APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 7 No. 1 (2026) p. 978-999
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

DeepInvisi: A Fileless Malware Detection Tool Using Deep Learning Approach

Yong Wei Di[1], Isredza Rahmi A Hamid[1]*

[1] *Fakulti Sains Komputer dan Teknologi Maklumat*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: rahmi@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2026.07.01.053

Article Info

Received: 3 August 2025
Accepted: 15 June 2026
Available online: 30 June 2026

Keywords

Fileless malware, fileless malware detection, deep learning, transformer model, bilstm model

Abstract

Fileless malware is malicious code operating directly in a computer's memory, bypassing traditional hard disk detection. In the digital age, it is a common threat to use legitimate tools to evade security, making it extremely challenging to identify. This is critical as traditional anti-malware solutions like Kaspersky, TotalAV, and Aqua Security often have limited capabilities in comprehensively detecting these sophisticated, memory-resident threats, leaving organizations vulnerable. This project introduces DeepInvisi, a tool addressing this detection gap by using a hybrid deep-learning approach. Transformer models will analyze complex sequential data, while Bidirectional Long Short-Term Memory (BiLSTM) models handle temporal dependencies for comprehensive analysis of PowerShell command histories and memory activities. The proposed solution targets any users or normal people with electronic devices, aiming to protect them from fileless malware. The expected outcome is a real-time detection tool offering automatic updates, instant threat blocking, and real-time notifications for individual users and the public.

1. Introduction

Fileless malware, a significant and rising cybersecurity threat[1], operates in memory without leaving disk traces, which challenging the traditional antivirus detection. It leverages legitimate system tools like PowerShell through stages of delivery, persistence, and execution[2],[3]. Tools like VirusTotal and Kaspersky use behavioral analysis and memory scanning, conventional methods struggle with fileless malware's in-memory nature and minimal evidence[4],[5],[6], which lead to high false positives[7]. The rapid evolution of fileless malware also renders existing strategies quickly outdated.

This project introduces DeepInvisi, a tool that employs deep learning to analyze system behavior patterns for detecting fileless malware. Its objectives are to design behavioral-based detection models using a hybrid Transformer and Bidirectional Long Short-term Memory approach and then evaluate its accuracy and false positive rate through alpha and beta testing. DeepInvisi is intended for individual and general users, providing real-time background monitoring and threat response via alerts. It comprises four main modules that are user interface for control; real-time monitoring for collecting data like PowerShell commands, and memory activity, a deep learning analysis engine using the Transformer-BiLSTM model and a threat detection and response module that blocks threats, assigns risk scores, and issues real-time notifications.

The rest of this paper is organized as follows. Section 2 discusses related work on fileless malware. Section 3 discusses methodology; section 4 is about design and analysis. Section 5 discusses the results of the DeepInvisi Fileless Malware Detection Tool and section 6 is the conclusion.

2. Related work

Section 2 discusses fileless malware, operation of fileless malware, type of fileless malware, techniques for detecting fileless malware, transformers model, LSTM model, and existing tools.

2.1 Fileless Malware

Fileless malware is an advanced form of malicious software that evades traditional detection methods by operating entirely within a device's memory (RAM) instead of leaving files on the hard disk[3]. Unlike traditional malware, which relies on detectable files or programs, fileless malware exploits legitimate built-in tools like PowerShell, Windows Management Instrumentation (WMI), and other trusted software to inject malicious scripts directly into the system's memory. This makes it difficult for signature-based antivirus systems to detect as it is good at hiding and can cause serious harm[8]. Its crucial to urgently find better solutions to spot it as it become more common nowadays[1].

2.2 Operation of Fileless Malware

Fileless malware operates in three key phases: delivery, persistence, and execution[3]. During the delivery phase, attackers exploit system vulnerabilities or trick users into opening malicious files like Office documents or PDFs with embedded macros. In some cases, they exploit network vulnerabilities to inject code directly into the system's memory. Once delivered, the malware embeds itself in volatile memory (RAM), bypassing traditional file storage mechanisms.

The persistence phase involves the malware ensuring it remains active even after the system reboots[5]. This is often achieved by modifying system components, such as the Windows registry or task scheduler, to create scripts or tasks that execute on restart. Tools like WMI and PowerShell are frequently used to make these modifications, enabling the malware to run without leaving detectable files.

Finally, in the execution phase, the malware performs its harmful activities by leveraging trusted system tools and scripts. It might use command-line utilities like msiexec, BITSAdmin, or CertUtil to execute payloads, steal data, or deepen system infiltration.

2.3 Type of Fileless Malware

Fileless malware operates without traditional files, using trusted system tools and running primarily in memory (RAM), making it difficult to detect. There are four types of fileless malware such as memory-only malware, script-based malware, fileless cryptojacking, and "Living off the Land". Memory-only malware exists in RAM and vanishes after a reboot, and script-based malware uses scripting languages like PowerShell to execute malicious commands, often bypassing detection[9][10]. Fileless cryptojacking malware exploits resources to mine cryptocurrency, employing tools like PowerShell with techniques like Base64 encoding to evade detection[11]. Besides, "Living Off the Land" (LOL) attacks leverage trusted utilities like rundll32.exe and regsvr32.exe for malicious purposes, blending into legitimate processes and evading traditional defenses[12]. "Living off the Land" attacks are the most malicious due to their ability to misuse trusted system tools.

2.4 Detection Techniques for Fileless Malware

Detecting fileless malware requires identifying malicious activities without relying on traditional file-based methods. Behavior-based detection analyzes patterns, such as PowerShell command monitoring, which tracks unauthorized scripts or commands indicative of malicious intent[13]. Advanced techniques include memory dumps, which capture and analyze in-memory activity to reveal malicious processes[14], and real-time monitoring using machine learning and dynamic analysis to detect anomalies and adapt to evolving threats. Table 1 shows the behavior analysis features like Memory Dump and PowerShell Command.

Table 1 *Behavior Analysis Feature for Detecting Fileless Malware*

Behavior Analysis Feature	Fileless Malware Type	Detection Details	Features
Memory Dump	Memory-Only Malware	Captures in-memory-only malware by recording all active processes and data in RAM	1. Direct Memory Injection 2. Excessive Memory Usage 3. In-memory data manipulation
PowerShell Commands	Script-Based Malware	Identifies script-based malware by monitoring PowerShell commands for unusual behavior.	1. Remote file downloads 2. Privilege escalation 3. Suspicious script execution
	Living-Off-the-Land (LOL) Malware	Detect LOL malware by tracking PowerShell commands used to execute scripts, download additional payloads, or manipulate system functions.	1. PowerShell used maliciously 2. Running Abnormal Privilege

2.5 Deep Learning Algorithms for Fileless Malware Detection

In malware detection, deep learning models like Transformers[15],[16],[17] and Bidirectional Long Short-Term Memory(BiLSTM)[18] can analyze sequences of data, such as the patterns of actions a program takes. The transformer model in Fig. 1 uses a self-attention mechanism, which looks at the entire sequence of actions all at once. This helps it understand the context and relationships between different actions, even if they happen far apart. This is useful in malware detection because malicious software often shows unusual patterns in its memory dump, and PowerShell commands and Transformers can pick up on these clues. Self-attention as shown in Equation[19]:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

In attention mechanisms, the query matrix (Q) represents the current item of interest, while the key matrix (K) provides the context of all available items. The value matrix (V) holds the actual informational content of these items. By comparing the query with all keys, the mechanism determines how much attention to pay to each corresponding value, with the dimensionality of the key vectors ($\sqrt{d_k}$) used as a scaling factor to stabilize the process.

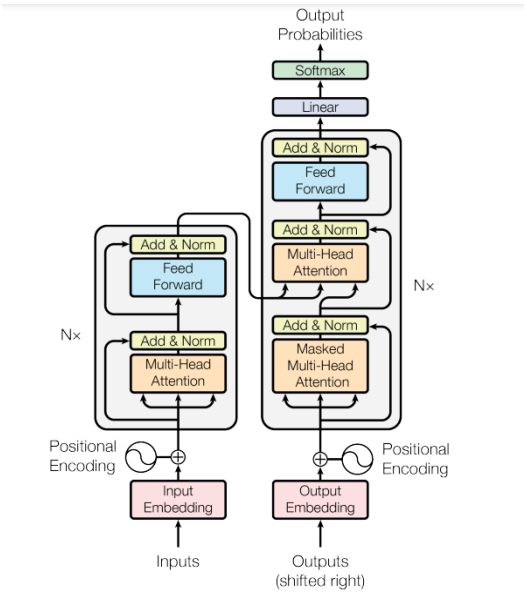

Fig. 1 *Transformer Model Architecture*[17]

The LSTM model in Fig. 2 is designed to remember important information over long periods. It uses special "gates" that help it decide what information to keep and what to forget. This helps analyze sequences of system

calls made by a program because LSTM can track how these calls change over time, spotting suspicious behavior that may indicate malware. Building on this, a Bidirectional LSTM (BiLSTM) further enhances this capability by processing in both forward and backward directions. This dual perspective allows the BiLSTM to capture a more comprehensive context of how calls relate to each other over time, improving its ability to identify subtle, suspicious patterns indicative of malware. By combining the strengths of both models, a reliable system for detecting harmful software is based on the way how it interacts with the operating system.

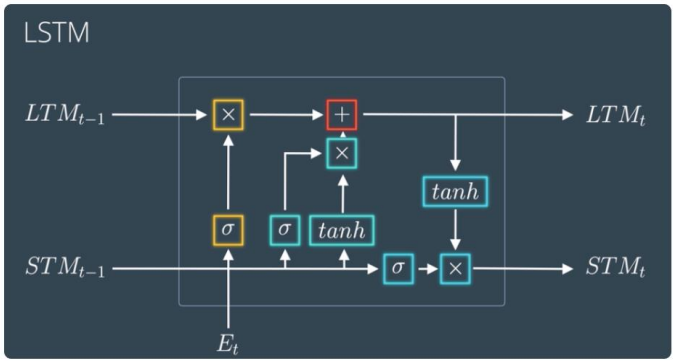

Fig. 2 *LSTM Model Architecture*[19]

2.6 Study on Related Tools

Detecting fileless malware is a major challenge because it does not rely on traditional file-based signatures and often operates directly in memory. This section examines key tools specifically designed in TotalAV[20], Kaspersky[5], and Aqua Security[6] to tackle fileless malware, focusing on detection capabilities and methodologies. They are evaluated based on platform, user interface and control panel, real-time monitoring and data collection, data collection components, fileless malware detection, memory and process analysis, machine learning model, operate without internet, threat detection and response and programming language.

