Computers & Security 150 (2025) 104231

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Full length article

Unveiling the veiled: An early stage detection of fileless malware

Narendra Singh, Somanath Tripathy *

Department of Computer Science and Engineering, Indian Institute of Technology Patna, Bihta, Patna, 801106, India

ARTICLE INFO

Keywords:
Fileless malware attack
Memory analysis
Early stage detection
MITRE ATT&CK enterprise matrix

ABSTRACT

The threat actors continuously evolve their tactics and techniques in a novel form to evade traditional security solutions. Fileless malware attacks are one such advancement, which operates directly within system memory, leaving no footprint on the disk, so became challenging to detect. Meanwhile, the current state-of-the-art approaches detect fileless attacks at the final (post-infection) stage, although, detecting attacks at an early-stage is crucial to prevent potential damage and data breaches. In this work, we propose an early-stage detection system named *Argus* to detect fileless malware at early-stage. *Argus* extracts key features from acquired memory dumps of suspicious processes in real-time and generates explained features. It then correlates the explained features with the MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework to identify fileless malware attacks before their operational stage. The experimental results show that *Argus* could successfully identify, 4356 fileless malware samples (out of 5026 samples) during the operational stage. Specifically, 2978 samples are detected in the pre-operational phase, while 1378 samples are detected in the operational phase.

1. Introduction

Cybersecurity threats have evolved rapidly, with traditional malware adopting new forms and techniques to evade detection systems. Among these emerging threats, fileless malware poses significant challenges as it operates entirely in memory, leaving no trace on the disk (Liu et al., 2023). Fig. 1 illustrates the rise of fileless malware attacks against various organizations during the last 2-decades. For instance, in POSHSPY attack 2017 (Dunwoody, 2017), the russian state-sponsored APT29 group employed the "Living-off-the-Land" (LotL) technique, where threat actors hijack legitimate system tools (such as PowerShell and scripts) to execute malicious code. Similarly, APT19, Turla, APT33, APT34 and other threat groups used fileless techniques for enhanced evasion and increased persistence in the system (Barr-Smith et al., 2021).

Traditional detection techniques (Kolbitsch et al., 2010; Comparetti et al., 2010; Chen et al., 2022; Wong et al., 2023; Huang et al., 2021) often rely on dissecting standalone malware binaries or executing it in a sandbox environment. However, malwares often delete their binaries or employ hardware locking, to restrict them to run, only on infected machines. Moreover, the advanced threats, like fileless malware, operates entirely in memory without writing any files to disk, which creates a challenge for the detection techniques.

Of late, several approaches (Sanjay et al., 2018; Dai et al., 2018; Lee et al., 2021; Botacin et al., 2020; Bozkir et al., 2021; Demmese et al., 2023; Khalid et al., 2023; Kara, 2023; Saad et al., 2019; Borana et al., 2021) have been proposed to detect fileless attacks. These methods often involve comprehensive memory analysis using multiple tools like Volatility tool (Volatility Framework). However, these approaches require frequent context-switching between various tools, which imposing a significant cognitive burden on analysts, slowing down the investigation.

This work presents a novel early-stage detection technique for fileless malware attack, named *Argus*. *Argus* monitors the system (in real-time) for suspicious processes and appends their PIDs into queue for further analysis. A suspicious process PID is dequeued from the queue and its memory snapshots is captured using a command-line utility *ProcDump*.[1] Further, custom plugins are developed to extract key features from the memory snapshot. A feature explanation dataset is prepared from behavioural reports and is used to fine-tune the feature explainer model (Llama model), which generates explained features corresponding to those key features. Finally, these explained features correlate with the MITRE-attack-dataset, which is prepared from the MITRE ATT&CK enterprise matrix (MITRE ATT&CK). The MITRE-attack-dataset is used to fine-tune the early-stage detector (BERT model, combined with an MLP), to detect fileless attacks at early-stage.

This paper emphasizes the following key contributions:

* Corresponding author.
E-mail addresses: narendra_2021cs21@iitp.ac.in (N. Singh), som@iitp.ac.in (S. Tripathy).

[1] https://learn.microsoft.com/en-us/sysinternals/downloads/procdump

https://doi.org/10.1016/j.cose.2024.104231
Received 6 August 2024; Received in revised form 19 October 2024; Accepted 25 November 2024
Available online 16 December 2024

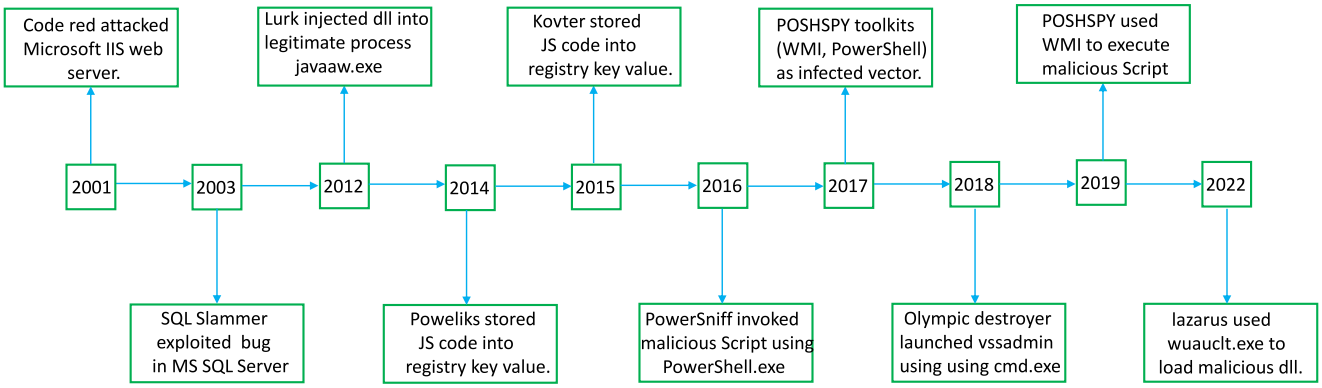

Fig. 1. Fileless malware attack from 2001–2022.

- This work introduces a framework called *Argus*, which utilizes the memory forensic and MITRE ATT&CK framework (enterprise matrix) to detect fileless malware in its early-stage.
- We have created two semantically similar datasets: Feature explanation dataset and MITRE-attack-dataset. The Feature explanation dataset is created from behavioural reports and is used to fine-tune a generative model for the feature explanation task. MITRE-attack-dataset derived from the MITRE ATT&CK knowledge base (enterprise matrix), which contains information on adversary tactics and techniques based on real-world observations. This dataset is used to fine-tune the BERT model, combined with an MLP, to detect fileless attacks at early-stages.
- Custom plugins are developed using volatility documentation[2] to extract key features from memory snapshot. Further, explained features are generated using the Llama model. Then, explained features automatically correlate it with the MITRE ATT&CK knowledge base to detect fileless attacks in the early-stage.
- *Argus* could successfully identify 4356 out of 5026 fileless malware samples at the operational stage when tested on a recent benchmark dataset.

This paper is structured as follows: Section 2 discusses the fileless malware attack lifecycle. Section 3 explores several related studies and highlights the differences between current SOTA. Section 4 introduces *Argus* framework and its components. Section 5 presents an in-depth analysis of the obtained results, along with a comparative evaluation against the state-of-the-art (SOTA). Section 6 shows implementation detail of *Argus*. Finally, Section 7 concludes the paper.

2. Fileless malware attack lifecycle

Fileless malware attacks typically unfold in several stages (Liu et al., 2023; Kara, 2023; Sudhakar and Kumar, 2020) to achieve their objectives, including data exfiltration and ransomware deployment. While the tactics and techniques employed by threat actors may evolve over time, the core stages of fileless malware attacks typically remain unchanged (Aqua report 2023). The attack life cycle of fileless malware can be segmented into four distinct stages, as shown in Table 1.

1. **Initial Stage:** In this stage, attacker gathers information and gains an initial foothold within the target system. To achieve this, adversaries could employ Phishing: Spearphishing Attachment technique (T1566.001)[3] stated in the MITRE enterprise matrix, where the attacker tricks the users by clicking malicious links or downloading files to establish foothold to the target machine.

Table 1
Fileless malware attack lifecycle and corresponding adversary goals.

Stages	Corresponding tactics	Adversary goals
Initial Stage	Reconnaissance	Collects data about the victim.
	Initial Access	Gain initial access on system
Pre-operation Stage	Execution	Execute malicious code to carry out an attack
	Persistence	Establishing a persistent on system.
	Privilege Escalation	Obtaining kernel level access.
Operational Stage	Defense Evasion	Evade detection.
	Credential Access	Steal user credentials.
	Discovery	Monitoring victim environment.
	Lateral Movement	Moving laterally within the system.
Final Stage	Collection	Collect relevant statistics
	Command and Control	Establish communication with compromised system
	Exfiltration	Stealing sensitive information
	Impact	Destroy or manipulate gathered data.

