

CHAPTER 3 - RESEARCH METHODOLOGY

3.1 Research Design Overview

This research adopts a **hybrid, event-driven detection design** to address the unique challenges of fileless and Living-off-the-Land (LOTL) attacks in Windows enterprise environments. As discussed in Chapter 2, fileless and LOTL attacks rarely rely on persistent malware artifacts; instead, they manifest through sequences of legitimate system activities, script execution, and administrative tool abuse. Consequently, the research design emphasizes **behavioral observation, contextual correlation, and semantic representation** under real-time operational constraints.

The proposed methodology follows a **streaming detection paradigm**, where system telemetry events are collected, processed, and analyzed incrementally as they occur. Rather than performing batch or offline analysis, each incoming event is evaluated in near real time to enable timely detection while preserving execution context. This design choice aligns with the characteristics of LOTL attacks, which often unfold gradually and evade detection when events are analyzed in isolation.

To capture complementary aspects of malicious behavior, the research design integrates two analytical branches: **contextual behavior modeling** and **semantic command analysis**. Contextual modeling focuses on process-level relationships, execution duration, and interaction patterns by constructing an incremental process graph. These features capture *how* system activities are related over time, which is particularly relevant for identifying persistence, process chaining, and lateral execution patterns commonly observed in LOTL attacks. Semantic analysis, on the other hand, focuses on *what* actions are being executed by transforming normalized command strings into semantic embeddings that approximate command intent while remaining robust to syntactic variation and obfuscation.

The outputs of the contextual and semantic branches are combined through a **lightweight fusion model** based on a multilayer perceptron (MLP). This design intentionally avoids computationally intensive architectures, such as full graph neural networks or end-to-end transformer training, in order to satisfy real-time detection requirements. The fusion model produces a probabilistic maliciousness score for each evaluated event, enabling flexible threshold selection based on operational constraints.

Explainability is incorporated as a first-class design objective rather than a post-hoc addition. Feature attribution techniques are applied to the fusion model to provide insight into which contextual and semantic features contribute most significantly to each detection decision. This design supports analyst trust, facilitates model validation, and aligns with enterprise deployment requirements discussed in Chapter 2.

Overall, the research design prioritizes **practical detectability, interpretability, and operational feasibility**, while remaining grounded in behavioural and semantic analysis principles. The following sections detail each stage of the methodology, from telemetry acquisition and preprocessing to contextual modeling, semantic analysis, fusion-based inference, and explainable evaluation.

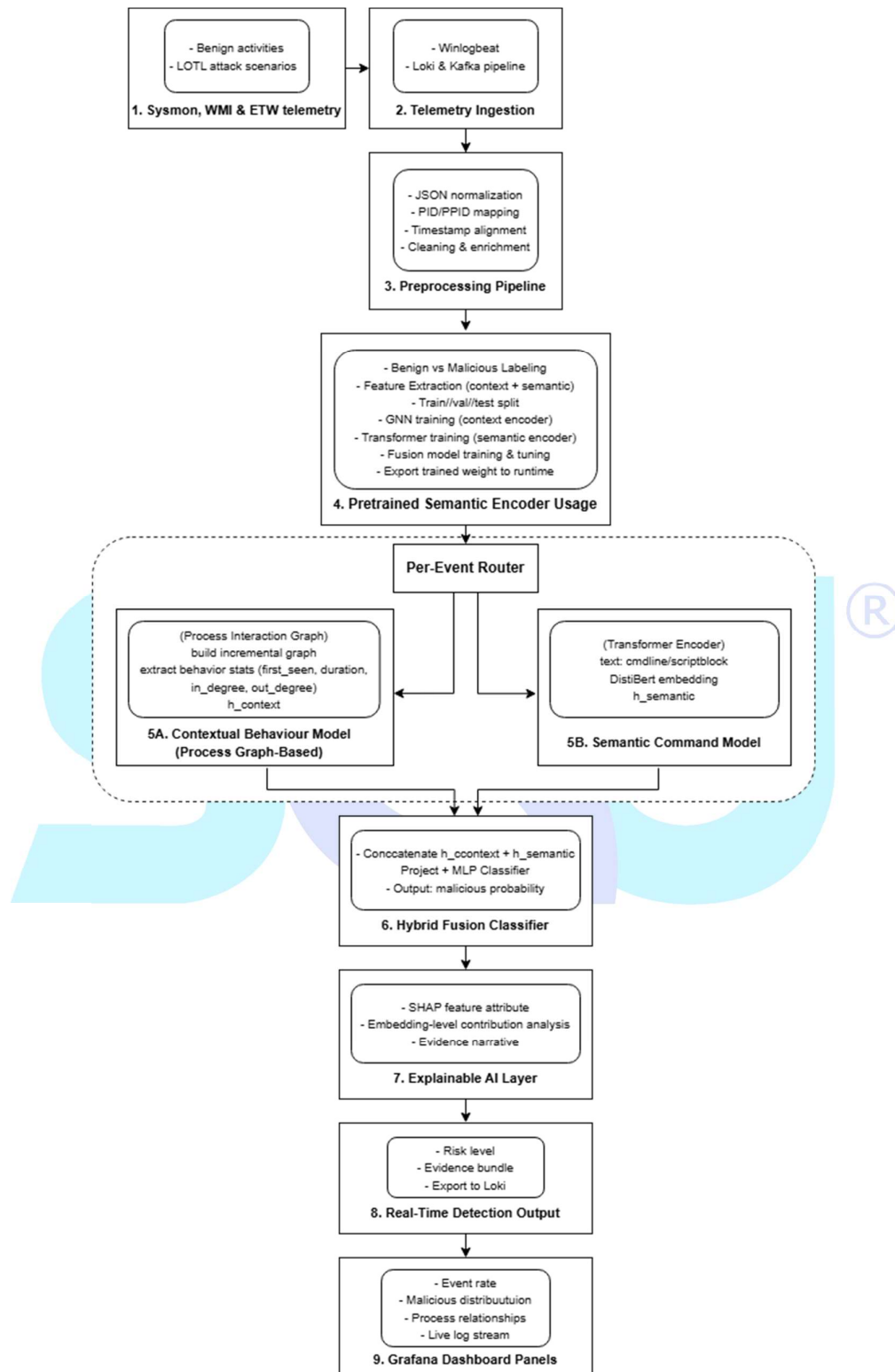


Figure 3.1.1 Architecture of the proposed research framework

Figure 3.1 presents framework begins with **live telemetry sources**, including Sysmon, Windows Management Instrumentation (WMI), and Event Tracing for Windows (ETW), which provide complementary visibility into process execution, command invocation, and low-level system behavior. These telemetry streams are ingested in real time through a streaming pipeline that supports continuous data processing.

After ingestion, incoming events undergo **preprocessing and normalization**, where heterogeneous telemetry formats are aligned into a unified schema. This stage ensures consistency across different data sources and prepares the events for downstream analysis. Following preprocessing, each event is routed dynamically based on its characteristics and analytical requirements.

The architecture then branches into two parallel analytical paths. The first path constructs an **incremental contextual behavior graph**, capturing process relationships, execution lineage, and interaction patterns across Sysmon, WMI, and ETW signals. This contextual modeling enables the system to represent long-term behavioral dependencies that are often indicative of LOTL activity. The second path performs **semantic command analysis**, focusing on the interpretation of executed commands through normalization, tokenization, and semantic embedding.

The outputs of both analytical paths are fused in a **hybrid inference module**, where contextual and semantic representations are combined to produce a malicious probability score in real time. Finally, an **Explainable AI (XAI) component** provides interpretability by highlighting influential features, attention weights, and contextual evidence, enabling analysts to understand why a particular event is classified as malicious or benign. Detection results are then exposed to downstream systems such as dashboards or SIEM platforms for monitoring and response.

3.2 Data Acquisition and Telemetry Sources

The effectiveness of detecting fileless and Living-off-the-Land (LOTL) attacks depends heavily on the availability of **high-fidelity system telemetry** that captures execution behavior beyond static file artifacts. Since fileless attacks frequently abuse legitimate operating system components, the proposed methodology relies on **native Windows telemetry sources** that expose process behavior, execution context, and management-related activities.

To achieve comprehensive behavioral visibility, this research utilizes three complementary telemetry sources: **Sysmon**, **Windows Management Instrumentation (WMI)**, and **Event Tracing for Windows (ETW)**. Each source provides a distinct perspective on system activity and collectively enables semantic and contextual analysis.

3.2.1 Sysmon Telemetry

System Monitor (Sysmon) is used as a primary telemetry source to capture detailed process-level events that are essential for identifying LOTL techniques. Sysmon provides enhanced visibility compared to default Windows event logging by recording fine-grained execution details.

In this research, Sysmon telemetry is leveraged to capture:

- Process creation events, including executable paths and command-line arguments
- Parent-child process relationships
- Process identifiers (PID) and parent process identifiers (PPID)
- Temporal execution information

These attributes are critical for constructing **execution chains** and **process lineage**, which are frequently abused in fileless attacks. For example, LOTL attacks often involve legitimate parent processes spawning scripting engines or command interpreters in atypical sequences. Sysmon telemetry enables the detection framework to identify such anomalies by analyzing process relationships rather than relying on binary signatures.

3.2.2 Windows Management Instrumentation (WMI)

Windows Management Instrumentation (WMI) is a powerful system management framework that is commonly abused by adversaries for remote execution, persistence, and lateral movement. Unlike traditional malware execution, WMI-based attacks often leave minimal forensic artifacts, making them particularly difficult to detect.

In this research, WMI telemetry is collected to capture:

- WMI process invocation and execution context

- Management-related commands executed via WMI interfaces
- Indicators of remote or automated administrative actions

WMI telemetry is especially relevant to fileless and LOTL attacks because it enables attackers to execute commands without dropping executable files on disk. By incorporating WMI telemetry into the detection framework, the system gains visibility into stealthy execution paths that would otherwise remain hidden from signature-based defenses.

3.2.3 Event Tracing for Windows (ETW)

Event Tracing for Windows (ETW) provides low-level runtime tracing of system and application behavior. ETW complements Sysmon and WMI by capturing execution traces that reflect **how** processes behave at runtime rather than merely **what** processes are executed.

In this research, ETW telemetry contributes:

- Runtime execution patterns
- Script block and execution traces
- Temporal signals related to command execution and process activity

ETW is particularly valuable for detecting obfuscated or encoded command execution, which is a common technique in fileless attacks. By correlating ETW traces with Sysmon and WMI events, the proposed framework can infer semantic intent even when commands are heavily obfuscated.

3.2.4 Multi-Source Telemetry Integration

A key methodological aspect of this research is the **integration of heterogeneous telemetry sources** rather than treating each source independently. Sysmon, WMI, and ETW events are ingested concurrently and aligned using timestamp normalization and entity correlation mechanisms.

This integration enables:

- Cross-validation of events across sources
 - Construction of richer behavioral context
-

- Reduction of false positives caused by isolated event analysis

By correlating telemetry at multiple layers of system behavior, the framework captures both **structural context** (e.g., process lineage) and **semantic meaning** (e.g., command intent), which are essential for detecting fileless and LOTL attacks.

