

From Raw Honeypot Logs to Campaign-Level Intelligence: A Framework for Bot Detection and Attacker Profiling

by

Fatemeh Nejati

M.Sc. in Mathematics, Kharazmi University, 2019

B.Sc. in Mathematics, Shiraz University, 2016

A Thesis Submitted in Partial Fulfilment of
the Requirements for the Degree of

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisors: Ali A. Ghorbani, Ph.D., Faculty of Computer Science
Rongxing Lu, Ph.D., Faculty of Computer Science

Examining Board: Sajjad Dadkhah, Ph.D.,
Faculty of Computer Science, Chair
Suprio Ray, Ph.D., Faculty of Computer Science
Hong Chen, Ph.D.,
Department of Electrical and Computer Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

June 2026

© Fatemeh Nejati, 2026

Abstract

Modern cyber attacks are increasingly automated and occur at a large scale, which makes them difficult to analyze by using traditional methods. Although honeypots can collect detailed data about attacker activity, this data is often noisy, repetitive, and hard to interpret. As a result, security analysts need approaches that focus on understanding attacker behavior rather than relying only on simple indicators such as IP addresses or event counts.

This thesis presents the Bot Detection and Behavioral Profiling Framework, a scalable and interpretable approach for analyzing large-scale Secure Shell (SSH) honeypot data to detect bots, understand attacker behavior, and identify coordinated attack campaigns. Modern attackers often rely on automation, which creates large amounts of noisy and complex data that are difficult to analyze directly. To address this, the proposed framework transforms raw honeypot logs into structured behavioral insights through three main stages: preprocessing, command normalization, and attacker profiling.

First, the raw data is cleaned and organized into meaningful sessions. A graph-based masking method then separates stable parts of commands from changing elements, which reduces noise while preserving important behavior. Next, a rule-based grouping method removes repeated sessions and improves efficiency. Finally, a behavior-based detection engine identifies automated attackers and builds attacker profiles

based on their actions, timing, and intent. These profiles are used to construct an attacker similarity graph to reveal groups of attackers with similar behavior.

The framework is evaluated by using an eleven-month Cowrie honeypot dataset with more than 1.5 million command interactions. The results show that the framework reduces similarity computations by 94.79% and achieves a bot detection accuracy of 94.89%. Overall, this work demonstrates how raw honeypot data can be transformed into useful, campaign-level security insights.

Dedication

To my husband, for your love and for always believing in me when I doubted myself.

To my family, whose endless love has remained with me, no matter the distance.

To my sister, whose memory stays close to my heart.

To my supervisors, for all your guidance, and support throughout this journey.

And to all those who lit my path and guided me toward a meaningful life.

Table of Contents

Abstract	ii
Dedication	iv
Table of Contents	v
List of Tables	x
List of Figures	xi
List of Symbols	xii
List of Abbreviations	xiv
1 Introduction	1
1.1 Overview and Motivation	1
1.2 Problem Statement	2
1.3 Research Objectives	4
1.4 Research Contributions	4
1.5 Research Questions	5
1.6 Thesis Organization	6
2 Background and Related Work	8
2.1 Background	8
2.1.1 Honeypot Technologies and Cowrie Data Collection	9

2.1.1.1	Interaction Levels	9
2.1.1.2	The Cowrie Honeypot Architecture	9
2.1.2	Behavioral Modeling and Data Representation	10
2.1.2.1	Basic Definitions	10
2.1.2.2	Session Abstraction and behavioral Traces	11
2.1.2.3	Heterogeneous Information Networks	11
2.1.2.4	Graph-Topological Command Masking	12
2.1.2.5	Run-Length Encoding	12
2.1.3	Behavioral Features and Attacker Profiles	13
2.1.3.1	Bot and Human Attackers	13
2.1.4	Similarity and Alignment Metrics	14
2.1.4.1	Structural Similarity Using Nilsimsa	14
2.1.4.2	Semantic Similarity Using Cosine Similarity	14
2.1.4.3	Transformer-Based Embeddings	14
2.1.4.4	Exact Sequence-Based Matching	15
2.1.4.5	Hybrid Similarity Scoring	15
2.1.4.6	Fuzzy Command Overlap	16
2.1.5	Attacker Similarity Graph and Connected Components	16
2.1.5.1	Connected Components	17
2.1.6	Security Frameworks and Threat Characterization	17
2.1.6.1	MITRE ATT&CK	17
2.1.6.2	Cyber Kill Chain	17
2.1.6.3	Campaign-Level Attribution	18
2.1.7	Unsupervised Learning and Clustering Evaluation	18
2.1.7.1	K-Means Clustering	18
2.2	Related Work	19
2.2.1	Honeypot-Based Cyber Attack Data Collection	19

2.2.2	Botnets and Automated Attack Ecosystems	20
2.2.3	Preprocessing and Session Modeling	21
2.2.4	Command Abstraction and Representation in Attack Analysis	22
2.2.5	behavioral Similarity and Pattern Matching	23
2.2.6	Graph-Based Threat Intelligence and Campaign Attribution .	23
2.2.7	MITRE ATT&CK and behavioral Semantics	24
2.2.8	Rule-Based and Learning-Based Bot Detection	25
2.2.9	Comparative Summary of Related Work	26
2.2.10	Comparative Summary of Related Work	26
2.2.11	Summary and Positioning of This Work	29
3	Methodology	30
3.1	Module 1: Data Collection and Preprocessing	31
3.1.1	Stage 1: Dataset Collection	33
3.1.2	Stage 2: Session Reconstruction	33
3.1.2.1	Event Filtering and Noise Reduction	34
3.1.2.2	Session Grouping and Trace Construction	35
3.1.2.3	Temporal Ordering and Sequence Preparation	35
3.2	Module 2: Session Reduction and Command Normalization	36
3.2.1	Stage 1: Dynamic Command Masking via a Heterogeneous Cyber-Knowledge Graph (H-CKG)	37
3.2.1.1	Graph-Based Token Classification and Command Nor- malization	38
3.2.1.2	Hybrid Safety Filtering	42
3.2.2	Stage 2: Hierarchical session similarity analysis and reduction through rule-based session grouping.	42
3.2.2.1	Sequence-Based Session Grouping and Complexity Reduction	43

3.2.2.2	Rule-Based Session Reduction	44
3.3	Module 3: Bot Detection and Attacker Profiling	48
3.3.1	Stage 1: Feature Extraction	49
3.3.1.1	Feature Summary	53
3.3.2	Stage 2: Rule-Based Bot Detection Engine	53
3.3.2.1	Behavior-weighted score.	55
3.3.2.2	Similarity-based components.	55
3.3.2.3	Bot Label Assignment	57
3.3.2.4	Stage 3: Attacker Profile Construction	59
3.3.2.5	Profile Construction	59
3.4	Conclusion	61
4	Evaluation and Analysis	63
4.1	RQ1: Efficacy of Topological Masking and Hybrid Filtering	65
4.1.1	Quantitative Discovery Results	66
4.1.2	Hybrid Safety Filter Validation	68
4.1.3	Discussion	68
4.2	RQ2: Effectiveness of Rule-Based Session Grouping and Reduction	69
4.2.1	Reduction in Computational Complexity	69
4.2.2	Reduction in Dataset Size	71
4.2.3	Discussion	72
4.3	RQ3: Bot Detection Accuracy	73
4.3.1	Semi-Supervised Centroid Validation	74
4.3.2	Quantitative Results and Error Analysis	74
4.3.3	Discussion	76
4.4	RQ4: Behavioral Differentiation and Similarity Structure of Bot and Human Attackers	77
4.4.1	Methodology	77

4.4.2	Quantitative Results	78
4.4.3	Discussion	80
4.5	RQ5: Attribution and Detection of Coordinated Multi-Identity Attack	
	Groups	80
4.5.1	Methodology	81
4.5.2	Quantitative Results	82
4.5.3	Discussion	83
4.6	Chapter Summary	85
5	Conclusion, Limitations, and Future Work	87
5.1	Conclusion	87
5.2	Limitations	89
5.3	Future Work	90
5.4	Final Remarks	93
	Bibliography	98
	Vita	

List of Tables

2.1	Summary of related-work contributions and remaining gaps.	26
3.1	Summary of Module 1 stages and their effects on the data.	36
3.2	Summary of Module 2 stages and their effects on the data.	48
3.3	Command categories and severity weights used for maliciousness scoring.	51
3.4	Session-level and attacker-context behavioral features extracted in Module 3.	54
3.5	Rule-based conditions and point assignments used to compute the behavior-weighted score.	56
3.6	Similarity-based scoring rules used to compute the similarity signal and clone penalty.	57
3.7	Session-level bot label assignment rules.	59
4.1	Confusion matrix for bot detection results.	75
4.2	Bot detection performance metrics.	75
4.3	Average normalized behavioral profile values for human-operated at- tackers and automated bots.	78
4.4	Four largest behavioral groups identified from the attacker similarity graph using a cosine similarity threshold greater than 85%.	84

List of Figures

3.1	High-level architecture of the Bot Detection and Profiling Framework, illustrating the data flow from raw ingestion to final attacker classification.	32
3.2	Overview of the 4-layer Heterogeneous Cyber-Knowledge Graph (HCKG)	39
4.1	Mapping of research questions to the corresponding stages of the analysis pipeline.	66
4.2	Topological classification of command tokens based on degree centrality. Most tokens are identified as polymorphic variables, while only a small stable syntax core is preserved.	67
4.3	Reduction in pairwise similarity comparisons achieved by rule-based grouping.	71
4.4	Session reduction across the processing pipeline.	72
4.5	behavioral profile comparison between automated bot attackers (orange) and human-operated attackers (green).	79
4.6	Attacker Similarity Graph of behavioral Group 15. Nodes represent distinct source IP identities, and edges indicate strong behavioral similarity ($> 85\%$). The dense connections suggest coordinated activity across multiple distributed attackers.	83

List of Symbols

T	Behavioral trace representing the ordered sequence of commands in a session.
c_i	The i th command in a behavioral trace.
n	Total number of commands in a session or behavioral trace.
t	Token extracted from a command.
C	Set of commands used for token reuse and command masking analysis.
$\deg^-(t)$	In-degree of token t , representing the number of commands containing token t .
N	Total number of sessions considered in a pairwise comparison process.
U	Number of unique behavioral signatures after exact signature-based grouping.
A	Behavioral feature vector used in similarity computation.
B	Behavioral feature vector used in similarity computation.
$S_{\text{structural}}$	Structural similarity score between command or session representations.

S_{semantic}	Semantic similarity score between behavioral or tactic representations.
S_{total}	Hybrid similarity score combining structural and semantic similarity.
α	Weight assigned to structural similarity in the hybrid similarity score.
β	Weight assigned to semantic similarity in the hybrid similarity score.
K	Number of clusters used in K-Means clustering.
$\mathcal{S}_A$	Set of sessions associated with attacker A .
S_i	Individual session belonging to an attacker.
U_i	Number of unique behavioral templates associated with attacker i .
x_i	Session-level behavioral feature vector.
$x_i^{(j)}$	The j th feature value of session feature vector x_i .
X_A	Aggregated behavioral profile vector representing attacker A .
d	Dimensionality of a behavioral feature vector.
$\mathcal{C}$	Set of indices corresponding to continuous behavioral features.
$\mathcal{B}$	Set of indices corresponding to binary behavioral features.
$\binom{N}{2}$	Number of unique pairwise comparisons among N sessions.

List of Abbreviations

ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge.
C2	Command and Control.
CIC	Canadian Institute for Cybersecurity.
CSV	Comma-Separated Values.
CVE	Common Vulnerabilities and Exposures.
DNS	Domain Name System.
FN	False Negative.
FP	False Positive.
GIN	Graph Isomorphism Network.
H-CKG	Heterogeneous Cyber-Knowledge Graph.
HIN	Heterogeneous Information Network.
HTTP	Hypertext Transfer Protocol.
IP	Internet Protocol.
JSON	JavaScript Object Notation.
MITRE	MITRE Corporation; used in the context of the MITRE ATT&CK framework.
RLE	Run-Length Encoding.

RQ	Research Question.
SBERT	Sentence-BERT.
SSH	Secure Shell.
TN	True Negative.
TP	True Positive.
TTPs	Tactics, Techniques, and Procedures.
UNB	University of New Brunswick.
URL	Uniform Resource Locator.

Chapter 1

Introduction

1.1 Overview and Motivation

Modern adversaries no longer rely only on manual intrusion attempts. They increasingly use bots, scripts, scanners, and distributed infrastructure to target large numbers of internet-facing systems in an automated way. These campaigns can perform credential attacks, reconnaissance, malware delivery, and persistence attempts at a scale that is difficult for traditional defenses to manage [1]. Recent long-term Secure Shell (SSH) honeynet measurements continue to show sustained hostile automation against exposed services, including large volumes of brute-force attempts, scanning behavior, and repeated post-access actions [2].

SSH remains a common and persistent target because it is widely used for remote administration and system management, with recent studies identifying more than 15 million SSH servers exposed on the public internet [3]. Exposed SSH services regularly receive password guessing attempts, unauthorized login attempts, post-compromise command execution, and malware deployment.

Honeypots provide an effective way to observe this threat landscape. A honeypot is a

controlled system designed to attract attackers and record their actions. Unlike many traditional defensive tools that mainly focus on blocking or alerting, honeypots can capture attacker commands, interaction sequences, timing behavior, and post-access activity [4].

However, collecting honeypot data is easier than extracting value from it. Large deployments can generate millions of raw events and hundreds of thousands of sessions. These logs often contain repeated activity, random command parameters, reused malware scripts, changing infrastructure, and many distributed source IP addresses [5]. Manual analysis at this scale is slow, expensive, and often impractical.

As a result, defenders need methods that move beyond simple event counting or indicator collection. Instead of only listing IP addresses or commands, security teams need approaches that can identify automation, distinguish attacker types, summarize repeated behavior, and uncover coordinated campaigns [6].

This thesis addresses that need by proposing a scalable and interpretable analysis framework for honeypot data. The framework is designed to normalize noisy telemetry, detect bots, profile attacker behavior, reduce redundancy, and identify coordinated multi-identity attack groups through similarity analysis.

1.2 Problem Statement

Although honeypots are widely used in cybersecurity research and defensive operations, many current analysis approaches remain limited.

The first challenge is scale and noise. Large honeypot datasets often contain repeated commands, duplicated sessions, attacker-specific variables, and incomplete interactions. While these logs contain valuable information, the volume and inconsistency of raw data make direct analysis difficult.

Because of this complexity, many studies rely on simpler indicators such as source IP addresses, geolocation, or event counts [4]. However, these signals do not always reflect the true structure of attacker behavior. Modern attackers frequently rotate IP addresses, use rented cloud infrastructure, or spread activity across many visible identities. As a result, related attacks may appear unrelated when viewed only through network identifiers [7].

This creates a need for behavior-based analysis. Rather than identifying geographical or network origins, a more effective paradigm focuses on modeling the underlying behavioral mechanics of the adversary. However, separating automated bots from human-operated attackers is not always straightforward. Some bots are noisy and obvious, while others are lightweight, fast, and use limited interaction. Human attackers may also use scripts during manual intrusions. This creates overlap between categories [8].

A further challenge is the lack of reliable ground-truth labels. Many security datasets do not provide verified labels for attacker type or automation level. This limits the use of purely supervised learning methods and increases the need for interpretable approaches that can work under limited labeling conditions.

Finally, many existing studies focus on descriptive statistics or isolated indicators and do not move toward attacker-level profiling or campaign-level understanding.

These limitations show that data collection alone is not enough. There is a clear need for methods that can clean raw telemetry, reduce redundancy, detect automation, and model attacker behavior to reveal hidden relationships between distributed attackers.

In response, this thesis develops a unified, behavior-centered framework that connects command normalization, bot detection, attacker profiling, and campaign discovery directly from raw honeypot interactions.

1.3 Research Objectives

The main aim of this thesis is to design and evaluate a scalable and interpretable framework that transforms large-scale honeypot logs into structured attacker-focused insights for bot detection, attacker profiling, and campaign-level analysis.

To achieve this aim, the thesis addresses the following objectives:

- Develop a graph-based masking method that separates stable command syntax from attacker-specific variables.
- Reduce duplicated honeypot sessions through rule-based grouping and behavioral template extraction.
- Design an interpretable bot detection pipeline using behavioral indicators rather than black-box prediction alone.
- Build attacker-level behavioral profiles that summarize repeated patterns across multiple sessions.
- Construct attacker similarity graphs to identify coordinated multi-identity attack groups that are less visible through IP-based analysis.
- Evaluate the framework using a large real-world SSH honeypot dataset.

1.4 Research Contributions

This thesis makes several contributions to honeypot analytics and attacker behavior analysis.

First, it proposes an integrated multi-stage framework that connects preprocessing, detection, profiling, and attribution within one consistent pipeline.

Second, it introduces a graph-based command masking approach that automatically identifies changing command variables and improves normalization quality in noisy command telemetry.

Third, it presents a rule-based grouping strategy that removes repeated sessions and significantly reduces the computational cost of large-scale similarity analysis.

Fourth, it develops an interpretable behavioral bot detection method that is evaluated for its ability to identify automated attackers without relying entirely on opaque machine learning models.

Fifth, it develops attacker-level behavioral profiles that are used to examine measurable differences between automated bots and human-operated attackers.

Sixth, it applies attacker similarity graphs to identify potential coordinated groups, shared workflows, and common infrastructure that are less visible through IP-based analysis alone.

Finally, the thesis provides a large empirical evaluation using an eleven-month Cowrie SSH honeypot dataset containing more than 1.5 million command interactions, 20,220 reconstructed sessions, and 4,896 unique attackers.

1.5 Research Questions

To evaluate the proposed framework, this thesis is organized around the following research questions:

1. **RQ1 – Efficacy of Topological Masking and Hybrid Filtering:** How effectively does the proposed graph-based masking approach, combined with a hybrid safety filter, identify and normalize variable components of attack commands?

2. **RQ2 – Effectiveness of Rule-Based Session Grouping and Reduction:**

How effectively does the proposed grouping approach reduce redundancy, computational cost, and dataset size in large honeypot telemetry?

