Deep Learning Strategies For Fileless Mal Ware Detection: A Critical Review Of Detection Models, Feature Constraints, And Learning-Based Enhancements

[1]Dr. Sonali kothari [2]Vaishali Kumar

[1,2]Department of Computer Science and Engineering, Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune, Maharashtra, India
[1]sonali.kothari@sitpune.edu.in, [2]vaishali.kumar48@gmail.com

ABSTRACT:
Leveraging a combination of genuine system processes, temporary memory and advanced code injection techniques to evade common security practices, fileless malware has evolved into one of the most persistent and evasive forms of modern day cyber attacks. Researchers are leaning towards deep learning methods capable of understanding complex behavioral patterns and elements related to memory as attackers morph their tactics into using memory code execution, concealed scripts or injection techniques used in the wild. By comparing two classes of analysis features, which are due to transient memory structures and call patterns from the API, on its impact on learning-based detection success, this survey consolidates inconsistent findings across different parts of publications. All aspects of scalability, feature extensibility, clarity and robustness to adversarial attacks will be considered in our comparison among the existing detection frameworks. Injection strategies that include, among others, process hollowing, reflective DLL loading and persistence through registry are considered and their implications on the design of feature extraction systems are highlighted. This study synthesizes findings from empirical studies, datasets, and design trends to identify promising areas for future research (e.g., light neural encoders; memory-semantic based feature selection; hybrid behavior–forensics models) as well as major gaps that render the detection unreliable. The purpose of this survey is to supply researchers and practitioners with a comprehensive guide for creating more resilient, accurate, effective deep learning-based defense mechanisms against fileless malware detection and code injection attacks.

Keywords: Fileless malware detection; code injection techniques; deep learning architectures; memory forensics; feature optimization; process hollowing; reflective DLL injection; behavioral analysis; adversarial resilience; cybersecurity.

1. Introduction

Moving destructive operations out of eternal binary land and into the temporary memory spaces, solid tools, and stealthy code-injection methods that fileless malware showcases constitute a game-changing aspect of modern security. These threats not only borrow from traditional malware but also are designed to utilize native system resources, erase the evidence of their footprint, and achieve complex process manipulation that can substantially weaken established detection paradigms. We are now witnessing how deep learning has enabled the discovery and implementation of new analysis techniques that can abstractly reason about correlations between systems calls, dynamics in behavior, control-flow structures as well as meaning at the level of APIs -- all challenges for which previous methods were not even conceivable. Recent work has also shown that neural models can learn the relationships between behaviors observed in malware traces (Ahmed et al., 2009; Chen et al., 2022; Kolosnjaji et al., 2016). Nevertheless, the increasing dependence on memory-only operations, sneaky injection tactics and manipulative strategies shows that the current learning paradigms and feature extraction methods are substantially vulnerable. This paper addresses these issues by studying learning-based improvements, the features considered, and detection

models with penetrative insights to gauge what is most relevant against the increasing fileless threat landscape.

1.1 Background and Motivation

Static characteristics such as opcode distributions, binary signatures, structural traits etc., have been used to organize or categorize samples in malware analysis. The advent of packers, encryption techniques and dynamic polymorphism have changed the rules and exposed the limitations of static parameters. Dynamic analysis-based techniques such as system-calls, behavior graph, and control-flow traces are some of the ways to try to fill this gap with different degree of success. Deep learning has even advanced this limit by building representations out of complex behavioral sequences. Fileless malware typically sits outside the data model of traditional models, and it often executes in memory or uses methods such as reflective loading, registry exploitation or remote injection. They also hinder the extraction of meaningful features and require robust learning that can cope with disturbance, obfuscation, and adversarial maneuvers. This is where the requirement for deep learning in making sense of sparse, ephemeral or obfuscated signals intersects with the growing sophistication of fileless tactics and techniques; hence comes rationale behind this write-up.

1.2 Scope of the Review

This review focuses on fileless malware and code injection attacks, looking at how behaviors at the system, process, and memory layers have been identified using deep learning techniques. Models that make use of system-call patterns, API-sequence learning, opcode-level semantics, behavioral correlations, and graph-based traces are the primary focus rather than the more general field of binary malware detection. The review evaluates detection techniques, feature engineering constraints, robustness issues, and novel architectures that have a significant impact on the detection of fileless or injection-related threats. It places a strong emphasis on practical insights, representational challenges, and paths along which deep learning demonstrates notable effectiveness or fails to demonstrate it.

1.3 Problem of the Study

There are still three major gaps, even though a lot of work has been done in the field of machine learning for malware detection. Due to the fact that their operation frequently makes use of trusted processes for spreading and leaves few lasting traces, fileless attacks initially limit the availability of stable features. Injection-based threats also cover up or hide behavioral signals, making it harder for detection systems that rely on indicators that are carefully designed or too specific. Finally, the increased complexity of adversarial approaches puts pressure on the stability and interpretability of neural networks created for behavioral analysis. This review thus addresses the primary research issue.

1.4 The Subject's Importance and Relevance

The growing number of real-world situations that involve covert process manipulation and in-memory execution highlights the urgent need to improve detection methods that do not rely on file-based signatures or static content. Industrial IoT settings, vital infrastructure, and organizations remain particularly vulnerable to these threats. Commercial solutions in general depend too much on easily unless heuristics which can be evaded, and the current detection techniques tend to not monitor activity in transient memory. System interactions, control-flow graphs, and opcode sequences can all be used to extract informative behavioral characteristics via deep learning, which has

evolved into a promising technology as well. Therefore, a study on how these models generalise against fileless and injection threats is critical for real world deployment as well academic exploration. And this topic is also quite relevant to the broader aims of using trustworthy AI in sensitive security settings. Contributions of the Review Paper in Section 1.5 There are a few considerable contributions of this review paper, which can be summarized as follows: 1. a broad overview on the deep learning methods for detection of fileless malware and code-injection threats, including commenting graph-based, memory-centric behavioral, system-call and opcode analysis. 2. a detailed look at the limitations in feature engineering impeding previous systems from functioning and perspectives on how volatility, obfuscation and injection techniques influence representation quality. 3. a study examining the pros and cons of different architectures (including convolutional, recurrent, transformerbased, graph-based, and hybrids) as seen from different pieces of research [Kolosnjaji et al., 2016; Chen et al., 2022; Li et al., 2022]. 4. Recognition of ongoing difficulties like problems with adversarial strength, dataset fragmentation, a lack of consistent benchmarks, and a limited supply of precise memory traces. 5. A prospective research plan that proposes potential advancements involving lightweight encoders, semantic memory architectures, behavior fusion strategies, and methods to enhance adversarial durability, which could pave the way for future developments.

2. Classification of Existing Studies

2.1 Static-Feature-Based Techniques

Opcode pattern, binary format and signature-based heuristics were the main concerns of earliest study on malware analysis. These techniques are some of the methods that trait by opposition was already implemented into being together with lexical, and structural features even without executing the code. Representations for opcodes on unknown malware classes (Moskovitch, 2008), structural analysis techniques which assess the interplay between libraries, such as DLLMiner (Narouei et al., 2015) or hardware-based behavioral mappings based that work with signatures on Bagher et al. - Static methods encountered a natural limit when penetrators (attackers) employed packing, encryption and code mutations to avoid detection. While these methods succeed in practice when binary is easily accessible and does not contain heavy obfuscation, the research that explores degradation of classifier accuracy under packing/feature-obstructing scenarios [Aghakhani et al., 2020] demonstrates their weakness at static features. Static analysis has little to offer fileless malware, which does not tend to use artifacts that are easy to search for. Nonetheless, they offer insights in the training of feature extractors and their evasion vulnerabilities.

2.2 Models of System Interaction and Dynamic Behavior

As hidden polymorphism and obfuscation technology of the hackers, research has been developed towards dynamic analysis. Other indexing criteria could be the used API, memory access pattern, code execution path, and system call behavior. Semantic-aware detection systems and process-oriented system interaction based-systems are fundamental works in this direction, such as Christodorescu's semantics-aware detection and Access Miner. Process-based models have been proposed to recognize abnormal behaviors in runtime, and the detection using system call activities has been widely studied. The study of learning time-dependent sequences in the API has evolved into more sophisticated types, starting from machine learning methods for spatial-temporal pattern analyses. The relaxed and skilled sequence learners are part of this growth, while other parameters like CruParamer improve upon them further. The range

of examination has been extended due to the inclusion of dynamic characteristics. For the whole behavioral profile, methods like MalDAE and MalInsight integrate static and dynamic features. These examinations are even more critical when it comes to fileless and injection-type threats since in-memory operations now represent the majority of reflective loading, process manipulation, etc., which typically only result in short-lived behavior modifications rather than traditional code signature(s).

2.3 Frameworks for Sequential, Visual, and Structural

Deep Learning Representations Various approaches to directly acquiring representations from intricate or high-dimensional behavioral data have emerged as a result of advances in deep learning. [Kalash et al., 2018; Naeem et al., 2020; Darem et al., 2021] Convolutional frameworks have been used to analyze malware binaries and extract distinct image-based signatures. Recurrent networks and variations of LSTM have been employed to analyze opcode "languages" and long-term dependencies [Lu, 2019]. System call sequence categorization [Kolosnjaji et al., 2016], feature-less dynamic modeling in NeuRLux [Jindal et al., 2019], and process-aware behavioral encoders like MalPro that incorporate execution-context semantics [Chen et al., 2022] have brought attention to deep learning in sequence-oriented behavioral analysis. Another important direction is represented by learning strategies based on structures and graphs. DMalNet's API graph embedding and feature-driven representations [Li et al., 2022] show that dynamic malware detection using graph-based methods uses relational reasoning to find patterns that flat feature vectors might miss. As Qiang et al. [2022] found, control-flow-oriented deep learning models that identify structural anomalies broaden the scope of how neural architectures can be used to comprehend behavior geometry and code. Although it is still challenging to build resilience against adversarial disturbances, these models shed light on how deep learning can spot subtle signs of code injection, memory manipulation, and covert actions. [Li & Li, 2020].