3.2.5 Relevance to Fileless and LOTL Detection

The selected telemetry sources are intentionally aligned with the threat model addressed in this research. Fileless and LOTL attacks exploit trusted system components, making binary-based detection insufficient. Instead, detection must focus on **execution behavior, contextual relationships, and semantic intent**.

Sysmon, WMI, and ETW collectively provide the necessary signals to support this detection paradigm. Their integration within the proposed framework ensures that malicious misuse of legitimate tools can be identified based on behavior and context rather than static indicators.

The telemetry collected from these sources serves as the raw input to the detection framework. However, due to differences in data format, structure, and semantics, the collected telemetry must undergo preprocessing and normalization before analysis. The following section describes the telemetry ingestion pipeline and preprocessing mechanisms used to prepare the data for semantic and contextual modeling.

3.3 Telemetry Ingestion and Streaming Pipeline

The telemetry ingestion layer serves as the interface between raw endpoint event generation and downstream semantic analysis. Its primary role is to ensure that heterogeneous telemetry data from Sysmon, WMI, and ETW can be collected, buffered, and forwarded reliably without disrupting real-time constraints.

The detection of fileless and Living-off-the-Land (LOTL) attacks necessitates the capability to process system telemetry in a continuous and near-real-time manner. In contrast to conventional offline forensic analysis where event logs are examined retrospectively real-time detection architectures must ingest, parse, and analyze event streams as they are generated in order to facilitate early-stage identification of malicious behaviours. This requirement is particularly critical for fileless and LOTL attacks, whose operational timelines may be short and whose

artifacts are often transient or memory-resident. Delayed analysis can therefore result in missed detection opportunities, especially during early lifecycle phases such as initial execution, privilege escalation, or lateral movement.

To address these constraints, the proposed methodology adopts a stream-oriented telemetry ingestion pipeline explicitly designed to support low-latency processing, scalable event handling, and incremental analytical updates. Rather than treating telemetry as static log repositories, the architecture conceptualizes endpoint data as continuous event streams, enabling progressive correlation and behavioural modelling as attack patterns unfold. This design paradigm aligns with modern stream-processing principles, where computation is performed on data-in-motion, thereby reducing detection lag and improving responsiveness.

The telemetry ingestion layer functions as the critical interface between raw endpoint event generation and downstream semantic and behavioral analysis components. Its primary responsibility is to ensure that heterogeneous telemetry sources such as Sysmon process creation events, Windows Management Instrumentation (WMI) activity logs, and Event Tracing for Windows (ETW) streams can be collected, normalized, buffered, and forwarded in a reliable and fault-tolerant manner without violating real-time performance constraints. Given the differences in schema, granularity, and emission frequency across these sources, the ingestion layer must also perform lightweight preprocessing tasks, including timestamp harmonization, field standardization, and preliminary filtering of irrelevant noise.

Furthermore, to preserve scalability under high event throughput, the ingestion pipeline incorporates mechanisms for backpressure management, asynchronous queuing, and modular extensibility. This ensures that bursts of benign system activity do not degrade analytical performance or introduce unacceptable latency. By establishing a robust and stream-aware ingestion foundation, the proposed framework enables higher-level semantic reasoning and lifecycle-aware detection modules to operate on consistent, timely, and contextually enriched telemetry data, thereby strengthening the overall effectiveness of real-time LOTL attack detection.

The telemetry ingestion layer acts as the interface between raw endpoint event generation and downstream semantic analysis. Its primary function is to reliably collect, normalize, buffer, and

forward heterogeneous telemetry from sources such as Sysmon, WMI, and ETW without violating real-time constraints.

Detection of fileless and Living-off-the-Land (LOTL) attacks requires continuous, near-real-time processing of system telemetry. Unlike retrospective forensic analysis, real-time architectures must ingest and analyze event streams as they are generated to enable early identification of malicious behavior. This is particularly critical for fileless and LOTL attacks, which often have short operational timelines and transient artifacts. Delayed analysis may therefore result in missed detection during early stages such as execution, privilege escalation, or lateral movement.

To address these challenges, the proposed methodology adopts a stream-oriented ingestion pipeline designed for low-latency processing, scalable event handling, and incremental analysis. Telemetry is treated as continuous data streams rather than static logs, enabling progressive correlation and behavioral modeling as attacks evolve. This stream-processing paradigm reduces detection lag and improves responsiveness.

Given variations in schema, granularity, and emission frequency across telemetry sources, the ingestion layer performs lightweight preprocessing, including timestamp harmonization, field standardization, and noise filtering. To maintain scalability under high event throughput, the pipeline incorporates backpressure management, asynchronous queuing, and modular extensibility. This stream-aware foundation supports consistent and timely telemetry delivery, thereby enhancing the effectiveness of real-time LOTL attack detection.

3.3.1 Telemetry Collection Mechanism

Telemetry events are collected at the endpoint level using lightweight agents or native forwarding mechanisms. These collectors monitor relevant event sources and forward telemetry records as structured log entries. The design prioritizes minimal performance overhead on the monitored host to avoid interference with normal system operations.

Each telemetry event is transmitted immediately upon generation, preserving its original timestamp and contextual attributes. This approach ensures that the ingestion pipeline reflects the true temporal order of system activities, which is essential for subsequent contextual correlation and semantic interpretation.

3.3.2 Streaming and Logical Event Separation

Upon ingestion, telemetry events are processed through a streaming pipeline that supports **logical separation of event streams** based on their source and type. Rather than treating all telemetry as a single monolithic stream, the pipeline maintains distinct logical channels for Sysmon, WMI, and ETW events.

This logical separation provides several advantages:

- It preserves source-specific semantics during early processing stages.
- It enables independent buffering and flow control for different telemetry types.
- It facilitates selective routing of events to specialized analytical components.

Importantly, logical separation does not imply physical isolation or duplication of data. Instead, it represents a design abstraction that allows the pipeline to manage heterogeneous telemetry more effectively.

3.3.3 Real-Time Constraints and Buffering Strategy

The ingestion pipeline is architected to function within strict real-time constraints, recognizing that delays in telemetry acquisition and forwarding can directly diminish detection accuracy and timeliness. In the context of fileless and LOTL attacks where malicious activities may unfold rapidly and leave only ephemeral traces latency in event delivery can result in missed correlations between sequential behaviours. Therefore, the pipeline prioritizes responsiveness while maintaining operational stability under varying workloads.

To address the challenge of fluctuating telemetry rates, lightweight buffering mechanisms are incorporated to absorb short-term bursts of high event volume without introducing substantial processing delays. These buffers act as intermediate holding structures that decouple event generation from downstream analysis components, thereby preventing temporary spikes in telemetry—such as those caused by benign system updates or batch administrative operations—from overwhelming the analytical modules. By smoothing irregular input rates, buffering enhances system resilience without compromising near-real-time performance.

Importantly, the buffering strategy is intentionally designed to be both transient and bounded. Events are retained only for short durations sufficient to ensure orderly processing, and strict

capacity limits are enforced to prevent uncontrolled accumulation. This bounded design mitigates risks associated with memory exhaustion, prolonged queuing delays, and cascading backpressure effects. When thresholds are approached, adaptive flow-control mechanisms can prioritize critical security-relevant events or apply selective filtering to maintain system responsiveness.

Through this balanced approach, the ingestion layer achieves a principled trade-off between reliability and latency. It ensures that telemetry is delivered consistently to downstream semantic and behavioural analysis components while preserving the responsiveness required for early-stage detection. Consequently, the system remains capable of continuous operation even under dynamic and unpredictable telemetry conditions, reinforcing the robustness of the overall real-time LOTL detection framework.

3.3.4 Timestamp Preservation and Ordering

Accurate detection time measurement requires precise handling of event timestamps. Therefore, the ingestion pipeline preserves original event timestamps and associates them with ingestion timestamps upon arrival. This dual-timestamp approach enables the framework to distinguish between:

- Event occurrence time, and
- Event processing time.

Maintaining this distinction is critical for computing detection latency and evaluating the real-time performance of the proposed framework.

3.3.5 Interface with Downstream Processing

Following ingestion and the application of transient buffering, telemetry events are forwarded to the preprocessing and normalization stage. At this boundary, events remain explicitly source-aware, preserving metadata that identifies their origin (e.g., Sysmon, WMI, or ETW). Maintaining source provenance at this stage is important for two reasons: first, it supports routing decisions in later stages, where different event types may require specialized parsing or feature extraction; second, it enables more accurate semantic interpretation, as the same

behavioural concept may be represented differently depending on the telemetry source and its logging semantics.

The ingestion pipeline is also intentionally decoupled from downstream analytical components in order to preserve architectural modularity. By separating concerns, the system ensures that the ingestion mechanism remains stable and reusable even as detection logic, semantic modelling strategies, or feature engineering methods are revised over time. This modular design reduces implementation coupling and allows preprocessing, semantic enrichment, and detection modules to evolve independently, thereby supporting maintainability and extensibility in long-term research and operational deployments.

Although the telemetry ingestion pipeline provides reliable and low-latency delivery of heterogeneous event streams, the collected data at this stage remains largely unstructured, inconsistent in schema, and highly source-specific. As a result, it is not yet suitable for higher-level semantic reasoning or cross-source correlation. To enable robust contextual analysis, telemetry events must first be transformed into a unified and standardized representation that supports consistent feature extraction, lifecycle-aware interpretation, and downstream modelling.

Accordingly, the next section describes the preprocessing and normalization procedures employed to align, clean, and standardize telemetry data across sources prior to semantic analysis. This includes resolving schema inconsistencies, harmonizing timestamps and identifiers, and constructing a consistent event representation that facilitates subsequent detection and interpretability objectives.

3.4 Preprocessing and Normalization

Raw telemetry collected from Sysmon, WMI, and ETW is inherently heterogeneous with respect to structural format, semantic meaning, and event granularity. Each source captures system activity through distinct logging mechanisms, resulting in differences in schema design, field naming conventions, timestamp precision, and the level of contextual detail recorded. For example, Sysmon provides rich process and network activity traces with relatively standardized identifiers, whereas WMI telemetry often reflects higher-level management operations, and ETW can produce high-frequency, low-level traces depending on the providers enabled. In the absence of systematic preprocessing, attempting to directly analyze these heterogeneous

streams would produce inconsistent interpretations, elevated noise, and reduced analytical reliability, ultimately degrading detection performance.

Accordingly, preprocessing and normalization represent a critical stage within the proposed methodology. This stage serves as the foundation for all subsequent semantic reasoning and detection components by ensuring that telemetry is coherent, comparable, and suitable for cross-source correlation. It mitigates the risk that model outputs reflect logging artifacts rather than genuine behavioural signals, which is particularly important in fileless and LOTL contexts where detection depends heavily on subtle behavioural deviations rather than overt malicious binaries.