3. **RQ3 – Bot Detection Accuracy:** How accurately does the proposed rule-based detection pipeline identify automated attackers, and what attacker behaviors remain difficult to detect?

4. **RQ4 – behavioral Differentiation and Similarity Structure:** To what extent do attacker behavioral profiles distinguish automated bots from human-operated attackers, and do bots show higher intra-class similarity?

5. **RQ5 – Attribution and Coordinated Group Detection:** Can attacker similarity graphs reveal coordinated multi-identity attack groups that are not visible through IP-based analysis alone?

Together, these questions provide a step-by-step evaluation of the framework, beginning with data normalization and reduction, then moving to detection, profiling, and campaign-level analysis.

1.6 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2 presents background concepts and related work covering honeypots, bot activity, attacker profiling, graph analysis, and behavioral threat intelligence.

Chapter 3 describes the proposed methodology, including data collection, preprocessing, masking, grouping, bot detection, attacker profiling, and graph-based similarity modeling.

Chapter 4 presents the experimental evaluation and analysis structured around the five research questions.

Chapter 5 concludes the thesis, discusses limitations, and outlines future research directions.

Overall, this thesis demonstrates how large-scale honeypot telemetry can be transformed into structured attacker-centric insights for research and operational analysis. To build the foundation for this work, the next chapter reviews honeypots, attacker behavior modeling, graph-based analysis, and related detection approaches.

Chapter 2

Background and Related Work

This chapter provides the background and related work needed to understand the proposed framework. The first section introduces the main concepts that are used in this thesis, including honeypot data collection, Cowrie telemetry, behavioral modeling, command normalization, similarity analysis, attacker profiling, and clustering. The second section reviews previous studies on honeypot-based attack analysis, botnet behavior, command representation, behavioral similarity, graph-based attribution, MITRE ATT&CK mapping, and bot detection. Together, these sections explain the technical foundation of the thesis and show the research gap that motivates the proposed behavioral framework for SSH honeypot telemetry analysis.

2.1 Background

This section introduces the key concepts needed to understand the proposed bot detection and attacker profiling framework. It covers honeypot data collection, behavioral modeling, similarity measurement, graph-based analysis, threat characterization, and clustering.

2.1.1 Honeypot Technologies and Cowrie Data Collection

A **honeypot** is a security monitoring system that acts as a decoy by appearing as a legitimate and vulnerable target to attract attackers. Honeypots are used to observe attacker behavior, detect new threats, and collect detailed interaction data [9].

Honeypot telemetry refers to the structured data collected from these systems. It can include commands, timestamps, session identifiers, source information, and other metadata. In this thesis, honeypot telemetry is the main source of data for analyzing attacker behavior.

2.1.1.1 Interaction Levels

Honeypots are commonly classified based on their level of interaction:

- **Low-interaction honeypots:** These provide limited simulation and collect basic information about attacker activity.
- **Medium-interaction honeypots:** These simulate services and allow attackers to execute commands. They provide more realistic behavioral data.
- **High-interaction honeypots:** These use real or near-real systems and collect rich data, but they require more maintenance and involve higher risk [5].

2.1.1.2 The Cowrie Honeypot Architecture

Cowrie is a medium-interaction honeypot that simulates SSH and Telnet services. It is used in this thesis because it records detailed attacker interactions in a controlled environment [10].

Cowrie supports behavioral analysis through:

- a simulated command execution environment,
- a realistic virtual file system such as `/home` and `/etc`,
- structured logging of attacker interactions.

Each interaction is stored as a JSON event. In this thesis, `cowrie.command.input` events are used as the main data source because they represent commands entered by attackers after accessing the honeypot.

These command events are later grouped into sessions and transformed into behavioral traces.

2.1.2 Behavioral Modeling and Data Representation

The goal of behavioral modeling is to transform raw honeypot logs into structured representations that can be analyzed.

2.1.2.1 Basic Definitions

A **command** is a shell instruction entered by an attacker, such as `wget`, `chmod`, or `curl`.

A **token** is a smaller part of a command. For example, a command may contain command names, file names, URLs, IP addresses, options, and arguments.

A **session** is a sequence of interaction events within one connection between an attacker and the honeypot. In this thesis, a session represents one continuous interaction trace.

An **attacker** is an entity associated with a source identity, such as an IP address, that may generate one or more sessions over time. Although IP addresses are used as

the initial attacker identity, this thesis focuses on behavior because related attackers may use different IP addresses.

2.1.2.2 Session Abstraction and behavioral Traces

A **behavioral trace** represents the sequence of commands issued during a session. It is defined as:

$$T = \{c_1, c_2, \dots, c_n\} \quad (2.1)$$

where each c_i is a command.

This representation allows the framework to analyze attacker behavior as a sequence of actions instead of relying only on network identifiers such as IP addresses.

2.1.2.3 Heterogeneous Information Networks

A **Heterogeneous Information Network (HIN)** is a graph that contains different types of nodes and edges.

In this thesis, attacker activity is modeled using the following structure:

$$\text{Attacker} \rightarrow \text{Session} \rightarrow \text{Command} \rightarrow \text{Token}$$

This structure helps capture the relationships between attackers, their sessions, their commands, and the tokens inside those commands.

2.1.2.4 Graph-Topological Command Masking

Attack commands often contain both stable and changing parts. For example, commands such as `wget` or `chmod` may appear many times, while file names, IP addresses, and URLs may change often.

To reduce this noise, command masking is applied. Tokens are classified into two groups:

- **Stable tokens:** Common command elements that appear frequently, such as `wget`, `chmod`, and `sh`.
- **Variable tokens:** Changing elements such as IP addresses, URLs, random file names, and temporary paths.

Variable tokens are replaced with generic placeholders. This helps the framework focus on behavior instead of random command details.

The stability of a token t is measured using its reuse across commands:

$$\deg^-(t) = |\{c \in C : t \in c\}| \quad (2.2)$$

Tokens with high reuse are treated as stable tokens, while tokens with low reuse are treated as variable tokens.

2.1.2.5 Run-Length Encoding

Run-Length Encoding (RLE) is used to compress repeated actions. For example:

$$\{c_1, c_1, c_1, c_2\} \rightarrow \{(c_1, 3), (c_2, 1)\} \quad (2.3)$$

This reduces repeated behavior and makes tactic sequences easier to summarize.

2.1.3 Behavioral Features and Attacker Profiles

A **behavioral feature** is a numerical value that describes attacker activity. Examples include the number of commands, session duration, average time between commands, command diversity, and maliciousness score.

A **behavioral feature vector** represents an attacker or a session using a set of these numerical features.

An **attacker behavioral profile** is an aggregated representation of an attacker across multiple sessions. It summarizes the attacker’s timing behavior, command structure, repeated patterns, and attack intent.

This profile is later used for bot detection, bot-human comparison, and attacker similarity analysis.

2.1.3.1 Bot and Human Attackers

In this thesis, a **bot** is an automated attacker that executes commands through scripts, malware, or automated tools. Bots often show fast execution, repeated command patterns, and similar behavior across sessions.

A **human attacker** is an attacker whose behavior appears more manual. Human attackers usually show more variation, slower timing, and less repetition.

This distinction is behavioral rather than absolute. Some human attackers may use scripts, and some bots may perform only a few commands. Therefore, this thesis uses multiple behavioral signals instead of relying on only one feature.

2.1.4 Similarity and Alignment Metrics

Similarity analysis is used to compare sessions, group repeated behavior, and identify related attackers.

2.1.4.1 Structural Similarity Using Nilsimsa

Nilsimsa is a fuzzy hashing method used to measure structural similarity between command sequences. It produces similar hash values for similar inputs.

Nilsimsa similarity scores range from -128 to 128 , where higher values indicate stronger similarity [11].

2.1.4.2 Semantic Similarity Using Cosine Similarity

behavior can also be represented as numerical vectors. Cosine similarity measures how close two vectors are in direction:

$$\text{Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|} \quad (2.4)$$

This helps compare behavior even when the exact commands are different.

2.1.4.3 Transformer-Based Embeddings

Sentence-BERT (SBERT) is used to convert text or sequences into vector representations. These vectors capture semantic meaning and context.

In this thesis, SBERT is applied to tactic sequences to compare attackers at a semantic level.

2.1.4.4 Exact Sequence-Based Matching

Sequence-based matching compares two behavioral sessions using the ordered sequence of commands executed within each session. Two sessions are considered an exact match when they contain the same normalized commands in the same order. Formally, let

$$S_i = [c_1, c_2, \dots, c_m], \quad S_j = [c'_1, c'_2, \dots, c'_n] \quad (2.5)$$

denote two command sequences. They are identical if and only if

$$m = n \quad \text{and} \quad c_k = c'_k \quad \text{for all } k \in \{1, \dots, m\}. \quad (2.6)$$

This method is useful for identifying repeated execution patterns, particularly when automated scripts reproduce the same command sequence across multiple sessions. However, exact sequence matching does not identify near-duplicate sequences containing minor command changes, insertions, or deletions; therefore, approximate similarity measures such as Nilsimsa are also required.

2.1.4.5 Hybrid Similarity Scoring

To capture both command structure and behavioral meaning, structural and semantic similarity are combined:

$$S_{total} = \alpha S_{structural} + \beta S_{semantic} \quad (2.7)$$

where $S_{structural}$ is the structural similarity score, $S_{semantic}$ is the semantic similarity score, and α and β are weighting parameters.

2.1.4.6 Fuzzy Command Overlap

Fuzzy command overlap measures the proportion of shared commands between two sessions while allowing differences in command order and additional commands. Let (A) and (B) denote the sets of distinct normalized commands contained in two sessions. Their overlap is calculated as

$$\text{Overlap}(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)}. \quad (2.8)$$

The denominator uses the size of the smaller command set to measure how much of the smaller session is contained within the larger one. A score of (1) indicates that all commands in the smaller session also appear in the larger session, whereas lower values indicate less shared behavior. Unlike exact sequence matching, this measure tolerates command reordering and minor additions or omissions. In this thesis, sessions with an overlap score of at least (0.70) are considered to exhibit substantial behavioral overlap.

2.1.5 Attacker Similarity Graph and Connected Components

An **attacker similarity graph** is used to represent relationships between attackers.

In this graph:

- nodes represent attackers,
- edges represent strong similarity between attacker profiles.

The graph is undirected because similarity is treated as a mutual relationship.

In this thesis, an edge is created when the similarity score is greater than 85%.

This threshold is used to keep only strong behavioral relationships and reduce noisy connections [12].

2.1.5.1 Connected Components

A **connected component** is a group of nodes where each node can be reached from the others through graph edges.

In this thesis, connected components represent groups of attackers with similar behavior. These groups are used to study coordinated activity and campaign-level behavior.

2.1.6 Security Frameworks and Threat Characterization

Security frameworks are used to describe attacker behavior at a higher level.

2.1.6.1 MITRE ATT&CK

MITRE ATT&CK is a knowledge base that describes attacker tactics and techniques [13].

In this thesis, commands are mapped to ATT&CK tactics using rule-based logic. A **tactic sequence** is the ordered list of tactics observed in a session.

This helps describe attacker intent instead of only listing raw commands.

2.1.6.2 Cyber Kill Chain

The Cyber Kill Chain describes the main stages of an attack [14].

In this thesis, the Kill Chain provides a high-level view of attack progression, while

MITRE ATT&CK is used for more detailed tactic mapping.

2.1.6.3 Campaign-Level Attribution

Attackers may use multiple IP addresses to hide their identity or distribute activity. This behavior is related to Sybil attacks [15].

Identity aliasing refers to grouping different visible identities based on similar behavior.

A **command-and-control (C2)** infrastructure refers to systems used to control or coordinate attacks.

An **attack campaign** is a group of sessions or attackers that show similar behavior, shared tactics, or common infrastructure.

These concepts help move the analysis from single sessions to campaign-level understanding.

2.1.7 Unsupervised Learning and Clustering Evaluation

Honeypot data usually does not include ground-truth labels for bot and human attackers. Because of this, fully supervised evaluation is difficult.

This thesis uses a semi-supervised evaluation approach. Clustering is first used to group similar behavior. Then, representative samples are checked to support evaluation.

2.1.7.1 K-Means Clustering

K-Means is an unsupervised clustering method that separates data into K groups based on similarity.

In this thesis, $K = 2$ is used to separate bot-like and human-like behavior.

These clusters are not treated as perfect labels. Instead, they provide a reference point for evaluating the consistency of the proposed rule-based detection method.

2.2 Related Work

This section reviews previous work related to honeypot data, attacker behavior, and bot detection. The goal is to understand how researchers collect and analyze attack data, what methods they use, and what is still missing. This also helps explain why a unified behavioral framework is needed.

2.2.1 Honeypot-Based Cyber Attack Data Collection

Honeypots are widely used to observe attackers in a controlled environment. They allow researchers to collect real attack data such as login attempts, commands, and interaction patterns. Early work showed that honeypots are useful for studying attacks, while later work improved their realism and data quality [9, 5, 4].

Dong et al. [16] analyze SSH attacks using data collected from multiple Cowrie honeypots over time. Their study examines attacker commands, login attempts, source IPs, and session activities using large-scale honeypot logs and visual analysis techniques. The results reveal common attack patterns and differences between automated and human attackers.

Munteanu et al. [2] analyze SSH attack activity using data collected from a large-scale honeynet over several years. Their study shows that attackers are becoming more exploratory and often perform reconnaissance before launching attacks. The authors also identify different bot families and analyze attacker behavior using command

sequence clustering.

Haikal et al. [17] study large-scale attack activity using a distributed honeypot system called HoneyTrap. Their analysis shows that most attacks target web services and SSH, with many brute-force login attempts and coordinated scanning behavior. The study also highlights how attackers use adaptive and organized attack strategies across different network services.

Overall, these studies show that honeypots are effective for collecting real attack data and identifying large-scale patterns. However, most existing work focuses on statistics and trends rather than modeling attacker behavior across sessions or building detailed behavioral profiles.

2.2.2 Botnets and Automated Attack Ecosystems

Many modern attacks are automated. Botnets, scripts, and malware are used to scan systems, try passwords, and execute commands without human interaction. These attacks often follow repeated and structured patterns.

Kepner et al. [1] study botnets as coordinated networks of infected devices controlled by attackers. They introduce an “observe–pursue–counter” approach for monitoring, tracking, and defending against botnet activity. Their results show that botnets generate repeated and coordinated network behavior that can be detected through large-scale traffic analysis.

Almazarqi et al. [18] propose BotPro, a data-driven framework for tracking and profiling IoT botnets at a global scale. Their approach combines honeypot data with threat intelligence, DNS, and routing information to analyze how botnets spread and operate across networks over time.

These studies show that botnets generate structured and repeated attack patterns

that can be observed at scale. However, most existing work focuses on network-level analysis and global patterns, and does not examine the detailed command-level behavior needed for fine-grained attacker modeling.

2.2.3 Preprocessing and Session Modeling

Raw honeypot logs are noisy and hard to analyze directly. They often include failed login attempts, repeated events, and incomplete sessions. This makes it difficult to understand attacker behavior without preprocessing.

Log parsing methods such as Drain [19] and Spell [20] group similar log messages and convert them into structured templates. These methods separate constant parts of logs from variable values and organize them into meaningful groups, which helps reduce noise and makes analysis easier. For example, Drain processes logs in a streaming way and uses a fixed-depth tree to efficiently group similar messages [19]. However, these methods are mainly designed for system and application logs, and they are not well suited for capturing attacker command behavior in honeypot environments.

Karaarslan et al. [21] propose an AI-based approach for analyzing Cowrie honeypot logs using autonomous agents. Their system automatically processes and summarizes attacker sessions, analyzes command behavior, and generates reports to help researchers understand attack patterns more efficiently.

As a whole, preprocessing helps organize large amounts of data, but it can also remove important behavioral details. In honeypot analysis, it is important to preserve the meaning and order of attacker commands.

2.2.4 Command Abstraction and Representation in Attack Analysis

Attack commands usually contain both fixed and changing parts. The fixed parts are the command names and core structure, while the changing parts include values such as IP addresses, file names, and URLs.

Hussain et al. [22] propose a language model approach to learn the structure of command-line commands. They train a BERT-like model using a masking technique to predict missing tokens in commands. By analyzing these predictions, the model can separate constant parts from variable values and group similar commands using clustering.

Huang et al. [23] propose CmdCaliper, a command embedding model designed for cybersecurity tasks. Unlike general text models, it focuses on the meaning of commands rather than just their structure. Their results show better performance than existing embedding models in tasks such as malicious command detection and command similarity.

Zhou et al. [24] use large language models and BERT for log parsing and anomaly detection. Their approach extracts structured templates from raw logs while preserving semantic and contextual information.

These studies show that learning-based methods can model command structure and meaning. However, they are not designed for noisy honeypot data, where attackers often change parameters and behavior across sessions.

2.2.5 behavioral Similarity and Pattern Matching

Similarity methods are used to find repeated attack behavior. These methods compare logs, commands, or sessions to detect patterns and group similar activities.

Flynn and Olukoya [25] use approximate matching and fuzzy hashing techniques to detect malicious activity in log data. Their method creates compact representations of logs so that similar attack patterns can still be matched even when logs slightly differ.

Othman et al. [26] use sentence transformer models to measure semantic similarity between attack descriptions and vulnerabilities. Their approach maps attacks and CVE reports into a shared embedding space using cosine similarity.

Wang et al. [8] analyze large-scale honeypot traffic and show that bot behavior often follows repeated payload patterns. Their work uses these repeated patterns to help explain automated bot activity.

In general, these studies show that similarity is important for detecting repeated attack behavior. However, most methods focus on only one type of similarity, such as structure or meaning, and do not combine both.

2.2.6 Graph-Based Threat Intelligence and Campaign Attribution

Graphs are useful for modeling relationships between attackers, infrastructure, and attack activities. They can reveal hidden connections that are difficult to observe in raw data.

Kurniawan et al. [27] propose CyKG-RAG, a knowledge graph-enhanced retrieval augmented generation framework for cybersecurity analysis. Their approach com-

bines large language models with cybersecurity knowledge graphs and vector similarity search to improve threat detection and cybersecurity analysis.