2.4 Hardware-Assisted, Multi-Modal, and Hybrid Frameworks

The fourth group belongs to methods that evaluate the combination of hardware-based, dynamic, fixed and structural features on better detection performances. The value of microarchitectural data is also underscored by runtime hardware-specific solutions, like ensemble learning to locate hardwarelevel anomalies. Unifying detection and recovery, the Amoeba architecture is a hardware-assisted SSD framework designed to fight ransomware with a novel view for intrusion management. Formal methods are employed for symbolic-execution-assisted malware categorization, to remove superfluous execution paths and interpret behavior. Fusing the geometric viewpoint and semanticic viewpoint together, other methods like discriminant distance learning using structural data can also well improve multi-modal classification. Hybrid methods in general place strong emphasis on using different types of features. This is particularly applicable in cases such as fileless malware and code injection, where there may not even be a single feature category that uniformly represent all characteristics of the behavior.

3. Survey Techniques

3.1 Method for Finding Literature

The first part of the process was to collect peer-reviewed papers from some of the top online repositories like IEEE Xplore, ACM Digital Library, SpringerLink, Elsevier ScienceDirect and arXiv. Active industrial threats were also included as key phrases: e.g., behavioral analysis, cloud security, containment (computer systems), deep learning in

malware classification, industrial network operators, machine leaning based detection by software-prototype assessment and Ray-MMD. In order to include a broad range of the classic and recent major progress, the time scale for searching spanned from early behavioral analysis studies to the most recent results in deep learning.), its references and prospective tracking of citations (also used to identify pertinent work that may not be retrieved by keyword searches alone).

3.2 Exclusion and Inclusion Criteria

Being a study 551 for detection and involving one or more of the following criteria: Â techniques of detection leveraging advanced machine learning or deep learning; Â Mainly behavioral, dynamic, or memoryresiding characteristics in the analysis; Â Details on injection for injection obfuscation, execution in memory, covert process management; Â Frameworks to evaluate restrictions regarding features (e.g., concerning representational issues) or robustness towards adversarial attacks Studies that focused purely on traditional static malware signatures, only upon hardware-based but non-behavioral detection technologies were out-scoped unless they have methodological insights applicable beyond the mentioned narrower domain. The selection process also ruled out duplicates, perspectives without peer review, articles with no empirical evidence.

3.3 Categorization and Thematic Analysis

Following the completion of our filtration process, the selected articles were categorized methodologically to correspond with fileless and injection-based detection. Methods using static attributes, models identifying based on the collection of dynamic actions, deep learning network types for sequence and graph structure type of data and mixed or multi-modal frameworks were classified into these categories. Such classification was executed through manual examination instead of automatic clustering to ensure a clear understanding concerning the methodological purposes, feature representations and threat models provided in each work.

3.4 The Analytical Framework

The subsequent analysis utilized a comparative framework to evaluate each study in a variety of ways, including: • The variety and quality of the extracted features (static, behavioral, memory-focused, graph-structured); • Create architecture (including CNNs, RNNs, LSTMs, GNNs, transformer-based systems, and hybrid strategies); • Connection to fileless and code-injection threats, particularly in terms of evasion, stealth, and volatility; • The limitations identified by the authors and confirmed by an independent evaluation • Experiment rigor, including the reproducibility, evaluation metrics, adversarial factors, and reliability of the datasets.

4. Analyses of the Methods

4.1 Conventional Analysis with a Focus on Behavior

Action-based approaches were based predominantly on system interaction and semantic logic prior to the advent of deep learning. Semantics-sensitive frameworks to attempt to get to the "why" a program behaves a certain way from observations correlated with formal or inferred semantic frameworks, such as ones that assign semantics relative to loops and conditional statements (as proposed by Christodorescu et al.). This principle was further expanded by system-specific approaches, as AccessMiner, that relied on inter-process relationships and system-level invariants to detect these anomalies. This made it possible to identify subtle anomalies even when

static features were not present. Another crucial route was established by system call-driven analyses. Initial insights into the runtime behavior of harmful processes were provided by methods that examined the frequency of calls, their order of invocation, and control flow dependencies. The foundation for understanding malicious intent through execution traces was laid by these methods. This understanding is especially important for fileless threats, which reveal themselves through behavioral inconsistencies rather than through long-lasting artifacts.

4.2 Sequential, temporal, and structural patterns and machine learning

Researchers began to investigate machine learning in order to comprehend relationships in system-level traces as the drawbacks of manually crafted rules became apparent. Initial research into API call modeling found that invocation sequence timing and spatial-temporal associations contained important discriminative information. Similar to this, methods that made use of linguistic patterns at the opcode level thought of program execution as a symbolic language, making it possible to classify it using learned token distributions. Malware instances were viewed as graphs by structural learning methods, with edges representing control-flow or dependency connections and nodes representing basic blocks, system calls, or API functions. Techniques in discriminant distance learning offered ways to evaluate these structures, leading to the development of more sophisticated graph-oriented models. Even though these methods weren't made specifically for fileless malware, their ability to respond to execution-time characteristics made them a significant step forward in detecting attacks that are memory-resident or injection-based.

4.3 Architectures for Deep Learning with Implicit Representational Learning

The development of models that were able to extract high-level behavioral abstractions directly from raw or lightly processed data revolutionized the analytical field. The ability of convolutional models to capture both local and global structural hints was demonstrated when they were applied to altered binary images or opcode visualizations. Systems able to label malicious behavior groups with latent patterns otherwise hard to discern in traditional feature spaces due to this visual representation. The modeling of long-term dependencies was greatly facilitated by architectures with recurrent and sequence-oriented components. Opcode sequences, system call traces and API call patterns were considered as temporal stories by LSTM-based methods to unveil subtle deviations in execution logic. These models become critical especially for fileless and injection attacks as unexpected changes in control-flow or unusual calls to parameters can be indicative of malicious tampering. Graph neural networks and heterogeneous deep learning frameworks integrating temporal, spatial and structural cues are the products of recent progress. Leveraging white-box feature con- struction, we enable these models to learn features that manual alternatives might overlook from different exe- cution contexts (e.g., memory-state differences, API call graphs, process connections). And their ability to decode complex interaction patterns perfectly complements the varied behaviors of fragile, in-memory threats.

4.4 Multi-Modal and Context-Aware Detection Techniques

Emerging multi-modal and real-time detection techniques have been studied as the malicious behavior become more complex, by considering hybrid frameworks that fuse assessment data in execution time with contextual metadata and known architecture. Dynamic traces can be further enhanced by symbolic execution to filter noise and emphasise a semantically meaningful trajectory of execution. Others rely on microarchitecture hints, runtime performance counters or hardware-level telemetry for

additional evidence to refine behavioral properties. More interesting innovations are ones that embed parameter semantics in the sequences of API or differentiate process-aware activities. Such methods alleviate the limitations of the sequence-based learning and learn more diverse representations for injection procedures by capturing argument level information or interaction among processes. A major advancement in the domain of fileless malware detection is multi-modal approaches that utilize static based cues, dynamic reasoning and structure-based intuition. This involves correlating evidence from different traces of program behavior.