2. **Pre-operational Stage:** After establishing a foothold in the target system, the fileless attack progresses to the pre-operational stage. At this stage, the attacker establishes persistent connections to ensure that the adversary can access the system even after a reboot. To achieve this, the adversary could use the Modify Registry technique[4] (ID: T1112) mentioned in MITRE enterprise matrix to add entries in the Windows registry, hide configuration information, and run the malware automatically.
3. **Operational Stage:** In the operational stage, the adversary could use Command and Scripting Interpreter: PowerShell(ID: T1059.001)[5] stated in MITRE enterprise matrix to abuse the PowerShell script to download and execute malicious payloads directly in memory. The attacker exploit Living off the Land (LoTL) functionalities of legitimate programs such as mshta.exe to execute a malicious script.
4. **Final Stage:** At this stage, attackers exfiltrates sensitive data, such as financial records or intellectual property, from the compromised system. For this, the threat actor could utilize Exfiltration Over C2 Channel[6] technique from MITRE ATT&CK framework, where the attacker can steal data by exfiltrating it over an existing C2 channel.

3. Related works

Researchers have explored various studies to identify fileless attacks. According to Sanjay et al. (2018), fileless malware attacks are

[2] https://volatility3.readthedocs.io/en/stable/simple-plugin.html
[3] https://attack.mitre.org/techniques/T1566/001/
[4] https://attack.mitre.org/techniques/T1112/
[5] https://attack.mitre.org/techniques/T1059/001/
[6] https://attack.mitre.org/techniques/T1041/

Table 2
Comparison of the proposed approach (Argus) with current SOTA.

Existing work	Sample source	Analysis type	Feature/model used	MAF	ESD
Dai et al. (2018)	Open Malware Benchmark	Memory (Process Dump)	Grayscale Image/ ML models	No	No
Lee et al. (2021)	Hybrid Analysis and GitHub	Sandbox	APIs calls/ MITRE framework.	Yes	No
Botacin et al. (2020)	Collected Malware Samples	Memory (Process Dump)	Malicious patterns/AV accelerator	No	No
Bozkir et al. (2021)	Microsoft Malware Classification Challenge Dataset, Malimg, Malevis	Memory (Process Dump)	RGB image/ ML algorithms	No	No
Demmese et al. (2023)	Cobalt Strike beacon payload obtained from Palo Alto Networks.	Process Network Traffic	Grayscale Image/ML model.	No	No
Khalid et al. (2023)	Sample collected from AnyRun, PloySwarm and Virus Total	Memory (Full Dump)	Malicious patterns/ ML algorithms	No	No
Kara (2023)	Create own dataset (memory snapshot)	Memory (Full Dump)	Memory forensics/ Various tools	No	No
Trizna et al. (2024)	Avast-CTU and Speakeasy dataset	Sandbox	APIs, Network Activity/ Transformer encoder	No	No
Sajid et al. (2023)	VirusShare and MalShare	Sandbox	API patterns/ MSG Classifier	Yes	No
Zhou et al. (2023)	VirusShare and VirusTotal	Process Monitor	I/O behaviours, Network Traffic/ Ransomspector (Tang et al., 2020)	No	No
Proposed Work(Argus)	**VirusShare, AnyRun PolySwarm and** Barr-Smith et al. (2021), Kara (2023)	Memory (Process Dump)	**Feature Explainer/ Early Stage Detector**	**Yes**	**Yes**

MAF: MITRE ATT&CK Framework, ESD: Early Stage Detection.

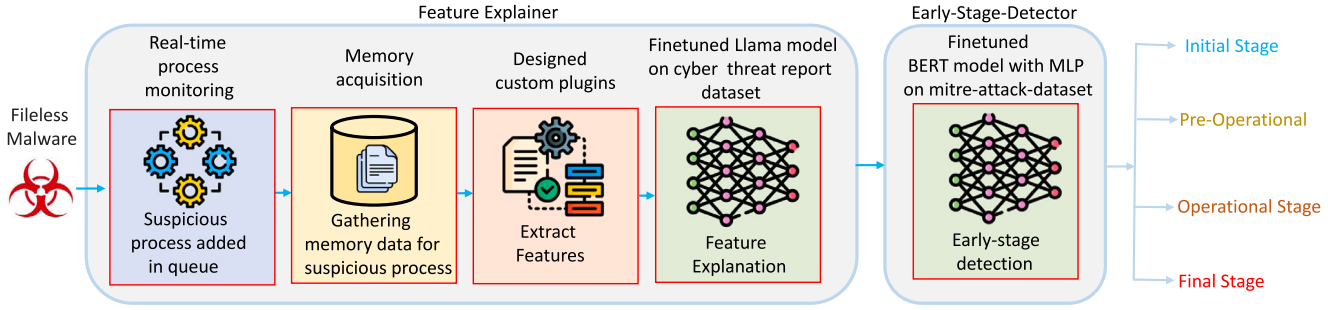

Fig. 2. Overview of our proposed approach.

categorized based on their execution (RAM-resident and script-based). Furthermore, they used a sandbox to monitor PowerShell executions within a controlled environment and verify scripts before execution using YARA rules to detect fileless malware. The method (Dai et al., 2018) extracted malware memory dump during runtime using a sandbox and converted the memory dump files into fixed-size grayscale images. Then, extracted features from these images are used to train a multilayer perceptron for malware classification. The study (Lee et al., 2021) analysed ten fileless malware using Cuckoo Sandbox and mapped attack technique with MITRE ATT&CK framework. The study also classified attacks into three categories: attack, evasion, and collection, based on techniques used in infiltration, evasion, and data manipulation. Authors in Botacin et al. (2020) proposed MINI-ME to detect fileless malware attacks based on near-memory and in-memory evaluation approaches. It focused on detecting code injection-based fileless attacks in both kernel and userland spaces on Windows systems. Further, it enhanced antivirus (AV) scan operations by accelerating fileless malware detection in memory. The approach in Bozkir et al. (2021) collects memory dump data using Procdump, capturing both physical and virtual memory data of target processes. The memory dump data are converted into RGB images and resized for representation. Additionally, GIST and HOG descriptors are used to extract visual features from the images. Finally, machine learning algorithms (Random Forest, XGBoost, Linear SVM, Sequential Minimal Optimization, and J48) are used for malware family classification. Demmese et al. (2023) proposed machine learning based fileless malware traffic classification method. It captured network traffic from the Cobalt Strike payload and converted it into grayscale images. Subsequently, they applied convolutional neural network and demonstrated how ML techniques can utilize image visualization to classify evasive malicious traces. The method (Khalid et al., 2023) captured memory snapshots from malicious and non-malicious samples in a virtual machine. It extracts features from memory dump using the Volatility memory forensics tool and uses random forest algorithm to detect fileless malware. They claimed that the model achieved an accuracy of 93.33% with a true positive rate of 87.5% and zero False Positive Rate. Kara (2023) proposed an approach to prevention and detection of fileless malware using memory analysis. They obtained malware samples from a Turkish cybersecurity firm and executed them in isolated environments to collect memory data. Moreover, they used several forensic tools, such as ProcDump, process monitor, autoruns, volatility tool, and many others, to examine memory data, and analyse system behaviour, network activity, and registry entries, to identify potential threats.

Alani et al. (2023) proposed an explainable obfuscated-malware detector that extracts five features from memory dump files and uses the extreme gradient boosting algorithm to identify malicious activity. The author in Moussaileb et al. (2018) introduced an Intrusion Detection System (IDS) for early detection of crypto-ransomware based on file system exploration. Similarly, Rhode et al. (2018) proposed a behaviour-based model for early-stage malware detection. The method utilized short behavioural data snapshots of executables and employed an ensemble of recurrent neural networks to identify malicious files. Authors in Trizna et al. (2024) proposed a model called Nebula, which

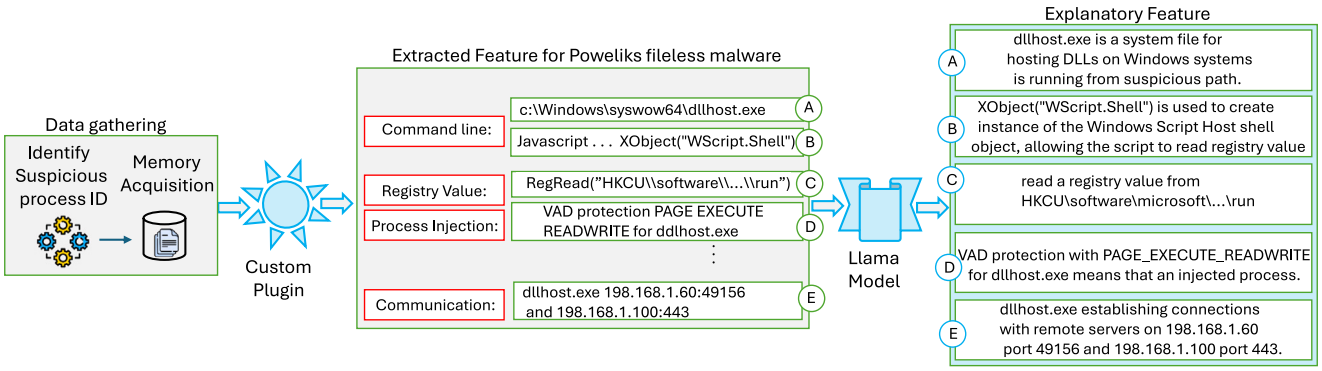

Fig. 3. The process of generating explained feature.

preprocesses multiple behavioural data, such as API calls, file operations, and network activity. The preprocessed data is utilized as input into a Transformer-based neural architecture to perform malware detection and classification. symbSODA (Sajid et al., 2023) is a configurable and verifiable cyber deception system that uses symbolic execution to extract Malicious Sub-Graphs (MSGs) from malware execution traces. These MSGs represent malware behaviours which are mapped manually to the MITRE ATT&CK framework, providing insights into malware tactics. Zhou et al. (2023) introduced the ANIMAGUS framework, which performs an imitation-based ransomware attack. The attack mimics the I/O behaviour patterns of benign programs in order to evade detection techniques.