The primary objective of preprocessing and normalization is to transform heterogeneous telemetry into a unified, semantically consistent representation that supports downstream contextual modeling and semantic analysis. This transformation involves aligning schemas across sources, standardizing field formats, resolving inconsistent identifiers, and cleaning events to remove redundancies or irrelevant noise. By constructing a normalized event representation, the framework enables later analytical modules to reason over telemetry in a source-agnostic yet semantically meaningful manner, thereby improving the robustness and generalizability of the proposed detection approach.

3.4.1 Schema Alignment and Unified Representation

The first preprocessing step involves aligning all incoming telemetry events to a **unified JSON-based schema**. This schema standardization ensures that common attributes such as timestamps, host identifiers, process identifiers, and execution context are represented consistently across Sysmon, WMI, and ETW events.

During schema alignment:

- Source-specific field names are mapped to standardized attributes.
- Missing or optional fields are handled explicitly to avoid ambiguity.
- Metadata identifying the original telemetry source is preserved.

This unified representation enables subsequent analytical components to operate independently of the original event format while still retaining source-level context when needed.

3.4.2 Timestamp Normalization and Temporal Alignment

Accurate temporal reasoning is essential for detecting fileless and LOTL attacks, which often rely on rapid sequences of actions. As different telemetry sources may record timestamps using different formats or resolutions, timestamp normalization is performed to align all events to a common time reference.

Each event is assigned:

- An **event timestamp**, representing the original occurrence time.
- An **ingestion timestamp**, representing the time at which the event enters the processing pipeline.

This dual-timestamp strategy allows precise reconstruction of event sequences and supports reliable measurement of detection latency, which is a key evaluation metric in this research.

3.4.3 Entity Correlation and Contextual Mapping

To support contextual behavior modeling, preprocessing includes the correlation of entities across events. In particular, process-related attributes such as process identifiers (PID), parent process identifiers (PPID), and user or session identifiers are extracted and mapped.

This entity correlation enables:

- Reconstruction of process lineage and execution chains.
- Identification of anomalous parent–child relationships.
- Aggregation of events belonging to the same execution context.

By establishing these correlations early in the pipeline, the framework prepares telemetry data for incremental contextual graph construction in subsequent stages.

3.4.4 String Normalization and Command Decoding

Fileless and LOTL attacks frequently employ obfuscation techniques, such as encoded command strings, to evade detection. To mitigate this challenge, preprocessing includes normalization and decoding of string-based fields related to command execution.

This step involves:

- Canonicalization of command-line strings.
- Decoding of encoded or escaped command segments where applicable.
- Removal of irrelevant or non-informative tokens.

The resulting normalized command representation preserves semantic meaning while reducing noise, thereby enhancing the effectiveness of semantic command analysis.

3.4.5 Noise Reduction and Event Filtering

Not all telemetry events contribute equally to the detection of fileless and Living-off-the-Land (LOTL) attacks. In practice, raw endpoint telemetry contains a substantial volume of benign and repetitive activity that is unrelated to adversarial behavior. Consequently, the preprocessing stage incorporates selective filtering mechanisms to reduce noise, control computational overhead, and improve the overall tractability of downstream analysis. Events that are empirically known to be irrelevant, excessively redundant, or weakly informative for fileless and LOTL detection are deprioritized or excluded prior to semantic enrichment and contextual modeling.

This filtering strategy improves system efficiency while preserving sufficient behavioral coverage for meaningful detection. By reducing the number of low-value events passed downstream, the framework enables analytical modules to focus on telemetry patterns that are more likely to reflect suspicious activity, such as anomalous process execution chains, misuse of scripting engines, unusual management instrumentation activity, or atypical ETW traces. In addition, limiting noise at this stage supports improved model stability, as it reduces the likelihood that detection outputs are dominated by benign operational artifacts.

Importantly, filtering rules are designed conservatively in order to avoid prematurely discarding potentially significant evidence. Given that LOTL attacks intentionally mimic legitimate administrative behavior, overly aggressive filtering could remove subtle but important signals required for identifying malicious intent. Therefore, the framework prioritizes retaining events that may become relevant through later contextual correlation, even if they appear benign when evaluated in isolation.

Following preprocessing and normalization, telemetry events are transformed into a structured and semantically aligned representation suitable for advanced analysis. At this point, the framework proceeds to route normalized events into specialized analytical branches based on their characteristics, source provenance, and semantic category. The next section describes the event routing and analytical branching strategy employed in the proposed framework, outlining how differentiated processing paths enable more precise semantic modeling and lifecycle-aware detection.

3.5 Event Routing and Analytical Branching

After preprocessing and normalization, telemetry events are not processed uniformly by a single analytical model. Instead, the proposed framework employs a **deterministic event routing mechanism** that directs each normalized event to the most appropriate analytical branch based on its characteristics. This design decision is motivated by the heterogeneous nature of fileless and LOTL attack signals, which manifest differently across behavioral context and command semantics.

The primary objective of event routing is to ensure that each analytical component receives **relevant and semantically meaningful input**, thereby improving detection accuracy while maintaining real-time performance.

3.5.1 Motivation for Analytical Branching

Fileless and LOTL attacks often combine multiple behavioral aspects, such as unusual process relationships, suspicious command execution, and abnormal runtime patterns. Treating all events identically would dilute these signals and increase computational overhead.

By separating analysis into specialized branches, the framework is achieved:

- Improved signal-to-noise ratio for each model
- Reduced Unnecessary Computation
- Clear separation between contextual behavior and semantic intent

This branching strategy reflects the observation that **contextual relationships** and **command semantics** represent distinct but complementary dimensions of malicious behavior.

3.5.2 Deterministic Routing Logic

Event routing is implemented as a **rule-based and deterministic dispatcher**, rather than a learned model. Routing decisions are based on explicit event attributes, such as:

- Telemetry source (Sysmon, WMI, or ETW)
- Presence of command-line or script-related fields
- Event type indicating process relationships or execution flow

For example:

- Events representing process creation, parent–child relationships, or execution chains are routed to the **contextual behavior modeling branch**.
- Events containing command strings, script content, or execution parameters are routed to the **semantic command analysis branch**.

This deterministic approach ensures predictability and transparency, which are important for both experimental reproducibility and explainability.

3.5.3 Contextual Behavior Branch

The contextual behaviour branch is designed to model the structural and temporal relationships among system entities, with particular emphasis on process interactions, execution sequences, and inter-component dependencies. Rather than evaluating events in isolation, this branch aggregates routed telemetry into an incremental behavioural representation that captures how processes, scripts, services, and other system components interact over time. By constructing a continuously updated view of system activity, the framework enables the identification of patterns that may not be evident at the single-event level but emerge through relational and sequential analysis.

Events assigned to this branch contribute to the development of a dynamic interaction model, which may include parent–child process relationships, command invocation sequences, privilege transitions, and cross-process communication patterns. Temporal ordering plays a critical role in this modelling process, as LOTL attacks frequently exploit legitimate tools in carefully orchestrated stages. By preserving execution chronology, the system can detect

deviations from typical operational workflows, such as unusual process ancestry chains, unexpected script-to-binary execution flows, or atypical invocation contexts for administrative utilities.

This branch is particularly effective in identifying LOTL techniques that depend on abnormal execution chains or the strategic misuse of trusted parent processes. For example, adversaries may launch scripting engines from uncommon parent processes, chain multiple built-in utilities to achieve stealthy lateral movement, or embed malicious logic within legitimate administrative workflows. Such behaviours often evade signature-based detection because each individual action appears benign. However, when analyzed within a contextual and temporal framework, the structural anomalies become more apparent.

By emphasizing relational and lifecycle-aware modelling, the contextual behaviour branch enhances the framework's ability to detect stealth-driven, multi-stage attack strategies that blend into normal system operations. This structural perspective complements other analytical branches by providing a foundation for reasoning about how seemingly legitimate events combine to form malicious behavioural narratives.

3.5.4 Semantic Command Analysis Branch

The semantic command analysis branch processes events that carry command-level information. These events are analyzed to infer the **intent and meaning** of executed commands, even when obfuscation or encoding techniques are used.

Routing command-related events to a dedicated semantic branch allows the framework to apply specialized normalization and representation techniques that would be inefficient or ineffective if applied to all telemetry indiscriminately.

3.5.5 Advantages of the Routing Strategy

The event routing and analytical branching strategy offers several methodological advantages:

- It supports modular system design, enabling independent evolution of analytical components.
 - It improves detection effectiveness by aligning analysis methods with event characteristics.
-

- It enhances explainability by clearly associating detection outcomes with specific analytical paths.

Importantly, routing does not preclude cross-branch correlation. The outputs of both analytical branches are later fused to produce a unified detection decision.

Once telemetry events have been routed to their respective analytical branches, the framework performs contextual behavior modeling and semantic command analysis in parallel. The following section describes the contextual behavior modeling approach used to represent execution relationships and behavioral patterns indicative of fileless and LOTL attacks.

3.6 Contextual Behavior Modeling

Contextual behavior modeling is designed to capture execution-level characteristics of system activity that are indicative of fileless and Living-off-the-Land (LOTL) attack behavior. As discussed in Chapter 2, LOTL attacks rarely manifest through single malicious events; instead, they exhibit behavioral patterns such as prolonged execution, process chaining, and repeated invocation of legitimate system utilities. To capture these characteristics, this research models system activity uses an **incremental process interaction graph** and extracts lightweight contextual features from each process node.

3.6.1 Contextual Graph Construction

To model execution context and process relationships in real time, system activity is represented as an **incremental process interaction graph**. This graph-based representation is designed to capture the structural and temporal dependencies between system processes that are commonly exploited in fileless and Living-off-the-Land (LOTL) attacks.

Formally, the contextual graph is defined as a directed graph

$$G = (V, E),$$

where each node $v \in V$ corresponds to a unique process instance, identified by a combination of process identifier (PID), parent process identifier (PPID), executable image, and creation timestamp. Each directed edge $e = (v_i, v_j) \in E$ represents a behavioral relationship in which process v_i directly triggers or spawns process v_j .

Graph construction is performed **incrementally** as telemetry events are ingested. When a process creation or execution-related event is observed, a corresponding node is added to the graph if it does not already exist. Directed edges are created based on parent–child relationships derived from telemetry sources such as Sysmon, WMI, and ETW. This incremental construction strategy avoids full graph recomputation and enables continuous updates under real-time constraints.

In addition to structural relationships, temporal information is preserved by associating each node with event timestamps and execution duration metadata. This allows the graph to reflect not only which processes are connected, but also **when** these interactions occur. Such temporal preservation is essential for detecting LOTL behaviours that manifest through delayed execution, prolonged scripting sessions, or staged command invocation.

The resulting process interaction graph captures key contextual characteristics of LOTL attacks, including execution chaining (e.g., powershell → cmd → wmic), repeated invocation of legitimate utilities, and extended process lifetimes. By maintaining lineage and temporal ordering, the contextual graph provides a compact yet expressive representation of system behavior suitable for downstream feature extraction and real-time analysis.

3.6.2 Feature Representation of Nodes and Edges

Following the construction of the contextual process interaction graph, behavioral information is encoded through **explicit node-level contextual features** rather than direct graph learning. This design choice reflects the objective of balancing interpretability, computational efficiency, and real-time feasibility, while preserving behaviorally meaningful signals relevant to fileless and Living-off-the-Land (LOTL) attacks.