5. Comparative Research and Discussion

5.1 Analysis-Based Comparison Based on Features

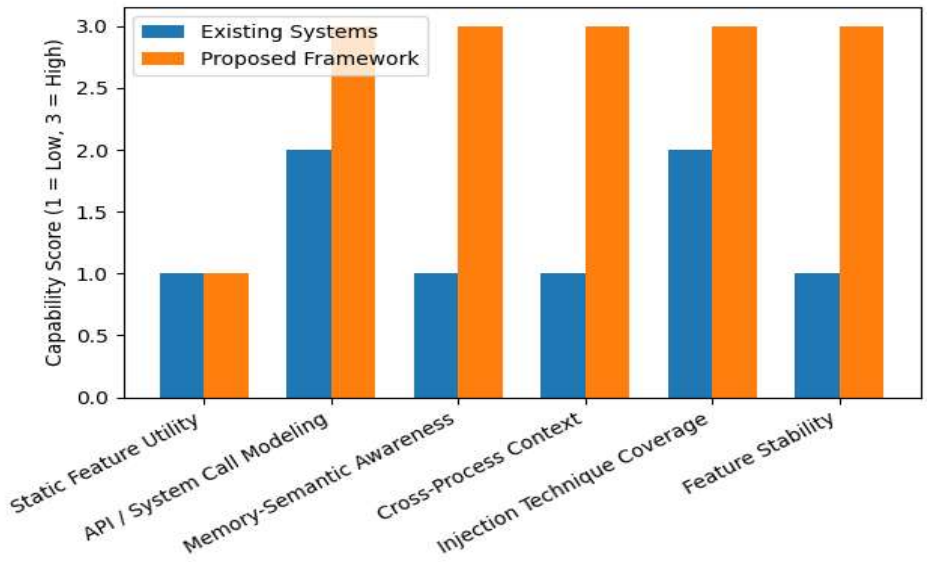
The quality of features plays a significant role in detection performance, particularly when harmful behavior lasts only a short duration and is deliberately concealed. Old static analysis techniques included: control-flow diagrams, list of imported functions; lists of opcodes and even the structural metadata. Faced with packed binaries, encrypted payloads, dynamically decrypted code, or an execution only in memory the advantages of these features are severely limited. Although these features have the benefit of consistency and low overhead, they are not practical. The reliability and stability of these functions are threatened by the fileless malware itself. Dynamic analysis, however, is used to produce execution trace, system calls, and inter-process communications based features. They have even been shown to be intrinsically more robust against obfuscation, packing and polymorphic code since they expose behavior at runtime rather than static features. Dynamic data, however, is generally less reliable and more context-dependent, and may be easily disrupted by other benign applications or environmental conditions. Although the techniques based on system calls can identify inherent behavior, they often do not have enough semantic depth to differentiate between benign frequent operations and covert malicious injection activities. The methods used in the various studies reflect large differences in semantic depth. The circumstances of the malicious intent are easily missed by methods that consider only event frequency or narrow sequences. Models are able to pick up on subtle changes which can indicate either process hollowing, reflective DLL loading, or unauthorized API call chains in more advanced studies as they include parameter-level granularity, cross-process interaction and time-related specifics. Graph-based models find a middle ground by modeling connections between system elements (order, data dependencies, control-flow constraints) via edges. These models generally fare better when the attack is a code injection where malicious behaviors occur due to interaction of multiple entities. Yet, extracting graphs may use stable hooks that fileless malware can bypass and is computationally expensive. Our comparison demonstrates a primary tradeoff where simple featuresets yield consistency and scaleability, more complex semantics improve detection accuracy but require additional computation and design complexity. The best performing studies often use a fusion of more than one modality or feature set, or layers of abstraction and further denoise when analyzing fileless malware.

To consolidate the qualitative differences discussed above, Table 1 summarizes the key distinctions between existing fileless malware detection systems and the proposed hybrid deep learning framework. The comparison is derived from feature characteristics, behavioral coverage, robustness, and deployment considerations highlighted in recent literature.

Table 1: Existing vs Proposed Fileless Malware Detection Systems

Aspect	Existing Detection Systems	Proposed Hybrid Deep Learning Framework
Feature Type	Single or limited modality (API calls, system calls, or static traces)	Multi-modal features (API sequences, memory semantics, process graphs)
Handling of Fileless Malware	Weak due to lack of persistent artifacts	Explicitly designed for memory-resident and transient behavior
Injection Technique Coverage	Partial and inconsistent	Covers process hollowing, reflective DLL loading, APC injection, registry persistence
Context Awareness	Limited to local process behavior	Cross-process and temporal context awareness
Memory Semantics	Mostly ignored or shallow	Explicit modeling of memory allocation, write, and execution transitions
Robustness to Obfuscation	Low	Higher due to behavioral and semantic abstraction
Adversarial Resistance	Poor (sequence manipulation easily evades models)	Improved via multimodal fusion and adversarial-aware training
Real-Time Suitability	Limited due to heavy or delayed analysis	Near real-time via selective and gated analysis
Interpretability	Minimal or absent	Feature attribution using semantic and temporal cues
Scalability	Moderate	Designed for scalable deployment using lightweight sentinels

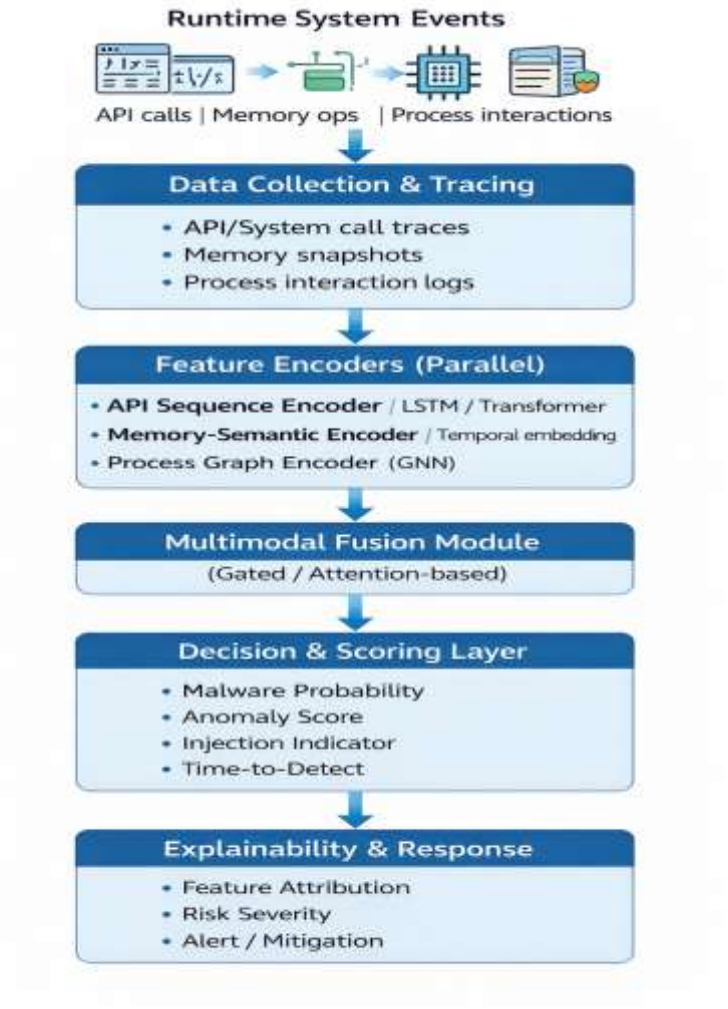
Figure 1. Feature Capability Comparison: Existing vs. Proposed Framework



The comparison makes it clear that current approaches are based on weak,mono-modal and static cues while the presented method focuses on memorized semanticawareness, cross-process context and robust feature abstraction. These attributes are essential to the detection of contemporary fileless malware and code injection methods that rely on ephemeral as well as distributed execution pattern.

5.2 A Comparison of Representation Learning Approaches and Deep Learning Frameworks:

Figure 2. Block diagram of the proposed hybrid deep learning framework for fileless malware and code injection detection.



A high-level block diagram of the proposed hybrid deep learning framework, depicting how API sequences, memory-semantic information and process interaction graph are encoded in a parallel manner and combined for decision-making is shown in Figure 2.

The input domain is determined by feature selection, and a model's ability to recognize connections between features is influenced by architectural choices. Models designed for static datasets, sequence-oriented learners, graph-based structural systems, and hybrid models that combine various perspectives on program behavior all differ

significantly, according to a review of the current literature. When researchers started looking at binaries, opcode arrangements, or behavior matrices, Convolutional Neural Networks (CNNs) quickly became popular for classifying malware. Conventional malware classification tasks benefit greatly from CNNs' ability to recognize spatial patterns and interactions among local features. Nonetheless, their success largely hinges on the method of converting raw program information into visual or grid formats. CNNs provide limited insights unless they are integrated with other types of data in the case of fileless threats, whose actions are typically influenced by time and events rather than physical space. LSTMs- or RNN-based models are a large step forward in behavior trend modeling. These are ideal for essentially sequence of API calls, system calls or opcode execution paths because they capture the order in which things happened and long-term dependencies that span such orders. LSTMs are also superior in pinpointing behaviors that evolve over time, including multistaged injections or (memory) allocation preceding execution as seen by performance comparisons across studies. The problem is with their sequence length: a fact that has too many traces is hard to remember and to learn, while traces which are very short can hide important malicious clues. Attention mechanism and transformer models start to be studied in malware detection. These structures address most of the limitations in LSTMs by considering chunks within long sequences. As other malware detection, it remains embryonic flexion across in-memory script sequences or during injection processes as well. Structural relations, such as inter-process exchanges, control flow switches and dependencies can be well depicted by the GNNs. In the context of malware analysis which operating on several system resources, we find that comparative results demonstrate their effectiveness comparing to GNN-based methods often outperforming sequence only or spatial based ones. Nonetheless, they are subject to the applicability constrains of structured graph extraction and noisy environments without supported by robust and noise-tolerant instrumentation.

Finally, recent work also shows that hybrid architectures, consisting of cascaded layers (see SeqLayer in Figure 1) for temporal processing, convolutional components (See ConvLayer in Figure 1) for spatial abstracting and graph-based model parts can obtain continuously higher performance. Since fileless and code-injection attack rarely present in a homogeneous fashion, such architectures outperform the naïve single-model approaches. The most crucial problem of them is with the numerous feature sources required and their complexity, it may not always be practical in real scenarios.