Recent fileless malware detection techniques using memory forensic are summarized in Table 2. To the best of our knowledge, the existing SOTA method primarily focuses on detecting fileless malware threats after they have already been executed, without considering the prior stages. However, detecting fileless malware attacks in the later stage could have caused data exfiltration or system disruption. To address this limitation, we introduce *Argus*, a novel approach that aims to detect fileless malware attacks in early-stages.

4. *Argus:* the proposed framework

Fig. 2 presents the overview of our proposed approach *Argus*, which consists of two phases. (1) Feature explainer: monitors the system for suspicious processes and captures memory snapshots of the process. Then, it extracts key features using custom plugins and generates feature explanations. (2) The early-stage detector: fine-tunes (pre-trained BERT (Bidirectional Encoder Representations from Transformers) model, coupled with MLP (Multilayer perception)) on MITRE-attack-dataset, prepared from the MITRE ATT&CK enterprises matrix,[7] which correlates it with the generated feature explanation to identify the active stage of the fileless malware attack.

4.1. Feature explainer

The following subsections detail the data gathering, key feature extraction process, and the fine-tuning Llama model for the feature explainer task, as shown in Fig. 3.

4.1.1. Data gathering and key feature extraction

Argus continuously monitors the system for suspicious processes using built-in Windows Management Instrumentation (WMI).[8] WMI provides a powerful interface to access management information operates on Windows OS. We used Python WMI module[9] for identifying the suspicious processes in real-time, looking into the unusual names, high resource usage, and unusual network activity. Argus appends these process PIDs into the queue (Q) for further analysis. Then, it selects a suspicious process PID from the queue (Q) and automatically invokes ProcDump via "ma" command (using the Python subprocess module) to capture the memory dump of the suspicious process. Subsequently, custom plugins are designed to extract raw features from the acquired memory image and save them into a text file. Raw features of fileless malware are extracted from the following.

- Parent-Child Process Relationships: Fileless malware often exploits legitimate system processes to run malicious code, leading to abnormal parent–child process relationships. For example, a script (like VBScript or JavaScript) is executed by mshta.exe, which is spawned by an unexpected parent process.
- Tracing execution paths: Malware (such as Powelik malware) uses a non-standard path (`c:\Windows\syswow64\dllhost.exe`) to carry out malicious activity. Thus, identifying such deviations from standard execution paths could be due to malicious activity.
- Monitoring Sensitive Registry Keys: Fileless malware often modifies registry keys to ensure persistence after system reboots and avoid detection. Detecting unauthorized modifications to sensitive registry keys or security settings indicate the presence of fileless malware.
- Identifying Code Injection Attempts: Malware often uses code injection techniques to avoid detection by injecting malicious code into legitimate processes. So, code injection attempts could help to identify malicious activities that disguise themselves as legitimate processes.
- Recognizing Signs of Process Hollowing: Fileless malware frequently uses process hollowing to conceal its malicious activities. Recognizing signs of process hollowing (a process that appears to be legitimate but is executing malicious code) helps detect fileless malware.
- Suspicious Network Activity: Fileless malware often communicates with C2 servers to download additional payloads, or exfiltrate stolen data. So, anomalous network connections initiated by process could be a strong indicator of fileless malware activity.

The custom plugins are created by referring *Volatility* documentation.[10] The first step involves creating a context, that provides the environment for the subsequent operations. Then, we set the configuration in the context by specifying the parameters (such as lists, choices, and translation layers). To simplify the configuration process, *Volatility* includes a feature called "automagic" that helps to automate configuration processes. Subsequently, the custom plugin is executed within this configured context and generates results by rendering the

[7] https://attack.mitre.org/matrices/enterprise/
[8] https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page
[9] https://pypi.org/project/WMI/
[10] https://volatility3.readthedocs.io/en/latest/simple-plugin.html

Table 3
Example of regex to extract key features.

Regular expression	Identifier	Extracted key features
r"^[a-zA-Z]:\\(?:[^\\/:*?"<>| \r\n]+\\)*[^\\/:*?"<>\r\n]+ (?:\.[^\\/:*?"<>| \r\n]+)?$"	Execution Path	C:/Windows/Syswow64/ddlhost.exe
r"\b(HKCC| HKLM | HKCU | HKCR | HKU | HKEY_CLASSES_ROOT | HKEY_CURRENT_USER| HKEY_LOCAL_MACHINE | HKEY_USERS | HKEY_CURRENT_CONFIG) \\(?:[A-Za-z0-9-_\.]+\\?)*\b"	Path Registry	HKEY_LOCAL_MACHINE\Software\ Microsoft\Windows\CurrentVersion
r'VAD protection:\s+(PAGE_EXECUTE_READWRITE| PAGE_EXECUTE_READ| PAGE_EXECUTE| PAGE_READWRITE|PAGE_READONLY | PAGE_NOACCESS | PAGE_WRITECOPY| PAGE_EXECUTE_WRITECOPY)'	Process/Hollow Injection	VAD protection PAGE_EXECUTE_READWRITE
r'h[tx][tx]ps?:[\\/][\\/](?:[0-9a-zA-Z_./\\-]+)'	URLs	hxxp://110.10.79.65:80/ download/microsoft.jpg
r'scr ip \b(?:\d{1,3}\.){3}\d{1,3}:\d{1,5}\b' + r' dst ip\b(?:\d{1,3}\.){3}\d{1,3}:\d{1,5}\b'	Network Activity	src ip 198.168.1.60:49156 dst ip 198.168.1.100:443
. . .	. . .	. . .

Table 4
Example of created feature explanation dataset for feature explanation task.

Extracted feature	Explanation
C:/Windows/Syswow64/dllhost.exe	dllhost.exe running from suspicious path located c:/windows/syswow64
VAD protection PAGE EXECUTE READWRITE for ddlhost.exe	Injecting a malicious executable into dllhost.exe Windows process
198.168.1.60:49156 and 198.168.1.100:443 mshta.exe	network flow observed between 198.168.1.60 on port 49156 and 198.168.1.100 on port 443 for mshta.exe process
mshta.exe JavaScript "WScript.Shell"	This script uses mshta.exe to run obfuscated JavaScript, creating a WScript.Shell ActiveX object to interact with the Windows registry and execute malicious commands
RegRead("HKCU\\software \\Khfvb8b\\sdknfklnfo")	reads a registry key from the Windows Registry under the HKCU (HKEY_CURRENT_USER) hive to run on startup and persist on the system.
. . .	. . .

plugin in a TreeGrid format, which allows output in a structured format. Further, we preprocessed output (stored in the text files) using regular expressions (regex) to extract the key features. Some of the key features are presented in Table 3.

4.1.2. Llama model training

We selected Llama models for generating explained features due to their robust performance, cost-effectiveness, and open-source availability. It provides scalability and avoids the high recurring costs associated with proprietary models such as GPT-3, GPT-4 and PaLM.

We collected 19,178 behaviour reports from the threat report ATT&CK Mapper (TRAM) dataset and built a new dataset for the feature explanation task. Each behaviour report contains detailed descriptions of malicious activities in the form of sentences, labels, and document titles. Regular expressions are used to extract key features from these descriptions (as summarized in Table 3) and pair them with corresponding explanations. For example, behaviour report contains as sentence "REG_DWORD /f reg.exe add 'HKLM/SYSTEM/CurrentControlSet/Control/Terminal Server/v fSingleSessionPerUser/t, we extracted the key feature reg.exe add 'HKLM/SYSTEM/CurrentControlSet/Control/Terminal Server' along with its explanation: "adds or modifies a registry key to disable the restriction of single user sessions, allowing multiple sessions per user on the terminal server". Accordingly, we created feature explanation dataset containing 7617 data points for the feature explanation task, as summarized in Table 4.

Once feature explanation dataset is created, the cleaning process begins with NLTK tool kit to remove stop words (e.g., "and," "the," "is"), which do not contribute significantly. Lemmatization or stemming is applied to convert words to their base or root forms. Then, the dataset is prepared according to the Llama model template (shown in Fig. 4). The <s> and </s> tags enclose the entire sequence, marking the beginning and end of key features and their explanations. Within these tags, the <<SYS>> and <</SYS>> tags enclose the system message "explain". The key feature follows the <<SYS>> tag and precedes the </INST> tag, containing the corresponding feature explanation. This structured format ensures that the data is well-organized, facilitating effective training and inference for the model.

The embedding layer uses an *AutoTokenizer* to convert structured data into a token sequence and fix the sequence length. It adds <pad> tokens to shorter sequences and truncates longer sequences to ensure all inputs are the same length. The embedding layer then converts these sequences into numerical representations, such as token IDs and attention masks. Finally, it transforms these numerical representations (from a vocabulary size of 32,000) into dense vectors of size 5120. The model includes 40 decoder layers, each featuring a self-attention mechanism and a multi-layer perceptron (MLP). Each decoder layer also contains normalization layers and produces vector of 5120 dimensions. Finally, normalized input is applied to the output layer (lm_head) that maps the 5120 dim vector to the vocabulary size (32000) to generate logits for each token. The loss is computed between the predicted logits and actual next tokens. This is followed by backpropagation to calculate gradients and an optimization step to update the model parameters.

