Deep Learning in Memory Forensics for Stealthy Keylogger Detection

Maliny Thanaraj1, a), Naveen Palanichamy1, 2, b), Su-Cheng Haw1, 2, c), Kok-Why Ng1, 2, d) and A Samydurai3, e)

*1Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100, Cyberjaya, Malaysia*

*2Centre for Digital Innovations, CoE for Immersive Experience, Multimedia University, Persiaran Multimedia, 63100, Cyberjaya, Malaysia*

*3Department of Information Technology, SRM Valliammai Engineering College, Tamil Nadu, India.*

*b) Corresponding author: p.naveen@mmu.edu.my  
a) malinythanaraj27@gmail.com*

*c)sucheng@mmu.edu.my*

*d)kwng@mmu.edu.my*

*e) samyduraia.cse@srmvalliammai.ac.in*

**Abstract**. Memory-resident keyloggers are malware that operates from within volatile memory, avoiding traces on the disk and eluding antivirus software. Recent studies show that these threats utilize techniques such code injection, API hooking, and in-memory encryption to persist in RAM, evading detection. While memory forensic software, like Volatility and Rekall, excel at static analysis, their manual nature means these tools lack automation and scalability for real-time analysis. Likewise, frameworks centered around behavioural detection are unreliable when it comes to fileless malware, largely due to a focus on actionable triggers. This research seeks to fill those gaps by combining memory forensics with deep learning to create a hybrid framework tailored to memory-resident keyloggers. Memory Forensics is the extraction and analysis of volatile artifacts from RAM to uncover hidden processes, while deep learning automates complex pattern recognition and classification. Forensic artefacts were extracted with Volatility and MemProcFS using pslist, malfind, and messagehooks plugins and were subsequently structured. Convolutional neural networks (CNNs) were trained with grayscale images for spatial anomaly detection, and time-series data was input into long short-term memory (LSTM) networks to learn sequential behavior. Public datasets alongside memory dumps created by virtual machines were employed to assess the hybrid system. The results achieved a precision of 98.2% for the CNN and 94.4% for the LSTM, with AUC scores of 0.99 and 0.97 respectively, which affirms the successful implementation of forensics depth and AI pattern recognition. The hybrid system offers a scalable solution for augmenting real-time gaps in modern memory forensic workflows.

# Introduction

Memory-resident keyloggers are sophisticated malware that reside entirely in volatile memory (RAM), leaving no disk traces. They use process hollowing, API hooking, and in-memory encryption to evade antivirus tools and sandboxes. Carbone [1] emphasized how their memory-only presence complicates detection, while Case and Richard [2] detailed their runtime manipulation to remain hidden. As reliance on cloud, virtualization, and IoT grows, such threats are increasingly common [3].

Memory forensics is a key approach to detecting them. Tools like Volatility and MemProcFS extract artifacts like hidden processes, injected DLLs, and keyboard hooks using plugins such as pslist, malfind, and messagehooks [4]. However, these tools are manual, making them time-consuming and unfit for real-time or large-scale use [5], [6]. Deep learning introduces automation, CNNs classify visual memory patterns, while LSTMs model temporal behavior across processes and DLLs [7], [8], [9]. Yet, dataset diversity and model generalizability remain issues [10].

This research proposes a hybrid framework combining forensic extraction with CNN and LSTM models. By converting memory data into deep-learning-friendly formats, it enables semi-automated keylogger detection across public and simulated datasets, blending forensic depth with AI scalability.

The rest of this paper is organized as follows: Section 2 discusses related work, Section 3 describes the proposed methodology, Section 4 presents the experimental results and analysis, and Section 5 concludes with insights and future work.

# Literature review

## Memory-Resident Keyloggers

Memory-resident keyloggers function without using persistent storage, operating solely within volatile memory which leaves minimal traces forensic investigators could analyze. As Carbone noted in his analysis of malware in memory, “they avoid disk-based detection by not using the filesystem at all” [11]. Case and Richard [12] demonstrated that such malware survives reboots using code injections to persist throughout runtime only to erase itself immediately afterward. Fragmentation and polymorphic techniques designed to conceal malware further complicate their detection, especially in virtualized and IoT settings [13], [14], [15]. Also, commanding keystroke logging through APIs such as SetWindowsHookEx() keeps the alert threshold dormant [16], [17]. Recent ransomware such as Conti (2021), BlackCat (2022), and LockBit (2023) adopt memory-resident tactics like in-memory encryption and process injection, reinforcing the urgency of memory forensics in modern detection efforts [31], [32], [33].

