, using the space between the EOF and the end of the cluster, or also hidden partitions, in which could reside data or malware that are invisible to the "visible" partition tables [1].

2.1.3 Covert Channels: Simmons (1984), Szczypiorski (2003)

The concept of covert channel was formalized by Simmons in the book "Advances in Cryptology: Proceedings of Crypto 83" [2] as a communication means, not intended by the system, capable of transferring information by exploiting parameters not designed for this purpose. He introduced this idea as the prisoner's problem: two prisoners can exchange only messages that go through a warden, but they want to exchange additional secret messages.

So, Simmons defined the concept of subliminal channels as a hidden channel that exploits authentication or signature protocols to exchange additional data. Messages seem normal to the warden, but they carry secret information, extractable only by the receiver. This is the first formalization of the concept of a covert channel: a communication means not intended by the system, that uses parameters that are not created for transporting data [2].

Later, Szczypiorski introduced the term "network steganography" in 2003 in "Steganography in TCP/IP networks: State of the Art and a Proposal of a New System–HICCUPS" [3], proposing systems like HICCUPS that use a network steganography system on WLAN. The main idea is to use corrupted checksum

frames intentionally: for normal stations they are corrupted, and so they are discarded, but for malicious nodes, they contain information, from which they extract meaningful information. The HICCUPS model defines different hidden channels at the MAC layer, including one based on integrity and checksum, generating a real covert channel that exploits protocol parameters, such as the corrupted frames, not intended by the communication [3].

2.1.4 Today's Challenges

In our days, modern threats do not limit themselves to hiding data in images or protocols, but they include synthetic cover objects (GANs), malware integrating steganographic payloads, and hardware channels that bypass network defenses.

Techniques such as StegoMalware[1] highlight that steganography is not just an academic topic of study, but it can be a real vector for malware distribution, C&C, and data exfiltration. Hardware covert channels demonstrate that even with network isolation, threats do not vanish. The only way to counteract steganography used in a malicious way is to implement a real line of defense, with efficient countermeasures such as statistical monitoring, timing analysis, resource isolation, cache partitioning, and to reinforce all the weaker points of our infrastructures.

[1]StegoMalware hides malicious code or data within seemingly harmless content, such as images or multimedia files, to evade security controls.

2.2 Basic Concepts and Metrics

2.2.1 What is Steganography?

Steganography, from Greek *steganos* and *graphia* which respectively mean "covered or concealed" and "writing", is a practice of representing information within another message or physical object, in a way that the presence of the concealed information is not evident to an unsuspecting person's examination. In the field of computer science, a file, message, or video is concealed within another file, message, image, or video. So the main scope of steganography is to hide the very presence of the communication.

It should not be confused with cryptography, which is the science and practice of hiding or coding information so that only authorized people can read and process it, and also with watermarking, which can be a faint design, logo, or hidden code embedded into paper, digital images, videos, or files; it proves who made the item, shows ownership, and stops people from copying work without permission [1].

The feature that distinguishes steganography from other practices is the willingness of using resources that are not supposed to be used for communication.

Then, the difference between a Channel, a Side Channel, a Covert Channel, and a Steganographic Covert Channel also needs to be clarified:

- **Legit Channel:** a communication means used to transmit data between entities, authorized by the system. It uses, for example, TCP sockets, pipes, shared files, etc. It is subject to access control mechanisms and security policies [4].
- **Side Channel:** a means not created for communication that accidentally reveals information due to a misconfiguration or side effect of the system. It can reveal timing, execution patterns, power consumption, radio frequency emissions, cache or memory access patterns. There is not the typical pattern of a malicious active sender and receiver, but just a malicious node that exploits data leakage [4].
- **Covert Channel:** a channel not intended nor authorized by the security policy, but intentionally misused by two entities, usually a sender and receiver, to exchange information. It violates the security policies, allowing communication between nodes that are not supposed to communicate [4].
- **Steganographic Covert Channel:** a kind of covert channel that uses a legitimate object or data flow to transmit secret information, in order to hide

the very existence of the communication [5] .

2.2.2 Steganography vs Steganalysis

Steganography and steganalysis represent two complementary aspects of information hiding. Steganography aims to embed secret information into a legitimate object or data flow while minimizing the observable evidence of the communication. Steganalysis instead attempts to determine whether hidden information is present, usually without necessarily recovering the concealed message.

The interaction between these two fields can be described as a continuous competition. Steganographic techniques seek to increase capacity and robustness while reducing their statistical, structural, temporal, or behavioral traces. Steganalysis methods attempt to identify these traces through statistical analysis, machine learning, deep learning, anomaly detection, or the correlation of evidence from multiple system layers [6].

Steganalysis can be specific or blind. Specific methods target a known embedding technique and exploit its characteristic traces, whereas blind methods attempt to detect a wider range of techniques without prior knowledge of the algorithm or carrier. Although blind steganalysis is more suitable for real world scenarios, it remains challenging because covert channels can use different media, low transmission rates, adaptive behavior, or previously unseen mechanisms.

For this reason, effective detection should combine statistical, temporal, behavioral, and host level evidence rather than relying on a single indicator.

The fundamental challenge in this competition lies in balancing the competing objectives of both fields. From the steganographer's perspective, the goal is to maximize the amount of hidden information while minimizing detectability. From the steganalyst's perspective, the goal is to identify anomalies without generating excessive false positives. This balance is quantified through a set of evaluation metrics, that capture the trade offs between capacity, stealthiness, robustness, and computational cost. These metrics provide a common framework for comparing different techniques and understanding their practical limitations [6].

2.2.3 Evaluation Metrics

The three main features for evaluating a steganographic technique are:

- **Steganographic bandwidth (capacity)** refers to the amount of secret bits that a specific technique can carry within a time unit. For example, if a

method allows the transmission of 100 bits per second over a VoIP channel, this represents the steganographic bandwidth; alternatively, if it is able to hide 0.5 bits per pixel in an image, this is the pixel capacity. A higher bandwidth enables the transmission of more secret bits, but at the same time, it leaves more statistical traces [7, p. 73].

- **Undetectability** is the property by which an adversary cannot detect the presence of a steganogram[2] inside the cover medium. The widely used approach to intercept a steganogram is static steganalysis: analyze the statistical features of the captured file or flow, such as bit distribution, LSB, and timing patterns, and compare them to the "clean" cover medium [7, p. 73].

- **Robustness** is the resistance to the number of transformations or alterations that a steganogram can undergo before the secret message is damaged or destroyed. There exists a large number of legitimate data transformations, such as JPEG compression for images, noise cancelling for audio files, packet loss or delay, etc. A weaker steganographic technique suffers from all of these alterations [7, p. 73].

The magic triangle, introduced by Fridrich and presented in the Chapter 2 of the book "Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures"[7, p. 74], correlates the three features previously described: bandwidth, undetectability, and robustness cannot be maximized simultaneously. However, increasing the bandwidth also increases detectability, requiring a careful balance: the payload must be limited to preserve adequate security and resistance to alterations.

[2]Steganogram is the file or stream resulting from embedding a secret message into a cover, such that the presence of the message remains hidden [8].

2.3 Network Covert Channels

A network covert channel is an unauthorized communication means that exploits overt network traffic as a carrier for secret data. By hiding data inside the overt communication, the impact on the overt transmission is minimal, and thus the covert transmission is concealed effectively.

The model involves a Covert Sender (CS), who chooses how to encode secret bits on the shared medium; a Covert Receiver (CR), who receives the secret bits and knows how to decode them; and a shared medium, which, in this scenario, is the network. The CS and CR can reside on the same host that generates the overt traffic, on a router or gateway along the traffic path, or on different machines that share the same observable network traffic [7].

The first fundamental distinction of Network Covert Channels is based on how secret data are encoded into the carrier, resulting in storage and timing channels.

2.3.1 Storage Channels

Storage Channels manipulate redundant or unused fields in protocols, and the CR reads these values based on a shared pattern. Manipulation can be related to the payload, header, optional, reserved or unused fields, or the protocol structure itself.

As reported in Chapter 3 of the book "Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures" [9], the main techniques are:

1. **Size Manipulation of Protocol Data Units**, a technique that encodes secret data by varying the size of IP packets or frames. Different packet sizes correspond to different symbols: for example, small packets may represent bit 0, while large packets represent bit 1 [9, pp. 96, 97].

2. **Sequence Modulation in PDUs**, in which the CS changes the order of the protocol parameters such as the order of HTTP header rows, DHCP options, or FTP command sequences. The CR identifies the pattern by observing the item order [9, p. 97].

3. **Add Redundancy to PDUs**, where some bits, fields, or options that are not needed are added to the normal functionality of the protocol. Then, this additional space is filled with secret data [9, pp. 97, 98].

4. **Random Values in PDUs**, which exploits the pseudo random numbers present in many protocols, e.g., the Initial Sequence Number of the TCP protocol. These values will be changed inserting other data [9, p. 98].

5. **Value Modulation in PDUs**, in which the CS hides secret data by choosing a parameter from different legitimate values that a protocol can use. An analogy could be a person who writes two identical letters: one is written with a black pen, communicating the secret bit 0; the other with a red pen, communicating the secret bit 1. The content of the letter remains visible and apparently harmless, but whoever knows the pattern can read the embedded message [9, pp. 98, 99].

   Similarly, in a network scenario, the CS can exploit the TTL field present in the header of IPv4 packets: by agreeing beforehand on the meaning of specific values, e.g., TTL equal to 64 means bit 0 and the value 128 means bit 1, the TTL sequence 64, 128, 64 means 010 [9, pp. 98, 99].

   The CR interprets these values as agreed, and so an external viewer can just see legitimate protocol values. However, the choice of values should be in line with normal network traffic [9, pp. 98, 99].

6. **Reserved/Unused Bits in PDUs**, in which data are inserted in the header that network stacks do not usually interpret. It is the easiest technique to implement, but at the same time it is the easiest to detect, because the steganalyst can directly check those fields [9, p. 99].

2.3.2 Timing Channels

Timing channels exploit the temporal characteristics of network packets to transmit secret information. The CR observes the time it takes for a packet to travel from source to destination, or the delay between consecutive packets. By analyzing these timing variations, the receiver can extract embedded data from the overt communication.

The most common timing techniques are:

1. **Rate or Throughput of Network Traffic**, in which the CS varies the number of packets sent within a determined time interval. For example, a low transmission rate can represent bit 0, while a high rate represents bit 1. It is possible to use multiple frequencies to encode more symbols beyond binary encoding [9, p. 100, 101].

2. **Inter Packet Times**, where data are encoded into the time interval between two consecutive packets. To improve undetectability, the channel can mimic the time interval distribution of legitimate traffic [9, p. 101].

3. **Message Sequence Timing**, in which the CS manipulates the arrival time of a response, an ACK, or a command [9, p. 102].

4. **Artificial Message or Packet Loss**, where specific packets are intentionally discarded by the CS. The encoding could be: received packet as bit 0, dropped packet as bit 1. The CR must agree on an initial sequence number to remain synchronized with the CS, enabling correct distinction between intentional packet loss and random network losses [9, p. 104].

5. **Artificial Retransmissions**, where the CS generates duplicate packets or frames, and the number or simply the presence of the retransmission represents the hidden data. This technique is particularly effective for WLAN frames or DNS requests [9, p. 105].

6. **Manipulated Message Ordering**, where packets are sent in a different order from the original sequence. The CR uses the sequence number to reconstruct the correct order and interprets the altered order as the covert message [9, p. 105].

7. **Collision and Timing of Frames**, where in shared access networks (e.g., Ethernet or Wi-Fi), the CS can interfere with collisions, delays, and backoff mechanisms. The CR observes this behavior and decodes it into bits [9, p. 106].

8. **Indirect Timing Channels**, which consist of the CS not communicating directly with the CR, but instead modifying the behavior of an intermediate node, for example by increasing its load. The CR observes the resulting effects on the node [9, p. 108].

2.4 Memory Based Covert Channels

A memory based covert channel is an unauthorized communication mechanism that enables two malicious entities to exchange secret information through shared memory resources.

The covert sender encodes data by intentionally modifying memory related behaviour, for example through cache contention, accesses to specific DRAM banks, or changes in memory allocation. The covert receiver observes the resulting variations in access latency, execution time, or resource availability and uses them to reconstruct the hidden message.

Memory based covert channels can be widely classified according to the shared memory resource exploited to encode the hidden information. In particular, the secret message may be represented through the state and access timing of DRAM resources, through the state of shared cache lines, or through data embedded directly into shared memory structures and RAM resident objects.

2.4.1 DRAM Based Channels

In DRAM based memory covert channels, the CS and CR share the same hardware resource, which may be a memory controller, DRAM bank, row buffer, or memory bus. The CS encodes data by manipulating specific memory states, while the CR decodes the information by measuring memory access latencies.

Dynamic Random Access Memory (DRAM) is the most prevalent type of RAM in modern computer systems, serving as the primary volatile memory for real time data processing.

DRAM is organized into multiple banks, each containing rows and columns [10, p. 2]. Each bank has a row buffer that temporarily holds the contents of an activated row. The memory controller manages memory requests by issuing commands such as ACT (activate) to open a row, PRE (precharge) to close the active row, and RD or WR (read or write) to transfer data. When a memory request targets an already active row, it results in a row buffer hit, which can be served immediately. In contrast, when a request targets a different row, the controller must deactivate the current row and activate the new one, triggering a row buffer conflict or miss [10].

This mechanism forms the foundation of the covert channel: requests to the active row can be satisfied without additional activation, whereas requests to inactive rows require the controller to execute a precharge operation on the current row, activate the target row, wait for the DRAM timing constraints to be satisfied,

and then complete the read or write operation [11].

This produces two distinct access latencies depending on whether a row hit or row conflict occurs, as expressed by the formula

$$T_{\text{row hit}} < T_{\text{row conflict}} \tag{2.1}$$

[11]

As demonstrated in "Amplifying Main Memory-Based Timing Covert and Side Channels using Processing-in-Memory Operations" by Kanellopoulos [11], this latency difference amounts to approximately 74 CPU cycles, which constitutes a sufficiently large temporal gap to be detected by user space applications.

This represents the fundamental principle of DRAM covert channels: the CR infers the memory state by measuring access latencies. The encoding scheme follows a simple binary representation: if the CR observes interference in memory access patterns, it interprets this as bit 1; otherwise, it interprets it as bit 0 [11].

2.4.2 Cache Based Channels

Cache based channels exploit the CPU cache as a shared resource between the CS and CR: the sender modulates the cache state, and the receiver deduces information by measuring access times.

A cache is a small, high speed memory that temporarily stores data and instructions frequently used by the processor. By keeping this information close to the processor, it reduces the need to access slower main memory [12].

Each cache is organized into multiple memory levels, and each level is divided into several sets. Each set contains multiple cache lines, and each cache line includes four fields: a valid bit, a tag, a data block and, for those memories that use a write back policy, a dirty bit [13].

Write Through and Write Back policy Cache write policies define how modifications to cached data are propagated to main memory. Under a *write through* policy, each write operation immediately updates both the cache and the corresponding main memory location [12]. In contrast, modern CPUs commonly use a *write back* policy, in which updates are first made in the cache and are written to main memory only when the modified cache line is evicted or flushed.

When a write hit occurs, the new value is written only to the cache line and the corresponding dirty bit is set, while the main memory is not immediately updated.

The memory is updated later, when the cache line is evicted.

If the evicted line is *clean* (dirty bit equal to 0), its content is identical to the copy in main memory. Therefore, the cache controller can simply overwrite the line without generating any memory access, and no additional latency is incurred.

On the contrary, if the evicted line is *dirty* (dirty bit equal to 1), its content has been modified and differs from the copy in main memory. Before the line can be reused, the cache controller must perform a *write back* operation, writing the updated content back to main memory. This additional memory write introduces a measurable latency overhead compared to the clean case [13].

This asymmetry between clean and dirty evictions creates a timing signature that can be exploited to build covert channels. The sender can selectively render certain cache lines dirty by writing to the corresponding addresses, or leave them clean. The receiver then forces the replacement of those lines and measures the eviction latency: a low latency indicates a clean line, whereas a higher latency reveals a dirty line that required a write back. By associating these two conditions with logical values, the sender and receiver can exchange information without using an explicit communication channel.

Classification of Cache Based Covert Channels Cache based covert channels can be classified into two families: those that exploit the *hit/miss* status of cache lines, and those that exploit the *clean/dirty* state, commonly referred to as *write-back (WB) channels* [13].

In hit/miss-based channels, the sender modulates the presence of specific cache lines. To transmit a bit, the sender either accesses a set of addresses to bring the corresponding lines into the cache (encoding a logical 1) or avoids accessing them (encoding a logical 0). The receiver then looks at the same cache sets and measures the access time to a carefully chosen set of addresses. A low access time indicates a cache hit, meaning the lines are still resident, while a high access time indicates a cache miss, meaning the lines have been evicted. This mechanism is conceptually similar to DRAM timing covert channels, but operates at the cache level rather than at the row buffer level [14].

In dirty state based channels, or WB channels, the sender modulates the dirty bit of cache lines instead of their presence. To transmit a bit, the sender writes to selected addresses, marking the corresponding cache lines as dirty, or avoids writing, leaving them clean. The receiver then forces the eviction of those lines,

for example by accessing an eviction set that maps to the same cache sets, and measures the latency required to complete the replacement. A low eviction latency indicates that the lines were clean and could be overwritten without memory access, whereas a higher latency reveals that the lines were dirty and required a write back to main memory. By associating clean and dirty states with logical values, the sender and receiver can exchange information by exploiting the timing difference induced by the write back mechanism [13].

2.4.3 Memory Structure and Shared Memory Covert Channels

In addition to timing based covert channels that exploit DRAM and cache latencies, another important class of memory covert channels leverages the content and organization of data structures in RAM. These channels, often referred to as *memory structure* or *shared memory* covert channels, that summarize other subclasses, encode information directly in the values, fields, or layout of shared memory objects rather than in access timing.

In these schemes, the CS and CR share some form of memory resource, such as shared memory segments, kernel data structures, system buffers, queues, or tables accessible by multiple processes. The sender modulates specific attributes of these structures, and the receiver infers the transmitted information by reading the same memory locations. Unlike DRAM based and cache based timing channels, where the information is represented by measurable latency differences, memory structure channels encode data in the actual content or configuration of memory objects.

Typical mechanisms are the modification of redundant or rarely used fields, such as the least significant bits of counters, identifiers or timestamp. An example of exploiting timestamp field is presented by Neuner et al. in "Time is on my side: Steganography in filesystem metadata" [15]. In their Time On My Side system, the authors exploit the redundant sub second portion of file timestamps in NTFS, each timestamp includes a nanosecond component that is not normally visible to users or fully utilized by the operating system. They replace these nanosecond fields with encrypted secret data, using the gap between the high precision stored on disk and the lower precision exposed to users and applications to hide information in a way that is statistically indistinguishable from normal filesystem activity [15].

Another approach involves modulating the presence, absence, or ordering of objects in memory, such as entries in a queue, records in a buffer, or files in a directory, where the existence or position of an object represents logical values. This class of techniques is closely related to access pattern attacks [15], which

exploit predictable memory access behaviors and shared memory structures to infer or influence sensitive information. As discussed by Hassan et al., access pattern attacks (APA) target vulnerabilities in page tables, secure enclaves, and memory deduplication mechanisms, observing how memory is accessed rather than relying solely on timing variations [16].

The survey highlights [16] that these channels can be particularly stealthy, as they do not necessarily induce significant timing anomalies and may require analysis of data content or structural properties for detection. Furthermore, it discusses various countermeasures,including normalization of data structures, validation of metadata, and monitoring of shared memory access patterns.

Memory structure and shared memory covert channels thus represent a complementary approach to timing based memory channels, demonstrating that not only access latencies but also the content and organization of memory objects can be exploited to establish hidden communication paths between processes.

2.5 Limitation Resource Based Classification

Both network and memory covert channels face significant limitations when classified solely by the shared resource they exploit.

Network timing channels are highly sensitive to network conditions: jitter, congestion, variable delay, packet loss, and reordering can all compromise message decoding. They require strict synchronization between sender and receiver, and to avoid detectable statistical anomalies, they must limit timing variations and transmitted data volume, often reducing available bandwidth [9].

Network storage channels, while more stable, are easily detected when anomalous reserved or unused fields are manipulated, and their capacity is typically limited to a few bits per packet. Moreover, these secret bits may be altered or destroyed by NAT, proxies, firewalls, and middleboxes that normalize or rewrite packet headers [9].

Similarly, memory channels face architectural dependencies: hybrid techniques may exploit both cache and DRAM characteristics simultaneously, making rigid classification difficult.

The rapid evolution of memory technologies and sophisticated mitigation techniques further challenge traditional taxonomies.

These limitations highlight a fundamental problem: classifying covert channels by the resource they exploit ties the taxonomy to specific implementations and hardware architectures, making it difficult to generalize across domains or compare techniques that use different resources but share similar underlying principles.

This motivates the need for a new taxonomy that should no longer describe where the steganographic technique is applied, but rather how the information is hidden and represented [17].

2.6 Generalized Taxonomy

The limitations of resource based classification highlighted in the previous section motivate the need for a more abstract and domain agnostic taxonomy. The previous taxonomy was designed to address the field of network steganography, classifying methods based on the protocol or medium used. However, this approach resulted in a protocol dependent taxonomy: similar techniques were classified under different names across different domains by different researchers. Moreover, most hybrid techniques, i.e., those employing multiple domains, were difficult to classify. Most importantly, the continuous evolution and innovation in network, filesystem, text, media, cloud, and AI domains rendered the domain based classification increasingly inefficient.

Therefore, the innovative approach presented in the paper "A Revised Taxonomy of Steganography Embedding Patterns" by Steffen Wendzel et al. [17] proposes that the **taxonomy should no longer describe where the steganographic technique is applied, but rather how the information is hidden and represented.**

A hiding pattern is defined as an abstract and generic description of a hiding method. The key innovation lies in the generalization of the taxonomy: the same pattern can be applied to a network packet, file, image, text, or cyber physical system.

The goals that the paper's authors aimed to emphasize are threefold:

1. the introduction of a unified terminology for techniques originating from different domains;

2. avoiding the reinvention of already existing techniques;

3. enabling the comparison of different methods that use different carriers but share similar underlying concepts.