Maturi [28] proposes a graph-based framework for integrating synthetic honeypot logs with structured threat intelligence. Their approach transforms honeypot data into STIX-compliant objects and stores them in a Neo4j graph database for attack visualization and relationship analysis.

Xiang et al. [7] propose IPAttributor, a cyber attacker attribution framework that combines intrusion logs with threat intelligence data to identify related attacker groups using similarity analysis and clustering.

Orsini and Han [29] propose DYNAMO, a weakly supervised framework for network attack campaign attribution using self-supervised learning and density-aware active learning. Their approach reduces labeling overhead while improving attribution accuracy and unseen campaign detection on imbalanced network traffic datasets.

These studies show that graph-based methods and attribution frameworks can help identify relationships between attacks and attacker groups. However, many approaches depend on external threat intelligence or network-level traffic data rather than directly modeling attacker behavior from raw command interactions.

2.2.7 MITRE ATT&CK and behavioral Semantics

Frameworks such as MITRE ATT&CK describe attacker behavior at a higher level and help explain how attacks progress.

Al-Sada et al. [30] review how the MITRE ATT&CK framework is used in cybersecurity research and practice. They show that attacker behavior follows sequences of tactics, techniques, and procedures (TTPs) that can help identify and compare attack behavior.

Rahman et al. [31] propose ALERT, a framework that uses large language models to extract attack techniques from cyber threat intelligence reports and map them to the MITRE ATT&CK framework. Their approach uses active learning to reduce the need for large labeled datasets while maintaining strong performance.

Overall, these studies show that ATT&CK is useful for understanding attacker goals and high-level behavior. However, most existing work focuses on reports and alerts rather than raw honeypot interactions.

2.2.8 Rule-Based and Learning-Based Bot Detection

Bot detection is often performed using machine learning and graph-based methods. These approaches can achieve high detection accuracy, but they usually depend on labeled datasets or network-level traffic features.

Lo et al. [32] propose XG-BoT, a graph neural network-based framework for botnet detection and network forensics. Their approach models network traffic as communication graphs and uses a Graph Isomorphism Network (GIN) to detect malicious botnet nodes.

Qian et al. [33] propose a distributed IoT botnet detection framework based on graph partitioning and graph convolutional networks. Their approach uses Cluster-GCN and distributed training to improve botnet detection efficiency on large communication graphs.

Mahboubi et al. [34] present a survey of modern botnet detection techniques, including AI-driven, graph-based, and behavioral methods. The study also discusses challenges such as encrypted traffic, adversarial evasion, and dataset limitations.

Kotenko et al. [35] propose a machine learning-based framework for cyber attacker profiling in risk analysis. Their approach uses network traffic, event logs, and bash

command histories to distinguish attacker teams based on behavioral patterns.

These studies show that learning-based approaches are effective for bot detection and attacker profiling. However, many methods require labeled data and mainly focus on network-level features rather than detailed behavioral modeling from honeypot command interactions.

2.2.9 Comparative Summary of Related Work

Table 2.1 summarizes the reviewed studies according to the problem addressed, the main approach, and the remaining gap relative to this thesis. The comparison shows that previous studies provide important solutions for individual stages of honeypot analysis, but do not integrate command normalization, session-level behavioral modeling, bot detection, attacker profiling, and coordinated-group discovery within a unified framework.

2.2.10 Comparative Summary of Related Work

Table 2.1 summarizes the main contribution and remaining gap of each reviewed study. The comparison shows that existing work addresses individual components of honeypot and attacker analysis, but does not integrate them into a unified command-level behavioral framework.

Table 2.1: Summary of related-work contributions and remaining gaps.

Study	Contribution	Gap
Honeypot-Based Cyber Attack Data Collection		
Provos [9]	Introduced practical virtual honeypots.	No command-level behavioral modeling.
Yang et al. [5]	Improved honeypot realism and data quality.	Focuses mainly on data collection.

Continued on the next page

Table 2.1 – continued

Study	Contribution	Gap
Javadpour et al. [4]	Reviewed honeypot designs and applications.	No implemented analysis framework.
Dong et al. [16]	Analyzed long-term Cowrie attack activity.	Limited cross-session attacker profiling.
Munteanu et al. [2]	Clustered command sequences and bot families.	No unified behavioral pipeline.
Haikal et al. [17]	Characterized distributed and coordinated attacks.	Limited post-compromise command analysis.
Botnets and Automated Attack Ecosystems		
Kepner et al. [1]	Modeled coordinated botnet behavior at scale.	Uses network rather than command behavior.
Almazarqi et al. [18]	Combined honeypots and external intelligence for botnet profiling.	Focuses on infrastructure-level relationships.
Preprocessing and Session Modeling		
He et al. [19]	Introduced efficient tree-based log parsing.	Does not preserve session behavior or command order.
Du and Li [20]	Developed online log-template extraction.	Not designed for honeypot command sequences.
Karaarslan et al. [21]	Automated Cowrie session analysis and reporting.	Lacks explicit normalization and reduction.
Command Abstraction and Representation		
Hussain et al. [22]	Learned stable and variable command elements.	Not tailored to multi-session honeypot data.
Huang et al. [23]	Developed cybersecurity-specific command embeddings.	Does not preserve behavioral provenance.
Zhou et al. [24]	Used language models for semantic log parsing.	Targets general logs rather than attacker sessions.
Behavioral Similarity and Pattern Matching		
Flynn and Olukoya [25]	Applied fuzzy matching to malicious logs.	Does not capture behavioral semantics.
Othman et al. [26]	Used embeddings for semantic attack comparison.	Analyzes descriptions rather than raw commands.

Continued on the next page

Table 2.1 – continued

Study	Contribution	Gap
Wang et al. [8]	Identified repeated bot payload patterns.	Does not combine multiple similarity methods.
Graph-Based Threat Intelligence and Attribution		
Kurniawan et al. [27]	Combined knowledge graphs, LLMs, and retrieval.	Relies on existing threat knowledge.
Maturi [28]	Represented honeypot intelligence using STIX and Neo4j.	Focuses on integration and visualization.
Xiang et al. [7]	Combined logs and threat intelligence for attribution.	Depends on external intelligence sources.
Orsini and Han [29]	Reduced labeling needs for campaign attribution.	Uses network traffic rather than commands.
MITRE ATT&CK and Behavioral Semantics		
Al-Sada et al. [30]	Reviewed ATT&CK-based behavioral analysis.	Does not map raw honeypot commands.
Rahman et al. [31]	Automated ATT&CK mapping from reports.	Processes reports rather than interactions.
Bot Detection and Attacker Profiling		
Lo et al. [32]	Used graph neural networks for botnet detection.	Relies on communication-graph features.
Qian et al. [33]	Developed scalable distributed botnet detection.	Does not analyze command-level behavior.
Mahboubi et al. [34]	Surveyed modern botnet-detection techniques.	Provides no implemented honeypot framework.
Kotenko et al. [35]	Profiled attackers using multiple behavioral sources.	Does not cover the full analysis pipeline.
This thesis	Integrates preprocessing, command normalization, similarity analysis, bot detection, profiling, and coordinated-group discovery.	Addresses SSH honeypot command behavior; external validation remains future work.

2.2.11 Summary and Positioning of This Work

The reviewed studies show that honeypots can collect valuable attack data, botnets create repeated patterns, and similarity and graph methods can help analyze behavior. However, most studies focus on only one part of the problem.

There is still no single framework that connects all stages, from raw honeypot logs to session reconstruction, command normalization, behavioral similarity analysis, bot detection, attacker profiling, and coordinated campaign discovery.

This thesis addresses this gap by proposing a unified behavioral framework for SSH honeypot telemetry analysis. The framework reconstructs sessions, normalizes commands, detects bots using behavioral patterns, builds attacker profiles, and identifies coordinated attacker groups based on behavioral similarity.

Chapter 3

Methodology

This chapter presents the methodological design of the proposed Bot Detection and Profiling Framework, which transforms large-scale Cowrie honeypot telemetry into structured attacker profiles for further analysis. The theoretical foundations of this framework are introduced in Chapter 2.

The proposed methodology integrates key concepts such as behavioral tracing, command normalization, and attacker-centric profiling in a unified analytical framework. The main objective of this chapter is to transform raw, high-entropy honeypot logs into structured intelligence that is suitable for bot detection, attacker profiling, and campaign-level analysis. In this study, honeypot events are not treated as isolated occurrences but as sequential behavioral activities of attackers that are analyzed in relation to prior actions across multiple commands, sessions, and network identities.

To achieve this goal, the proposed framework consists of three primary modules. The first module focuses on data collection and preprocessing, where raw Cowrie telemetry is collected from the honeypot and then filtered, reconstructed, and transformed into ordered behavioral traces that are suitable for subsequent analysis.

The second module performs command normalization by constructing a Heteroge-

neous Cyber-Knowledge Graph (H-CKG). This graph is used to distinguish stable command syntax from variable components, which enables the masking of non-structural tokens within command sequences. The sessions are then reduced in a rule-based session grouping stage and prepared for further analysis.

The third module generates attacker-level profiles from the reduced session representations using extracted features, bot detection labels, and attack tactic sequences that match MITRE ATT&CK tactics.

Moreover, the pipeline supports higher-level attribution analysis. By examining attacker profiles in a shared behavioral space, the framework enables similarity-based grouping and community-level analysis that reveal coordinated attack campaigns and Sybil-style activity that is not detectable through conventional IP-based analysis.

Figure 3.1 illustrates the overall structure of the pipeline and shows how data flows from raw telemetry collection to attacker profiles.

The following sections describe each module in detail. Data collection and preprocessing are presented in Section 3.1. Command normalization and session reduction are discussed in Section 3.2. Feature extraction, the rule-based bot detection engine, and attacker profile construction are described in Section 3.3. The evaluation of the system is presented in Chapter 4.

3.1 Module 1: Data Collection and Preprocessing

This module transforms low-level event logs into structured session-level traces. Since the quality and completeness of telemetry are critical for accurately capturing attacker behavior, this module focuses on constructing coherent, noise-reduced, and temporally ordered representations of attacker interactions. It consists of two main stages:

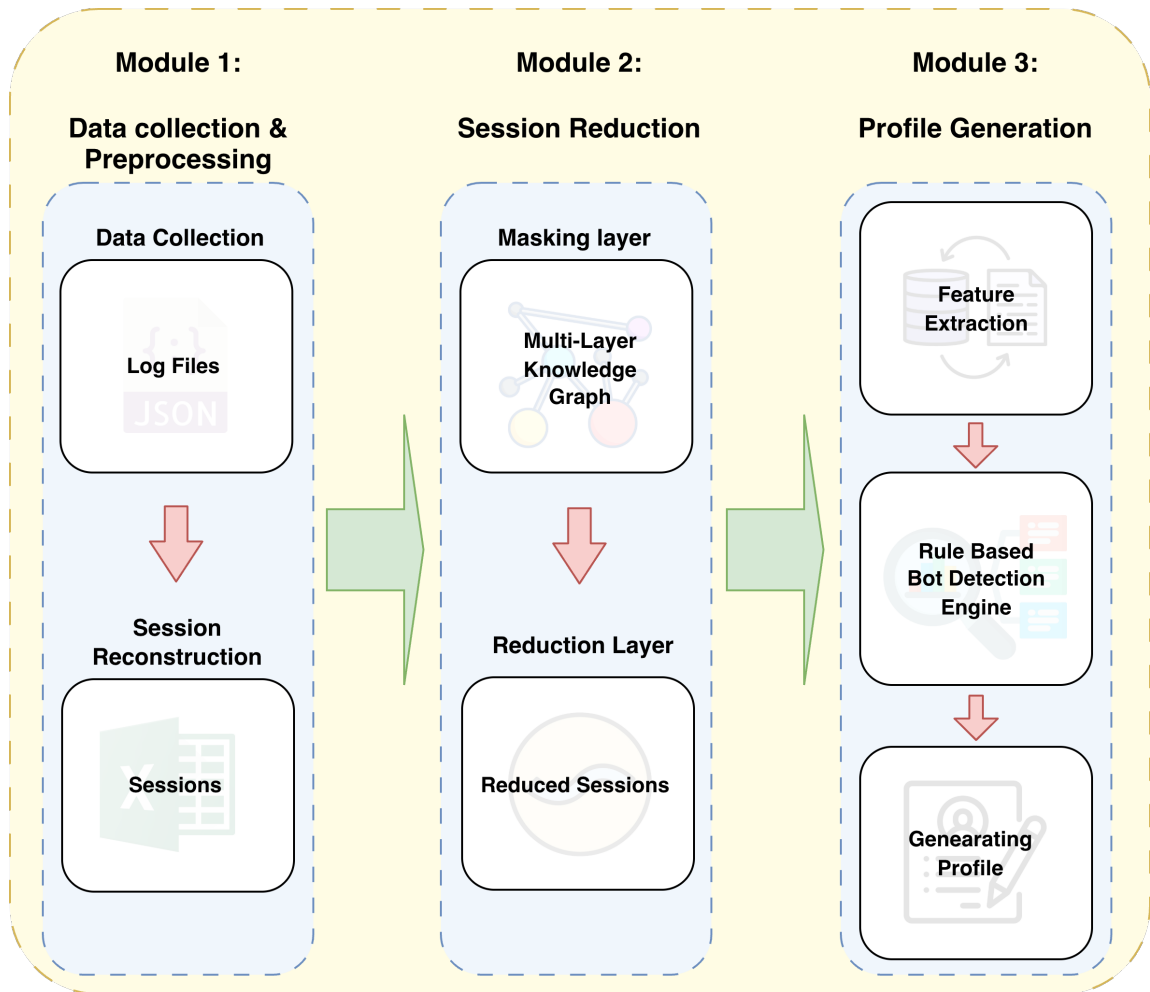


Figure 3.1: High-level architecture of the Bot Detection and Profiling Framework, illustrating the data flow from raw ingestion to final attacker classification.

- **Stage 1:** Dataset collection from the CIC Cowrie honeypot logs.
- **Stage 2:** Session reconstruction through event filtering, session grouping, and temporal sequence preparation.

These stages are described in detail in the following subsections.

3.1.1 Stage 1: Dataset Collection

This research uses a large-scale dataset collected from the Canadian Institute for Cybersecurity (CIC) Cowrie honeypot at the University of New Brunswick [36]. Cowrie is a medium-interaction honeypot that records both authentication activity and post-authentication attacker behavior. After a successful login, it captures commands entered by attackers and their interactions with the simulated file system in a controlled environment.

The dataset includes eleven months of log files from January to November 2025 and consists of 288 JSON log files containing 8,702,601 recorded events. These logs initially produced 959,346 raw sessions. After filtering non-behavioral and empty sessions, the dataset was reduced to 20,220 interactive command-level sessions associated with 4,896 unique attackers.

3.1.2 Stage 2: Session Reconstruction

After the data collection stage, a preprocessing pipeline is applied in order to construct coherent attacker sessions. This pipeline transforms raw event streams into structured behavioral traces using three sequential stages: (1) event filtering and noise reduction, (2) session reconstruction, and (3) temporal ordering and sequence

preparation. These steps ensure that only meaningful attacker interactions are preserved and organized into a suitable format for downstream analysis.

3.1.2.1 Event Filtering and Noise Reduction

Raw honeypot logs contain various low-level events that are not useful for reflecting attacker intent. These include authentication attempts, protocol maintenance messages, and empty connections. Such events introduce noise and disrupt the continuity of command-level behavioral analysis.

To focus on post-compromise activity, a strict filtering strategy is applied to keep events corresponding to executed commands. Therefore, only events with the identifier `cowrie.command.input` are preserved, as they represent successful command execution and are useful for behavioral and session-level analysis. For this purpose, the following categories of events are removed:

- **Failed Authentication Attempts:** These events capture pre-compromise activity, such as brute-force login attempts. Although they are meaningful for analyzing authentication attacks, they do not represent post-compromise command execution or attacker behavior and are therefore excluded from this study.
- **Keep-Alive and Protocol Maintenance Events:** These are generated to maintain network connectivity and do not carry any adversarial intent.
- **Empty Sessions:** Automated scanners may establish connections without executing any commands. These sessions contain no behavioral information and are removed to reduce noise and dataset dimensionality.

This filtering step significantly reduces noise while preserving the behavioral signals necessary for accurate session reconstruction.

3.1.2.2 Session Grouping and Trace Construction

After filtering, the remaining events are grouped based on their unique session identifiers in order to reconstruct individual attacker interactions. However, due to the asynchronous nature of logging and concurrent activity, a single logical session may be fragmented across multiple log entries.

Session reconstruction combines these fragmented events into coherent interaction traces, so that each session shows a continuous sequence of attacker actions. This step enables the analysis of attacker behavior at the session level rather than at the level of isolated events.

3.1.2.3 Temporal Ordering and Sequence Preparation

Once sessions are reconstructed, commands within each session are ordered chronologically using millisecond-resolution timestamps provided by the Cowrie logging engine. This step ensures that the execution sequence and timing relationships between commands are preserved.

To standardize session representations, explicit boundary markers are also inserted to denote the beginning and end of each session. These markers enable consistent computation of temporal features such as time-to-first-command and session duration.

In addition, compound commands that contain shell operators, such as `;`, `&&`, and `|`, are kept as complete command strings during preprocessing. Instead of splitting these commands into tokens immediately, tokenization is delayed until the graph is constructed in Module 2. At that stage, each token is still connected to the command it came from, the session in which the command appeared, and the attacker associated with that session. This allows the framework to examine how tokens are

reused and to normalize variable parts of commands without losing their origin. The complete order and structure of the compound command are still available in the original raw command string. However, the current graph does not directly represent the position of each token or the execution relationships between the linked subcommands.

Table 3.1 summarizes the key preprocessing steps applied during the session reconstruction stage and their effects on the data.

Table 3.1: Summary of Module 1 stages and their effects on the data.