4.2. Early-stage detector

The following subsections detail MITRE-attack-dataset preparation and the training procedure of the early-stage detector, as illustrated in Fig. 5.

4.2.1. MITRE-attack-dataset creation

The MITRE ATT&CK enterprise matrix[11] is used to extract MITRE-attack-datasets for detecting fileless malware attacks in the early-stage.

[11] https://attack.mitre.org/matrices/enterprise/

Table 5
Part of prepare MITRE-attack-dataset.

Technique ID	Technique name	Malicious behaviour description	Tactics
T1566	Phishing: Spearphishing Link	Link to a malicious site that downloads a fake Flash Installer	Initial Access
T1059	Command and Scripting Interpreter: Windows Command Shell	Executing commands through cmd.exe to run malicious binaries	Execution
T1053	Scheduled Task/Job: Scheduled Task	Launched a scheduled task to gain persistence using the schtasks /create /sc command.	Persistence
T1055	Process Injection	Inject malicious code into the svchost.exe process for execution	Privilege Escalation
T1112	Modify Registry	Sets HKCU\Software\Microsoft\WindowsNT \CurrentVersion \Windows\ Load to point to its executable	Defense Evasion
T1555.004	Credentials from Password Stores: Windows Credential Manager	Utilized the PowerShell cmdlet Invoke-WCMDump to extract credentials from the Credential Manager on compromised systems	Credential Access
T1057	Process Discovery	Identify the processes running on the system using the tasklist command or task manager.	Discovery
T1021.006	Remote Services: Windows Remote Management	Utilized WinRM via PowerShell to run commands to execute the malicious payload.	Lateral Movement
T1571	Non-Standard Port	Used a custom binary protocol over TCP port 443 for C2	Command and Control
T1005	Data from Local System	Exfiltrated files stolen from local systems.	Exfiltration
T1529	System Shutdown/Reboot	leveraged the ExitWindowsEx function with the EXW_SHUTDOWN flag to remotely shut down compromised systems	Impact

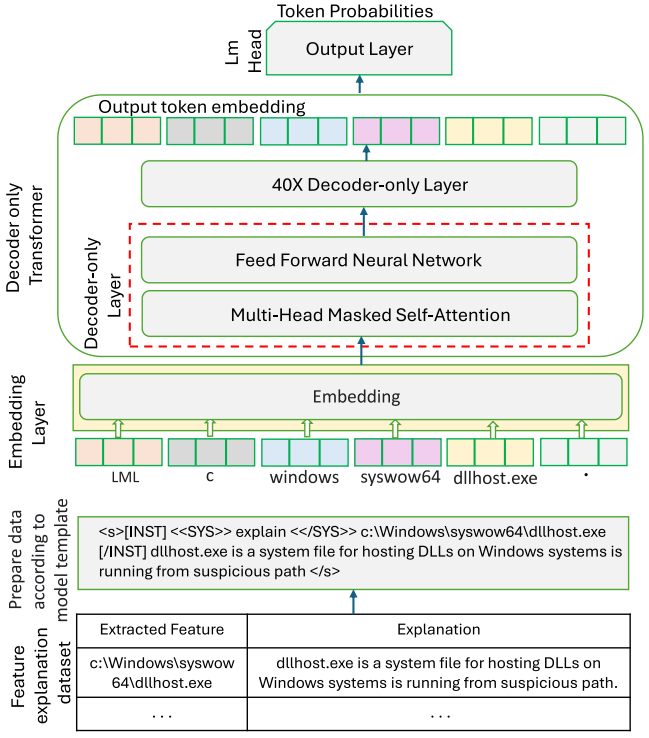

Fig. 4. Structure of feature Llama model.

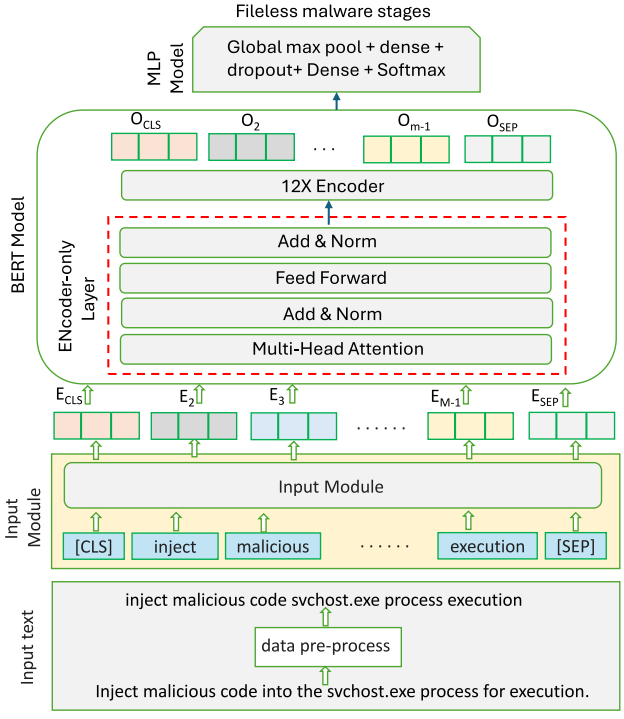

Fig. 5. Structure of early stage detector.

The MITRE ATT&CK (enterprise matrix) is a curated knowledge base that provides tactics, techniques and procedures used by adversaries across the *Attack life cycle*. The framework consists of 14 different tactics and 201 techniques, with 424 sub-techniques. Tactics represent the high-level objectives that adversaries want to achieve during an attack. Techniques are specific methods or procedures that threat actor uses to achieve their tactical objectives. For example, let an adversary aims to download and executes malicious payload. If could use the Command and Scripting Interpreter: PowerShell[12] technique, where threat actors exploit PowerShell commands and scripts to carry out malicious activities (such as downloading and executing malicious payloads). While creating the MITRE-attack-dataset, we referred to each tactic

[12] https://attack.mitre.org/techniques/T1059/001/

containing different techniques. These techniques consist of different procedural examples, each providing detailed functional descriptions of malicious behaviour. For instance, the technique Obfuscated Files or Information: Fileless Storage (T1027.011) from the MITRE ATT&CK matrix contains different procedure examples, including NETWIRE. It exhibits malicious behaviour by storing its configuration information in the registry under "`HKCU/Software/Netwire`". In this manner, we have created the MITRE-attack-dataset, which contains 15,764 distinct malicious behaviours and techniques to determine corresponding tactics as shown in Table 5.

4.2.2. Data pre-processing and model training

Algorithm 1 MITRE-attack-dataset Preprocessing.

Input: MITRE attack-data
Output: EmbeddedMatrix
Initialization:
s_length ← Maximum sequence length for padding.
sp_tokens ← {[CLS], [SEP], [PAD]}.
dataset ← [ ]

for each mal_des in MITRE attack-data **do**
  while mal_des_length > s_length – 2 **do**
    des ← mal_des[:s_length - 2]
    dataset ← dataset.append(des)
    mal_des ← mal_des[s_length - 2:]
  end while
  if mal_des_length < s_length – 2 **then**
    mal_des ← PadSeq to slength of mal_des
    dataset ← dataset.append(mal_des)
  end if
end for

for each line in dataset **do**
  tokens ← Tokenizer(line)
  inp_seq ← {[CLS]} + {[tokens]} + {[SEP]}
  seg_id ← CreateSegmentID(inp_seq)
  att_mask ← CreateAttentionMask(inp_seq)
  EmbeddedMatrix ← sum(inp_seq, seg_ids, att_masks)
end for

After data preparation, the data preprocessing step is carried out. The model uses malicious behaviour description data available in columns(3) of Table 5. Then, preprocess the data (remove punctuation, stop words, and convert all words to lowercase) and fix the sequence length. Subsequently, padding is applied to ensure uniform sequence length using a special padding token ([PAD]). Further, the preprocessed data is input into the input module.

The input module converts the preprocessed data into a token sequence ($T_{[CLS]}$, T_1, T_2, …, T_m, $T_{[SEP]}$) using a *WordPiece* tokeniser. Finally, it adds special tokens such as [CLS] (for classification) and [SEP] (to separate sentences) to the input sequence. Moreover, input module utilizes three distinct embedding layers (Token Input ID, Token Type ID, and Token Attention Mask ID) to generate different embedding vectors ($E_{[CLS]}$, E_1, E_2, ...E_m, $E_{[CLS]}$). These embeddings are combined to create a final embedded vector for each token. The data preprocessing process is shown in Algorithm 1.

The embedded matrix is fed into the BERT (Bidirectional Encoder Representations from Transformers) model. It extracts features from the sequence and produces a tensor with dimensions (None, 40, 768). It represents 40 tokens with 768-dimensional feature vectors. Next, global max pooling is applied to the BERT output, capturing informative features with shape (None, 768). This vector is then further processed through a Multi-Layer Perceptron (MLP), comprising a dense layer with 128 neurons and ReLU function. Further, a dropout layer is utilized to mitigate overfitting with a 0.5 rate, and another dense layer is incorporated with 64 neurons. Finally, the last layer of MLP has a softmax function with four neurons that yield probabilities for different stages of fileless malware. The model has over 108 million trainable parameters, showing its capability to learn complex patterns. The optimization process minimizes a categorical cross-entropy loss function throughout the training, utilizing backpropagation to update the early-stage detector parameters. The fine-tuning process is shown in the Algorithm 2.