[17, p. 3]

The new taxonomy is structured around two fundamental classification dimensions: the **embedding vs. representation distinction** and the **temporal vs. non temporal classification** [17, p. 5].

The first dimension addresses *how* the information is handled in the steganographic process.
Embedding patterns describe the actions performed by the sender to insert secret data into the cover object. They answer the question: "Which action takes place to embed the secret data?" The taxonomy presented in [17, p. 5] identifies several general objects that can be manipulated: an **event** is something that happens (occurs), while an **element of a sequence** is an individual item within a sequence, such as a packet or a character.

A **feature** is an item property, such as the size or colour of an image, whereas an **interval** represents the time between two events. Finally, a **state or value** represents the current condition or numerical value associated with an object.

Regarding the object, the main embedding actions are: **interval modulation**, **occurrences**, **element loss**, **positioning** (e.g., of an element), **enumeration** (e.g., of the number of occurrences), **state or value modulation**, and **feature structure modulation**. [17, p. 10]

In contrast, **representation patterns** describe how the hidden message appears or how it is represented in the cover object. They answer the question: "Which event does the receiver observe to reconstruct the message?" This distinction is crucial because the sender's embedding mechanism may differ from what the receiver actually observes: for instance, a sender might embed data by manipulating packet timing (embedding), but the receiver might detect this as a pattern of packet loss or retransmission (representation) [17, p. 5].

The second dimension addresses *when* the information is encoded. **Temporal patterns** exploit time related properties, such as intervals between events, timing of packet transmission, or occurrence frequency. These patterns are analogous to the traditional timing channels but are generalized to apply across all domains (network, filesystem, multimedia, etc.). **Non temporal patterns**, on the other hand, exploit properties that do not depend on time, such as element values, positions, structures, presence or absence. These patterns are analogous to the traditional storage channels but, again, are domain agnostic [17, p. 6].

By combining these two dimensions, the taxonomy provides a comprehensive

framework that can describe any steganographic technique regardless of the underlying domain, enabling systematic comparison and analysis across different fields.

Naming Convention To avoid having different names for equivalent techniques and to ensure uniformity across patterns, a new nomenclature is introduced. It is composed of:

- **Prefix (E or R)**: indicates the mechanism used. If it is **E** (Embedding), it describes how the secret data is embedded into the cover object. If it is **R** (Representation), it indicates how the message is observed by the receiver.
- **Category number**: identifies the general category of the pattern (e.g., State/Value Modulation, Element Positioning, etc.).
- **Subcategory number**: specifies a more detailed variant within the category (e.g., LSB modulation, feature modulation, etc.).
- **Domain indicator (optional)**: a letter that indicates the application domain, such as **n** for network, **f** for filesystem, **t** for text, **m** for multimedia, etc.
- **Instance number (optional)**: a final number that identifies a specific variant or instance of the pattern within the domain.

[17, p. 6]

The numerical codes in the taxonomy do not describe operational parameters of the technique, but rather indicate its hierarchical position: the first number identifies a general category, while subsequent numbers identify increasingly specific subcategories.

For example, the code **E2.1n1** identifies a network embedding technique based on element enumeration. In this technique, the sender transmits selected packets more than once to imitate TCP retransmissions. The secret information is represented by changing the number of duplicate packets observed by the receiver: the presence or absence of a duplicate packet, or the number of repetitions, can encode different logical values [17, p. 6].

A memory based example can be associated with the code **E2.2m1**, which identifies an embedding variant based on the temporal positioning of memory events. In a DRAM based covert channel, the sender deliberately accesses selected rows or memory locations to create row buffer hits or conflicts. The receiver measures the resulting access latency and interprets the temporal position of the observed memory events as the hidden message [17, p. 6].

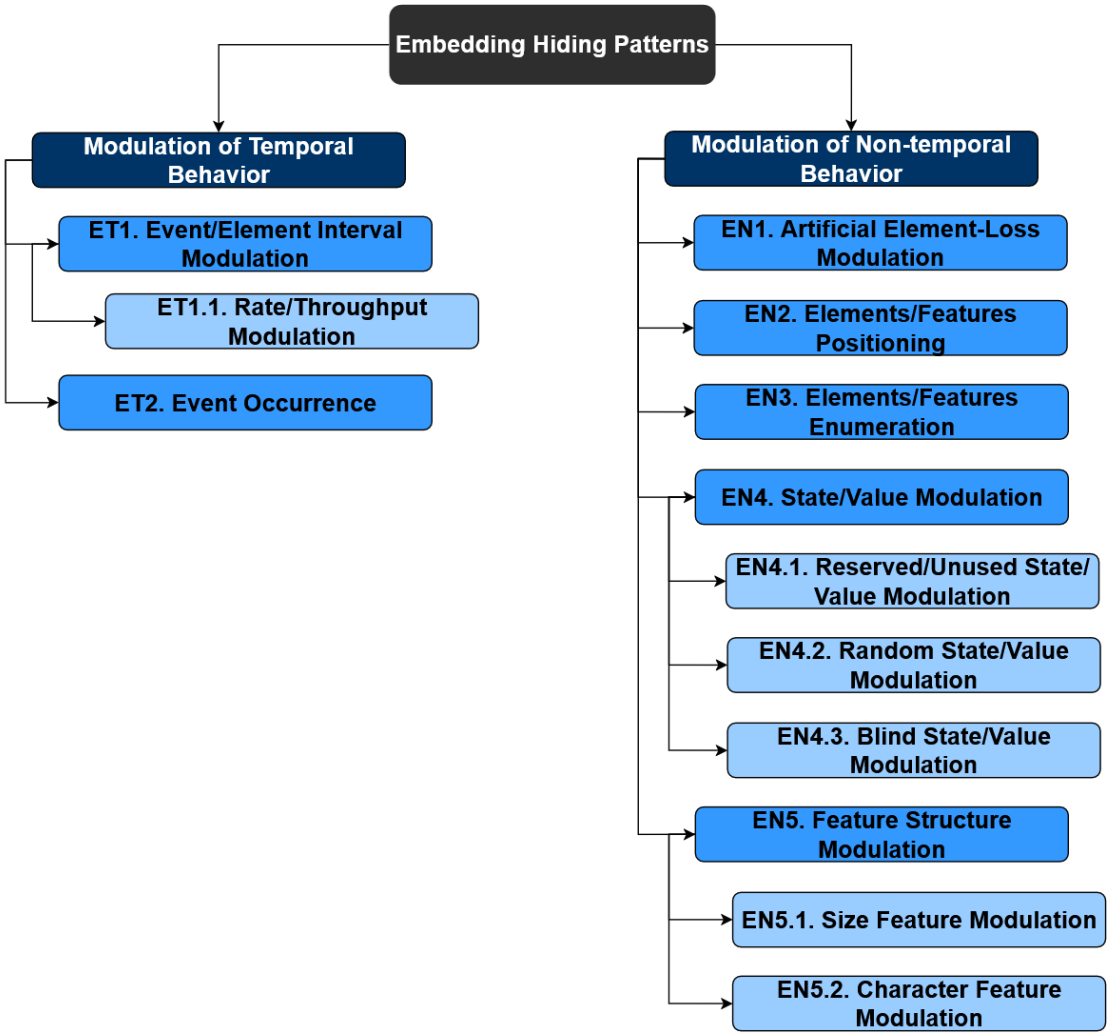


Figure 2.1: A Revised Taxonomy of Steganography Embedding Patterns. Reproduced from Wendzel et al. [17, p. 8], licensed under CC BY 4.0.

2.7 Conclusion Remarks

This chapter introduced the fundamentals of steganography and covert channels, with particular focus on steganography network covert channels that manipulate storage structure and timing patterns of network traffic and steganography memory based covert channels that exploit DRAM timing, cache states, and shared memory structures.

The analysis revealed a key limitation: traditional resource based taxonomies tie classification to specific domains and hardware architectures, making it difficult

to compare techniques that share similar principles but operate on different media.

This motivated the adoption of Wendzel et al.'s generalized taxonomy [17], which classifies steganographic methods based on how information is hidden and represented rather than where it is embedded. This domain agnostic framework enables systematic comparison across network, memory, filesystem, and multimedia domains, providing the conceptual foundation for the forensic detection methodology presented in the following chapters.

Chapter 3

Forensic Classification and Detection Methodologies

3.1 Paradigm Shift: from "carrier medium" to "trace nature"

Traditional covert channel taxonomies are based on the carrier medium, an approach that is useful for implementation but inadequate for forensic detection. The recent work by Wendzel et al. [17] addressed this limitation by proposing a domain agnostic taxonomy that classifies techniques not by where they are applied, but by how information is hidden and represented through embedding and representation patterns. While this abstraction successfully unifies terminology across different steganography domains, it remains focused on the steganographer's perspective rather than the detector's.

For forensic detection, a further shift is required: instead of asking "*where is the data hidden?*" (carrier medium) or "*how is the information hidden?*" (embedding pattern), the forensic analyst must ask "*how does the anomaly manifest?*" (**trace nature**).

What is a Forensic Trace? Following Roux et al. (2022), a forensic trace is defined as "the remnants of past activities (such as an individual's presence and actions)" that can be studied "through their detection, recognition, recovery, examination and interpretation to understand anomalous events of public interest (e.g., crimes, security incidents)" [18, 19, p. 2]

For example, a timing channel on ARP and a timing channel on DNS can both

manifest as a statistical deviation in inter packet delays, even though they use different protocols. This classification, based on trace nature, is more useful for the forensic analyst, because it allows detector prioritization based on the observed trace and supports cross layer correlation based on trace nature, rather than on the specific protocol.

This trace centered perspective identifies the types of anomalies that should be sought across different layers of a system. However, detecting such anomalies is not sufficient on its own: they must be collected, preserved, verified, and presented according to a rigorous forensic procedure. The following section therefore introduces the digital forensic investigation framework adopted in this thesis, which provides the methodological structure for handling throughout the investigation.

3.2 Digital Forensic Investigation Framework

Digital forensics investigations follow a structured five phase framework. The framework identifies these phases as: Identification, Collection, Acquisition, Evaluation and Presentation. Although, this structured approach is based on the coursework provided by the thesis supervisor, it is fully consistent with standard digital forensic principles. This framework was intentionally adopted to align the investigation with the supervisor's guidelines.

Phase 1: Identification consists in recognizing all potential sources of digital evidence that may be relevant to the investigation [20, slide 26]. This activity is critical and must be carried out before the acquisition phase, as it provides a comprehensive overview of all elements to be verified prior to entering the practical stages of the forensic process. In the context of covert channel detection, identification includes locating network monitoring points (such as packet capture interfaces and temporal metadata sources), identifying suspect hosts (by examining active network connections and running host processes), and determining whether volatile or non volatile memory should be acquired when the channel exploits RAM. It also involves collecting information and logs from higher level system components [20, slide 26, 27].

It is essential to recall that digital evidence exhibits different orders of volatility. Data in RAM and active network connections can disappear immediately upon device shutdown; cloud based or synchronized data can be modified or overwritten remotely; whereas data stored on disk are generally more persistent. The concept of volatility is crucial because it directly influences the prioritization of subsequent acquisition steps: the most volatile data must be collected first [20, slide 27]. In

summary, the first phase consists in generating a structured map of all possible evidentiary sources and their interrelations.

Phase 2: Collection focuses on the systematic collection of evidence while preserving its integrity. The primary objective of this phase is to implement procedures that minimize the risk of evidence tampering, alteration, or data loss [20, slide 28].

To achieve this, the first action is to isolate the devices under investigation to prevent remote interference. This typically involves disconnecting them from all network connections. For mobile devices, specialized containment solutions such as Faraday bags should be used to block all wireless communications and prevent remote access.

Every decision and action taken during this phase must be documented in a detailed record describing the condition and state of the evidence. This record should specify which devices were powered on, whether screens were active or locked, and any other observable indicators. A well documented report is essential to prevent legal challenges concerning the integrity and admissibility of the evidence, as it provides a clear justification for each procedural choice [20, slide 29].

Once isolation and documentation are complete, the next step is to prepare the environment for acquisition in a manner that ensures no alteration of the original data. For physical storage devices, this is commonly achieved using hardware write blockers[1]. Additionally, all network connections, automatic synchronization services, and system notifications should be disabled to prevent unintended modifications.

It should be noted that this last requirement presents a particular challenge in scenarios involving live systems or remote acquisition, where maintaining full integrity while the system remains operational is inherently more complex [20, slide 31].

Phase 3: Acquisition ensures that evidence is extracted and duplicated from the collected sources in a manner that guarantees the acquired data is a faithful replica of the original, while simultaneously preserving data integrity. This phase involves creating bit stream copies (forensic images) rather than operating directly

[1]A write blocker is a hardware or software device that prevents any data from being written to a storage medium during forensic examination. This ensures that the original evidence remains unaltered and enables the acquisition of forensically sound, legally defensible copies [20, slide 19].

on the original media [20, slide 32].

The forensic value of covert channel evidence heavily depends on its timely and controlled preservation. Unlike conventional evidence stored only on disk, covert channel indicators may exist in volatile memory, transient network activity, or persistent host artifacts. Such evidence can be modified, overwritten, or lost through normal system operation, device shutdown, synchronization mechanisms, or remote access. Therefore, the acquisition process must minimize its impact on the original environment while producing evidence whose integrity and handling can be subsequently verified.

It is important to note that the acquisition procedure varies depending on the type of evidence being collected. Live acquisition must be employed for volatile data, such as RAM contents, whereas static acquisition is appropriate for non-volatile data, such as hard disks. Specific methods for each scenario are presented in Section Taxonomy of Traces (3.3) [20, slide 33].

When dealing with **volatile evidence (RAM)**, which contains runtime artifacts including active processes, network connections, injected code, and encryption keys, it is essential to select a method that minimizes system interference and avoids tampering with the evidence. A common approach consists of performing a RAM dump, also called memory acquisition, together with capturing data from running processes and active network connections, including network traffic [20, slide 35].

Conversely, **non volatile evidence (disk)** includes all evidence that is permanently stored on non volatile storage and does not vanish when the device is powered off. The first operation is to shut down each device carefully. For encrypted devices, a RAM dump must be performed before shutdown to retrieve any decryption keys, which are typically stored in memory. The device must then be isolated, for example by using a write blocker, and a forensic imaging tool should be used to create a complete image of the storage device. Finally, the forensic image should be stored on a secure forensic storage device [20, slide 35].

In all cases, cryptographic hash functions (e.g., MD5, SHA-256) should be used to verify and document data integrity before and after acquisition [20, slide 36]. This **hashing process** converts data into a fixed length string of bits that uniquely represents the original data. It ensures the integrity of digital evidence: if, during an investigation, a file is altered, its hash value will no longer match the value computed when the evidence was acquired. Once tampered with, the evidence is no longer admissible in court. Therefore, hashing must be performed every time

evidence is handled, for example both when it is copied and when it is moved, because it forms the foundation of the chain of custody [20, slide 18].

Chain of custody documentation, which records every instance of evidence handling, is essential to maintain legal admissibility. Particular attention must be paid to this aspect, as any weakness or gap in the chain of custody can compromise the admissibility of evidence in legal or forensic proceedings [20, slide 37]. As presented by Prof. Atzeni in [20, slide 16], the chain of custody is *"a documented and unbroken process of handling evidence from the time it is collected until it is presented in court"*. The chain of custody guarantees that digital evidence has not been changed, damaged, or accessed by unauthorized people. To keep the chain of custody valid, investigators must know how to correctly document how evidence is collected, stored, and who accesses it. Collecting digital evidence from devices without changing or harming the original data is very difficult; it requires knowledge and skills in many areas, such as disk imaging, capturing live data, understanding file systems, using hash functions, working with write blockers, and using forensic acquisition and analysis tools correctly [20, slide 16].

Phase 4: Evaluation examines the preserved evidence to ensure its integrity, authenticity, and admissibility. At this stage, the evidence is analyzed, verified, and validated to confirm that it remains unaltered and trustworthy for legal proceedings or investigation [20, slide 38].

This phase includes comparing freshly computed hash values with those recorded at acquisition, validating and documenting the chain of custody at each transfer step, and verifying forensic images to confirm the absence of alterations or gaps. For live data, timestamp and metadata analysis should be performed to verify file creation, access, and modification times against the incident timeline[20, slide 39, 40].

Cross reference and consistency checks are also required, correlating digital evidence with external logs or other data sources (e.g., logs, emails) to confirm that it relates to the suspected system or device. The entire process must comply with applicable legal and internal standards; collection, preservation, and evaluation procedures must be coherent with current legal requirements, and documentation must demonstrate this compliance. Finally, the evaluation should include a review of possible anti forensic techniques, such as steganographic methods [20, slide 40].

Phase 5: Presentation consists in preparing and presenting the investigation findings in a clear, accurate, and legally admissible form for judicial proceedings. The objective is to translate the technical results of the forensic analysis into a

format that can be understood by non technical stakeholders, such as lawyers, judges, or corporate executives. Since the quality and clarity of the presentation can significantly influence the outcome of legal proceedings or internal investigations, this phase is very critical [20, slide 41].

At this stage, all data collected, analyzed, and interpreted during the investigation must be reviewed, identifying the key points and verifying that all conclusions must be explicitly linked to verifiable evidentiary elements. The entire forensic process must be documented in a formal report. Such reports should include technical glossaries, timeline tables, and flowcharts to facilitate comprehension by non technical audiences, while excluding any irrelevant or speculative material [20, slide 42].

The framework described above defines how digital evidence should be identified, collected, acquired, evaluated, and presented. In covert channel investigations, particular attention must be given to the technical procedures used to acquire and preserve volatile, non volatile, and network evidence. Once these have been properly acquired and preserved, the investigator can examine them for anomalies associated with covert communication. Since such anomalies may appear across different layers and data sources, their analysis should not rely only on the underlying carrier medium. The following section therefore introduces a trace based classification of covert channel detection methodologies, distinguishing between signature based, statistical, and latent or heuristic traces.

3.3 Taxonomy of Traces

As previously discussed, we can introduce a classification that explicitly adopts the steganalist's perspective, based on the type of trace that a covert channel leaves in the carrier. This leads to a taxonomy of signature based, statistical, and latent or heuristic based traces [21, 22].

3.3.1 Signature based Traces

Signature based traces are characterized by known patterns associated with specific hiding techniques, such as characteristic byte sequences, fixed headers, or recognizable text signatures embedded by steganographic tools. In practice, these signatures can be searched for using existing forensic utilities and rule based detectors, which are effective when the suspect files were produced by well known steganography software [21].

Some common examples include:

- Intentional **manipulation of protocol header fields**, such as IP timestamp, TCP flags, or ICMP payload. Network protocols are defined by RFC compliant standards[2], and each header field has an expected value, a valid range, and a specific meaning according to the RFC specification [24].

  A covert channel, such as E1.3n1 (Reserved/Unused State/Value Modulation) in the Wendzel taxonomy [17], can manipulate these fields in ways that violate RFC specifications. This generates signature based traces that can be detected through deep packet inspection, which verifies conformity to RFC policies.

  A similar example involves unused or reserved bits: by definition, any non zero value in these fields represents suspicious behavior [24].

  Another example of intentional manipulation of protocol field is through magic bytes manipulation: Magic bytes are specific to binary files and rely on matching signatures which are varying in length in file headers or tails. Due to inadequate standards for file content, file format creators include headers for uniquely identifying files of their type [25]. A covert channel may exploit this feature by inserting magic bytes into protocols or payloads that are not expected to carry them, such as embedding file signatures within network packet payloads. In forensic analysis, if magic bytes match the expected signature, the file is identified as that specific format; otherwise, it is treated as suspicious. However, it works only on binary file types having magic bytes

[2]RFC (Request for Comments) specification is a formal official document from the IETF that defines how Internet technologies, protocols, and standards work [23].

associated with them, representing a limitation for ASCII based file detection [25].

- Presence of **NTFS Alternative Data Streams (ADS)**. NTFS (New Technology File System) is the default Windows filesystem, which supports alternative data streams. This feature allows a file to handle multiple data streams: the primary stream is visible in directory listings, while alternate streams are hidden and do not increase the visible file size. This creates a discrepancy between the declared file size and the actual disk space consumed, which represents a deterministic trace recognizable through filesystem forensic tools [26].

Signature steganalysis is effective for detecting known tool signatures: some tools like Aletheia[3] can detect traces left by steganographic tools such as Outguess, nsF5, Steghide [22].

Deep packet inspection and traffic monitor can be performed using tools such as Wireshark and Windump, to catch behaviour that differ from the normality. Finally, filesystem forensic analysis can detect malicious use of NTFS Alternative Data Streams through enumeration of all streams associated with a file, not only the primary data stream [26].

Advantages and Disadvantages The main advantage of signature based methods is the specificity in identifying known patterns introduced by steganography tools. From an explainability perspective, this makes the detection process easier to interpret and demonstrate, as the evidence consists of concrete, well defined byte sequences or structural artifacts that can be directly shown and verified.

However, the majority of existing network steganography techniques employ single layer hiding mechanisms, which are inherently limited in terms of bandwidth, exhibit low robustness, and are more easily detected by modern intrusion detection systems [26].

The method cannot detect zero day or custom covert channels that do not match known patterns [22]. Furthermore, effectiveness decreases under traffic normalization countermeasures, such as packet reordering or header sanitization, which can obscure or eliminate structural anomalies [26].

When covert channels avoid structural violations, the next step is to consider statistical or behavior based detection methods.

[3]Aletheia, is a novel steganalysis tool with multiple possible options, such as performing calibration, RS and other attacks [22]

3.3.2 Statistical Traces

Statistical traces are characterized by measurable deviations from established baselines. Unlike signature based traces, they do not appear as repetitive patterns but as numerical irregularities in statistical distributions, correlations, or entropy measurements [21].

Common examples include:

- **Timing anomalies**: these are deviations in the timing distribution of network events. A common example is the deviation in inter packet delays, where the time interval between two consecutive packets differs from the baseline distribution [27]. Different delay values can represent different symbols, such as bit 0 and bit 1, creating detectable deviations from legitimate traffic patterns. Other examples of timing anomalies include packet ordering issues and burstiness patterns, which operate in a similar manner.