## Memory Forensics Technique

Analyzing memory forensics is a process conducted on computers' random access memory (RAMs) to detect possible breaches in them that may take the form of code injection or process concealment. Extracting memory-level data can be done with tools like Volatility, MemProcFS, and Rekall. With each having its unique functionalities, Volatility uses plugins like pslist and malfind, while MemProcFS enables dump browsing as virtual filesystems. Rekall offers faster parsing but lacks plugin richness [18], [19], [20], [21]. Despite the advantages mentioned, these tools are limited by the need for them to be operated manually which poses real-time application and scalability challenges [15], [22]. More recently, efforts have been made towards reducing manual dependence with some applying memory visualization techniques like Acien et al. who visualized memory contents for CNN-based scanning and Tareen and Akhunzada who used grayscale transformation to mark spots of anomalies [23], [24].

## Deep Learning for Malware Detection

Deep learning models have been widely adopted for malware classification. CNNs perform well on spatial analysis of grayscale memory dumps [13], [25], while LSTMs handle sequential forensic data. Hasan and Dhakal [14] showed improved detection using LSTMs trained on API call chains. Singh et al. [26] simulated keylogger activity and successfully modeled behavior using LSTM. Still, issues remain, Noor and Qadir [22] highlighted a lack of publicly labeled datasets, and MRm-DLDet’s CNN model suffers from hardware demands in real-time deployment [25]. Hybrid models have been proposed to reduce load and improve accuracy. Acien suggested CNN pre-processing [23], Forecast Labs introduced symbolic prediction [27], and Khan et al. proposed LSTM filtering for real-time use [28].

## Comparative Analysis of Detection Frameworks

Different detection tools have varying strengths and limitations. Volatility and MemProcFS were chosen for their forensic depth and structured output, while tools like Rekall and AVML were excluded due to limited plugin support and lack of deep learning compatibility. HookTracer [29] enables real-time API monitoring but is noisy. MRm-DLDet offers high CNN accuracy but lacks sequential modeling and demands heavy hardware. Forecast [27] uses symbolic prediction without deep learning. The proposed CNN–LSTM system improves accuracy, offers partial real-time detection, and enhances usability through plugin-level integration.

## Identified Gaps in Literature

Despite progress, key challenges persist. Many forensic tools still depend on manual analysis, slowing investigations and increasing errors. Most models operate on static dumps, lacking real-time detection. Limited dataset diversity hinders generalization, and many approaches isolate either spatial or temporal features, missing hybrid benefits. Usability is also limited by technical complexity.

This study introduces a hybrid CNN–LSTM model trained on structured Volatility and MemProcFS data. CNN detects spatial anomalies; LSTM models runtime behavior. Combined with a GUI, the system enables semi-automated, analyst-friendly detection. While Fatima et al. [30] proposed a conceptual framework, this work delivers practical implementation and testing.

# Methodology

This study presents a hybrid detection framework combining memory forensics and deep learning for identifying stealthy memory-resident keyloggers as shown in Figure 1. The methodology covers memory acquisition, forensic feature extraction, data transformation, model training, evaluation, and GUI deployment. Volatility and MemProcFS act as the forensic core, while CNN and LSTM models handle classification. The system also supports real-time memory capture via WinPMEM and allows automated plugin execution through a GUI interface with user-selected options.



**FIGURE 1.** Hybrid detection framework pipeline

## Design, Dataset and Feature Extraction

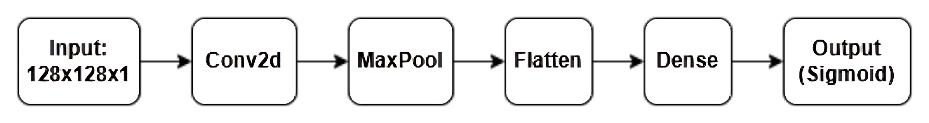
The system operates in five stages: (i) memory acquisition from public datasets and Windows virtual machines or directly from live systems using WinPMEM, (ii) artifact extraction via Volatility (*pslist, malfind, dlllist, messagehooks*) and MemProcFS, (iii) conversion into image and sequence formats, (iv) CNN/LSTM model training, and (v) GUI-based deployment. Datasets included benign and malicious dumps, labeled based on plugin outputs showing anomalies like injected code or active hooks. An 80:20 split was used for training and testing. Volatility plugins were essential for identifying active processes, memory injections, DLL loads, and keyboard hooks. MemProcFS enabled filesystem-like navigation of .raw dumps. Plugin outputs were saved as CSV (for LSTM) and binary matrices (for CNN).