A. Node-Level Feature Representation

Each process node p_i in the contextual graph is represented by a fixed-length contextual feature vector derived from its execution behavior and graph connectivity. The contextual feature vector is defined as:

$$\mathbf{h}_{context}(p_i) = [t_{first}, t_{duration}, deg_{in}, deg_{out}]$$

where:

- t_{first} denotes the relative time at which the process is first observed within the execution session,
- $t_{duration}$ represents the total lifetime of the process,
- deg_{in} corresponds to the number of incoming edges, indicating how many processes trigger or reference the process,
- deg_{out} corresponds to the number of outgoing edges, indicating how many child processes are spawned.

These features are selected based on their alignment with documented LOTL behavioral patterns:

- **Temporal emergence (first_seen):** LOTL attacks often occur during post-compromise phases rather than at system startup, making the timing of process appearance a relevant contextual indicator.
- **Process persistence (duration):** Script interpreters and interactive shells used in LOTL attacks tend to remain active longer than typical benign processes, especially when orchestrating multiple actions.
- **Execution dependency (in_degree):** While many LOTL processes have a single parent, variations in triggering behavior provide additional context regarding execution flow.
- **Execution fan-out (out_degree):** LOTL attacks frequently involve chaining multiple legitimate utilities, resulting in higher fan-out as processes spawn additional child processes.

By explicitly encoding these features, the model captures **how processes behave and interact over time**, rather than relying solely on static attributes or raw event counts.

B. Edge Semantics and Their Role

Edges in the contextual graph represent **behavioral relationships**, such as parent–child process creation or execution triggering. While edges are critical for preserving execution lineage and contextual structure, they are not directly encoded as numerical features in the proposed

approach. Instead, their influence is implicitly captured through node-level degree features (in_degree and out_degree).

This design avoids the complexity of edge-level feature engineering or graph neural message passing, which can introduce significant computational overhead and reduce explainability in real-time detection settings. By abstracting edge information into interpretable node statistics, the approach maintains contextual awareness while remaining lightweight and suitable for streaming analysis.

C. Rationale for Explicit Feature Design

The use of explicit contextual features aligns with the objectives identified in Chapter 2: enabling real-time detection, preserving interpretability, and supporting explainability through feature attribution methods. Because each feature corresponds to a concrete behavioral property, downstream analysis such as SHAP-based feature attribution can directly relate model decisions to observable system behavior.

Overall, the feature representation strategy captures essential LOTL behavioral cues while avoiding unnecessary model complexity, providing a robust contextual foundation for subsequent semantic fusion and classification.

3.6.3 Example of Contextual Feature Computation

To illustrate how contextual features are computed from the process interaction graph, this subsection presents a concrete example based on an observed process execution. The example demonstrates how raw execution telemetry is transformed into a contextual feature vector that reflects LOTL-relevant behavioral characteristics.

Consider a PowerShell process instance observed during execution monitoring. The process is created by a parent process and remains active while spawning multiple child processes to invoke system utilities. Based on the incremental process interaction graph, the following contextual features are extracted for this process node.

Table 3.1 Contextual Feature Computation

Feature	Value	Description

first_seen	12.3 s	Time of first observation relative to session start
duration	184.5 s	Total lifetime of the process
in_degree	1	Number of incoming edges (parent process)
out_degree	6	Number of child processes spawned

Table 3.1 illustrates an example of contextual feature computation derived from the incremental process interaction graph. Each feature represents a specific execution-level characteristic of a process that is relevant for identifying fileless and Living-off-the-Land (LOTL) attack behaviour. Unlike static attributes, these features capture temporal and structural properties of process execution within its surrounding context.

The first_seen feature indicates the relative time at which the process is first observed during an execution session. In LOTL scenarios, malicious processes often appear after an initial delay rather than during system startup, reflecting post-compromise activity. This temporal positioning provides useful context for distinguishing benign background processes from suspicious late-stage execution.

The duration feature measures the total lifetime of the process. LOTL-related processes such as scripting engines or interactive shells often remain active for extended periods while orchestrating multiple actions. As a result, longer execution durations can indicate persistence and command staging behavior commonly associated with LOTL techniques.

The in_degree feature represents the number of incoming edges in the contextual graph, corresponding to how many processes directly trigger or invoke the process. Most LOTL processes have a single parent process; however, variations in this feature help capture differences in execution flow and dependency structure.

The out_degree feature captures the number of child processes spawned by the process. Elevated out-degree values are indicative of execution chaining, where a single process coordinates the invocation of multiple legitimate system utilities. This behavior is a hallmark of LOTL attacks that rely on tool chaining rather than standalone malicious binaries.

Together, these contextual features form a compact yet expressive representation of process behaviour. By combining timing, persistence, and interaction structure, the contextual feature

vector provides a behaviorally meaningful summary that complements semantic command analysis in the hybrid detection framework.

The resulting contextual feature vector for the process p_i is defined as:

$$\mathbf{h}_{context}(p_i) = [12.3, 184.5, 1, 6]$$

This vector reflects behavioural patterns commonly associated with LOTL activity. The relatively long execution duration indicates process persistence, while the elevated out-degree suggests execution chaining across multiple legitimate utilities. The delayed appearance of the process further aligns with post-compromise behaviour rather than benign system initialization.

This example demonstrates how simple yet interpretable contextual features can capture execution characteristics that are difficult to identify through isolated event analysis. The computed feature vector is subsequently normalized and transformed into a contextual embedding, as described in the following subsection.

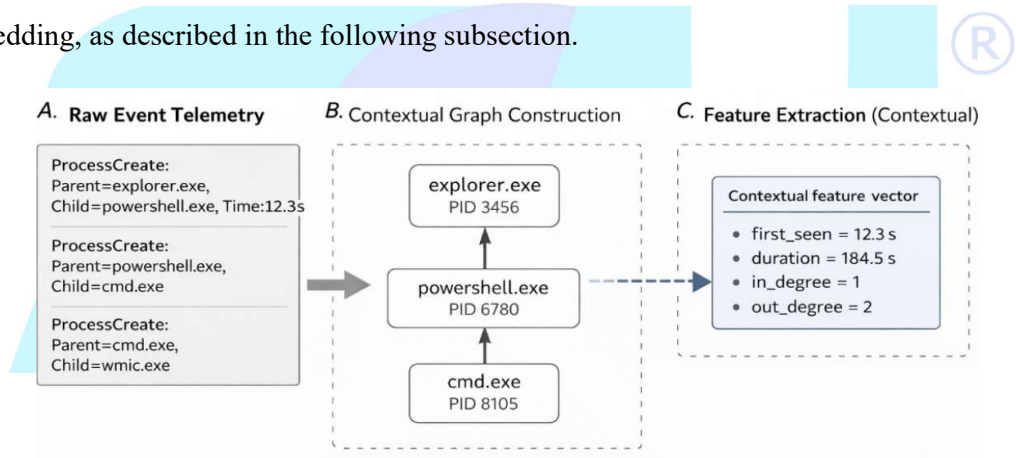


Figure 3.6.1 Example of Contextual Graph Construction from Raw Event Telemetry

Figure 3.2 illustrates an example of how raw execution telemetry is transformed into a contextual process interaction graph. A sequence of system events is incrementally mapped into process nodes and parent–child edges, preserving execution lineage and temporal relationships. This graphical representation provides the structural foundation for contextual feature extraction described in the preceding sections.

3.6.4 Contextual Anomaly Intuition

Although the proposed framework does not employ a standalone anomaly detection model, the extracted contextual features provide a strong **behavioural intuition for anomalous execution patterns** commonly associated with fileless and Living-off-the-Land (LOTL) attacks. This intuition arises from deviations in execution timing, process persistence, and interaction structure, rather than from explicit anomaly scoring.

In benign workloads, most processes exhibit short lifetimes, limited fan-out, and predictable parent–child relationships. System utilities are typically invoked in isolated or narrowly scoped contexts and terminate shortly after completing their intended function. In contrast, LOTL-related activity often manifests through prolonged execution, delayed initiation, and extensive chaining of legitimate tools, resulting in contextual feature patterns that differ markedly from baseline behaviour.

For example, a long-running scripting process that spawns multiple child processes over an extended period may not violate any single rule or signature. However, when viewed through the contextual feature space, such behaviour reflects a combination of persistence (duration), execution orchestration (out_degree), and late-stage activity (first_seen) that collectively suggests malicious intent. These deviations form the basis of the contextual anomaly intuition leveraged by the framework.

Importantly, no fixed thresholds are applied to individual contextual features. Instead, contextual information is forwarded to the hybrid fusion classifier, which learns discriminative decision boundaries from labeled data. This approach avoids brittle threshold-based heuristics and enables adaptation to environment-specific behaviour while preserving sensitivity to LOTL execution patterns.

By providing a structured yet flexible representation of execution context, the contextual modelling component enables early identification of suspicious behaviour that may otherwise remain indistinguishable at the individual event level. This intuition complements semantic analysis by highlighting *how* system activities unfold over time, rather than focusing solely on *what* commands are executed.

3.6.5 Role of Contextual Modeling in LOTL Detection

Contextual behaviour modelling plays a critical role in the detection of Living-off-the-Land (LOTL) attacks by capturing execution-level characteristics that are not observable through isolated events or command content alone. LOTL techniques deliberately abuse legitimate system utilities in ways that appear benign at the individual process or command level, making context-aware analysis essential for effective detection.

The contextual modelling component provides visibility into **how processes behave over time**, including their execution order, persistence, and interaction patterns. By modelling process lineage and extracting timing- and structure-based features, the framework is able to identify suspicious execution flows such as prolonged scripting sessions, repeated invocation of administrative tools, and multi-stage orchestration of legitimate binaries. These behaviours are strongly associated with LOTL attack techniques documented in real-world incident analyses.

Importantly, contextual modelling does not attempt to directly infer malicious intent. Instead, it functions as an **execution behavior descriptor**, highlighting deviations from typical process lifecycles and interaction patterns. This role is particularly valuable in LOTL scenarios, where malicious activity often blends seamlessly with normal administrative operations and cannot be distinguished through syntax-based or signature-based inspection.

Within the proposed framework, contextual modelling serves as the first analytical branch that filters and characterizes execution behaviour before semantic interpretation. While contextual features alone may be insufficient to conclusively classify an activity as malicious, they provide essential signals that narrow the detection scope and reduce ambiguity. When combined with semantic command analysis in the fusion stage, contextual modelling enables the detection of LOTL attacks that evade traditional rule-based, frequency-based, and single-modality detection approaches.

Overall, the contextual modelling component establishes a foundational understanding of system behaviour, ensuring that detection decisions are informed not only by *what* commands are executed, but also by *when*, *how*, and *in what execution context* they occur.

3.7 Semantic Command Analysis

Semantic command analysis is designed to capture the **intent and meaning** of command-line and script-based activity commonly abused in fileless and Living-off-the-Land (LOTL) attacks. While contextual modelling focuses on *how* processes behave and interact, semantic analysis focuses on *what* actions are being performed through command content.