- **Entropy changes**: entropy measures how random a data sequence is. More precisely, entropy is defined as

$$H(X) = -\sum_{i} P(x_i) \log_2 P(x_i) \tag{3.1}$$

  [28]

  where $P(x_i)$ is the probability of occurrence of the event (byte) x_i.

  It is a measure of the amount of uncertainty, unpredictability, or randomness associated with a random variable or a message in the payload content. In image steganography, it quantifies the average information content of a data source in bits. Since information is added into the cover image by modifying its original contents, randomness may either increase or decrease [28].

Detection methods correspond to the types of statistical traces identified above, and they include entropy and joint entropy analysis, Chi Square and RS steganalysis [28, 22]. These techniques are implemented in steganalysis tools such as StegDetect and Virtual Steganography Laboratory (VSL). They are particularly valuable when only the stego object is available. [22]

An information theoretic approach uses **entropy and joint entropy** as detection measures for LSB steganography. As reported in the paper [28], applying entropy and joint entropy techniques yields detection rates of 70% and 98%, respectively, before the histogram attack[4]. After the attack, the detection rate decreases

[4]In this scenario, the histogram attack refers to a normalization procedure of the stego image

to approximately 30% for entropy analysis and 93% for joint entropy analysis, highlighting their sensitivity to this type of countermeasure. [28]

Chi square analysis is a statistical method which compares the expected distribution of the data to the observed distribution [29] and the related formula is

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{3.2}$$

[29] where O_i represents the **observed frequency**, E_i represents the **expected frequency**, and n represents the number of categories or intervals considered.

This technique, as presented in the paper by Michaylov [22], is mostly used in image steganography for LSB embedding. Before the embedding of secret values in an image, adjacent pixels typically have similar values, producing a natural frequency distribution. In the embedding scenario, the image shows anomalous frequencies that differ from the original ones [22].

The analysis concentrates on Pairs of Values (POVs) exchanged during secret data embedding inside the digital image, where POVs could be pixel values, quantized DCT coefficients, or palette indices [22].

If hidden information is present, the embedded information alters the histogram of colour frequencies in a particular way, with POV frequencies situated further from the mean POV. A high χ^2 value indicates a large variance between expected and observed data [22].

RS analysis is another method for classifying statistical traces, and it is mainly applied in image steganalysis. The technique is based on splitting the image into pixel groups and classifying them as regular or singular, based on their sensitivity to LSB flipping. A group is tagged as regular if LSB flipping increases the group noise; otherwise, it is tagged as singular. In a normal image, the ratio has a specific value based on the image features; so the analysis is based on searching changing in the noise distribution of the pixel groups [22].

However, after LSB embedding, this ratio decreases significantly, revealing the statistical alteration caused by the embedding. RS analysis is able to detect embedding rates higher than 0.005 bits per pixel. This means that covert channels

---

histogram, altering the pixel value distribution. The goal of the paper is to verify the robustness of the entropy methods against this attack [28].

which embed small amounts of data can evade RS analysis, so this technique cannot detect smaller values than this threshold [22].

Advantages and Disadvantages Statistical detection methods complement the signature based approaches, providing a second layer of defense against covert channels that avoid structural violations. This trace category offers the ability to detect unknown techniques without specific signature knowledge. In fact, statistical steganalysis outperforms signature based methods in terms of effectiveness and robustness, when only the stego object is available.[22].

Despite that, chi square analysis reliably detects messages that are sequentially embedded, but does not provide high accuracy for randomized ones. As already presented, another limitation concerns the effectiveness of RS steganalysis: if the embedded message inside the image is less than 0.005 bits per pixel, it is undetectable by RS steganalysis [22].

Overall, existing image steganalysis techniques still cannot universally detect hidden messages, and it is currently impossible to utilize an algorithm with both low computational requirements and high accuracy.

But what if statistical techniques are evaded? In this case, latent and heuristic techniques can be employed.

3.3.3 Latent/Heuristic Traces

The term "latent" refers to the fact that these traces are not directly observable in the raw data, but exist in a latent space learned by the model during training. "Heuristic" indicates that detection rules are learned from data rather than predefined by experts [30].

Latent and heuristic traces are not traces in the traditional sense, but rather detection approaches used to identify traces when deterministic and statistical analysis fail. In this section, the detection methods themselves will be analyzed directly. These approaches require machine learning (ML) or deep learning (DL) models for identification [31, 32].

Examples include:

- **Deep learning steganalysis systems**: these systems use Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNN), and autoencoders to detect hidden data in both the spatial domain and JPEG images. Recent surveys present that the CNN architecture can autonomously learn spatial features from images, reaching a detection accuracy higher than 90% for modern steganographic techniques [31].

For example, the study proposed by Bravo-Ortiz et al. [33] introduces CVTStego-Net[5], a convolutional vision transformer architecture specifically designed for spatial domain image steganalysis.

- **Behavioral ML models**: these models analyze network traffic flows to identify behavioral anomalies related to covert channels. A study conducted in 2023 by Li, Song & Yang [27] proposed a generic and sensitive detection approach for network timing covert channels, implementing neural networks to learn legitimate patterns of inter packet delay variation and identify anomalous samples as covert traffic. According to the paper Li, Song & Yang (2023) affirm that, *"The experimental results show that our approach effectively detects all currently representative channels in the absence of their knowledge, presenting once to twice higher sensitivity than the state-of-the-art solutions"* [27, p. 1].

  Similarly, DNS covert channel detection frameworks utilize ML models with customized behavioral profiles for network users, allowing the identification of zero day attacks without any known signature [34].

- **Feature learning approach**: in this sector, DL models are particularly valuable because they autonomously extract features from raw data, avoiding the need for feature engineering required by traditional ML methods, such as SRM, SPAM, and PHARM [6] used for steganalysis [31].

  Similarly, autoencoders have been applied to identify covert channels in IoT systems, learning normal traffic patterns in a latent space[7] and reporting all reconstruction errors as anomalies [32, 36].

Advantages

Higher detection rates against modern techniques Deep learning excels at detecting advanced steganographic techniques, including GAN based steganography and adaptive embedding methods that evade traditional statistical analysis.

[5]It is a convolutional vision transformer for spatial domain image steganalysis that merges the strengths of convolutions and the advantages of attention mechanisms to capture both local and global dependencies [33].

[6]They are specific feature extraction sets: SRM (Spatial Rich Models) is an advanced high-dimensional spatial models used to detect steganography by analyzing relationships between neighboring pixels; SPAM (Subtractive Pixel Adjacency Matrix) is a subtractive pixel adjacency matrix, i.e., a set of traditional statistical descriptors used to identify alterations in multimedia files; PHARM (Phase Aware Projection Model) is a phase aware projection model used to analyze images and uncover the presence of hidden payloads. [35, 31]

[7]A latent space is a compact representation of raw data. This compression process helps to identify the important features of the input (also known as feature map) [30].

This capability is particularly evident in CNN based detectors, which demonstrate superior performance against modern algorithms specifically designed to resist conventional steganalysis.
The advantage stems from deep learning's ability to learn complex feature representations that capture subtle modifications invisible to handcrafted statistical features [31].

Adaptability through continuous learning A key strength of ML based detection lies in its adaptability: models can be continuously updated with new training data to keep pace with the evolution of new covert channels. Furthermore, unsupervised learning[8] allows the identification of zero day attacks without any prior knowledge of specific signatures[34, 32].

Detection of complex non linear patterns Beyond adaptability, deep learning models can identify intricate feature correlations that statistical methods cannot capture. For instance, ML models can detect timing jitter anomalies in encrypted traffic, which are completely invisible to traditional analysis [27].

Disadvantages

Low explainability Although ML and DL methods can achieve high accuracy and precision, their limited explainability poses significant challenges in legal contexts. If these methods cannot be clearly explained to a judge, they will never be admissible as evidence in court proceedings [38].

Training data requirements In addition to explainability concerns, effective ML detection requires large labeled datasets containing both cover and stego samples across diverse tools and payload sizes. However, public comprehensive labeled datasets for network steganography are very limited and do not encompass full diversity, forcing researchers to create their own. This limitation is significant, as limited or biased training data leads to poor generalization in real world scenarios [31, 39].

Overfitting risk Closely related to the data requirement problem is the risk of overfitting, where ML models may memorize training patterns rather than learning generalizable principles. Consequently, models learn the training data too closely,

[8]Unsupervised learning is a framework in machine learning where, in contrast to supervised learning, algorithms learn patterns exclusively from unlabeled data. [37]

and when applied to unseen techniques or different image characteristics, they fail to generalize [31]. This problem is particularly dangerous in forensics, where real cases often differ from controlled training scenarios.

3.4 Detectability vs Explainability

As already seen in the section Disadvantages (3.3.3) of Latent based traces, specifically in Low explainability, the trade off between detectability and explainability is the central challenge in modern forensic detection system design.

Detectability measures an algorithm's ability to identify covert channels, i.e., using accuracy, recall, and F1-score, while explainability measures the analyst's ability to explain why an anomaly was detected, establishing causality and producing court admissible evidence [38].

The black box problem AI systems used in forensic investigations face significant explainability challenges. UNI-CRI and INTERPOL define explainability as follows: *"Explainability (in a narrow sense) refers to the ability of developers and users of an AI system to understand its functioning, meaning how the system makes decisions or generates outputs. It focuses on the inner workings of the AI system, its internal logic or underlying processes"* [38].

This means that explainability is not just about achieving accurate results, but about being able to understand and articulate the reasoning process behind those results. In forensic contexts, this distinction is critical: a detection system must not only correctly identify anomalies, but also provide a clear principle that can be communicated to and understood by human analysts and judicial authorities.

The "black-box" nature of DL models not only limits explainability but also generates legal challenges for evidence admissibility. As underlined by UNI-CRI and INTERPOL, 2023: *"While the requirements for evidence admissibility are different in each jurisdiction, a sufficient degree of explainability needs to be ensured for any AI system used to obtain and examine criminal evidence"* [38, p.587].

In practice, this means that judges may reject AI based conclusions without causal explanations, particularly in criminal proceedings where evidence must withstand cross examination [38].

Recognizing that ML or DL high detectability alone is insufficient when results cannot be explained in court, this thesis moves beyond single method detection toward a multi source forensic approach, which is introduced in the next section.

3.5 Information Fusion: Multi Source Evidence and Cross Layer Correlation

Analysis based on a single type of trace, such as those presented in Section Taxonomy of Traces (3.3), is no longer sufficient. A more effective approach is to introduce multi source forensic analysis based on causal correlation between heterogeneous evidence. By correlating traces across different layers, such as network, host, and memory, it becomes possible to reconstruct, from different perspective, not only that an anomaly occurred, but also how and why it arose.

The key insight is that a covert channel, which may appear invisible or ambiguous at one layer, can leave complementary traces at others. Correlating these traces can confirm the presence of the channel or reveal malicious intent that would remain hidden in a single layer analysis.

Correlation Models The core principle of information fusion lies in correlating heterogeneous traces to infer causal relationships. We may identify three complementary correlation models:

- **Temporal correlation:** captured data are divided into fixed duration time windows (e.g., 10 seconds). Data from different layers are then aligned to the same temporal intervals to identify co occurrences between suspicious behaviours. For example, a command execution trigger on the host may coincide with a burst in network traffic.
- **Entity based correlation:** This model maps relationships between entities such as processes, sockets, IP addresses, ports, and files, reconstructing how processes interact with network activity. The resulting structure provides a high level overview of how different components interact during a potential covert communication.
- **Causal pattern recognition:** This approach identifies patterns that deviate from normal behaviour or that are consistent with known covert channel strategies. Some example may include DNS query bursts aligning with file access events and DRAM row activation latency spikes, or inter packet delay patterns co occurring with specific process scheduling events.

This multi source analysis framework will be applied to the timing covert channel described in Proof of Concept (4), specifically in sections 4.3.4 and 4.4.2, which exploits inter packet delays between two ARP messages in the communication. We will evaluate whether cross layer correlation improves detection efficiency and effectiveness from a forensic perspective.

Related Work An example from literature of a multi source approach, not exactly in covert channels but still in the forensic domain, can be found in the work of Fu, Du, and Luo on "Data correlation based analysis methods for automatic memory forensic" [40]. The authors propose a methodology that correlates different types of memory evidence, including processes, files, users, DLLs, and network connections, by analyzing key OS data structures [40]. These relationships are represented as correlation graphs centered around processes, providing a high level semantic view of the evidence [40].

The authors evaluate their approach through two simulated scenarios: a spy activity involving password file exfiltration via email, and a malware installation case. In both cases, correlation graphs enable investigators to reconstruct criminal scenarios by connecting otherwise isolated evidence. The main limitation identified is the dependency on specific OS data structures, which may change across Windows versions [40].

3.6 Conclusion Remarks

This chapter has presented a comprehensive taxonomy of forensic traces and detection methodologies, from signature based methods through statistical analysis to latent or behavioral approaches based on machine learning.

The analysis shows a fundamental trade off: while ML and DL methods achieve higher detection rates against modern covert channels, their "black-box" nature limits explainability and court admissibility. On the other hand, signature and statistical methods, are more interpretable but fail against unknown or adaptive techniques.

This tension between detectability and explainability motivates the multi source forensic approach presented in the next chapter, which integrates network and host evidences through cross layer correlation. The methodology focuses on interpretable detection techniques that can establish causal relationships between evidence sources, enabling both effective detection of stealthy covert channels and the construction of forensically robust, court admissible cases. The chapter Proof of Concept (4) presents this approach through a forensic pipeline for SHP covert channel detection over ARP (4.3.4) and DNS (4.4.2) and an overview on cache memory based channels (4.7).

Chapter 4

Proof of Concept

4.1 Presentation of studied Covert Channels

This Chapter examines the channels used as the basis for implementing the forensic analysis: the Silent History Channel (SHP) and the Write Back based Channel. These channels are based, respectively, on the work of Weissenborn and Wendzel in the paper "History covert channels with relative timing pointers: Design, error resilience, and detectability", and on the work of Cui, Yang, Cheng in the papers "Abusing Cache Line Dirty States to Leak Information in Commercial Processors".

In this chapter, we present how the original channels operate, the fundamental features they exploit to function as covert channels, and a generalized framework that builds a scheme based on their behavior while abstracting all parameters to extract the underlying structure of the channel. We also describe the implementation of the channel, which is carried out only for the SHP scenario, and the forensic analysis, based on the multi source analysis presented in Section Information Fusion: Multi Source Evidence and Cross Layer Correlation (3.5).

4.1.1 Silent History Channel over ARP and DNS

The Silent History Protocol (SHP) is a History Covert Channel (HCC), that hides information in the Inter Signal Delays between POIs and signals the relevant delays to the receiver through ARP pointer messages that identify the corresponding POIs [41, p. 3 *3.2.1. Basic algorithm*].

Both the sender and the receiver passively observe the same traffic flow, mainly consisting of broadcast and multicast packets [41, p. 4, *3.2.2 Packets of interest (POI)*], such as ARP messages, DHCP requests, and similar network packets. These selected Packets of Interest (POIs) serve as shared references for the communication.

The sender continuously monitors the Packets of Interest (POIs) and computes Inter Signal Delay (ISD) i.e. the time interval between two consecutive POIs [41, p.5, *3.2.3. Network-timing input sources*]. The ISD value is then processed through a hash function, in order to obtain a pseudorandom bit sequence. A portion of this hash output is compared with the next chunk of the secret message [41, p. 3 *3.2.1. Basic algorithm*].

When the bits match the message chunk, the sender sends a small, crafted ARP packet called an ARP pointer signal. This packet does not directly contain the secret data; instead, it contains the information that allows the receiver to identify the specific POI associated with the transmitted symbol [41, p. 7, *4. Implementation*]. For example, the pointer may indicate how many packets were observed between the reference POI and the target POI [41, p. 5, *3.2.3. Network-timing input sources*].

The receiver, observing the same background traffic, recognizes the ARP pointer as a signal belonging to the covert channel, because it is sent to a dedicated IP address [41, p. 4, 16 *4. Implementation*, *Available Code of the Experiment*]. Using the pointer information, the receiver identifies the corresponding POI and reconstructs the same ISD, applies the same hash function, and recovers the associated secret message chunk [41, p. 3, *3.2.1. Basic algorithm*].

The key innovation of SHP is its synchronization mechanism. In real networks, delays and timing variations make absolute time unreliable for coordination. SHP, instead, relies on relative synchronization based on the order of the observed events and on the coordinates embedded in the pointer. Therefore, the receiver does not need to know the exact absolute time at which the sender detected the POI ; it only needs to identify the same event within the shared network traffic. The whole secret message is reconstructed progressively by combining the chunks obtained from the different POIs identified during the transmission.

4.1.2 Write Back Based Covert Channel

A write back based channel exploits the difference in latency between the replacement of a clean cache line and the replacement of a modified (dirty) cache line [13, p. 1, *Abstract*]. A cache line is defined as dirty when its content has been modified inside the cache, but the modification has not yet been saved in main memory. In contrast, a cache line is defined as clean when the data stored in the cache is the same as the data stored in RAM [13, p. 3, *II.B Background and terminology*].

The transmission starts with an initialization phase, during which the receiver accesses the cache lines mapped to the target set and verifies that there are no

dirty lines. In this way, the system starts from a known state, allowing the receiver to more easily identify the modifications later introduced by the sender [13, p. 4, *IV. WB Timing-Based Channels*].

During the encoding phase, message bits are represented through two different operations on the cache lines: to transmit a bit 1, the sender modifies one or more cache lines, setting the corresponding dirty bit to 1; to transmit a bit 0, no operation is performed and the cache state is left unchanged. When a cache line is modified, the new value is temporarily kept in the cache and is not immediately written to RAM, so the line becomes dirty [13, p. 4, *IV. WB Timing-Based Channels*].

When the sender modifies a cache line, the new value is temporarily kept in the cache and is not immediately written to RAM. The cache line therefore becomes dirty [13, p. 4, *IV. WB Timing-Based Channels*].

During the decoding phase, the receiver measures the access latency of a replacement set. This is a set of addresses used to force the replacement of the cache lines that were previously loaded [13, p. 4, *IV. WB Timing-Based Channels*]. When the cache is full and a new cache line must be loaded, one of the existing lines is evicted. The eviction time depends on the state of the line: if the line is clean, it only needs to be removed from the cache, which introduces a baseline delay; if the line is dirty, its modified data must first be saved to main memory, introducing an additional delay of approximately 10 clock cycles due to the write back operation to RAM [13, p. 10, *VI. Stability of WB Channels*].

From the measured latency, the receiver deduces the transmitted bit: normal latency indicates that a clean cache line has been replaced and is decoded as bit 0; higher latency, approximately 10 clock cycles longer, indicates that a dirty cache line has been replaced and is decoded as bit 1 [13, p. 5, 7 *WB Timing-Based Channels*, *V. Evaluation*].

In this way, the channel transmits information without directly modifying network traffic or using shared memory to transfer the message. The sender encodes the bits through the state of the cache lines, while the receiver reconstructs them by observing the latency difference produced by the replacement of clean and dirty cache lines.

In order to properly analyze and implement channels based on these concepts, a generalized framework for these two channels is presented in the next section.

4.2 Generalized Framework

4.2.1 Silent History Protocol Channel Framework

From an architectural perspective, an SHP based covert channel can be represented by four main components: the Shared Entropy Source, the Feature Extraction Function, the Signaling Channel, and the Reference Mechanism.

Component 1: Shared Entropy Source

The first component is the shared entropy source. This is the information observed by both the sender and the receiver. It can be any physical or computational process that produces a sequence of unpredictable bits, symbols, or events that both parties can observe without communicating directly with each other. The main properties of this component are:

- **Shared access:** both the sender and the receiver must be able to observe the same source;
- **Unpredictability:** the source should have high entropy. Otherwise, a third party could predict the next bits, symbols, or packets that the receiver will observe;
- **Density:** the data sequence should be sufficiently frequent or dense to allow the sender to find a match within a reasonable amount of time. If the source is too sparse, the transmission rate of the channel decreases.

Any source that satisfies these properties can be used as a shared entropy source.

Component 2: Feature Extraction Function

The second component is the function that maps the data from the shared source into values that can be compared with the message to be transmitted. In the original SHP channel, this function is represented by a hash function applied to the source data, considering the last Nbits of the hash output. The main properties of this function are:

- **Determinism:** the same input symbol or data must always produce the same output;
- **Robustness to noise:** the function should tolerate noise and small variations in the observed data, whenever possible.

Component 3: Signaling Channel

The third component is the channel used by the sender to communicate the coordinates of the matching symbol or data to the receiver. These coordinates allow the receiver to retrieve the corresponding element from the shared entropy source. The main properties of the signaling channel are:

- **Reachability:** the signal must reliably reach the receiver;
- **Timeliness:** the signal should arrive as quickly as possible, before the shared source produces additional packets or events that could make the pointer inaccurate;
- **Steganographic appearance:** the signal should be difficult to distinguish from legitimate background traffic.

Component 4: Reference Mechanism

The fourth component is the mechanism that allows the receiver, after receiving the pointer, to identify exactly which data item, packet, or symbol from the shared source must be examined. In SHP , this mechanism is based on **ISD** or **ISPN**. The main property of this component is **uniqueness**. The data identified by the pointer must be the same for both the sender and the receiver. In other words, both parties must be able to converge on the same packet or symbol.

SHP Parametrization

An SHP based covert channel can therefore be represented using five parameters:

1. **Shared Entropy Source (SEC);**
2. **Feature Extraction Function (FE);**
3. **Signaling Channel (CS);**
4. **Temporal Reference Function (FTR);**
5. **Number of bits transmitted per signal**, which represents the channel capacity (N).

4.2.2 Write Back Channel Framework

The main difference between SHP and the hardware based covert channel described below is that SHP does not modify the shared entropy source: the sender passively waits until a useful symbol or data value is produced, for example, a value whose

hash matches the message to be transmitted. In contrast, in memory based channels, the sender actively modifies the shared medium (the cache line) in order to transmit bits.

A similiar framework to the SHP one can be derived to describe the cache based covert channel.