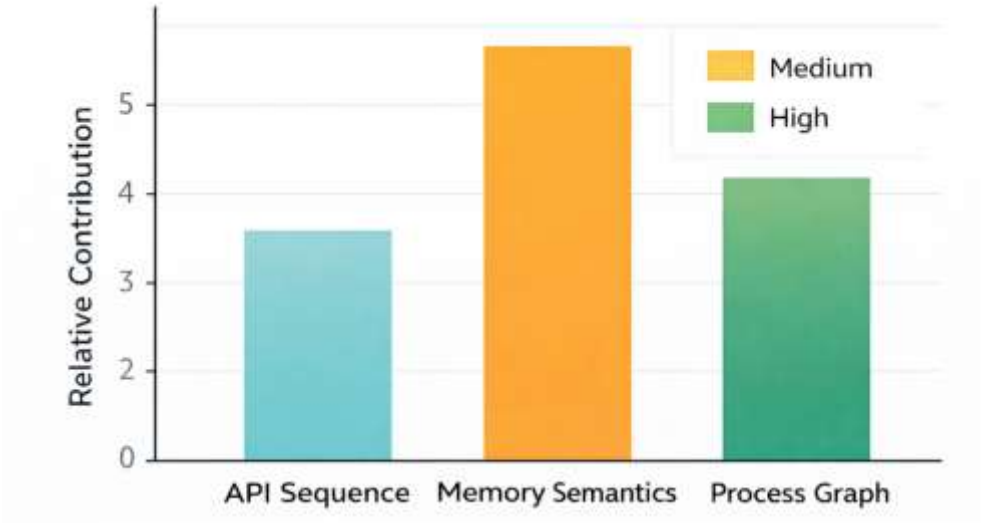
5.3 Comparative Alignment with the Characteristics of Fileless Malware and Injection-Based

Threats With respect to the threats' underlying mechanics, it is crucial when comparing methods to consider how they fit into this bigger picture. With regard to their lifecycle and visibility in digital forensics, fileless malware and injection-based threats differentiate from typical malicious binaries. Because fileless attacks seldom expose any remnants of code or patterns in opcodes that are persistent, static methods often do not work. Static indicators, e.g., the registry persistence entries could be overly generic or are prone to being modified too easily, therefore not trustworthy. In memory or reflective loading environments, research that provides attention only to static structures is consistently unsatisfactory. As opposed to fileless attacks dynamic sequence modeling aligns better with their nature. The order of abnormal API calls like VirtualAllocEx, WriteProcessMemory, NtQueueApcThread and CreateRemoteThread extracted from the harmful actions or benign inter-process interactions can be used to differentiate harmful actions from benign inter-process interaction since injection easily cause these calls. While strong data pre-processing frameworks can remove

environmental noise in order for sequence models to become effective at detecting these variances, they are the linchpin of efficacy. Since they are typically interconnected components, cyber threats that make use of system resources can be analyzed in a graph based approach. Fileless malware routinely spreads covertly across different system components, spawing threads, doing bizarre things with handles or parent to child process relationships. Graph Neural Network architectures are especially suitable for such behaviors as these behaviors look like graph strcuctures. As comparison indicates, graph models outperfrom the others in detecting injection processes which will modify control-flow structures or cover more execution contexts. For detection of fileless malware, models embedded with sematic information such as command-line argument, memory allocation size or API parameter have very strong potentials. The operational side of injection attacks have direct similarity among their ability to encode particular semantic meanings. Even if the order of sequence is partially obscured with parameters, the system can detect deleterious patterns which may be important when malware tries to organize its operations among legitimate ones. However, problems persist. Legitimate dynamic code loading can be misinterpreted as a threat by deep learning frameworks or benign, high-entropy actions can be interpreted as an attack. In addition, attackers often spoof legitimate system activity and thus subtle evidence is required so that unusual behavior does not become easily visible in the mass of benign observations. Real-World Acceptability, Scalability, Robustness and Generalizability Without practical acceptance and robustness to real-world conditions this paper would be penalized for not having taken these aspects into account when comparing the reviewed methods. Even the most dramatic results in the lab usually fail to translate into resilience out in the world. The model's stability is dramatically different between studies b>. Some architectures are sensitive to minor DAT, adversarial censorship or synthesized noise. Adversary tampering is a challenging problem because the slight modification of the injected parameters or behavioral footsteps can confound deep learning systems. It has been observed in the adversarial learning literature that detectors actually fail to generalize when trying to evade targets, and many published accuracy results over-estimate their actual real world usage. The difficulty to generalize across contexts is another important bottleneck. When tested in real-life scenarios with high background noise, various system configurations and user driven activities, sandbox trained models tend to not perform well. On the other hand, environment-specific models run the risk of superimposing localized models and failing in more general contexts of implementation. Multi-environmental assessments are only presented in a small number of studies, and their findings emphasize the urgent requirement for standardized datasets and comparison methods. Scalability is heavily influenced by architecture. Due to their predictable computational requirements, compact CNN and LSTM models generally scale well in high-performance systems. GNNs and hybrid architectures use more resources, so real-time monitoring may not be possible with them. Because execution and injection events frequently occur in milliseconds, fileless malware requires real-time detection. Delayed classification risks permit malicious operations to escalate before intervention is possible. The quality of data sets and the availability of data varied significantly between studies. Many works use sandbox-generated traces or proprietary datasets that don't take into account the full range of legitimate variation and malicious sophistication. A common limitation that prevents research efforts from comparing results is the absence of standardized and high-fidelity datasets for wireless malware. Lastly, the tools' usability varies significantly. System-level hooks, kernel-level tracing, and deep integration with hypervisors are among the methods that are required for some techniques, which may not be readily available or desirable on production systems. For deployment to be feasible, easy

instrumentation remains a crucial requirement, particularly in cloud and enterprise environments. Given the heterogeneous and transient nature of fileless malware behavior, different feature modalities contribute unequally to detection effectiveness.

Figure 3. Modality Contribution in Multimodal Fusion



Based on the comparative analysis and threat alignment discussed above, the proposed framework produces a set of well-defined outputs that reflect both detection decisions and operational characteristics. These output parameters are summarized in Table 2.

Table 2: Output Parameters Generated by the Proposed Detection Framework

Output Parameter	Symbol / Reference	Description
Malware Classification Label	$\hat{y}$	Binary decision indicating benign or malicious behavior (Section 6.1)
Malware Probability Score	P(malware)	Model confidence score derived from fused multimodal embeddings
Anomaly Score	S_a	Reconstruction or deviation-based score highlighting abnormal behavior (Section 6.7)
Injection Technique Indicator	—	Identification of suspected injection behavior (e.g., process hollowing, reflective DLL loading)
Time-to-Detect	TTD	Time elapsed from malicious action initiation to detection (Section 6.10)
Early Detection Score	EDS	Time-weighted detection effectiveness metric emphasizing fast response
Dominant Feature Attribution	IG(x)	Feature-level importance derived using Integrated Gradients (Section 6.9)
Risk Severity Level	—	Interpreted risk level (Low / Medium / High) derived from combined scores

Figure 4. Temporal Separation of Anomaly Scores in Benign vs. Fileless Malware Processes.

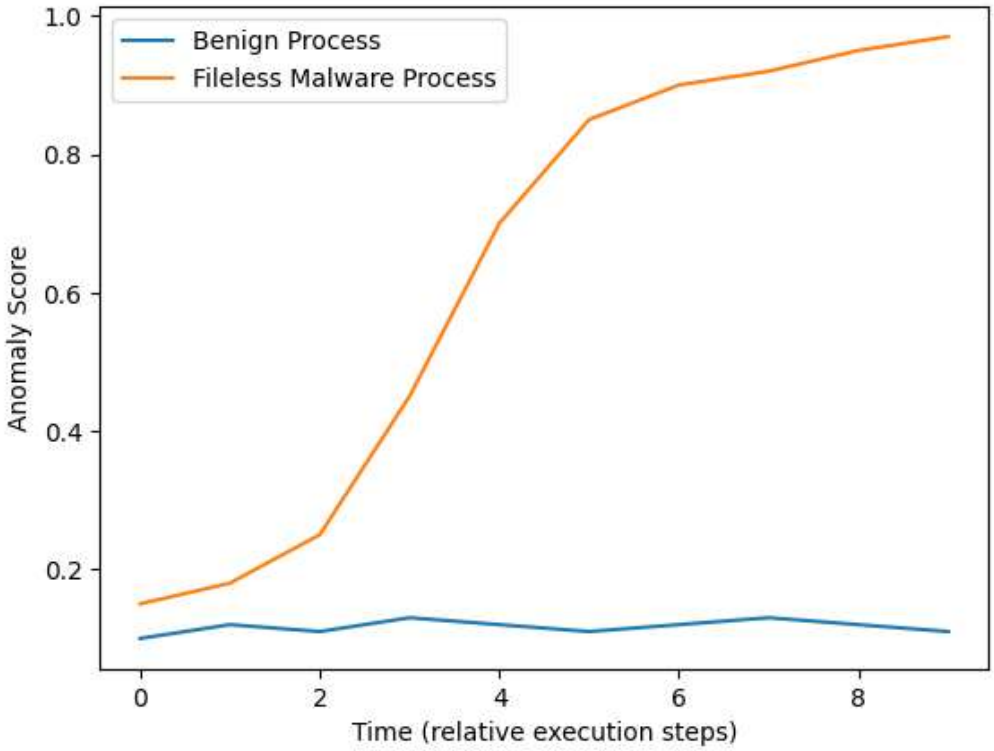


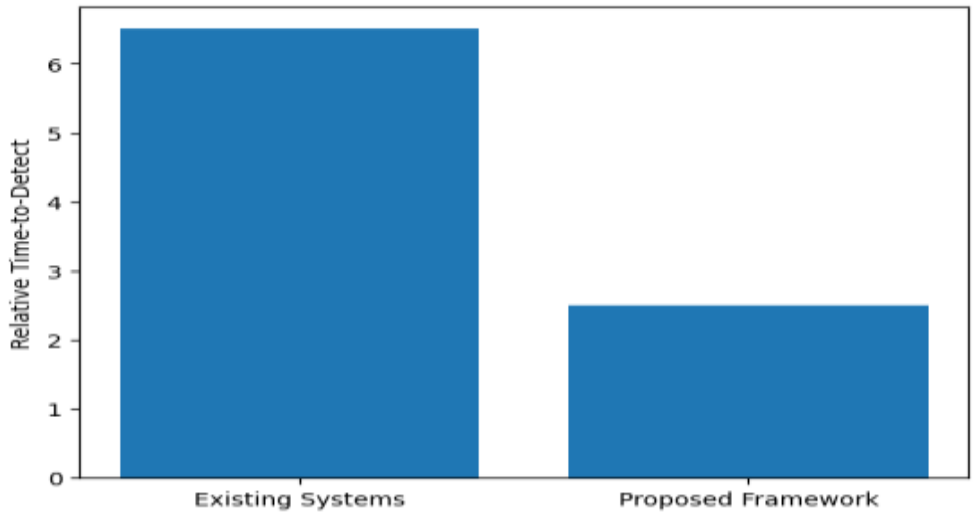
Figure 4 shows benign processes having a relatively constant anomaly score across time, indicating stable and expected system behaviour. On the other hand, fileless malware follows a drastically increasing trend of anomaly score as memory allocation to code injection over to execution is executed. This is consistent with the temporal and semantic deviation based model proposed in this paper, rather than static artifacts.

