Chapter Title: Detecting Malicious PDFs with Embedded JavaScript: A Hybrid Approach

**Table of Contents**

1. **Introduction**
2. **Background and Literature Review**
3. **Methodology**
4. **Tools and Technologies Used**
5. **Dataset**
6. **Environmental Set Up**
7. **System Architecture**
8. **Implementation Details**
9. **Results and Discussion**
10. **Future Work**
11. **NovelAspects**

**INTRODUCTION**

In today's digital environment, the data is exchanged in various forms. Securing these data is an utmost concern, given the rate at which threats are evolving. Malicious contents refer to any harmful data or information which may include harmful links, phishing emails, or malwares, designed in order to make users compromise their security paving the way for attack. Malware is a subset of malicious content, it specifically refers to software created to disrupt, damage, or gain unauthorized access to computer systems, or networks. Malware is designed to harm systems.

With the evolution of technology, malware is becoming more complex. It can be broadly classified into File based and file less they differ in terms of techniques in compromising systems and evading detection. File based malwares typically relies on files stored on the disk to execute and spread. These types of malwares usually require the users to download or open the files to infect the system. These can be detectable through methods such as signature scanning and behavioural analysis. On the other hand, fileless malware exploits in-memory processes and legitimate system tools, such as PowerShell or Windows Management Instrumentation (WMI), to execute malicious activities directly in memory. This approach minimizes its footprint, While the term ’fileless malware’ might imply the absence of files, it actually refers to a technique where no executable files are left on the hard disk during the attack.

Attackers are always working to find a way to infect the systems with the malwares and steal or damage data. One of the most used forms of communication to exchange data in today's scenario is PDF files. The PDFs are becoming the effective targets for the attackers as they can embed malicious contents in it.

The PDF format provides many advantages when it is used as an exploitation method:

• creation of malicious PDFs is easy.

• it enables JavaScript functionality.

Attackers thus take the help of JavaScript which is dynamic in nature to embed it in the PDF documents and thus perform various malicious activities. JavaScript is often embedded in specific sections of the PDF structure, such as:

* **Action Dictionaries**: Where scripts can be triggered by an /OpenAction, executing when the document is opened.
* **Annotations and AcroForm Fields**: JavaScript can be embedded in annotations or form fields, triggering actions like MouseEnter or Submit.
* **Embedded File Streams**: Scripts may also be included within embedded streams, often obfuscated to evade detection, where the /JS or /JavaScript keys point to the code.

These interactive features allow attackers to conceal malicious code, making PDFs a potent tool for cyberattacks.

**BACKGROUND AND LITERATURE REVIEW**

PDF files, being the most widespread document format, are now being exploited as a vector for malware attacks. Malicious JavaScript code in obfuscated form can be embedded within these files. The obfuscation makes it difficult to interpret the code’s intentions. Existing methods, like static document analysis are not very efficient to determine the actual intention of the code, as these scripts can evade detection from antivirus since they hide themselves inside legitimate documents. Therefore, dynamic code execution for behavioural analysis of the code is essential for detection of malicious content in a file. However, dealing with dynamic analysis is difficult as code execution might harm the system and it requires extensive sandboxing.

**Literature Review**

“JAVASCRIPT MALWARE BEHAVIOR ANALYSIS AND DETECTION USING SANDBOX ASSISTED ENSEMBLE MODEL” [1]

P. Kishore, S. K. Barisal, and D. P. Mohapatra, “Javascript malware behavior analysis and detection using sandbox assisted ensemble model,” in 2020 IEEE Region 10 Conference (TENCON), Osaka, Japan, November 16–19 2020.

This paper presents a sandbox-assisted ensemble model for detecting JavaScript malware embedded in web pages. The methodology involves using the malware-jail sandbox for dynamic code execution, extracting and selecting relevant features, and applying an ensemble model combining SMO, Voted Perceptron, and AdaBoost algorithms. This approach achieves high accuracy, effectively identifying both obfuscated and de-obfuscated malicious JavaScript with 99.6% accuracy.

Take Away:

The ensemble model accurately detects JavaScript malware through dynamic sandbox analysis and feature selection, achieving 99.6% accuracy in identifying malicious code.

“EFFICIENT REAL-TIME FILELESS MALWARE DETECTION TOOL FOR MACRO-BASED MICROSOFT OFFICE DOCUMENTS” [2]

S M Aminur Rahman, “Efficient Real-Time Fileless Malware Detection Tool For Macro-Based Microsoft Office Documents”, University of the West of England, Bristol, UK, Journal Contribution Posted on March 05,2023.

This paper presents CERBERUS, a real-time fileless malware detection tool for macro-based Microsoft Office documents. The methodology involves collecting process information, identifying Microsoft Office processes, monitoring for child processes, checking for PowerShell activity, and taking action if malicious behaviour is detected. CERBERUS also logs process details, notifies administrators, and suspends the system for memory forensics if needed.

Take Away:

real-time detection of fileless malware in Office documents by monitoring processes and PowerShell activity. It enhances protection through detailed logging, administrator notifications and suspending system's activity.

“FILELESS MALWARE THREATS: RECENT ADVANCES, ANALYSIS APPROACH THROUGH MEMORY FORENSICS AND RESEARCH CHALLENGES” [3]

I. Kara, “Fileless malware threats: Recent advances, analysis approach through memory forensics and research challenges,” Expert Systems with Applications, vol. 214, no. 6, pp. 119–133, October 2022.

The study utilized a dataset of fileless malware samples, including the Kovter virus, The proposed method involved gathering memory data using tools like Volatility and conducting a detailed analysis of the memory dump files to identify and understand the behaviour of fileless malware. The workflow of the proposed approach included three phases: gathering memory data, conducting memory analysis, and analysing the results.

Take Away:

Memory forensics effectively reveals how file less malwares hide in memory, and enhances the ability to defend against fileless malware.

“AN ASSISTIVE TOOL FOR FILELESS MALWARE DETECTION” [4]

P. Borana, V. Sihag, G. Choudhary, M. Vardhan, and P. Singh, "An Assistive Tool for Fileless Malware Detection," in Proceedings of the Conference, Sardar Patel University of Police, Security and Criminal Justice, Jodhpur, VIT Bhopal University, Bhopal, and National Institute of Technology, Raipur, India, Aug. 2021.

The tool detects fileless malware by monitoring real-time processes and network activity, identifying high-priority and suspicious processes, and tracking command-line arguments and dynamic link libraries. It also scans files and directories using VirusTotal and analyses process hierarchies to uncover hidden threats.

Take Away:

The tool detects fileless malware by monitoring processes and network activity, and scanning files with VirusTotal.

**METHODOLOGY**

Our proposed approach consists of 5 stages:

**1. Static Analysis**

**PDF Download Detection and JavaScript Extraction**

In the initial phase, the system monitors the files being downloaded. Our approach uses a python library called "Watchdog" to observe any PDF documents being downloaded. Watchdog is a python API library used to monitor file system activities. If a PDF document is detected then it is used in further steps.

Next, we check if any JavaScript code is embedded in the PDF document, if found then the JavaScript code is extracted. This is done with the help of pdfextract tool of Origami framework which can extract binary resources such as images, scripts, fonts, data streams, attachments from pdfs and creates a dump folder for that pdf. Our focus is on the "scripts" subfolder of the dump, as it contains the JavaScript code which is embedded in PDFs. This provides the embedded code, whether in obfuscated or de-obfuscated form, which plays a significant role in the detection process.

**2. Dynamic Analysis**

**Dynamic analysis of JavaScript in a Sandbox**

After the extraction of JavaScript code, the code is passed to malware-jail sandbox for analysis. After dynamic analysis of the code in the sandbox which involves executing the script, a log will be created which has information that provides insights into the actions the JavaScript intended to perform. Dynamic analysis is named as such because it involves running the JavaScript code and monitoring its behavior in real-time, as opposed to static analysis. Static analysis, in contrast, only involves examining the code without actually executing it.

**Dynamic elements:**

**Execution-based:** The JavaScript is actually executed in an environment similar to where it would naturally run (a browser or server), allowing observation of real-world actions.

**Behavior Monitoring:** Instead of merely reading the code (static), dynamic analysis tracks what the code does.

**Environment Interaction:** The sandbox allows interaction between the JavaScript and a simulated environment, which reveals any malicious intent that may only be visible during execution.

**Runtime Execution:** The JavaScript code is executed in a simulated, controlled environment.

**Behavioral Logging:** Dynamic analysis captures and logs behaviors as they occur during runtime, which may include network communication, file changes, or API usage.

**Detection of Runtime-only Behavior:** Some malicious activities are only triggered when the code runs, such as conditional logic based on the user’s environment. Static analysis might miss these, but dynamic analysis will reveal them.

In conclusion, dynamic analysis allows for a more thorough understanding of JavaScript's true behavior by focusing on what the code does when executed. This approach uncovers malicious activities that could be hidden or obfuscated in the static code.

**Feature Selection**

To define features that are useful for classification, we analysed a total of 3074 samples, which consisted 1559 malicious JavaScript samples. We conducted statistical analysis of the logs, mentioned in the previous section, to understand the features of malicious code to differentiate it from benign code. Before selecting the features, we referred to existing research that presented static features of malicious JavaScript code. However, our approach focuses on the features obtained after dynamic analysis of the scripts.

To define the most relevant features for classification, we manually reviewed logs of both malicious and benign code samples after executing it in the malware-jail sandbox and selected around 8 features. Remarkably, benign JavaScript code completely deviates from these features, allowing us to effectively differentiate malicious samples from benign samples. Each of the features are defined briefly below.

1. Total Bytes Allocated: This measures the total memory usage to find any unusual behaviours. Excessive memory usage often indicates that it is performing resource intensive activities like creating multiple processes and attempting to hide it.
2. Embedded iframes: iframes are targeted by the attackers to embed malicious scripts or redirect the users to malicious websites which are common methods of phishing attacks and drive-by download attacks.
3. Registry Access: Scripts often try to change system configurations, accomplish persistence in the system by modifying Windows Registry with the help of "WScript.Shell" object in scripting environments.
4. ActiveX Controls: It is used by scripts to conduct unauthorised activities, such as making network connections or accessing file systems by running arbitrary code which poses a huge security risk.
5. Data Types and Handling: Some objects and methods in data handling signifies malicious activities, such as:

* "ADODB\_Stream": used for writing binary data, which can include executing malicious scripts.
* "Scripting\_Dictionary": used to store data in memory.
* "VBScript\_RegExp": used for obfuscation to hide its intention.

1. Network Activities: Malicious scripts try to communicate with external servers to download extra payloads, or connect to harmful servers using the "MSXML2.XMLHTTP" object, which allows scripts to send requests and thus interact with servers.
2. Write Operations: write operations such as "document.write" and "document.writeln" are used by the scripts to dynamically insert content which can be malicious code.
3. Directory and File Operations:  Scripts might use "FileSystemObject" to access, delete or modify files on the system, and they might install malicious software into these accessed files.

**3. Machine Learning-Based Classification**

**Model Training and Classification**

After analysing all the samples we had, we created a dataset in CSV format having 3074 rows with the features mentioned in the section above.

To classify the extracted JavaScript code as malicious or benign, we built a model using a random forest algorithm with the dataset mentioned in the previous section. The model training took around 0.4537 seconds. We created a python script to get the defined features from the extracted JavaScript code and feed it as input to this model to detect malicious behaviour. Finally, it identifies the downloaded PDF as a malicious PDF if it has malicious JavaScript embedded in it; else, it considers it as a benign PDF.

**4. Memory Analysis**

**File-Based Malware Detection (Executables) (Yet to be implemented)**

If the PDF is classified as Malicious, the system checks if the malware is file-based by looking for associated executable files:

● The system searches for executable files that are intended to drop on the hard disk by the malicious PDF, we can see the possibility of this in dump files from malware jail sandbox’s dump.

● If executables are found, the PDF is flagged as file-based malware.

● The user is notified, and the system destroys the PDF to prevent further harm.

**File-less Malware Detection (Memory Analysis) :**If no executables are found, the system investigates the possibility of file-less malware (malware that resides only in memory):

● The system captures a memory image using tools like avml or lime, which save the current memory state of the system.

● The memory image is analysed using Volatility, a tool that specializes in memory forensics.

● Volatility looks for indicators of file-less malware in the memory. If such indicators are found, the system flags the PDF as file-less malware.