Table 2 compares DeepInvisi with existing tools like TotalAV, Kaspersky, and AQUA Security across features such as platform support, real-time monitoring, fileless malware detection capabilities, and the machine learning models used. While all tools offer user interfaces and threat detection, DeepInvisi differentiates itself with specialized fileless malware detection using a Transformer-LSTM model and robust memory/process analysis, aiming to close the gap in effectively identifying fileless malware where current tools show limitations.

Table 2 *Comparison Between Existing Tools and Proposed Tool*

Feature	TotalAV	Kaspersky	AQUA Security	DeepInvisi
Platform	Desktop & Mobile	Desktop, Mobile & Web	Container-focused	Desktop
User Interface and Control Panel	Yes	Yes	Yes	Yes
Real-time Monitoring and Data Collection	No	Yes	Yes	Yes
Data Collection Components	No	Yes	Yes	Yes
Fileless Malware Detection	Limited	Limited	Limited	Yes
Memory and Process Analysis	No	Yes	No	Yes
Machine Learning Model	Basic Heuristics (Not Defined)	Advanced Heuristics (Not Defined)	Yes (Not Defined)	Yes (Transformer-LSTM)
Operate Without Internet	Yes	Yes	No	Yes
Threat Detection and Response	Yes	Yes	Yes	Yes
Programming Language	Undefined	Undefined	Undefined	Python

3. Methodology

This section discusses the six phases of Agile methodology which are requirement gathering, designing, development, testing, deployment, and feedback & reviewing.

3.1 Agile Methodology

Agile Methodology[21] in Fig. 3 is an iterative and flexible approach to software development, breaking work into smaller phases called sprints to deliver value incrementally. Agile involves phases like Requirement Gathering, Design, Development, Testing, Deployment, and Feedback & Review, which repeat in every sprint to ensure continuous improvement. This approach suits the evolving field of fileless malware detection, allowing iterative updates based on the latest threat intelligence.

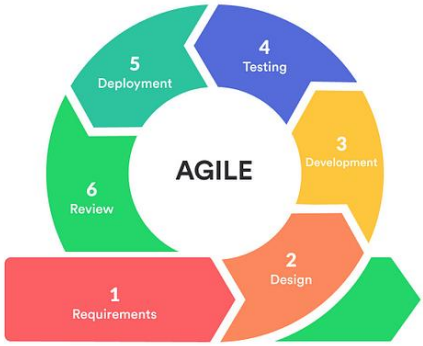

Fig. 3 *Agile Methodology*[21]

The development of DeepInvisi followed a structured lifecycle, starting with the Requirement Gathering Phase to define problems, objectives, and scope by researching fileless malware behaviors and existing tools, resulting in a clear problem statement and an Agile-based Gantt chart. Next, the Designing Phase produced a detailed architectural plan, user interface mockups, and system workflows using UML diagrams, integrating Transformer and LSTM models for advanced detection and culminating in a comprehensive Python-based development design.The subsequent Development Phase involved coding DeepInvisi in Python with TensorFlow, implementing modules for data collection, processing, and behavior detection, defining OS compatibility and prerequisites, and using PowerShell and memory dump datasets for training, ultimately yielding a functional prototype.

The Testing Phase validated the tool's performance through simulated fileless malware attacks, evaluating metrics like accuracy, precision, and recall, and using feedback to refine the tool and minimize false positives, leading to a reliable detection system. The Deployment Phase focused on releasing DeepInvisi for user testing with manuals, applying updates based on real-world feedback via a beta version, resulting in a ready-to-use tool. Finally, the Feedback & Reviewing Phase systematically collected user input through User Acceptance Testing (UAT) to assess usability and performance, guiding improvements to the user interface and detection algorithms and informing future enhancements.

4. Analysis and Design

This section explains the components of the analysis and design phase. This phase includes the user, functional and non-functional requirements, system architecture, deep learning process, and the Unified Modeling Language (UML)diagram[22].

4.1 DeepInvisi Fileless Malware Detection Tool Architecture

Fig. 4 illustrates the operation of the DeepInvisi fileless malware detection tool, which uses hybrid Transformer+ BiLSTM models for PowerShell script classification and memory analysis. The tool starts from the Home Page, where the "Start Monitoring" button activates real-time PowerShell monitoring. The system continuously monitors PowerShell activities via script block logging events and process creation monitoring. When suspicious PowerShell commands are detected, classify them using the trained model. When classified as malicious, the system automatically triggers memory analysis. If the memory analysis confirms malicious behavior, the tool automatically terminates the source script file and displays real-time notifications. The

detection events are logged for historical reporting, while background monitoring continues for ongoing protection against fileless malware threats.

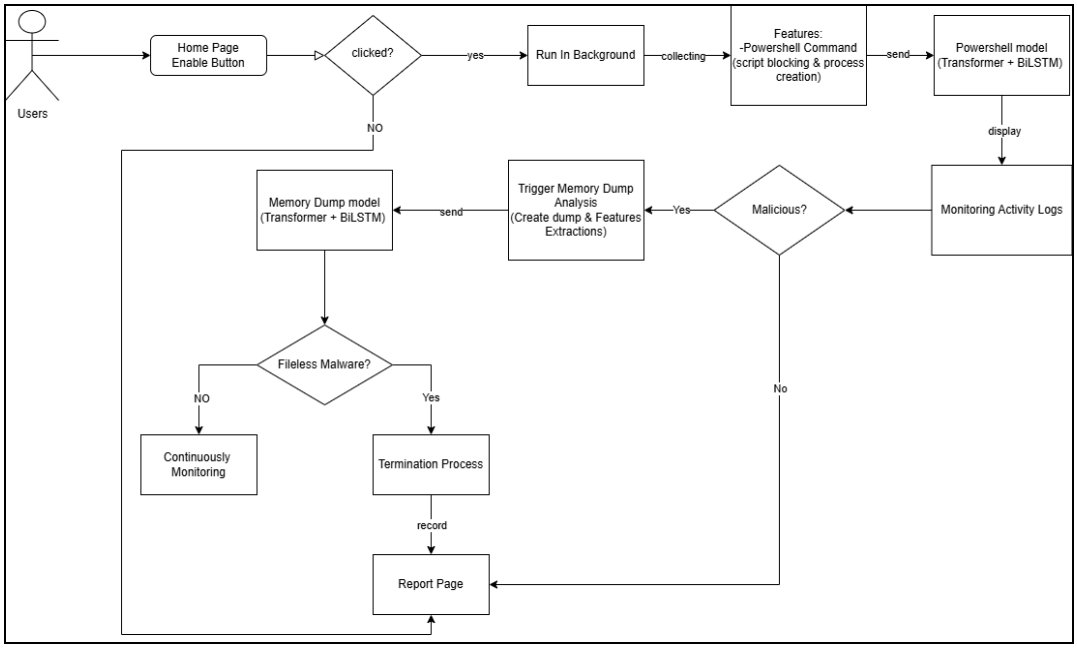

Fig. 4 *DeepInvisi Fileless Malware Detection Tool Architecture*

Fig. 5 illustrates the deep learning process for training the DeepInvisi Fileless Malware Detection Tool. The DeepInvisi starts with collecting raw data like PowerShell commands, and memory dumps. The PowerShell commands dataset includes 4,286 benign .ps1 files and 3,582 malicious .ps1 files, making it suitable for detecting suspicious script activities. The memory dump dataset is also in CSV format, containing features like process lists, DLL counts, handles, callbacks, and service information, along with labels for classification. The data is split into 70% for training, 15% for validation, and 15% for testing. Feature extraction converts raw data into usable input for the Transformer and BiLSTM model. During training, the model learns patterns from the training data, while the validation data helps fine-tune parameters and avoid overfitting. Testing evaluates the model on unseen data to ensure accurate detection in real-world scenarios. If performance is unsatisfactory, adjustments are made, and the process repeats until the model achieves reliable results.

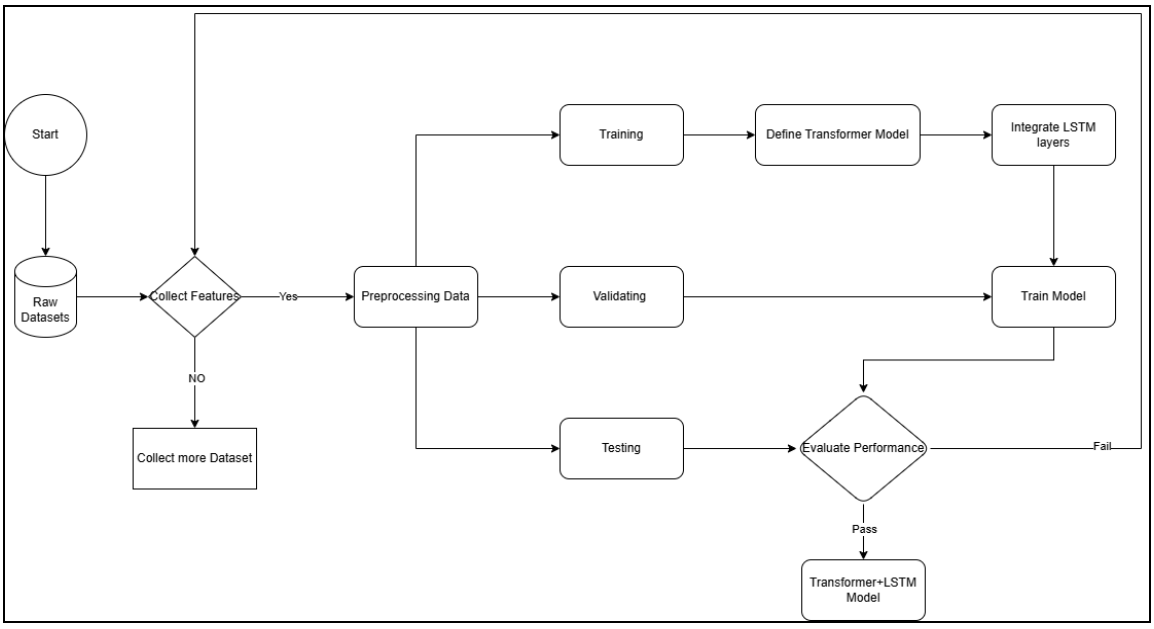

Fig. 5 *Model training process for DeepInvisi Fileless Malware Detection Tool*

4.2 User Requirements

To ensure the DeepInvisi effective and user-friendly, the tool is aimed at general publics who need a reliable and simple way to detect fileless malware. By continuously monitoring critical system features, the tool should provide users with accurate information about any malicious activity. Table 3 shows the user requirements including real-time monitoring, detailed threat alerts, comprehensive reports, compatibility, and seamless background operation for the DeepInvisi Fileless Malware Detection Tool.