## Data Preprocessing and Model Configuration

For CNN, memory artifacts were converted into 128×128 grayscale images using OpenCV and normalized. However, for LSTM, CSV data was scaled, padded, and reshaped to [samples, 1, features].

### CNN Architecture:

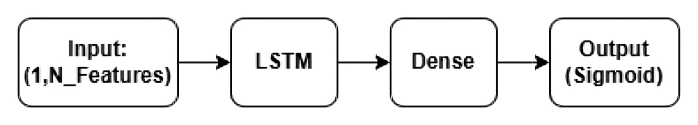
Figure 2 shows the CNN architecture which begins with an input of 128x128x1grayscale image.



**FIGURE 2.** CNN model architectures

### LSTM Architecture:

Figure 3 presents the architecture of the LSTM to work with CSV files that are reshaped to [1,N\_Features]



**FIGURE 3.** LSTM model architectures

## Evaluation Metrics and Interface

This section presents the evaluation metrics used to measure model performance, including accuracy (Equation (1)), precision (Equation (2)), recall (Equation (3)), F1-score (Equation (4)), and AUC, which are computed based on true/false positive and negative rates shown in Table 1.

(1)

(2)

(3)

(4)

**TABLE 1.** Model performance metrics

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **AUC** |
| CNN | 97.8% | 0.98 | 0.96 | 0.97 | 0.99 |
| LSTM | 94.4% | 0.92 | 0.93 | 0.93 | 0.97 |

All detection stages, including live memory capture, plugin execution, and classification, are managed within a single GUI. This enhances usability and facilitates semi-automated forensic triage without command-line interaction.

# Results and discussion

## Evaluation Overview

This section presents the outcomes of testing the hybrid detection system using CNN and LSTM models. Models were trained separately on structured forensic data from Volatility plugin outputs and evaluated using accuracy, precision, recall, and F1-score. Outputs were validated using console metrics, training logs, and GUI interaction.

Figure 4 and Figure 5 show the input and output system interface. The detection system comprises three layers: memory preprocessing, deep learning classification, and a Tkinter-based GUI. The interface allows memory dump uploads, model selection (CNN/LSTM), and displays classification results with confidence levels.

A screenshot of a computer program

AI-generated content may be incorrect.

**FIGURE 4.** GUI interface for uploading memory dumps

A screenshot of a computer

AI-generated content may be incorrect.

**FIGURE 5.** Output interface showing classification result (malicious/benign)

## CNN Evaluation

The CNN was trained on 128×128 grayscale images from plugin outputs (e.g., malfind, dlllist) using Conv2D, ReLU, MaxPooling2D, Flatten, and Dense layers. It trained over 5 epochs with binary cross-entropy and Adam optimizer. As shown in Table 2 and Figure 6, CNN achieved high precision with low false positives due to strong spatial feature extraction and minimal overfitting.

**TABLE 2.** CNN evaluation

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Accuracy | 98% |
| Precision | 0.98 |
| Recall | 0.97 |
| F1-Score | 0.97 |

A screen shot of a number

AI-generated content may be incorrect.

**FIGURE 6.** CNN terminal output showing accuracy, precision, recall

## LSTM Evaluation

The LSTM model processed normalized .csv sequences from process and DLL logs, with a single LSTM layer (128 units), dropout (0.2), and a sigmoid output. Trained for 5 epochs, it modeled runtime anomalies well (Table 3, Figure 7). Slightly lower precision than CNN was linked to noise in plugin data, but recall remained high, showing effective threat detection.

**TABLE 3.** LSTM evaluation

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Accuracy | 94% |
| Precision | 0.92 |
| Recall | 0.93 |
| F1-Score | 0.93 |

A screenshot of a computer code

AI-generated content may be incorrect.