**Features to be considered for file less malware detection:**

**1.Check for File System Artifacts:**

**Check:** Verify if there are no suspicious files or executables on the file system.

**Evidence:** Use file system analysis tools or checksums of known directories.

**Conclusion:** If no suspicious files are found and the malware is still active, it suggests a fileless nature.

**2. Analyze Process and Memory Artifacts**

**Running Processes:**

**Check:** Identify processes related to known applications or services (e.g., Adobe Flash, browsers)

**Evidence:** Fileless malware often injects or manipulates existing processes rather than creating new ones.

**Injected Code:**

**Check:** Use malfind to find injected code within legitimate processes.

**Evidence:** The presence of injected or hidden code within processes like browsers or Adobe applications points to fileless malware.

**Hidden Processes and DLLs:**

**Check:** Look for hidden processes or DLLs

**Evidence:** Hidden or unlinked DLLs associated with legitimate processes suggest exploitation techniques common in fileless malware.

**5. Mitigation**

**Final Actions and Destruction of the PDF (Yet to be implemented)**

After the analysis:

● If file-based or file-less malware is detected, the system notifies the user and takes action to destroy the malicious PDF.

● If no signs of malware are found, the PDF is flagged as safe.

**TOOLS USED**

**1. Watchdog**

Watchdog is a Python library used for real-time monitoring of file system events such as file creation, modification, deletion, and movement. It works across major platforms like Windows, macOS, and Linux providing instant detection of changes.

In our approach, we have customized Watchdog to specifically monitor PDF documents. When a PDF file is detected, the system triggers further steps in our process, ensuring a targeted and efficient workflow.

**2. Origami**

Origami is a Ruby framework for parsing, modifying, and creating PDFs, with support for features like encryption, digital signatures, and forms. It includes tools like **pdfextract**, which extracts embedded resources such as images, scripts, and fonts from a PDF's internal structure.

Other general-purpose PDF analysis tools like PDF-Parser, PDFTK, and Apache PDFBox differ from Origami in several key aspects:

**Implementation and Integration**

General Tools: implemented in different languages (e.g., Python, Java) and are designed for broad use cases like text extraction, PDF manipulation, and content rendering. They are versatile but not specifically tailored for security-focused tasks.

Origami: Though implemented in Ruby, Origami’s design allows it to integrate well into security and forensic workflows, making it easier to use alongside other security tools.

**Heuristic Analysis**

General Tools: Most general-purpose tools do not include heuristic analysis for detecting potentially malicious content. They focus on extracting and processing content without assessing its safety or intent.

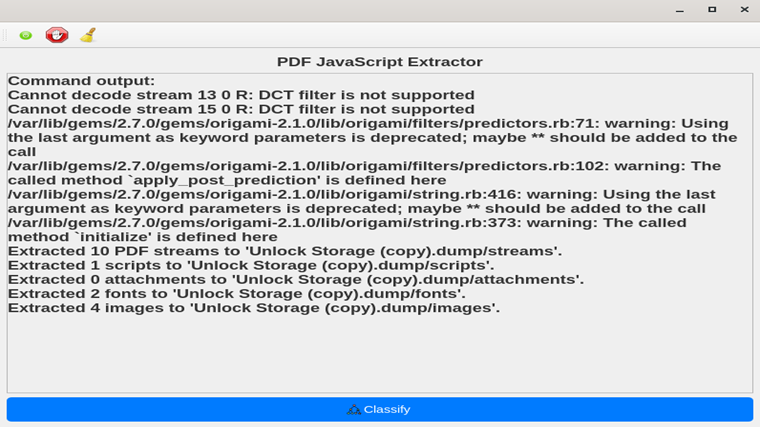
Origami: Origami is equipped with heuristic checks that specifically look for signs of malicious behavior in embedded JavaScript, offering a security advantage.

**Depth of PDF Structure Analysis**

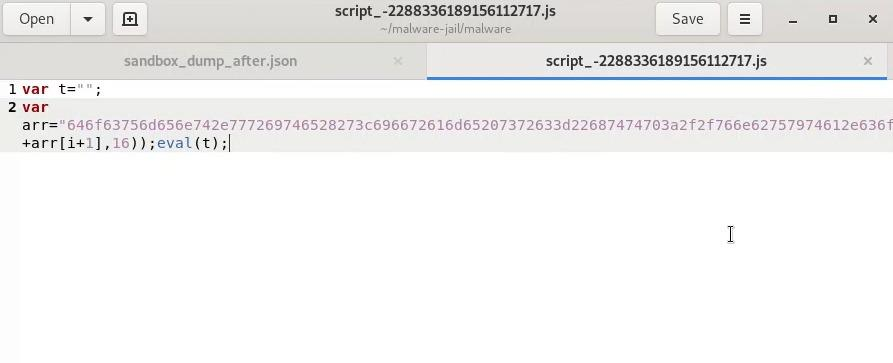
General Tools: Tools like PDF-Parser and PDFTK provide basic analysis of the PDF structure, such as listing objects or extracting text. They don't typically delve into how different elements of the PDF interact or analyze scripts embedded within the structure.

Origami: Origami goes beyond basic extraction, offering a detailed analysis of the entire PDF structure, which is essential for identifying suspicious interactions within the document.

In our approach, we use the **pdfextract** tool to check if any JavaScript code is embedded within a PDF document. If JavaScript is detected, the tool extracts the code, which is stored in the "scripts" subfolder of the dump. This folder contains the embedded JavaScript, whether in obfuscated or de-obfuscated form. Extracting this code is crucial for analysing potential malicious activities and plays a key role in the detection process.



Extracted files embedded in PDF using origami(pdfextract tool)

Extracted JS code in obfuscated form

**3. Malware-jail sandbox**

Malware-Jail is a Node.js tool for analysing JavaScript malware within a controlled environment. This sandbox is a semi-automatic which extracts payload, performs dynamic analysis of the payload and provides a controlled environment.

Malware Jail Sandbox differs from other sandboxes in several key ways:

**JavaScript Deobfuscation:** Unlike other sandboxes that are designed for a wide range of file types and malware, Malware Jail is particularly optimized for analyzing JavaScript, Malware Jail specializes in de-obfuscating JavaScript code, effectively revealing hidden malicious functions, a feature that many other sandboxes might lack or not perform as thoroughly.

**Customizable Analysis Environment:** It offers extensive customization options specifically for JavaScript analysis, allowing users to simulate different environments, which may not be as flexible in general-purpose sandboxes like Cuckoo.

**Detailed Behavioral Monitoring:** Malware Jail provides more granular and detailed monitoring of JavaScript execution, capturing intricate behaviors that other sandboxes may miss, especially those focused on broader malware types.



Figure 3.Deobfuscated js code from Malware jail sandbox

**4. AVML**

AVML (Azure Virtual Machine Memory Loader) is a command-line tool used to capture the memory of Linux systems, including virtual machines. It works by reading the system’s memory directly and saving it to a compressed file. This memory capture is a key step in our process, and we may also use **LiME** as an alternative for capturing memory images. Both tools ensure reliable memory acquisition without the need for additional kernel modules.

**5. Volatility**

Volatility is an open-source memory forensics tool designed to analyse and extract information from volatile memory dumps. It can extract valuable data such as running processes, open files, and network connections, making it highly effective for identifying malware and malicious activities hidden in memory. Supporting Windows, Linux, and macOS, Volatility offers extensive plugin support for various analysis tasks, including process listing and registry extraction, all accessible through a flexible command-line interface. In our approach, we will use Volatility to analyse captured memory images and detect potential threats using specific volatility commands.

**Wide OS Support:** Analyzes memory from multiple operating systems with OS-specific plugins. but tools like Redline and Memoryze are more Windows-centric.

**Diverse File Formats:** Supports various memory dump formats without needing conversion.

**Specialized Malware Detection:** Includes plugins for identifying malware techniques like code injection and rootkits.

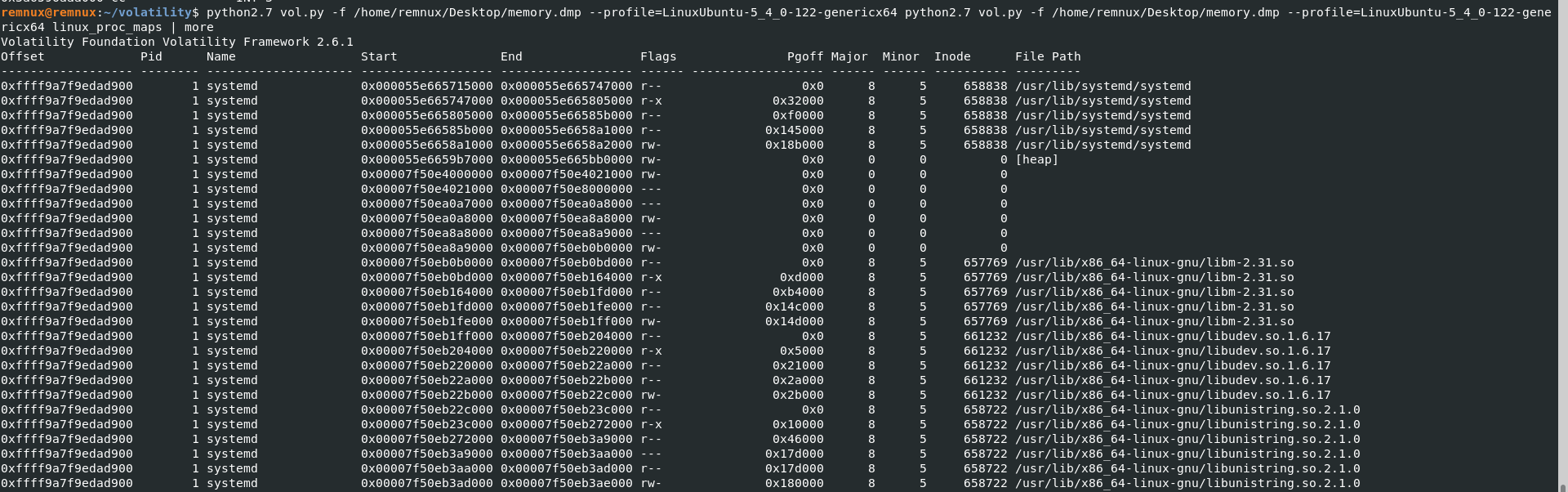
**In-Depth System Analysis:** Provides detailed inspection of processes, threads, and system configurations and is generally more detailed and comprehensive than in other tools like Rekall or FTK Imager.

**Network Forensics:** Analyzes active and historical network connections and user sessions.

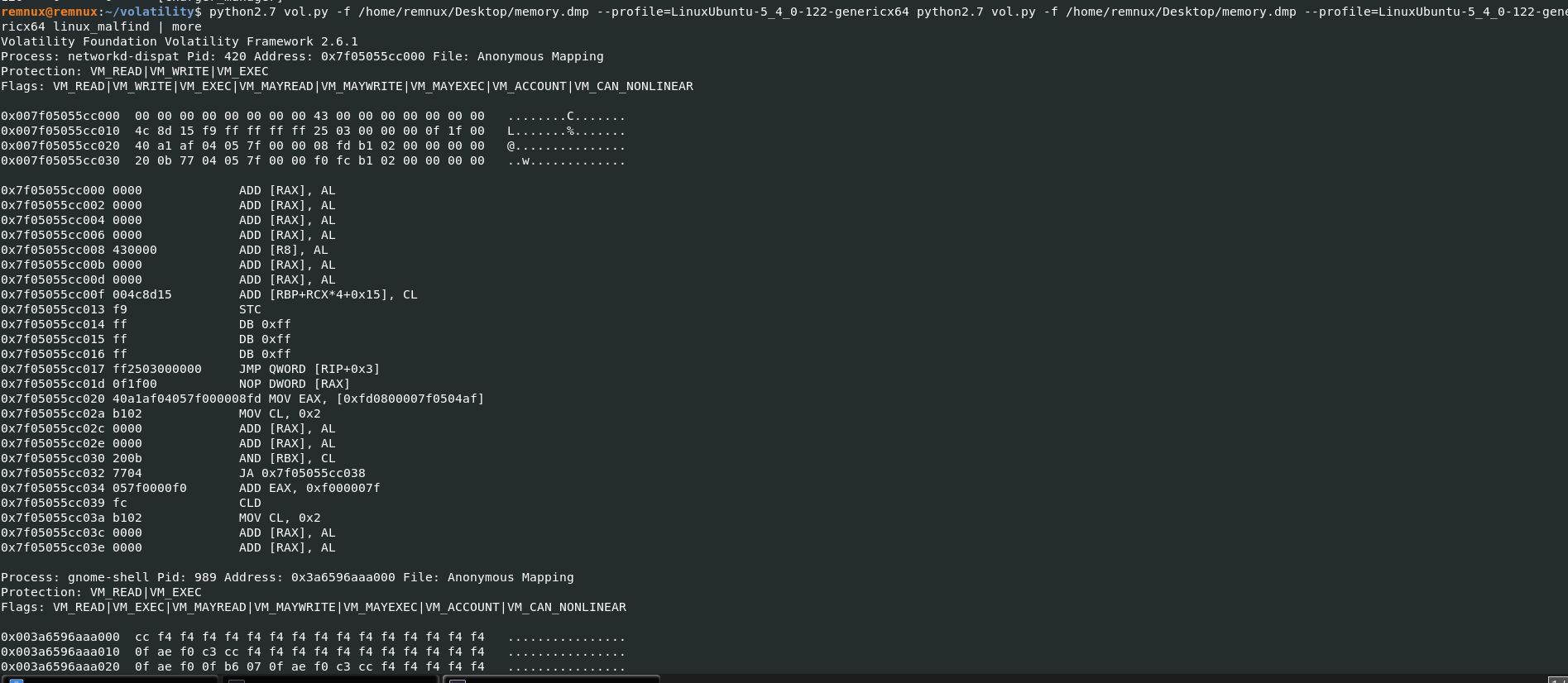
**Artifact Recovery:** Extracts files, passwords, and encryption keys from memory.