Stage	Step	Operation	Effect on Data
Stage 1	Dataset Collection	Collection of CIC Cowrie honeypot JSON log files from January to November 2025.	Provides the raw attacker telemetry used as input for preprocessing and session reconstruction.
Stage 2	Event Filtering and Noise Reduction	Retention of <code>cowrie.command.input</code> events only.	Removes pre-compromise and non-behavioral events while preserving post-compromise command-level activity.
	Session Grouping and Trace Construction	Grouping events by session ID.	Merges fragmented events into coherent attacker interaction traces representing continuous behavioral sequences.
	Temporal Ordering and Sequence Preparation	Chronological sorting of events and injection of sequence boundaries.	Preserves command execution order and enables temporal feature computation.

3.2 Module 2: Session Reduction and Command Normalization

Two primary characteristics of botnet-based attacks that make behavioral analysis challenging are variability and redundancy. Variability refers to the randomization of command arguments, such as file names, IP addresses, URLs, or passwords. These arguments are frequently modified by automated malware scripts in order to evade signature-based detection systems. As a result, commands that reflect the same attack behavior might appear syntactically different.

On the other hand, redundancy occurs when identical or highly similar attack scripts are executed repeatedly across multiple sessions. These repeated command sequences do not provide additional analytical value, but they increase the size of the dataset and introduce unnecessary repetition and noise.

Module 2 addresses both variability and redundancy in two stages, which are explained in detail in this section:

- **Stage 1:** Dynamic command masking by a Heterogeneous Cyber-Knowledge Graph (H-CKG).
- **Stage 2:** Hierarchical session similarity analysis and reduction through rule-based session grouping.

These two stages are described in detail in the following subsections.

3.2.1 Stage 1: Dynamic Command Masking via a Heterogeneous Cyber-Knowledge Graph (H-CKG)

Differentiating structural command components from variable elements is challenging in large-scale datasets because of high variability in command syntax; however, this distinction plays an important role in removing irrelevant variations while preserving the underlying behavioral patterns. Rule-based masking approaches that rely on manually designed patterns are limited and do not generalize well to evolving attack behaviors.

To address this, the proposed framework adopts a data-driven masking strategy that automatically distinguishes structural command components from variable components based on observed attacker behavior across sessions. In this approach, a four-layer directed knowledge graph, referred to as the Heterogeneous Cyber-Knowledge

Graph (H-CKG), is constructed to model the relationships between attackers, sessions, commands, and tokens. This graph captures token reuse patterns across the dataset and is used as the basis for command normalization. In this framework, masking is performed by replacing high-entropy variable tokens with generic placeholders while preserving stable structural tokens that define the underlying command logic.

3.2.1.1 Graph-Based Token Classification and Command Normalization

The Heterogeneous Cyber-Knowledge Graph (H-CKG) is organized hierarchically into four layers:

- **Attacker Layer:** Each unique source IP address is represented as an attacker node at the first level.
- **Session Layer:** Each session associated with an attacker is represented as a distinct node, preserving temporal boundaries.
- **Command Layer:** Each unique raw command string is represented by a single command node. Repeated executions of the same command do not create duplicate command nodes; instead, their occurrences are preserved through the relationships between session and command nodes. If the same command is executed multiple times within one session, the corresponding Session-to-Command edge records the number of executions. Therefore, node deduplication reduces redundant storage without discarding command-frequency information.
- **Token Layer:** Commands are decomposed into whitespace-separated atomic tokens in the fourth layer.

Preservation of Command Frequency. The graph distinguishes between a unique command and its execution occurrences. A command node stores the command string only once, while its repeated use is represented through Session-to-Command relationships. For example, if the same command is executed in 100 different sessions, the single command node is connected to those 100 session nodes. If it is executed repeatedly within the same session, the execution count is stored as a weight on the corresponding edge. Consequently, the graph preserves command reuse while avoiding duplicate storage of identical command strings.

In this graph, directed edges capture the relationships between layers (Attacker $\rightarrow$ Session $\rightarrow$ Command $\rightarrow$ Token), which allows each token to be traced back to its originating attacker and also enables analysis of token usage across the entire dataset. Figure 3.2 depicts a sample instance of this graph structure.

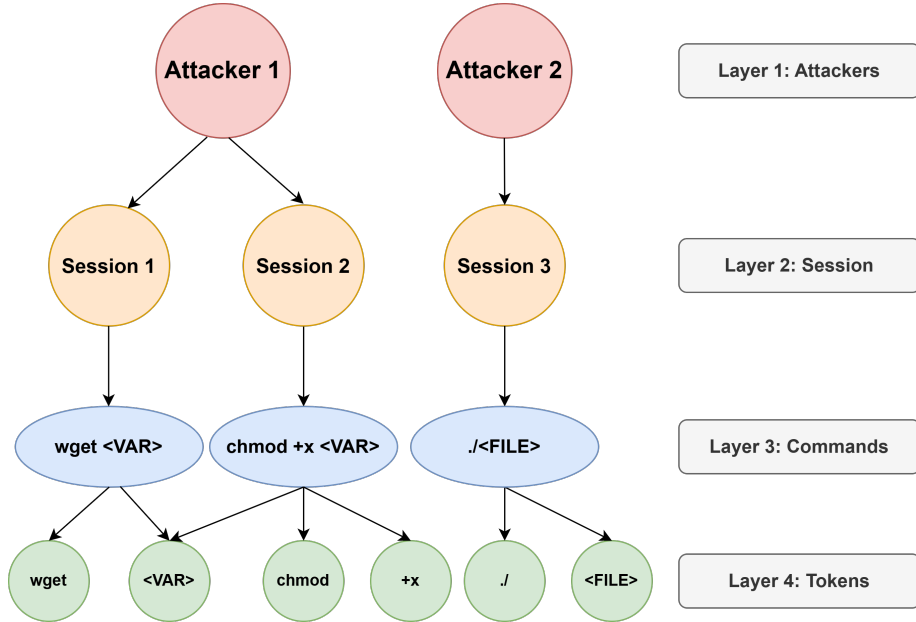


Figure 3.2: Overview of the 4-layer Heterogeneous Cyber-Knowledge Graph (HCKG)

Normalization Process. For each raw command, the command string is first divided into an ordered sequence of whitespace-separated tokens. For example, the

command `wget http://example.com/a.sh -O /tmp/a.sh` is represented by the tokens `wget`, `http://example.com/a.sh`, `-O`, and `/tmp/a.sh`. The command and its tokens are then added to the H-CKG through Command-to-Token edges while retaining their relationships with the originating session and attacker. For each token, the in-degree is calculated as the number of distinct command nodes connected to that token. The resulting value is then evaluated using the masking threshold and the hybrid safety filter. Frequently reused structural tokens, such as `wget` and `-O`, are retained, whereas low-reuse variable tokens, such as the URL and file path, are replaced with the generic placeholder `<MASK>`. The normalized command therefore becomes `wget <MASK> -O <MASK>`. Although a frequency dictionary could count the number of distinct commands containing each token, the graph additionally preserves the relationships among attackers, sessions, commands, and tokens. This allows each token and its masking decision to be traced back to its original command and behavioral context.

Rationale for the Graph Representation. A simple frequency dictionary could be used to count the occurrences of each token and apply a frequency-based masking threshold. However, such a representation would retain only the token and its frequency, without preserving the contexts in which the token appears. In contrast, the H-CKG connects each token to its associated commands, sessions, and attackers through the hierarchical paths of the graph. This structure makes it possible to trace a token back to its original behavioral context, examine whether it is reused by one or multiple attackers and sessions, and analyze relationships such as token co-occurrence within commands. Therefore, token in-degree is one operation performed over the graph, while the main purpose of the H-CKG is to preserve the multilevel relationships and provenance that would be lost in a simple frequency dictionary.

Token Variability Analysis. After constructing the graph, variability is evaluated at the token layer (Layer 4) by measuring the in-degree of each token. In this context, the in-degree of a token is the number of distinct command nodes connected to that token. Thus, token in-degree represents reuse across distinct command structures rather than the total number of command executions. Repeated executions of the same command are preserved separately through the Session-to-Command relationships and their occurrence counts.

Tokens with high in-degree values are considered structural components of command syntax, as they are consistently reused across multiple commands. In contrast, tokens with low in-degree values are treated as variable elements, such as IP addresses, file names, URLs, or randomly generated identifiers, which are intentionally varied by automated scripts.

Based on this observation, tokens are classified into two categories: structural tokens and variable tokens. Variable tokens are replaced with generic placeholders to reduce syntactic variability while keeping the underlying command structure.

In this approach, selecting an appropriate threshold is important to balance over-masking and under-masking. In this study, a threshold of five is selected empirically based on the observed token reuse distribution in the dataset. Tokens appearing fewer than five times are generally associated with high-variability elements, such as IP addresses, URLs, file names, and randomly generated identifiers. In contrast, tokens appearing more frequently are more likely to represent stable command syntax. A lower threshold may retain variable tokens, while a higher threshold may incorrectly mask legitimate structural components. Therefore, the threshold of five is treated as an empirical normalization parameter that provides a practical balance for this dataset.

3.2.1.2 Hybrid Safety Filtering

In order to reduce the risk of misclassification in the previous step, a Hybrid Safety Filter is applied as a post-processing step after token variability analysis. This step addresses cases where certain variable tokens might appear frequently and be incorrectly classified as structural tokens based only on their in-degree.

Therefore, tokens are examined based on their syntactic structure as well. If a token matches the format of an IP address, it is masked regardless of its frequency. For example, in a command such as `wget http://185.234.217.12/bins.sh`, the token `185.234.217.12` might appear frequently across sessions and therefore have a high in-degree value. However, since it is a variable infrastructure element rather than a structural command component, it is masked by the hybrid filter.

This additional filtering step ensures that frequently reused but semantically variable tokens are not mistakenly kept, which improves the robustness and accuracy of the normalization process.

3.2.2 Stage 2: Hierarchical session similarity analysis and reduction through rule-based session grouping.

After the masking process, each session is transformed into a normalized sequence of commands in which variable tokens are removed and structural tokens are preserved. This normalization produces comparable data that is suitable for the subsequent grouping and reduction stages. The goal of this module is to reduce redundant sessions by merging similar sessions into representative behavioral patterns while preserving important behaviors required for further analysis.

3.2.2.1 Sequence-Based Session Grouping and Complexity Reduction

This grouping stage is performed at the attacker level, where all sessions associated with a single attacker are analyzed together. This enables the framework to capture the overall behavioral patterns and attack activities of each attacker.

A naive approach would compare every session with all other sessions, which leads to high computational cost. Given N sessions, this results in $\frac{N(N-1)}{2}$ pairwise comparisons, corresponding to a complexity of $O(N^2)$, which becomes impractical for large datasets.

To address this limitation, the framework employs a **sequence-based grouping strategy** as part of the rule-based session grouping process. The normalized command sequences within each session are combined to form a unique *behavioral signature*, which represents the execution pattern of the session.

Instead of comparing sessions pairwise, each signature is used as a key in a lookup structure (e.g., a dictionary or hash map). Sessions that share identical normalized command sequences are directly assigned to the same group. In other words, grouping is performed by matching signatures rather than comparing sessions.

This process allows all sessions to be processed in a single pass without requiring explicit comparisons between sessions. As a result, the initial grouping step operates in linear time, $O(N)$, and produces a set of unique behavioral signatures that represent distinct execution patterns.

Let U denote the number of unique behavioral signatures that are obtained after grouping. In practice, due to repeated automated executions in botnet activity, many sessions share identical behavior, resulting in $U \ll N$.

After this exact grouping step, similarity-based analysis is applied only to these unique signatures in order to identify highly similar or overlapping behavioral pat-

terns. Similarity between behavioral signatures is computed using a hybrid approach that combines Nilsimsa-based fuzzy hashing and sequence-based matching. As a result, the number of comparisons is reduced from $\frac{N(N-1)}{2}$ to $\frac{U(U-1)}{2}$, corresponding to a complexity of $O(U^2)$. This significantly improves scalability while preserving distinct behavioral patterns.

3.2.2.2 Rule-Based Session Reduction

After behavioral signatures are generated, a rule-based reduction stage is applied at the attacker level to remove redundant or overlapping sessions while preserving representative attack behavior. The reduction process is performed in two main stages.

First, Rules A and B evaluate the original sessions and their behavioral-signature memberships. These rules determine whether a session should be preserved directly, represented by another session, or removed because it is an identical or highly similar repetition. Exact-signature groups retain information about how many original sessions they represent.

After these initial decisions, Rule C is applied between the remaining distinct signature groups belonging to the same attacker. Although these groups do not have identical signatures, their representative sessions may still exhibit sequence containment or behavioral overlap. Rule C therefore determines whether such groups can be consolidated into a broader representative behavioral pattern. If no reduction rule is satisfied, Rule D preserves the remaining sessions as distinct behaviors.

Rule A: Single-session attacker. If an attacker has only one session in the dataset, the session is preserved, and the attacker is labeled as *non-periodic*. No comparison or reduction is required.

Rule B: Repetition and similarity reduction. For attackers with multiple sessions, the framework first examines the behavioral signatures of the original sessions to identify repeated or highly similar execution patterns.

- **Rule B1: Masked identical sessions.** If multiple sessions associated with an attacker have the same behavioral signature, they contain identical masked command sequences in the same order. These sessions are treated as repeated executions of the same pattern. One representative session is preserved, while the remaining duplicate sessions are removed. The original number of represented sessions is retained, and the attacker is labeled as *periodic* when the same pattern is repeatedly observed.
- **Rule B2: Near-duplicate signature groups.** If an attacker has multiple distinct behavioral signatures, the representative sequences of these signature groups are compared using a hybrid similarity score based on Nilsimsa and sequence matching. If all relevant signature pairs exceed the predefined similarity threshold, the groups are treated as variations of the same general behavioral pattern. The longest session is retained as the representative, and the attacker is labeled as *non-periodic* because the observed executions are similar but not exactly identical.

The similarity threshold is set to **85**. Therefore, two behavioral signatures are considered highly similar when their hybrid similarity score is equal to or greater than 85. This relatively high threshold was selected empirically based on the distribution of similarity scores in the dataset. It ensures that only sessions with clearly similar command structures are consolidated, while reducing the risk of incorrectly grouping unrelated behaviors. Similar but non-identical sessions may represent adaptive or semi-automated activity rather than strictly periodic execution.

This rule reflects that similar but non-identical activity may indicate adaptive or semi-automated behavior rather than strictly repeated periodic execution.

Rule C: Behavioral consolidation across remaining signature groups. Rule C is applied only when the repetition and near-duplicate conditions in Rule B do not fully reduce the attacker’s sessions. At this stage, the framework compares the representative sessions of the remaining distinct behavioral-signature groups.

Although these groups have different signatures, one behavioral sequence may still be contained within another or may cover substantially the same activity. The objective is therefore to preserve the session with the greatest behavioral coverage while removing incomplete or overlapping representations.

- **Rule C1: Strict sequence containment.** If the normalized command sequence of one representative session is a subsequence of another in the same order, the shorter session is considered an incomplete version of the longer behavior. The shorter session is removed, while the longer session is preserved.
- **Rule C2: Behavioral containment (order-insensitive).** If strict sequence containment is not satisfied, the framework checks whether the command set of one representative session is contained within another, regardless of command order. If so, the session with less behavioral coverage is removed, and the more complete session is retained.
- **Rule C3: Relaxed behavioral containment.** If neither strict sequence containment nor complete set containment is satisfied, the framework applies a fuzzy command-overlap measure. For two sessions (A) and (B), the overlap score is calculated as the number of shared distinct masked commands divided by the number of distinct commands in the smaller session:

$$\text{Overlap}(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)} \quad (3.1)$$

If the overlap score is at least (0.70), the sessions are considered behaviorally overlapping. The session with broader command coverage is retained as the representative, while the other session is removed.

Under Rule C, the objective is behavioral coverage rather than repetition frequency. Therefore, only the most representative session is retained, and the repetition counts of the removed signature groups are not combined with the retained session.

Rule D: No reduction applied. If none of the previous reduction conditions are satisfied, the remaining behavioral-signature groups are considered distinct. Their representative sessions are preserved for subsequent analysis.

The output of this module is a reduced dataset in which each session represents a distinct behavioral pattern rather than an individual execution instance. This reduction step plays an important role in decreasing redundancy introduced by automated scripts while preserving diverse behavioral patterns.

In summary, Module 2 transforms raw honeypot data into normalized behavioral representations that are suitable for downstream analysis. The table summarizes these operations and their effects on the dataset. The resulting dataset is compact, representative, and well-suited for further behavioral analysis.

Table 3.2: Summary of Module 2 stages and their effects on the data.

Stage	Step	Operation	Effect on Data
Stage 1	Graph-Based Token Classification and Command Normalization	Construct a four-layer Heterogeneous Cyber-Knowledge Graph and measure token in-degree to distinguish structural tokens from variable tokens.	Models attacker-session-command-token relationships and replaces variable command components, such as file names, URLs, and random identifiers, with generic placeholders.
	Hybrid Safety Filtering	Apply structure-based validation to mask IP-formatted tokens regardless of frequency.	Prevents frequently reused infrastructure elements from being incorrectly preserved as structural tokens.
Stage 2	Sequence-Based Session Grouping and Complexity Reduction	Convert normalized command sequences into behavioral signatures and group identical signatures using a lookup structure.	Groups identical behavioral signatures and reduces similarity computations from $O(N^2)$ to $O(U^2)$ while preserving distinct behavioral patterns.
	Rule-Based Session Reduction	Apply attacker-level rules for single-session preservation, repetition detection, containment, overlap analysis, and no-reduction cases.	Produces a reduced set of representative behavioral patterns while removing redundant, incomplete, and overlapping session executions.

3.3 Module 3: Bot Detection and Attacker Profiling

As discussed in Module 2, raw honeypot logs are transformed into normalized and comparable session representations, which serve as the input for further analysis in Module 3. This section explains how structured behavioral profiles of attackers are constructed from a combination of statistical feature extraction, a rule-based detection engine, and a lightweight MITRE-aligned tactic summarization. The module is organized into three main stages, which are described in detail in the following subsections.