Table 3 *User Requirements for DeepInvisi Fileless Malware Detection Tool*

User Requirement	Description
Real-Time Monitoring	The tool should automatically monitor the system in real-time without needing the user to configure anything after clicking a start button only.
Detailed Threat Alerts	Users will receive clear and detailed alerts whenever suspicious activity related to fileless malware is detected.
Comprehensive Summary	The tool should provide easy-to-understand summary that include detailed information about threats and their behaviors.
Compatibility	The tool must work smoothly on various operating systems and configurations, including virtual machines.
Seamless Background Operation	The tool should run in the background without interrupting users' regular work or affecting system performance.

4.2.1 Functional Requirements

DeepInvisi includes specific functionalities that allow it to detect and respond to fileless malware effectively. These features ensure that it can identify threats based on the behavior of the malware and provide users with the necessary information to act. Unlike traditional tools, DeepInvisi constantly analyzes system features to detect malicious patterns. Table 4 shows the functional requirements including continuous features monitoring, fileless malware detection, deep learning engine, detailed threat reports, threat response module, and real-time alerts for DeepInvisi Fileless Malware Detection Tool.

Table 4 *Functional Requirements for DeepInvisi Fileless Malware Detection Tool*

Functional Requirement	Description
Continuous Feature Monitoring	The tool should automatically monitor system features like PowerShell commands, and memory dump.
Fileless Malware Detection	The tool must use advanced techniques to identify fileless malware based on its behaviour instead of relying on file signatures.
Deep Learning Engine	The tool should use Transformer + LSTM models to analyse data and improve detection accuracy.
Detailed Threat Summary	The tool will generate detailed summary that include the source of the threat, the malicious behaviour observed, and recommended actions.
Threat Response Module	When a threat is detected, the tool should be able to stop the malicious activity or alert users with actionable steps.
Real-Time Alerts	The tool must immediately notify users whenever a potential fileless malware attack is detected, ensuring a quick response.

4.2.2 Non-Functional Requirements

Non-functional requirements as in Table 5 define the quality, performance, and overall user experience of the DeepInvisi on the aspect of fast and efficient, easy to use, reliable performance, security, scalability and compatibility. These requirements ensure the tool is reliable, efficient, and secure for everyday use.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 978-999

Table 5 *Non-Functional Requirements for DeepInvisi Fileless Malware Detection Tool*

Non-Functional Requirement	Description
Fast and Efficient	The tool should work quickly and efficiently, ensuring that malicious activity is detected and reported without delay.
Easy to Use	The tool should have a simple design so that users can easily understand how it works and interact with it if needed.
Reliable Performance	The tool must perform consistently without errors, ensuring that it accurately detects fileless malware in all cases.
Security	The tool should include robust security measures to protect user data and ensure that malicious actors cannot interfere with its operation.
Scalable	The tool should handle increasing amounts of data and usage without slowing down or failing.
Compatibility	The tool must work on different types of systems, including both physical and virtual machines, without compatibility issues.

4.3 Use Case

Fig. 6 shows how users interact with the DeepInvisi, which offers four main features. The Enable to Run in Background feature activates real-time monitoring, which allows the tool to silently scan for threats like malicious PowerShell commands. Users can Disable to Run in Background option to save system resources. The Live Monitoring feature provides real-time analysis and alerts for immediate action. The Memory Analysis provides the full detailed information of the memory dump. Finally, the Reports feature notifies users of detected threats and includes details on system behavior and corrective actions.

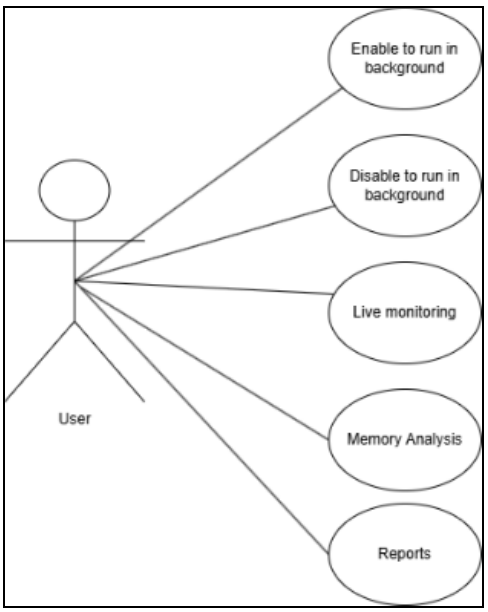

Fig. 6 *Use Case Diagram of DeepInvisi Fileless Malware Detection Tool*

Fig. 7 illustrates the steps involved in the DeepInvisi tool which provides real-time protection and user notifications for threats. The process starts at the Home Screen, where users can enable or disable the "Run in Background" feature. If enabled, the tool operates silently, continuously monitoring system activity. Then, the Live Monitoring analyzes features like PowerShell commands, and memory dumps, sending data to the detection model. If fileless malware is detected, the tool terminates the threat and notifies the user. If no malware is found, monitoring continues to ensure system security.

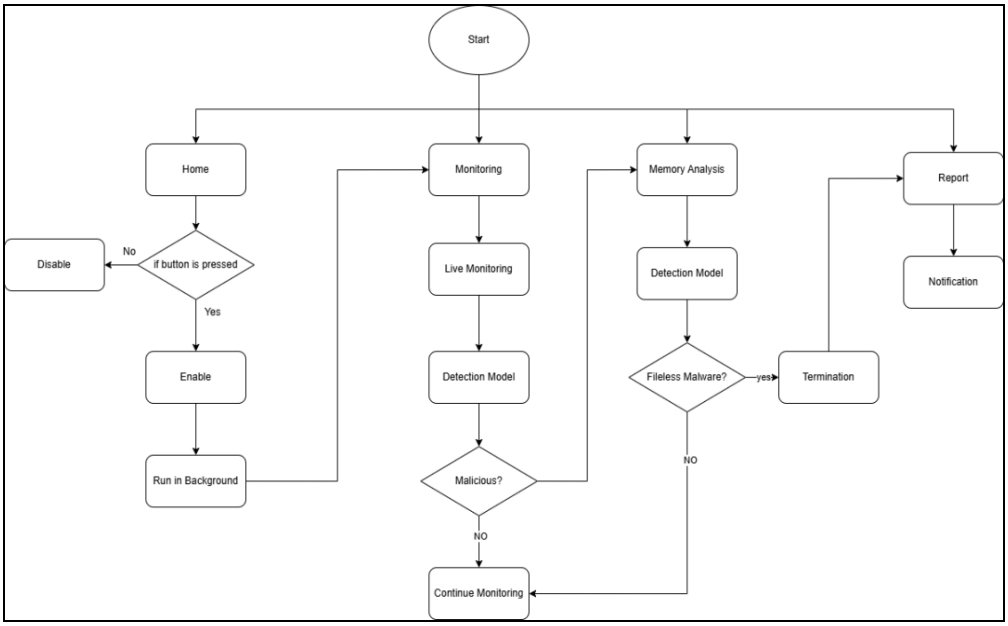

Fig. 7 *User Activity Diagram of DeepInvisi Fileless Malware Detection Tool*

Fig. 8 illustrates the DeepInvisi tool's workflow using a UML Sequence Diagram. The process begins on the Home Page, where the user enables the "Run in Background" feature to activate silent monitoring. The tool then enters Live Monitoring mode, analyzing activities with its Detection Model to identify potential threats. If a threat is detected, it triggers the Termination process, saving the data and notifying the user. The user can review detailed monitoring results in the Report section, which summarizes key activities and findings. DeepInvisi ensures seamless monitoring, threat detection, and reporting for efficient system protection.

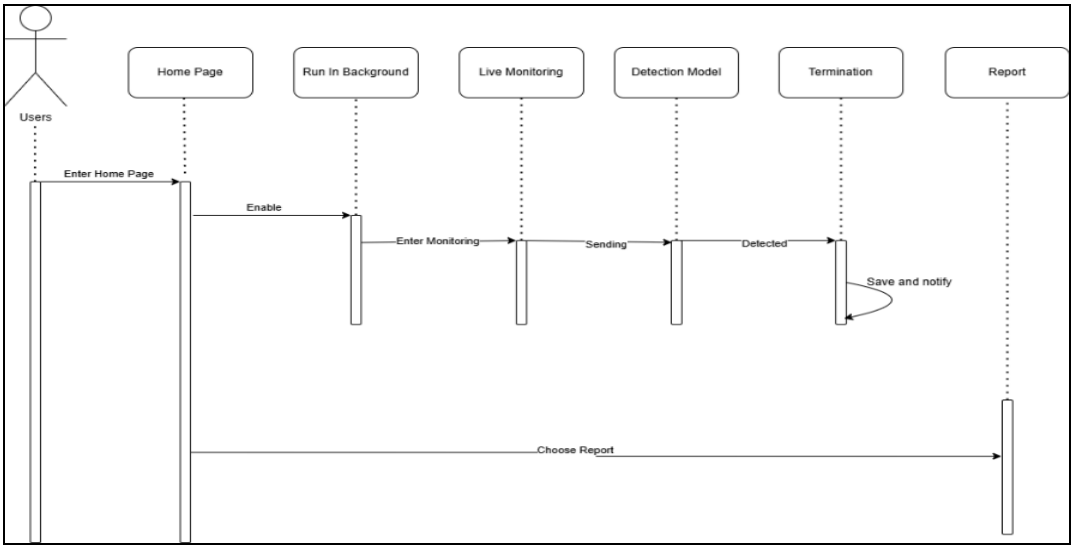

Fig. 8 *User Sequence Diagram of DeepInvisi Fileless Malware Detection Tool*

4.4 User Interface

The Home Page of the DeepInvisi is shown in Fig. 9. In the center of the tool, there is a large yellow button with a play icon labeled "ENABLE". This button starts the tool's monitoring to detect fileless malware by collecting system behavior data. On the left side, there are three options: Home, Monitoring, and Report, with an icon for easy recognition. The Home option highlights the current page. The monitoring takes users to a dashboard for live data, and the Report shows detailed findings about detected threats. The "ENABLE" button turns into a "DISABLE" with a pause icon being clicked, allowing users to stop the monitoring process anytime as shown in Fig. 10.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 978-999