**Custom Plugins**: Allows creation and integration of custom plugins for specialized tasks.

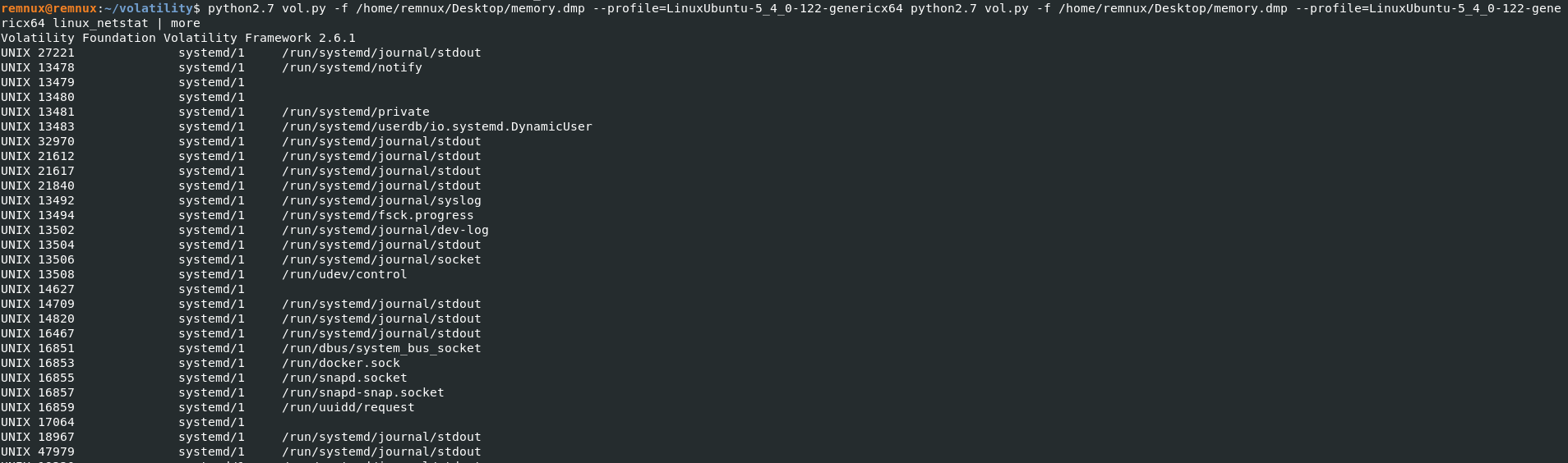
|  |  |
| --- | --- |
| Commands | Description |
| linux\_pslist | Display the list of running processes in memory. |
| linux\_bash | Recover and display bash command history from memory. |
| linux\_netstat | Show active network connections from memory analysis. |
| linux\_lsmod | Display loaded kernel modules from memory. |
| linux\_check\_afinfo | Verify address family information from memory structures. |
| linux\_dmesg | Display kernel message buffer from memory. |
| linux\_lsof | List open files for each process from memory. |
| linux\_malfind | Identify potentially malicious memory regions. |
| linux\_proc\_maps | Show memory mappings for each process. |
| linux\_psaux | List processes with full command-line arguments. |
| linux\_check\_creds | Display and check process credentials from memory. |
| ps aux or top | ps aux lists all running processes with details such as user, CPU and memory usage, and process IDs. top provides a real-time, interactive view of running processes. |
| pstree | pstree shows a tree-like view of processes, indicating parent-child relationships similar to the Volatility pstree plugin. |
| ps aux | grep -i <hidden\_process\_name> | While there isn't a direct equivalent for hidden processes (which can be more complex to detect), you can use ps aux combined with grep to find known processes. Hidden or rootkit-hiding techniques require more specialized tools like chkrootkit or rkhunter. |
| ldd <executable> | ldd lists shared libraries (similar to DLLs) used by a given executable. To list libraries for all processes, you might use a combination of ps and ldd. |
| systemctl list-units --type=service or service --status-all | systemctl list-units --type=service lists active services managed by systemd, while service --status-all lists services managed by older init systems. |
| lsof | lsof lists open files and can show open file descriptors, which is analogous to handles in Windows. |
| netstat -tulnp or ss -tulnp | netstat -tulnp or ss -tulnp lists network connections and listening ports, similar to the netscan plugin in Volatility. |
| lsmod | lsmod lists the loaded kernel modules. While it doesn’t specifically show unlinked modules, it provides information on currently loaded modules. |
| lsmod | As with ldrmodules, lsmod lists all loaded kernel modules. |
| ps aux or top with appropriate user filters | To identify processes with elevated privileges, you can use ps aux to see which processes are running as root or with specific capabilities. For detailed privilege information, ps -eo pid,uid,euid,suid,cmd can be useful. |
| dmesg | used to check for kernel messages and functions. |
| grep | search through configuration files. |
| ps -eLf | ps -eLf lists threads and their details, similar to what thrdscan provides in Volatility. |



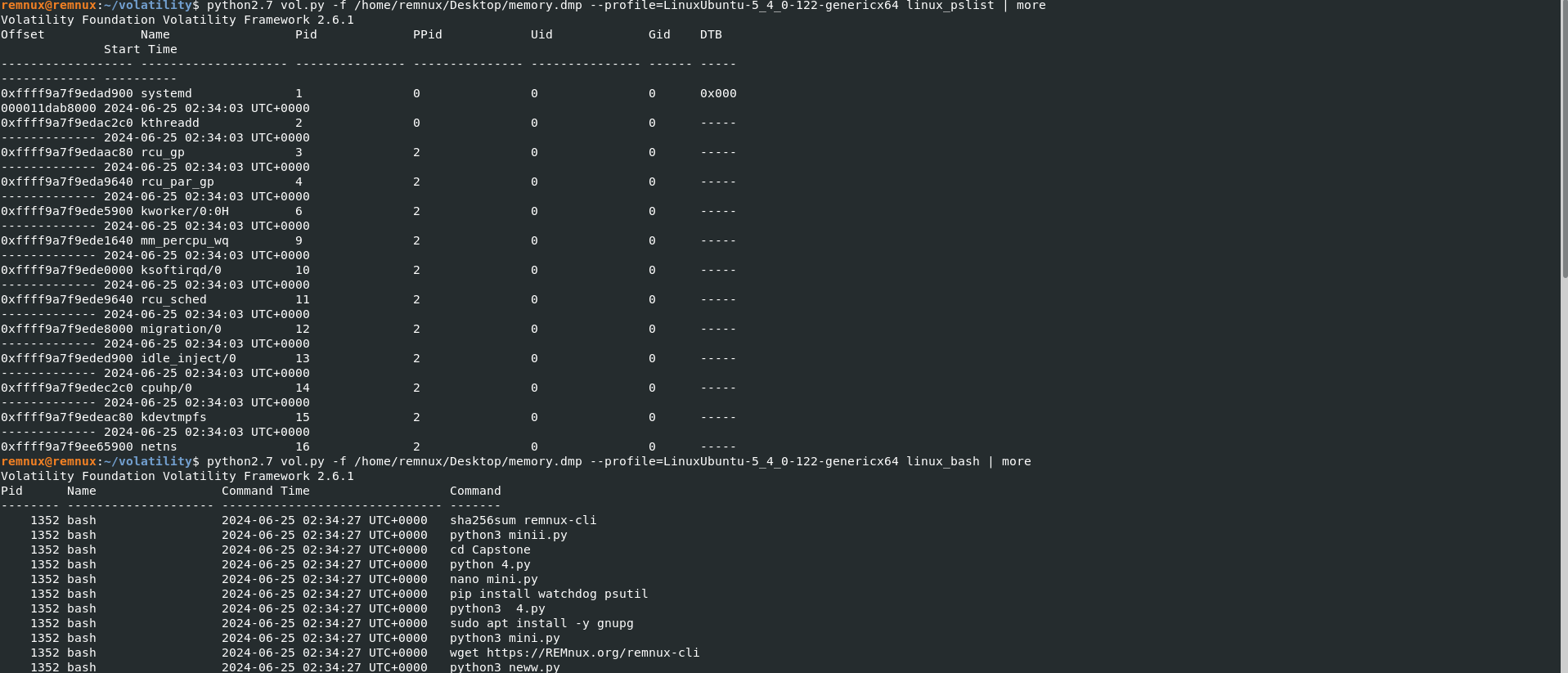
1. output of command **linux\_proc\_maps**



1. output of command **linux\_malfind**



(c) output of command **linux\_netstat**



(e) output of command **linux\_pslist , linux\_bash**

**DATASET**

**FILE BASED MALWARE:**  
<https://github.com/HynekPetrak/malware-jail/tree/master/malware>

**VirusTotal Java Script Sample files**

**Feature selection and Conversion into Feature Vector(Done Manually by running scripts in Malware jail sandbox )**

After analysing all the samples we had, we created a dataset in CSV format having 3074 rows with the features mentioned as shown in Figure 3. This dataset includes numerical values for the "Total bytes allocated" feature, while the rest of the features have binary values: 0 for absent and 1 for present. Each row is labelled as either malware or benign.

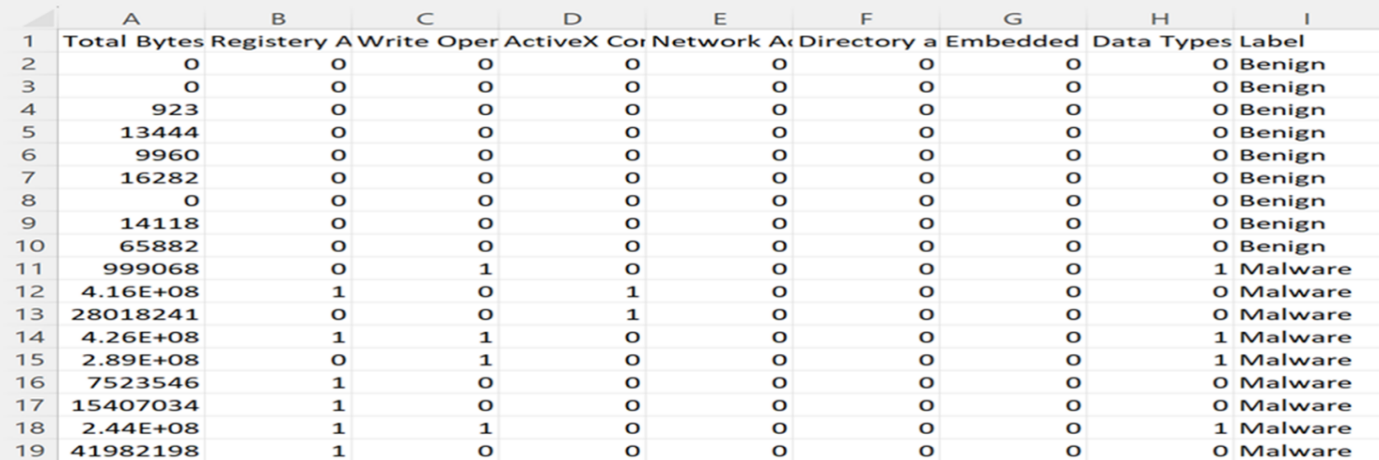


Figure 4.Snapshot of the dataset

**FILE LESS MALWARE:**

**<https://github.com/rpgeeganage/file-less-ransomware-demo>**

**ENVIRONMENT SETUP STEPS:**

1. Install VirtualBox (version 7.0.10): Download and install VirtualBox 7.0.10

2. Download REMnux VM:REMnux is based on Ubuntu 20.04 (x86/amd64).

3. Set Up REMnux VM:Import the REMnux OVA or create a new VM in VirtualBox, specifying Ubuntu 20.04 as the OS type.Allocate necessary resources (RAM, CPU, storage) as required by REMnux.