- **Stage 1:** Feature Extraction
- **Stage 2:** Rule-Based Bot Detection Engine
- **Stage 3:** Attacker Profile Construction

3.3.1 Stage 1: Feature Extraction

To profile attackers based on their behavior, a set of behavioral, temporal, semantic, and structural features is extracted that captures interaction patterns, timing characteristics, command semantics, and structural similarities from all sessions. This stage forms the foundation of the detection and profiling process.

The extracted features are organized into four categories, each capturing a distinct aspect of attacker behavior:

(1) Interaction volume and session density. This category captures how many sessions an attacker performs and how dense each session is and includes two features.

- **Total sessions per attacker:** Total number of sessions related to each attacker. This feature provides attacker-level context about how often the attacker interacts with the honeypot.
- **Commands per session:** Number of commands executed within a session. This feature reflects how active a session is.

(2) Temporal dynamics and execution rhythm. This category captures the timing behavior of attacker activities. These features describe how fast commands are executed and how actions are distributed over time within a session. This category includes five features:

- **Session duration:** The time difference between the earliest and latest command timestamps recorded within a session.
- **Average inter-command time:** The average elapsed time between each pair of consecutive command timestamps within a session.

- **Average time per command:** Total session duration divided by the number of executed commands.
- **Burst rate:** Maximum number of commands executed within any one-second window, normalized by the total number of commands in the session.
- **Time to first command:** Time delay between the session start marker and the first executed command.

(3) **Command semantics and attack progression.** This category describes what the attacker is trying to do and how the attack progresses over time. Instead of analyzing commands individually, each session is represented based on the types of actions that are performed, such as reconnaissance, tool setup, persistence, or explicit malicious activity.

Each command is mapped to a semantic category using simple keyword-based rules. Based on this mapping, two features are extracted for each session:

- **Attack tactic sequence:** An ordered sequence of high-level tactic labels derived from the commands executed within a session. Compound commands containing shell operators, such as `;`, `&&`, and `||`, are first divided into individual segments. Each segment is then assigned one of the predefined tactic labels, including *Discovery*, *Execution*, *Persistence*, *Defense Evasion*, and *Command and Control*, using keyword-based rules. Segments that do not match any predefined tactic are excluded. The resulting sequence is compressed using run-length encoding (RLE), as defined in Section 2.1, by replacing consecutive repetitions with tactic-count pairs. For example,

[Discovery, Discovery, Execution, Execution, Persistence]

becomes

Table 3.3: Command categories and severity weights used for maliciousness scoring.

Command Category	Weight	Description
Reconnaissance / Discovery	1	Information gathering and environment exploration commands.
Tool Setup / Download	2	Commands used to download, install, or prepare malicious tools or payloads.
Persistence / Privilege Manipulation	2	Commands related to maintaining access or modifying permissions.
Execution / System Compromise	3	Commands directly involved in payload execution or compromise activities.
Benign / Uncategorized	0	Commands without clear malicious intent.

$[(\text{Discovery}, 2), (\text{Execution}, 2), (\text{Persistence}, 1)]$.

This reduces repetition while preserving the order and progression of attacker actions.

- **Maliciousness score:** A numerical value that reflects the overall level of malicious activity within a session. This score is computed using a predefined severity weighting scheme, where each command category is assigned a weight based on its role in the attack process. In this way, actions with stronger effects contribute more to the final score.

A discrete scale from 0 to 3 is used, where higher values correspond to actions that play a more critical role in compromise progression. For example, commands that are related to system compromise are assigned the highest weight, while lower-impact activities such as reconnaissance are assigned smaller weights. Benign or irrelevant commands receive a weight of zero.

The maliciousness score is calculated by summing the weights of all categorized commands within a session. Table 3.3 summarizes the command categories and their corresponding severity weights.

(4) behavioral repetition and similarity structure. Most automated attack campaigns repeat similar command patterns across multiple sessions. Individual

sessions are often short and may appear harmless on their own. However, repeated execution of similar behaviors, even with small changes, across many sessions is a strong sign of automation. To capture this pattern, the frequency and structural similarity of session command sequences are analyzed across the dataset.

Each normalized session is represented as a single string with masked variables and is compared with other sessions using the Nilsimsa fuzzy hashing scheme. Based on these comparisons, a similarity graph is constructed. Two sessions are connected if their similarity score exceeds a predefined threshold. Clusters are then defined as connected components in this graph. Based on this structure, the following features are extracted for each session:

- **Cluster ID:** Identifier of the session group assigned using Nilsimsa-based similarity analysis. Each normalized session is compared as a concatenated command sequence, and sessions with similarity scores greater than or equal to 85 are connected in a similarity graph. Connected sessions are assigned to the same cluster, while sessions with no similar counterparts are assigned a value of -1 .
- **Cluster size:** Number of sessions that belong to the same cluster as the given session.
- **Similarity degree:** Number of direct neighbors of the given session in the Nilsimsa similarity graph. A direct neighbor is a session whose Nilsimsa similarity score with the given session is greater than or equal to 85.
- **Maximum similarity:** Highest Nilsimsa similarity score observed between the given session and any other session.

Cluster size and similarity degree capture different aspects of similarity. Cluster size counts all sessions within the same cluster, including both direct and indirect

connections. For example, if Session A is similar to B, and B is similar to C, all three sessions belong to the same cluster, even if A and C are not directly similar. In contrast, similarity degree only counts sessions that are directly similar to the given session.

These features measure how much behavior is reused across sessions. Large cluster sizes and high similarity degrees indicate repeated use of common command patterns, which is typical of automated attacks. In contrast, isolated sessions or small clusters are more consistent with human-driven behavior.

3.3.1.1 Feature Summary

Table 3.4 demonstrates the final feature vector, consisting of 13 features, that are extracted from each session. These features are used by the rule-based detection engine for bot detection, as well as for subsequent attacker-level aggregation and the construction of attacker-level behavioral profiles in the following sections.

3.3.2 Stage 2: Rule-Based Bot Detection Engine

After the feature extraction phase, the extracted features are used to design a rule-based detection engine that labels each session as either bot-driven or human-driven behavior. To achieve this, the system uses a scoring approach that combines multiple behavioral features into a single decision score.

The thresholds and weights used in this detection framework are determined empirically based on exploratory analysis of the dataset. These values are selected to reflect common characteristics of automated attacks, such as fast execution speed and repeated command patterns, which are less typical in human-driven behavior.

For each session, the detection engine computes a final score by combining two

Table 3.4: Session-level and attacker-context behavioral features extracted in Module 3.

Category	Feature	Meaning
Volume	Total sessions per attacker	Total number of sessions associated with the attacker.
	Commands per session	Number of executed commands within the session.
Temporal dynamics	Session duration	Elapsed time between the earliest and latest timestamps in the session.
	Average time per command	Average time spent per command within the session.
	Average inter-command time	Average delay between consecutive commands.
	Burst rate	Maximum number of commands within a one-second window, normalized by the total number of commands.
	Time to first command	Delay between the session start and the first executed command.
Intent	Maliciousness score	Weighted sum of command-category severities within the session.
	Attack tactic sequence	Ordered sequence of tactic labels derived from commands.
Similarity	Cluster ID	Identifier of the group of similar sessions; -1 indicates that the session is not assigned to a similarity cluster.
	Cluster size	Number of sessions assigned to the same similarity cluster.
	Similarity degree	Number of sessions with similarity above the predefined threshold.
	Maximum similarity score	Highest Nilsimsa similarity score observed with any other session.

components:

- (i) **behavior-weighted score:** This component is based on session-level features such as interaction volume, maliciousness score, temporal execution characteristics, and execution rhythm. It captures properties of each session without comparing it to other sessions.
- (ii) **Similarity-based components:** This component captures repeated behavioral patterns across sessions using Nilsimsa-based similarity analysis. It includes both the similarity-based signal, which reflects repeated behavioral templates, and a clone penalty, which captures near-identical command reuse as strong evidence of automation.

3.3.2.1 Behavior-weighted score.

The behavior-weighted score is a session-level detection component that reflects the characteristics of activity within a session, including interaction volume, maliciousness score, execution speed, and execution rhythm. The thresholds and weights used in this score are derived from the empirical analysis described earlier and are designed to capture session-level characteristics of automated behavior.

Although session duration is extracted as a temporal feature, it is not directly included in the behavior-weighted score. Session duration reflects how long a connection remains active, but after some analysis, we realized that it does not reliably distinguish automated and human-driven activity because both types of sessions may be short or long due to attacker objectives, network conditions, or idle periods. Therefore, the score emphasizes more discriminative temporal indicators, including inter-command timing, average time per command, burst rate, and time to first command.

The behavior-weighted score is computed as the sum of the points assigned to all satisfied conditions defined in Table 3.5. Each rule contributes independently, and multiple conditions may be satisfied at the same time. The resulting score reflects the level of automation in each session.

3.3.2.2 Similarity-based components.

This component captures repeated and similar behavioral patterns across sessions using Nilsimsa-based similarity analysis, as defined in the feature extraction stage. Sessions with similar command structures are identified based on their similarity scores, and these relationships are used to derive two complementary signals.

The first signal, referred to as the **similarity-based signal**, measures how often

Table 3.5: Rule-based conditions and point assignments used to compute the behavior-weighted score.

Dimension	Condition	Points	Interpretation
Volume (session-level)	Total sessions per attacker > 20	+3	High repetition across sessions indicates automated campaigns.
	$5 < \text{Total sessions per attacker} \leq 20$	+1	Moderate repetition suggests partial or assisted automation.
Volume (command-level)	Commands per session > 15	+2	Dense command execution suggests scripted behavior.
Intent	Maliciousness score > 20	+5	Strong evidence of repeated attack-oriented actions.
	$10 < \text{Maliciousness score} \leq 20$	+3	Indicates recurring malicious or tool-related activity.
Speed	Average inter-command time < 1.0 s	+3	Sub-second delays are typically not achievable by human operators.
	$1.0 \leq \text{Average inter-command time} < 3.0$ s	+1	Faster-than-human interaction suggests automation assistance.
	Average time per command < 2.0 s	+1	Sustained rapid execution across commands.
	Time to first command < 0.5 s	+10	Immediate response after session start strongly indicates automation.
Rhythm	Burst rate < 0.1	+2	Consistent execution patterns support automation.

a session shares similar behavior with other sessions. For each session, the number of similar sessions is counted based on the same similarity threshold that is used during feature extraction. This signal captures both direct and indirect similarity relationships across sessions. Sessions that belong to large similarity neighborhoods receive higher scores, which indicates frequent reuse of similar behavioral templates. In contrast, sessions with few or no similar counterparts receive lower scores, which reflects limited evidence of automation at the pattern level.

The second signal, referred to as the **clone penalty**, captures near-identical command reuse across sessions. This component is based on the maximum similarity score between a given session and all other sessions. When this score exceeds a high similarity threshold, an additional penalty is applied. The threshold is selected based on the observation that similarity scores close to the maximum value correspond to near-identical command sequences produced by deterministic scripts. Such high levels of similarity are unlikely to occur in human-driven interactions, where natural variations are expected, and therefore indicate a high likelihood of automation.

These two signals together provide complementary evidence of automated behavior.

Table 3.6 summarizes the thresholds and point assignments that are used to compute these similarity-based components.

Table 3.6: Similarity-based scoring rules used to compute the similarity signal and clone penalty.

Component	Condition	Points	Rationale
Similarity-based signal	Similarity neighbor count ≥ 200	+2	The session belongs to a widely reused behavioral template.
Similarity-based signal	$50 \leq \text{similarity neighbor count} < 200$	+1	Moderate reuse suggests repeated scripted behavior.
Similarity-based signal	Similarity neighbor count < 50	0	Limited reuse provides weak evidence of automation.
Clone penalty	Maximum similarity score ≥ 100	+3	Near-exact duplication indicates clone-like scripted execution.

3.3.2.3 Bot Label Assignment

After computing the final decision score, the framework assigns a binary label to each session using an interpretable rule-based process. Each session is labeled as either bot-driven or human-driven behavior.

A session is classified as bot-driven when it satisfies one of two complementary conditions that are designed to capture both strictly repetitive and non-repetitive forms of automation.

1. **Strict periodic repetition.** Sessions that show exact repeated execution patterns are directly labeled as bot-driven. This periodic behavior is identified during the session reduction stage in Module 2 and through the command normalization process. During normalization, variable elements such as IP addresses, file names, and dynamically generated values are replaced with generic tokens. As a result, command sequences that appear different at the raw level

may become identical after masking. Such exact repetition indicates deterministic script execution and therefore is a strong indicator of automated behavior.

2. **Score-based detection.** Sessions that do not show strict repetition are classified based on their final decision score. A session is labeled as bot-driven when its score is greater than or equal to a threshold of 10. This threshold is selected based on the distribution of scores and reflects the combined effect of multiple behavioral indicators. In most cases, reaching this score requires several strong signals, such as fast execution or high maliciousness, rather than a single feature.

This condition also allows the detection of low-frequency or single-session attacks, referred to as *lone-wolf* automation. These attacks may not repeat across sessions and therefore have low similarity-based signals. However, they can still be detected based on their internal behavioral patterns.

For example, a session may include a sequence of automated commands such as downloading a payload, changing permissions, and executing it, which are all performed within a very short time interval. Although this behavior appears only once and does not match other sessions, the combination of fast execution and high maliciousness produces a high final score, which leads to correct detection as a bot-driven session.

For this reason, the threshold is not selected too high, so that non-repetitive but clearly automated sessions are not missed.

These two conditions allow the framework to capture different forms of automated activity. The periodicity rule identifies deterministic and highly repetitive scripts, while the score-based condition captures more adaptive or less frequent automated behaviors. This design ensures that both large-scale bot campaigns and isolated

automated sessions are effectively detected. The labelling logic is summarized in Table 3.7.

Table 3.7: Session-level bot label assignment rules.

Condition	Decision Rule	Rationale
Strict periodic repetition	Sessions exhibiting identical execution patterns after masking are labeled as bot-driven.	Deterministic repetition strongly indicates scripted automation and is unlikely to result from human interaction.
Score-based detection	When no strict repetition is observed, sessions with a final score of ≥ 10 are labeled as bot-driven; otherwise, they are labeled as human-driven.	Captures non-repetitive and single-session automation (<i>lone-wolf</i> bots) using combined behavioral signals.

3.3.2.4 Stage 3: Attacker Profile Construction

This section presents the final stage of the framework, which focuses on generating attacker profiles. For this purpose, all session-level information is aggregated into a unified profile representation.

Each session is represented using all 13 behavioral features extracted in the previous stage, together with the bot label produced by the rule-based detection engine. These features cover volume, temporal, intent, and similarity-based characteristics. The session-level representations are then aggregated across all sessions associated with the same attacker to construct the final attacker profile.

3.3.2.5 Profile Construction

To construct an attacker-level profile, all session-level information for each attacker is aggregated across sessions belonging to the same attacker. Each session is represented by its features, label, and corresponding tactic sequence. These session-level representations are then combined into a single attacker profile.

This aggregation is important because attacker behavior may vary across sessions,

and a single session may not reflect the overall activity. Combining multiple sessions provides a more stable and representative view of attacker behavior.

Formally, let an attacker A be associated with a set of sessions $\mathcal{S}_A = \{S_1, S_2, \dots, S_k\}$. For each session S_i , a feature vector $\mathbf{x}_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(d)})$ is extracted, where d denotes the total number of extracted features. The attacker-level profile $\mathbf{X}_A$ is constructed by applying aggregation functions across all sessions.

Different aggregation strategies are used depending on the feature type:

- **Continuous features.** Continuous features (e.g., number of commands, session duration, maliciousness score, temporal features, and similarity measures) are aggregated using the mean:

$$X_A^{(j)} = \frac{1}{k} \sum_{i=1}^k x_i^{(j)}, \quad \forall j \in \mathcal{C}.$$

- **Binary features.** Binary features (e.g., periodic behavior and session-level bot labels) are aggregated using the maximum:

$$X_A^{(j)} = \max_{1 \leq i \leq k} x_i^{(j)}, \quad \forall j \in \mathcal{B}.$$

- **Count-based features.** The total number of sessions is preserved directly, as it represents the overall activity level of the attacker.
- **String-based features.** Non-numerical attributes are handled using appropriate strategies. Sequence-based features, such as attack tactic sequences, are concatenated across sessions to preserve behavioral context. Identifier attributes, such as attacker IP and cluster ID, are retained by selecting a representative value.

The resulting attacker profile provides a compact representation that summarizes both behavioral patterns and key indicators of automation. These profiles are then used for further analysis and interpretation of attacker behavior.

3.4 Conclusion

This chapter presented the design and implementation of the proposed Bot Detection and Profiling Framework, which transforms raw Cowrie honeypot telemetry into structured behavioral intelligence. The methodology was organized into three main modules that convert high-volume log data into meaningful representations of attacker behavior that are suitable for further analysis.

Module 1 focused on data collection and preprocessing. In this stage, raw honeypot events were filtered, reconstructed into coherent sessions, and ordered in time to preserve execution context. This step ensured that only meaningful post-compromise interactions were kept for analysis.

Module 2 addressed the challenges of variability and redundancy by introducing a graph-based command normalization approach using the Heterogeneous Cyber-Knowledge Graph (H-CKG), followed by rule-based session grouping. This module produced normalized and reduced session representations that preserve behavioral patterns while removing redundant executions.

Module 3 focused on extracting behavioral intelligence from the processed sessions. A set of complementary features capturing interaction volume, temporal dynamics, command semantics, and structural similarity were extracted to describe attacker behavior. These features were then used in an interpretable, rule-based detection engine to distinguish automated behavior from human-driven activity. The detection strategy supports both large-scale automated campaigns and low-frequency or single-

session attacks.

Finally, session-level information was aggregated into attacker-level profiles using well-defined aggregation operators. This step provides a consolidated view of attacker behavior across multiple sessions that preserves both recurring patterns and key indicators of automation. These profiles serve as a stable foundation for further analysis.

Overall, the proposed framework provides a systematic, scalable, and interpretable approach for analyzing attacker behavior in honeypot environments. It enables reliable differentiation between bot-driven and human-driven activity and supports further analysis, such as behavioral comparison, similarity-based grouping, and investigation at the campaign level.