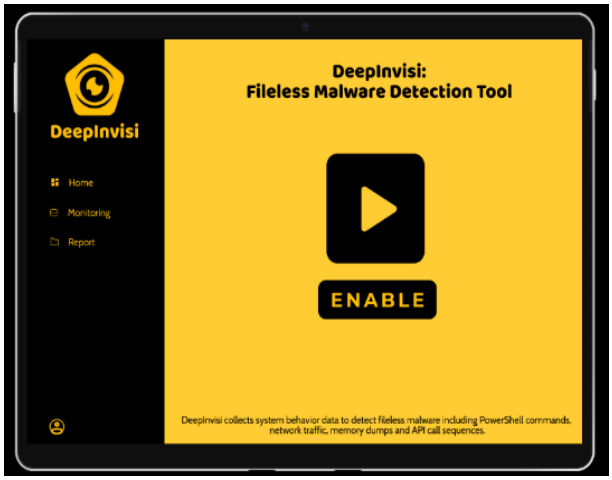

Fig. 9 *Home Page of DeepInvisi with Enable Button*

Fig. 10 *Home Page of DeepInvisi with Disable Button*

The Live Monitoring page of the DeepInvisi Fileless Malware Detection Tool in Fig. 11 shows real-time activity as the tool captures features of fileless malware. The gray box in the center displays detailed logs, such as captured PowerShell commands, and memory dump activities, giving users insight into the detection process. The Report page in Fig. 12 shows notifications about detected fileless malware along with detailed information. The page provides a clear overview of all threats found during the monitoring process. This helps users understand the nature of the threats. The design is simple, making it easy for users to review and analyze the reports.

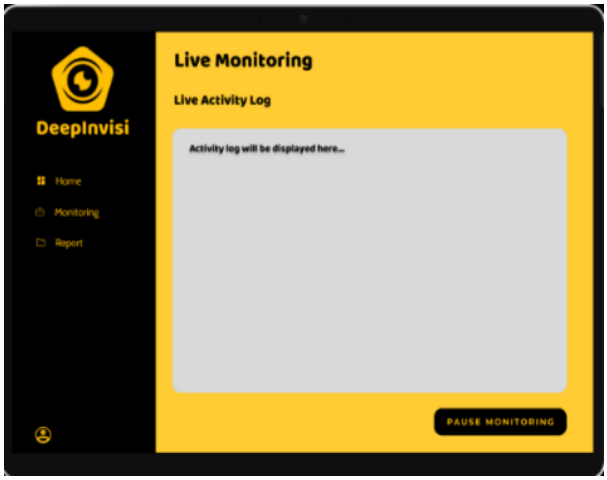

Fig. 11 *Live Monitoring Page of DeepInvisi*

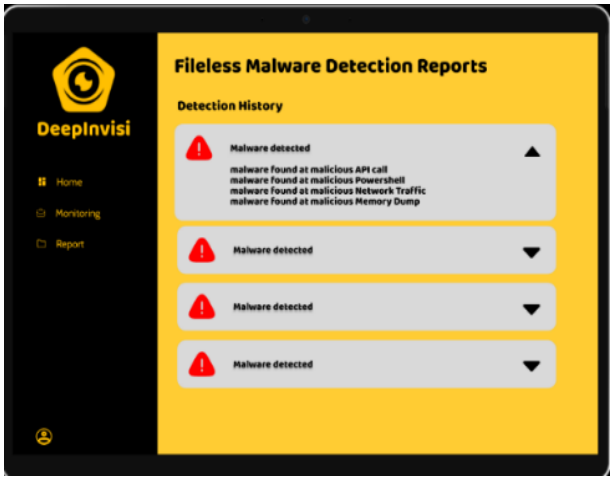

Fig. 12 *Report Page of DeepInvisi*

4.5 Test Plan

The Test Plan ensures the DeepInvisi Fileless Malware Detection Tool works effectively and reliably. It verifies the tool's ability to monitor system features, detect fileless malware, and alert users about threats in real-time while generating accurate reports. The plan covers functional and non-functional tests, including monitoring, threat detection, report generation, and compatibility across environments. Testing during the implementation phase will identify and resolve any issues. Table 6 provides a detailed outline of the test plan.

Table 6 *Test Plan for DeepInvisi Fileless Malware Detection Tool*

No.	Test List	Description	Actual Result
1	Real-Time Monitoring Test	Verify the tool's ability to monitor system features like PowerShell commands and analysis memory dump continuously.	Pass/ Fail
2	Malware Detection Test	Evaluate the accuracy of detecting fileless malware based on behavior patterns.	Pass/ Fail
3	Threat Alert Test	Check if the tool sends immediate and clear alerts when suspicious activity is detected.	Pass/ Fail
4	Report Generation Test	Assess the accuracy and detail level of the reports generated about detected threats.	Pass/ Fail
5	Performance Test	Measure the tool's speed and efficiency under different workloads.	Pass/ Fail
6	Compatibility Test	Ensure the tool works seamlessly across different operating systems and environments.	Pass/ Fail

5. Result And Discussion

This section discusses the DeepInvisi for detecting advanced fileless malware which provide DeepInvis's architecture, interface, core features, and how it analyzes PowerShell scripts and memory dumps. The section also covers the deep learning models used, including the training and performance, and evaluates DeepInvisi's overall effectiveness in real-time threat detection and automated response.

5.1 System Implementation and Core Components

The DeepInvisi has been implemented as a comprehensive solution integrating real-time PowerShell monitoring, sophisticated memory analysis, and deep learning-based threat detection. It comprises a Python-based backend service and an Electron-based frontend interface, designed to deliver detection capabilities alongside a user-friendly experience.

5.1.1 DeepInvisi Main Application Window

The DeepInvisi tool's main window, developed using the Electron framework, provides a desktop interface with flexibility with web technology. The frontend, built on Electron, interfaces with a Python-based WebSocket server that forms the backend. Real-time WebSocket messaging facilitates immediate communication of threat alerts and status updates between these layers. Key operational features include automatic startup and management of the Python backend, ensuring seamless integration. Dynamic indicators provide users with live status updates on backend connectivity, PowerShell monitoring, and model loading. Designed with security as a priority, the application operates with necessary administrator privileges within defined security contexts. A centralized control panel allows users to manage monitoring activities, offering clear visual feedback on the system's current state.

5.1.2 DeepInvisi Navigation System

A consistent header-based navigation bar in Fig. 13 provides seamless access to all application modules, maintaining user context across different views. The application's navigation is designed for ease of use and clarity, featuring visual highlighting to indicate the currently active page, ensuring users always know their location within the system.

Fig.13 *Header of DeepInvisi Fileless Malware Detection Tool*

5.1.3 DeepInvisi Home Page Interface

The home page in Fig. 14 serves as DeepInvisi's central command center, providing users with a system overview, quick access to actions, and real-time status monitoring. A key feature is the control panel's Start/Stop Monitoring button, which allows users to toggle PowerShell monitoring; this button dynamically updates its text (e.g., "Start Monitoring" to "Stop Monitoring") and visual state (e.g., orange for inactive, red for active), and is intelligently disabled if the backend is disconnected or models fail to load.

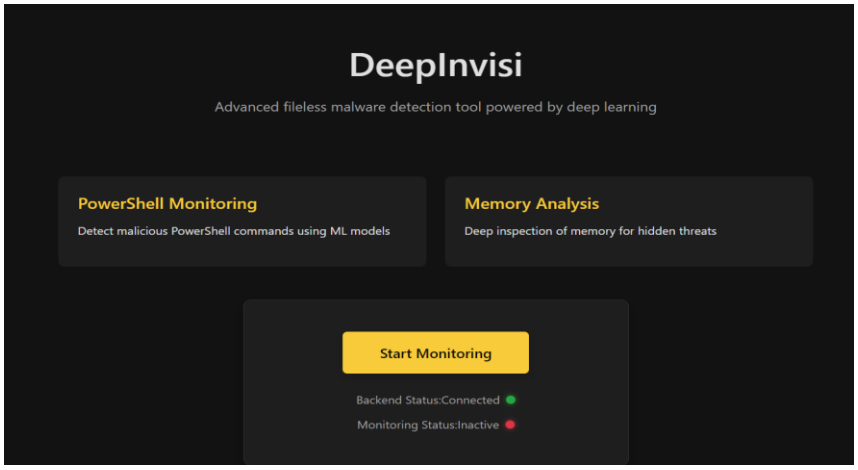

Fig.14 *Home Page of DeepInvisi Fileless Malware Detection Tool*

5.1.4 DeepInvisi Monitoring Page

The PowerShell Monitoring page in Fig. 15 is dedicated to real-time threat detection visualization and provides detailed analysis of PowerShell activity. Live PowerShell detection events are streamed via WebSockets with minimal delay and displayed in a comprehensive table, each clearly classified to indicate its threat level. Key advanced monitoring features include an automatic threat response capability, where the backend can terminate scripts identified as high-confidence malicious threats; this action also automatically triggers a subsequent memory analysis for deeper investigation. Furthermore, users can interact with the displayed events through click-to-expand functionality, allowing for in-depth analysis of individual detections.