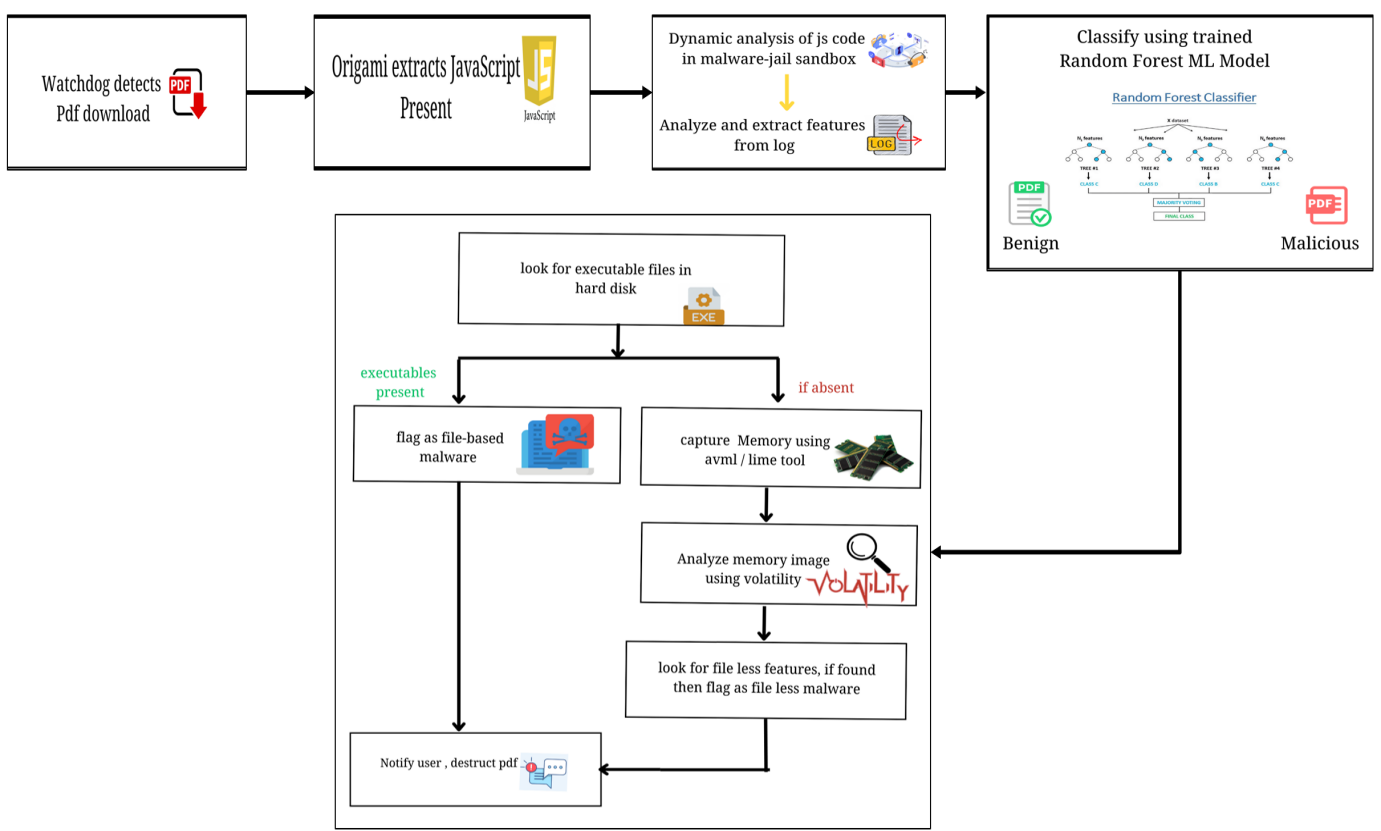
4. Install Required Tools:Use REMnux’s pre-installed tools for malware analysis or install additional ones:Install Node.js version 6.6.0 for the malware-jail sandbox:Install Malware-jail sandbox version 0.19.Watchdog for monitoring PDF downloads.Install Origami gem version 2.1.0 for extracting JavaScript from PDF files:

5. Install Python Versions:Ensure both Python 3.8.10 and Python 2.7 are installed in the VM to support various tools and frameworks.Set up virtual environments to handle both Python versions if needed.

6. Install Machine Learning Libraries: In Python 3.8.10, install `scikit-learn` and other required libraries for the Random Forest classifier.

7. Set Up Memory Analysis:Install and configure AVML or LiME for memory capture.Install Volatility 2.6.1 (compatible with Python 2.7) for analyzing memory dumps and detecting file-less malware.

**ARCHITECTURE**

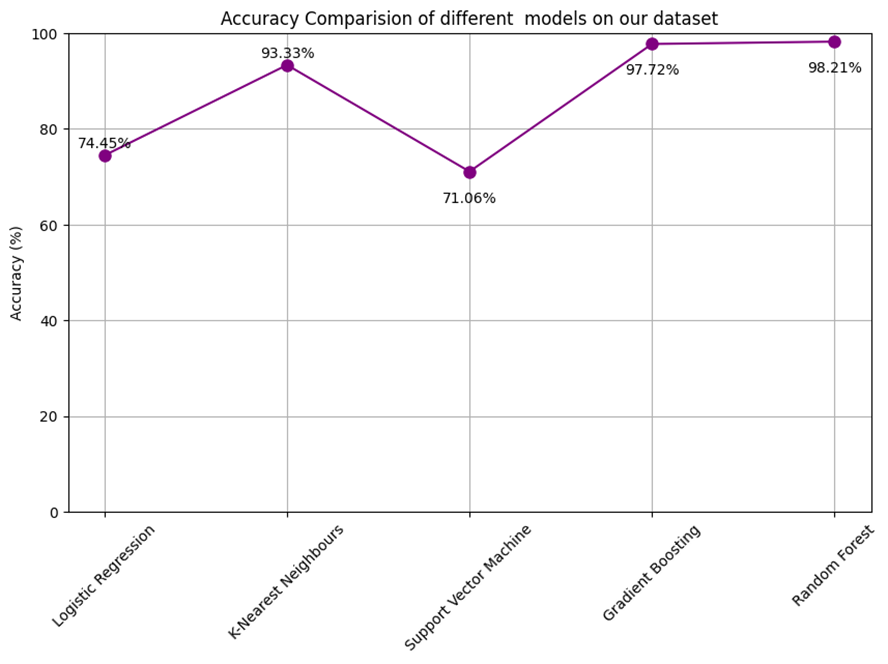
****System Architecture

**AI ML CONCEPTS**

A Random Forest is a machine learning model that combines many decision trees to make more accurate and stable predictions. Instead of just one decision tree, a Random Forest builds many trees and combines their predictions. Each tree is trained on different parts of the data and different features. The final prediction is based on the majority vote of all the tree’s outputs.

Advantages:

* By combining the results of many trees, Random Forest gives more accurate predictions.
* It can handle datasets with many features and complex patterns between those features.
* It can handle binary data particularly well because they can easily split data based on binary features.
* It tells us which features are most important in making predictions, which helps in understanding the model.

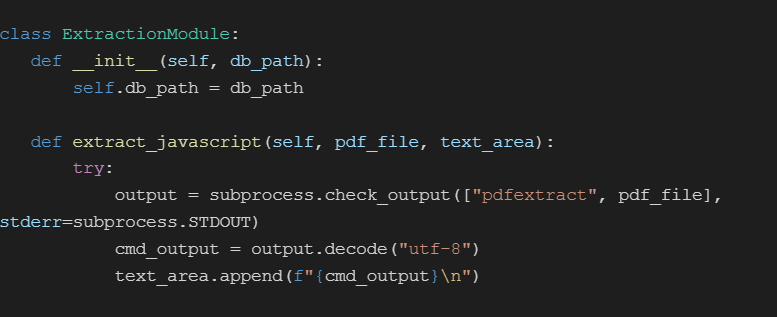


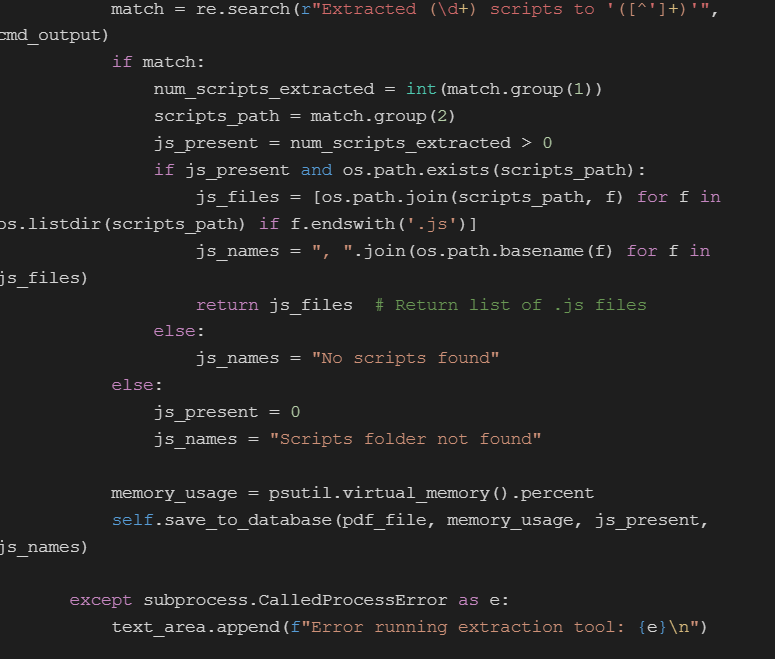
Accuracy Comparison of different models on our dataset

**Implementation”**

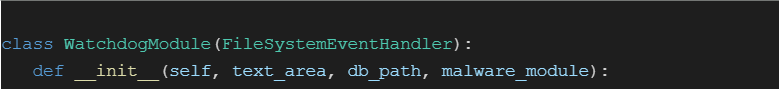
**CODE Snippets**

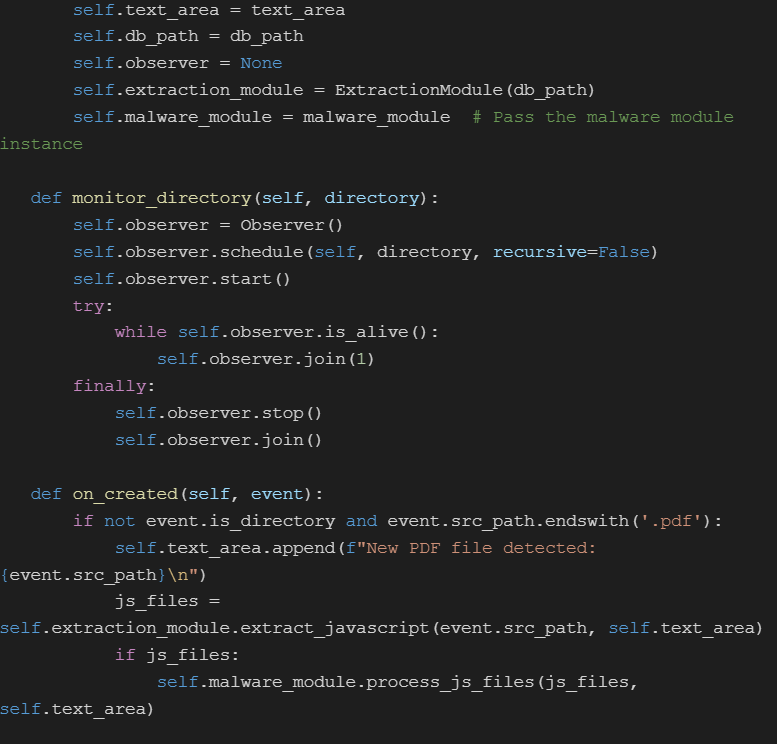
**1.Origami JavaScript Extraction Code Snippet:**