Component 1: Shared Hardware Resources

The first component is the shared hardware resource. This is the hardware structure physically shared by the sender and the receiver. The sender interacts with this resource to generate a latency variation that the receiver measures later. The resource must have two main properties:

- **Physical sharing:** the resource must be accessible to both the sender and the receiver without requiring direct communication between them;
- **Amplifiability:** the resource must provide a mechanism that allows the sender to make the latency difference between two conditions sufficiently large to distinguish the covert signal from normal system noise.

Component 2: Modulation Function

The second component is the modulation function. It defines how the sender translates each message bit into an operation on the shared hardware resource. This function must have the following properties:

- **Distinguishability:** the action used to transmit bit 1 must produce a measurable latency that differs from the latency associated with bit 0. This property is closely related to the amplifiability of the shared resource;
- **Controllability:** the sender must be able to activate and deactivate the signal accurately;
- **Reset capability:** after transmitting a bit, the shared resource must return to a neutral state so that the next bit can be transmitted correctly.

Component 3: Measurement Function

The third component is the measurement function. It describes how the receiver measures the observed latency and converts it into a received bit. This function requires:

- **A decision threshold:** a threshold T is used to separate the two possible states. If the measured latency is below T, the receiver decodes bit 0; if it exceeds T, the receiver decodes bit 1;

- **Robustness to noise:** the measurement must remain reliable in the presence of normal system variations;
- **Sufficient separation:** the two latency distributions must be sufficiently separated and have limited overlap.

Component 4: Synchronization Protocol

The fourth component is the synchronization protocol. This is the mechanism that coordinates the sender and the receiver in time. For each transmitted bit, the receiver must be synchronized with the sender in order to measure the latency during the appropriate signal window.

The main required property is **temporal synchronization**. The receiver must perform its measurement while the sender's signal is active. This component is particularly critical in the original channel: the signal window is very short, therefore requires synchronization at the nanosecond scale.

Hardware Channel Parameters

The framework extracted from the two covert channels can therefore be described using four parameters:

1. **Shared Hardware Resource (SHR):** describes the measurable and amplifiable contention affecting the shared resource;
2. **Modulation Function (MF):** produces distinguishable states of the shared hardware resource for bit 0 and bit 1;
3. **Measurement Function (MesF):** uses a threshold T to separate the two latency distributions while minimizing their overlap;
4. **Synchronization Protocol (SP):** ensures that the sender and the receiver operate within the same time window.

4.3 SHP over ARP: Implementation and Forensic Analysis

For the implementation of the covert channels, the source code provided in the additional data of the paper was used. It is available at the following URL: `https://github.com/gXeeXqBHuHDFTaEnff3Z/SHP` and is released under the MIT License, which permits reuse, modification, and distribution provided that the original copyright notice and license text are included. The full license text is available in the LICENSE file of the repository.

The published code serves as the basis for the implementation of the SHP over ARP scenario and was subsequently modified to implement the SHP over DNS case.

The following section presents the generalized framework introduced earlier, applied to the SHP over ARP scenario, including all components that characterize this specific case. Subsequently, all parameters selected for the practical implementation of the channel are described.

4.3.1 Experimental Environment

All experiments were conducted on a single laptop equipped with an Intel Core i5 120U processor (1.40 GHz, x86_64 architecture), 16 GB of RAM, and a 1 TB SSD, Microsoft Windows 11 at 64 bit. Development and execution were performed in Visual Studio Code using PowerShell as the system shell. No virtual machines were employed: the Covert Sender and Covert Receiver ran as separate processes on the same physical host, sharing the same network interface.

The machine was connected to a typical home local area network (LAN) together with other computers and IoT devices. This setting provided natural background traffic composed of web browsing, multimedia streaming, automatic system and application updates, and local broadcast protocols. The overt traffic was not artificially generated but corresponded to the normal activity present on the network during the experimental sessions, allowing the covert channel to be evaluated under realistic, non idle conditions.

Each experimental run lasted on average around 300 seconds, with variations depending on the length of the secret message to be transmitted. The overall experimental campaign spanned approximately two months, during which multiple sessions were executed to evaluate different combinations of channel parameters (BPF filter, input source, bit length, rounding factor, multihashing, ECC, silence_poi,

and silence_cc). Specifically, a total of 10 simulation runs were performed: 5 dedicated to the baseline scenario and 5 to the proposed SHP scenario.

4.3.2 Generic Framework: SHP over ARP scenario

Component 1: Shared Entropy Source The first component is the shared entropy source, represented by the Packets of Interest (POIs) selected from broadcast and multicast network traffic. The system analyzes different layers of the captured traffic:

- **Ethernet frames**, which are required to identify Layer 2 broadcast frames that can be used as POIs;
- **IPv4** and **IPv6 packets**, which are used to filter subnets and source and destination addresses;
- **ARP messages**, which are required to read the fields used by the ARP pointer.

The actual source of entropy is represented by the POIs. These are broadcast or multicast packets filtered directly by the kernel through the **broadcast_ BPF** filter. BPF stands for Berkeley Packet Filter and is a filtering mechanism that operates at kernel level, before packets are delivered to Scapy in user space. This approach is more efficient than applying the filter inside the application. Packets that do not match the BPF filter are discarded directly by the kernel, without creating additional processing overhead in Python. As a result, packet capture is faster and the workload at the application level, where the Python script runs, is reduced.

The Python script receives packets that have already been filtered as broadcast or multicast traffic. However, these packets are still raw bytes and must be interpreted. For this reason, the script applies a second filter based on a protocol whitelist. This whitelist limits the analysis to Ethernet frames, IPv4 or IPv6 packets, and ARP messages.

This second filtering step is necessary because a broadcast packet may contain several protocol layers, for example: Ethernet → IP → UDP → mDNS.

Without the whitelist, Scapy would attempt to dissect protocols such as mDNS and DHCP. These protocols are not relevant to the script and are not used in the analysis.

The two filters therefore perform different functions:

- the BPF filter answers the question: Should this packet be delivered to the Scapy process or discarded?
- the protocol whitelist answers the question: For a packet that has already been received, which protocol layers should be analyzed?

In other words, BPF prevents unnecessary packets from reaching the application, while the whitelist prevents relevant packets from being analyzed beyond the layers required by the protocol.

Component 2: Mapping Function The second component is the mapping function. It determines whether a packet can produce a match with the message that the Covert Sender (CS) wants to transmit. This function is controlled by two main parameters:

- **–inputsource:** in this implementation, the input source is the Inter Signal Delay (ISD). The ISD is the time interval between two consecutive POIs and it is extracted from the numerical timestamp values associated with the packets;
- **–deskew:** it applies a SHA-3 hash function to the ISD. The purpose is to obtain a more uniform bit distribution and to make the resulting bit values appear pseudorandom.

The mapping process is applied by the isMatch() function when the system checks the POIs. The function compares the output of the mapping process with the next chunk of the secret message.

Component 3: Pointer Signaling Channel The third component is the pointer signaling channel, which is implemented through ARP messages.

The ARP message is used as a pointer by the Covert Sender (CS), which transmits it, and by the Covert Receiver (CR), which reads it. The CS uses crafted ARP messages to send the coordinates of the relevant packet to the CS. ARP pointers act as an explicit signaling channel.
The ARP message contains several fields:

- **Multihash count:** indicates whether multiple hashing is applied to the ISD and, if so, how many hash operations are performed;
- **Checksum and flag:** contains a 6 bit checksum of the transmitted chunk and a message type marker;
- **Special strings:** such as **STATIC_BITSTRING_INIT**, which indicates the beginning of the session; **STATIC_BITSTRING_STOP**, which indicates the end of the session; and **STATIC_BITSTRING_RETRY**, which

requests a retransmission. The CR identifies the ARP messages sent by the CS through the sender's IP address. This address is defined in advance as **STATIC_IP_CC**

Component 4: POI Reference Mechanism The fourth component is the mechanism used by the CR to identify the correct POI. In SHP, this mechanism is based on the Inter Signal Delay. The CS and the CR observe the same network traffic. When the CS identifies a match, it calculates the ISD as the difference between the timestamp of the current packet and the timestamp of the previous matching packet. It then applies the SHA-3 hash function to this value and sends the specially created ARP packet.

When the CR receives the ARP pointer message, it first verifies that the message comes from the expected IP address. It then uses the information contained in the message to reconstruct the same ISD used by the CS. To do this, the CR identifies the relevant packet, compares its timestamp with the timestamp of the previous matching packet, calculates the ISD, and applies the same hash function. By following the same procedure as the CS, the CR obtains the same hash output and therefore reconstructs the same message chunk.

The key point is that the ARP message does not directly contain the secret data. Instead, it only communicates information similar to: "a match has occurred at this point." The actual message chunk is reconstructed by the CS from the same broadcast packet already observed in the network traffic, using the same processing pipeline applied by the CS.

4.3.3 Implementation

The demo is organized into three main components:

- **DYSTel:** an implementation of the DYST (Did You See That) covert channel, which preceded SHP;
- **SHPsimulator:** an offline simulator of the SHP covert channel that takes PCAPNG captures as input;
- **SHPlive**: a live implementation of the SHP covert channel.

Available Parameters According to the experimental demo, several parameters can be selected to configure the channel. The following configuration was chosen for the present implementation, together with the rationale for each choice.

- **path_secret secret_message_short.txt** specifies the secret message that the Covert Sender (CS) wants to transmit to the Covert Receiver (CR). A short message was used to make the experimental runs faster and easier to repeat.

- **poi broadcast_ BPF** selects the filter used to capture packets, either directly in the kernel (**broadcast_ BPF**) or at the application level (**broadcast_domain**). The points of interest (POIs) are filtered in the kernel using **broadcast_ BPF**. With this configuration, the fitness value is 0.8483 and the watchdog rate is 1.0. When **broadcast_domain** is used instead, the fitness value decreases to 0.38 and the watchdog rate to 0.50. The decrease occurs because, with **broadcast_domain**, filtering is performed in user space, so more irrelevant events are received and interfere with the matching process. In contrast, **broadcast_ BPF** filters packets earlier, directly in the kernel, resulting in a cleaner source of events.

- **inputsource ISD** selects the input source used to find a match with the secret message, with available options including Inter Signal Delay (ISD), Inter Signal Packet Number (ISPN), and Inter Packet Delay (IPD). ISD was chosen. The fitness value is 0.8483, the throughput is 0.8483 bits per second, and the watchdog rate is 1.0. With ISPN, the fitness value is 0.5183, the throughput is 0.7878 bits per second, and the watchdog rate is 0.658. ISPN reduces performance compared with ISD because packet counting is more discrete and less stable than measuring the time difference between packets. In broadcast traffic, timing based separation of symbols is more effective.

- **bitlength 8** defines the size of the message chunk that must match the selected input source, with possible values of 8, 4, 3, or 2 bits. In the best runs, the configuration with 8 bits achieved a fitness value of 0.8483 and a watchdog rate of 1.0. This is the most practical chunk size because it transmits one byte for each match. A bit length that is too small reduces the amount of useful information transmitted in each match, whereas a bit length that is too large makes it more difficult to find reliable matches. In this setup, 8 bits provide the best balance.

- **rounding_factor 0** allows some input source values to be approximated in order to create a more permissive matching window, with possible values of 0 or 2. The fitness value is 0.8483 and the watchdog rate is 1.0. With a rounding factor of 2, the fitness value decreases to approximately 0.2997 and the watchdog rate to 0.592. A rounding factor greater than 0 reduces performance because it groups different values into the same interval, creating collisions and therefore mismatches. With a rounding factor of 0, the temporal granularity is preserved.

- **multihashing 7** specifies whether multiple hash operations are applied to the selected input source and how many operations are performed, with possible values of 7, 2, or 0. The fitness value is 0.8483 and the watchdog rate is 1.0. With multihashing set to 2 or 0, the channel performance almost drops to zero. A lower multihashing value reduces performance because it provides fewer attempts to find a match. In noisy conditions or when traffic is limited, the system often remains without valid matches.

- **ecc none** specifies the error correction mode. Setting the Error Correction Code (ECC) to none is the configuration that keeps the channel operational in this setup. With **hamming+**, an ECC mode based on Hamming coding that uses redundancy to correct transmission errors, both the fitness value and the watchdog rate are 0.0. The **ecc none** configuration performs better in this setup because error correction introduces additional overhead that does not compensate for the errors produced by the channel.

- **silence_poi 2** and **silence_cc 2** define the waiting interval between consecutive POIs and control channel events, respectively. The value 2 was selected for **silence_poi** because it filters bursts that are too close together without excessively limiting the channel. The same value was selected for **silence_cc 2** for the same reason, but in this case it is applied to the control channel. Very high values for **silence_poi 2** or **silence_cc 2** reduce the number of available events and decrease the throughput. Very low values allow bursts that are too close together to pass through and increase the probability of collisions.

Several runs were performed using different parameter combinations. The purpose was to identify the configuration that provided the best performance under the observed network conditions.
The best performing configuration was: **broadcast_ BPF:ISD:8b:0r:7m**

This configuration was selected by considering several performance indicators, including the mismatch rate, the watchdog rate, the fitness value, the total transmission time, and the accuracy of message reconstruction by the CR.

Performance Metrics The main metrics used to evaluate the channel were the following:

- **Mismatch:** indicates an event in which the expected message or code does not match the value observed or reconstructed by the receiver. In practice, it represents an alignment or decoding error: the system attempts to read a symbol but obtains a different value;

- **Watchdog rate:** measures how often the control mechanism detects a valid event or a correct match. A high value indicates that the channel is stable and that the signals arrive reliably. A lower value may indicate synchronization problems or a high number of unusable events;
- **Fitness:** is an overall score that summarizes the performance of the channel. It mainly rewards useful throughput and decoding stability.

The fitness function is defined as:

$$\text{fitness} = \begin{cases} 0, & \text{if CAF} \leq 1, \\ \text{bps} \times \dfrac{N_{\text{ECC}}}{N_{\text{PR}}}, & \text{otherwise.} \end{cases}$$

where:

- *bps* is the number of bits transmitted per second and represents the channel throughput;
- N_{ECC} is the number of correct ECC matches, namely the number of times that the encoding process produced a valid match;
- N_{PR} is the total number of pointer packets received, representing the number of available matching opportunities;
- CAF is the amplification factor, which indicates whether the channel amplifies the message sufficiently. If $CAF \leq 1$, the configuration is rejected and its fitness is set to zero.

The final configuration was selected because it provided a low mismatch rate, a high watchdog rate, a high fitness value, a reasonable transmission time, and accurate message reconstruction by the CS.

4.3.4 Multiple Source Forensic Pipeline

The forensic analysis uses a multi source approach. It combines network level data with host level artefacts to investigate the possible presence of the covert channel. The analysis is performed at two complementary levels: a macro level, based on complete runs, and a micro level, based on individual time windows.

Macro level analysis: run level statistics The macro level analysis considers each complete execution session as one experimental run. Multiple baseline runs are compared with SHP runs using nonparametric statistical tests and false discovery rate correction. The purpose is to identify which overall metrics can reliably distinguish covert channel activity from normal variations in network traffic. This level of analysis helps reduce the effect of random or isolated anomalies and provides a more robust comparison between the baseline and covert channel scenarios.

Micro level analysis: window by window investigation The micro level analysis focuses on individual 10 second time windows within each run, in order to verify whether the differences observed at the run level are also visible at specific points in time. In particular, this analysis investigates whether network anomalies, such as increased ARP traffic, occur at the same time as suspicious host activity, such as the execution of a channel like process. This makes it possible to distinguish meaningful cross layer events from isolated variations that may only appear relevant when visually inspecting the data.

Operational Implementation of the Forensic Analysis

The forensic analysis component was implemented within the **SHPlive** directory of the covert channel demonstration. The dedicated **forensic_analysis** directory contains two components, **macro_analysis** and **micro_analysis**, which implement the two levels described above.

The **macro_analysis** directory contains ten experiment specific subdirectories, five associated with baseline captures and five associated with SHP captures. Each run directory is organised into three subdirectories: **arp**, **host**, and **correlation**. The arp and host scripts are shared between the macro and micro levels, since the same network traffic and host artefacts must be examined regardless of the temporal granularity adopted. Only the scripts contained about the correlation part differ between the two levels, implementing respectively the run level aggregation used in the macro analysis and the window level partitioning used in the micro analysis.

The **macro_statistics** directory contains the statistical comparison results derived from the macro level analysis, including the Mann–Whitney U test results and the Benjamini–Hochberg correction outputs, used to compare baseline and SHP runs while accounting for the multiple hypothesis testing problem.

The following sections describe the scripts composing the ARP analysis, host evidence, and correlation layers, with particular emphasis on the different implementations of the correlation procedure at the macro and micro levels.

Network Level Monitoring: ARP Level The ARP analysis is based on the idea that an ARP based covert channel leaves a different temporal signature from baseline traffic: greater continuity, longer bursts, fewer empty windows, and greater persistence over time.

Script: arp_analyzer.py The script first extracts all ARP packets from the PCAP file using scapy.PcapReader. It stores some basic information, such as the timestamp, the ARP operation type, the source and destination addresses, and whether the packet is broadcast.

Then packets are grouped into fixed duration time windows, for example, 10 second windows, according to the configuration used in the experiment. This makes it possible to observe how the traffic changes over time.

For each window, the script calculates simple metrics, such as the number of ARP packets, the number of requests and replies, how often broadcast packets appear, and how regular the intervals between packets are. These metrics are used to determine whether the traffic is fragmented, periodic, or concentrated into blocks.

The purpose is to describe the temporal structure of the ARP traffic and to determine if the scenario with the covert channel active shows a more organized or anomalous behavior than the baseline scenario.

The script converts a raw packet capture into tables that can be used to compare the ARP level with the other analysis levels and to support the forensic interpretation.

Host Level Monitoring: HOST Level The second level is based on the idea that a covert channel does not leave only network traces, but also host level traces, such as processes, loaded DLLs, execution arguments, and suspicious combinations of activities.

Script: host_session_monitor.py The initial idea behind this host level analysis script was to create a forensic monitor that periodically observes the system and saves useful traces to CSV files. These traces can help determine whether there is a process related to packet sniffing, packet capture, or the possible generation of suspicious ARP traffic.

The script repeatedly scans the system at regular intervals (2s) and collects two main types of information:

- running processes;
- modules loaded into memory.

It then attempts to identify which processes are more relevant from a forensic point of view. For example, it looks for Python processes, processes that load libraries commonly used for packet capture, such as Npcap and Wireshark, and processes whose arguments or keywords are compatible with ARP transmission or with a covert channel scenario. The purpose of the script is to collect and correlate different types of local evidence for each process. This makes it possible to verify whether suspicious processes are associated with packet sniffing modules loaded into memory, unusual network connections, or other artifacts compatible with packet capture or traffic generation activities.

Correlation layer This level is used to verify whether host evidence and ARP metrics increase together in the same time windows or in the same run. In other words, it checks if there is a multi level consistency between the process activity and the network behaviour. Since it differs slightly from both the macro and micro analyses, it is best presented in two distinct sections.

ARP baseline thresholds In order to evaluate the values extracted from the PCAPNG captures, baseline thresholds were created for selected ARP metrics. These thresholds describe the normal ARP behaviour observed in the experimental environment when the SHP covert channel was not active. The values measured in a new run can then be compared with the baseline thresholds to identify possible deviations from normal network behaviour.

For most metrics, the threshold was calculated using the 95th percentile of the baseline distribution. The 95th percentile represents the value below which approximately 95% of the baseline observations fall. Therefore, values above this threshold belong to the upper tail of the baseline distribution and can be considered relatively uncommon with respect to the observed normal behaviour.

The following metrics use an upper tail threshold, where higher values indicate a possible deviation from the baseline profile:

$$\textbf{arp_count} > 7.00$$
$$\textbf{arp_active_window_pct} > 73.1$$
$$\textbf{active_run_mean} > 4.30$$
$$\textbf{active_run_max} > 12.8$$
$$\textbf{arp_count_mean} > 2.75$$
$$\textbf{arp_broadcast_ratio_mean} > 0.559$$

The **arp_count** threshold is applied at the window level. A 10 second window is flagged when it contains more than seven ARP packets. The remaining thresholds are applied at the run level because they summarise the ARP behaviour of the entire capture rather than a single time window.

Binary entropy threshold Binary entropy is interpreted in the opposite direction. In the SHP scenario, lower entropy is associated with a more persistent and less balanced sequence of ARP active and ARP inactive windows. For this reason, the threshold is not based on the 95th percentile but on the 5th percentile of the baseline distribution:

$$\textbf{binary_entropy} < 0.838$$

A run with binary entropy lower than 0.838 is therefore considered more persistent and less variable than almost all baseline runs. This does not prove that SHP is present, because other legitimate network conditions may also produce persistent ARP activity. It only indicates a deviation from the baseline behaviour measured in the experimental environment.

Reliability of the thresholds The most reliable threshold is the **arp_count** window level threshold. It was calculated using 159 individual 10 second windows collected from five baseline runs. Because it is based on a larger number of observations, it provides a more stable estimate of the upper range of normal ARP packet volume.The other thresholds are less robust because they are calculated from run level summary values.
Each baseline run contributes only one value for each metric; therefore, these thresholds are based on only five observations. They remain useful as descriptive reference values, but they should be interpreted cautiously because a small number of baseline runs may not fully represent the natural variability of the network environment.

Macro Analysis

Building on this definition, the present section details how the macro level analysis was implemented and interpreted for the baseline and SHP scenarios. It first describes the cross layer correlation methodology and the script used to aggregate window level evidence into run level indicators. It then reports the global results table, followed by the comparison of baseline variability, the consistency observed across SHP runs, and the joint host network correlation. Finally, it presents the nonparametric statistical tests, together with the Benjamini–Hochberg correction, used to determine which run level metrics reliably distinguish the SHP covert channel from normal baseline traffic.

Cross layer correlation methodology: script arp_host_correlation.py
Cross layer evidence is identified at the temporal window level. A window is classified as correlated when, within the same time interval, both an anomaly in ARP traffic with respect to the baseline threshold and the presence of at least one process classified as channel like are observed. Specifically, the network anomaly is defined as the arp_count metric exceeding this baseline threshold. When both conditions occur within the same window, the two layers intersect and the cross_layer_event flag is activated.