Chapter 4

Evaluation and Analysis

This chapter presents a systematic evaluation of the proposed bot detection and behavioral profiling framework introduced in Chapter 3. The evaluation uses an eleven-month real-world dataset collected from a Cowrie SSH honeypot deployment at the Canadian Institute for Cybersecurity [36]. The dataset contains more than 1.5 million command interactions, 20,220 reconstructed sessions, and 4,896 unique attackers. This provides a large and diverse foundation for evaluating the scalability, robustness, and analytical effectiveness of the framework.

The main objective of this chapter is to evaluate how effectively the proposed framework transforms high-volume, noisy honeypot telemetry into structured behavioral intelligence. The analysis considers not only detection performance but also computational efficiency, interpretability, and the ability to reveal higher-level attack structures that are not directly visible in raw honeypot logs.

To achieve this, the chapter is organized around five research questions. Each question is linked to a specific stage of the data processing pipeline. Together, these questions provide a step-by-step validation of the framework, from command normalization and data reduction to bot detection and campaign-level analysis.

This chapter addresses the following research questions:

1. **RQ1 – Efficacy of Topological Masking and Hybrid Filtering:** How effectively does the proposed graph-based masking approach, combined with a hybrid safety filter, identify and normalize variable components of attack commands compared to static normalization methods?
2. **RQ2 – Effectiveness of Rule-Based Session Grouping and Reduction:** How effectively does the proposed rule-based session grouping approach reduce computational complexity and dataset size compared to raw session-level analysis?
3. **RQ3 – Bot Detection Accuracy:** To what extent does the proposed rule-based detection pipeline accurately classify automated threats when validated against a semi-supervised behavioral clustering ground truth, and what specific detection gaps does it show for low-interaction behaviors?
4. **RQ4 – Intra-Class behavioral Similarity of Bot and Human Attackers:** When attackers are represented using aggregated behavioral profile vectors, do automated bot attackers show higher intra-class behavioral similarity than interactive human attackers?
5. **RQ5 – Attribution and Detection of Coordinated Multi-Identity Attack Groups:** Can the Attacker Similarity Graph reveal coordinated multi-identity attack groups that are missed by IP-based analysis?

Each section of this chapter corresponds directly to one of the research questions listed above. The evaluation is based on quantitative measurements, outputs generated by the implemented pipeline, and cross-validation against raw honeypot evidence where needed. Overall, the results show that the proposed framework scales

effectively to multi-million-command datasets while enabling behavioral, structural, and campaign-level analysis beyond traditional event-based approaches.

Figure 4.1 shows the relationship between the data processing pipeline and the research questions evaluated in this chapter. Each research question is linked to a specific output of the framework, ensuring that the evaluation is tied to clear processing stages. RQ1 evaluates topological masking and hybrid filtering for normalized command representations. RQ2 examines how these normalized outputs support rule-based grouping for dataset reduction and computational efficiency. RQ3 evaluates bot detection performance using the processed data. RQ4 analyzes intra-class behavioral similarity at the attacker level. RQ5 extends this analysis to identify coordinated groups and shared infrastructure. This mapping shows the connection between all evaluation stages and provides a clear pipeline-based validation of the proposed framework.

4.1 RQ1: Efficacy of Topological Masking and Hybrid Filtering

Research Question. How effectively does the proposed graph-based masking approach, combined with a hybrid safety filter, identify and normalize variable components of attack commands in high-entropy honeypot data?

Objective. This research question examines whether graph structure can separate stable command syntax from attacker-specific variables. The method is based on the assumption that common syntax tokens appear frequently and form highly connected nodes in the command co-occurrence graph, while attacker-specific parameters appear less frequently and remain weakly connected.

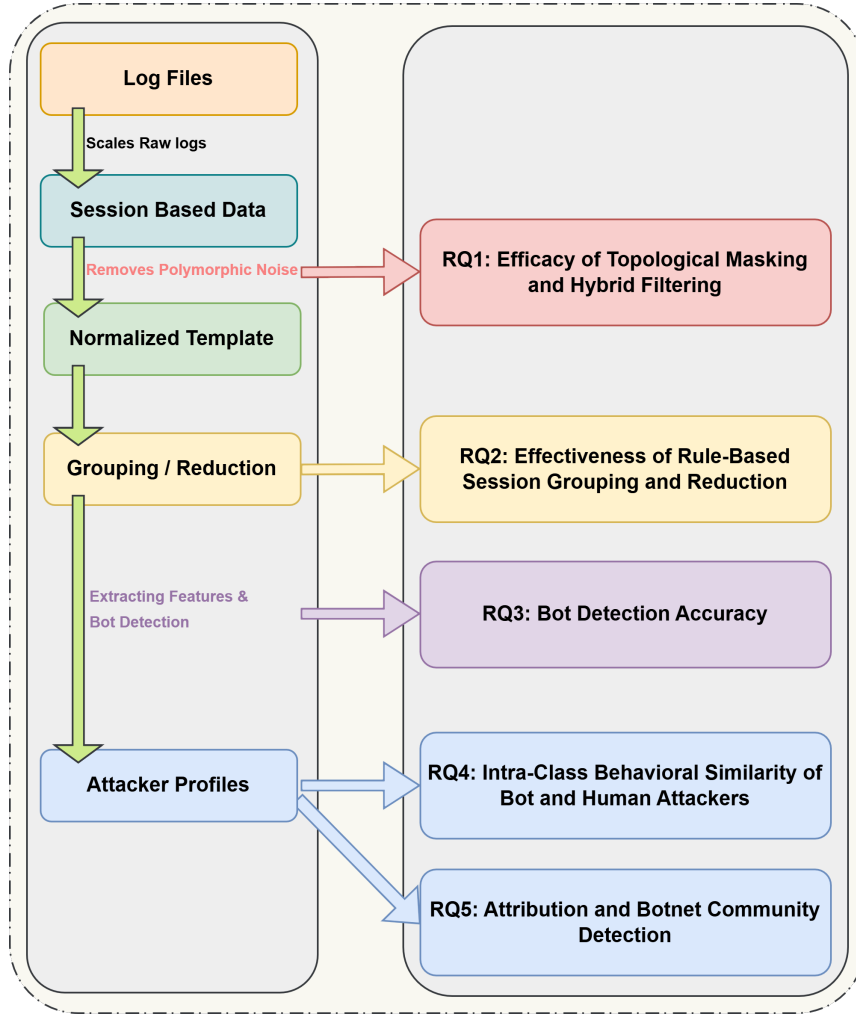


Figure 4.1: Mapping of research questions to the corresponding stages of the analysis pipeline.

Evaluation Approach. The evaluation focuses on (i) the ability of the graph-based method to distinguish structural syntax from polymorphic variables, and (ii) the contribution of the Hybrid Safety Filter in correcting topology-based masking errors and improving normalization reliability.

4.1.1 Quantitative Discovery Results

Analysis of the graph-derived masking dictionary shows a clear imbalance in attacker command composition. Across the evaluated dataset, a total of **18,396** unique

command tokens were extracted and classified using graph-topological analysis.

As shown in Figure 4.2, only **313 tokens (1.7%)** were identified as stable syntax hubs and preserved, while **18,083 tokens (98.3%)** were classified as polymorphic variables and normalized.

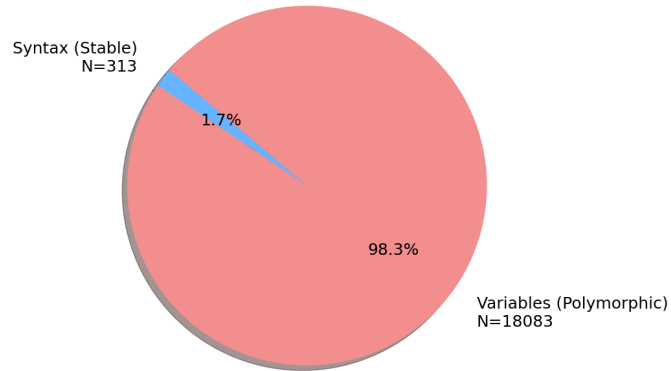


Figure 4.2: Topological classification of command tokens based on degree centrality. Most tokens are identified as polymorphic variables, while only a small stable syntax core is preserved.

This result shows that attacker commands are dominated by variable content rather than reusable syntax. It also shows why static rule-based normalization is difficult in high-entropy honeypot data.

Beyond simple token counts, the graph-based approach automatically identified 16,265 complex variable strings, including randomized filenames, attacker-generated identifiers, and password- or hash-like values. These tokens usually appeared only once or twice across the dataset and showed no stable structural reuse. Detecting such variables with static regular expressions would require manual effort and prior knowledge. In contrast, the graph-based method performs this normalization automatically without predefined rules.

4.1.2 Hybrid Safety Filter Validation

Although token frequency is useful for separating syntax from variables, it has a limitation. Attacker-controlled infrastructure that appears frequently, such as command-and-control servers, may be incorrectly classified as stable syntax.

The analysis confirms this issue. The graph-based method alone labeled **70 unique IPv4 addresses** as SYNTAX (Keep) because they appeared repeatedly across sessions. These tokens correspond to attacker-controlled endpoints and would have been incorrectly preserved.

The Hybrid Safety Filter corrects this problem by enforcing IP-based masking regardless of graph degree. All 70 misclassified IP tokens were successfully identified and normalized. This prevents indicator leakage and improves the consistency of the normalized output.

4.1.3 Discussion

The results show that the proposed graph-based masking approach effectively separates structural syntax from polymorphic variables in high-entropy honeypot data. The observed **98.3% normalization rate** indicates that most command tokens are variable components rather than stable syntax.

The Hybrid Safety Filter also plays an important role by correcting topology-based errors. Without this step, frequently reused malicious infrastructure would remain unmasked and reduce the quality of normalized representations.

Overall, these findings show that the combined approach provides a scalable, adaptive, and reliable method for command normalization. This creates a strong foundation for later stages of the framework, including session reduction, similarity analysis,

and behavioral profiling.

4.2 RQ2: Effectiveness of Rule-Based Session Grouping and Reduction

Research Question. How effectively does the proposed rule-based grouping approach reduce computational cost and dataset size in high-entropy honeypot data?

Objective. This research question examines whether the proposed grouping and reduction strategy can remove redundant sessions while preserving meaningful behavioral patterns. The method is based on the observation that automated attacks often repeat the same command sequences across many sessions with only small variations.

Evaluation Approach. The evaluation focuses on (i) the reduction in pairwise comparisons required for similarity analysis, and (ii) the reduction in dataset size achieved through rule-based grouping and normalization.

4.2.1 Reduction in Computational Complexity

The raw Cowrie telemetry is large and contains 288 JSON log files, 8,702,601 events, and 959,346 raw sessions. Direct similarity analysis at this scale is not practical. For this reason, the evaluation uses the reconstructed interactive dataset, which contains 20,220 command-level sessions associated with 4,896 unique attackers.

In a session-level analysis, the number of pairwise comparisons grows quickly as the number of sessions increases. For a dataset with N sessions, the total number of global comparisons is:

$$\binom{N}{2}$$

For the 20,220 interactive sessions, a full global pairwise comparison would require:

$$\binom{20,220}{2} = 204,414,090 \text{ comparisons.}$$

However, the proposed framework does not compare all sessions globally. Instead, similarity analysis is performed at the attacker level. For each attacker, comparisons are computed only among that attacker's sessions. Therefore, the total comparison cost before grouping is calculated as:

$$\sum_i \binom{S_i}{2}$$

where S_i is the number of sessions associated with attacker i . Before grouping, this attacker-level comparison cost is 4,940,555 comparisons.

After rule-based grouping, repeated sessions are reduced to unique behavioral templates for each attacker. The comparison cost after grouping is therefore calculated as:

$$\sum_i \binom{U_i}{2}$$

where U_i is the number of unique behavioral templates for attacker i after grouping. After reduction, the attacker-level comparison cost becomes 257,518 comparisons. This corresponds to a 94.79% reduction in pairwise similarity comparisons.

Figure 4.3 shows this reduction at the attacker level. Each point represents one attacker. The horizontal axis shows the number of comparisons before grouping,

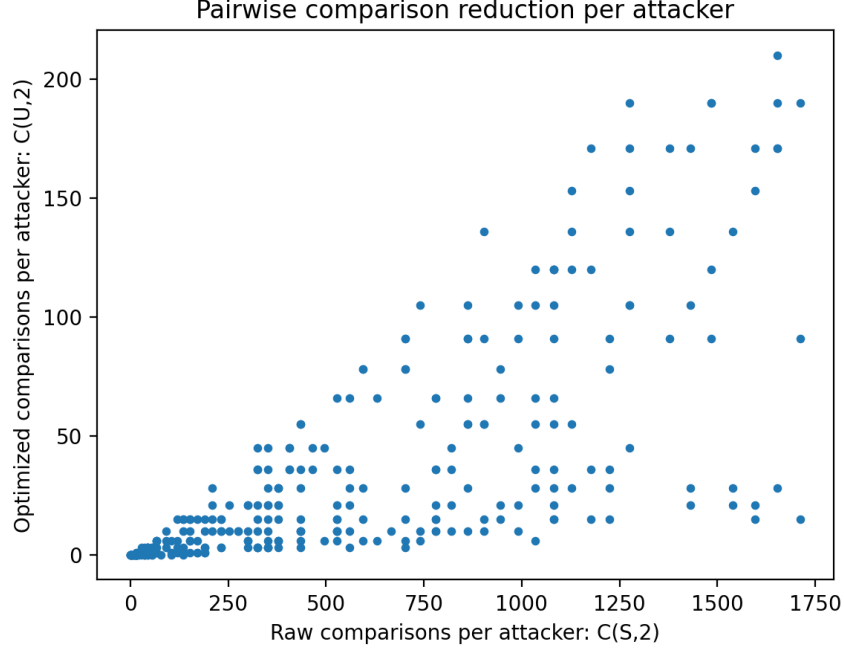


Figure 4.3: Reduction in pairwise similarity comparisons achieved by rule-based grouping.

calculated as $C(S_i, 2)$, where S_i is the number of sessions for attacker i . The vertical axis shows the number of comparisons after grouping, calculated as $C(U_i, 2)$, where U_i is the number of unique behavioral templates after reduction.

Points far below the diagonal line indicate strong reduction, meaning that many repeated sessions were grouped into a smaller number of unique behaviors. As shown in the figure, most points are clearly below the diagonal line. This shows that rule-based grouping removes many redundant comparisons. The effect is strongest for attackers with many sessions, where repeated automated behavior is common.

4.2.2 Reduction in Dataset Size

In addition to reducing computational cost, the proposed approach also greatly reduces dataset size by removing redundant sessions.

As shown in Figure 4.4, the original dataset contains 959,346 raw sessions, including

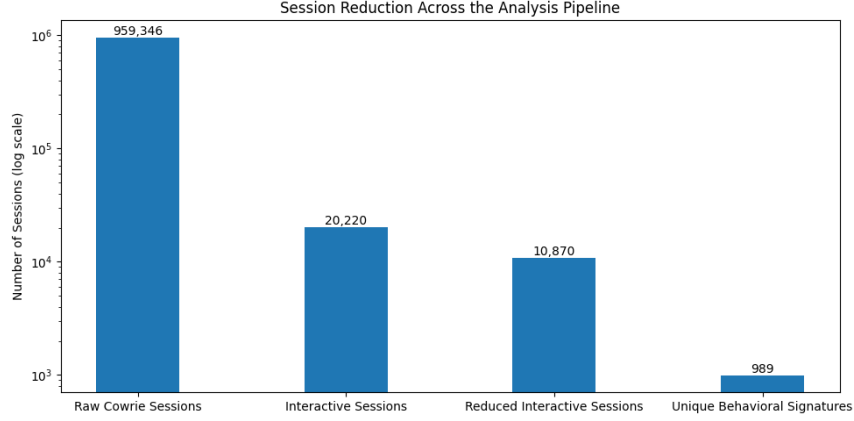


Figure 4.4: Session reduction across the processing pipeline.

noise and incomplete interactions. After filtering, this is reduced to 20,220 interactive sessions.

A further reduction step removes repeated sessions, resulting in 10,870 sessions. Finally, rule-based grouping collapses these sessions into 989 unique behavioral templates.

This corresponds to a reduction of about **20.4×** relative to the interactive dataset, and more than **970×** compared to the raw dataset.

At the token level, dynamic masking reduces the number of unique command tokens from 18,396 to 244, which is a **75.4×** reduction. This allows the analysis to focus on stable command structure instead of variable parameters.

4.2.3 Discussion

The results show that the proposed rule-based grouping and reduction approach effectively removes redundant sessions while preserving meaningful behavioral patterns.

The large reduction in both dataset size and computational cost indicates that much

of the honeypot activity consists of repeated executions of a limited number of attack templates. By grouping these repeated behaviors, the framework reduces noise and simplifies the data without losing important information.

Overall, the reduced dataset provides a compact and stable representation of attacker behavior. This supports efficient similarity analysis, clustering, and attacker profiling in the later stages of the framework.

4.3 RQ3: Bot Detection Accuracy

Research Question. How accurately does the proposed rule-based bot detection pipeline identify automated attackers, and what types of behavior are difficult to detect?

Objective. This research question evaluates the accuracy and reliability of the proposed rule-based behavioral bot detection pipeline. The pipeline assigns a binary label (bot vs. non-bot) to each session using aggregated behavioral features such as execution speed, session persistence, temporal patterns, and malicious command intent.

Evaluation Approach. To evaluate the rule-based bot detection pipeline, a reliable set of ground-truth labels is needed. Raw honeypot data does not include verified labels, and no publicly available dataset with the same structure and behavioral characteristics was found despite an extensive search. As a result, the labels had to be constructed for this evaluation.

To address this limitation, a stratified 20% subsample of sessions was selected for testing. This subset is large enough to remain representative while keeping the manual review process manageable.

To reduce bias, the original rule-based labels were temporarily ignored. Instead, a semi-supervised approach was used. behavioral similarity was first used to cluster sessions, and then a small number of representative samples were manually inspected and labeled. These verified labels were then extended to the remaining sessions within each cluster, creating a consistent reference set for evaluation.