****

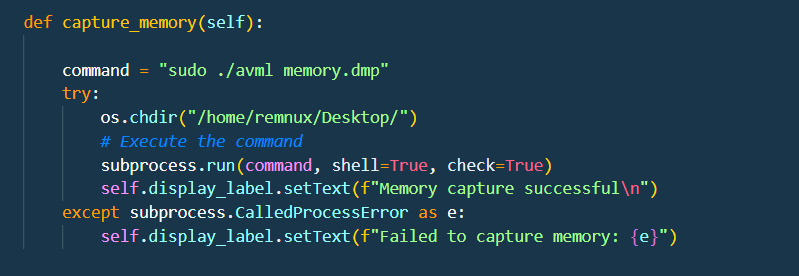
****

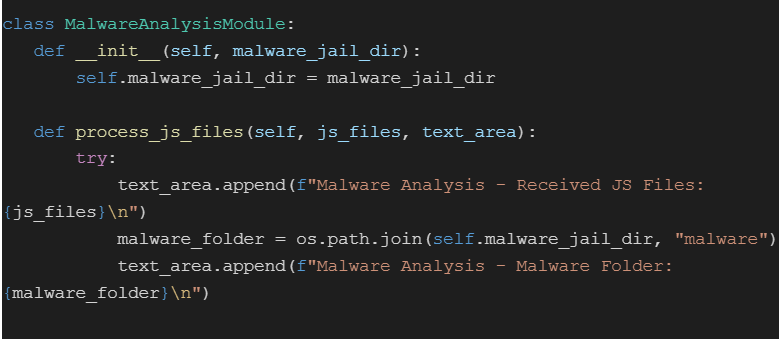
**2.WatchDog Module:**

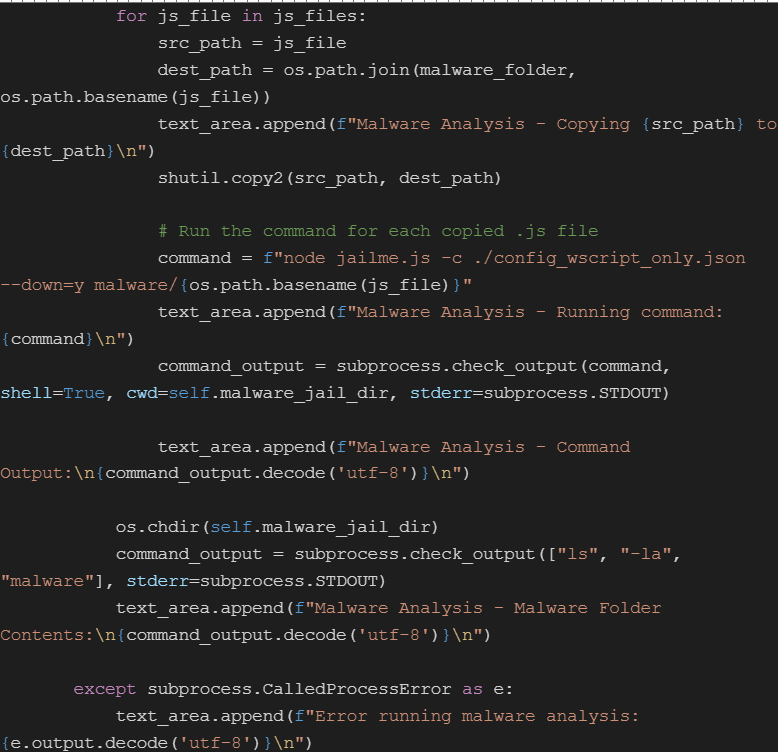
****

****

**3.AVML code for memory capture**

****

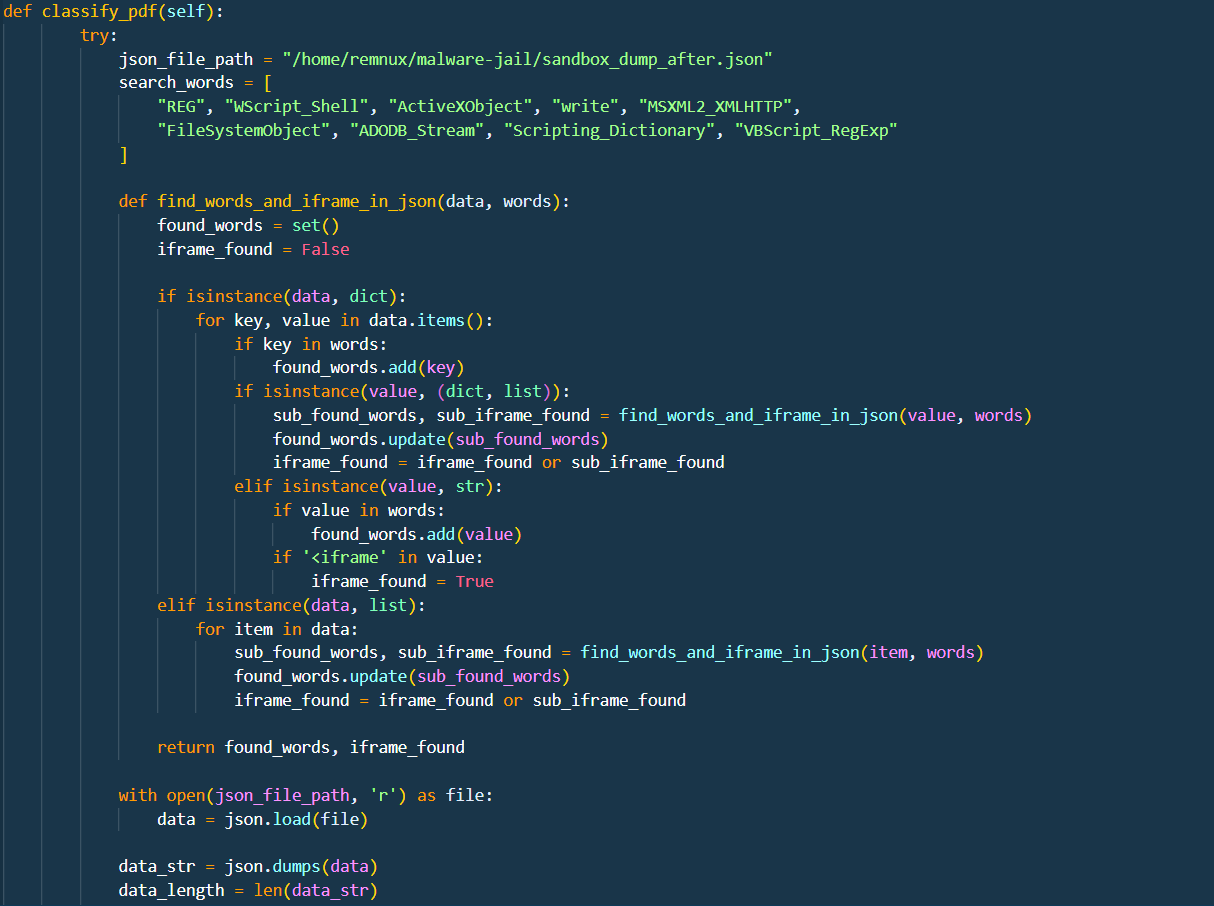
**4.Malware Analysis Module: **

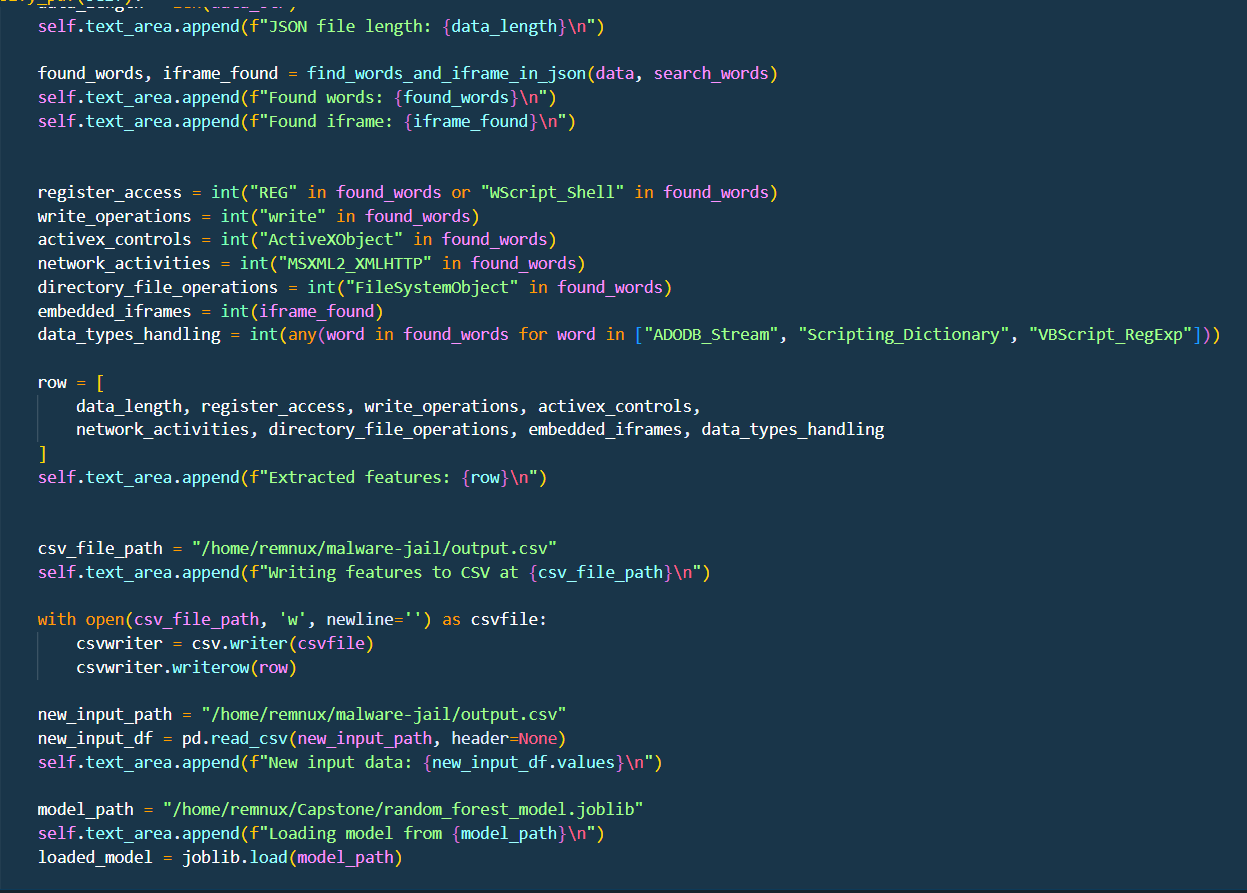
****

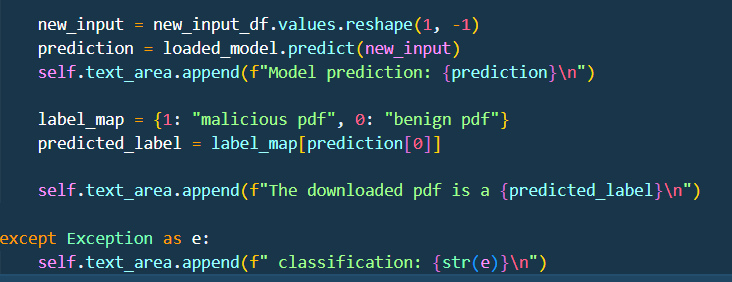
**5.Analysis using volatility**

****

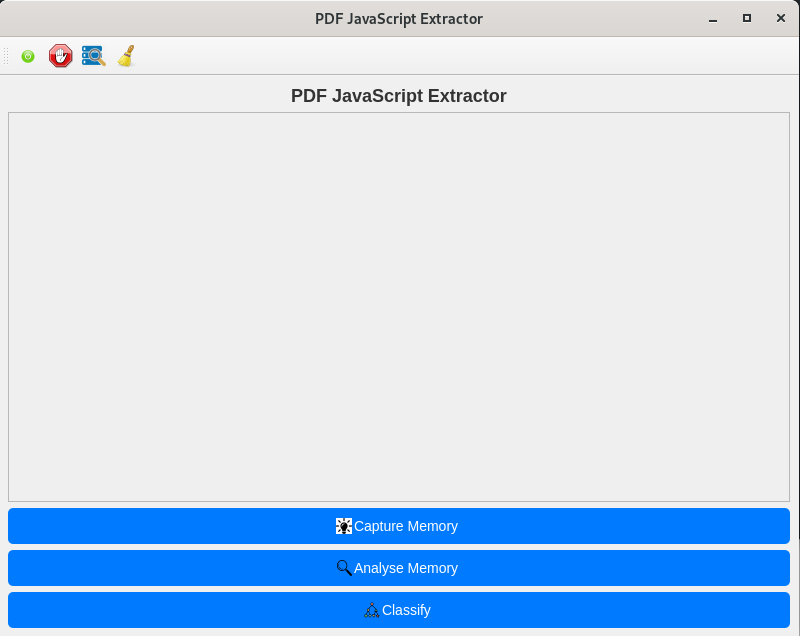
****

**6.PDF Classification**

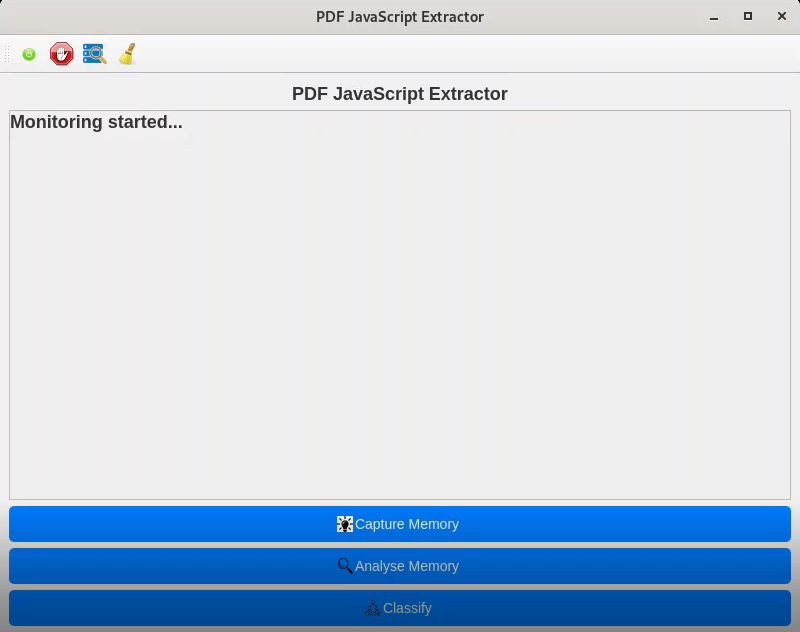
****

****

**RESULTS AND DISCUSSION:**

****

**Home Page**



**Monitoring using watchdog**

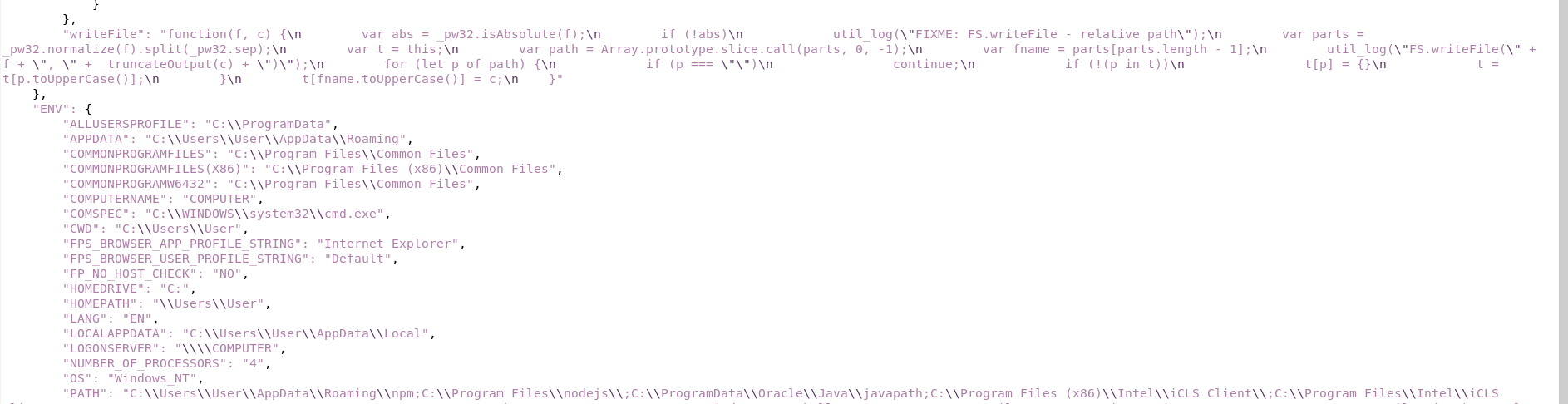


**Detecting and extracting JavaScript code in PDF**

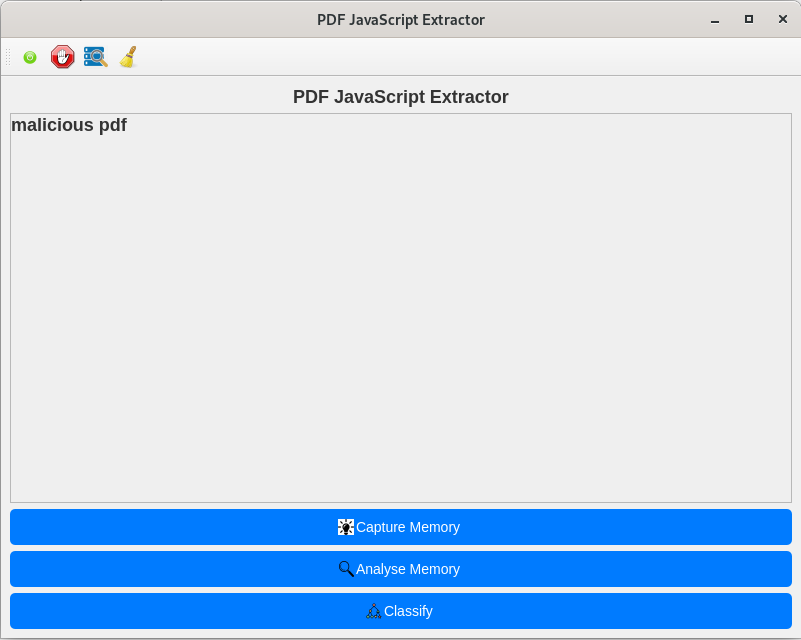


**Log contents after execution of script**

The proposed system has a user interface which is built using python module called PyQt5. Figure 7 shows the home page of our system. Figure 9 depicts how the PDFs are scanned by Origami and scripts are extracted from it, if present. Figure 10 and Figure 11 shows the results of malicious PDF detection.