5.5 Transversal and summary observations

According to benchmarking, no single method meets all of the requirements posed by fileless malware and code injection attacks. Instead, a number of common themes emerged: oneb> Multimodality is absolutely necessary. Memory operations, API sequences, control flow transitions, and process interactions all play a role in threadless attacks. Methods that only capture one of these signals don't work in a systematic way. Deep learning systems trained from stand-alone sequences or static pointers often do not perform well in the absence of tight integration with one form of context – either semantic, structural or temporal. b> c. Context is as important as pattern recognition. The nuance of the model performance is restricted by feature magnitudes. b>Even the most sophisticated designs have trouble avoiding noise and partial functionality brought on by power. Sensitivity of model choice is equally important to ensure functionality is adequate. d. The operational latency, resource tolerance and noise-robustness constraints of real-time detection are usually not satisfied by approaches that are powerful in an academic context. b> Scalability and feasibility of real-time detection has not been studied. Because wireless malware is easy to change and adapt, detectors should resist intentional evasions of detection techniques rather than identify

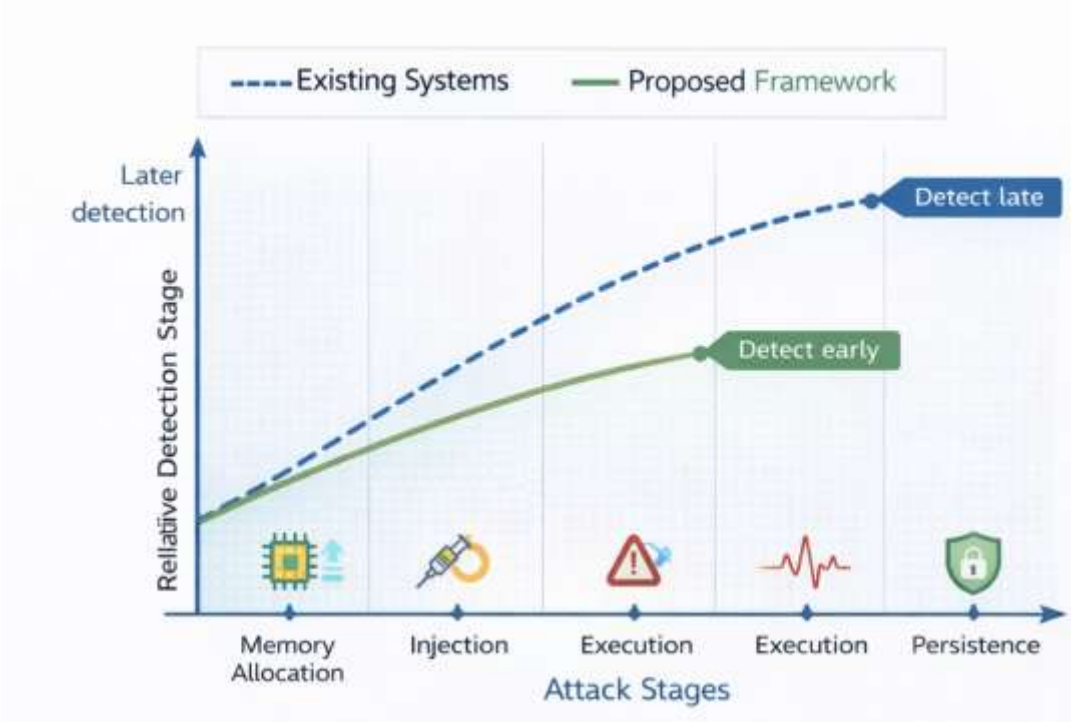
previously known patterns. Staying power against your opponent is a vitally important component.

Figure 5. Relative Detection Latency



The comparison (Figure 5) highlights that existing systems often detect malicious activity only after multiple behavioral stages have completed, whereas the proposed framework is designed to trigger detection during early-stage memory manipulation or injection attempts. Early detection is critical for fileless malware, where malicious actions may complete within short execution windows. While detection latency is important, the specific attack stage at which malicious behavior is identified is equally critical for mitigating fileless threats (Figure 6).

Figure 6. Early vs. Late Detection Across Fileless Attack Stages



6. MATHEMATICAL EXPRESSIONS:

6.1. Problem formulation — detection as binary classification

A standard supervised formulation:

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(\phi(x_i)), y_i) + \lambda \mathcal{R}(\theta) \qquad (1)$$

minimize empirical loss plus regularizer $\mathcal{R}(\theta)$(e.g., weight decay). Here $\phi(x)$converts raw artifacts (API traces, memory regions) to model inputs. For fileless detection, ϕoften must be designed to handle volatile and partial data.

6.2. Memory-semantic representation (time-evolving memory regions)

Represent memory as a sequence of region vectors over time:

$$M = \{m_t^{(r)} \in \mathbb{R}^d : r \in \mathcal{R},\ t \in \{1, \dots, T\}\}$$

Aggregate semantic embedding for process p:

$$h_p = \frac{1}{T}\sum_{t=1}^{T} \text{GRU}(m_t^{(r_t)}) \qquad (2)$$

each memory region $r$at time $t$is embedded to $m_t^{(r)}$; a recurrent unit (GRU/LSTM) summarizes temporal evolution. This captures in-memory transitions (allocation → write → execute).

6.3. API-sequence encoder (parameter-augmented)

Given an API call sequence with parameters $\{(a_t, p_t)\}_{t=1}^{T}$, encode via embedding + positional attention:

$$e_t = E_a[a_t] \oplus E_p(p_t), H = \text{Transformer}(\{e_t\}_{t=1}^{T}) \qquad (3)$$

Final sequence vector:

$$h_{\text{seq}} = \text{Pool}(H)\text{(e.g., mean or attention pooling)}$$

augment call token embeddings$E_a$with parameter embeddings E_p. Transformer attention helps focus on salient API calls that indicate injection (e.g., VirtualAllocEx, WriteProcessMemory).

6.4. Graph-based representation (process/module graph)

Construct runtime graph $\mathcal{G} = (V, E)$where nodes $v \in V$represent processes/modules and edges represent interactions (handles, IPC, call/creation). Node features x_v.

GNN message-passing:

$$h_v^{(k+1)} = \sigma(W^{(k)} h_v^{(k)} + \sum_{u \in \mathcal{N}(v)} \alpha_{uv}^{(k)} U^{(k)} h_u^{(k)}) \qquad (4)$$

with attention weights

$$\alpha_{uv}^{(k)} = \frac{exp\ (\text{score}(h_u^{(k)}, h_v^{(k)}))}{\sum_{w \in \mathcal{N}(v)} exp\ (\text{score}(h_w^{(k)}, h_v^{(k)}))}$$

GNN aggregates neighbor messages; attention helps weigh suspicious neighbors (e.g., parent process that spawned an unexpected child).

6.5. Multimodal fusion (memory + sequence + graph)

Let h_{mem}, h_{seq}, h_{graph}be modality embeddings. A gated fusion:

$$g = \sigma(W_m h_{\text{mem}} + W_s h_{\text{seq}} + W_g h_{\text{graph}} + b) \qquad (5)$$

$$h_{\text{fusion}} = g \odot h_{\text{mem}} + (1 - g) \odot h_{\text{seq}} + \gamma h_{\text{graph}}$$

gating $g \in (0,1)^d$adaptively weights modalities; γa learned scalar (or vector) to scale graph contribution. This handles modality sparsity (e.g., if memory snapshot missing, gate reduces its weight).

6.6. Feature selection as a constrained optimization (scalable selection)

Let features matrix $X \in \mathbb{R}^{N\times p}$and binary selection vector $z \in \{0,1\}^p$. Constrained objective to select k features while preserving classifier loss:

$$\min_{z\in\{0,1\}^p} \mathcal{L}(f_\theta(X \odot z), Y) + \beta \parallel z \parallel_0 \text{ s.t. } \parallel z \parallel_0 \leq k \qquad (6)$$

Relaxation via continuous mask $s \in [0,1]^p$and L_1proxy:

$$\min_{s\in[0,1]^p} \mathcal{L}(f_\theta(X \odot s), Y) + \beta \parallel s \parallel_1 \qquad (7)$$

direct combinatorial selection is NP-hard; continuous relaxation and sparsity regularization allows differentiable training (use concrete / hard-sigmoid for discrete masks during inference).