LOTL attacks heavily rely on legitimate system utilities such as PowerShell, Command Prompt, WMIC, and scripting engines to execute malicious actions without deploying executable files. At the syntactic level, these commands often resemble benign administrative operations. However, their semantic structure such as encoded payloads, suspicious parameter combinations, or execution patterns can reveal malicious intent that is not visible through process-level context alone.

In the proposed framework, semantic command analysis operates as a parallel analytical branch that processes normalized command strings extracted from telemetry events. Commands are transformed into numerical representations using a transformer-based semantic encoder, enabling the model to capture latent relationships between tokens, execution patterns, and malicious behaviours across different LOTL techniques.

By separating semantic interpretation from contextual behaviour modelling, the framework avoids overloading a single model with heterogeneous information. Instead, semantic and contextual representations are learned independently and later combined through a hybrid fusion mechanism. This design supports both interpretability and real-time feasibility while addressing key limitations identified in existing fileless detection approaches.

3.7.1 Command Extraction and Normalization

Semantic command analysis begins with the extraction of command-line and script execution content from system telemetry events. In fileless and Living-off-the-Land (LOTL) attacks, malicious actions are frequently executed through legitimate system utilities such as PowerShell, Command Prompt, WMIC, and scripting engines without deploying persistent executable files. As a result, command content provides critical insight into attacker intent that cannot be inferred from process metadata alone.

Relevant command data is extracted from process creation and script execution events captured through telemetry sources including Sysmon, WMI, and ETW. These events typically contain raw command strings that may include encoded payloads, obfuscated parameters, environment-specific values, or execution flags designed to evade signature-based detection. The extracted command strings are treated as unstructured textual input and processed independently from contextual behavior modeling.

Prior to semantic representation, extracted commands undergo a **normalization process** to reduce syntactic noise while preserving semantic structure. The normalization process consists of the following steps:

1. **Canonicalization:** All command strings are converted to lowercase to ensure consistency across different execution formats.
2. **Noise reduction:** Redundant whitespace, non-informative flags, and execution artifacts that do not affect semantic meaning are removed.
3. **Environment abstraction:** Host-specific values such as absolute file paths, usernames, IP addresses, and temporary directories are replaced with generic placeholders.
4. **Decoding of encoded payloads:** When applicable, encoded command segments (e.g., Base64-encoded PowerShell payloads) are decoded to recover their underlying semantic structure.
5. **Token normalization:** Semantically equivalent tokens and aliases are mapped to a unified representation (e.g., `iex` → `invoke-expression`).

The objective of command normalization is **not full deobfuscation or payload recovery**, but rather the preservation of semantic intent while minimizing superficial syntactic variation. This design enables semantically similar commands such as different encoded variants of PowerShell download-and-execute behavior to be represented consistently in downstream semantic analysis.

Example of Command Extraction and Normalization

Consider the following raw command extracted from a process creation event:

powershell.exe -NoP -Enc
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIAABTAHkAcwB0AGUAbQAuAE4AZQ
B0AC4AVwBIAQIAQwBsAGkAZQBwAHQAQKQAuAEQAAbwB3AG4AbABvAGEAZABTAHQAcgB
pAG4AZwAoACIAaAB0AHQAcaAA6AC8ALwAuAC4ALwAiACkA

After normalization and decoding, the command is transformed into a simplified semantic representation:

`powershell invoke-expression download-string`

This normalized form abstracts away encoding artifacts and execution flags while preserving the semantic intent of remote code retrieval and execution an operation commonly associated with LOTL attack techniques.

By applying this normalization pipeline, the semantic analysis component ensures that command representations are robust to obfuscation and syntactic variation, enabling effective semantic embedding and downstream fusion with contextual behavioural features.

3.7.2 Semantic Embedding Representation

After command extraction and normalization, each normalized command string is transformed into a numerical representation that captures its semantic characteristics. This transformation is achieved using a **pretrained transformer-based semantic encoder**, which maps textual command content into a fixed-length embedding vector. The embedding encodes latent semantic relationships between tokens, command structures, and execution patterns commonly observed in both benign administrative activity and Living-off-the-Land (LOTL) attacks.

Formally, for a normalized command c_i , the semantic embedding is defined as:

$$\mathbf{e}_{semantic}(c_i) = f_{enc}(c_i),$$

where $f_{enc}(\cdot)$ denotes the pretrained semantic encoder and

$$\mathbf{e}_{semantic}(c_i) \in \mathbb{R}^d$$

is the resulting embedding vector of dimensionality d . In this research, the embedding dimensionality is fixed (e.g., $d = 768$), consistent with common transformer-based sentence encoders.

The semantic encoder is used **as a fixed feature extractor** and is not fine-tuned during model training. This design choice is motivated by three considerations. First, it reduces computational overhead and supports real-time inference. Second, it mitigates overfitting risks given the limited size of labelled LOTL datasets. Third, it preserves general semantic representations learned from large-scale corpora, enabling robust generalization across diverse command variants.

Each dimension of the resulting embedding vector represents a latent semantic feature and is treated as an independent input feature for downstream fusion. For implementation and interpretability purposes, individual dimensions are referenced as `sem_0`, `sem_1`, ..., `sem_d`. These semantic features are later analyzed using explainability techniques to identify dimensions that contribute most strongly to malicious classification decisions.

To ensure numerical stability and compatibility with contextual features during fusion, semantic embeddings are optionally normalized using standard scaling techniques. The normalized semantic embedding vector provides a compact and expressive representation of command intent that complements execution-level contextual features.

Importantly, the semantic embedding does not encode explicit rules or signatures. Instead, it captures similarity in command intent, allowing syntactically different but semantically related commands such as multiple variants of encoded PowerShell download-and-execute behaviour to be mapped to nearby regions in the embedding space. This property is particularly valuable for detecting obfuscated LOTL attacks that evade string-based or signature-based detection methods.

3.7.3 Example of Semantic Feature Computation

To illustrate how semantic features are computed from normalized command content, this subsection presents a concrete example of semantic embedding generation. The example demonstrates how a normalized LOTL-related command is transformed into a high-dimensional numerical representation suitable for downstream classification and explainability analysis.

Consider the following normalized command string obtained after command extraction and normalization:

powershell invoke-expression download-string
--

This command represents a common LOTL technique involving the retrieval and execution of remote content using a legitimate scripting utility. The normalized command is passed through the pretrained semantic encoder described in Section 3.7.2, producing a fixed-length semantic embedding vector.

Formally, the resulting semantic embedding is expressed as:

$$\mathbf{e}_{\text{semantic}} = [e_1, e_2, e_3, \dots, e_d],$$

where each element e_k corresponds to a latent semantic feature learned by the encoder, and d denotes the embedding dimensionality.

Sample Semantic Feature Vector (Partial Representation)

Table 3.2 presents a partial view of the resulting semantic embedding, highlighting selected dimensions that later exhibit high importance in explainability analysis.

Table 3.2 Sample Semantic Feature Vector

Feature Name	Value
sem_12	0.12
sem_47	-0.08
sem_282	0.34
sem_473	0.21
sem_611	0.27

Table 3.2 presents a partial example of the semantic feature vector generated from a normalized command using the pretrained semantic encoder. Each feature labelled as sem_xxx corresponds to a single dimension of the high-dimensional semantic embedding vector produced by the transformer-based model. The numerical values represent latent semantic activations rather than manually engineered features or predefined rules.

It is important to note that individual semantic features (e.g., `sem_12`, `sem_282`) do not have explicit human-interpretable meanings. Instead, they collectively encode semantic relationships learned by the encoder from large-scale textual data. Commands with similar intent such as different variants of PowerShell download-and-execute behaviour tend to produce embedding vectors with similar activation patterns across these dimensions.

The values shown in the table illustrate how a single command instance is transformed into a numerical representation suitable for downstream classification. For example, higher absolute values in certain semantic dimensions may indicate stronger association with execution patterns or command structures commonly observed in LOTL attacks. These semantic features are later combined with contextual behavioural features through the hybrid fusion model.

In subsequent analysis, explainability techniques such as SHAP are applied to the fused model to identify which semantic dimensions contribute most significantly to malicious classification decisions. This allows semantic importance to be assessed at the feature level without requiring direct interpretation of individual embedding dimensions.

While individual embedding dimensions do not correspond to human-interpretable rules, their relative values encode semantic similarity and execution intent. In subsequent analysis, explainability techniques such as SHAP are applied to identify which semantic dimensions contribute most strongly to malicious classification decisions.

This example demonstrates how complex command semantics are compactly represented as numerical features, enabling effective integration with contextual behavioural features in the hybrid fusion model.

3.7.4 Role of Semantic Analysis in LOTL Detection

Semantic analysis plays a critical role in detecting Living-off-the-Land (LOTL) attacks by providing insight into the **intent and operational meaning** of command-line and script-based activity. In LOTL scenarios, attackers deliberately leverage legitimate system utilities to execute malicious actions, causing individual commands to appear syntactically benign and indistinguishable from normal administrative usage when examined in isolation.

While contextual modelling captures *how* processes behave over time, semantic analysis addresses *what* actions are being performed. This distinction is particularly important for LOTL

detection, as many attack techniques involve specific command structures such as remote content retrieval, credential access, lateral movement, or system reconnaissance that cannot be reliably identified through process metadata alone.

Semantic embeddings enable the detection framework to recognize similarity in command intent across syntactically different executions. For example, multiple variants of encoded PowerShell commands may differ in flags, encoding schemes, or string structure, yet share a common semantic objective. By mapping such commands into a shared embedding space, the semantic analysis component enables generalization across obfuscated and polymorphic LOTL techniques without relying on fragile string matching or handcrafted rules.

Importantly, semantic analysis alone is insufficient for definitive LOTL detection, as legitimate administrative tasks may exhibit similar command structures. However, when semantic features are combined with contextual behavioural signals such as execution timing, persistence, and process chaining the framework gains the ability to distinguish benign administration from malicious abuse of legitimate tools.

Within the proposed framework, semantic analysis functions as a complementary analytical branch that enriches detection decisions with command-level intent. This role directly addresses limitations identified in prior fileless detection approaches, which often rely solely on behavioural context or statistical patterns without capturing the semantic meaning of executed commands.

3.7.5 Complementarity with Contextual Modeling

Semantic command analysis and contextual behavior modeling are designed as **complementary analytical components** within the proposed framework, each addressing distinct yet interrelated aspects of Living-off-the-Land (LOTL) attack behavior. This separation of concerns enables the detection system to capture both execution context and command intent without overloading a single analytical model.

Contextual modeling focuses on execution-level characteristics, capturing *when* processes appear, *how long* they persist, and *how* they interact with other processes through parent-child relationships. These features are effective for identifying suspicious behavioral patterns such as delayed execution, long-running scripting sessions, and extensive process chaining. However,

contextual features alone cannot reliably distinguish between benign administrative workflows and malicious LOTL activity, as both may exhibit similar execution structures.