**Part of dump file of malware jail sandbox showing the write operation that is  
being performed by extracted malicious script .**

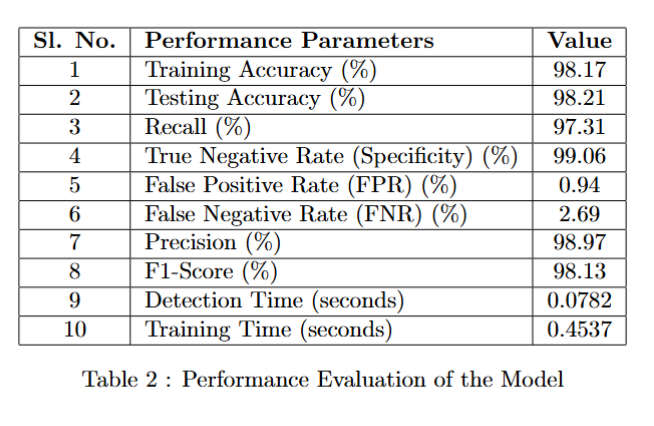


**Result for malicious PDF file**

**Result for benign PDF file**

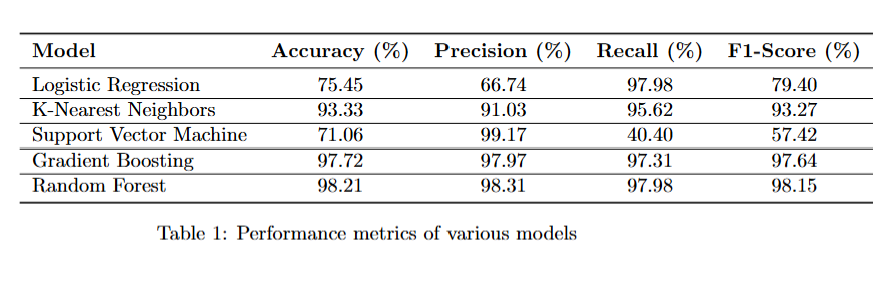
Our system classifies a PDF file as malicious or benign with an accuracy of 98.21%. The performance of our model is shown in Table 1.

Table 1: Performance Evaluation of the Model

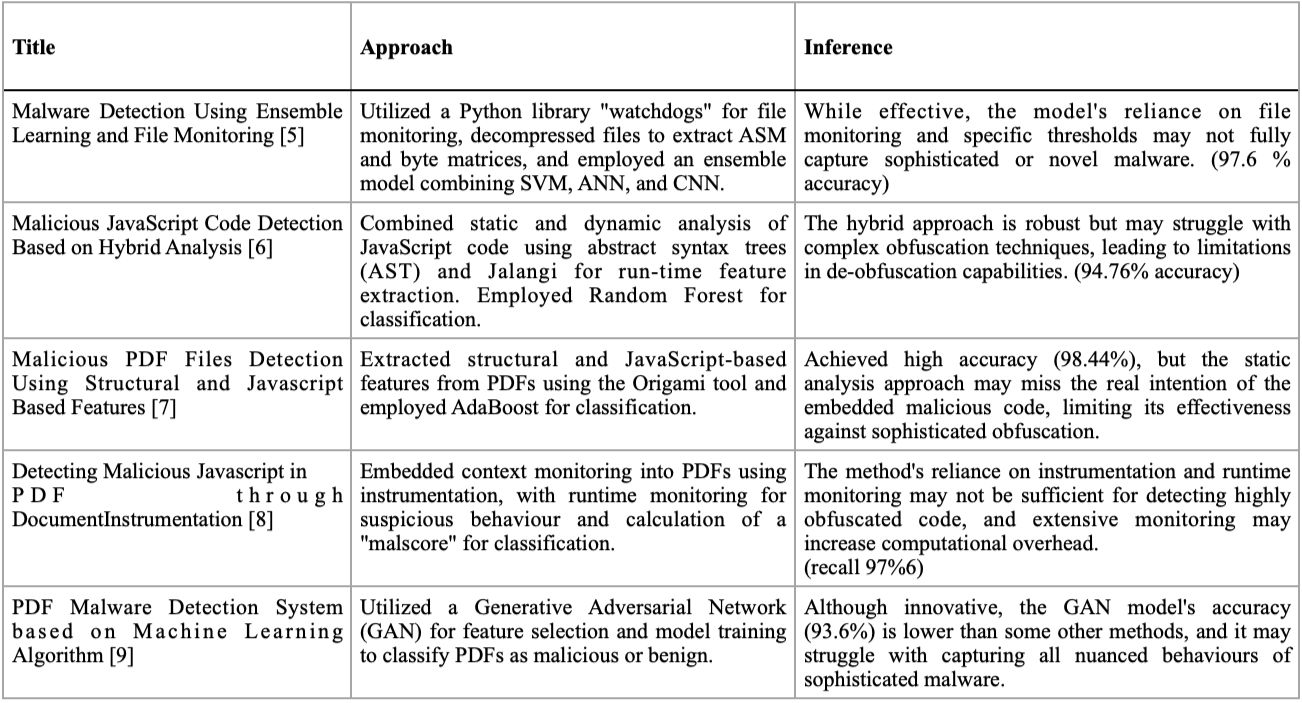


The detection time taken by our model is 0.07 seconds, which ensures it can be implemented in real-life scenarios. We accomplished higher precision and recall rates, 98.7% and 97.31% respectively. Table 2 depicts that our approached model, which is Random Forest, suppresses the performance of all other models when trained on our dataset.