**FIGURE 7.** LSTM terminal output showing accuracy and loss

## Comparative Model Analysis

As shown in Figure 8 and Table 4, CNN slightly outperformed LSTM due to its structured visual input. LSTM remains effective for modeling temporal behaviors in stealthy malware. The hybrid system improves over standalone tools like Volatility or HookTracer by fusing forensic plugins with deep learning.

The system detects memory-resident keyloggers by combining CNN for spatial patterns and LSTM for behavioral analysis. Strong performance and GUI support highlight its potential for practical forensic use, enabling semi-automated analysis of volatile memory threats.

**TABLE 4.** Model performance comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **Strengths** |
| CNN | 98% | 0.98 | 0.97 | 0.97 | High visual precision, low FP |
| LSTM | 94% | 0.92 | 0.93 | 0.93 | Strong on sequential anomaly |
| Volatility | Manual | N/A | N/A | N/A | Deep forensic granularity, slow |
| HookTracer | Real-Time | 0.81 | 0.84 | 0.82 | Fast but prone to noise |



**FIGURE 8.** Bar graph – accuracy of volatility, HookTracer, CNN, LSTM, Hybrid

# Conclusion and future work

This research introduced a hybrid approach for detecting memory-resident keyloggers, a malware that operates exclusively within volatile memory and evades traditional detection tools. The system automates memory dump classification using forensics and deep learning as malicious or benign. Volatility and MemProcFS were used to extract forensic artifacts such as hidden processes, DLLs, message hooks, and injected code. These features informed both manual analysis and the training of CNN and LSTM models. The CNN achieved 98% accuracy with an AUC of 0.99 using grayscale image representations, while the LSTM reached 94% accuracy with an AUC of 0.98 using sequential plugin data. The framework was deployed with a Tkinter-based GUI, enabling user interaction and real-time classification, effectively bridging the gap between static manual analysis and scalable AI-driven detection. The system also integrates plugin-level automation and live memory acquisition through WinPMEM, making it applicable in dynamic investigation workflows. Limitations include a small, potentially biased dataset and lack of support for Linux memory formats. Future work will address dataset balancing (e.g., via SMOTE), Linux memory support, plugin-level automation, and integration into live security infrastructures. Overall, this fusion of forensic depth and AI scalability enhances the accuracy and efficiency of detecting stealthy, memory-resident threats.