For each experimental run, the events detected across windows are aggregated into summary indicators. Each run produces a set of aggregate indicators, including the total number of analysed windows, the number of windows with an ARP anomaly, the count of windows with channel like activity, the total cross layer events, and their occurrence rate. These run level measures serve as independent observations to compare the baseline scenario with the covert channel scenario through the Mann–Whitney U test. When testing multiple metrics, p-values are adjusted using the Benjamini–Hochberg procedure to control the false discovery rate.

Run	ARP active (%)	Run mean	Run max	ARP mean	Bcast ratio	Binary entropy	Channel-like (%)	Cross-layer rate (%)
baseline1	57.58	2.38	9	1.91	0.315	0.983	0.00	0.00
baseline2	74.19	4.60	13	2.84	0.604	0.824	0.00	0.00
baseline3	38.71	1.33	3	0.65	0.129	0.963	0.00	0.00
baseline4	40.62	1.62	4	0.84	0.217	0.974	0.00	0.00
baseline5	68.75	3.14	12	2.41	0.383	0.896	0.00	0.00
shp1	84.62	5.50	21	2.23	0.560	0.619	87.18	0.00
shp2	75.61	4.43	9	2.22	0.571	0.801	87.80	0.00
shp3	83.33	4.29	9	2.53	0.532	0.650	77.78	2.78
shp4	87.80	9.00	19	1.95	0.545	0.535	82.93	0.00
shp5	86.36	6.33	8	4.55	0.720	0.575	68.18	13.64

Table 4.1: Selected ARP, host, and cross layer metrics for baseline and SHP runs.

Baseline vs. Baseline variability The baseline analysis was performed using five runs in which the SHP covert channel was not active. The results showed that normal ARP traffic in the experimental network can vary considerably. This is important because a high ARP packet count, or an unusual value for one metric, cannot alone be considered proof of covert channel activity.

At the window level, the 95th percentile of the ARP packet count distribution was 7 packets per window. Therefore, a window was considered anomalous when it contained at least 8 ARP packets, according to the rule **arp_count** > 7. However, some threshold exceedances were also found in normal baseline traffic: 3.03% of the windows in **baseline1**, 9.68% in **baseline2**, and 9.38% in **baseline5** exceeded this threshold. This shows that an isolated window with a high number of ARP packets can occur during normal network activity. For this reason, the **arp_count** threshold should be used as an alert that requires further investigation, not as direct proof of a covert channel.

The baseline runs also showed different levels of ARP activity at the run level. Four runs, namely **baseline1**, **baseline3**, **baseline4**, and **baseline5**, did not exceed any of the six run level thresholds. In contrast, **baseline2** exceeded all six thresholds. It had the highest percentage of ARP active windows (74.19%), the highest mean active run length (4.60), the longest active run (13 windows), the highest mean ARP count per window (2.84), and the highest mean ARP broadcast ratio (0.604). It also had the lowest binary entropy value (0.824).

These results do not mean that **baseline2** contained covert channel activity.

Instead, they show that **baseline2** was an extreme but legitimate example of ARP traffic in the available baseline data. This confirms that the thresholds can identify values that differ from the usual baseline pattern, but they cannot independently prove the presence of SHP or another covert channel.

Baseline vs. SHP correlation The comparison between baseline and SHP runs shows that the SHP scenario produced consistent host level and network level evidence at the same time. Four network metrics exceeded their baseline reference thresholds in every SHP run: **arp_active_window_pct** was always greater than 73.10%, **active_run_mean** was always above 4.31, **arp_broadcast_ratio_mean** was always higher than 0.560, and **binary_entropy** was always lower than 0.838. Together, these results indicate that ARP traffic in SHP runs was present in more time windows, remained active for longer consecutive periods, included a greater proportion of broadcast packets, and followed a more regular and persistent pattern than in the baseline runs.

Host level evidence was equally present. A process was labelled as channel like only when it was identified as a Python process whose command line contained at least three distinct parameter families associated with the experimental SHP implementation. The inspected parameter families included **--poi**, input source options such as **--inputsource** or **--input-source**, bit length options such as **--bitlength** or **--bit-length**, rounding factor options, **--multihashing**, **--ecc**, **--silencepoi**, **--silencecc**, and **--subchanneling**. The script also inspected packet related terms such as **--interface**, **--pcap**, **--capture**, **--promisc**, **--raw**, **--broadcast**, and **--arp** as additional contextual information. A channel like process was found in every SHP run, appearing in 88.24% to 96.55% of the host monitor samples, while no baseline run ever produced such a process. This shows that the process implementing the experimental channel was visible for most of the duration of each SHP execution; however, because the rule is based on parameter families known from the implemented channel, it should be interpreted as implementation specific forensic evidence rather than as a general detector for all covert channels.

In contrast, high volume ARP indicators were less consistent across SHP runs. Windows with **arp_count** > 7, namely at least 8 ARP packets in a 10 second window, were observed only in **shp1**, **shp3**, and **shp5**. The strict cross layer rule, requiring a channel like process and an individual ARP burst with **arp_count** > 7 in exactly the same 10 second window, detected events only in **shp3** and **shp5**. This does not contradict the broader correlation between host and network evidence. Instead, it shows that the SHP implementation did not consistently generate high volume ARP bursts while the process was observed: in several runs, the network pattern was distributed over time through persistent ARP activity rather than

concentrated into isolated high volume windows. Consequently, strict burst level co occurrence should be treated as a high specificity supporting indicator, whereas the session level association between channel like host evidence and persistent ARP activity provides the more consistent cross layer result in the evaluated experiment.

Overall, SHP was distinguished from baseline traffic by persistent ARP activity, longer average active sequences, a higher broadcast ratio, lower binary entropy, and the repeated presence of channel like host processes. High ARP volume peaks and strict host network co occurrence occurred only in a subset of runs, so they should be treated as secondary supporting evidence rather than primary indicators of the covert channel.

Statistical tests used The previous comparison shows that the baseline and SHP runs differ in several network level and host level metrics. However, an observed difference between the two groups is not automatically statistically meaningful. When several metrics are examined at the same time, some differences may appear important only because of natural variability or random effects in the collected data. This is particularly relevant in the present experiment, where the number of independent runs is limited and ARP activity can vary across baseline sessions. Therefore, statistical tests were used to determine whether the observed differences between baseline and SHP runs are supported by the available data, while reducing the risk of interpreting random or spurious differences as meaningful results.

To determine whether the observed differences between baseline and SHP runs were statistically supported, the analysis used the Mann–Whitney U test together with the Benjamini–Hochberg correction. The Mann–Whitney U test was applied separately to each metric to compare the baseline and SHP groups. The Benjamini–Hochberg correction was then used to adjust the resulting p–values for multiple comparisons.

Mann–Whitney U test The Mann–Whitney U test is a non parametric test used to compare two independent groups. It is useful when the sample size is small and when the data cannot be assumed to follow a normal distribution [42]. In this study, the two groups are the five baseline runs and the five SHP runs. The test is applied separately to each selected metric, including the ARP active window percentage, mean active run length, binary entropy, and channel like host sample percentage.

The test combines the values from the baseline and SHP groups and ranks them from the lowest to the highest value. It then evaluates whether the values in one group tend to be consistently higher or lower than those in the other group. The null hypothesis, denoted as H_0, states that there is no systematic difference

between the baseline and SHP runs for the analysed metric. In other words, under H_0, the two groups are assumed to come from the same distribution [42].

A small p-value indicates that the observed difference between the two groups would be unlikely if the null hypothesis were true. Therefore, a small p-value provides evidence against H_0. The Mann–Whitney U test was selected because it is a rank based alternative to the independent samples t-test and does not require normally distributed data. This is appropriate for the present experiment because each group contains only five runs and some metrics are bounded, discrete, skewed, or include repeated values [42].

Benjamini–Hochberg correction The Benjamini–Hochberg (BH) procedure is applied after the Mann–Whitney U tests. It is not an additional test used to compare baseline and SHP runs. Instead, it uses the p–values produced by all Mann–Whitney U tests in the same analysis [43].

The BH procedure addresses the multiple comparisons problem. When many metrics are tested using the same significance threshold, such as 0.05, some small p–values may appear by chance even if there is no real difference between the baseline and SHP scenarios. The BH method reduces this risk by correcting the set of p–values obtained from the different tests [43].

More specifically, the procedure controls the false discovery rate, which is the expected proportion of false positive results among the metrics considered statistically significant. The method orders the p–values from the smallest to the largest and adjusts them according to the total number of tests performed [43].

In this work, the BH correction was applied to the $p$values obtained from the predefined network level, host level, and cross layer metrics. The corrected value is stored in the **p_value_bh** column. A metric was considered statistically supported after correction when:

$$p_{\mathrm{BH}} < 0.05$$

The **significant_bh_0_05** column is set to **True** when this condition is met and to **False** otherwise. This prevents single uncorrected p–values from being interpreted as strong evidence when they may be caused by testing several related metrics at the same time.

Script: run_level_statistics.py The **run_level_statistics.py** script implements the run level statistical analysis in a reproducible way. It reads a single input

file, **all_runs_summary.csv**, which contains one row for each baseline or SHP run. Each row includes the run identifier, the scenario label, and the aggregated metrics obtained from the ARP, host, and cross layer analysis. The input file includes network level metrics, such as **arp_active_window_pct**, **active_run_mean**, **active_run_max**, **arp_count_mean**, **arp_broadcast_ratio_mean**, and **binary_entropy**; the host level metric **channel_like_sample_pct**; and the cross layer metric **cross_layer_event_rate**.

For each available metric, the script:

1. Separates the observations into the baseline and SHP groups.

2. Performs a two sided Mann–Whitney U test to evaluate whether the metric differs between the two groups.

3. Applies the Benjamini–Hochberg correction to all valid p–values produced during the same execution.

4. Saves the results in two CSV output files.

The output file contains the Mann–Whitney U statistic, the original p–value, the Benjamini–Hochberg adjusted p–value, and the final significance flag for each tested metric.

Interpretation of statistically supported results These results provide evidence from two different levels of analysis.

Metric	**Mann–Whitney** U	p **value**	**BH adjusted** p–value	**Significant after BH**
arp_active_window_pct	0.0	0.00794	0.0211	True
active_run_mean	2.0	0.0317	0.0634	False
active_run_max	8.0	0.397	0.397	False
arp_count_mean	7.0	0.309	0.353	False
arp_broadcast_ratio_mean	4.0	0.0952	0.152	False
binary_entropy	25	0.00794	0.0211	True
channel_like_sample_pct	0.0	0.00729	0.0211	True
cross_layer_event_rate	7.5	0.179	0.239	False

Table 4.2: Run level Mann–Whitney U test results comparing baseline and SHP scenarios ($n = 5$ for both groups), with Benjamini–Hochberg adjusted p–values.

Network level findings The **arp_active_window_pct** metric measures the percentage of 10 second windows that contain at least one ARP packet. The baseline runs had a median value of 57.58%, while the SHP runs had a median value of 84.62%. All SHP values were higher than all baseline values. The Mann–Whitney U test showed complete separation between the two groups, with $U = 0.0$, and the result remained statistically supported after Benjamini–Hochberg correction, with $p_{\mathrm{BH}} = 0.021$. This indicates that ARP traffic was present during a larger proportion of the observation period in the SHP sessions.

The **binary_entropy** metric describes the variability of the sequence of ARP active and ARP inactive windows. Baseline runs had a median entropy of 0.963, while SHP runs had a lower median of 0.619. All SHP values were lower than all baseline values, with $U = 25.0$ and $p_{\mathrm{BH}} = 0.021$. Lower binary entropy is consistent with a less variable and more persistent ARP activity pattern, meaning that ARP traffic remained active across more consecutive windows during the SHP sessions. However, this result should be interpreted as being consistent with greater persistence, rather than as proof that the observed pattern can only be caused by SHP.

Host level findings The **channel_like_sample_pct** metric measures the percentage of host samples containing at least one process classified as channel like. A channel like process was defined as a Python process with command line arguments matching at least three parameter families associated with the experimental covert channel configuration.

All baseline runs had a value of 0%, while the SHP runs ranged from 88.24% to 96.55%, with a median of 91.89%. The difference showed a separation between the two groups, with $U = 0.0$, and remained statistically supported after Benjamini–Hochberg correction, with $p_{\mathrm{BH}} = 0.021$. This result shows that the host level rule identified the command line configuration associated with the experimental channel in the SHP sessions, without incorrectly classifying normal monitoring tools in the baseline sessions.

This finding should nevertheless be interpreted carefully. The indicator was designed using parameter families associated with the experimental SHP configuration. It therefore validates the controlled host level evidence in this experiment, but it does not demonstrate a universal detection method for all covert channels or all attacker implementations.

Metrics without statistical support The remaining metrics did not show statistically supported differences after the Benjamini–Hochberg correction. These active_run_mean, active_run_max, arp_count_mean, arp_broadcast_ratio_mean, and cross_layer_event_rate.

The **active_run_mean** metric showed a higher median in SHP runs than in baseline runs and a large observed effect, but its corrected p–value was above 0.05. It can therefore be described as an exploratory trend, but not as a statistically supported result. Metrics based mainly on ARP volume, including the proportion of windows with **arp_count** > 7, did not reliably distinguish the two scenarios. This suggests that SHP does not necessarily generate very high ARP bursts; instead, it appears to produce ARP activity that is more persistent over time.

The strict **cross_layer_event_rate** also did not distinguish the groups reliably. This metric required both a channel like host process and an ARP window with more than seven packets. Because many SHP windows contained channel like processes but did not exceed the ARP volume threshold, the cross layer event rate was zero in three of the five SHP runs.

Concluding remarks Under the tested experimental conditions, the SHP runs were distinguished from the baseline runs by two network level features, more persistent ARP activity and lower binary entropy, and by the repeated presence of channel like host processes. These three metrics remained statistically supported after correction for multiple comparisons.

In contrast, measures based only on high ARP packet volume and the strict burst based cross layer rule were not sufficiently reliable. Overall, the results provide preliminary evidence that persistent ARP activity, combined with host level command line artefacts, can support forensic differentiation between baseline traffic and the SHP covert channel scenario.

Micro Analysis

This section takes a closer look at the results through a micro level forensic analysis. The previous analysis showed that many individual metrics can vary because of normal network activity. However, three metrics showed statistically supported differences between baseline and SHP runs: **arp_active_window_pct**, **binary_entropy**, and **channel_like_sample_pct**. The first metric measures how often ARP traffic is present. The second metric measures how regular the ARP activity pattern is. The third metric identifies processes with command line arguments similar to those used by the experimental covert channel.

The host level metric also has a limitation. It depends on command line information. An attacker could hide or change the command line arguments, use a different process name, or compile the channel into a binary program. In these cases, the **channel_like_sample_pct** metric may not detect the process.

This section therefore examines whether the correlation between host and network evidence are directly related in the same 10 second windows.

Script: arp_host_correlation_micro.py For the micro level analysis, a dedicated correlation script, **host_arp_correlation_micro.py**, was developed. The script compares network and host evidence window by window.

For each experimental run, the script automatically uses UNIX timestamps to align host artefacts with the 10 second time windows created during the ARP network analysis. The host artefacts include information such as running processes, active network and loaded memory modules.

Instead of searching only for fixed process names, it examines process command line arguments, as in the previous analysis: it correlates these host side indicators with network side events, including timing anomalies and ARP traffic bursts, specifically when the window **arp_count** exceeds the threshold of 7 packets. A time window is considered suspicious when host and network evidence occur in the same interval.

Finally, it calculates Pearson and Spearman correlation coefficients, together with their corresponding p–values, for all paired host and network metrics. These statistical measures help determine whether the observed correlations within individual windows are likely to be meaningful or whether they may simply be caused by normal background activity or random variation.

Analysis of the SHP scenarios

- **Scenario shp1.** The host layer detects active processes with covert channel parameters across several intervals (including Process IDs (PIDs) 48556 and 34528 loading **wpcap.dll** and **packet.dll**). However, the ARP traffic volume remains below or equal to the threshold of 7 packets during these intervals. A single ARP burst exceeding the threshold occurs outside the execution window of the channel like processes (window 37, with 8 ARP packets). Consequently, no simultaneous cross layer events are recorded for this run.
- **Scenario shp2.** Although processes associated with the covert channel and packet capture drivers are identified on the host (such as PIDs 44280 and

48112), the ARP packet volume remains consistently within baseline values across the entire session, never exceeding 7 packets in any 10 second interval. As a result, no strict cross layer correlation is triggered.

- **Scenario shp3.** A strict cross layer event is identified in window 5. In this interval, the host layer records active channel like processes, loading packet capture libraries (**wpcap.dll** and **packet.dll**, specifically PIDs 12716 and 46680), coinciding exactly with an anomalous burst of 9 ARP packets (**arp_count** > 7). Across the other windows where channel processes remain active, ARP traffic stays within the baseline range, limiting the strict co occurrence to this specific interval.

- **Scenario shp4.** Similar to **shp2**, channel like processes with structural parameters and capture libraries (PIDs 23548 and 42992) are detected by the host monitor. However, the network layer records an ARP count consistently below the threshold of 7 packets throughout the session, resulting in zero strict cross layer windows.

- **Scenario shp5.** The strongest cross layer correlation is observed in this scenario, where anomalous ARP bursts coincide with channel like host activity across three consecutive windows: window 17 (9 ARP packets), window 18 (12 ARP packets), and window 19 (15 ARP packets). In all three windows, the host monitor concurrently records active processes configured with covert channel parameters and loaded packet capture drivers (PIDs 37964 and 46732), meeting both criteria for a strict cross layer event.

Analysis of baseline scenarios In the five baseline sessions, where the covert channel is not active, no strict cross layer events are recorded: the number of 10 second windows showing simultaneous channel like host processes and anomalous ARP volume remains zero.

At the host level, none of the observed processes is defined as a candidate covert evidence, and no legitimate tool loads packet capture libraries. Although isolated windows exceed the ARP baseline threshold in some control sessions due to ordinary background network activity, the concurrent absence of suspicious host processes prevents any false positive. These observations confirm that requiring evidence from both layers ensures high specificity in the absence of the covert channel.

Scenario	N	Pearson r	Pearson p	Spearman ρ	Spearman p	Sig.
baseline1	33	+0.0833	0.6448	-0.0475	0.7930	No
baseline2	31	+0.0785	0.6748	+0.0170	0.9276	No
baseline3	31	-0.0751	0.6882	-0.0751	0.6879	No
baseline4	32	+0.1757	0.3362	+0.1618	0.3762	No
baseline5	32	+0.0914	0.6189	+0.2205	0.2253	No
shp1	39	+0.0116	0.9442	+0.1455	0.3769	No
shp2	41	+0.0394	0.8069	+0.0128	0.9366	No
shp3	36	+0.1704	0.3205	+0.1439	0.4025	No
shp4	41	+0.1503	0.3483	+0.1460	0.3623	No
shp5	22	+0.1173	0.6031	+0.1322	0.5575	No

Table 4.3: Pearson and Spearman correlation results between **channel_like_proc_count** and **arp_count** .

Correlation between host activity and ARP volume Correlation tests between host process counts and ARP packet volume (**arp_count**) show weak results. In all sessions, both Pearson and Spearman coefficients stay close to zero (between -0.07 and $+0.22$), and the p-values remain above 0.05. This confirms that host and network metrics do not share a simple linear or rank based dependency within a single window. Therefore, individual metrics alone cannot reliably identify the channel, confirming the importance of combining multiple indicators.

Scenario	N	Observed (r)	Permutation (p)	Significant ($\alpha = 0.05$)
baseline1	33	N/A*	N/A*	No
baseline2	31	N/A*	N/A*	No
baseline3	31	N/A*	N/A*	No
baseline4	32	N/A*	N/A*	No
baseline5	32	N/A*	N/A*	No
shp1	39	+0.0116	0.9439	No
shp2	41	+0.0394	0.8097	No
shp3	36	+0.1704	0.3176	No
shp4	41	+0.1503	0.3360	No
shp5	22	+0.1173	0.5441	No

Table 4.4: Permutation correlation results. N/A* indicates that no correlation could be computed for the baseline scenarios due to zero variance in the host metric.

Permutation test results: channel like process count and ARP volume Non parametric validation using a permutation test (10,000 resamples) on the

relationship between the suspicious process count (**channel_like_proc_count**) and ARP packet volume (**arp_count**) shows no statistically significant correlation in the SHP attack scenarios ($p > 0.05$). Specifically, the observed correlation coefficients remain weak and close to zero, ranging from $r = 0.0116$ ($p = 0.9439$ in **shp1**) to $r = 0.1704$ ($p = 0.3176$ in **shp3**).

In the baseline sessions, the complete lack of variability in the host metric, since channel like processes never run, makes it impossible to compute a correlation, so no value is produced. These results confirm that the bivariate linear relationship between this single host behavioural metric and raw ARP traffic volume is weak and insufficient to characterize the attack at the level of a single time window, reinforcing the need for more complex multi evidence classification models.

Limits of the micro level analysis and Concluding remarks The first limitation concerns the monitoring environment. The covert channel and the forensic tools used similar packet capture technologies. Therefore, packet capture libraries such as **wpcap.dll** and **Packet.dll** may be loaded by the channel, by the monitoring tools, or by both. Their presence alone cannot prove that the covert channel was active. Removing the monitoring tool footprint may also be risky: if these libraries are removed from the analysis, real evidence produced by the covert channel could also be removed. This would be especially problematic in a blind forensic investigation, where the analyst does not know in advance which processes or libraries are legitimate.

The second limitation concerns the weak statistical correlation between host and network indicators, as discussed above. The available data do not show, at window level, a strong or reliable relationship between ARP packet count and the channel like process indicator in individual time windows.

Taken together, these limitations show that an individual window can appear convincing but still be unreliable as independent evidence. A window with several channel like processes and a local ARP traffic increase may be useful during forensic triage, but it should not be treated as proof of a covert channel on its own. The SHP channel does not produce large ARP bursts in every active window; instead, the stronger evidence emerges across the full run, where ARP traffic is active in more windows, the ARP pattern is more persistent, binary entropy is lower, and channel like processes are repeatedly present at the host level. For this reason, point in time correlations and high ARP volume peaks should be treated as supporting evidence only. A stronger forensic conclusion should rely on the complete context: repeated observations across several runs, persistent ARP activity, host level process evidence, and statistical comparison with baseline traffic.