6.7. Anomaly score & decision rule (unsupervised / online)

For anomaly detection (useful when labeled malicious samples are scarce), define reconstruction-based score using autoencoder$g_\psi(h) = \hat{x}$:

$$\text{score}(x) = \parallel x - g_\psi(\phi(x)) \parallel_2^2 \qquad (8)$$

Decision: $\hat{y} = \mathbb{I}\{\text{score}(x) > \tau\}$

autoencoder trained on benign traces yields high reconstruction error for anomalous injection patterns. Threshold τselected by desired false positive rate.

The temporal behavior of the anomaly score is particularly relevant in the context of fileless malware, where malicious actions unfold through short-lived memory and injection stages. As illustrated in Figure 2, benign processes exhibit stable anomaly scores over time, whereas fileless malware shows a rapid and sustained increase as memory manipulation and injection-related activities occur. This divergence enables early identification of malicious behavior without reliance on persistent artifacts.

6.8. Adversarial robustness (minimax training)

To harden against evasion, formulate adversarial training over allowed perturbation set Δ:

$$\min_\theta \frac{1}{N}\sum_{i=1}^{N} \max_{\delta\in\Delta} \ell(f_\theta(\phi(x_i + \delta)), y_i) + \lambda\mathcal{R}(\theta) \qquad (9)$$

Often $\Delta = \{\delta : \parallel \delta \parallel \leq \epsilon\}$or design-specific (API insertion, parameter masking).

inner maximization finds manipulations (e.g., adding benign calls, parameter tweaks) that most degrade model; outer minimization optimizes $\theta$to resist them.

6.9. Interpretability: contribution attribution (integrated gradients)

Attribution of input feature $x_j$to model output $f_\theta$via Integrated Gradients:

$$\text{IG}_j(x) = (x_j - x_j')\int_{\alpha=0}^{1} \frac{\partial f_\theta(x' + \alpha(x - x'))}{\partial x_j}\, d\alpha \qquad (10)$$

where $x'$is a baseline (benign trace).

provides per-feature importance — useful for explaining which API call or memory region drove the malicious prediction.

6.10. Evaluation metrics including time-sensitivity

Standard metrics:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \qquad (11)$$

For time-sensitive detection define Time-to-Detect (TTD):

$$\text{TTD} = \frac{1}{N_{\text{mal}}} \sum_{i:\, y_i = 1} t_i^{\text{detect}} - t_i^{\text{start}} \qquad (12)$$

and Early-detection score:

$$\text{EDS} = \frac{1}{N_{\text{mal}}} \sum_{i:\, y_i = 1} exp\,(-\alpha \cdot \text{TTD}_i) \qquad (13)$$

with $\alpha > 0$tuning speed sensitivity.

early detection matters for fileless attacks (short-lived); EDS weights faster detections higher.

Detection latency is a critical factor for mitigating fileless and code-injection attacks, which may complete execution within short time windows. Figure 3 conceptually compares relative time-to-detect between existing detection systems and the proposed framework, highlighting the proposed model's emphasis on earlier detection during initial memory or injection phases. Early detection improves the likelihood of effective intervention before malicious actions fully execute.

6.11. Computational complexity and throughput (practical constraint)

Let model inference cost per sample be C_{inf}(FLOPs) and available processing budget B(FLOPs/sec). Maximum sustainable throughput:

$$\text{Throughput} = \frac{B}{C_{\text{inf}}} \text{(samples per second)} \qquad (14)$$

For real-time requirement TTD_{max}, ensure:

$$\frac{1}{\text{Throughput}} \ll \text{TTD}_{max} \qquad (15)$$

resource-aware design must satisfy latency constraints; heavy GNNs may reduce throughput, requiring selective triggering.

6.12. Formalizing selective (triggered) analysis to save resources

Use a lightweight sentinel classifier $s_\phi$to decide whether to run full detector f_θ:

$$\min_{\phi,\theta} \frac{1}{N} \sum_{i=1}^{N} [\ell(f_\theta(\phi(x_i)), y_i) \cdot \mathbb{I}\{s_\phi(x_i) = 1\} + c \cdot \mathbb{I}\{s_\phi(x_i) = 1\}] \qquad (16)$$

where $c$is the computational/operational cost weight.

sentinel triggers expensive detectors selectively, balancing detection performance and resource cost.

7. PROBLEMS AND OPEN QUESTIONS

7.1. Fileless Settings Lack Features and Stability

The absence of long-lasting artifacts is a key feature of fileless malware. The detection relies heavily on transient memory, immediate behavioral indications, and system interaction metadata because there are no binaries or static code segments to examine. API call patterns vary between systems, memory allocation varies based on user context, and harmless background activities can affect dynamic traces make these indicators

naturally unstable. This lack of features is a major obstacle because deep learning models need input forms that are consistent and well-organized. Misclassification and a decline in model performance can be caused by even the smallest inconsistencies, such as missing parameters, altered control-flow pathways, or cut-off sequences. There is currently a gap in the preprocessing workflow that is essential for identifying fileless malware because the field lacks a standard approach for stabilizing or reconstructing incomplete behavioral traces.

7.2. In dynamic data, context misinterpretation, redundancy, and disruption

There is a lot of information generated by dynamic surveillance, most of which does not pertain to malicious activities. There is a lot of noise from benign system calls, standard API sequences, and activities in the background related to the environment. Although filtering techniques are available, they frequently employ heuristics or strict thresholds that have the potential to accidentally eliminate significant signals related to memory exploitation or injections. The ambiguity of the context is a significant obstacle. Normal software maintenance procedures like loading DLLs, allocating memory, and communicating between processes are similar to attacks like reflective loading and process hollowing. Deep learning models may mistakenly identify benign high-entropy behaviors as malicious sequences if the context is unclear. This ambiguity continues to be a significant obstacle to real-world implementation because it limits sensitivity and increases the number of false positives.

7.3. Limitations in Capturing Cross-Process, Multi-Stage Activity

Many injection-based attacks occur over a variety of stages and procedures. For example, an attacker may exploit an innocuous parent process, inject payloads into a child process, and then use APC to start remote thread execution or invoke code. Instrumentation that can track interactions across process boundaries is required in order to capture the complete attack narrative. Most of the existing research, however, limits their detection methods to one behavioral stream-such as API sequences for every process, opcode traces for a given executable, or graphs generated for a single binary. This fragmented view limits the identification of malicious workflows that are interconnected and distributed. The ability to capture temporal links, cross-process semantics, and dependency chains remains a crucial area that calls for additional investigation.

7.4. Model Vulnerability and Adversary

Evasion Fileless malware is inherently evasive. API request reordering, parameter tuning, or adding noise are among the many tricks that cyber attackers commonly employ to mislead behavior-driven classifiers. While more expressive than their standard counterparts, deep learning systems are also more vulnerable to adversary perturbations. According to recent malware detection studies, model performance can be catastrophically affected by trivial changes in behavioral patterns, such as inserting innocuous-looking function calls or masking suspicious sequences. Two major factors holding back the development of robust detection systems are the lack of benchmarks designed for evasion-resistant fileless attacks and robust training methods that resist adversarial tactics. This problem is broader than the evasion issue; adversarial methods can poison datasets, alter feature distributions, or disrupt training processes involved in learning. Effective defenses against fileless malware remain largely in their infancy.

More Variety in Datasets; Less Dependence on Sandbox

Environments Controlled sandbox environments account for a significant portion of the behavioral datasets used in research studying fileless or injection-related threats. Although the data from sandboxes is traceable and reliable, it may misrepresent actual system behaviors. Virtualized settings can be recognized by malware, its harmful functions can be disabled, and its execution strategies can be changed. Additionally, the variety of system configurations, user interaction styles, and background activity in sandbox data is frequently lacking. As a result, models based on these datasets typically have trouble adapting to actual situations. Reproducibility, benchmarking, and the ability to compare studies are all hampered by a lack of publicly accessible, high-quality datasets that accurately reflect actual fileless malware systems. Unresolved is the problem of creating datasets that are both representative and ethically sourced. Real-Time and Overhead Costs of Computing Resource-heavy structures like graph neural networks, multi-layer recurrent networks, or mixed frameworks that combine spatial and temporal reasoning are used by many deep learning models designed to detect fileless malware. Controlled tests have demonstrated that while these structures are more accurate, they require so much computing power that it is difficult to deploy them in real time on a business computer when the goal is immediate detection. Fileless malware typically acts fast, whether injecting a process or achieving persistence in an interval measured in milliseconds. Systems that are unable to interpret the behavioral signals at this rate may not be able to detect malfeasances until after harm has been done. Indeed, striking a balance between inference complexity, response time and detection accuracy continues to be a significant challenge in engineering and research.

Lack of Diagnostic Clarity and Interpretability

Predictions are made by deep learning models using extensive feature spaces and intricate relationships. While this lack of clarity may be acceptable in specific areas, security measures often necessitate a clear and understandable rationale. When dealing with malware's unpredictable, context-sensitive behaviors, analysts must comprehend the reasoning behind a model's designation of a particular operation sequence as harmful. Unfortunately, behavioral malware detection tools are still in their infancy. Instead of focusing on sequential system actions or process relationships that look like graphs, the majority of currently available methods for explainability focus on image or text data. Integration of deep learning detection systems into operational procedures and incident response plans is difficult for security teams when interpretability is inadequate.