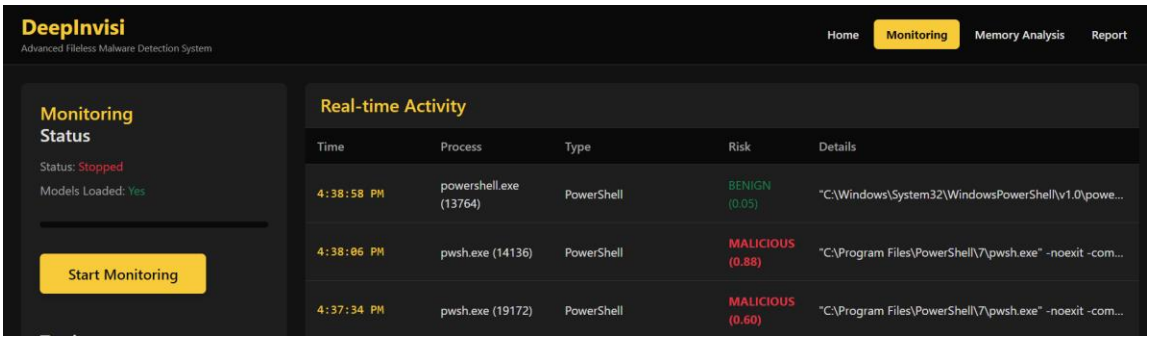

Fig.15 *Monitoring Page of DeepInvisi Fileless Malware Detection Tool*

5.1.5 DeepInvisi Memory Analysis Page

The Memory Analysis page in Fig. 16 offers advanced capabilities for in-depth inspection of process memory. Its architecture relies on a Python backend where it handles the critical tasks of memory capture, feature extraction, DL-based classification, and the identification of suspicious patterns. The frontend interface provides users with essential components, including controls for process selection, refreshing the process list, and initiating analysis. Upon completion, the memory analysis results are presented in detail shown in this page.

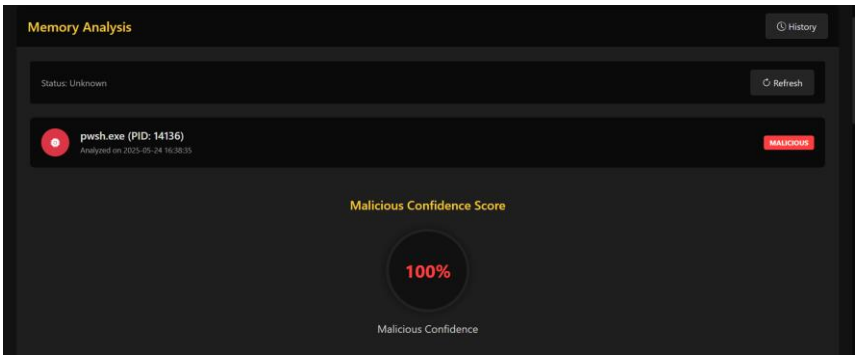

Fig.16 *Memory Analysis Page of DeepInvisi Fileless Malware Detection Tool*

5.1.6 DeepInvisi Report Page

The Report page in Fig. 17 offers comprehensive analysis of historical detection data, equipped with filtering, visualization options. The historical data is retrieved from the backend; it dynamically updates summaries, tables, and charts. An advanced filtering system empowers users to refine reports by specific date ranges and detection verdicts. Key statistics, such as total detections and counts for malicious, suspicious, and benign events, are prominently displayed on a detection summary dashboard. Furthermore, the page provides access to detailed event logs for thorough documentation and review.

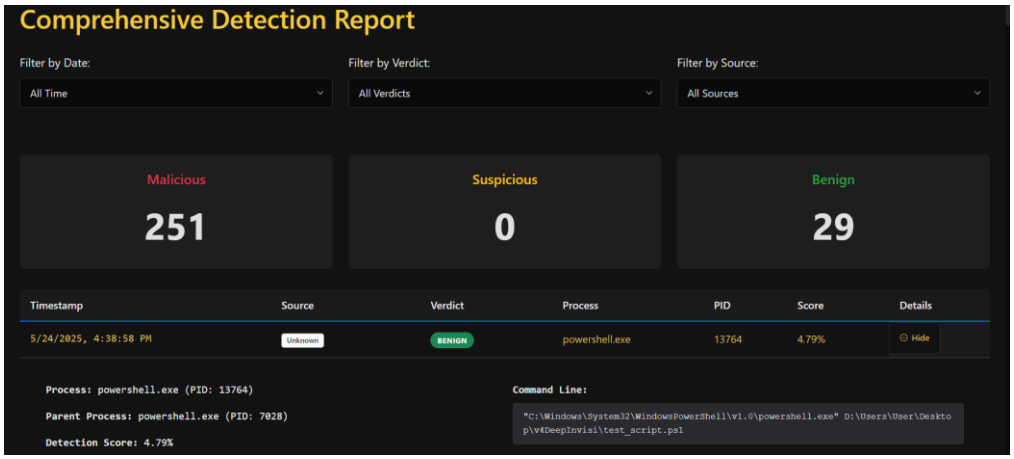

Fig.17 *Report Page of DeepInvisi Fileless Malware Detection Tool*

5.1.7 DeepInvisi Pop-up Notification Module

The Notification System module in Fig. 18 is designed to provide immediate alerts for critical security events, ensuring prompt analyst attention. Its architecture features a multi-modal notification system, which employs native desktop notifications, in-app modals, and audio alerts to signal critical threats. A key component is the dynamic modal system, presenting interactive modals that display detailed information about the detected event, including the verdict, process information, confidence score, and identified suspicious patterns.

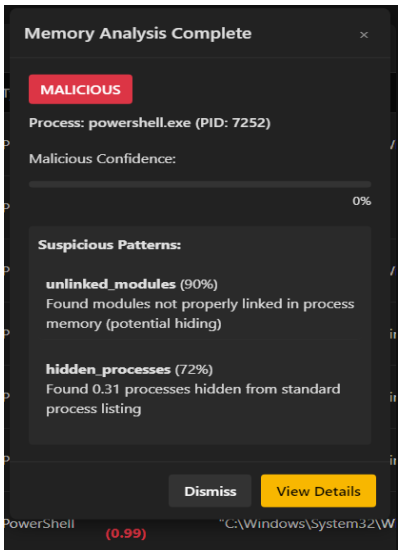

Fig.18 *Pop-up notification of DeepInvisi Fileless Malware Detection Tool*

5.2 Real-time Monitoring and Analysis Capabilities

DeepInvisi monitors your system in real-time using multiple techniques, especially looking for PowerShell threats and suspicious signs in memory dump. It gathers data, runs it through an analysis process, and uses deep learning trained model to decide if something is a fileless malware threat.

5.2.1 Powershell Monitoring Architecture

The core of PowerShell monitoring resides in *backend/powershell_monitor.py*, which contains several specialized monitors managed by the *CombinedPowerShellMonitor* class in Fig. 19. The *CombinedPowerShellMonitor* is central to this architecture. Its initialization demonstrates a robust approach to managing individual monitoring components. The *CombinedPowerShellMonitor* employs a dual-pronged approach for detecting PowerShell activity: the *WmiPowerShellMonitor* uses WMI to track powershell.exe and pwsh.exe process creation, analyzing their command lines for script execution indicators (like -file, .ps1, -encodedcommand) and extracting .ps1 file paths; concurrently, the *PowerShellScriptBlockMonitor* monitors Windows Event ID 4104 from the "Microsoft-Windows-PowerShell/Operational" event log to capture detailed script block content and associated Process IDs. A watchdog thread ensures these individual monitors are resiliently restarted if they become inactive.

```python
class CombinedPowerShellMonitor(PowerShellMonitor):
    """Combines multiple PowerShell monitoring strategies"""

    def __init__(self, callback=None):
        """Initialize the combined monitor"""
        super().__init__(callback)
        self.monitors = []
        self.active_monitors = {}

        # Create individual monitors with robust error handling
        try:
            self.wmi_monitor = WmiPowerShellMonitor(callback=self._relay_event)
            self.monitors.append(self.wmi_monitor)
            self.active_monitors['wmi'] = False
        except Exception as e:
            logger.error(f"Failed to initialize WMI monitor: {e}", exc_info=True)
            self.wmi_monitor = None

        try:
            self.scriptblock_monitor = PowerShellScriptBlockMonitor(callback=self._relay_event)
            self.monitors.append(self.scriptblock_monitor)
            self.active_monitors['scriptblock'] = False
        except Exception as e:
            logger.error(f"Failed to initialize script block monitor: {e}", exc_info=True)
            self.scriptblock_monitor = None
```

Fig.19 *PowerShell Real-time Monitoring*

5.2.2 Automatic Powershell Threat Response Implementation

DeepInvisi's backend automatically responds to high-risk PowerShell threats. When a PowerShell activity is classified as "MALICIOUS" and its risk score is sufficiently high, the system automatically triggers an in-depth memory analysis of the suspicious process. If this memory analysis further confirms the process is malicious, DeepInvisi will then automatically attempt to terminate it. Automatic memory analysis in Fig. 20 is initiated under similar conditions which are an enabled configuration flag, a "MALICIOUS" verdict, and the ml_score meeting a specific *mem_analysis_threshold_score*. When triggered, it calls *self.analyze_memory_command* as an asyncio task, which in turn uses *self.analyze_process_memory* to perform the in-depth memory scan of the suspicious process.

```python
# If the PowerShell command is detected as malicious and auto-analyze is enabled,
 # trigger memory analysis for the process
if (broadcast_data['verdict'] == 'MALICIOUS' and
    self.auto_analyze_memory and
    broadcast_data['process_id'] > 0):

    logger.info(f"PowerShell command detected as malicious. Triggering memory analysis for PID {broadcast_data['process_id']}")

    # Trigger memory analysis in a non-blocking way
    await self.analyze_process_memory(
        broadcast_data['process_id'],
        broadcast_data['process_name']
    )
```

Fig.20 *Trigger Memory Dump Analysis*

For process termination in Fig. 21, implement cleanup of specific known test script files, DeepInvisi employs the *_terminate_all_test_scripts* asynchronous method within backend/main.py. This function iterates through a predefined list of absolute file paths. For each specified path, it dynamically constructs a PowerShell command designed to check for the file's existence using Test-Path and then forcibly remove it with Remove-Item -Path '{path}' -Force -ErrorAction Continue.

```
# Use PowerShell's Test-Path and Remove-Item which works reliably
        for path in test_paths:
            try:
                # Construct a PowerShell command that works reliably
                ps_command = f"""
                if (Test-Path '{path}') {{
                    Write-Host 'Found file at: {path}';
                    Remove-Item -Path '{path}' -Force -ErrorAction Continue;
                    if (Test-Path '{path}') {{
                        Write-Host 'Removal failed';
                    }} else {{
                        Write-Host 'Successfully removed';
                    }}
                }} else {{
                    Write-Host 'File not found at: {path}';
                }}"""
```