Semantic analysis complements this limitation by capturing *what actions* are being performed through command content. By encoding command intent into a semantic embedding space, the framework gains visibility into operations such as remote content retrieval, encoded script execution, and abuse of system utilities behaviours that may remain indistinguishable at the contextual level. Nevertheless, semantic features in isolation may also be ambiguous, as legitimate system administration often employs similar command constructs.

The proposed framework resolves these limitations through complementary integration. Contextual modelling constrains semantic interpretation by providing execution context, while semantic analysis disambiguates contextual patterns by revealing command intent. This mutual reinforcement enables more robust discrimination between benign and malicious LOTL activity than either approach could achieve independently.

By explicitly separating and later fusing contextual and semantic representations, the framework avoids the pitfalls of monolithic models that attempt to learn heterogeneous behaviour from a single feature space. Instead, each analytical branch specializes in capturing a specific dimension of LOTL behaviour, and their combination through hybrid fusion enables accurate, interpretable, and real-time detection.

3.8 Hybrid Fusion and Inference

Contextual behavior modeling and semantic command analysis each provide valuable but partial perspectives on system activity. Contextual modeling captures *how* an action is performed within the system, while semantic analysis captures *what* the action represents in terms of intent. However, neither perspective alone is sufficient to reliably detect fileless and Living-off-the-Land (LOTL) attacks. To address this limitation, the proposed framework employs a **hybrid fusion and inference mechanism** that integrates both representations into a unified detection decision.

The objective of this stage is to combine contextual and semantic information in a manner that preserves their complementary characteristics while enabling efficient real-time inference.

3.8.1 Fusion of Contextual and Semantic Representations

The fusion stage integrates execution-level contextual information with command-level semantic understanding to form a unified representation for classification. This integration is designed to preserve the complementary strengths of both analytical branches while enabling joint learning of their interactions.

For each observed process event, two independent feature representations are available:

- a contextual embedding vector

$$\mathbf{e}_{context} = [e_1^{(c)}, e_2^{(c)}, \dots, e_{d_c}^{(c)}] \in \mathbb{R}^{d_c},$$

derived from process behaviour and interaction structure, and

- a semantic embedding vector

$$\mathbf{e}_{semantic} = [e_1^{(s)}, e_2^{(s)}, \dots, e_{d_s}^{(s)}] \in \mathbb{R}^{d_s},$$

derived from normalized command content.

These vectors are concatenated to form a fused representation:

$$\mathbf{e}_{fusion} = \mathbf{e}_{context} \parallel \mathbf{e}_{semantic} \in \mathbb{R}^{d_c+d_s},$$

where $\parallel$ denotes vector concatenation. This operation retains all contextual and semantic information while enabling the classifier to learn cross-feature dependencies between execution behavior and command intent.

The fusion design deliberately avoids early feature aggregation or weighting schemes at this stage. Instead, fusion is performed through straightforward concatenation, allowing the downstream classifier to learn optimal feature interactions directly from labelled data. This approach reduces design bias and improves transparency, as each input feature retains its original semantic meaning.

By preserving explicit contextual and semantic dimensions in the fused representation, the framework supports fine-grained attribution of detection decisions. This property is essential for explainability, as it enables downstream analysis to identify whether a classification

decision is driven primarily by behavioural context, semantic command intent, or their combination.

To clarify how contextual and semantic features are integrated at the fusion stage, a concrete example of feature concatenation is provided. The following table illustrates a representative subset of contextual and semantic features for a single observed event, demonstrating how heterogeneous feature types are combined into a unified representation prior to inference. While only a limited number of semantic dimensions are shown for readability, the full semantic embedding is included during model execution.

Table 3.3 Example of Feature Fusion

Feature Type	Feature Name	Value	Description
Contextual	first_seen	0.21	Normalized time of first process observation
Contextual	duration	0.67	Normalized process lifetime
Contextual	in_degree	0.25	Normalized number of parent relations
Contextual	out_degree	0.58	Normalized number of child processes
Semantic	sem_12	0.12	Latent semantic embedding dimension
Semantic	sem_47	-0.08	Latent semantic embedding dimension
Semantic	sem_282	0.34	High-impact semantic feature (SHAP)
Semantic	sem_473	0.21	Latent semantic embedding dimension
Semantic	sem_611	0.27	Latent semantic embedding dimension
...	...	...	Remaining semantic dimensions (up to 768)

Table 3.3 illustrates an example of the feature fusion process performed at the hybrid fusion stage. Contextual behavioural features and semantic embedding dimensions are concatenated to form a unified feature vector. While only a subset of semantic dimensions is shown for

clarity, the full semantic embedding is included during inference. This fused representation serves as the input to the probabilistic classifier described in the subsequent section.

To further illustrate the fusion mechanism described above, the following figure visualizes how contextual behavioral features and semantic command embeddings are combined. The diagram depicts the concatenation of both feature vectors into a unified representation, which serves as the input to the inference model for malicious probability estimation.

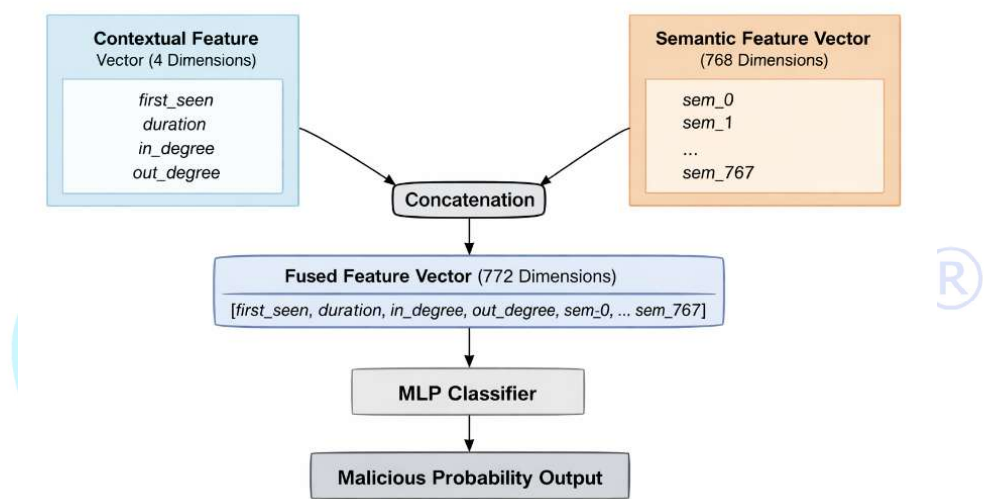


Figure 3.8.1 Context-Semantic Fusion for Real-Time Fileless & LOTL Detection

Figure 3.3 illustrates the fusion mechanism between contextual process features and semantic command representations within the proposed detection framework. Contextual features are derived from the incremental process interaction graph, capturing behavioural attributes such as process lifetime, parent-child relationships, and execution lineage. In parallel, semantic features are extracted from command-line arguments and script content using a transformer-based embedding model. Both feature vectors are concatenated and passed into a fusion-based multilayer perceptron (MLP) classifier to estimate the malicious probability of each event in real time. This fusion design enables complementary reasoning by jointly modelling behavioural context and semantic intent.

3.8.2 Inference and Malicious Probability Estimation

Following feature fusion, the unified representation is processed by a lightweight neural classifier to estimate the probability of malicious behavior for each observed event. The objective of this stage is not to perform complex sequence modeling or graph learning, but to produce a **stable, interpretable, and real-time-compatible probability score** suitable for fileless and Living-off-the-Land (LOTL) detection.

The fused feature vector

$$\mathbf{e}_{fusion} \in \mathbb{R}^{d_c+d_s}$$

is passed through a multilayer perceptron (MLP) consisting of fully connected layers with nonlinear activation functions. The final output layer applies a sigmoid activation function to generate a probabilistic estimate:

$$\hat{y} = \sigma(\mathbf{W}\mathbf{e}_{fusion} + \mathbf{b}),$$

where $\hat{y} \in [0,1]$ represents the estimated probability that the observed event corresponds to malicious LOTL activity.

This probabilistic formulation enables flexible decision-making under varying operational constraints. Rather than assigning binary labels directly, the model outputs a continuous maliciousness score that can be thresholded according to desired trade-offs between detection sensitivity and false positive rates.

A. Decision Threshold Selection

The decision threshold τ is selected empirically based on validation performance. During evaluation, multiple threshold values are assessed to analyze their impact on precision, recall, and false positive rate. The selected threshold balances detection effectiveness with operational feasibility, reflecting the realities of enterprise security environments where excessive false alerts can degrade analyst trust.

Once selected, the threshold is fixed during runtime inference, enabling consistent and predictable detection behavior.

B. Event-Level Inference Flow

Inference is performed on a per-event basis as telemetry events arrive. For each event, contextual and semantic features are computed incrementally and fused to produce a probability score. This event-level inference strategy avoids batch processing and supports near real-time detection, ensuring timely identification of suspicious LOTL activity without requiring full session reconstruction.

3.8.3 Decision Thresholding

The fusion-based classifier produces a continuous probability score $\hat{y} \in [0,1]$ that represents the likelihood of malicious fileless or Living-off-the-Land (LOTL) activity for each observed event. To convert this probabilistic output into an actionable detection decision, a decision threshold is applied.

Rather than using a fixed or heuristic threshold, the threshold value is determined empirically based on validation performance. Multiple candidate thresholds are evaluated to analyze their impact on key detection metrics, including precision, recall, and F1-score. This evaluation enables informed trade-offs between detection sensitivity and false positive rates, which are critical considerations in operational security environments.

Formally, an event is classified as malicious if:

$$\hat{y} \geq \tau,$$

where τ denotes the decision threshold. The optimal threshold τ^* is selected by maximizing the F1-score on the validation dataset:

$$\tau^* = \arg \max_{\tau} F1(\tau).$$

This criterion is chosen to balance precision and recall under the class imbalance conditions commonly observed in fileless and LOTL datasets. Maximizing the F1-score ensures that the model maintains strong detection capability while limiting excessive false alerts that could reduce analyst trust.

Once the optimal threshold is identified, it is fixed and applied consistently during runtime inference. This approach prevents threshold drift during live operation and ensures predictable detection behaviour. The use of probabilistic outputs further allows the framework to support

different operational modes such as high-sensitivity monitoring or precision-oriented alerting by adjusting the threshold without retraining the model.

The effects of threshold selection on detection performance are further examined in Chapter 4, where receiver operating characteristic (ROC) and precision–recall analyses are presented.

3.8.4 Real-Time Inference Considerations

The proposed detection framework is designed to operate under real-time constraints commonly encountered in enterprise security monitoring environments. Rather than performing batch or offline analysis, inference is conducted incrementally on a per-event basis as telemetry data is received. This design enables timely detection of fileless and Living-off-the-Land (LOTL) activity while preserving execution context.

Real-time inference is achieved through a lightweight processing pipeline. Contextual features are updated incrementally from the process interaction graph, allowing changes in execution behavior such as extended process lifetime or increased child process creation to be reflected immediately. In parallel, semantic command embeddings are computed from normalized command strings using a fixed, pretrained encoder, avoiding the overhead of runtime model fine-tuning.