# References

1. R. Carbone, “Malware memory analysis for non-specialists,” *Digital Investigation* **11**, S1–S9 (2014).
2. A. Case and G. G. Richard, “Memory forensics: The path forward,” *Digital Investigation* **20**, 23–33 (2017).
3. I. Hamid and M. M. H. Rahman, “A comprehensive literature review on volatile memory forensics,” *Electronics* **13**, 3026 (2024).
4. I. Hamid, R. Alajlan, and K. Riad, “Advancing malware artifact detection and analysis through memory forensics: A comprehensive literature review,” *J. Theor. Appl. Inf. Technol.* **29**, 122–134 (2024).
5. B. Noor and S. Qadir, “Machine learning and deep learning based model for the detection of rootkits using memory analysis,” *Appl. Sci.* **13**, 10730 (2023).
6. S. M. R. Hasan and A. Dhakal, “Obfuscated malware detection: Investigating real-world scenarios through memory analysis,” *arXiv:2404.02372* (2024).
7. J. Liu, Y. Feng, X. Liu, J. Zhao, and Q. Liu, “MRm-DLDet: A memory-resident malware detection framework based on memory forensics and deep neural network,” *Cybersecurity* **6**, 3 (2023).
8. Á. Acien, J. Morales, J. V. Monaco, R. Vera-Rodriguez, and J. Fierrez, “TypeNet: Deep learning keystroke biometrics,” *arXiv:2101.05570* (2021).
9. A. Singh, P. Choudhary, A. K. Singh, and D. K. Tyagi, “Keylogger detection and prevention,” *J. Phys.: Conf. Ser.* **2007**, 012005 (2021).
10. D. Yaffe and D. Hendler, “Early detection of in-memory malicious activity based on run-time environmental features,” in *Lecture Notes in Computer Science* **12426**, 466–486 (2021).
11. Á. A. Royo, M. S. Rubio, W. Fuertes, M. C. Cuervo, C. A. Estrada, and T. Toulkeridis, “Malware security evasion techniques: An original keylogger implementation,” in *Adv. Intell. Syst. Comput.* **1365**, 375–384 (2021).
12. K. Agrawal, N. Parmar, N. Sharma, and P. Mishra, “Detection of keyloggers in browser memory using memory forensics,” in *Proc. Int. Conf. Intelligent Systems* (2022).
13. E. Lopez and K. Sartipi, “Detecting the insider threat with long short-term memory (LSTM) neural networks,” *arXiv:2007.11956* (2020).
14. I. Khan, S. Qureshi, and S. Latif, “Anomaly detection in memory dumps using LSTM with signature pre-filters,” *Neural Comput. Appl.* **35**, 4713–4726 (2023).
15. W. U. Tareen and A. Akhunzada, “System memory visualization for malware detection in enterprise systems,” *Security Privacy* **5**, e177 (2022).
16. O. Alrawi et al., “Forecasting malware capabilities from cyber attack memory images,” in *Proc. USENIX Security Symp.* (2021).
17. A. Case et al., “HookTracer: Automatic detection and analysis of keystroke loggers using memory forensics,” *Comput. Secur.* **96**, 101872 (2020).
18. H. Fatima, M. Singh, and F. A. Bakar, “A hybrid framework for keylogger detection using CNN-LSTM and memory forensics,” *Malaysian J. Digit. Forensics* **6**, 102–115 (2023).
19. A. Christopher, N. Mandela, T. Mbinda, and J. Makopa, “The hidden threat: Detecting API hooking in memory forensics using open-source tools,” in *Proc. ICCAMS 2023* (2023).
20. V. C. Ekele, A. A. Adebiyi, and O. E. Igbekele, “Keylogger detection: A systematic review,” in *Proc. SEB-SDG 2023* (2023).
21. J. Saxe and K. Berlin, “eXpose: A character-level convolutional neural network for detecting malicious URLs, file paths and registry keys,” *arXiv:1702.08568* (2017).
22. M. Bayzid, M. Shoikot, J. Hossain, and A. Rahman, “Keylogger detection using memory forensic and network monitoring,” *Int. J. Comput. Appl.* **177**, 17–21 (2019).
23. Forecast Labs, “Symbolic malware intent forecasting using memory snapshots,” in *Forecast Technical Papers* (2023).
24. K. Griffin, J. Schneider, and W. Lee, “Memory-based threat detection using control-flow graph hashing,” *IEEE Trans. Inf. Forensics Secur.* **14**, 1800–1814 (2019).
25. S. Chatterjee, R. Dutta, and I. Sengupta, “RAM scraper malware detection through anomaly signatures in memory dumps,” *J. Inf. Secur. Appl.* **54**, 102553 (2020).
26. F. Liu, X. Xia, Y. Zhang, and Y. Liu, “Malicious process detection based on process memory images and deep learning,” *Comput. Sci. Inf. Syst.* **19**, 1455–1472 (2022).
27. M. Cohen and J. Metz, “MemProcFS: A file system interface for memory forensics,” (2021).
28. M. H. Ligh, A. Case, J. Levy, and A. Walters, *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory* (Wiley, 2014).
29. Forecast, “Behavioral intent prediction in memory forensics,” *Cyber Threat Intell. J.* **11**, 211–227 (2023).
30. R. Fatima, R. Khalid, and T. Khan, “Fusion of Volatility artifacts with deep learning for memory-resident malware detection,” *J. Netw. Comput. Appl.* **216**, 103560 (2023).
31. M. H. A. Khan, S. M. R. Hasan, and N. J. Khan, “Deep learning-based hybrid detection of zero-day malware using memory forensics and autoencoder reconstruction,” *arXiv:2301.06229* (2023).
32. A. R. Tareen and A. Akhunzada, “System memory visualization for malware detection in enterprise systems,” *Security Privacy* **3**(4), e177 (2020).
33. M.A. Daniel, S.-C. Chong, L.-Y. Chong, and K.-K. Wee, “*Optimising Phishing Detection: A Comparative Analysis of Machine Learning Methods with Feature Selection*,” Journal of Informatics and Web Engineering **4**(1), 200–212 (2025).