Fig.21 *Termination or Remove the ps1 file*

5.2.3 Memory Dump Capture Process Details

The *MemoryAnalyzer* class, specifically its *_create_memory_dump* method in Fig. 22, orchestrates memory dump capture using procdump.exe. The system first verifies procdump's availability, checking a configured path and then the system PATH. Dump creation initially attempts to use a run_procdump.bat wrapper script if found. All procdump executions are subject to a timeout. Dump files, named using a pattern like "{safe_process_name}_pid{pid}_{timestamp}.dmp", are saved in a configurable directory (self.dump_dir), and _cleanup_dump handles their removal post-analysis or on error.

```
def _create_memory_dump(self, pid, process_name=None):

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        file_name = f"{process_name or pid}_{pid}_{timestamp}.dmp"
        dump_path = os.path.join(self.dump_dir, file_name)

        logger.info(f"Creating memory dump for PID {pid} at {dump_path}")
```

Fig.22 *Create Memory Dump using prodump.exe*

5.2.4 Memory Feature Extraction with Volatility3

The *MemoryAnalyzer._extract_memory_features* method leverages Volatility3 to extract features from memory dumps. It iterates through a predefined list of Volatility3 plugins. A feature vector of 49 distinct memory characteristics, defined by *self.MEMORY_FEATURE_ORDER* in Fig. 23 and mapped using *self.FEATURE_MAP*, is initialized with all values set to 0.0. As plugin outputs are processed, corresponding features in this vector are updated. The method ensures the final output is an ordered dictionary containing all 49 features, ready for further analysis or model input.

```
# Updated to match the expected 49 features from MD_model.ipynb
MEMORY_FEATURE_ORDER = [
    # Process list features
    'pslist_nproc', 'pslist_nppid', 'pslist_avg_threads', 'pslist_avg_handlers',
    # DLL list features
    'dlllist_ndlls', 'dlllist_avg_dlls_per_proc',
    # Handle features
    'handles_nhandles', 'handles_avg_handles_per_proc', 'handles_nfile',
    'handles_nevent', 'handles_ndesktop', 'handles_nkey', 'handles_nthread',
    'handles_ndirectory', 'handles_nsemaphore', 'handles_ntimer', 'handles_nsection',
    'handles_nmutant',
    # Ldrmodules features
    'ldrmodules_not_in_load', 'ldrmodules_not_in_init', 'ldrmodules_not_in_mem',
    'ldrmodules_not_in_load_avg', 'ldrmodules_not_in_init_avg', 'ldrmodules_not_in_mem_avg',
    # Malfind features
    'malfind_ninjections', 'malfind_commitCharge', 'malfind_protection', 'malfind_uniqueInjections',
    # Process view features
    'psxview_not_in_pslist', 'psxview_not_in_eprocess_pool', 'psxview_not_in_ethread_pool',
    'psxview_not_in_pspcid_list', 'psxview_not_in_csrss_handles', 'psxview_not_in_session',
    'psxview_not_in_deskthrd', 'psxview_not_in_eprocess_pool_false_avg', 'psxview_not_in_ethread_pool_false_avg',
    'psxview_not_in_pspcid_list_false_avg', 'psxview_not_in_csrss_handles_false_avg',
    'psxview_not_in_session_false_avg', 'psxview_not_in_deskthrd_false_avg',
    # Module features
    'modules_nmodules',
    # Service features
    'svcscan_nservices', 'svcscan_kernel_drivers', 'svcscan_fs_drivers',
    'svcscan_process_services', 'svcscan_shared_process_services', 'svcscan_nactive',
    # Callback features
    'callbacks_ncallbacks'
]
```

Fig.23 *49 Features of Extraction for Memory Analysis*

5.3 Deep Learning Model Implementation and Performance

DeepInvisi leverages two sophisticated deep learning models for its core detection capabilities, a Transformer-BiLSTM model for PowerShell script analysis and memory dump analysis with feature classification. These models are trained offline using detailed Jupyter notebooks and then integrated into the real-time backend.

5.3.1 Powershell Detection Model

The PowerShell detection model utilizes hybrid neural network architecture, combining a custom TransformerEncoder layer with a multi-input structure to process both tokenized script sequences and pre-extracted textual features. The TransformerEncoder, featuring multi-head self-attention, processes embedded script tokens, which then pass through a Bidirectional LSTM layer shown in Fig. 24.

```python
# Define the Multi-Input Transformer + BiLSTM model
def create_multi_input_model(max_seq_len, vocab_sz, emb_dim, num_transformer_heads, transformer_ff_dim, lstm_sz, num_text_feats, dropout_rt):
    # Input and Embedding for Sequential Data
    sequence_input = Input(shape=(max_seq_len,), name='sequence_input')
    embedding_layer = Embedding(input_dim=vocab_sz, output_dim=emb_dim, input_length=max_seq_len)(sequence_input)

    # Transformer Encoder
    # The TransformerEncoder expects 3D input [batch, sequence, features] which Embedding provides
    transformer_output = TransformerEncoder(emb_dim, num_transformer_heads, transformer_ff_dim, rate=dropout_rt)(embedding_layer)

    # BiLSTM Layer
    # The BiLSTM also expects 3D input
    bilstm_output = Bidirectional(LSTM(lstm_sz, return_sequences=False))(transformer_output) # return_sequences=False as we take the final state
    bilstm_output = Dropout(dropout_rt)(bilstm_output)

```

Fig.24 *Transformer + BiLSTM Layers for Powershell Model*

The PowerShell detection model is trained using a dataset split into 70% training, 15% validation, and 15% test sets in Fig. 25, with stratification to maintain label distribution. The training process employs the *tf.keras.optimizers.Adam* optimizer with a learning rate of 1e-4 and binary_crossentropy as the loss function, while tracking accuracy, precision, and recall. The model is trained for up to 50 epochs with a batch size of 32.

```
Sequence Train shape: (5317, 256), Textual Train shape: (5317, 7), Labels Train shape: (5317,)
Sequence Val shape: (1139, 256), Textual Val shape: (1139, 7), Labels Val shape: (1139,)
Sequence Test shape: (1140, 256), Textual Test shape: (1140, 7), Labels Test shape: (1140,)
```

Fig.25 *Training, Validation, Testing Data for PowerShell Model*

For the Post-training, the best model is evaluated on the test set for loss, accuracy, precision, recall, and F1-score. The trained PS_model acheived test loss: 0.4051, test accuracy: 83.51%, test precision: 83.95%, test recall: 77.25%, and test F1-score: 80.46% as depicted in Fig. 26.

```
Evaluating on Test Data:
Test Loss: 0.4051
Test Accuracy: 83.51%
Test Precision: 83.95%
Test Recall: 77.25%
Test F1-Score: 80.46%
```

Fig.26 *Result of Trained PowerShell Model*

The PowerShell model, with an optimal threshold of 0.6, achieved an accuracy of 84.47%, precision of 86.82%, recall of 76.25%, and an F1-Score of 81.19% on a test set of 1140 samples as illustrated in Fig. 27. This means that it correctly identified 382 malicious scripts (TP) and 581 benign ones (TN), while misclassifying 58 benign scripts as malicious (FP) and missing 119 malicious scripts (FN), indicating a reasonable balance for initial screening.

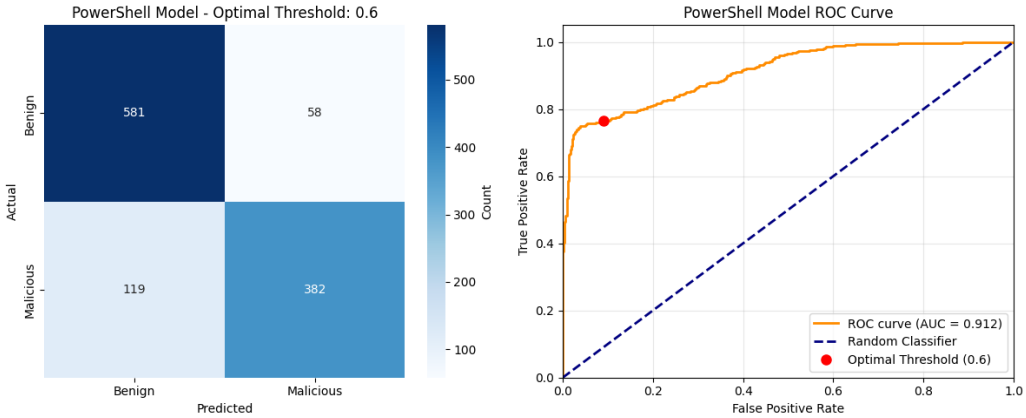

Fig.27 *Confusion Matric and ROC Curve for PowerShell Model*

5.3.2 Memory Detection Model

The memory detection model processes tabular feature data in Fig. 28, derived from Volatility3 outputs as captured in MalwareMemoryDump.csv, employing Transformer-based architecture. After preprocessing, which included standard scaling for all numerical data due to the absence of categorical features, the input consisted of 49 features. This output is then reshaped to simulate a sequence of length one, enabling processing by a custom TransformerEncoder layer, which features multi-head self-attention. The Transformer's output is subsequently fed into two stacked Bidirectional LSTM layers before a final classifier head with a sigmoid activation determines the binary output in Fig. 29.