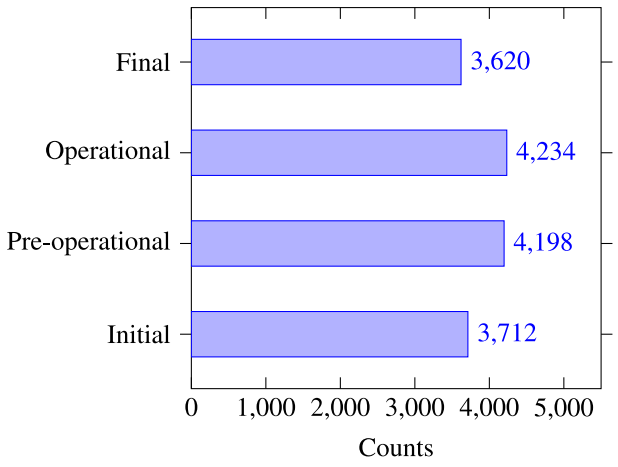

Fig. 6. MITRE-attack-dataset distribution across stages.

Algorithm 2 Fine-tuning Process of Early-Stage Detector.

Input: EmbeddedMatrix
Output: Trained EarlyStageDetector
Initialization:
$Epochs$ ← define epochs
$TrainingData$ ← Emb_Matrix
$BertModel$ ← LoadPretrainedBERTModel()
$TrainableLayer$ ← UnfreezeLastLayers($BertModel$)
$EarlyStageDetector$ ← $BertModel.append(MLP)$

for each $epoch$ **in** $Epochs$ **do**
  for each $data$ **in** $TrainingData$ **do**
    $Pred$ ← $EarlyStageDetector(data)$
    $Loss$ ← CatCrossEntropy($Pred, Labels$)
    Update EarlyStageDetector
  end for
end for

5. Experimental evaluation

5.1. Dataset

The feature explanation dataset, derived from the threat report ATT&CK Mapper (TRAM), comprises 7617 samples specifically designed for the feature explanation task. Each sample in the dataset includes key features, which are paired with their corresponding explanations. Notably, these explained features are semantically similar to MITRE-attack-dataset, enabling early detection of fileless attacks.

The MITRE-attack-dataset is prepared from the MITRE ATT&CK framework (enterprises matrix), which contains 15,764 distinct malicious behaviours. The MITRE-attack-dataset covers four distinct stages of the fileless malware attack lifecycle. The distribution across different stages of the MITRE-attack-dataset is depicted in Fig. 6.

Table 6
Fileless malware sample.

Country	Threat group	Collected fileless malware sample
Russia	APT29	276
China	APT10	239
Iranian	APT33	486
Russia	Trula	956
–	Gallmaker	128
Russia	TA505	396
Russia	Dragonfly	872
China	APT3	424
Ilker dataset (Kara, 2023)		1249
Total		5026

We have used recently released benchmark datasets[13] to evaluate the proposed approach (*Argus*). The dataset contains 1249 fileless malware samples, belonging to 11 different malware families. Additionally, 3777 APT malware is collected from various sources (sta; Barr-Smith et al., 2021; vir; san; Pol), which often employ fileless techniques to evade existing detection systems. The distribution of these samples is detailed in Table 6.

5.2. Design and requirements

5.2.1. Model configuration for Llama model

We split the feature explanation dataset in an 80:20 ratio, resulting in 6093 data points (80% of 7617) to fine-tune the feature explainer model and 1524 data points (20% of 7617) used for evaluation. During the fine-tuning of the Llama models, we employed QLoRA (Quantized Low-Rank Adaptation), with hyperparameters (lora_alpha = 16, lora_dropout = 0.1, and rank = 64). The QLoRA significantly reduces storage and computational requirements by quantizing the pre-trained model to 4 bits and freezing its parameters. Additionally, the training parameters include an output directory, training epochs (25), batch size (32), learning rate(2e-4), weight decay (0.001), and the paged_adamw_32 bit optimizer. Finally, the SFT trainer (supervised fine-tuning) fine-tuned the model by updating only the parameters of the QLoRa layers while keeping the rest of the model frozen.

5.2.2. Model configuration for early-stage detector model

We utilized fine-tune process to train early-stage detector, which requires comparatively less labelled data than training a model from scratch. We opted for this approach due to the small size of the MITRE-attack-dataset, which only consists of 15,764 distinct malicious behaviours. Out of these, we randomly selected 12,611 samples (which is 80% of the total) that span across different stages. The remaining 3153 data points (which is 20% of the total) are kept aside to assess the early-stage detector performance. When fine-tuning sequential models coupled with MLP, selecting appropriate hyperparameters is crucial for optimal performance. Key hyperparameters include the learning rate (2e-5), the batch size (32), training epochs (25), and weight decay(0.01). The AdamW optimizer is used, with beta values typically set to 0.9 for $\beta 1$ and 0.999 for $\beta 2$ with dropout rates (0.3) to mitigate overfitting.

5.3. Evaluation metrics

5.3.1. Evaluation metrics for explainer model

To evaluate feature explanation tasks, we employ two metrics: ROUGE (Recall-Oriented Understudy for Gisting Evaluation) and BERTScore. ROUGE is a set of metrics used to evaluate the quality of generated text by comparing the actual text. The key variants of ROUGE are defined in Eqs. (1) and (2).

ROUGE-1 measures the overlap of unigrams (single words) between the generated text and the actual text.

$$ROUGE-1 = \frac{|\ unigrams_{generated} \cap unigrams_{actual}\ |}{|\ unigrams_{actual}\ |} \tag{1}$$

Where $|\ unigrams_{generated} \cap unigrams_{actual}\ |$ denoted number of overlapping unigrams and $|\ unigrams_{actual}\ |$ denoted total number of unigrams in the actual text.

ROUGE-L measures the longest common subsequence (LCS) between the generated text and the actual text. The LCS takes into account the order of words, making ROUGE-L more sensitive to the structure of the text.

$$ROUGE-L = \frac{LCS(generated, actual)}{|actual|} \tag{2}$$

Where LCS(generated, actual) is the length of the longest common subsequence between the generated text and the actual text and |actual| referred length of the actual text.

BERTScore is an automated metric that evaluates the similarity between generated and actual text by harnessing the capabilities of the DistilBERT model. It tokenizes both predicted and actual sentences, computes cosine similarity between their embeddings to create a similarity matrix S, and calculates precision and recall by comparing token similarities. The detailed computation of precision, recall, and their harmonic mean (F1 score) are provided in Eqs. (3) to (5).

$$BERT(P) = \frac{1}{A}\sum_{c\in A}\max_{r\in P} S(c,r) \tag{3}$$

$$BERT(R) = \frac{1}{P}\sum_{r\in P}\max_{r\in A} S(c,r) \tag{4}$$

$$BERT(Fs) = \frac{2\times BERT(P)\times BERT(R)}{BERT(P)+BERT(R)} \tag{5}$$

Where P is the set of tokens in the predicted text, A is the set of tokens in the actual text, and S(c, r) is the similarity score between predicted token c and actual token r.

5.3.2. Evaluation metrics for early-stage detector

Here, we define fileless malware as the positive class. Therefore, True Positives (TP) refer to cases where the model accurately detects malware samples as malicious. Conversely, False Positives (FP) denote instances where the model wrongly identifies benign samples as malware. Similarly, True Negatives (TN) occur when benign samples are correctly identified as benign, while False Negatives (FN) indicate instances where the model incorrectly labels malware samples as benign. The model's performance is assessed using standard evaluation metrics, including Accuracy(A), Precision(P), Recall(P), and F1-score(Fs). These metrics are explained in detail as follows: **Accuracy** is calculated as the total number of correct predictions divided by the total number of predictions made, as shown in Eq. (6).

$$Accuracy(A) = \frac{(TP+TN)}{(TP+TN+FN+FP)} \tag{6}$$

Precision measures the proportion of correctly predicted positives out of all positive predictions made by the model. Precision is calculated using Eq. (7).

$$Precision(P) = \frac{TP}{(TP+FP)} \tag{7}$$

Recall: is also known as the true positive rate and measures the proportion of correctly positive predictions out of all actual positive observations. Recall is calculated using Eq. (8).

$$Recall(R) = \frac{TP}{(TP+FN)} \tag{8}$$

F1-score: is calculated by computing the harmonic mean of precision and recall, as demonstrated in Eq. (9).

$$F1-score(Fs) = \frac{(2\times Pre.\times Rec.)}{(Pre.+Rec.)} \tag{9}$$

[13] https://ilkerkara.karatekin.edu.tr/RequestDataset.html

Table 7
Feature explainer and early stage detector configuration.