The fusion and inference stages are intentionally kept computationally simple. Feature concatenation and MLP-based probability estimation incur minimal latency, enabling near real-time classification without requiring full session reconstruction or long observation windows. This design ensures that detection decisions can be generated promptly, even for long-running LOTL activity that evolves gradually over time.

To maintain stable performance during live operation, the decision threshold selected during validation is fixed during runtime inference. This prevents adaptive threshold drift and ensures consistent detection behavior. At the same time, probabilistic outputs allow analysts to interpret confidence levels and prioritize alerts based on risk.

Overall, the real-time inference design balances detection effectiveness, computational efficiency, and interpretability. By combining incremental contextual updates, fixed semantic embedding extraction, and lightweight probabilistic inference, the framework supports practical

deployment in operational environments without sacrificing sensitivity to stealthy LOTL attacks.

3.8.5 Interpretation of Hybrid Decisions

The hybrid fusion design enables detection decisions to be interpreted through the combined contributions of contextual behavioural features and semantic command representations. Rather than producing opaque predictions, the framework is structured to allow post-hoc analysis of how different feature groups influence malicious probability estimation.

At the contextual level, interpretation focuses on execution behaviour such as process persistence, execution timing, and interaction structure. Features including process duration, time of first observation, and parent–child relationships provide intuitive explanations for why certain activities are considered suspicious, particularly in Living-off-the-Land (LOTL) scenarios where malicious behaviour unfolds gradually over time.

At the semantic level, interpretation focuses on command intent encoded within semantic embedding dimensions. Although individual semantic features do not correspond to explicit human-readable rules, their relative contributions indicate the presence of command structures and execution patterns commonly associated with LOTL techniques, such as encoded script execution or remote content retrieval.

The hybrid fusion approach enables analysts to assess whether a detection decision is primarily driven by execution context, semantic intent, or their interaction. For example, a long-running process with extensive child process creation may be flagged due to contextual signals, while short-lived but semantically suspicious commands may be detected based on semantic contribution. In many cases, both factors jointly contribute to elevated malicious probability.

This interpretability is particularly valuable in enterprise environments, where legitimate administrative activity often overlaps with malicious LOTL behaviour. By exposing the relative influence of contextual and semantic features, the framework supports informed analyst judgment, reduces false alert ambiguity, and increases trust in automated detection outcomes.

The interpretation of hybrid decisions is further supported by explainable artificial intelligence techniques described in the following section, which provide quantitative feature attribution for individual detection events.

3.9 Explainable Artificial Intelligence (XAI) Component

While the hybrid fusion and inference mechanism produces a probabilistic detection outcome, security analysts and system operators require more than a binary decision or probability score. In practical security operations, it is essential to understand **why** a particular event is classified as malicious in order to support trust, validation, and response actions. To address this requirement, the proposed framework integrates an **Explainable Artificial Intelligence (XAI) component** as an integral part of the detection pipeline.

The objective of the XAI component is not to replace the detection model, but to provide **interpretable explanations** that highlight the most influential contextual and semantic factors contributing to each detection decision.

3.9.1 Motivation for Explainability in Fileless Attack Detection

Explainability is a critical requirement in fileless and Living-off-the-Land (LOTL) attack detection due to the inherent ambiguity between malicious activity and legitimate system administration. Unlike traditional malware, fileless attacks do not rely on static artifacts such as executable files that can be easily inspected or validated. Instead, they exploit legitimate system utilities and scripting environments, causing malicious behaviour to closely resemble benign operational activity.

In enterprise environments, detection systems that generate alerts without transparent reasoning can significantly hinder incident response. Security analysts must determine whether an alert represents genuine malicious activity or normal administrative behaviour. Without explainability, even highly accurate models may be underutilized or disregarded due to lack of trust, especially when false positives involve commonly used tools such as PowerShell or command-line utilities.

Machine learning based detection further amplifies the need for explainability. While hybrid models that combine contextual and semantic features can achieve strong detection performance, their decision processes are not inherently interpretable. In the absence of explainable mechanisms, it becomes difficult to justify why specific events are flagged, which features are responsible, and whether detections align with known LOTL attack patterns.

Explainability is also essential for validating model behaviour and ensuring alignment with domain knowledge. In fileless detection, analysts often reason in terms of behavioural indicators such as process persistence, execution chaining, and suspicious command intent. An explainable detection framework allows these intuitions to be verified quantitatively, ensuring that the model bases its decisions on meaningful behavioural and semantic signals rather than spurious correlations.

For these reasons, explainability is incorporated as a core component of the proposed framework rather than a post-hoc enhancement. By providing feature-level attribution for both contextual and semantic inputs, the framework enables transparent decision-making, supports analyst trust, and facilitates effective investigation of fileless and LOTL attack activity.

3.9.2 Explanation Scope and Granularity

Explainability in the proposed framework is designed to operate at multiple scopes and levels of granularity, enabling both high-level model understanding and detailed investigation of individual detection events. This multi-granular approach is essential in fileless and Living-off-the-Land (LOTL) attack detection, where analysts require both global insight into model behaviour and localized explanations for specific alerts.

At the **global level**, explainability focuses on understanding overall model behaviour across the dataset. Global explanations identify which contextual and semantic features consistently contribute most to malicious classification. This perspective is useful for validating that the model aligns with domain knowledge for example, emphasizing execution duration, delayed process initiation, or semantic patterns associated with encoded commands. Global explanations also support model auditing and feature relevance assessment.

At the **local (event-level) scope**, explainability provides insight into why a specific event was classified as malicious or benign. Local explanations attribute the prediction of an individual instance to its contributing features, allowing analysts to understand how contextual behavior and command semantics jointly influenced the decision. This is particularly important in operational settings, where analysts must quickly assess the legitimacy of flagged activity.

In terms of granularity, explanations are provided at the **feature level** rather than at the raw input level. For contextual modelling, this includes interpretable behavioural features such as

process lifetime, execution timing, and interaction structure. For semantic analysis, explanations operate on embedding dimensions that capture latent command intent. Although individual semantic dimensions are not directly human-interpretable, their relative contributions indicate the presence of suspicious command structures or execution patterns.

By supporting both global and local explanations at appropriate levels of granularity, the framework balances interpretability with practical usability. Analysts can gain confidence in the model's overall behaviour while also obtaining actionable insights for individual detection events. This design ensures that explainability enhances, rather than overwhelms, the detection and investigation process.

3.9.3 Example of Explainable Decision Analysis

To demonstrate how explainability supports the interpretation of detection decisions, this subsection presents an example of explainable analysis for an individual event classified by the proposed framework. The objective of this example is to illustrate how contextual and semantic features jointly contribute to the estimated malicious probability.

Consider an observed event involving a scripting process that executes a command associated with remote content retrieval. After contextual and semantic feature extraction, the fused feature vector is processed by the classification model, producing a malicious probability score exceeding the selected decision threshold. Rather than relying solely on this numerical output, explainability techniques are applied to analyze the factors contributing to the decision.

At the contextual level, features such as first seen and duration exhibit strong positive contributions to the prediction. A delayed initial appearance of the process within the execution timeline (`first_seen`) suggests post-compromise activity, while an extended execution lifetime (`duration`) indicates persistence and orchestration behaviour commonly associated with Living-off-the-Land (LOTL) attacks. These contextual signals align with known behavioural characteristics of fileless attack execution.

At the semantic level, selected embedding dimensions such as `sem_282` demonstrate notable influence on the prediction. Although individual semantic dimensions do not correspond to explicit human-readable rules, their contribution reflects the presence of command structures and execution intent patterns learned by the semantic encoder. In this example, semantic

features reinforce the suspicion raised by contextual behaviour by indicating command-level characteristics consistent with encoded or staged execution.

Figure 4.1 illustrates a local explainability visualization for the analyzed event, highlighting the relative contributions of contextual and semantic features to the final prediction. Positive contributions increase the malicious probability, while negative contributions indicate mitigating factors associated with benign behaviour. The combined explanation reveals that the detection decision is driven by the interaction between behavioural persistence and semantic command intent, rather than by a single dominant feature.

This example demonstrates how explainable decision analysis enables analysts to understand not only *that* an event was flagged, but *why* it was flagged. By providing feature-level attribution across both contextual and semantic dimensions, the framework supports transparent, defensible, and actionable detection of fileless and LOTL attack activity.

3.9.4 Contextual Contribution Explanation

Contextual contribution explanation focuses on interpreting how execution-level behavioural features influence detection decisions within the hybrid framework. Unlike semantic embeddings, contextual features are derived from explicitly defined process behaviour attributes, making their contributions more directly interpretable to security analysts.

In the proposed framework, contextual explanations highlight how temporal and structural characteristics of process execution contribute to malicious probability estimation. Features such as `first_seen`, `duration`, `in_degree`, and `out_degree` are designed to capture behaviors commonly associated with fileless and Living-off-the-Land (LOTL) attacks, including delayed execution, persistence, and process chaining.

For example, a high contribution from the `first_seen` feature indicates that a process appears later in the execution timeline rather than during normal system initialization. This behaviour often corresponds to post-compromise activity, where attackers execute commands after initial access has been established. Similarly, a strong positive contribution from the `duration` feature reflects prolonged process execution, which may indicate orchestration of multiple malicious actions rather than short-lived benign tasks.

Structural features such as `out_degree` provide insight into process interaction patterns. An elevated contribution from this feature suggests that a process is responsible for spawning multiple child processes, a common characteristic of LOTL attack workflows that rely on chaining legitimate utilities. Conversely, low or negative contributions from contextual features may indicate execution patterns consistent with benign administrative behaviour.

By isolating contextual contributions, the framework allows analysts to validate detection decisions against known behavioural indicators of fileless attacks. This alignment between model explanations and domain knowledge enhances trust in automated detection and enables more efficient investigation. Contextual contribution explanation also supports model debugging by revealing whether the classifier relies on meaningful behavioral signals rather than incidental correlations.

Overall, contextual explainability ensures that detection decisions remain grounded in observable execution behavior, providing transparent and actionable insights that complement semantic contribution analysis.

3.9.5 Semantic Contribution Explanation

Semantic contribution explanation focuses on interpreting how command-level semantic representations influence detection decisions within the hybrid framework. Unlike contextual features, semantic features are derived from high-dimensional embedding vectors generated by a pretrained transformer-based encoder. As a result, semantic contributions are interpreted in terms of relative influence rather than explicit human-readable rules.

In the proposed framework, semantic contribution explanation identifies which embedding dimensions exert the greatest impact on malicious probability estimation. These dimensions capture latent command intent, execution patterns, and structural similarities between commands commonly observed in fileless and Living-off-the-Land (LOTL) attacks. Although individual semantic features (e.g., `sem_282`) do not correspond to predefined attack signatures, their contributions reflect learned associations between command structures and malicious behaviour.