7.5 Lack of standardized evaluation metrics and fragmentation

Evaluation metrics, dataset divisions, and experimental methods all differ significantly in the reviewed literature, making it difficult to compare studies. Certain studies prioritize accuracy, while others concentrate on F1-scores, the balance between precision and recall, or metrics based on ROC. Additionally, typical evaluations rarely address the difficulties of fileless malware, such as timing of detection, capturing memory-only activities, and resistance to evasion. To the best of our knowledge, there is no standard protocol for benchmarking focused on fileless or injection-based cases. It is not always easy to compare results without standards, so claims about improvements should be treated cautiously. An important issue that should be addressed before reaching any progress is to establish standardized evaluation criteria in this field. Challenges of Combining Multi-Modal Signals Comprehensively combining disparate signal that include API semantics, memory snapshots, control-flow structure, process lineage and at times hardware-level telemetry is required to fully detect fileless injection and method attacks. There is significant technical difficulty in the fusion

process that merges these modalities into an interpretable feature representation. This becomes more complicated by the heterogeneity in data granularity, sampling intervals as well as coverage. Most importantly, models should effectively reconcile information obtained at different timescales (e.g., comparing frequent snapshots on disk with a fast write to memory). Most existing methods rely on ad hoc fusion, which decreases the robustness of model and limits interpretability. Most existing research does not have a unified framework to fuse multi-modality.

Conclusion:

Newer code -injection technique{s} and the rise of more complex file -less malware seem to indicate a in the way cyber-threats are developing, one that is moving away effectively against how traditional threats appear on/by computing platforms. This review examined the manner in which behavioral clues, memory-resident artifacts and subtle abnormalities in process interactions are starting to be exploited by deep learning models in order to transform detection techniques. The findings demonstrate that the current neural architectures can well model transient and fileless actions to a great extent, which may due to the effectiveness of these models are determined by restricted memory configuration (volatile), flexible API records (dynamics) and injection operation asymmetry properties. The substantial fragmentation in terms of datasets, evaluation protocol and reporting standard is one important take-away of this analysis. This introduces a lack of reproducibility for something notorious (it is hard to compare detection systems). Finally, the review also highlights how more interpretable and transparent models are needed since adversaries are increasingly exploiting uncertainties in learnt representations to deceive or tamper with deep learning detectors. Thus, it is of utmost importance to achieve adversarial robustness in future studies. Nonetheless, road signs in ongoing research indicate a few promising paths forward. It seems that the hybrid models with memory semantic-aware feature selection, time conscious shaped architectures and investigative forensics along with kindled neural encoding might be the ones to take it further. Collaborative activities for developing standardized assessment pipelines and more comprehensive benchmark data sets, which are specifically tailored to fileless and infection-related threats, will also be needed towards the continued progress. In summary, the above-mentioned code injection analysis, memory forensics and deep learning integration could be a solid platform for further detection systems. To enable defense mechanisms to be up-to-the-task against the latest in-memory attacks, strong foundation will require carefully crafted features and solid learning strategies as well as collaborative research practices.

9. Directions for Future Research

9.1. Modeling of Memory-Semantic Features

System calls, APIs sequences and overall control-flow patterns represent the main pre-existing behavior features. Yet, a myriad of destructive fileless operations happen right inside the memory of the process, such as stack pivoting, in-memory decoding, reflective DLL loading or spawning illegitimate threads. Memory-semantic approaches should be investigated for the context of memory allocation, permission adjustments and data-to-code transitions. The interaction of compact forensics with neural encoding, the evolution of representations in memory region over time and the role palette access graphs play are some of the questions deserving consideration. These techniques may help to detect subtle injection behaviors that are missed using shallow behavioral indicators. Multi-Hop, Cross-Process Behavioral Reasoning Even though fileless attacks often consist of multi-step, inter-process relationships or directly spread

through legitimate parent-child relations, most of the existing behaviors are analyzed by an individual process. Next-generation systems will need to be able to trace multi-hop sequences of behavior as malicious actions propagated across process boundaries. Some potential research directions in this space are: • graph neural network based architectures that model inter-process communication and inheritance, • hierarchical sequence models to combine temporal patterns across processes contexts, and • federated behavior traces to ensure model continuity across processes. This line of research is needed to comprehend complex attack sequences that take place in disjoint or modular stages.

Detection with extremely low latency and real-time learning Detection systems that can make decisions in a very short amount of time are necessary for the rapid execution of injection methods like APC-based loading or remote thread creation. Real-time deep learning frameworks with impressive results and low overhead should be the focus of future research. Models that make heavy use of pruning, compact transformers, real-time adaptive feature selection, and methods for streaming neural inference are all potential enhancements. Operational defenses in enterprise and cloud environments will require the ability to deal with fast events while maintaining accuracy. Robustness in combat and defensive ML tactics In order to confuse or undermine learning models, adversaries will increasingly develop strategies as deep learning gains prominence in malware identification. Research into adversarial training, robust representation learning, and defense-aware feature encoding is still at an early stage concerning fileless threats. Future frameworks could include: • enhancements to adversarial features based on real-world threat scenarios, • designs for detection that can stand up to noise or sequential changes in behavioral data, and • techniques for verifying the integrity of model inputs in real time. The creation of natural evasion-resistant systems remains a pressing and unexplored problem. Frameworks for Multi-Modal Fusion Malware that does not use files rarely exhibits consistent patterns within any one type of feature. A significant area of ongoing research is the integration of hardware-level signals, memory snapshots, process lineage graphs, API parameter semantics, opcode patterns, and hardware-level signals into unified models. Some of the key challenges include: • Cross-modal attention mechanisms, that can align information across different streams • Architectures that can connect disparate sampling rates and granularities • Fusion methodologies that are intuitive for diagnostic purposes Detection systems should identify evil stories that cannot be properly reconstructed from one data source due to the developments of multi-modal fusion.

REFERENCES

1. Aghakhani H, Gritti F, Mecca F, Lindorfer M, Ortolani S, Balzarotti D, Vigna G, Kruegel C (2020) When malware is packin' heat; limits of machine learning classifiers based on static analysis features. In: Proceedings of Symposium on Network and Distributed System Security (NDSS). The Internet Society
2. Ahmed F, Hameed H, ZubairShafiq M, Farooq M (2009). Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In: Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, AISec '09, pp 55–62
3. Bagher BM, Mahdi A, Asghar T (2019) HLMD: a signature-based approach to hardware-level behavioral malware detection and classification. J. Supercomput. 75(8):5551–5582 Article Google Scholar
4. Canzanese R, Mancoridis S, Kam M (2015) System call-based detection of malicious processes. In: 2015 IEEE International Conference on Software Quality, Reliability and Security, pp 119–124

5. Chen Xiaohui, HaoZhiyu, Li Lun, Cui Lei, Zhu Yiran, Ding Zhenquan, Liu Yongji (2022) Cruparamer: learning on parameter-augmented API sequences for malware detection. IEEE Tran Inf Forensics Secur 17:788–803 Article Google Scholar
6. Chen X, Tong Y, Du C, Liu Y, Ding Z, Ran Q, Zhang Y, Cui L, Hao Z (2022) Malpro: Learning on process-aware behaviors for malware detection. In: 2022 IEEE Symposium on Computers and Communications (ISCC), pp 01–07
7. Christodorescu M, Jha S, Seshia SA, Song D, Bryant RE (2005) Semantics-aware malware detection. In: 2005 IEEE Symposium on Security and Privacy (S P'05), pp 32–46
8. Darem A, Abawajy J, Makkar A, Alhashmi A, Alanazi S (2021) Visualization and deep-learning-based malware variant detection using opcode-level features. FutGenerComputSyst 125:314–323 Article Google Scholar
9. Han W, Xue J, Wang Y, Huang L, Kong Z, Limin M (2019) Maldae: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. ComputSecur 83:208–233 Article Google Scholar
10. Han W, Xue J, Wang Y, Liu Z, Kong Z (2019) Malinsight: a systematic profiling based malware detection framework. J NetwComputAppl 125:236–250 Article Google Scholar
11. Jindal C, Salls C, Aghakhani H, Long K, Kruegel C, Vigna G (2019) Neurlux: dynamic malware analysis without feature engineering. In: Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC '19, pp 444–455
12. Kalash M, Rochan M, Mohammed N, Bruce ND, Wang Y, Iqbal F (2018) Malware classification with deep convolutional neural networks. In: 2018 9th IFIP international conference on new technologies, mobility and security (NTMS), pp 1–5
13. Kolosnjaji B, Zarras A, Webster G, Eckert C (2016) Deep learning for classification of malware system call sequences. In: Kang BH, Bai Q, editors, AI 2016: Advances in Artificial Intelligence, pp 137–149
14. Kong D, Yan G (2013) Discriminant malware distance learning on structural information for automated malware classification. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 1357–1365
15. Lanzi A, Balzarotti D, Kruegel C, Christodorescu M, Kirda E (2010) Accessminer: Using system-centric models for malware protection. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS '10, pp 399–412
16. Li C, Cheng Z, Zhu H, Wang L, Lv Q, Wang Y, Li N, Sun D (2022) DMalNet: dynamic malware analysis based on API feature engineering and graph learning. ComputSecur 122:102872 Article Google Scholar
17. Li D, Li Q (2020) Adversarial deep ensemble: evasion attacks and defenses for malware detection. IEEE Trans Inf Forensics Secur 15:3886–3900 Article Google Scholar
18. Lu R (2019) Malware detection with LSTM using opcode language. arxiv:1906.04593
19. Min D, Park D, Ahn J, Walker R, Lee J, Park S, Kim Y (2018) Amoeba: an autonomous backup and recovery SSD for ransomware attack defense. IEEE Comput Architect Lett 17(2):245–248 Article Google Scholar
20. Moskovitch R, Feher C, Tzachar N, Berger E, Gitelman M, Dolev S, Elovici Y (2008) Unknown malcode detection using opcode representation. In: Intelligence and Security Informatics, pp 204–215
21. Naeem H, Ullah F, Naeem MR, Khalid S, Vasan D, Jabbar S, Saeed S (2020) Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. Ad Hoc Netw 105:102154 Article Google