Feature explainer		Early-stage detector	
Layer (type)	Output shape	Layer (type)	Output shape
Embedded Layer	Embedding(in_features=32000, out_features=5120)	InputLayer	(None, 40)
LlamaDecoderLayer (40x decoder layer)	Linear4bit((in_features=5120, out_features=5120)	TFBertModel (12x encoder layer)	(None, 40, 768)
LlamaMLP (gate_proj)	Linear4bit(in_features=5120, out_features=13824)	GlobalMaxPooling1D	(None, 768)
LlamaMLP (up_proj)	Linear4bit(in_features=5120, out_features=13824)	MLPDense	(None, 128)
LlamaMLP (down_proj)	Linear4bit(in_features=13824, out_features=5120)	MLPDropout	(None, 128)
lm_head	Linear(in_features=5120, out_features=32000)	MLPDense	(None, 64)
		MLPDense	(None, 4)

Table 8
Explainer evaluation for feature explanation task.

Models	ROUGE-1	ROUGE-L	BERT(P)	BERT(R)	BERT(Fs)
Llama2-7B	0.107	0.090	0.759	0.769	0.778
Lama2-13B	**0.294**	**0.240**	**0.840**	**0.835**	**0.837**
Llama-70B	0.105	0.094	0.748	0.792	0.769
Llama3-8B	0.152	0.137	0.820	0.831	0.826
Llama3-70B	0.163	0.133	0.789	0.802	0.795
Llama2-8B prompt	0.124	0.110	0.686	0.779	0.729
Llama3-8B prompt	**0.260**	**0.238**	**0.830**	**0.819**	**0.825**
Llama3-70B prompt	0.159	0.130	0.799	0.815	0.807

Table 9
Early-Stage Detector performance on MITRE-attack-dataset.

Models	Accuracy(A)	Precision(P)	Recall(R)	F1-score(Fs)
RNN	0.837	0.803	0.953	0.872
LSTM	0.871	0.862	0.970	0.913
Attn-LSTM	0.925	0.902	0.986	0.942
Bi-LSTM	0.912	0.879	0.879	0.879
Attn-BiLSTM	0.953	0.957	0.968	0.963
BERT-MLP	**0.9705**	**0.9706**	**0.9706**	**0.9706**

	Stages	Precision(P)	Recall(R)	F1-score(Fs)
BERT-MLP	Initial stage	**0.962**	**0.978**	**0.970**
	Pre-operational stage	**0.961**	**0.954**	**0.958**
	Operational Stage	**0.981**	**0.973**	**0.977**
	Final stage	**0.960**	**0.973**	**0.967**

Table 10
Argus performance on benchmark dataset.

Threat group	Total samples	Initial stage	Pre-operation stage	Operational stage	Final stage	Fail to detect
APT29	276	0	168	84	13	11
APT10	239	0	148	73	2	16
APT33	486	0	288	124	53	21
Trula	956	0	623	213	93	27
Gallmaker	128	0	76	42	2	8
TA505	396	0	237	85	56	18
Dragonfly	872	0	541	208	101	22
APT3	424	0	315	87	9	13
Ilker	1,249	0	582	462	182	23

5.4. Result analysis

The following subsections provide a detailed result analysis and performance comparison of Argus with the current state-of-the-art.

5.5. Feature explainer performance

Table 8 presents the result of various Llama models (different versions and sizes) on test data for the feature explanation task. The Llama3-8B model, with prompt engineering, achieved ROUGE-1, ROUGE-L, and BERT F1 scores of 0.260, 0.238, and 0.825, respectively. ROUGE-L scores for all models ranged from 0.090 to 0.240, demonstrating their ability to generate meaningful explanations. The Llama2-13B model recorded the highest BERT Precision, Recall, and F1 values of 0.840, 0.835, and 0.837, respectively. Additionally, it achieved ROUGE-1 and ROUGE-L scores of 0.294 and 0.240, respectively, indicating a higher n-gram overlap and longest common subsequence with the actual text. Furthermore, the Llama2-13B model demonstrates superior performance compared to other models. Notably, larger model does not always lead to better performance, as evidenced by the Llama3-70B model. Since larger models typically have more parameters, this could lead to overfitting on smaller datasets (as we have 7617 data points for the feature explanation task). Finally, we selected the Llama2-13B model as the feature explainer due to its superior performance in the feature explanation task.

5.6. Early-stage detector performance

Table 9 displays the performance of the early-stage detector on test data. We evaluated various sequence models, including Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (BiLSTM), Bi-LSTM with attention mechanism and pre-trained BERT model combined with MLP, all of which are trained on the MITRE-attack-dataset. The pre-trained BERT model combined with MLP (BERT-MLP) achieves the highest accuracy of 0.9705, with precision, recall, and F1-score values of 0.9706, 0.9706, and 0.9706, respectively. The Attn-LSTM and Attn-BiLSTM models also performed well, with accuracies of 0.925 and 0.953, respectively. The comparative analysis of these sequence models with the BERT-MLP has shown significant improvement in accuracy, ranging from 2%–14%. Furthermore, the BERT-MLP performance is evaluated across different stages. In the initial stage, it achieved an F-1 score of 0.970 with precision, and recall values of 0.962 and 0.978, respectively. At the pre-operational stage, it achieved precision and recall values of 0.961 and 0.954, respectively. In the operational and final stages, it achieved F-1 score values of 0.977 and 0.967, respectively. The results show that the BERT-MLP outperforms other sequential models, demonstrating its capability to detect fileless attacks. Therefore, the BERT-MLP is selected as the early-stage detector model due to its superior and consistent accuracy across different stages.

Table 11
Argus comparison with existing SOTA.

Methods	Detection accuracy	Initial stage	Pre-operational stage	Operational stage	Final Stage	Fail to detect
Ilker (Kara, 2023)	93.46%	0	0	0	4697	329
Ahmet (Bozkir et al., 2021)	97.21%	0	0	0	4885	141
Matilda (Rhode et al., 2018)	93.87%	0	221	497	3998	310
Routa (Moussaileb et al., 2018)	87.56%	0	0	1354	3046	626
Proposed *Argus*	**96.84%**	**0**	**2978**	**1378**	**511**	**159**

Table 12
Computational time comparison of argus with existing works.

Methods/process size	Time taken in second			
	1 KB–25 MB	26 MB–100 MB	101 MB–500 MB	501 MB–930 MB
Ilker (Kara, 2023)	34.223	112.173	898.453	1754.784
Ahmet (Bozkir et al., 2021)	19.673	64.325	121.56	166.631
Dai (Dai et al., 2018)	17.646	59.874	116.242	141.234
Khalid (Khalid et al., 2023)	28.162	97.567	140.143	301.230
Proposed (Argus)	11.252	44.742	79.801	136.343

5.7. Argus performance on benchmark dataset

Once the feature explainer and early-stage detector models are selected, their respective configurations are described in Table 7. Consequently, the performance of *Argus* evaluated across various APT threat groups, including the Ilker dataset, as shown in Table 10. The Ilker dataset, containing the highest number of fileless malware samples, out of 1249 samples, 582 samples were detected at the pre-operation stage, 462 were detected at the operational stage, 182 were detected at the final stage, and 23 samples were not detected. Similarly, for the Trula threat group, out of 956 APT malware samples, 623, 213, and 93 samples were detected at the pre-operation stage, operational stage and final stage, respectively, and 27 samples remain undetected. Notably, *Argus* did not detect any threats at the initial stage since it relies on memory analysis, which occurs after the malware has achieved initial access. Finally, *Argus* detected 2978 samples (out of total 5026 samples) at the pre-operation stage, 1889 samples at the subsequent stages, and 59 samples failed to detect. These results indicate that *Argus* is most effective in detecting fileless malware at the pre-operation stage.

5.8. Argus performance comparison with existing SOTA

We conducted a comparative analysis to demonstrate the effectiveness of *Argus* against a recent approach, as summarized in Table 11. All existing methods (Kara, 2023; Bozkir et al., 2021; Rhode et al., 2018; Moussaileb et al., 2018) are implemented on the same dataset outlined in Table 6. It is observed that from Table 11, the method (Kara, 2023) achieved a detection accuracy of 93.46% (4697 out of 5026 samples) at the final stage, leaving 329 samples undetected. Similarly, the Ahmet method (Bozkir et al., 2021) successfully detected 4885 fileless malware samples, achieving a detection accuracy of 97.21% at the final stage, with 141 samples remaining undetected. However, it is important to note that detection occurred at the final stage, where threat actors could exfiltrate data via the command and control server channel. Further, method (Moussaileb et al., 2018) successfully detected 221, 497, and 3998 samples at pre-operational, operational and final stages respectively. In contrast, *Argus* demonstrated robust performance by identifying 2978 fileless malware samples at the Pre-operational stage and 1378 samples at the Operational stage. The results demonstrate *Argus* efficiency in detecting fileless malware attacks at an early stage and outperform existing state-of-the-art methods with an impressive detection accuracy of 96.84%.

5.9. Computational performance comparison

The performance evaluation is conducted by selecting 1000 random processes of each different size. The analysis is focused on determining the average time taken by feature generation and early-stage detection for each memory dump. The Ilker method (Kara, 2023) took the longest time (an average of 1754.784 s for memory dump sizes between 501 MB and 930 MB), as it relies on several tools (such as volatility). Moreover, the methods proposed by Ahmet (Bozkir et al., 2021) and Dai (Dai et al., 2018) required less time compared to the Ilker method, since they used process memory dumps and generated image features to detect fileless malware. For *Argus*, the observed average detection time for different sizes of process dump in the range of (1 KB–25 MB), (26 MB–100 MB), (101 MB–500 MB), and (501 MB-932 MB) are 11.252, 44.742, 79.801 and 136.343 respectively, as depicted in Table 12. Consequently, *Argus* outperforms existing SOTA, which takes 11.252 s to analyse smaller processes and 136.343 s to analyse larger processes.