4.4 SHP over DNS: Implementation and Forensic Analysis

4.4.1 Generic Framework: SHP over DNS scenario

Following the analysis of SHP over ARP, this chapter focuses on the experimental evaluation of SHP over DNS, where DNS traffic is used as the data plane entropy source while ARP is retained as the signaling channel. The chapter describes the adaptation of the generalized framework, the implementation and configuration of the channel, and the multi source forensic analysis used to compare the baseline scenario with the active covert channel scenario.

Component 1: Shared Entropy Source This component relies on Packets of Interest (POIs) taken from the network. Unlike the previous scenario that used broadcast messages, here the channel takes its POIs from regular DNS traffic.

In the DNS configuration, the BPF filter has the same role as before: it prevents unnecessary packets from reaching the Scapy process and allows only traffic relevant to the capture to pass through. When **dns_only** is enabled, the filter is configured to include ARP traffic, UDP traffic on port 53, and TCP traffic on port 53.

Therefore, DNS traffic becomes the source observed for the data component, while ARP remains available for signaling. At the application level, the script applies a whitelist so that packets are analyzed only up to the protocol layers required by the channel.

Component 2: Mapping Function The second component is the mapping function, based on hashing and deskewing. This function remains the same as the one described for SHP over ARP because the basic logic of the channel does not change in the DNS configuration.

The CS observes the POIs, extracts an input value, and compares it with the message chunk that must be transmitted.

Component 3: Pointer Communication and Signaling Channel The third component is the signaling mechanism, which relies on ARP packets to send pointers and coordinate the transmission.

In this configuration, ARP frames are used to transmit the pointer, while DNS traffic continues to carry the informational content of the channel. Since pointers

could not be transmitted reliably through DNS, the protocol uses a separate signaling channel based on dedicated ARP messages.

This separate channel helps maintain synchronization between the sender and the receiver and supports the reconstruction of the message.

ARP signaling therefore provides the operational references required by the protocol, such as start, stop, block checksum, retry, and resynchronization. These signals define the behavior of the channel and allow the receiver to interpret the received chunks correctly. Their operation is described in greater detail in the section dedicated to the channel implementation.

Component 4: POI Identification Mechanism The fourth element allows the CR to recognize the right packet of interest. Here, both endpoints rely on the Inter Signal Delay (ISD), measuring the time gap between packets to verify the match.

This component remains unchanged compared with SHP over ARP because the reference mechanism depends on the sender and the receiver observing the same traffic flow and applying the same timing rule.

In the DNS configuration, the CS calculates the Inter Signal Delay from the observed traffic, derives the corresponding hash value, and sends the related ARP pointer. The CR observes the same traffic and verifies that the ARP message comes from the dedicated IP address. It then reconstructs the same ISD and obtains the message chunk.

Implementation

Control Plane and Data Plane In the previous scenario, SHP over ARP, the covert channel was designed so that the shared entropy source consisted of ARP packets. In this scenario, the general structure of the channel remains the same, but the source used to match the secret message changes.

The sender observes DNS packets to obtain the chunks of the secret message, particularly the timing between packets, namely the Inter Signal Delay (ISD). This modification keeps the same basic logic, but moves the observation point to a different protocol. This protocol still belongs to the family of broadcast related protocols, but it is not ARP.

The channel configuration also remains the same as in the previous scenario. The available parameters are unchanged:

- the secret message is read from a file using –**path_secret**;
- the packet filter can be selected between **broadcast_bpf** and **broadcast_domain**;
- the input source can be **ISD**, **ISPN**, or **IPD**;
- the chunk size is defined by –**bitlength**;
- –**rounding_factor** makes the matching process more permissive;
- –**multihashing** allows multiple matching attempts to be performed on the same source;
- –**ecc** can be disabled or used to introduce error correction;
- –**silence_poi** and –**silence_cc** define the minimum interval between consecutive events.

The best configuration tested is: **broadcast_bpf:ISD:3b:0r:2m**
This configuration provided the best balance between channel stability, matching accuracy, and overall transmission speed.

The criteria used to evaluate the configuration also remained the same. **Mismatch** indicates an alignment error between what the sender expects and what the receiver reconstructs. The **watchdog rate** measures how often the channel correctly follows valid events. **Fitness** summarizes the overall quality of the channel. It is equal to zero if **CAF** does not exceed 1; otherwise, it increases with useful throughput and matching accuracy.

The main modification concerns the separation between the data channel and the control channel. The data channel uses DNS traffic, while the signaling channel uses ARP. This choice is based on the fact that DNS traffic, when used as a source of entropy, is more variable and more exposed to noise. ARP, on the other hand, provides a simpler and more stable protocol for exchanging explicit control signals.

For this reason, ARP is not used to carry the secret message. Instead, it is used to manage synchronization, confirmation, and error recovery operations. Separating these two levels makes the channel more robust and, at the same time, easier to analyze from a forensic point of view.

ARP signaling therefore performs a central function. The sender and the receiver exchange different types of packets:

- **START** packets initiate the channel;

- **STOP** packets terminate the channel;
- **CHECKSUM** packets verify the correctness of the received chunks;
- **RETRY** packets are sent when the verification fails;
- **RESYNC** packets realign the local states after an error or a problematic block.

In the code, this mechanism is explicitly implemented. ARP packets carry control signals encoded in two binary fields, which are interpreted by the receiver as operational instructions.

More specifically, the sender reads the secret message from a file, converts it into bits, and divides it into fixed size chunks. Each chunk is compared with the selected input source, namely the ISD calculated from the DNS traffic.

When the matching process is successful, the sender considers that packet a valid transmission point and updates its internal state. After a certain number of chunks, the data is grouped into blocks of eight elements. Only at this stage is a block checksum generated and transmitted to the receiver through ARP.

This two level verification process, based on both chunks and blocks, is important because it reduces the probability that a single synchronization error will compromise the reconstruction of the entire message.

The receiver follows a similar but symmetric process. When it captures a valid data packet, it uses parsing functions to extract the corresponding bit string and applies any required transformations, such as hashing, deskewing, or subchanneling.

The result is compared with the expected part of the message. If the comparison is correct, the chunk is accepted and added to the current block.

When the block is complete, the receiver waits for the second checksum transmitted by the sender. If the two checksums match, the block is committed to the reconstructed message. If they do not match, the block is discarded and a **RETRY** message is sent.

This logic is necessary in timing channels, and particularly in this scenario, because the transmission is based on the time intervals between Packets of Interest. If the receiver incorrectly identifies even one packet, it loses synchronization with the sender and the error can propagate through the rest of the message reconstruction.

As a result, a single synchronization error may cause the receiver to interpret subsequent packets incorrectly and compromise the rest of the decoding process.

For this reason, the signaling channel is also the most interesting component from a forensic point of view. ARP control packets produce a much clearer trace than the DNS packets used by the data plane because they contain explicit and repeated events, such as channel initialization, verification, retransmission, resynchronization, and termination.

In summary, the covert channel consists of two separate but connected levels:

- the data plane uses DNS traffic as the entropy source and as the basis for matching the chunks of the secret message;
- the control plane uses ARP to synchronize, verify, and correct the transmission.

This design keeps the channel logic unchanged compared with the previous scenario, while making the distinction between information transfer and communication state management clearer. For this reason, the ARP signaling channel becomes the main source of forensic evidence.

Available Parameters In this scenario, the parameters were selected not only according to channel performance, but also with the aim of keeping the channel as stable as possible, verifying the correctness of the data reconstructed by the receiver, and achieving a reasonable balance between transmission time and overall channel efficiency.

- **–poi broadcast_bpf**: the relevant traffic is filtered at kernel level using a BPF filter. In this way, the channel operates on a cleaner packet set with less noise;
- **–inputsource ISD**: the input source is the Inter Signal Delay, the time interval between packets;
- **–bitlength 3**: the message is divided into 3 bit chunks. This choice is related to the design of the channel because transmission remains reliable with this value, whereas increasing the bit length too much reduces robustness;
- **–rounding_factor 0**: no rounding is applied, so the comparison remains precise;

- –**multihashing 2**: the channel performs only two matching attempts. This makes the process simpler, but also slightly less tolerant of errors;

- –**silence_poi 2** and –**silence_cc 2**: these parameters introduce a short silence window between Packets of Interest and between control messages. This reduces the risk of interpreting events that are too close together as separate signals;

- –**dns_only**: enables the use of DNS traffic alone as the data source, while signaling remains separate and is performed through ARP. This parameter does not change the control channel, but restricts the data plane to DNS packets;

- –**resync_interval 8**: every 8 chunks, the sender forces a resynchronization. In this way, the sender and receiver realign their internal states and limit the propagation of errors;

- –**retry**: enables retransmission when the receiver detects an inconsistent block or chunk. In practice, it allows the receiver to request that the sender transmit the incorrect part again;

- –**block_size 8**: the channel groups the chunks into blocks of 8. Since each chunk contains 3 bits, each block contains 24 bits and is verified at the end of its transmission.

Experimental Environment

In the DNS scenario, the experimental environment followed the same general setup described above, with one additional measure to ensure sufficient DNS activity.

In a typical home LAN, the natural rate of DNS queries generated by user devices and background services was not always high enough to allow the covert channel to transmit the secret message within a reasonable time. To address this limitation, a DNS query generator script was employed to increase the volume of overt DNS traffic during the experiments.

This approach emulates a more densely populated network or a location close to a heavily used DNS resolver, where the frequency of DNS requests is naturally higher. The generator only produced additional legitimate DNS queries; the covert information continued to be encoded in the timing and pattern of these queries, without modifying their content or structure.

4.4.2 Multi Source Forensic Pipeline

As the previous scenario about the SHP over ARP covert channel, the analysis is structured into two main part: the macro and micro analysis, in order to analyse the behaviour of this channel through arp, host evidence and their correlation, in this scenario that present a difference in the entropy source. The main objective is to determine if the integration of these multisource can be also significant at statistical level.

Operational Implementation of the Forensic Analysis Inside the **SHPlive** directory of the covert channel demonstration, a **DNS** folder was created to host the code for the SHP over DNS channel. Within this directory, the **forensic_analysis** folder maintains the same layout adopted in the previous forensic analysis. Specifically, it is organized into **macro** and **micro** subfolders, each containing the ten captured scenarios (five baseline and five SHP). In each scenario, three dedicated subdirectories, **arp**, **host**, and **correlation**, store the corresponding analysis results.

Network Level Monitoring: ARP and DNS Level At this level, the objective is not only to count ARP packets, but also to determine whether the traffic shows a temporal structure compatible with a signaling scenario.

The analysis is also extended to DNS traffic, not as the main channel, but as contextual traffic that can help interpret and compare the behaviour of ARP traffic in the baseline scenario and in the scenario with the covert channel active.

Script network_analyzer.py The **network_analyzer.py** script analyzes a network capture by separating the ARP and DNS components. It segments the packets into fixed time windows and calculates a set of metrics for each window to assess the regularity, continuity, and structural anomalies of the traffic. In addition to the ARP evaluation, the script generates a CSV file containing the computed DNS metrics aligned across the same time intervals.

Host Level Monitoring: host_session_monitor_dns.py script The script implements the host level forensic monitor. It adopts the same methodology as the host monitor used in the previous SHP over ARP analysis: the host state is sampled periodically to log relevant information regarding active processes, loaded memory modules, and network connections into CSV files.

The primary difference lies in the tracking of DNS related activities. In addition to identifying Python processes and packet sniffing libraries, this version captures process artifacts and monitoring signals associated with DNS traffic generation and manipulation.

ARP and DNS baseline thresholds To detect potential anomalies in the DNS covert channel scenario, baseline thresholds were established for selected ARP and DNS metrics using the same 95th percentile criterion defined previously. Values exceeding these upper tail thresholds indicate deviations from the normal, inactive baseline profile.

$$\textbf{arp_count} > 11.05$$
$$\textbf{arp_active_window_pct} > 94.63$$
$$\textbf{active_run_mean} > 25.80$$
$$\textbf{active_run_max} > 27.00$$
$$\textbf{arp_count_mean} > 7.94$$
$$\textbf{arp_broadcast_ratio_mean} > 0.87$$
$$\textbf{dns_count} > 170.05$$
$$\textbf{dns_active_window_pct} > 100.00$$
$$\textbf{dns_count_mean} > 116.76$$
$$\textbf{dns_unique_qnames_mean} > 1.00$$

The **arp_count** and **dns_count** thresholds are applied at the window level, flagging 10 second intervals when packet volumes exceed these limits. The remaining thresholds are applied at the run level because they summarise the network behaviour across the entire capture rather than a single time window.

Binary entropy thresholds As the previous case, binary entropy is interpreted in the opposite direction. Because the covert channel produces constant traffic rather than alternating pauses, entropy values decrease. To detect this decrease, the detection boundary is set at the 5th percentile of the baseline distribution:

$$\textbf{binary_entropy} < 0.17$$
$$\textbf{dns_binary_entropy} < 0.00$$

Macro Analysis

Cross layer correlation script: arp_host_correlation.py As in the previous scenario, this script implements the combined host network analysis used to detect traces of the ARP covert channel. It processes system process logs and network traffic data, both structured into 10 second time windows. At the host level, it identifies Python instances whose command line arguments contain parameter families associated with the covert channel, such as **--poi** and **--bitlength**, labelling them as channel like when they meet a minimum configuration threshold. At the network level, it compares ARP traffic volume against baseline derived normality

thresholds. The algorithm then performs a windowed temporal alignment, raising a positive cross layer event flag only when a suspicious process and a network volume anomaly are detected simultaneously within the same time window.

Run	ARP active (%)	Run mean	Run max	ARP mean	B-cast ratio	Binary entropy	Channel-like (%)	Cross-layer rate (%)
baseline1	100.00	31.00	31	8.94	0.939	0.000	0.00	0.00
baseline2	73.17	5.00	10	3.95	0.605	0.839	0.00	0.00
baseline3	56.67	2.83	11	2.87	0.506	0.987	0.00	0.00
baseline4	50.00	1.67	4	2.07	0.352	1.000	0.00	0.00
baseline5	57.14	2.29	6	1.68	0.319	0.985	0.00	0.00
shp1	93.55	14.50	27	7.29	0.809	0.345	41.94	3.23
shp2	96.88	15.50	29	6.38	0.849	0.201	50.00	3.13
shp3	90.91	30.00	30	6.97	0.782	0.439	39.39	0.00
shp4	94.74	18.00	35	7.16	0.901	0.297	55.26	10.5
shp5	85.71	12.00	23	5.96	0.787	0.592	50.00	3.57

Table 4.5: Selected ARP, host, and cross-layer metrics for baseline and ARP runs.

Baseline vs. baseline Five baseline sessions were used to describe ARP behavior when ARP was absent. The results show that normal traffic is not uniform across sessions. ARP activity can increase substantially even in the absence of covert communication, so a large packet count or an unusual value in one metric cannot be interpreted as sufficient evidence by itself.

The window level threshold was obtained from the baseline distribution. The condition arp_count > 11.05 identifies windows containing at least 12 packets; this event was observed in two normal sessions: it occurred in 16.13% of the windows of baseline1 and in 7.32% of those of baseline2. These values show that short ARP bursts are part of legitimate network behavior. For this reason, an arp_count exceedance is treated as an indicator that deserves further analysis, rather than as evidence that ARP is active.

A broader view is provided by the run level results. baseline2, baseline3, baseline4, and baseline5 did not exceed any of the six run level thresholds. In contrast, baseline1 exceeded all six thresholds. It contained active ARP traffic in every window, with an active window percentage of 100%. Its mean active run length was 31.0 windows, and its longest active sequence also covered 31 windows. The same session had an average of 8.94 ARP packets per window, a mean broadcast ratio of 0.939, and a binary entropy of 0.00.

The profile of baseline1 is unusual compared with the other baseline sessions, but it does not represent malicious behavior. Instead, it shows the range of variation that can occur in the experimental network under normal conditions. This result supports the use of several indicators together, since a threshold exceedance at one network layer alone cannot confirm the presence of an ARP covert channel.

DNS carrier traffic To ensure the ARP covert channel had enough carrier traffic volume to transmit data within a reasonable time, a DNS traffic generator script was kept running in the background during the experiments. This addressed the normal, and otherwise insufficient, activity level of the LAN.

This architectural setup is clearly visible in the data extracted from the analysed baselines. DNS traffic volume is consistently high, averaging between 109 and 117 packets per 10 second window across all five reference scenarios, with a peak average of 117.03 in **baseline4**.

In addition, the request to reply ratio remains very stable, ranging between 0.82 and 0.90 depending on the session. Together, these elements describe a highly predictable DNS reference environment, marked by high volume and very low variability, with entropy equal to 0.

Baseline vs. ARP scenario variability Data from the baseline and ARP runs shows that the covert channel generated host level evidence and network ARP spikes: this happened even when the aggregate session metrics do not exceed the normal boundaries.

At the network level, volume indicators are particularly reactive: windows with **arp_count** > 11.05 appeared consistently in every single ARP run, accounting for 12.50% to 22.58% of all observation windows. This indicates that the channel implementation introduced packet bursts concentrated in time. A closer look at these anomalies shows that some exceeded the baseline threshold by a large margin. The highest value was observed in scenario **shp3**, with 21 ARP packets in a single time window. This value was almost 10 packets above the reference threshold and approximately 90% higher than the limit. All scenarios produced bursts of considerable intensity, in particular 19 packets in **shp4**, 18 in **shp5**, 17 in **shp1**, and 14 in **shp2**, occurring between 4 and 7 times within a single session. In the windows that exceeded the threshold, the average number of transmitted packets stayed stable between 12.7 and 17.6. These figures confirm that the malicious traffic does not produce small fluctuations from random noise, but instead injects concentrated bursts that nearly double the expected physiological volume for that

time window.

On the other hand, run level metrics showed only partial threshold violations. The active window percentage, defined as **arp_active_window_pct** $> 94.63\%$, was reached only in scenarios **shp2** and **shp4**, while the maximum active run length, defined as **active_run_max** > 27.00, triggered alerts in **shp2**, **shp3**, and **shp4**. Extreme limits, such as binary entropy below 0.17 or a mean ARP traffic value above 7.94 packets, defined as **arp_count_mean**, were never crossed. This confirms that the channel distributes its activity without significantly altering the overall average network volume.

Host level results also showed a clear difference between the two scenarios. Channel like evidence was detected in every ARP run and was present in 72.22% to 86.95% of the host monitoring samples. By contrast, no channel like processes were observed in the baseline runs. This suggests that the process associated with the experimental covert channel remained visible on the host during most of the ARP executions.

Cross layer results The cross layer correlation rule was applied to both baseline and ARP scenarios. This rule identifies an event when two conditions occur together in the same 10 second window: a host process classified as channel like, defined as a Python instance matching at least three specific parameter families, and a network anomaly, defined as ARP traffic above 11.05 packets.

Across the five baseline sessions, the combined analysis produces no false alarms. Some control sessions show isolated network threshold exceedances, with 5 anomalous windows in **baseline1** and 3 in **baseline2**, but the absence of the suspicious host process, with **channel_like_windows** $= 0$, keeps the cross layer event count at zero across all runs.

In the ARP scenarios, the correlation identifies positive events in every run, although they covered only a small part of each session. The channel like host process appears in 13 to 21 windows, while anomalous ARP bursts occur 4 to 7 times per session. The overlap of these two indicators within the same time window produces a limited number of cross layer events: 4 occurrences in **shp1** and **shp3**, 3 in **shp4** and **shp5**, and 1 in **shp2**.

These results indicate that the discontinuous timing of network bursts and the variability of the host process mean cross layer correlation does not cover the entire attack. Instead, it identifies specific transmission intervals where host and network evidence coincide.

Building on the previous analysis, this section examines whether arp_count and the channel_like flag hold up as statistically valid metrics. As in the earlier scenario, this is assessed using the Mann-Whitney U test with Benjamini– Hochberg correction for multiple comparisons.

Inferential statistical analysis script (run_level_statistics.py) As in the previous scenario, baseline and ARP sessions were compared at the aggregate level using descriptive statistics and the Mann–Whitney U test. Since multiple metrics were evaluated, Benjamini–Hochberg correction was applied to the resulting p-values. The output files report the corrected values (BH-adjusted p-value) and the corresponding significance indicators.

Metric	**Mann–Whitney** U	p **value**	**BH adjusted** p value	**Significant after BH**
arp_active_window_pct	5.0	0.1508	0.1508	False
active_run_mean	5.0	0.1508	0.1508	False
active_run_max	4.0	0.0952	0.1508	False
arp_count_mean	5.0	0.1508	0.1508	False
arp_broadcast_ratio_mean	12.5	—	—	False
binary_entropy	12.5	—	—	False
cross_layer_event_rate	0.0	0.0075	0.0375	True

Table 4.6: Run level Mann–Whitney U test results for the ARP and network volume metrics ($n = 5$ for both Baseline and ARP groups).

Interpretation of statistically supported results Statistical analysis using the Mann–Whitney U test with Benjamini–Hochberg correction was used to assess the significance of differences between baseline and ARP scenarios. Among all session level metrics examined, only one indicator reached full statistical validation after correction for multiple comparisons: the cross layer event rate, denoted **cross_layer_event_rate**, with $p_{\mathrm{BH}} = 0.0375$, $p = 0.0075$, and $U = 0.0$. This metric showed a clear separation between groups, rising from a median of 0% in control sessions to about 10.71% in attack runs. These findings show that no cross layer events were observed in the baseline dataset under the selected host and network criteria. However, the metric reflects specific host level assumptions that may be less decisive in realistic, blind execution settings where prior knowledge of the process is unavailable.

In contrast, metrics based just on network volume or ARP activity duration, such as **arp_active_window_pct**, **active_run_mean**, **active_run_max**, and **arp_count_mean**, showed clear increases in median values during the attack, but did not pass the corrected significance threshold, with $p_{\mathrm{BH}} = 0.151$. While these indicators offer useful exploratory trends on packet persistence, their variability within the baseline and the limited sample size mean they cannot be considered sufficient on their own for formal detection.