Raw_Type	pslist_nproc	pslist_nppid	pslist_avg_th	pslist_nprocs	pslist_avg_han	dllist_ndlls	dllist_avg	handles_n	handles_a	handles_n	handles_n	handles_n	handles_n	handles_n	handles_n	handles_n	handles_n	ldrmodule	ldrmodule	ldrmodule	ldrmodule
Benign	45	17	10.5555556	0	202.8444444	1694	38.5	9129	212.302	0	670	3161	46	716	887	104	671	125	184	257	53	95	53	0.03037
Benign	47	19	11.5319149	0	242.2340426	2074	44.1277	11385	242.234	0	840	3761	51	1011	1030	117	766	148	337	394	77	123	77	0.03617
Benign	40	14	14.725	0	288.225	1932	48.3	11529	288.225	0	1050	3996	45	784	1241	100	645	138	369	338	51	89	51	0.02611
Benign	32	13	13.5	0	264.28125	1445	45.1563	8457	264.281	0	630	2961	36	654	792	83	567	127	186	242	31	62	31	0.02148
Benign	42	16	11.452381	0	281.3333333	2067	49.2143	11816	281.333	0	908	3834	45	1252	942	103	825	135	375	429	102	143	102	0.04782
Benign	40	12	13.8	0	306.95	2082	52.05	12278	306.95	0	1080	4308	44	887	1199	101	768	140	381	382	76	113	76	0.03657
Benign	43	13	13.8139535	0	305.0232558	2169	50.4419	13116	305.023	0	1150	4727	48	923	1314	107	784	145	438	399	70	111	70	0.03283

Fig.28 *Dataset of MalwareMemoryDump.csv*

```
# 3. Transformer Encoder block
    # For sequence length 1, attention might not be as impactful, but we follow the structure.
    transformer_block = TransformerEncoder(embed_dim, num_heads, ff_dim, rate=dropout_rate)
    x = transformer_block(x)
    # You can stack more transformer_blocks if needed:
    # x = transformer_block(x)

    # 4. BiLSTM layers
    # Input to BiLSTM: (batch_size, 1, embed_dim)
    x = layers.Bidirectional(layers.LSTM(lstm_units, return_sequences=True))(x) # Output: (batch_size, 1, 2*lstm_units)
    x = layers.Dropout(dropout_rate)(x)
    x = layers.Bidirectional(layers.LSTM(lstm_units, return_sequences=False))(x) # Output: (batch_size, 2*lstm_units)
    x = layers.Dropout(dropout_rate)(x)
```

Fig.29 *Transformer + BiLSTM Layers for Memory Model*

The memory detection model is trained using the processed dataset, split into 70% training (41,017 samples), 15% validation (8,789 samples), and 15% test sets (8,790 samples) as in Fig. 30. The training process utilizes the tf.keras.optimizers.Adam optimizer with a learning rate of 1e-4 and binary_crossentropy as the loss function, while tracking accuracy, precision, and recall. Class weights are calculated and applied to handle potential data imbalance. The model is trained for up to 50 epochs with a batch size of 64.

```
X_train shape: (41017, 49), y_train shape: (41017,)
X_val shape: (8789, 49), y_val shape: (8789,)
X_test shape: (8790, 49), y_test shape: (8790,)
```

Fig.30 *Training, Validation, Testing Data for Memory Model*

For the post-training, the best model is evaluated on the test set for loss, accuracy, precision, recall, and F1-score. The trained MD_model achieved the result of, Test Loss: 0.0006, Test Accuracy: 99.98%, Test Precision: 99.95%, Test Recall: 100.00%, Test F1-Score: 99.98% as stated in Fig. 31.

```
Evaluating on Test Data:
Test Loss: 0.0006
Test Accuracy: 99.98%
Test Precision: 99.95%
Test Recall: 100.00%
Test F1-Score: 99.98%
```

Fig.31 *Result of Trained Memory Model*

The Memory model, evaluated with its optimal threshold of 0.7 on a larger test set of 8790 samples, demonstrated exceptionally high performance. It achieved 99.99% accuracy, 99.98% precision, 100% recall (sensitivity), and 99.98% specificity, resulting in an F1-Score of 0.9999. With these metrics, it correctly identified all 4395 malicious memory dumps (TP) and 4394 benign ones (TN), with only 1 false positive (FP) and 0 false negatives (FN), signifying its strong reliability for in-depth threat confirmation as in Fig. 32.

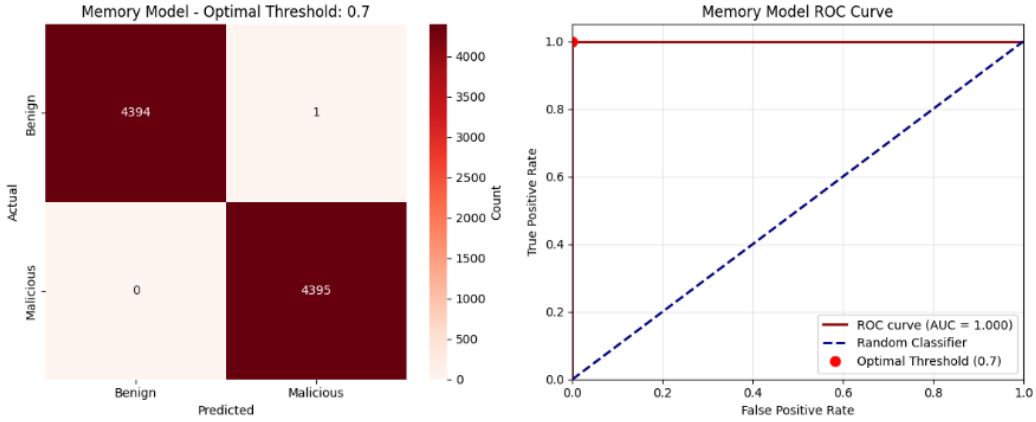

Fig.32 *Confusion Matric and ROC Curve for Memory Model*

5.4 Testing and Verification

This section explains how the DeepInvisi Fileless Malware Detection Tool is tested to validate its core functionalities, overall performance, compatibility across target environments, and the accuracy of its detection mechanisms.

5.4.1 Tool Testing

Table 7 presents the detailed test plan specifically designed for the DeepInvisi, executed following its development and integration of all modules. Tests were performed across target environments. The DeepInvisi tool successfully passed all planned tests, demonstrating the expected detection capabilities, response actions, and operational stability as per its design specifications.

Table 7 *Test Plan of DeepInvisi Fileless Malware Detection Tool*

No.	Test List	Description	Actual Result
1	Real-Time Monitoring Test	Verify the tool's ability to monitor system features like PowerShell commands and analysis memory dump continuously.	Pass
2	Malware Detection Test	Evaluate the accuracy of detecting fileless malware based on behavior patterns.	Pass
3	Threat Alert Test	Check if the tool sends immediate and clear alerts when suspicious activity is detected.	Pass
4	Report Generation Test	Assess the accuracy and detail level of the reports generated about detected threats.	Pass
5	Performance Test	Measure the tool's speed and efficiency under different workloads.	Pass
6	Compatibility Test	Ensure the tool works seamlessly across different operating systems and environments.	Pass

Table 8 and Table 9 present the malware detection performance of the DeepInvisi tool when tested against a set of malicious PowerShell scripts. For this evaluation, 10 distinct malicious PowerShell samples were sourced from reputable threat intelligence platforms: 5 samples from MalwareBazaar and 5 samples from Any.Run. The results demonstrate DeepInvisi's strong detection capabilities, as all 10 malicious PowerShell scripts, all samples were successfully identified as malicious. Furthermore, consistent with its automated response design, DeepInvisi automatically deleted these detected malicious .ps1 files, showcasing its effectiveness in neutralizing active PowerShell-based threats.

Table 8 *Testing Result on Fileless Malware samples from MalwareBazaar*

No	File Name	File Size (bytes)	File Type	First Seen	Result
1	payload_1748317361_2041.ps1	102076	ps1	2025-05-27 12:48:08 UTC	Detected
2	0.ps1	103768	ps1	2025-05-27 06:52:11 UTC	Detected
3	payload_1 (2).ps1	1153	ps1	2025-05-27 06:40:37 UTC	Detected
4	runner.ps1	365109	ps1	2025-05-29 12:53:49 UTC	Detected
5	df3beefdd998d9488ed81285c6 01b4206d2d286448af87fbe46e 5e262d812b0f.ps1	141996	ps1	2025-05-29 11:52:08 UTC	Detected

Table 9 *Testing Result on Fileless Malware Samples from Any.run*

No	File Name	File Size (bytes)	File Type	First Seen	Result
1	sample.ps1	147000	ps1	2025-05-23 01:34:13 UTC	Detected
2	file.ps1	6000	ps1	2025-05-25 03:57:11 UTC	Detected
3	4580_.txt.ps1	178000	ps1	2025-05-26 18:06:41 UTC	Detected
4	transform.ps1	6000	ps1	2025-05-29 01:25:15 UTC	Detected
5	2521_.htm.ps1	6000	ps1	2025-05-29 01:42:18 UTC	Detected

5.4.2 User Acceptance Testing

To ensure DeepInvisi is user-friendly, a User Acceptance Testing (UAT) phase was conducted. This testing gathered feedback directly from end-users on how easily the application is used. Table 10 details the UAT questionnaire with 15 questions, which was typically distributed using an online platform like Google Forms and rated the questions using a linear scale from Strongly Disagree (1) to Strongly Agree (5).

Table 10 *User Acceptance Testing (UAT) Form for DeepInvisi Fileless Malware Detection Tool*

Question	Description
1	The DeepInvisi interface is nice and easy (Home, Monitoring, Memory Analysis, Report).
2	On the Home page, are the status indicators (like for status of Backend, Monitoring, Models) clearly show what the system is doing.
3	Starting and stopping real-time monitoring is easy, and the app clearly shows if monitoring is active or inactive.
4	The Monitoring page displays important details for detected events (like command, verdict, PID, score) clearly.
5	The colors and visual cues for detection verdicts (Benign, Suspicious, Malicious) are easy to understand.
6	Is it easy to view the memory analysis for a chosen process.
7	The memory analysis results (like verdict and confidence score) are presented clearly and provide useful information.
8	The 'Suspicious Patterns' found during memory analysis provide helpful extra details for understanding potential threats.

Table 10 *(cont.)*

Question	Description
9	During my testing, DeepInvisi correctly distinguished between safe activities and potentially harmful ones.
10	The pop-up notifications for detected threats appear quickly and provide enough information at a glance.
10	The pop-up notifications for detected threats appear quickly and provide enough information at a glance.
11	The summary of past detections on the Report page dashboard is useful
12	Filtering past detections by date and verdict on the Report page works effectively.
13	The DeepInvisi app felt quick and responsive during normal use and did not noticeably slow down my computer.
14	The application ran smoothly without errors or crashes during my testing.
15	Overall, I am satisfied with DeepInvisi's ability to detect and present information about potential fileless malware threats.