4.3.1 Semi-Supervised Centroid Validation

The semi-supervised labelling process was implemented using a centroid-based validation strategy.

First, the selected sessions were clustered using K-Means with $K = 2$, representing bot and non-bot behaviors.

Next, for each cluster, the 20 sessions closest to the centroid were manually inspected. These central sessions represent the dominant behavior of each cluster and were labeled as bot-driven or human-driven based on execution speed, malicious command score, and interaction patterns.

To improve reliability, additional sessions near the boundary between clusters were also reviewed, since these cases are more likely to be ambiguous. The verified labels were then assigned to all sessions within each cluster, resulting in a human-validated ground truth.

4.3.2 Quantitative Results and Error Analysis

The predictions of the rule-based detection engine were compared with the verified ground truth. The confusion matrix and the related performance metrics are shown in Table 4.1 and Table 4.2.

Actual	Predicted	
	Bot	Human
	Bot	Human
	695 (TP)	26 (FN)
	24 (FP)	234 (TN)

Table 4.1: Confusion matrix for bot detection results.

Metric	Value (%)
Accuracy	94.89%
Precision	96.66%
Recall	96.39%
F1-Score	96.52%

Table 4.2: Bot detection performance metrics.

As shown in the confusion matrix, the model correctly classified most bot- and human-driven sessions, with 695 true positives and 234 true negatives. The number of misclassifications was low, including 24 false positives and 26 false negatives. The performance metrics support this result and show high overall accuracy with a strong balance between precision and recall.

The false positive cases correspond to human-driven sessions that were classified as bots. These sessions were often linked to users who executed commands rapidly or repeatedly, which can resemble automated behavior. However, the low number of such cases shows that the model generally avoids incorrect bot classification.

A closer review of the false negative cases revealed a distinct type of behavior referred to as **fast, low-interaction bots**. These sessions show clear signs of automation but were not detected because of their limited interaction patterns. They are characterized by:

- **Low interaction complexity:** Only a small number of simple commands were executed (for example, `uname`, `ls`, `echo`), which resulted in a low behavioral score.
- **Very high execution speed:** Commands were executed with near-zero delay (for example, $< 0.02s$), which is uncommon for human behavior and strongly suggests automation.

Although these sessions show automated behavior, their low command diversity pre-

vents them from reaching the detection threshold. This highlights a limitation of the current detection logic, which gives more weight to interaction depth and behavioral score than to execution speed alone.

4.3.3 Discussion

The results show that the proposed rule-based detection pipeline achieves high accuracy and strong overall performance. The high precision indicates that human-driven sessions are rarely misclassified as bots, while the high recall shows that most automated attackers are successfully detected.

However, the false negatives reveal an important limitation. The current detection logic gives more weight to behavioral score and interaction depth than to execution speed alone. As a result, fast but low-interaction bots may remain undetected.

This finding suggests that temporal features such as inter-command timing and time-to-first-command should receive more attention in the detection process. Stronger timing-based rules or adjusted thresholds could improve the detection of these cases.

Overall, the proposed pipeline achieves high detection performance while remaining interpretable and scalable, which makes it suitable for real-world honeypot analysis where labeled data is limited. Unlike black-box machine learning approaches, the rule-based design allows direct interpretation of detection decisions, which is valuable for security analysis and forensic investigation.

4.4 RQ4: Behavioral Differentiation and Similarity Structure of Bot and Human Attackers

Research Question. To what extent do aggregated attacker behavioral profiles distinguish automated bot attackers from human-operated attackers, and do bot attackers show higher intra-class behavioral similarity than human attackers?

Objective. This research question examines whether attacker-level behavioral profiles can separate automated attackers from human-operated attackers, and whether automation is associated with stronger internal behavioral consistency. While RQ3 focused on session-level bot detection, RQ4 moves the analysis to the attacker level, where multiple sessions are combined into one behavioral profile.

The main assumption is that automated attack tools usually follow repeated and predictable workflows. As a result, different bot instances are expected to generate similar behavioral profiles. In contrast, human-operated attackers are expected to show more variation because of exploratory actions, adaptive decisions, and less predictable interaction patterns.

4.4.1 Methodology

Each attacker is represented by an aggregated behavioral profile vector created from all related sessions. The profile is built from normalized attacker-level features that capture both behavioral intensity and structural patterns. These dimensions include:

- **behavioral Uniqueness** – how uncommon and individualized the observed actions are.
- **Exploration Diversity** – the range of commands and tactics explored by the

attacker.

- **Malicious Intensity** – the level of clearly malicious activity.
- **Largest Repeated Cluster** – the size of the largest shared behavioral cluster within each class.
- **Tactic Transition Rate** – how often the attacker moves between different attack stages or tactics.

Using the verified labels established in RQ3, attackers are divided into two groups: automated bots and human-operated attackers. Pairwise cosine similarity is then calculated between attackers within each class to measure intra-class behavioral similarity. In addition, average profile values are compared across both groups.

4.4.2 Quantitative Results

Table 4.3 presents the average normalized feature values for human-operated attackers and automated bots. The results show clear behavioral differences between the two groups.

Table 4.3: Average normalized behavioral profile values for human-operated attackers and automated bots.

Feature	Humans	Bots
Behavioral Uniqueness	0.260	0.026
Exploration Diversity	0.607	0.050
Malicious Intensity	0.419	0.991
Largest Repeated Cluster	0.414	1.000
Tactic Transition Rate	0.427	0.988
Mean Intra-class Similarity	0.550	0.992

Human-operated attackers scored higher in behavioral Uniqueness and Exploration Diversity. This suggests that human attackers are more likely to test different commands, try different actions, and behave in less predictable ways. Their activity is therefore more individualized and less repetitive.

Malicious Intensity captures the severity of attack-oriented activity, while Tactic Transition Rate captures how often the attacker moves between different attack stages. Bots score high in both because they often execute predefined multi-stage attack workflows that include several malicious actions.

The Largest Repeated Cluster feature was measured using the largest cluster found within each class. The largest human cluster contained 980 attackers, while the largest bot cluster contained 2370 attackers. This shows that repeated large-scale behavior was much stronger among bots than among human-operated attackers.

Bots also showed much higher mean intra-class similarity (99.2%) than human attackers (0.550). This means bot attackers behave very similarly to one another, while human attackers show much greater variation across profiles.

Figure 4.5 provides a visual comparison of these average behavioral profiles.

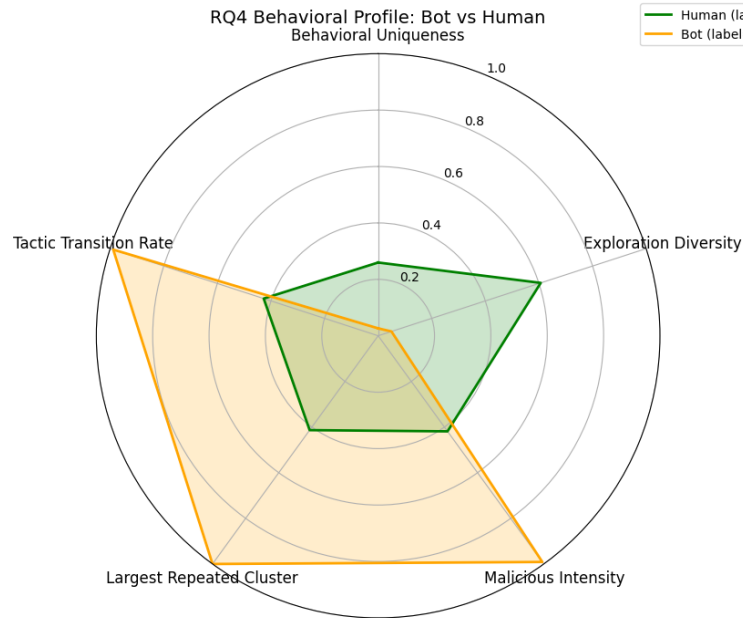


Figure 4.5: behavioral profile comparison between automated bot attackers (orange) and human-operated attackers (green).

The results from both the table and the figure show a clear separation between automated and human-operated attackers. Bots are more repetitive, more aggressive, and

more behaviorally consistent. Human attackers are more diverse, less predictable, and more exploratory.

4.4.3 Discussion

These results show that behavioral similarity is an important characteristic of automated attack activity. Even when bots use different IP addresses, their overall profiles remain highly similar because they rely on shared tools, repeated scripts, and common attack goals.

In contrast, human-operated attackers show much greater diversity. Their sessions include broader experimentation, different command choices, and more individualized workflows, which lowers similarity between attackers.

This supports the use of similarity-based attacker modeling. Because bots naturally group around common behavioral templates, attacker-level similarity analysis can help identify coordinated campaigns, reused automation logic, and distributed malicious infrastructure. At the same time, the higher variation among human attackers helps separate manual intrusions from large-scale automated activity.

Overall, RQ4 shows that attacker behavioral can be identified not only at the session level, but also at the attacker level through aggregated behavioral profiling. This directly supports the campaign attribution analysis presented in RQ5.

4.5 RQ5: Attribution and Detection of Coordinated Multi-Identity Attack Groups

Research Question. Can attacker behavioral similarity graphs reveal coordinated multi-identity attack groups that are not visible through IP-based analysis alone?

Objective. Building on the attacker behavioral profiles developed in RQ4, this research question examines whether distributed source IP addresses can be linked into coherent behavioral groups that suggest shared automation logic, reused tools, or common infrastructure. Traditional IP-based analysis treats each source IP address as an independent attacker, while graph-based behavioral analysis may uncover campaign-level relationships that are less visible under an IP-based view.

4.5.1 Methodology

An attacker similarity graph is constructed using a hybrid behavioral representation. First, each attacker’s sequence of high-level attack tactics is compressed using run-length encoding (RLE) and embedded using Sentence-BERT (SBERT) to capture the structure and ordering of the attack workflow. Second, a normalized behavioral feature vector summarizes quantitative characteristics such as session duration, maliciousness score, and command burst rate.

Pairwise similarity between attackers is then calculated using cosine similarity over the combined representation. An undirected edge is created between two attacker nodes only when similarity exceeds a threshold of **85%**. Preliminary testing showed that lower thresholds created overly broad groups, while higher thresholds separated attackers that were still highly similar.

Behavioral groups are extracted as connected components in the resulting graph. In this context, each node represents an observable source IP address, although multiple IP addresses may belong to the same operator or infrastructure. As a result, some groups may also contain indirectly connected attackers linked through chains of high pairwise similarity.

4.5.2 Quantitative Results

Analysis of the attacker similarity graph revealed multiple strongly connected behavioral groups that are not visible through IP-based analysis alone. These groups consist of attackers from different source IP addresses that show highly similar workflows, which suggests possible coordination or shared automation logic.

The strongest attribution evidence was observed in **behavioral Group 15**. This group contains **13 distinct source IP addresses** connected through very high behavioral similarity. Despite different network origins, all members executed an identical normalized command template. The group also showed an average pairwise similarity above 94%.

Further inspection showed that all 13 attackers attempted to retrieve a payload from the same external endpoint (61.215.151.173). This shared infrastructure provides strong evidence of a coordinated multi-identity campaign that would likely remain hidden under IP-based analysis.

Figure 4.6 visualizes this group and shows how graph-based behavioral analysis can link multiple IP identities into a single coordinated campaign.

To provide a broader view of the graph structure, Table 4.4 summarizes the four largest behavioral groups identified from the attacker similarity graph. Group size refers to the number of distinct source IP identities contained in each connected component. Larger groups usually reflect repeated attack templates or shared automation logic that appear across many distributed attackers.

Very large groups should also be interpreted carefully, as they may reflect widely reused commodity attack scripts or common intrusion playbooks rather than a single centrally managed campaign.

Attacker Similarity Graph: 13-Node Behavioral Group

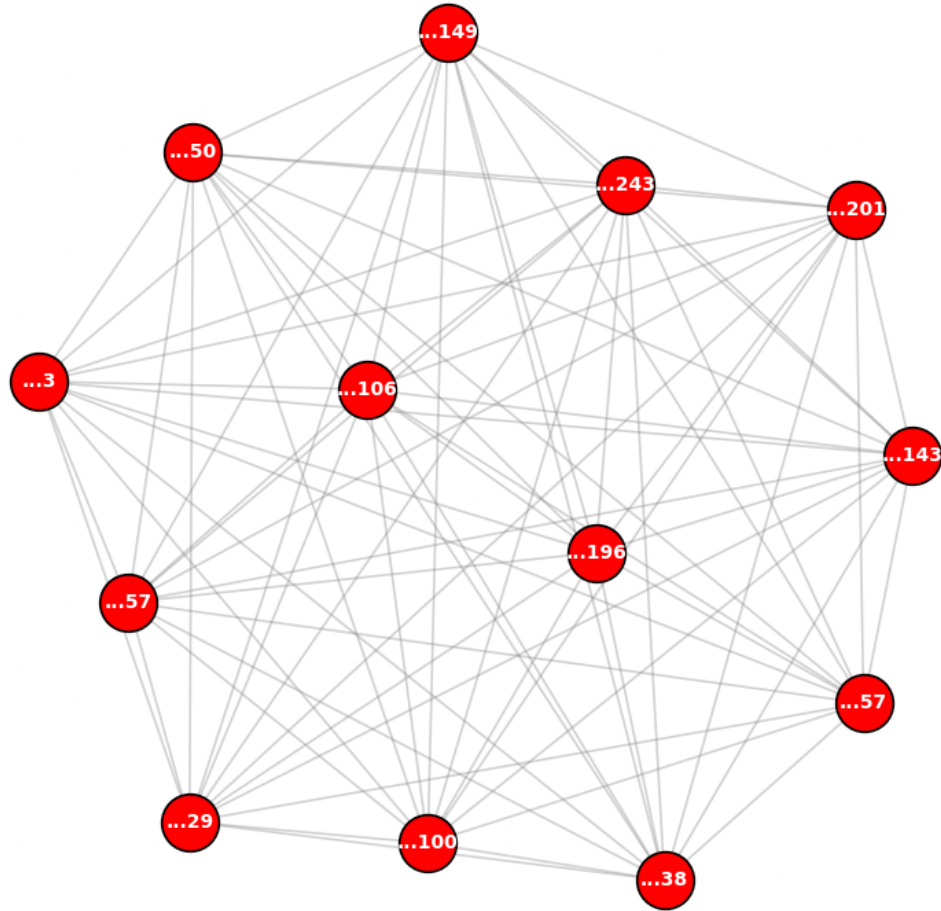


Figure 4.6: Attacker Similarity Graph of behavioral Group 15. Nodes represent distinct source IP identities, and edges indicate strong behavioral similarity ($> 85\%$). The dense connections suggest coordinated activity across multiple distributed attackers.

4.5.3 Discussion

These results show that graph-based behavioral grouping can support campaign-level attribution beyond IP-based analysis alone. behavioral Group 15 demonstrates that multiple distributed source IP addresses can be linked through shared behavioral and common external infrastructure, even when they appear unrelated at the network level.

Table 4.4: Four largest behavioral groups identified from the attacker similarity graph using a cosine similarity threshold greater than 85%.

Rank	Group ID	Size (IPs)	behavioral Characteristics
1	4	3350	Near-identical normalized command templates, consistently high maliciousness scores, and highly uniform execution patterns consistent with large-scale automated activity.
2	2	803	Reconnaissance-heavy workflows characterized by repeated system enumeration, low interaction diversity, and limited post-compromise actions.
3	1	236	Short and repetitive discovery-focused attack sequences with low variation and rapid execution speed consistent with scripted scanning tools.
4	39	127	Repeated low-interaction scanning behavior with shallow command depth and limited evidence of follow-on exploitation or persistence.

This is especially useful against IP rotation strategies, where coordinated attackers spread activity across many visible identities. Under a traditional IP-based view, these entities would be treated as separate attackers, which makes campaign detection more difficult.

At the same time, not every large behavioral group should be interpreted as a single operator-controlled botnet. Some groups may reflect shared malware families, reused public scripts, or common attack playbooks. For this reason, behavioral groups should be treated as strong indicators of coordination rather than absolute proof of common ownership.

Overall, RQ5 shows that attacker similarity graphs provide a practical way to move from isolated attacker observations to campaign-level understanding. This complements the behavioral profiling results presented in RQ4 and extends them toward attribution analysis.

4.6 Chapter Summary

This chapter presented a full evaluation of the proposed framework using a large real-world SSH Cowrie honeypot dataset collected over eleven months from the Canadian Institute for Cybersecurity [36]. The results show that the framework can transform noisy, high-volume honeypot telemetry into structured and useful behavioral intelligence.

RQ1 demonstrated that the graph-based masking method effectively separates stable command syntax from attacker-specific variables, while the Hybrid Safety Filter improves reliability by masking repeated malicious infrastructure. This created clean normalized command representations for later analysis.

RQ2 showed that rule-based grouping significantly reduced dataset size and comparison cost by collapsing repeated sessions into a smaller number of unique behavioral templates. This confirmed that large honeypot datasets contain substantial redundancy and can be processed more efficiently without losing important behavioral information.

RQ3 confirmed that the rule-based detection pipeline achieved strong bot detection performance with high accuracy, precision, recall, and F1-score. Error analysis also identified fast low-interaction bots as an important challenge for future refinement.

RQ4 demonstrated that attacker-level behavioral profiling clearly separates automated bots from human-operated attackers. Bots were more repetitive, more aggressive, and much more behaviorally consistent, while human attackers were more diverse and exploratory.

RQ5 extended the analysis from individual attackers to campaign-level behavior. The attacker similarity graph linked multiple source IP identities into coordinated behavioral groups that were not easily visible through IP-based analysis alone. be-

havioral Group 15 provided a clear example of distributed attackers sharing highly similar behavioral patterns and common infrastructure.

Taken together, the results of Chapter 4 validate the complete analysis pipeline. The framework improves data quality, reduces computational complexity, detects automated threats accurately, distinguishes attacker types, and uncovers coordinated campaigns. These findings strongly support the thesis claim that large-scale honeypot logs can be transformed into scalable and actionable attacker intelligence.