Since individual ARP metrics do not reach statistical significance after correction for multiple comparisons, formally validated multi source evidence cannot be established. Although cross layer correlation shows a temporal overlap between network bursts and host process activity, the lack of robust statistical support for the network side components means this overlap cannot be treated as independent, multi level supporting evidence.

Concluding remarks In summary, while ARP traffic volume spikes during ARP transmissions, normal network traffic also varies considerably across baseline runs. Because of this natural variation, none of the network only metrics remained statistically significant after multiple testing correction, meaning that traffic volume anomalies alone cannot confirm an attack.

On the other hand, the cross layer correlation rule produced no false alarms in control runs and proved statistically significant ($p_{\mathrm{BH}} = 0.0375$). However, this approach relies on knowing the specific command line parameters of the host process in advance. For this reason, cross layer correlation should be seen as a useful way to support existing evidence on suspicious time windows, rather than a general detection method for unknown attacks.

Micro Analysis

In this level of analysis, together with the ARP and host results obtained from the baseline and SHP captures, two key scripts were implemented: host_postprocess_scoring.py and correlation_aligner.py. The first script separates background noise, caused by the host monitor and the DNS traffic generator, from anomalies that may indicate SHP channel activity. Below is a short explanation of what each script does, followed by the results.

Host heuristic classification script (host_postprocess_scoring.py) The script monitors logs to separate background activity generated by monitoring and analysis tools from possible traces of the covert channel. This distinction is necessary because legitimate tools used during the experiment may generate host

related activity that could otherwise appear suspicious.

Each process receives an anomaly score based on four types of evidence: Python or other interpreter processes, packet capture libraries such as **wpcap.dll** or **packet.dll**, channel related command line parameters such as –**poi** or –**block_size**, and network connection information, including active sockets or selected ports. Processes showing multiple indicators are labelled as **candidate_covert_evidence**. An allowlist is used to reduce false positives caused by known legitimate experimental tools.

Cross correlation script: correlation_aligner.py As in the previous SHP over ARP scenario, this script performs a window based analysis to correlate ARP traffic events with channel like activity observed on the host. It collects ARP metrics and host activity logs and aligns them into 10 second intervals identified by **window_id**. For each interval, the script records indicators such as the ARP packet count and the presence of host processes labelled as **candidate_covert_evidence**.

The resulting dataset provides a single timeline of cross layer events, making it possible to identify time windows in which unusual ARP activity and suspicious host behaviour occur together. Pearson and Spearman correlation tests are then applied to evaluate whether these variables show linear or monotonic relationships across the analysis windows. The corresponding p-values are also considered to assess whether the observed correlations are statistically significant.

Analysis of the baseline scenarios In the five control sessions (**baseline1** to **baseline5**), the forensic scripts return a consistent result across all runs. No strict cross layer correlation is recorded in any of the baseline sessions: the count of windows showing alignment between an anomalous process and an ARP burst remains at zero. At the host level, the heuristic does not classify any process as **candidate_covert_evidence**; the Python instances associated with the monitoring script and the DNS generator are filtered by the allowlist and classified as **known_instrumentation** or **known_experimental_support**. In terms of memory modules, no Python process associated with the tracked parameters shows the loading of interception libraries such as **wpcap.dll** or **packet.dll**.

These results are consistent with the expected specificity of the detection model: in the absence of an active covert channel, the combination of structural parameters, network sniffing, and ARP bursts does not occur, which limits the generation of false positives even in the presence of legitimate Python scripts running on the host.

Analysis of the SHP scenarios The SHP scenarios show a similar pattern when ARP activity is compared with host level observations. The cross evidence affected windows were then examined in more detail by comparing process information with the corresponding loaded module records.

- In **shp1**, strict cross layer correlations occur in windows 6. Looking at this window, the monitoring processes (PIDs 56616 and 69340) and the DNS generator processes (PIDs 55316 and 62568) do not load packet capture modules. Four processes include parameters related to the covert channel configuration (PIDs 53476, 59564, 67652, and 69592); however, only PIDs 59564 and 67652 are associated with **wpcap.dll** and **packet.dll**.

- In **shp2**, no strict cross layer correlation occurred, as the ARP packet volume remained consistently within baseline values and never exceeded the threshold across the entire session. However, host level monitoring still observed specific events: while the two monitoring processes (PIDs 60860 and 65936) and the two DNS generator processes (PIDs 64804 and 69940) were not associated with interception libraries, among the processes carrying channel related parameters (PIDs 46180, 46216, 56164, and 66196), only PIDs 46180 and 46216 were observed loading Npcap/WinPcap related modules.

- In **shp3**, correlated events are found in windows 10: PIDs 44736 and 65568 belong to the monitoring script, whereas PIDs 62256 and 67196 belong to the DNS generator. None of these processes loads packet capture libraries. Of the four processes with channel related parameters (PIDs 52780, 67072, 68976, and 70120), only PIDs 67072 and 70120 are linked to **wpcap.dll** and **packet.dll**.

- In **shp4**, strict correlations are found in windows 9, 10, 12, and 15. In window 10, the monitoring processes (PIDs 54084 and 65780) and the DNS generator processes (PIDs 62548 and 69484) do not load capture related modules. Among the processes with channel related parameters (PIDs 17484, 55416, 64092, and 64396), the libraries **wpcap.dll** and **packet.dll** are associated only with PIDs 55416 and 64092.

- In **shp5**, the correlated window is 8. In window 8, neither the monitoring processes (PIDs 28532 and 62992) nor the DNS generator processes (PIDs 44484 and 45352) load sniffing libraries. Of the processes containing channel related parameters (PIDs 55008, 60304, 61844, and 67016), only PIDs 55008 and 61844 are associated with the packet capture libraries.

Overall, the observations are consistent across the five scenarios. The monitoring script and DNS generator create background Python activity, but their processes

are not associated with packet capture libraries. In the same windows in which anomalous ARP activity is observed, only some of the processes configured with channel related parameters load **wpcap.dll** and **packet.dll**. This combination of process configuration, loaded modules, and concurrent ARP activity is used as supporting evidence for the host level **candidate_covert_evidence** label in the analysed runs.

Correlation between channel like processes and ARP volume Analysis of the correlation files generated for all scenarios shows that the **channel_like_proc_count** feature, which counts Python processes with structural parameters typical of the covert channel, appears only in the SHP scenarios. In the baseline scenarios, where no covert channel is running, this feature remains at a null or constant value, resulting in zero statistical variance.

Correlation between **channel_like_proc_count** and **arp_count** was tested for each SHP scenario using both Pearson and Spearman coefficients. The results are summarized in Table 4.7.

Scenario	**Pearson** (r)	**Pearson** p**-value**	**Spearman** (ρ)	**Spearman** p**-value**
baseline1	0.1684	0.3650	0.2214	0.2314
baseline2	0.2563	0.1058	0.2848	0.0712
baseline3	0.1358	0.4742	0.0764	0.6883
baseline4	0.0093	0.9611	-0.0683	0.7199
baseline5	-0.3049	0.1147	-0.3278	0.0886
shp1	0.2593	0.1589	0.2602	0.1574
shp2	0.1545	0.3984	0.1533	0.4021
shp3	0.2975	0.0927	0.2519	0.1573
shp4	0.1475	0.3770	0.1781	0.2847
shp5	0.1374	0.4858	0.1376	0.4850

Table 4.7: Pearson and Spearman correlation between channel_like_proc_count and arp_count across the SHP scenarios.

In SHP scenario, the correlation coefficients range from about 0.13 to 0.30 across all scenarios, indicating a weak positive association: as channel like process activity increases, ARP packet volume tends to increase slightly as well, but the linear and rank based relationship is not strong at the level of individual time windows. The associated p–values are above the standard 0.05 threshold in almost every case, with **shp3** as a partial exception ($p \approx 0.092$). This means that, for this pair of metrics, there is no statistically significant relationship at the 95% confidence level. To further assess this relationship, a permutation test was also performed on the same pair of metrics, **channel_like_proc_count** and **arp_count**.

Scenario	N	**Observed** (r)	**Permutation** (p)	**Sig.** ($p < 0.05$)
baseline1	31	0.2962	0.1190	False
baseline2	41	0.3002	0.0784	False
baseline3	30	-0.3735	0.0002	True
baseline4	30	-0.3046	0.0880	False
baseline5	28	-0.3049	0.0794	False
shp1	31	0.0793	0.6609	False
shp2	32	-0.0145	0.9335	False
shp3	33	0.3441	0.0614	False
shp4	38	0.1738	0.2920	False
shp5	28	0.3452	0.0770	False

Table 4.8: Permutation test results for the correlation between **channel_like_proc_count** and **arp_count** across baseline and SHP scenarios.

Permutation test results: final score and ARP volume In order to verify if the timing match between suspicious host activity and network traffic anomalies was purely accidental, the relationship between **channel_like_proc_count** and **arp_count** was evaluated using a permutation test with 10,000 resamples. Across the SHP attack runs, the association was weak and did not reach statistical significance. ($p > 0.05$), with correlation coefficients spanning from $r = -0.0145$ in **shp2** to $r = 0.3452$ in **shp5**, accompanied by high empirical p-values ($p = 0.6609$ and $p = 0.9335$, respectively).

These findings indicate that raw process volume alone cannot reliably reflect covert transmission at the level of individual time windows. This limitation is underlined by the control sessions: in **baseline3**, for instance, an unexpected yet statistically significant negative correlation emerged ($r = -0.3735$, $p = 0.0002$), demonstrating that bivariate linear metrics can easily generate misleading patterns. Because simple linear relationships fail to provide a stable decision boundary, a supervised machine learning framework is essential to capture the complex, multivariate interactions across layers.

Concluding Remarks Window level analysis shows that tracking individual metrics in isolation is not enough to reliably identify covert channel traffic. The host heuristic classification and temporal alignment scripts confirmed that no false positives occur in the control, or baseline, sessions: legitimate Python processes linked to monitoring and DNS traffic generation are correctly excluded, showing no loading of packet capture libraries and no correlation with ARP bursts.

In contrast, the SHP scenarios show a clear temporal overlap between the activation of Python processes with structural parameters specific to the covert channel, such as **--poi**, the loading of suspicious network libraries, such as **wpcap.dll** and **packet.dll**, and anomalous ARP traffic peaks at the network level. However, statistical tests (Pearson, Spearman, and the permutation test) return weak and non significant correlation coefficients ($p > 0.05$).

These results suggest that simple bivariate metrics are insufficient to capture the relationship between host activity and network anomalies. While this indicates a potential need to explore multivariate or supervised machine learning approaches capable of handling complex interactions, the presence of sufficient exploitable signal, as opposed to background noise, remains to be formally verified empirically within this specific experimental scope.

4.5 Comparing SHP over ARP and SHP over DNS: evidence and forensic analysis

Forensic analysis of the two scenarios reveals different behaviours, mainly due to how each covert channel handles data transmission and signaling. The main differences found are described below.

Architecture and entropy source In the SHP over ARP scenario, broadcast and multicast traffic, including ARP messages, serves as both the entropy source and the communication channel for pointers. In SHP over DNS, these roles are separated: DNS traffic acts as the data plane, providing the entropy source, while ARP traffic is used only as the control plane, handling synchronization, checksums, start, stop and retransmissions (RETRY) messages.

Network level evidence In SHP over ARP, the attack shows up as persistent, continuous ARP traffic over time. The analysis found a higher percentage of active time windows, consistently above the baseline value of 73.10%, and lower binary entropy, indicating regular and prolonged activity. In SHP over DNS, the channel produces short but highly concentrated ARP traffic bursts. During SHP over DNS transmissions, peaks of up to 21 ARP packets were recorded in a single 10 second window, almost double the baseline critical threshold, set at the 95th percentile of 11.05 packets.

Host level evidence In both scenarios, host level analysis produced consistent results, identifying anomalous Python processes with specific command line arguments. These malicious processes tended to load network interception libraries, such as **wpcap.dll** and **packet.dll**, into memory.

Statistical validation of the metrics In SHP over ARP, aggregate network metrics, such as the percentage of active windows and binary entropy, passed the statistical tests (Mann–Whitney U test with Benjamini–Hochberg correction), showing that they could distinguish covert traffic from normal traffic. In SHP over DNS, by contrast, individual network volume metrics did not reach statistical significance at the session level, due to the fragmented and discontinuous nature of the bursts.

Significance of cross layer correlation This is the clearest difference between the two analyses. In SHP over ARP, the strict cross layer correlation rule, requiring the suspicious process and the ARP peak to occur in the exact same time window, did not prove statistically reliable, since the traffic was spread out over time rather

than concentrated. In SHP over DNS, on the other hand, the cross layer event rate turned out to be the only statistically robust and validated indicator ($p_{BH} = 0.0375$) in the entire experiment.

Methodological summary and limits A comparison between the two scenarios reveals fundamentally different detection challenges. In the SHP over ARP setup, network data alone provides strong evidence: the channel leaves clear, session wide traces, such as longer active runs and lower entropy, allowing analysts to identify anomalies before inspecting host logs.

In contrast, for SHP over DNS, network evidence alone is insufficient. Standard statistical tests (such as Pearson, Spearman, and permutation tests) show weak, non significant correlations ($p > 0.05$) between host activity and packet volumes, confirming that the relationship is non linear. Detecting this channel therefore requires supervised machine learning to capture complex multi source patterns. However, this shift introduces the challenge of black box models: in forensic investigations, high detection accuracy must be balanced with explainability to ensure that results can be clearly justified and admitted in court.

4.6 Evasion and countermeasures: the attacker's perspective

In cybersecurity, the analysis of a steganographic covert channel is not complete without considering the ongoing *cat and mouse* dynamic between defender and attacker. Understanding how a forensically aware attacker would adapt their behaviour is essential for assessing the real security of the network and for developing more resilient detection metrics. This section examines evasion strategies that could be used to bypass the forensic pipeline developed for the SHP, for both the ARP and DNS implementations.

Network level evasion: randomization and decoy traffic

The network forensic pipeline showed that SHP alters the temporal statistics of traffic. In the ARP implementation specifically, the channel makes traffic far more continuous than normal background behaviour, raising the **active_run_mean** metric to 4.67 compared to 1.83 in the baseline. To evade this metric, an attacker could randomize the duration of transmission bursts. Instead of sending the ARP pointer as soon as a match is found with the entropy source (ISD), the covert sender could intentionally delay transmission, breaking messages into irregular bursts to mimic the natural intermittency and unpredictability of legitimate network traffic.

In the DNS based SHP implementation, network analysis revealed abnormal persistence in ARP traffic, used as the control plane, driving the **zero_window_pct** metric down to 9% compared to 66–75% in the normal baseline. To counter this detection method, an attacker could inject decoy ARP traffic specifically calculated to restore the original statistics, or enforce prolonged artificial silence periods to bring the percentage of empty time windows back to levels indistinguishable from the baseline.

Host level evasion: obfuscation and decoupling

At the host level, the current pipeline detects the malicious process by searching for suspicious command lines, containing parameters that point to the activation of a covert channel. This evidence is reinforced by arguments passed in plain text on the command line (such as **--poi broadcast_bpf** and **--inputsource ISD**) and by the clear link to libraries (DLLs) used for packet sniffing.

If an attacker wanted to evade these metrics, the following countermeasures could be applied:

- **Process masking**: the Python code would be compiled into a standalone binary executable and renamed to resemble a legitimate system process, such as **svchost.exe** or **spoolsv.exe**.

- **Parameter obfuscation**: execution arguments and BPF filters would not be passed in plain text, but encrypted or hard coded directly into the malware.
- **Rootkit and process injection**: packet capture activity would be hidden by injecting the malicious code into an existing system process already authorized to communicate over the network, making the loading of sniffing libraries invisible.

Cross layer evasion: breaking the causal correlation

The forensic pipeline relies on connecting network events (ARP packets) with host activity. To evade detection, an attacker could separate the code that listens for traffic from the code that sends the crafted ARP packets. By running these two tasks in different threads and adding random delays between them, the attacker breaks the timing link between reading the entropy source and transmitting the covert signal. This makes it very hard for an analyst to prove a direct cause and effect relationship.

Overall, this highlights the ongoing arms race between covert channels and forensic detection: as new defenses emerge, attackers develop new evasion techniques, forcing both sides to become increasingly sophisticated.

4.7 Cache Based Covert Channels: A Theoretical Study

As introduced in Chapter 2.4, memory based covert channels transmit secret information by exploiting latency variations in shared microarchitectural structures. In this work, the theoretical study focuses on a cache based timing channel that leverages the replacement latency difference between clean and dirty cache lines in a write back cache architecture.

Due to strict microarchitectural and experimental constraints, this channel was not practically implemented in the experimental campaign. Instead, a generalized four layer forensic pipeline is proposed, based on the analytical models established in the reference literature. This framework defines a structured methodology to collect eviction latencies, evaluate deviations from a clean baseline, identify periodic transmission patterns, and correlate hardware level timing anomalies with host process activity.

Limits to Live Implementation A reliable and reproducible implementation of a cache based covert channel requires an exceptionally controlled execution environment that goes beyond the capabilities of a standard general purpose workstation.

Temporal synchronization The primary challenge is fine grained synchronization. The sender and receiver must operate within precisely aligned time windows. The sender alters the state of target cache lines to encode secret bits, while the receiver must measure the eviction latency within that exact timeframe. Any timing jitter, premature probe, or delayed measurement causes symbol loss or bit errors.

Hardware topology and cache indexing A functional implementation demands exhaustive knowledge of the processor's microarchitecture. Specifically, it requires knowing the exact cache hierarchy (L1, L2, and shared Last Level Cache), cache line sizes, set associativity, slice hashing functions, and replacement policies (such as LRU , pseudo LRU , or adaptive schemes). Both the sender and receiver must access specific virtual addresses that translate directly into the same physical cache set.

Write back dynamics and noise In a write back cache architecture, evicting a dirty cache line requires writing its modified content back to main memory, which introduces an observable delay compared to replacing an unmodified (clean) line.

This mechanism, described in the literature as a *Miss+Miss* channel, is sensitive to concurrent system activity. Achieving reproducible latency differences requires:

- pinning processes to specific CPU cores using strict core affinity and hyper-threading configurations;
- minimizing operating system interruptions, such as context switches, process migrations, and Dynamic Voltage and Frequency Scaling (DVFS);
- using low overhead, cycle accurate timing instructions (such as **RDTSC** or hardware performance counters) capable of measuring differences of only a few CPU cycles.

Given these constraints, attempting an implementation on a multi purpose machine without specialized kernel privileges would produce excessive noise. Developing a formalized forensic framework provides greater scientific rigor and investigative value.

The Four Layer Forensic Framework

To analyze cache based covert timing channels systematically, we organize the forensic investigation into a four layer pipeline. Instead of treating microarchitectural measurements as isolated events, this framework tries to transform raw eviction latencies into structured, legally defensible evidence. The analysis progresses sequentially from low level timing acquisition to host level process correlation.

Layer 1: Temporal Acquisition The investigative pipeline begins with temporal acquisition, which captures eviction latencies and organizes them into time series. In a write back cache channel, the target measurement is the time required by the receiver to access its replacement set and evict target lines. Using hardware performance counters or cycle accurate instructions (such as **RDTSC**), the monitor samples eviction latencies across regular, fixed duration time windows.

The primary goal, at this initial stage, is to build an ordered timeline of events. By running the acquisition under both normal operating conditions and suspected transmission sessions, the analyst creates two aligned datasets: a clean baseline capture and a suspicious trace. These time series serve as the foundation for all subsequent analytical steps.

Layer 2: Local Statistical Analysis Once the time series is established, the second layer evaluates whether the observed latency distributions deviate significantly from normal baseline behavior. The core objective is to identify whether

the suspicious session shows elevated delays, unusual stability, or altered tail distributions.

This evaluation relies on standard descriptive metrics calculated for each window, including the mean, median, and variance. While the mean and median indicate whether overall eviction delays have increased, the variance shows whether the latencies are consistent or highly variable. The analysis also uses the Cumulative Distribution Function (CDF) and high percentiles, such as the 90th and 95th percentiles. If the CDF moves toward higher values compared to the baseline, it means that many cache operations take longer than usual.

Furthermore, this layer evaluates **burstiness** to determine whether timing anomalies occur uniformly or gather into temporal clusters. Because covert channels rarely transmit continuously, they often produce transmission bursts separated by quiet intervals. In other words, dense clusters of delays suggest intentional transmission, whereas ordinary system noise is more evenly distributed.

Layer 3: Spectral and Periodicity Analysis Local statistics show where delays increase, but they cannot prove that these changes form a real message. The third layer solves this problem by checking whether the delays repeat at regular time intervals, separating normal noise from a synchronized covert signal.

To detect regular patterns, the analyst applies the Fast Fourier Transform (FFT) to the normalized time series. The FFT converts the signal into the frequency domain, revealing strong peaks that show whether cache line evictions repeat at a constant rate. In addition, autocorrelation helps identify how often the pattern repeats over time by measuring the similarity of the signal across different time lags.

Comparing these spectral signatures against the baseline is crucial. Normal workstation traffic may exhibit regular patterns due to operating system scheduling or timer interrupts, but it rarely produces the distinct, strong spectral peaks associated with synchronized covert signaling. Identifying regular autocorrelation lags and dominant frequencies provides strong evidence of intentional encoding.

Layer 4: Host Context Correlation The final layer relates low level hardware effects to operating system activity. A hardware timing anomaly alone cannot prove malicious intent; the analyst must confirm that the microarchitectural variation comes from specific processes running on the machine.

At this stage, the analyst cross references the anomalous latency windows with concurrent host telemetry. The investigation focuses on three key factors:

- **Active processes and threads**: identifying which executables were running during the exact intervals where latency spikes occurred, isolating candidate sender and receiver programs;
- **CPU affinity**: checking whether the suspect processes were pinned to the same physical core, hyperthread, or shared cache slice;
- **Execution patterns**: evaluating whether the candidate sender repeatedly modified memory addresses while the candidate receiver executed loops to measure eviction latency.

Establishing that a process repeatedly modified cache lines while another thread experienced increased replacement latency completes the causal chain. This contextual alignment transforms an isolated microarchitectural fluctuation into attributable forensic evidence.