Fig. 33 shows user feedback from 10 respondents who are computer science students on DeepInvisi was highly positive. Almost all users agreed or strongly agreed that the application is easy to use, with clear status indicators and straightforward monitoring controls. Key features like viewing event details, understanding color-coded verdicts, and performing memory analysis were well-received. Users also found DeepInvisi accurate in its detections, responsive, stable, and satisfied with its overall ability to present threat information. The strong positive responses across the board indicate excellent user acceptance of the application.

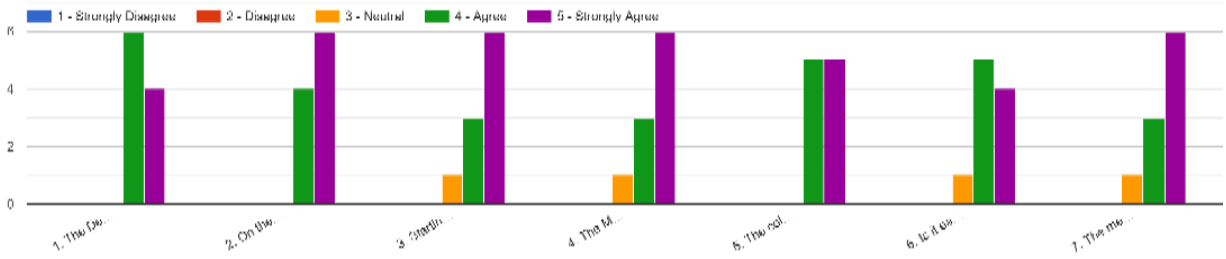

Fig.33 *User Acceptance Test Result (Q1-Q7)*

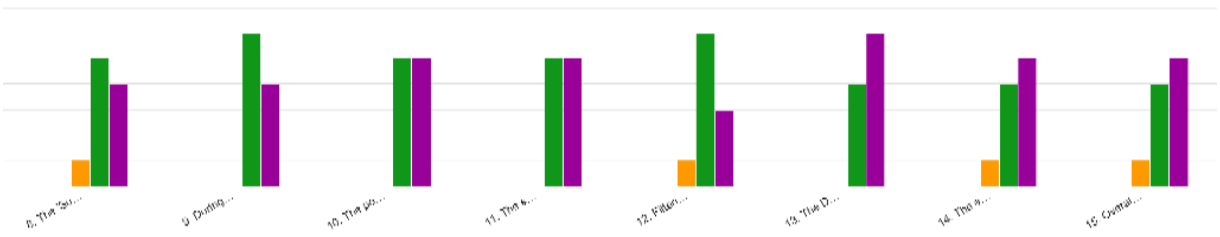

Fig.33 *User Acceptance Test Result (Q8-Q15)*

6. Conclusion

The DeepInvisi Fileless Malware Detection Tool has successfully achieved the objectives of this project. The system was designed as a behavioral-based detection model for fileless malware attacks utilizing a deep learning approach. A key accomplishment was the development of the 'DeepInvisi' tool, which analyzes system behavior patterns through a hybrid architecture combining Transformer and Bidirectional LSTM (BiLSTM) models for both PowerShell script and memory dump analysis. Finally, the proposed tool underwent initial testing to evaluate its accuracy and false positive rate, demonstrating its potential for effective threat detection.

The developed DeepInvisi Fileless Malware Detection Tool addresses the critical challenge of detecting elusive fileless malware by moving beyond signature-based methods and leveraging sophisticated behavioral pattern analysis. DeepInvisi offers several advantages. It provides a user interface for managing monitoring and

viewing results, and its backend effectively detects and can automatically respond to confirmed malicious PowerShell scripts and memory artifacts. The system's ability to generate pop-up notifications alerts users to threats, and its reporting features offer insights into historical detections.

Future improvements can significantly enhance DeepInvisi's capabilities and usability. A primary focus will be to refine the handling of "Suspicious" verdicts. One proposed enhancement involves a post-classification analysis for PowerShell detections initially deemed benign by the DL model. This secondary analysis would examine these benign scripts for specific, known malicious patterns or indicators. If such patterns are found within an otherwise benign-scoring script, its classification could be elevated to "Suspicious," triggering more cautious handling or user alerts. Additionally, expanding the dataset for both PowerShell and memory models with a wider variety of evolving fileless attack techniques will be crucial.

In conclusion, the developed DeepInvisi Fileless Malware Detection tool represents a significant advancement in detecting sophisticated fileless malware through deep learning-driven behavioral analysis. With ongoing refinement, particularly in an advanced analysis of suspicious patterns and continuous model updates, DeepInvisi has the potential to be a highly valuable asset in protecting users against these evolving cyber threats.

Acknowledgment

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

This journal requires that all authors take public responsibility for the content of the work submitted for review. The contributions of all authors must be described in the following manner:

The authors confirm contribution to the paper as follows: ***study conception and design:*** *W. D. Yong, I. R. A. Hamid;* ***data collection:*** *W. D. Yong, I. R. A. Hamid;* ***analysis and interpretation of results:*** *W. D. Yong, I. R. A. Hamid;* ***draft manuscript preparation:*** *W. D. Yong, I. R. A. Hamid. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] C. for I. S. (CIS), "Only in Memory: Fileless Malware – An Elusive TTP." [Online]. Available: https://www.cisecurity.org/insights/blog/only-in-memory-fileless-malware-an-elusive-ttp

[2] A. Afreen, M. Aslam, and S. Ahmed, "Analysis of Fileless Malware and its Evasive Behavior," in *2020 International Conference on Cyber Warfare and Security (ICCWS)*, 2020, pp. 1–8. doi: 10.1109/ICCWS48432.2020.9292376.

[3] P. Borana, V. Sihag, G. Choudhary, M. Vardhan, and P. Singh, "An Assistive Tool For Fileless Malware Detection," in *2021 World Automation Congress (WAC)*, 2021, pp. 21–25. doi: 10.23919/WAC50355.2021.9559449.

[4] J. SÁNCHEZ, "Exploring the VirusTotal Dataset | An Analyst's Guide to Effective Threat Research," VirusTotal. [Online]. Available: https://blog.virustotal.com/2024/08/VT-S1-EffectiveResearch.html

[5] Kaspersky, "Fileless Threats Protection." [Online]. Available: https://www.kaspersky.com/enterprise-security/wiki-section/products/fileless-threats-protection

[6] T. C. N. Experts, "How Fileless Attacks Work and How to Detect and Prevent Them," Aqua Security. [Online]. Available: https://www.aquasec.com/cloud-native-academy/application-security/fileless-attacks/

[7] G. Marín, P. Casas, and G. Capdehourat, "DeepMAL -- Deep Learning Models for Malware Traffic Detection and Classification," 2020. [Online]. Available: https://arxiv.org/abs/2003.04079

[8] K. Baker, "FILELESS MALWARE EXPLAINED." [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/malware/fileless-malware/

[9] B. N. Sanjay, D. C. Rakshith, R. B. Akash, and D. V. V Hegde, "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)*, 2018, pp. 234–239. doi: 10.1109/CSITSS.2018.8768769.

[10] D. Hendler, S. Kels, and A. Rubin, "Detecting Malicious PowerShell Commands using Deep Neural Networks," 2018. [Online]. Available: https://arxiv.org/abs/1804.04177

[11] S. Varlioglu, N. Elsayed, Z. ElSayed, and M. Ozer, "The Dangerous Combo: Fileless Malware and Cryptojacking," in *SoutheastCon 2022*, 2022, pp. 125–132. doi: 10.1109/SoutheastCon48659.2022.9764043.

[12] K. Ding, S. Zhang, F. Yu, and G. Liu, "LOLWTC: A Deep Learning Approach for Detecting Living Off the Land Attacks," in *2023 IEEE 9th International Conference on Cloud Computing and Intelligent Systems (CCIS)*, 2023, pp. 176–181. doi: 10.1109/CCIS59572.2023.10262997.

[13] X. Yang, G. Peng, D. Zhang, Y. Gao, and C. Li, "PowerDetector: Malicious PowerShell script family classification based on multi-modal semantic fusion and deep learning," *China Commun.*, vol. 20, no. 11, pp. 202–224, 2023, doi: 10.23919/JCC.fa.2022-0509.202311.

[14] J. Liu *et al.*, "MemAPIDet: A Novel Memory-resident Malware Detection Framework Combining API Sequence and Memory Features," in *2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2024, pp. 2918–2924. doi: 10.1109/CSCWD61410.2024.10580589.

[15] AWS, "What are Transformers in Artificial Intelligence?" [Online]. Available: https://aws.amazon.com/what-is/transformers-in-artificial-intelligence/

[16] F. Demirkıran, A. Çayır, U. Ünal, and H. Dağ, "An Ensemble of Pre-trained Transformer Models For Imbalanced Multiclass Malware Classification," 2022. [Online]. Available: https://arxiv.org/abs/2112.13236

[17] A. Vaswani *et al.*, "Attention Is All You Need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[18] W. Han, J. Xue, and K. Qian, "A Novel Malware Detection Approach Based on Behavioral Semantic Analysis and LSTM Model," in *2021 IEEE 21st International Conference on Communication Technology (ICCT)*, 2021, pp. 339–343. doi: 10.1109/ICCT52962.2021.9658113.

[19] J. Thickstun, "The Transformer Model in Equations," vol. 1, pp. 1–5, 2019, [Online]. Available: https://homes.cs.washington.edu/~thickstn/docs/transformers.pdf

[20] N. J. Rubenking, "TotalAV Antivirus Pro Review." [Online]. Available: https://www.pcmag.com/reviews/totalav-essential-antivirus?test_uuid=02LlF0iWKsilxYTJVF8uH5y&test_variant=B

[21] N. S. Yadav, V. Goar, and M. Kuri, "AGILE METHODOLOGY - A PERFECT SDLC MODEL WITH SOME IMPROVEMENTS," vol. 7, pp. 2511–2514, 2020, doi: 10.31838/jcr.07.19.306.

[22] S. M. Tan and I. R. A Hamid, "Anti-Ransomware Tools to Detect Crypto Ransomware Based on Machine Learning Approach", aitcs, vol. 5, no. 2, pp. 114–132, Dec. 2024, Accessed: Jun. 07, 2025. [Online]. Available: https://publisher.uthm.edu.my/periodicals/index.php/aitcs/article/view/16516