Chapter 5

Conclusion, Limitations, and Future Work

5.1 Conclusion

This thesis addressed the challenge of extracting useful intelligence from large-scale SSH honeypot data. Raw honeypot telemetry is noisy, repetitive, and difficult to analyze directly at scale. Attackers also use changing command parameters, repeated automation scripts, and distributed source IP addresses, which reduce the value of simple rule-based or IP-based analysis. The main goal of this research was to design a practical framework that can transform raw honeypot logs into structured behavioral intelligence for bot detection, attacker profiling, and campaign-level analysis.

To achieve this, a multi-stage behavioral analysis framework was proposed. The framework combines graph-based command masking, rule-based grouping, interpretable bot detection, attacker-level profiling, and graph-based attribution analysis. Rather than treating each command or session as an isolated event, the framework models repeated behavioral structure across multiple levels of the data.

The evaluation used an eleven-month real-world Cowrie SSH honeypot dataset containing more than 1.5 million command interactions, 20,220 reconstructed sessions, and 4,896 unique attackers. The results showed that the framework performs effectively across all major stages of analysis.

First, the graph-based masking method successfully separated stable command syntax from attacker-specific variables. Most command tokens were identified as variable content and normalized automatically, while the hybrid safety filter corrected repeated malicious infrastructure that would otherwise remain visible. This improved the quality and consistency of command representations.

Second, the rule-based grouping stage greatly reduced redundancy in the dataset. Repeated sessions were collapsed into a much smaller set of unique behavioral templates. This reduced computational cost while preserving meaningful behavioral patterns, making later similarity analysis more practical.

Third, the proposed rule-based bot detection pipeline achieved strong performance with high accuracy, precision, recall, and F1-score. At the same time, the decision process remained transparent and easy to interpret, which is important in security operations where explainability matters.

Fourth, attacker-level profiling revealed clear differences between automated bots and human-operated attackers. Bots were more repetitive, more aggressive, and much more similar to one another, while human attackers showed more exploratory and diverse behavior.

Finally, attacker similarity graphs revealed coordinated multi-identity groups that were less visible through IP-based analysis alone. Multiple source IP addresses could be linked through shared workflows, common infrastructure, and repeated behavioral patterns. This showed the value of moving from isolated event analysis toward campaign-level understanding.

Overall, the findings of this thesis show that behavioral structure is a powerful source of security intelligence. Even in noisy honeypot data, repeated patterns can be used to identify automation, distinguish attacker types, and uncover coordinated activity. The proposed framework provides a scalable and interpretable approach for doing this in practice.

5.2 Limitations

Although the results are strong, this research has several limitations that should be considered.

First, the evaluation was limited by the availability of suitable public honeypot datasets. The study was conducted using a single Cowrie SSH honeypot dataset because we could not find another publicly available SSH honeypot dataset with a similar structure, including session-based attacker logs suitable for behavioral analysis. As a result, the proposed method could not be tested on an independent dataset or compared across multiple honeypot deployments, which may limit the generalizability of the findings to other protocols, geographic regions, or deployment environments. In addition, the available dataset did not include verified ground-truth labels for bot detection. Therefore, the labels used in this study were created through a semi-supervised validation process rather than fully verified external labels. Although this approach provided a practical evaluation baseline, some uncertainty may still remain in borderline or mixed-behavior cases.

Second, some decisions in the framework were based on empirically selected thresholds and heuristic rules. These include masking decisions, similarity thresholds, and bot detection rules. This design was used to keep the framework transparent and interpretable, especially because there is no widely accepted benchmark or fully

labeled reference dataset for calibrating these choices in large-scale honeypot behavioral data. In addition, attacker behavior is noisy and highly variable, which makes fully automatic threshold selection difficult. Although the selected thresholds were supported by empirical testing, different datasets or deployment environments may require recalibration.

Third, the framework mainly focuses on command behavior after interaction begins. This design choice was made because command-level telemetry was the most complete and consistent source of information available in the dataset, while earlier network signals were more limited or outside the scope of the collected logs. As a result, the framework does not fully model scanning patterns, connection metadata, packet timing, or broader threat intelligence feeds that could provide additional context.

Finally, campaign attribution through behavioral similarity should be interpreted carefully. Direct ownership evidence is rarely available in honeypot studies because researchers usually observe attacker actions without access to the underlying infrastructure or operator identity. For this reason, similar behavior does not always prove common ownership or direct coordination. Different attackers may independently reuse the same malware family, public exploit scripts, or widely shared intrusion playbooks.

5.3 Future Work

This thesis provides a foundation for large-scale behavioral intelligence from honeypot data. Future work should build on this foundation by extending the framework into broader, more advanced, and more operational research directions.

First, future work should validate the proposed framework using additional honeypot

datasets collected from different environments. Although the current evaluation used a large eleven-month Cowrie SSH dataset, the behavioral patterns, thresholds, and detection rules may not transfer equally well to other settings. Testing the framework across different geographic regions, network environments, honeypot deployments, and collection periods would help assess its robustness and generalizability.

Second, an important next step is the development of a unified multi-protocol attacker intelligence framework. This research focused on SSH honeypot data, but the same behavioral approach can be extended to Telnet, HTTP, FTP, IoT, and industrial control system honeypots. A cross-protocol model would allow researchers to track how the same attacker groups reuse tactics across different services and technologies.

Third, future work should develop stronger ground-truth labelling strategies for attacker behavior in honeypot datasets. Since verified public labels were not available, this thesis used a semi-supervised centroid validation process to create a reference set for evaluation. Future studies could improve label reliability by combining expert review, independent reviewer agreement, malware or payload evidence, repeated infrastructure indicators, and external threat intelligence.

Fourth, future work should investigate adaptive and data-driven methods for selecting thresholds and detection parameters. In this thesis, several thresholds and heuristic rules were selected manually, including masking thresholds, similarity thresholds, and bot detection scores. Future versions of the framework could use adaptive thresholding, percentile-based calibration, statistical distribution analysis, or validation-driven parameter tuning to reduce manual recalibration when applied to new honeypot settings.

Fifth, future work should improve the graph-based masking strategy by incorporating semantic and contextual signals in addition to token frequency. Although

the Hybrid Safety Filter masks IP-formatted tokens, degree-based masking alone cannot fully distinguish structural syntax from frequently reused attacker-controlled artifacts. Future work could combine graph degree with token type recognition, semantic classification, threat intelligence indicators, URL and domain analysis, and context-aware command parsing to improve normalization robustness.

Sixth, future work should also improve the detection of fast, low-interaction bots. In this thesis, some automated sessions with very few commands remained difficult to detect because they did not accumulate enough behavioral evidence to pass the detection threshold. Future versions of the framework could introduce dedicated timing-based rules, stronger weighting for time-to-first-command and inter-command delay, and micro-session anomaly detection to better identify short but highly automated sessions. This would help reduce false negatives for bots that execute only a small number of commands at very high speed.

Seventh, future work should incorporate pre-interaction signals in addition to post-compromise command behavior. The current framework mainly analyzes commands entered after interaction begins, but earlier signals such as scanning frequency, failed authentication attempts, connection duration, source-port behavior, and login timing may provide useful context. Combining pre-interaction metadata with command-level behavioral profiles could improve early detection, support better attacker fingerprinting, and help identify automated scanners before they execute meaningful commands.

Eighth, the attacker similarity graph can be expanded into a global threat relationship graph that combines behavior, infrastructure, malware families, payload hosting servers, domain activity, and external intelligence feeds. Such a graph could support deeper attribution analysis and reveal hidden links between campaigns operating across different environments.

Finally, future work can investigate attacker evolution and concept drift over long time periods. Longitudinal studies may reveal how automation tools mature, how command templates change, and how attacker groups split, merge, or reappear. Future versions of the framework could use sliding-window analysis, periodic threshold recalibration, and continuous profile updating to maintain effectiveness as attacker behavior changes over time. This would provide valuable insight into the life cycle of threat campaigns.

5.4 Final Remarks

This thesis shows that large-scale honeypot logs contain much more value than simple event counts or indicator lists. When analyzed through behavioral structure, they can reveal attacker intent, automation patterns, and coordinated campaigns. The proposed framework demonstrates that scalable and interpretable attacker intelligence can be built directly from noisy deception data, creating practical value for both research and defensive operations.

Bibliography

- [1] J. Kepner, J. Bernays, S. Buckley, K. Cho, C. Conrad, L. Daigle, K. Erhardt, V. Gadepally, B. Greene, M. Jones *et al.*, “Zero botnets: An observe-pursue-counter approach,” *arXiv preprint arXiv:2201.06068*, 2022.
- [2] C. Munteanu, Y. B. Suriyanarayanan, G. Smaragdakis, A. Feldmann, and T. Fiebig, “Attacks come to those who wait: Long-term observations in an ssh honeynet,” in *Proceedings of the 2025 ACM Internet Measurement Conference*, 2025, pp. 628–644.
- [3] F. Bäumer, M. Brinkmann, and J. Schwenk, “Terrapin attack: Breaking ssh channel integrity by sequence number manipulation,” *arXiv preprint arXiv:2312.12422*, 2023.
- [4] A. Javadpour, F. Ja’fari, T. Taleb, M. Shojafar, and C. Benzaïd, “A comprehensive survey on cyber deception techniques to improve honeypot performance,” *Computers & Security*, vol. 140, p. 103792, 2024.
- [5] X. Yang, J. Yuan, H. Yang, Y. Kong, H. Zhang, and J. Zhao, “A highly interactive honeypot-based approach to network threat management,” *Future Internet*, vol. 15, no. 4, p. 127, 2023.

- [6] Q. Chen, Y. Liu, R. Tan, Z. Jin, J. Xiao, X. Wang, F. Zhang, and Q. Liu, “Shadowkube: enhancing kubernetes security with behavioral monitoring and honeypot integration,” *Cybersecurity*, vol. 8, no. 1, p. 63, 2025.
- [7] X. Xiang, H. Liu, L. Zeng, H. Zhang, and Z. Gu, “Ipattributor: Cyber attacker attribution with threat intelligence-enriched intrusion data,” *Mathematics*, vol. 12, no. 9, p. 1364, 2024.
- [8] S. Wang, C. Tu, Y. Zhang, M. Zhang, and P. Xue, “Mapping cyber bot behaviors: Understanding payload patterns in honeypot traffic,” *Sensors*, vol. 26, no. 1, p. 11, 2025.
- [9] N. Provos *et al.*, “A virtual honeypot framework.” in *USENIX Security Symposium*, vol. 173, no. 2004, 2004, pp. 1–14.
- [10] W. Z. Cabral, C. Valli, L. F. Sikos, and S. G. Wakeling, “Advanced cowrie configuration to increase honeypot deceptiveness,” in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2021, pp. 317–331.
- [11] U. Prasad and A. Chawla, “A unified evaluation of learning-based similarity techniques for malware detection,” *arXiv preprint arXiv:2602.15376*, 2026.
- [12] Q. Liu, Q. Huang, F. Fan, H. Wu, and X. Tang, “Detecting sybil addresses in blockchain airdrops: A subgraph-based feature propagation and fusion approach,” in *2025 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2025, pp. 1–7.
- [13] The MITRE Corporation, “MITRE ATT&CK Framework,” <https://attack.mitre.org/>, 2024, accessed: 2026-03-03.

- [14] Lockheed Martin Cybersecurity, “Cyber kill chain ®,” <https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html>, n.d., accessed: 2026-03-03.
- [15] J. R. Douceur, “The sybil attack,” in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
- [16] J. Dong, “Longitudinal analysis of ssh honeypot logs,” in *Wellington Faculty of Engineering Symposium*, 2023.
- [17] T. Haikal, E. Hammad, and S. Ismail, “Characterizing large-scale adversarial activities through large-scale honey-nets,” in *2025 IEEE 16th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2025, pp. 0771–0776.
- [18] H. A. Almazarqi, M. Woodyard, and A. K. Marnerides, “Botpro: Data-driven tracking & profiling of iot botnets in the wild,” *Computers & Security*, p. 104778, 2025.
- [19] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” in *2017 IEEE International Conference on Web Services (ICWS)*, 2017.
- [20] M. Du and F. Li, “Spell: Streaming parsing of system event logs,” in *2016 IEEE International Conference on Data Mining (ICDM)*, 2016.
- [21] E. Karaarslan, E. Güler, E. E. Yüce, and C. Coban, “Towards log analysis with ai agents: Cowrie case study,” *arXiv preprint arXiv:2509.05306*, 2025.
- [22] Z. Hussain, J. K. Nurminen, and P. Ranta-aho, “Training a language model to learn the syntax of commands,” *Array*, vol. 23, p. 100355, 2024.

- [23] S.-Y. Huang, C.-L. Yang, C.-Y. Lin, and C.-Y. Huang, “Cmdcaliper: A semantic-aware command-line embedding model and dataset for security research,” *arXiv preprint arXiv:2411.01176*, 2024.
- [24] Y. Zhou, Y. Chen, X. Rao, Y. Zhou, Y. Li, and C. Hu, “Leveraging large language models and bert for log parsing and anomaly detection,” *Mathematics*, 2024.
- [25] R. Flynn and O. Olukoya, “Using approximate matching and machine learning to uncover malicious activity in logs,” *Computers & Security*, vol. 151, p. 104312, 2025.
- [26] R. Othman, D. Rimawi, B. Rossi, and B. Russo, “From attack descriptions to vulnerabilities: A sentence transformer-based approach,” *Journal of Systems and Software*, p. 112615, 2025.
- [27] K. Kurniawan, E. Kiesling, and A. Ekelhart, “Cykg-rag: Towards knowledge-graph enhanced retrieval augmented generation for cybersecurity,” 2024.
- [28] S. Y. Maturi, “Decoy data nexus: Graph-based integration and analysis of synthetic honeypot logs through structured threat intelligence,” 2025.
- [29] H. Orsini and Y. Han, “Dynamo: Towards network attack campaign attribution via density-aware active learning,” in *SECRYPT 2024-21st International Conference on Security and Cryptography*. SCITEPRESS-Science and Technology Publications, 2024, pp. 91–102.
- [30] B. Al-Sada, A. Sadighian, and G. Oligeri, “Mitre att&ck: State of the art and way forward,” *ACM Computing Surveys*, vol. 57, no. 1, pp. 1–37, 2024.
- [31] F. I. Rahman, S. M. Halim, A. Singhal, and L. Khan, “Alert: A framework for efficient extraction of attack techniques from cyber threat intelligence reports

- using active learning,” in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2024, pp. 203–220.
- [32] W. W. Lo, G. Kulatilleke, M. Sarhan, S. Layeghy, and M. Portmann, “Xg-bot: An explainable deep graph neural network for botnet detection and forensics,” *Internet of Things*, vol. 22, p. 100747, 2023.
- [33] K. Qian, H. Yang, R. Li, W. Chen, X. Luo, and L. Yin, “Distributed detection of large-scale internet of things botnets based on graph partitioning,” *Applied Sciences*, vol. 14, no. 4, p. 1615, 2024.
- [34] A. Mahboubi, K. Luong, H. Aboutorab, H. T. Bui, S. Camtepe, K. Ansari, and B. Barry, “The evolving threat landscape of botnets: Comprehensive analysis of detection techniques in the age of artificial intelligence,” *Internet of Things*, p. 101728, 2025.
- [35] I. Kotenko, E. Fedorchenko, E. Novikova, and A. Jha, “Cyber attacker profiling for risk analysis based on machine learning,” *Sensors*, vol. 23, no. 4, p. 2028, 2023.
- [36] Canadian Institute for Cybersecurity, “Cowrie honeypot telemetry dataset,” <https://www.unb.ca/cic/>, 2025, dataset collected from a distributed Cowrie honeypot deployment operated by the Canadian Institute for Cybersecurity, University of New Brunswick.

Vita

Candidate's full name: Fatemeh Nejati

University attended:

- 2023-2026 University of New Brunswick - MSc. in Computer Science
- 2016-2019 Kharazmi University - MSc. in Mathematics
- 2011-2016 Shiraz University - BSc. in Mathematics

Publications:

- F. Nejati, M. Rabbani, M. Eskandarian, M. Mirani, G. Piya, I. V. Opushnyev, A. A. Ghorbani, and S. Dadkhah, CIC-Trap4Phish: A Unified Multi-Format Dataset for Phishing and Quishing Attachment Detection, arXiv preprint arXiv:2602.09015, 2026.
- M. Eskandarian, M. Rabbani, A. Kaniyamattam, F. Nejati, M. Mirani, G. Piya, I. V. Opushnyev, A. A. Ghorbani, and S. Dadkhah, A Lightweight Defense Mechanism against Next Generation of Phishing Emails using Distilled Attention-Augmented BiLSTM, arXiv preprint arXiv:2602.22250, 2026.
- M. Rabbani, J. Gui, F. Nejati, Z. Zhou, A. Kaniyamattam, M. Mirani, G. Piya, I. V. Opushnyev, R. Lu, and A. A. Ghorbani, Device Identification and Anomaly Detection in IoT Environments, IEEE Internet of Things Journal, vol. 12, no. 10, pp. 13625–13643, 2025.
- M. Rabbani, J. Gui, Z. Zhou, F. Nejati, M. Mirani, G. Piya, I. V. Opushnyev, R. Lu, and A. A. Ghorbani, A Lightweight IoT Device Identification Using Enhanced Behavioral-Based Features, Peer-to-Peer Networking and Applications, vol. 18, no. 2, Art. no. 2, 2025.
- F. Nejati, H. Sajedi, and A. Zohourian, Fragile Watermarking Based on QR Decomposition and Fourier Transform, Wireless Personal Communications, vol. 122, no. 1, pp. 211–227, 2022.

- M. Mohammadi and F. Nejati, Watermarking Based on Hessenberg Matrix Decomposition, *Soft Computing*, vol. 9, no. 1, pp. 146–157, 2021, doi: 10.22052/scj.2021.242811.0.
- F. Nejati, H. Sajedi, and M. Mohammadi, Fragile Watermarking for Image Authentication Using QR Factorization and Fourier Transform, in *Proc. 2019 5th Int. Conf. Web Research (ICWR)*, Tehran, Iran, 2019, pp. 45–49, doi: 10.1109/ICWR.2019.8765292.