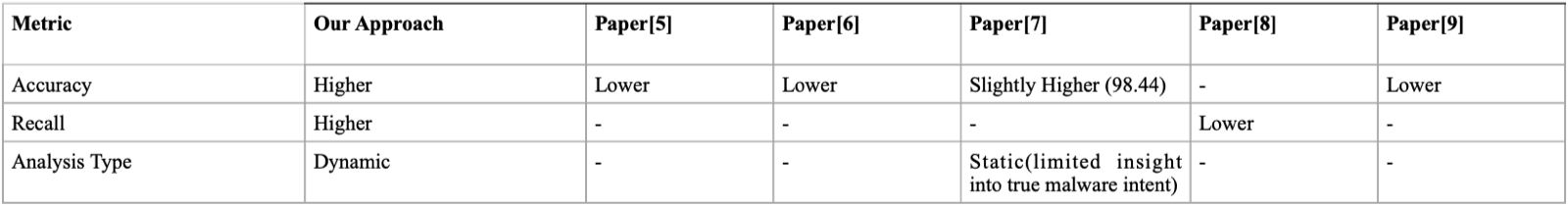
Table 2: Performance metrics of various models

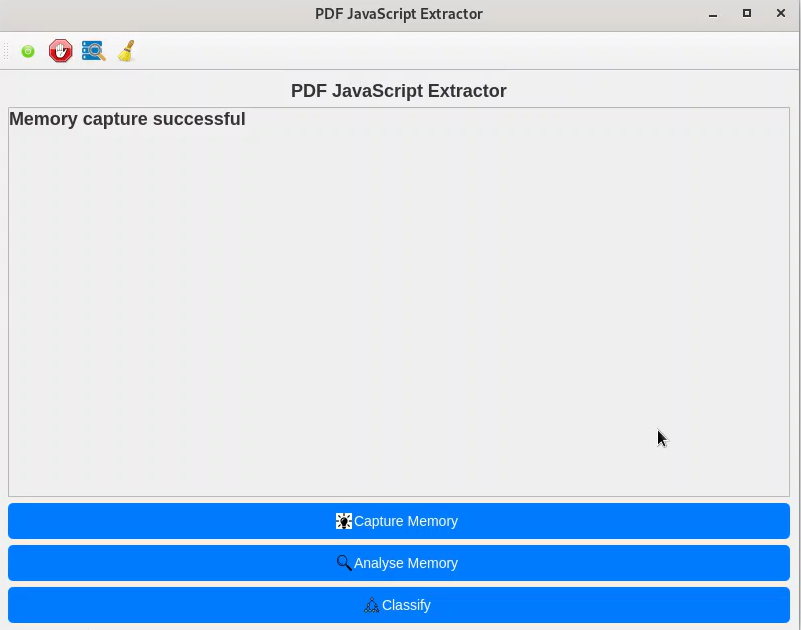


**Table 3: Depicts the other existing approaches**

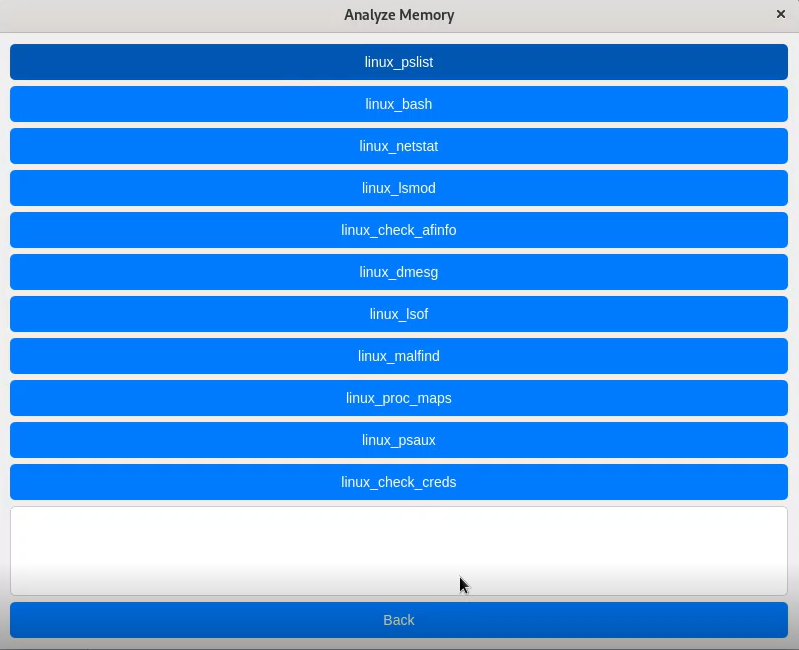
****

**Table 4: Comparison of the other approaches with our approach for classifying malicious JavaScripts**





**Capturing memory using AVML tool**

****

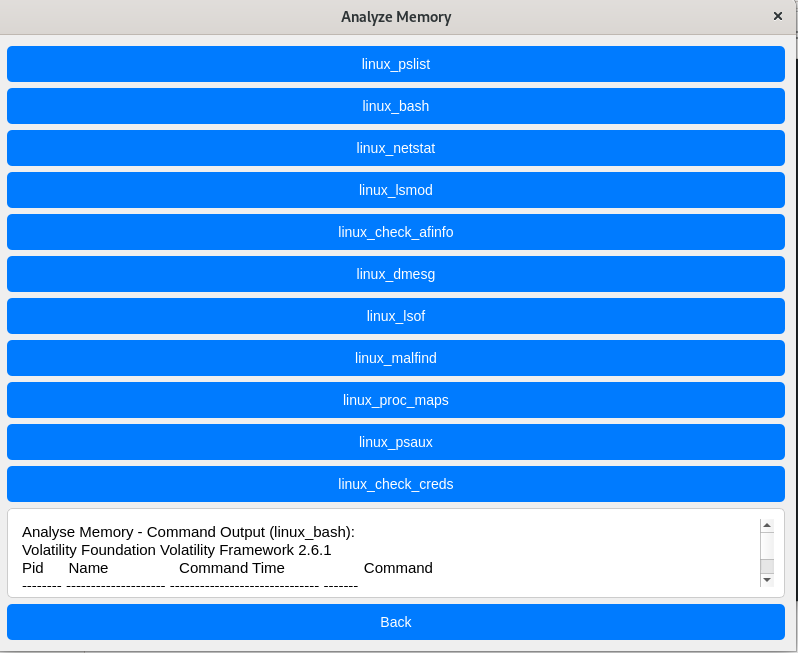
**Analyzing memory image using Volatility plugins**