22. Scholar Narouei M, Ahmadi M, Giacinto G, Takabi H, Sami A (2015) DLLMiner: structural mining for malware detection. SecurCommunNetw 8(18):3311–3322 Article Google Scholar
23. Qiang W, Yang L, Jin H (2022) Efficient and robust malware detection based on control flow traces using deep neural networks. ComputSecur 122:102871 Article Google Scholar
24. Sayadi H, Patel N, Sasan A, Rafatirad S, Homayoun H (2018) Ensemble learning for effective run-time hardware-based malware detection: a comprehensive analysis and classification. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp 1–6
25. Sebastio S, Baranov E, Biondi F, Decourbe O, Given-Wilson T, Legay A, Puodzius C, Quilbeuf J (2020) Optimizing symbolic execution for malware behavior classification. ComputSecur 93:101775 Article Google Scholar SL SD,
26. Sun G, Qian Q (2018) Deep learning and visualization for identifying malware families. In: IEEE Transactions on Dependable and Secure Computing
27. Tang A, Sethumadhavan S, Stolfo SJ (2014) Unsupervised anomaly-based malware detection using hardware features. In: Stavrou A, Bos H, Portokalidis G (eds) Research in attacks, intrusions and defenses. Springer International Publishing, Cham, pp 109–129 Chapter Google Scholar
28. Tang F, Ma B, Li J, Zhang F, Su J, Ma J (2020) Ransomspector: an introspection-based approach to detect crypto ransomware. ComputSecur 97:101997 Article Google Scholar
29. Tian D, Ying Q, Jia X, Ma R, Hu C, Liu W (2021) MDCHD: a novel malware detection method in cloud using hardware trace and deep learning. ComputNetw 198:108394 Article Google Scholar
30. Xiaofeng L, Fangshuo J, Xiao Z, Shengwei Y, Jing S, Lio P (2019) ASSCA: API sequence and statistics features combined architecture for malware detection. ComputNetw 157:99–111 Article Google Scholar
31. Alrawi, O., Ike, M., Pruett, M., Kasturi, R.P., Barua, S., Hirani, T., Hill, B., Saltaformaggio, B.: Forecasting malware capabilities from cyber attack memory images. In: Proceedings of the USENIX Security Symposium (2021)
32. Barabosch, T., Bergmann, N., Dombeck, A., Padilla, E.: Quincy: Detecting host based code injection attacks in memory dumps. In: Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA) (2017)
33. Barabosch, T., Eschweiler, S., Gerhards-Padilla, E.: Bee Master: Detecting Host Based Code Injection Attacks. In: Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA) (2014)
34. . Barabosch, T., Gerhards-Padilla, E.: Host-based code injection attacks: A popular technique used by malware. In: Proceedings of the International Conference on Malicious and Unwanted Software (MALWARE) (2014)
35. Bayer, U., Moser, A., Kruegel, C., Kirda, E.: Dynamic analysis of malicious code. Journal in Computer Virology (2006)
36. Biondi, F., Given-Wilson, T., Legay, A., Puodzius, C., Quilbeuf, J.: Tutorial: An overview of malware detection and evasion techniques. In: Proceedings of the In ternational Symposium on Leveraging Applications of Formal Methods (2018)
37. Borello, J.M., M´e, L.: Code Obfuscation Techniques for Metamorphic Viruses. Journal in Computer Virology (2008)
38. Botacin, M., de Geus, P.L., Gr´egio, A.: "VANILLA" Malware: Vanishing An tiviruses by Interleaving Layers and Layers of Attacks. Journal of Computer Vi rology and Hacking Techniques 15(4), 233–247 (Dec 2019)

39. Christodorescu, M., Jha, S., Kruegel, C.: Mining Specifications of Malicious Be havior. In: Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. Association for Computing Machinery (2007)
40. Galloro, N., Polino, M., Carminati, M., Continella, A., Zanero, S.: A systematical and longitudinal study of evasive behaviors in windows malware. Computers & Security (November 2021)
41. Griffin, K., Schneider, S., Hu, X., Chiueh, T.c.: Automatic generation of string signatures for malware detection. In: Proceedings of the International Workshop on Recent Advances in Intrusion Detection (RAID) (2009)
42. Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., Kirda, E.: UNVEIL: A Large-Scale, automated approach to detecting ransomware. In: Proceedings of the USENIX Security Symposium (2016)
43. Korczynski, D., Yin, H.: Capturing Malware Propagations with Code Injections and Code-Reuse Attacks. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS) (2017)
44. Gu, G., Liu, J.C.: Shadow Attacks: Automatically Evading System-Call-Behavior based Malware Detection. Journal in Computer Virology (2012)
45. Mantovani, A., Aonzo, S., Ugarte-Pedrero, X., Merlo, A., Balzarotti, D.: Prevalence and impact of low-entropy packing schemes in the malware ecosystem. In: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS) (2020)
46. Martignoni, L., Stinson, E., Fredrikson, M., Jha, S., Mitchell, J.C.: A Layered Architecture for Detecting Malicious Behaviors. In: Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID) (2008)
47. ., Patnaik, M., Rebeiro, C.: D-TIME: Distributed threadlessindepen dent malware execution for runtime obfuscation. In: Proceedings of the USENIX Workshop on Offensive Technologies (WOOT 19) (2019)
48. Quarta, D., Salvioni, F., Continella, A., Zanero, S.: Toward Systematically Exploring Antivirus Engines. In: Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) (2018)
49. Rossow, C., Dietrich, C.J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., Van Steen, M.: Prudent practices for designing malware experiments: Status quo and outlook. In: Proceedings of the IEEE Symposium on Security & Privacy (S&P) (2012)
50. Wang, Q., Hassan, W.U., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., Gunter, C.A., et al.: You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS) (2020)
51. Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L., Wang, G.: Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In: Proceedings of the USENIX Security Symposium (2020)
52. Omar Alrawi, Moses Ike, Matthew Pruett, RanjitaPaiKasturi, SrimantaBarua, TalebHirani, Brennan Hill, and Bren danSaltaformaggio. 2021. Forecasting malware capabilities from cyber attack memory images. In Proceedings of the USENIX Security Symposium.
53. Thomas Barabosch, Niklas Bergmann, Adrian Dombeck, and Elmar Padilla. 2017. Quincy: Detecting host-based code injection attacks in memory dumps. In Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17).
54. Thomas Barabosch, Sebastian Eschweiler, and ElmarGerhards-Padilla. 2014. Bee master: Detecting host-based code injection attacks. In Proceedings of the

Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'14).
55. Thomas Barabosch and ElmarGerhards-Padilla. 2014. Host-based code injection attacks: A popular technique used by malware. In Proceedings of the International Conference on Malicious and Unwanted Software (MALWARE'14).
56. Marcus Botacin, Paulo Lí–cio de Geus, and André Grégio. 2019. "VANILLA" malware: Vanishing antiviruses by in terleaving layers and layers of attacks. Journal of Computer Virology and Hacking Techniques 15, 4 (01 Dec. 2019), 233–247.
57. Min Du, Wenjun Hu, and William Hewlett. 2021. AutoCombo: Automatic malware signature generation through combination rule mining. In Proceedings of the ACM International Conference on Information & Knowledge Management.
58. Clemens Kolbitsch, Paolo MilaniComparetti, Christopher Kruegel, EnginKirda, Xiao-yong Zhou, Xiao Feng Wang, et al. 2009. Effective and efficient malware detection at the end host.. In USENIX Security Symposium,Vol.4.
59. David Korczynski and Heng Yin. 2017. Capturing malware propagations with code injections and code-reuse attacks. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17).
60. Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and DavideBalzarotti. 2021. Does every second count? Time-based evolution of malware behavior in sandboxes. In Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS'21).
61. Tamas K. Lengyel, Steve Maresca, Bryan D. Payne, George D. Webster, Sebastian Vogl, and AggelosKiayias. 2014. Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system. In Proceedings of the Annual Computer Security Applications Conference (ACSAC'14).
62. Lorenzo Martignoni, Elizabeth Stinson, Matt Fredrikson, SomeshJha, and John C. Mitchell. 2008. A layered architecture for detecting malicious behaviors. In Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID'08).
63. Mohammad N. Olaimat, MohdAizainiMaarof, and Bander Ali S. Al-Rimy. 2021. Ransomware anti-analysis and evasion techniques: A survey and research directions. In Proceedings of the International Cyber Resilience Conference (CRC'21).
64. JithinPavithran, Milan Patnaik, and Chester Rebeiro. 2019. D-TIME: Distributed threadless independent malware execution for runtime obfuscation. In Proceedings of the USENIX Workshop on Offensive Technologies (WOOT'19).
65. CERT Polska. 2012. More Human than Human- Flame's Code Injection Techniques. https://cert.pl/en/posts/2012/08/ more-human-than-human-flames-code-injection-techniques