6. Discussion

In this section, we explain how *Argus* detects kovter fileless malware[14] at an early-stage. The Kovter fileless malware is executed within an isolated Windows environment. As a consequence, the *mshta.exe* process is created, which appears to be a normal Windows process. Simultaneously, *Argus* is monitoring the system, hence could detect the suspicious process (mshta, PID: 11768) and adds it to the queue (Q) for further analysis, as shown in Appendix Fig. 9. Then *Argus* selects the suspicious process from the queue (Q) and invokes ProcDump tool to acquire volatile memory (Random Access Memory). Next, it finds parent–child relationship to understand the dependency between the processes, as shown in Appendix Fig. 10. Furthermore, *Argus* extracts the execution path of the suspicious process (PID: 11768), searching for hidden indicators of fileless malware, which often operates in memory without leaving traces on disk. Appendix Fig. 11 shows the *mshta.exe* process (PID: 11768) tries to execute JavaScript code and reads the contents of a specified registry key under `HKCU/software`. Following this, Kovter drops *bdac.dat* (batch) file in a user-specific directory within `HKCU/Software/Microsoft/Windows/CurrentVersion/Run` to avoid detection.

[14] 49a748e9f2d98ba92b5af8f680bef7f2

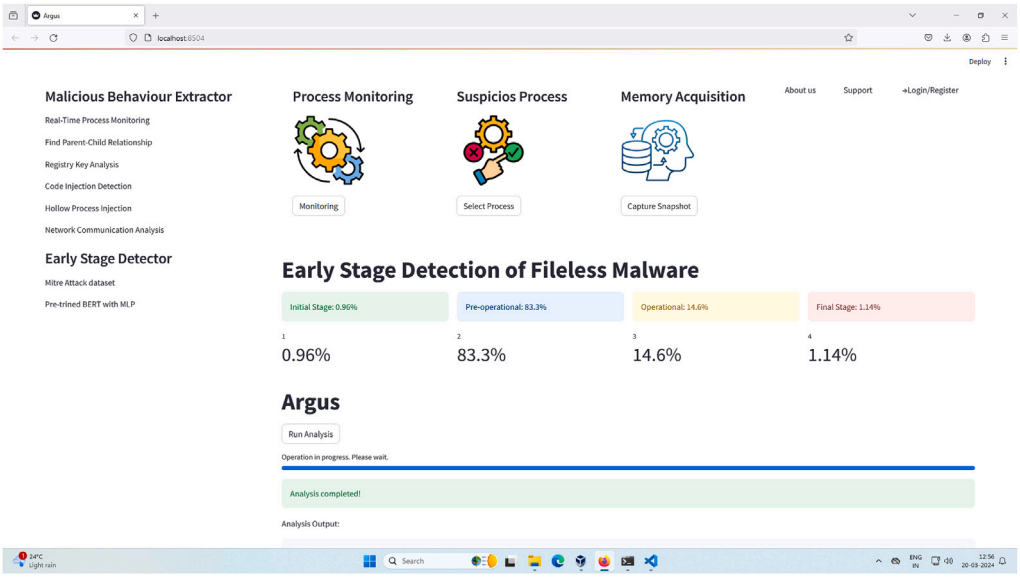

Fig. 7. Example of stage detection(Argus UI).

Also, *Argus* detects process network activity for malicious connections and could find that the "mshta.exe" process (PID: 11768) is trying to communicate with the c2 server, as shown in Appendix Fig. 12. Thus, *Argus* extracts key features using designed custom plugins and generates explained features, which are correlated with the MITRE-attack-dataset to detect fileless malware at an early-stage. Fig. 7 illustrates the prediction probabilities for Kovter fileless malware across different stages.

Like this, we experimented for all 5026 samples of fileless malware datasets shown in Table 6 and observed that *Argus* could able to detect 2978 samples in the pre-operational stage. During this stage, threat actors try to download and execute malicious payloads directly in memory using JavaScript or VB Script. The remaining 2048 undetected samples (false negatives at the pre-operational stage) progressed to the operational stage, as the malware used advanced techniques such as Living-off-the-Land (LotL) attacks to exploit legitimate system processes and passed the pre-opertaional stage without getting it detected. At he Operational stage, *Argus* could successfully detect 1378 samples and the remaining 670 undetected samples proceeded to the final stage. In this stage, *Argus* cloud able to detect 511 samples and 159 samples (false negatives). Here, attackers try to install additional malware, steal sensitive data, and move across the network to compromise other systems. Additionally, we collected a dataset of 1624 benign processes that were running on the Windows 10 operating system. *Argus* successfully detected 1563 processes as benign, while 61 processes were misclassified (False Positive). The true negative, false positive, true positive, and false negative at each stage are summarized in Fig. 8.

It may be noticed that some file-less malware are detected at the pre-operational stage, while others are detected at the operational or final stages. The malware detected at the pre-operational stage is due to its reliance on system components like scripts, registry keys, or processes that display unusual behaviour. However, Adversaries continuously evolve their tactics and techniques to evade detection, which makes challenging to detecting fileless malware at the pre-operation stage. Some fileless malware like CosmicDuke[15] (sample from APT29 Group) are fail to be detected at pre-operational stage but detected during their operational stage, as they leverage legitimate system processes (e.g., PowerShell, WMI or LotL techniques) to carry out malicious actions. Moreover, some advanced intelligent malware keeps them hiding without exploiting those features could not be identified until the final stage and Argus tries to filter them out in the operational stage. For instance, memory-resident malware like PlugX[16] (sample from APT 10 Group) reside entirely in memory and disappear after the system is rebooted. Since these malware do not use persistence mechanisms to remain in the system after a reboot, they can evade detection while performing malicious actions.

7. Conclusion

Fileless malware poses a significant challenge to cybersecurity defences by allowing attackers to bypass traditional security measures and maintain persistence within compromised systems. To address this, we introduced Argus, a robust framework for detecting fileless malware attacks at an early-stage. Argus continuously monitors system processes and acquires memory dumps for suspicious processes. A fine-tuned feature explainer model extracted key features from memory image of suspicious processes and generates explained feature using deep learning-based model. Further, the fine-tuned early-stage detector on prepared MITRE-attack-dataset and automatically correlated these explained features with the MITRE ATT&CK enterprises matrix to detect the present stage of fileless malware attack. The results show that *Argus* outperforms the existing state-of-the-art by identifying 86.66% (4356 out of 5026 samples) fileless malware attacks at the operational stage.

CRediT authorship contribution statement

Narendra Singh: Writing – original draft, Methodology, Formal analysis, Data curation, Conceptualization. **Somanath Tripathy:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

[15] f22606385080d35551e7f8e8f49b7de9.

[16] 841dfe3eaafe68cc0b989fbf55a34c9c.

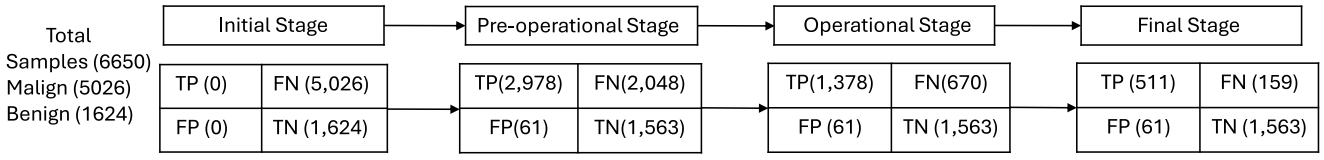

Fig. 8. True positive and False negative at each stage.

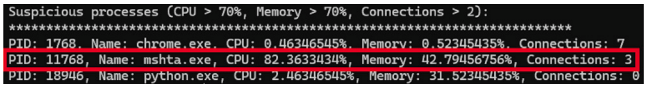

Fig. 9. Identify the suspicious process.

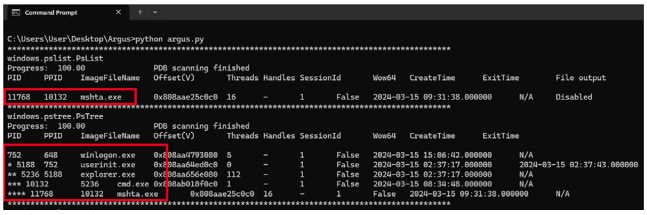

Fig. 10. Find parent–child relationship.

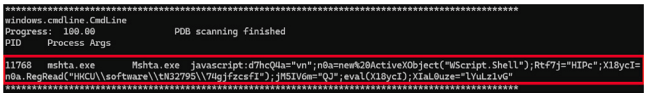

Fig. 11. mshta.exe execute JavaScript code within Windows registry.

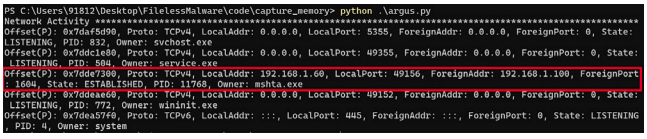

Fig. 12. Mshta.exe communicate with C2 server.