For instance, a strong positive contribution from specific semantic dimensions may indicate the presence of encoded script execution, remote content retrieval, or indirect command invocation

patterns. Conversely, lower or negative semantic contributions may suggest command usage consistent with benign administrative operations. This relative contribution analysis allows analysts to assess whether a detection decision is influenced by suspicious command intent rather than solely by execution context.

Semantic contribution explanation is particularly valuable when contextual behavior alone is insufficient to distinguish malicious activity from legitimate administration. In such cases, semantic features provide additional evidence by highlighting command-level characteristics that align with known LOTL tactics. When combined with contextual explanations, semantic contributions help clarify whether a detection decision is driven by the content of executed commands, their execution behaviour, or both.

By supporting semantic contribution analysis at the feature level, the framework enables explainability without requiring explicit interpretation of embedding dimensions. This approach preserves the expressive power of semantic embeddings while maintaining transparency and analyst trust in automated detection decisions.

3.9.6 Hybrid Explanation Synthesis

To provide a coherent explanation, the XAI component synthesizes contextual and semantic contributions into a unified explanation summary. This summary may take the form of:

- A ranked list of influential contextual behaviors,
- Highlighted semantic features or command segments,
- A combined explanation describing how context and semantics jointly contributed to the detection outcome.

This synthesis ensures that explanations remain interpretable and actionable for analysts, while accurately reflecting the hybrid nature of the detection model.

3.9.7 Role of XAI in Model Validation and Evaluation

Beyond operational interpretability, the XAI component also supports **model validation**. By examining explanation outputs during experimental evaluation, researchers can verify that the model's decisions align with expected attack characteristics rather than spurious correlations.

This capability is particularly important in the context of academic research, where explainability strengthens the credibility and reliability of experimental results.

While explainability provides insight into individual detection decisions, quantitative evaluation is required to assess the overall effectiveness and performance of the proposed framework. The next section introduces the detection time measurement and validation metrics used to evaluate real-time detection capability and accuracy.

3.10 Detection Time Measurement and Validation Metrics

Detection latency is measured to assess the suitability of a detection framework for real-time deployment, as emphasized in prior real-time malware detection and security monitoring studies.

A key objective of this research is not only to detect fileless and Living-off-the-Land (LOTL) attacks accurately, but also to detect them **as early as possible**. In real-world security scenarios, delayed detection may significantly reduce the effectiveness of mitigation and response. Therefore, detection time is treated as a primary evaluation criterion in addition to traditional classification performance metrics.

This section defines how detection time is measured and describes the validation metrics used to evaluate the proposed framework.

3.10.1 Definition of Detection Time

Detection time is defined as the elapsed time between the occurrence of a malicious activity and the moment the detection framework produces a malicious classification decision.

Formally, detection time T_{detect} is computed as:

$$T_{\text{detect}} = t_{\text{decision}} - t_{\text{event}}$$

where:

t_{event} denotes the timestamp of the initial malicious event as observed in telemetry,

t_{decision} denotes the timestamp at which the framework outputs a malicious classification.

This definition reflects the **end-to-end detection latency**, capturing the combined effects of telemetry ingestion, preprocessing, contextual and semantic analysis, inference, and decision output.

3.10.2 Detection Latency Components

To better understand the factors contributing to detection time, the total detection latency is decomposed into the following components:

1. **Ingestion Latency**

The time required for telemetry events to be collected at the endpoint and delivered to the processing pipeline.

2. **Processing Latency**

The time required for preprocessing, normalization, routing, contextual modeling, semantic analysis, and hybrid inference.

3. **Decision Latency**

The time required to produce and record the final detection decision.

By analyzing these components separately, the framework allows identification of potential bottlenecks and supports optimization of real-time performance.

3.10.3 Real-Time Detection Constraint

The proposed framework is designed to operate under real-time constraints, where detection decisions must be produced within a bounded time window after event occurrence. In this research, detection latency is evaluated at the millisecond scale to reflect realistic operational requirements.

Detection is considered effective if the framework is able to identify malicious behavior **during execution or shortly after initial manifestation**, rather than after full attack completion.

3.10.4 Validation Metrics

In addition to detection time, the framework is evaluated using standard classification performance metrics derived from the confusion matrix. These metrics include:

- **True Positive (TP)**: malicious events correctly classified as malicious
- **False Positive (FP)**: benign events incorrectly classified as malicious
- **True Negative (TN)**: benign events correctly classified as benign
- **False Negative (FN)**: malicious events incorrectly classified as benign

Based on these values, the following metrics are computed:

- **Detection Accuracy**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **False Positive Rate (FPR)**

$$\text{FPR} = \frac{FP}{FP + TN}$$

These metrics collectively provide insight into the trade-offs between detection sensitivity and false alarm rates.

3.10.5 Relationship Between Detection Time and Accuracy

Detection time and classification accuracy are evaluated jointly to assess the practical usefulness of the proposed framework. A detection system that is highly accurate but slow may be ineffective in real-world scenarios, particularly for fast-executing fileless attacks.

Therefore, experimental evaluation focuses on:

- The distribution of detection latency across benign and malicious events,

- The relationship between detection latency and classification confidence,
- The impact of threshold selection on detection time and false positive rate.

While detection time and validation metrics quantify the effectiveness of the proposed framework, controlled experimentation requires well-defined datasets. The next section describes the synthetic dataset generation process used to evaluate the framework under reproducible and controlled conditions.

3.11 Synthetic Dataset Generation

While detection time and validation metrics quantify the effectiveness of the proposed framework, controlled experimentation requires well-defined datasets. The next section describes the synthetic dataset generation process used to evaluate the framework under reproducible and controlled conditions.

3.11.1 Motivation for Synthetic Data

Fileless and LOTL attacks often leave minimal forensic traces and may not be consistently captured in publicly available datasets. Moreover, real-world telemetry data frequently lacks reliable ground truth labels, making it unsuitable for rigorous performance evaluation.

Synthetic data generation addresses these challenges by:

- Providing accurate and unambiguous ground truth labels,
- Enabling controlled variation of attack scenarios,
- Allowing repeatable experiments across multiple runs.

This strategy aligns with the experimental objectives of this research and supports reliable performance comparison.

3.11.2 Dataset Composition

The synthetic dataset consists of two primary categories of events:

1. Benign Events

These events represent normal system activity, including routine process execution, administrative tasks, and standard command usage. Benign events are generated to reflect realistic operating system behavior across different telemetry sources.

2. Malicious Events (Fileless and LOTL Scenarios)

Malicious events are generated by simulating fileless and LOTL attack techniques, such as encoded command execution, abnormal process relationships, and misuse of system management interfaces.

Each event is labeled explicitly as benign or malicious, enabling supervised evaluation of detection performance.

3.11.3 Telemetry Source Coverage

Synthetic events are generated across the same telemetry sources used in live detection, namely Sysmon, WMI, and ETW. This ensures consistency between training, validation, and live evaluation environments.

By preserving source-specific characteristics, the dataset accurately reflects the heterogeneity of real-world telemetry while maintaining a unified schema for analysis.

3.11.4 Temporal and Behavioral Control

A key advantage of synthetic data generation is the ability to control temporal and behavioral aspects of events. In this research, synthetic events are generated with:

- Controlled timing intervals,
- Defined execution sequences,
- Explicit parent–child relationships.

This control allows precise measurement of detection time and supports evaluation of early-stage detection capability.

3.11.5 Dataset Integration into the Research Framework

Within the proposed research framework, the synthetic dataset serves multiple roles:

- Training and calibration of the detection model,
- Validation of detection accuracy and false positive rate,
- Evaluation of real-time detection latency.

By integrating synthetic data into the same processing pipeline as live telemetry, the research ensures that experimental results reflect realistic operational conditions.

Having described the data acquisition, processing, modeling, and evaluation components, the final section of this chapter summarizes the overall research methodology and establishes a transition to the experimental results presented in Chapter 4.

3.12 Summary of Methodology

This chapter has presented the research methodology used to design, implement, and evaluate a semantic-aware real-time detection framework for fileless and Living-off-the-Land (LOTL) attacks. The methodology was structured to address the unique characteristics of fileless threats, which rely on legitimate system components and evade traditional signature-based detection mechanisms.

To facilitate efficient review and high-level analysis of the proposed methodology, a concise summary is provided in tabular form. Rather than requiring reviewers to examine detailed procedural descriptions across multiple subsections, Table 3.X presents a structured overview of the complete research workflow, including key components, applied techniques, and corresponding outputs.

Table 3.4 Summary of the Proposed Research Methodology

Stage	Component	Method / Technique	Key Output	Purpose
Data Acquisition	Telemetry Collection	Sysmon, WMI, ETW	Raw event streams	Capture low-level execution behavior and system activity
Ingestion	Streaming Pipeline	Winlogbeat, Kafka, Loki	Ordered event flow	Enable real-time processing with preserved temporal order
Preprocessing	Normalization & Correlation	JSON normalization, PID/PPID mapping, timestamp alignment	Unified event representation	Prepare heterogeneous telemetry for downstream analysis

Stage	Component	Method / Technique	Key Output	Purpose
Contextual Modeling	Process Interaction Graph	Incremental graph construction with handcrafted features	Contextual feature vector (first_seen, duration, in_degree, out_degree)	Capture behavioral patterns related to persistence and process chaining
Semantic Analysis	Command Representation	Transformer-based semantic embedding	Semantic feature vector (sem_*)	Encode latent command intent and obfuscation patterns
Feature Fusion	Context–Semantic Fusion	Feature concatenation	Fused feature vector	Combine behavioral context and semantic intent
Classification	Detection Model	Lightweight MLP classifier	Malicious probability score	Estimate likelihood of fileless and LOTL activity
Decision Making	Thresholding	Validation-based threshold selection	Binary detection decision	Balance detection accuracy and false positives
Real-Time Inference	Event-Level Processing	Incremental per-event inference	Near real-time alerts	Enable timely detection without full session reconstruction
Explainability	XAI Component	SHAP-based feature attribution	Contextual & semantic contribution scores	Support analyst trust and decision transparency
Evaluation	Performance Measurement	Precision, Recall, F1, ROC-AUC, latency	Quantitative metrics	Validate detection effectiveness and real-time feasibility

Stage	Component	Method / Technique	Key Output	Purpose
Dataset Design	Synthetic Dataset Generation	ATT&CK-aligned behavioral scenarios	Labeled dataset	Ensure coverage of fileless and LOTL attack techniques

Table 3.4 summarizes the proposed research methodology as an end-to-end detection framework for fileless and Living-off-the-Land (LOTL) attacks. The table highlights how raw multi-source telemetry is transformed into interpretable contextual and semantic representations, fused for probabilistic classification, and evaluated under real-time constraints.

The methodology emphasizes interpretable contextual feature extraction from incremental process graphs rather than graph neural network learning, ensuring both computational efficiency and explainability. Semantic understanding is incorporated through transformer-based embeddings to capture command intent and obfuscation patterns that are difficult to identify through behavioural analysis alone. The fusion-based classification strategy enables balanced detection decisions, while explainability mechanisms provide transparent insight into feature contributions. Overall, this tabular summary provides a compact and reviewer-friendly representation of the complete methodological workflow.