Combining the Evidence To build solid proof of a covert channel, the framework combines the results from all four layers into three main types of evidence:

1. **Difference from the baseline**: cache eviction times that are clearly higher than normal;
2. **Repeating temporal patterns**: regular frequency peaks and lags that show intentional, synchronized communication;
3. **Host consistency**: active processes and shared CPU cores that explain who caused the timing delays.

By connecting low level timing data with running processes on the host, this framework gives analysts a clear and transparent way to detect cache covert channels.

Chapter 5

Limitations and Future Works

5.1 Limitations of the Analysis and the Experimental Design

While the multi source forensic pipeline has, in some cases, shown the ability to correlate network and host artifacts to identify traffic anomalies that may indicate the presence of a covert channel, the experimental setup and evaluation methodology have several intrinsic limitations. These limitations should be kept in mind when interpreting the results, and they help define the requirements for applying this approach to real investigative scenarios in the future.

Risk of circularity and optimization bias

One of the most relevant methodological limitations concerns the risk of circularity in how the SHP channel's operating parameters were chosen. During implementation, parameters such as the BPF filter, the 8-bit bitlength, the multihashing value of 7, and a rounding factor of 0 were selected to maximize the channel's overall fitness, favoring useful throughput (bps) and decoding stability while reducing the mismatch rate. Using this performance optimized configuration to demonstrate that the channel is forensically detectable, however, introduces a significant bias.

In a real world scenario, an advanced, forensically aware attacker would not prioritize maximizing throughput, but would instead aim to minimize the channel detectability. Such an attacker would willingly accept a very low transmission rate or a higher error rate in exchange for keeping ARP traffic volume and binary entropy within limits indistinguishable from the baseline. As a result, demonstrating that a

performance optimized channel is detectable does not guarantee the same detection effectiveness against a variant specifically optimized to evade statistical detection.

This limitation points to a few directions for future work:

- Future experiments should include a phase where the covert channel is explicitly configured to evade detection metrics, for example by lowering the bit rate or enforcing artificial silence periods. This would make it possible to assess how robust the forensic pipeline is against a realistic, stealth attacker, and to identify the break even point at which the channel becomes indistinguishable from background noise.

- Multi objective optimization methods, such as genetic algorithms or reinforcement learning, could be used to find channel configurations that minimize detectability while maintaining a minimum acceptable throughput. This would shift the analysis from an idealized setting to a zero sum game between attacker and defender, producing more reliable detection metrics.

The ground truth problem and the need for blind scenarios

The second key limitation concerns how the results were validated and their reliance on ground truth. In the experiments conducted, for both the ARP and DNS implementations, time windows and processes were labeled as malicious or benign with prior knowledge of their true nature, since the covert channel was started and controlled directly during testing. This prior knowledge is methodologically appropriate for training and tuning the system, for instance when calculating the 95th percentile thresholds from baseline runs. For validation purposes, however, the analysis currently functions more as a confirmation analysis, confirming that the expected traces are indeed produced by the tool, rather than as genuine detection.

To validate the forensic effectiveness of this approach, future testing should include blind scenarios: datasets in which the analyst has no knowledge of whether, when, or through which processes the covert channel was activated, simulating the real conditions of uncertainty found in a forensic investigation.

This limitation points to a few directions for future work:

- Blind or CTF style test datasets could be created, in which the presence, parameters, and timing of the covert channel are randomized and hidden from the analyst. This would force the forensic pipeline, and any machine learning models involved, to generalize and prove their effectiveness without confirmation bias, showing that the system can operate under conditions of complete uncertainty.

- Rigorous cross validation, or out of sample testing, should strictly separate the data used to calculate thresholds, such as the 95th percentile, from the data used to measure false positives and false negatives. This would ensure that the model does not suffer from data leakage and that the derived rules remain statistically valid on network traffic never seen before.

Observer effect and limits of the allowlist approach

Another critical issue that emerged at the host level is the so called observer effect. The forensic monitoring tools built to capture network activity rely on the same technology stack, Python and packet capture libraries such as **wpcap.dll** and **packet.dll**, used by the covert channel itself. This introduces background noise into the system, where the monitoring tool contaminates the very environment it is trying to observe.

To address this during the experiment, an ad hoc allowlist was built to label monitoring and traffic generation processes as legitimate, classifying them as **known_instrumentation** or **known_experimental_support**, and excluding them from risk classification. In addition, the identification of suspicious processes currently depends on searching for specific, command line arguments, such as **--poi**, **--bitlength**, and **--inputsource**. In a real incident response case, an analyst would not have a ready made system allowlist available, and a skilled attacker would not pass the malware's parameters in plain text on the command line.

This limitation points to a few directions for future work:

- Monitoring could be moved out of band, decoupling it from the host by working at a lower level. On the host side, this could mean using virtual machine introspection (VMI) at the hypervisor level; on the network side, it could mean using hardware sensors such as network taps or SPAN ports. This would make monitoring effectively invisible to the observed host, removing the observer effect and avoiding conflicts in the use of system DLLs.
- String based rules that search for specific command line arguments could be replaced with advanced behavioural profiling through dynamic system call analysis, monitoring anomalies such as suspicious memory allocation combined with the opening of raw sockets. An attacker can easily obfuscate the command line or rename a process, but cannot as easily avoid the system calls required to inject traffic into the network interface.

Statistical and sample size limitations

Finally, the test infrastructure is limited by a small statistical sample size. For both the ARP and DNS implementations, the inferential analysis relied on a very small

number of runs, five baseline sessions and five SHP sessions. As the results show, network traffic can vary considerably even in baseline scenarios, as seen in the strong fluctuations of ARP traffic in **baseline2** and of DNS traffic in **baseline1**. This small sample makes percentile based thresholds, such as the 95th percentile for packet counts, highly dependent on individual observations, which weakens the robustness of the non parametric statistical tests (Mann–Whitney U) applied at the session level. Future work should include large scale sampling to stabilize the baseline and reduce false positives caused by natural network traffic fluctuations.

This limitation points to a few directions for future work:

- Long term, large scale sampling could be used to collect traffic sessions over extended periods, weeks or months, across diverse scenarios such as home, enterprise, and IoT networks, in order to build solid statistical distributions. A larger sample would absorb normal network fluctuations, such as traffic spikes caused by system updates, and would make inferential tests like the Mann–Whitney U test and percentile calculations statistically more robust.
- Static thresholds, such as counts above 11.05 packets per window, could be replaced with adaptive anomaly detection algorithms applied to time series data, such as ARIMA models or isolation forests. Since modern networks continuously change behaviour, an adaptive model would update its definition of normal activity in real time, making it possible to distinguish a legitimate burst, such as an authorized network scan, from a steganographic fluctuation.

5.2 Conclusions

This thesis explored the domain of steganographic covert channels, proposing a fundamental shift in digital forensics: moving from a classification based on the carrier medium to an approach focused on the nature of the trace (deterministic, statistical, or heuristic). The primary objective was to address the critical trade off between detection capability and the legal interpretability of evidence (explainability), overcoming the limitations of "black-box" models that achieve high accuracy but struggle to produce court admissible proof.

The experimental results, based on the implementation of the Silent History Protocol (SHP), demonstrated both the advantages and the limits of multi source forensic analysis. In the SHP over ARP scenario, covert activity generated persistent network traces and temporally structured bursts. Correlating these traffic anomalies with host level evidence, such as active processes, command line arguments, and memory loaded interception libraries like **wpcap.dll**, provided much

stronger forensic findings than inspecting any single source in isolation.

However, the SHP over DNS scenario revealed the clear limits of purely statistical approaches. In this setting, bivariate linear correlations (such as Pearson and Spearman tests) between host activity and packet volumes proved ineffective, producing weak and non significant coefficients. This suggests that for highly fragmented and asynchronous channels, temporal overlap alone is not enough to prove anything statistically and the signal might simply be too weak compared to background noise. Therefore, while hints suggest that multivariate methods could find patterns that bivariate analysis misses, this advantage has not been formally proven in this thesis. If advanced models were used to handle these complex interactions, they would bring back the challenge of explaining how they make decisions during legal proceedings.

As highlighted in the discussion of limitations, this methodology faces challenges against modern, forensically aware adversaries. The current pipeline exhibits potential constraints, including optimization bias toward channel performance, observer effects from monitoring tools (such as allowlists), and reliance on known ground truth. A sophisticated attacker could employ countermeasures such as randomizing traffic bursts, obfuscating process arguments, or logically decoupling execution threads to break host network correlation.

Alongside network analysis, this work laid the foundation for investigating in cache memory based covert channels. Due to strict hardware and synchronization constraints, this aspect was addressed theoretically by formalizing a four level forensic pipeline: temporal acquisition, local analysis, periodicity and spectral analysis, and host context correlation. This framework provides a structured conceptual model to identify latency anomalies and hidden communication patterns in hardware state.

In conclusion, this thesis demonstrates that no single indicator can reliably expose advanced covert channels. Reliable detection requires an integrated approach that uses cross layer correlation as a foundation, supported by future technical improvements. Future work should focus on testing against blind zero knowledge datasets, adopting external monitoring through Virtual Machine Introspection (VMI), and implementing dynamic system call profiling. Only through these advancements can incident response teams uncover stealthy communications while preserving the transparency required for forensic admissibility.

Bibliography

[1] Elzbieta Zielinska, Wojciech Mazurczyk, and Krzysztof Szczypiorski. «Development Trends in Steganography». In: *arXiv preprint arXiv:1202.5289* (Feb. 2012). DOI: 10.48550/arXiv.1202.5289 (cit. on pp. 4, 5, 7).

[2] Gustavus J. Simmons. «The Prisoners' Problem and the Subliminal Channel». In: *Advances in Cryptology: Proceedings of Crypto 83*. Boston, MA: Springer US, 1984. DOI: 10.1007/978-1-4684-4730-9_5 (cit. on p. 5).

[3] Krzysztof Szczypiorski. «Steganography in TCP/IP networks: State of the Art and a Proposal of a New System–HICCUPS». In: *Institute of Telecommunications' seminar, Warsaw University of Technology, Poland*. 2003 (cit. on pp. 5, 6).

[4] Fred B. Schneider. «Unexpected Communications Channels». In: *System Security*. Cornell University, 2025. Chap. 13. URL: https://www.cs.cornell.edu/fbs/publications/chptr.chnls.pdf (cit. on p. 7).

[5] A. K. Shukla, Vishu, and Amod Kumar. «Steganography for Invisible Communication: A Review». In: *International Journal of Advanced Research in Computer Science* 3.7 (2012), pp. 149–156. URL: https://api.semanticscholar.org/CorpusID:58304341 (cit. on p. 8).

[6] Trivikram Muralidharan, Aviad Cohen, Assaf Cohen, and Nir Nissim. «The infinite race between steganography and steganalysis in images». In: *Signal Processing* 201 (2022). DOI: https://doi.org/10.1016/j.sigpro.2022.108711 (cit. on p. 8).

[7] Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski. «Background Concepts, Definitions, and Classification». In: *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures*. John Wiley & Sons, Ltd., 2016. DOI: 10.1002/9781119081715.ch02 (cit. on pp. 9, 10).

[8] Wojciech Mazurczyk and Luca Caviglione. «Steganography in Modern Smartphones and Mitigation Techniques». In: *Commun. Surveys Tuts.* 17.1 (Jan. 2015). ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2350994. URL: https://doi.org/10.1109/COMST.2014.2350994 (cit. on p. 9).

[9] Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski. «Network Steganography». In: *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures*. John Wiley & Sons, Ltd., 2016. DOI: 10.1002/9781119081715.ch02 (cit. on pp. 10–12, 17).

[10] F. Bostancı et al. «Understanding and Mitigating Covert Channel and Side Channel Vulnerabilities Introduced by RowHammer Defenses». In: *arXiv:2503.17891* (2025). URL: https://arxiv.org/abs/2503.17891 (cit. on p. 13).

[11] Konstantinos Kanellopoulos, Fatma Nisa Bostanci, Ataberk Olgun, A. Giray Yaglikci, I. Emre Yuksel, Zülal Bingöl, Saugata Ghose, and Onur Mutlu. «Amplifying Main Memory-Based Timing Covert and Side Channels using Processing-in-Memory Operations». In: *arXiv preprint arXiv:2404.11284* (Apr. 2024). URL: https://arxiv.org/abs/2404.11284 (cit. on p. 14).

[12] Swadhesh Kumar and P K Singh. «An overview of modern cache memory and performance analysis of replacement policies». In: *2016 IEEE International Conference on Engineering and Technology (ICETECH)*. 2016. DOI: 10.1109/ICETECH.2016.7569243 (cit. on p. 14).

[13] Yujie Cui and Xu Cheng. «Abusing Cache Line Dirty States to Leak Information in Commercial Processors». In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2021). URL: https://api.semanticscholar.org/CorpusID:233296823 (cit. on pp. 14–16, 41, 42).

[14] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. «Flush+Flush: A Fast and Stealthy Cache Attack». In: *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*. Berlin, Heidelberg: Springer-Verlag, 2016. DOI: 10.1007/978-3-319-40667-1_14. URL: https://doi.org/10.1007/978-3-319-40667-1_14 (cit. on p. 15).

[15] Sebastian Neuner, Artemios G. Voyiatzis, Martin Schmiedecker, Stefan Brunthaler, Stefan Katzenbeisser, and Edgar R. Weippl. «Time is on my side: Steganography in filesystem metadata». In: *Digital Investigation* 18 (2016). ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2016.04.010. URL: https://www.sciencedirect.com/science/article/pii/S1742287616300433 (cit. on p. 16).

[16] MD Mahady Hassan, Shanto Roy, and Reza Rahaeimehr. «Memory under siege: A comprehensive survey of side-Channel attacks on memory». In: *Computers Security* 163 (2026). DOI: https://doi.org/10.1016/j.cose.2025.104810 (cit. on p. 17).

[17] Steffen Wendzel et al. «A Revised Taxonomy of Steganography Embedding Patterns». In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. New York, NY, USA: Association for Computing Machinery, 2021. DOI: 10.1145/3465481.3470069. URL: https://doi.org/10.1145/3465481.3470069 (cit. on pp. 18–23, 29).

[18] Claude Roux et al. «The Sydney declaration – Revisiting the essence of forensic science through its fundamental principles». In: *Forensic Science International* 332 (2022). DOI: https://doi.org/10.1016/j.forsciint.2022.111182. URL: https://www.sciencedirect.com/science/article/pii/S0379073822000123 (cit. on p. 23).

[19] Indra Abeysekera. «Open Source Tools: An Evaluation for Digital Forensic Investigations». In: *Frontiers in Research Metrics and Analytics* 11 (2026). DOI: 10.3389/frma.2026.1790333 (cit. on p. 23).

[20] Andrea Atzeni. *CF Terminology & Relevant Concepts*. Lecture slides from the course Computer Forensic and Crime Analysis. Academic year 2024/2025. 2024 (cit. on pp. 24–28).

[21] Rajasekhar Chaganti, Vinayakumar Ravi, Mamoun Alazab, and Tuan D Pham. «Stegomalware: A systematic survey of malware hiding and detection in images, machine learning models and research challenges». In: *arXiv preprint arXiv:2110.02504* (2021) (cit. on pp. 29, 31).

[22] Kristiyan Michaylov. «Exploring the Use of Steganography and Steganalysis in Forensic Investigations for Analysing Digital Evidence». In: *TScIT39*. Enschede, The Netherlands: University of Twente, 2023 (cit. on pp. 29–33).

[23] RFC Editor. *RFC Series*. URL: https://www.rfc-editor.org/series/rfc/ (visited on 08/26/2026) (cit. on p. 29).

[24] Shraddha Khonde, Rutuja Gaikwad, Pratiksha Chavan, and Dnyaneshwari Rakshe. «Network Steganography for Secure Communication: A Survey». In: *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)* (2022). DOI: 10.17148/IJARCCE.2022.111224 (cit. on p. 29).

[25] M. Srinivas, A. Nayak, and A. Bhatt. «Forged File Detection and Steganographic Content Identification (FFDASCI) Using Deep Learning Techniques». In: *CLEF 2019 Working Notes*. 2019. URL: http://ceur-ws.org/Vol-2380/paper_142.pdf (cit. on pp. 29, 30).

[26] Salam Ghanim Najeeb and Noor Alhuda Fadhil. «A Multi-Layer Network Steganography Framework Using ICMP and NTFS Alternative Data Streams for Secure Covert». In: *International Journal of Engineering and Techniques (IJET)* 12 (Jan. 2026) (cit. on p. 30).

[27] Haozhi Li, Tian Song, and Yating Yang. «Generic and Sensitive Anomaly Detection of Network Covert Timing Channels». In: *IEEE Transactions on Dependable and Secure Computing* 20 (2023). DOI: 10.1109/TDSC.2022.3207573 (cit. on pp. 31, 34, 35).

[28] Sonam Chhikara and Rajeev Kumar. «An Information Theoretic Image Steganalysis for LSB Steganography». In: *Acta Cybernetica* 24 (July 2020). DOI: 10.14232/actacyb.279174 (cit. on pp. 31, 32).

[29] The Editors of Encyclopaedia Britannica. *Chi-squared Test*. URL: https://www.britannica.com/topic/chi-squared-test (visited on 08/26/2026) (cit. on p. 32).

[30] Hamza Kheddar, Mustapha Hemis, Yassine Himeur, David Megías, and Abbes Amira. «Deep learning for steganalysis of diverse data types: A review of methods, taxonomy, challenges and future directions». In: *Neurocomputing* 581 (May 2024). DOI: 10.1016/j.neucom.2024.127528. URL: http://dx.doi.org/10.1016/j.neucom.2024.127528 (cit. on pp. 33, 34).

[31] Ntivuguruzwa Jean De La Croix, Tohari Ahmad, and Fengling Han. «Comprehensive survey on image steganalysis using deep learning». In: *Array* 22 (2024). ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2024.100353. URL: https://www.sciencedirect.com/science/article/pii/S2590005624000195 (cit. on pp. 33–36).

[32] Massimo Guarascio, Marco Zuppelli, Nunziato Cassavia, Giuseppe Manco, and Luca Caviglione. «Detection of Network Covert Channels in IoT Ecosystems Using Machine Learning». In: *ITASEC'22: Italian Conference on Cybersecurity*. CEUR-WS.org, June 2022 (cit. on pp. 33–35).

[33] Mario Alejandro Bravo-Ortiz, Esteban Mercado-Ruiz, Juan Pablo Villa-Pulgarin, Carlos Angel Hormaza-Cardona, Sebastian Quiñones-Arredondo, Harold Brayan Arteaga-Arteaga, Simon Orozco-Arias, Oscar Cardona-Morales, and Reinel Tabares-Soto. «CVTStego-Net: A convolutional vision transformer architecture for spatial image steganalysis». In: *Journal of Information Security and Applications* 81 (2024). DOI: https://doi.org/10.1016/j.jisa.2023.103695. URL: https://www.sciencedirect.com/science/article/pii/S221421262300279X (cit. on p. 34).

[34] Salvatore Saeli, Federica Bisio, Pierangelo Lombardo, and Danilo Massa. «DNS Covert Channel Detection via Behavioral Analysis: A Machine Learning Approach». In: *arXiv preprint arXiv:2010.01582* (2020). DOI: 10.48550/arXiv.2010.01582 (cit. on pp. 34, 35).

[35] Sharath Babu Cg and Komala K. «A Novel Approach for Secure IoT Architecture Using Deep Learning-Based Image Steganography and Multi-Factor Authentication – A Review». In: *International Journal of Engineering Research & Technology (IJERT)* 14.6 (2026). ACSCON 2026 (cit. on p. 34).

[36] Nunziato Cassavia, Luca Caviglione, Massimo Guarascio, Angelica Liguori, and Marco Zuppelli. «Learning autoencoder ensembles for detecting malware hidden communications in IoT ecosystems». In: *Journal of Intelligent Information Systems* 62 (2024). DOI: 10.1007/s10844-023-00819-8 (cit. on p. 34).

[37] IBM. *Che cos'è l'apprendimento non supervisionato?* URL: https://www.ibm.com/it-it/think/topics/unsupervised-learning (visited on 08/26/2026) (cit. on p. 35).

[38] Christian Kraetzer and Mario Hildebrandt. «Explainability and Interpretability for Media Forensic Methods: Illustrated on the Example of the Steganalysis Tool Stegdetect». In: *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2024) - Volume 4: VISAPP*. Jan. 2024. DOI: 10.5220/0012423800003660 (cit. on pp. 35, 37).

[39] Mohammad Wali Ur Rahman, Yu-Zheng Lin, Carter Weeks, David Ruddell, Jeff Gabriellini, Bill Hayes, Salim Hariri, Pratik Satam, and Edward V. Ziegler. «AI/ML Based Detection and Categorization of Covert Communication in IPv6 Network». In: *Cybersecurity* 9 (2026). DOI: 10.1186/s42400-025-00485-z (cit. on p. 35).

[40] X. Fu, Xiaojiang Du, and B. Luo. «Data correlation-based analysis methods for automatic memory forensic». In: *Security and Communication Networks* 8 (Sept. 2015). DOI: 10.1002/sec.1337 (cit. on p. 39).

[41] Christoph Weissenborn and Steffen Wendzel. «History covert channels with relative timing pointers: Design, error resilience, and detectability». In: *Journal of Information Security and Applications* (2026). ISSN: 2214-2126. DOI: 10.1016/j.jisa.2026.104431. URL: http://dx.doi.org/10.1016/j.jisa.2026.104431 (cit. on pp. 40, 41).

[42] Nadim Nachar. «The Mann-Whitney U: A Test for Assessing Whether Two Independent Samples Come from the Same Distribution». In: *Tutorials in Quantitative Methods for Psychology* 4 (Mar. 2008), pp. 13–20. DOI: 10.20982/tqmp.04.1.p013 (cit. on pp. 61, 62).

[43] José A. Ferreira. «The Benjamini-Hochberg Method in the Case of Discrete Test Statistics». In: *The International Journal of Biostatistics* 3.1 (2007). DOI: doi:10.2202/1557-4679.1065 (cit. on p. 62).