Appendix

Kovter fileless malware is executed within an isolated Windows environment, where the mshta.exe process is created, which appears like a normal Windows process. Meanwhile, *Argus* continuously monitors the system in real time using the WMI interface (through a Python script) to detect suspicious processes. In this case, it flagged the mshta.exe process with PID 11768, due to unusual process name, high resource usage, and abnormal network activity (Fig. 9).

Next, *Argus* invokes Procdump with the "-ma" argument to capture a memory dump of the suspicious mshta.exe process (PID 11768). Subsequently, *Argus* examines the relationships between different processes to understand their dependencies, as shown in Fig. 10. Further analysis reveals that mshta.exe tried to execute JavaScript code and to read a registry key located under `HKCU\software\Khfvb8b\sdknfklnfo` as shown in Fig. 11. Moreover, *Argus* also analyzes sensitive registry keys, code injection attempts, signs of process hollowing, and suspicious network connections initiated by the process. Particularly, *Argus* detects that the mshta.exe process (PID 11768) is trying to communicate with a command-and-control (C2) server, as shown in Fig. 12.

Thus, *Argus* extracts key features every 2 s using custom-designed plugins and generates explained features. These explained features are correlated with the MITRE-attack-dataset, which is derived from the MITRE ATT&CK Enterprise Matrix. The ATT&CK matrix is a comprehensive framework that provides a taxonomy of adversary tactics, techniques, and procedures (TTPs) used in real-world cyberattacks. By utilizing this dataset, the early-stage detector is able to bridge the semantic gap between low-level data (i.e., the explained features) and high-level MITRE tactics and techniques to detect fileless malware at an early-stage.

Data availability

No data was used for the research described in the article.

References

APT Malware Dataset. https://github.com/cyber-research/APTMalware [Online: last accessed jan 2024].

VirusShare: a repository of malware samples, Available: https://virusshare.com [Online: last accessed march 2024].

Anyrun: online Malware Analysis Sandbox, Available: https://app.any.run/ [Online: last accessed march 2024].

PolySwarm: crowdsourced Threat Detection, Available: https://polyswarm.network/ [Online: last accessed march 2024].

Alani, M.M., Mashatan, A., Miri, A., 2023. XMal: A lightweight memory-based explainable obfuscated-malware detector. Comput. Secur. 133, 103409.

Aqua report 2023, Fileless Attacks, Available: https://www.aquasec.com/cloud-native-academy/application-security/fileless-attacks/ [Online: last accessed March 2024].

Barr-Smith, F., Ugarte-Pedrero, X., Graziano, M., Spolaor, R., Martinovic, I., 2021. Survivalism: Systematic analysis of windows malware living-off-the-land. In: 2021 IEEE Symposium on Security and Privacy. SP, IEEE, pp. 1557–1574.

Borana, P., Sihag, V., Choudhary, G., Vardhan, M., Singh, P., 2021. An assistive tool for fileless malware detection. In: 2021 World Automation Congress. WAC, IEEE, pp. 21–25.

Botacin, M., Grégio, A., Alves, M.A.Z., 2020. Near-memory & in-memory detection of fileless malware. In: Proceedings of the International Symposium on Memory Systems. pp. 23–38.

Bozkir, A.S., Tahillioglu, E., Aydos, M., Kara, I., 2021. Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. Comput. Secur. 103, 102166.

Chen, X., Hao, Z., Li, L., Cui, L., Zhu, Y., Ding, Z., Liu, Y., 2022. Cruparamer: Learning on parameter-augmented api sequences for malware detection. IEEE Trans. Inf. Forensics Secur. 17, 788–803.

Comparetti, P.M., Salvaneschi, G., Kirda, E., Kolbitsch, C., Kruegel, C., Zanero, S., 2010. Identifying dormant functionality in malware programs. In: 2010 IEEE Symposium on Security and Privacy. IEEE, pp. 61–76.

Dai, Y., Li, H., Qian, Y., Lu, X., 2018. A malware classification method based on memory dump grayscale image. Digit. Invest. 27, 30–37.

Demmese, F.A., Neupane, A., Khorsandroo, S., Wang, M., Roy, K., Fu, Y., 2023. Machine learning based fileless malware traffic classification using image visualization. Cybersecurity 6 (1), 32.

Dunwoody, M., 2017. Dissecting one of apt29's fileless wmi and powershell backdoors (poshspy). Blog.

Huang, Y.-T., Lin, C.Y., Guo, Y.-R., Lo, K.-C., Sun, Y.S., Chen, M.C., 2021. Open source intelligence for malicious behavior discovery and interpretation. IEEE Trans. Dependable Secure Comput. 19 (2), 776–789.

Kara, I., 2023. Fileless malware threats: Recent advances, analysis approach through memory forensics and research challenges. Expert Syst. Appl. 214, 119133.

Khalid, O., Ullah, S., Ahmad, T., Saeed, S., Alabbad, D.A., Aslam, M., Buriro, A., Ahmad, R., 2023. An insight into the machine-learning-based fileless malware detection. Sensors 23 (2), 612.

Kolbitsch, C., Holz, T., Kruegel, C., Kirda, E., 2010. Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. In: 2010 IEEE Symposium on Security and Privacy. IEEE, pp. 29–44.

Lee, G., Shim, S., Cho, B., Kim, T., Kim, K., 2021. Fileless cyberattacks: Analysis and classification. ETRI J. 43 (2), 332–343.

Liu, S., Peng, G., Zeng, H., Fu, J., 2023. A survey on the evolution of fileless attacks and detection techniques. Comput. Secur. 103653.

MITRE ATT&CK, Enterprise Matrix, Available: https://attack.mitre.org/matrices/enterprise/windows/ [Online: last accessed March 2024].

Moussaileb, R., Bouget, B., Palisse, A., Le Bouder, H., Cuppens, N., Lanet, J.-L., 2018. Ransomware's early mitigation mechanisms. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. pp. 1–10.

Rhode, M., Burnap, P., Jones, K., 2018. Early-stage malware prediction using recurrent neural networks. Comput. Secur. 77, 578–594.

Saad, S., Mahmood, F., Briguglio, W., Elmiligi, H., 2019. Jsless: A tale of a fileless javascript memory-resident malware. In: Information Security Practice and Experience: 15th International Conference, ISPEC 2019, Kuala Lumpur, Malaysia, November 26–28, 2019, Proceedings 15. Springer, pp. 113–131.

Sajid, M.S.I., Wei, J., Al-Shaer, E., Duan, Q., Abdeen, B., Khan, L., 2023. SymbSODA: configurable and verifiable orchestration automation for active malware deception. ACM Trans. Privacy Secur. 26 (4), 1–36.

Sanjay, B., Rakshith, D., Akash, R., Hegde, V.V., 2018. An approach to detect fileless malware and defend its evasive mechanisms. In: 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions. CSITSS, IEEE, pp. 234–239.

Sudhakar, Kumar, S., 2020. An emerging threat fileless malware: a survey and research challenges. Cybersecurity 3 (1), 1.

Tang, F., Ma, B., Li, J., Zhang, F., Su, J., Ma, J., 2020. RansomSpector: An introspection-based approach to detect crypto ransomware. Comput. Secur. 97, 101997.

TRAM, Threat Report ATT&CK Mapper, Available: https://github.com/center-for-threat-informed-defense/tram/tree/main/data [Online: last accessed Feb 2024].

Trizna, D., Demetrio, L., Biggio, B., Roli, F., 2024. Nebula: Self-attention for dynamic malware analysis. IEEE Trans. Inf. Forensics Secur..

Volatility Framework, Volatile memory extraction utility framework, Available: https://github.com/volatilityfoundation/volatility [Online: last accessed Feb 2024].

Wong, G.-W., Huang, Y.-T., Guo, Y.-R., Sun, Y., Chen, M.C., 2023. Attention-based API locating for malware techniques. IEEE Trans. Inf. Forensics Secur..

Zhou, C., Guo, L., Hou, Y., Ma, Z., Zhang, Q., Wang, M., Liu, Z., Jiang, Y., 2023. Limits of i/o based ransomware detection: An imitation based attack. In: 2023 IEEE Symposium on Security and Privacy. SP, IEEE, pp. 2584–2601.

Narendra Singh Lodhi is currently pursuing his Ph.D. at the Department of Computer Science and Engineering, from Indian Institute of Technology, Patna, India. He received his M.Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Patna, India. Prior to this, he completed Bachelor of Engineering (Information Technology) from Institute of Engineering Technology, Indore, India. His research interests are Malware Analysis and Machine Learning Security.

Prof. Somanath Tripathy received his Ph.D. from IIT Guwahati in 2007. At present, he is a Professor in the Department of Computer Science and Engineering at the Indian Institute of Technology, Patna. He joined the faculty in December 2008. He was the Associate Dean, Academics from January 2016 to March 2017 and the Associate Dean, Administration from July 2021 to Nov 2023 at IIT Patna. His research interests include Cybersecurity, Malware detection and Secure machine learning, lightweight cryptography, Blockchain,. He has two patents and published more than 120 research papers in various reputed journals and conferences. He has been the Principal Investigator of several projects related to security. A malware detection app developed by his team is presented to BPRD, MHA (as a part of the sponsored project). Dr. Tripathy is currently an editor of the IETE Technical Review and an associate editor of the Multimedia Tools and Applications, an International Journal.