**Following are output screenshots of Volatility**

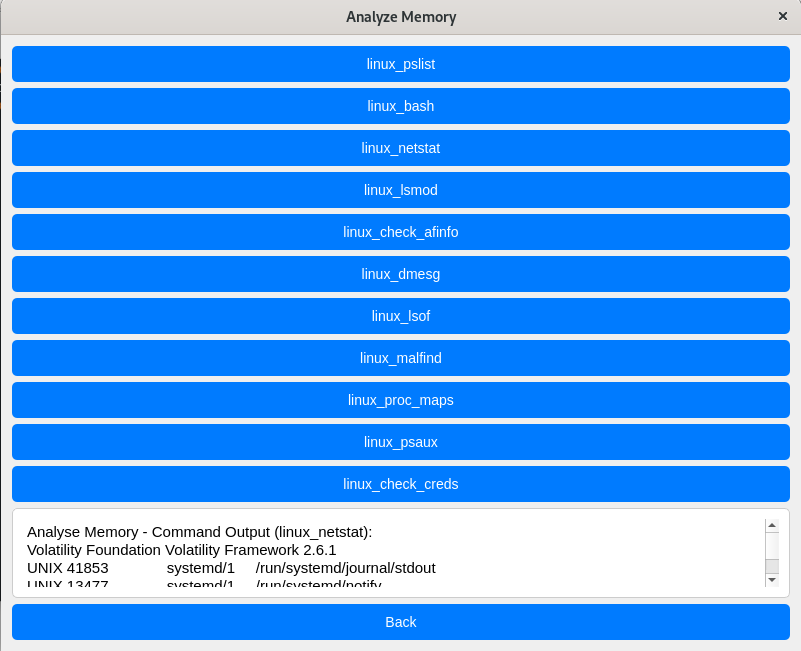
1. **linux\_pslist**

****

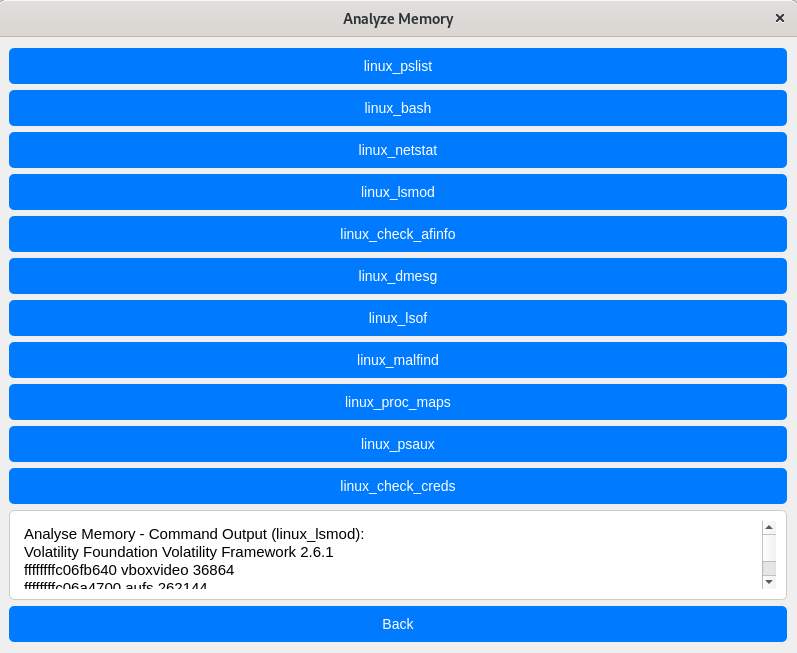
1. **linux\_bash**

****

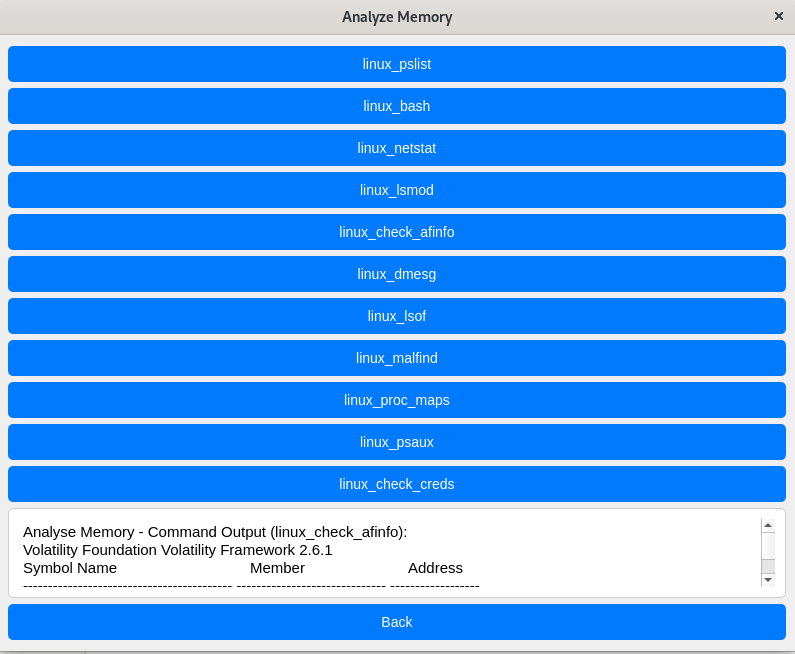
**© linux\_netstat**

****

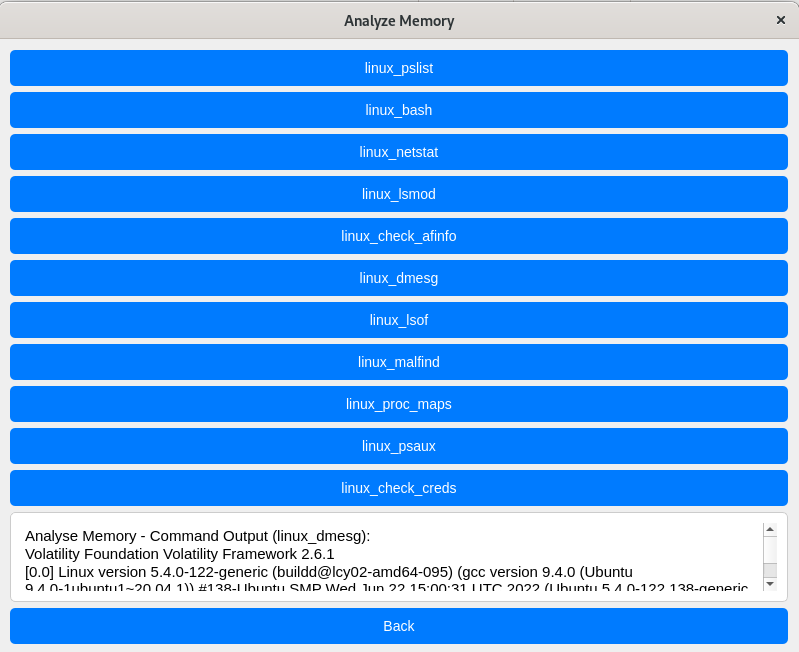
**(d)linux\_lsmod**

****

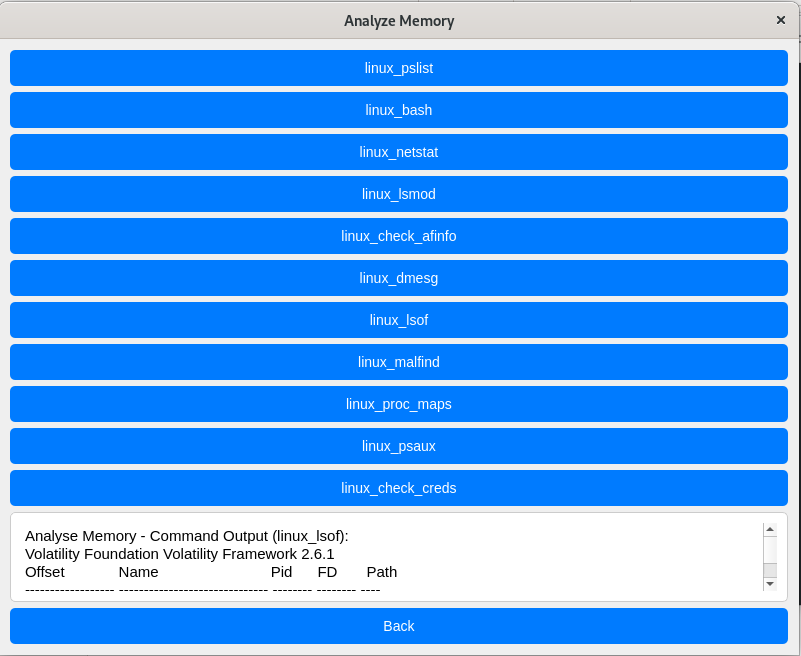
**(e) linux\_check\_afinfo**

****

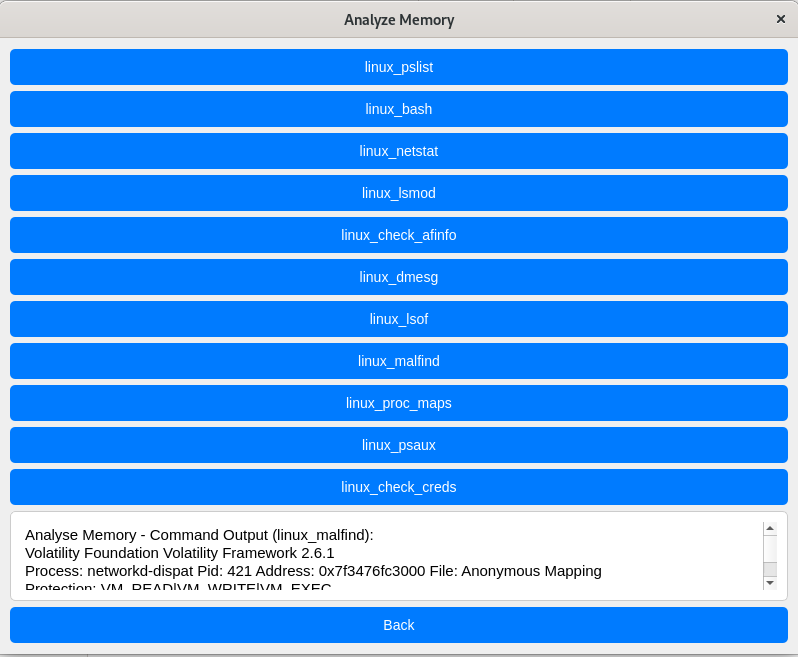
**(f) linux\_dmesg**

****

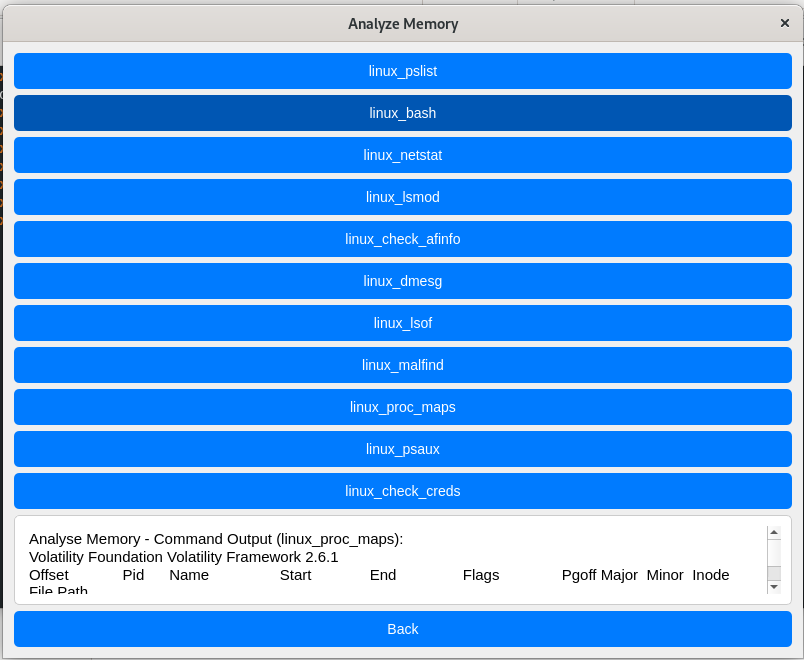
**(g) linux\_lsof**

****

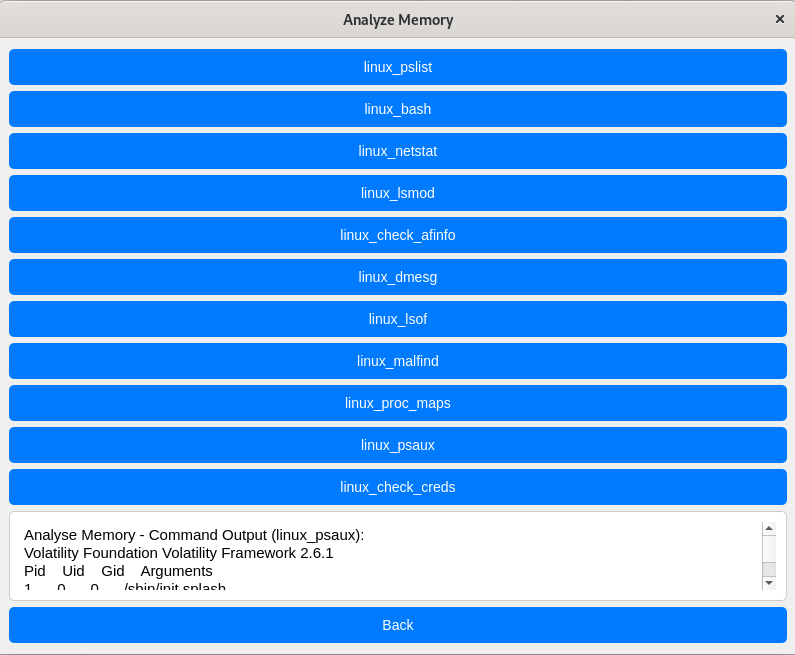
**(h) linux\_malfind**

****

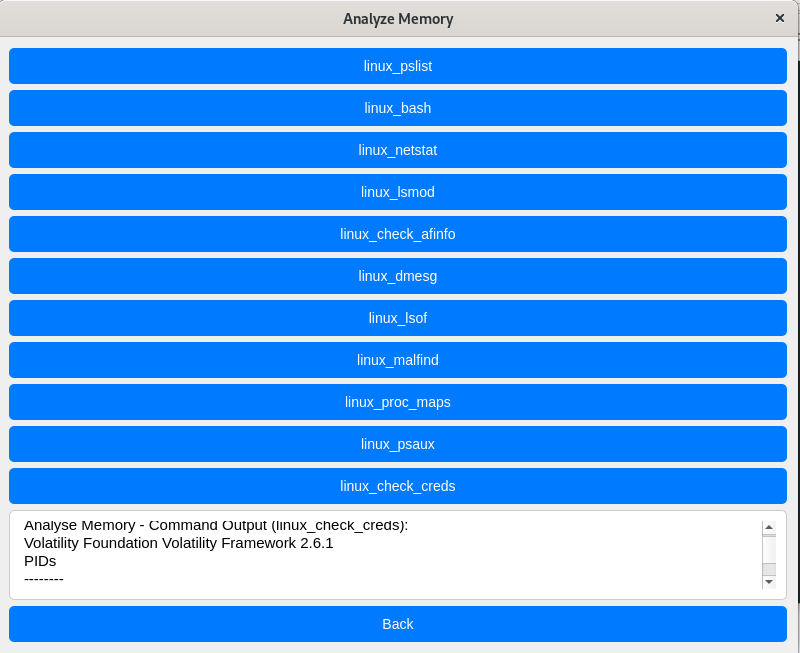
**(i) linux\_proc\_maps**

****

**(j) linux\_psaux**

****

**(k) linux\_check\_creds**

****

**CONCLUSION AND FUTURE WORK**

We proposed a malicious JavaScript embedded PDF detection method using a dynamic analysis approach. The system built looks for PDFs being downloaded and extracts JavaScript code embedded in it, if present. We used malware-jail sandbox to dynamically analyse the behaviour of the extracted code whether it is in the obfuscated form or de-obfuscated form. Based on the analysis, we defined features to build a model that can identify if the script is malicious or not. Accordingly, we are able to classify the downloaded PDF as malicious, if the malicious script is detected in it; otherwise, it is considered as benign. We trained our model using random forest and achieved higher accuracy of 98.21% and higher recall of 97.31%.

Future work will focus on expanding the system to analyse additional file types like Word documents and ZIP files, and handling of multiple files simultaneously. Also, will enhance memory analysis capabilities to detect more sophisticated file-less malware and improve the user interface for better usability. Additionally, we will develop cross-platform compatibility. Complete Automation of response actions will be introduced to minimize manual intervention and improve system efficiency.

**NOVEL ASPECTS**

1. Use of Watchdog to Trigger Extraction Module

It continuously monitors for PDF downloads ensuring real-time detection without manual intervention.

2. JavaScript Extraction from PDFs Using Origami

It is used for extracting JavaScript code embedded in PDF files allowing further analysis to detect if the script is malicious.

3. Dynamic Analysis of JavaScript code in Malware-jail sandbox

The extracted code is executed in a controlled environment, helping detect malicious actions that may not be evident in static code.

4. Handling Almost All Kinds of JavaScript Obfuscation

Our approach handles various types of obfuscation, ensuring that heavily disguised JavaScript can be properly analysed.

5. Feature Analysis from Logs of Sandbox and Feature Vector Creation

The system logs activity from the sandbox and is used to create a feature vector which is used by machine learning models for detection and classification.

6. Automated Memory Capture and Detection of Fileless Malware

When the PDF is suspected to be malicious, the system automatically captures a snapshot of memory and analyses it. This helps in detecting fileless malware, which doesn’t leave a footprint on the disk and operates solely in memory.

7. Integrated Static, Dynamic, and Memory Forensics Techniques

The combination of static, dynamic and memory forensic analysis approaches increases the chances of detecting complex malware.

References:

[1] P. Kishore, S. K. Barisal, and D. P. Mohapatra, “Javascript malware behavior analysis and detection using sandbox assisted ensemble model,” in 2020 IEEE Region 10 Conference (TENCON), Osaka, Japan, November 16–19 2020.

[2] S M Aminur Rahman , “Efficient Real-Time Fileless Malware Detection Tool For Macro-Based Microsoft Office Documents”, University of the West of England, Bristol, UK, Journal Contribution Posted on March 05,2023.

[3] I. Kara, “Fileless malware threats: Recent advances, analysis approach through memory forensics and research challenges,” Expert Systems with Applications, vol. 214, no. 6, pp. 119–133, October 2022.

[4] P. Borana, V. Sihag, G. Choudhary, M. Vardhan, and P. Singh, "An Assistive Tool for Fileless Malware Detection," in Proceedings of the Conference, Sardar Patel University of Police, Security and Criminal Justice, Jodhpur, VIT Bhopal University, Bhopal, and National Institute of Technology, Raipur, India, Aug. 2021.

[5] T. Vignesh, S. Reddy, S. Kumar, A. Chourey and C. P. Chavan, "Malware Detection Using Ensemble Learning and File Monitoring," 2023 2nd International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN), Villupuram, India, 2023, pp. 1-6, doi: 10.1109/ICSTSN57873.2023.10151567.

[6] X. He, L. Xu and C. Cha, "Malicious JavaScript Code Detection Based on Hybrid Analysis," 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Nara, Japan, 2018, pp. 365-374, doi: 10.1109/APSEC.2018.00051.

[7] Dabral, Sonal & Agarwal, Amit & Mahajan, Manish & Kumar, Sachin. (2017). Malicious PDF Files Detection Using Structural and Javascript Based Features. 10.1007/978-981-10-6544-6\_14.

[8] D. Liu, H. Wang and A. Stavrou, "Detecting Malicious Javascript in PDF through Document Instrumentation," 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Atlanta, GA, USA, 2014, pp. 100-111, doi: 10.1109/DSN.2014.92.

[9] P. P. P M and H. P, "PDF Malware Detection System based on Machine Learning Algorithm," 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2022, pp. 538-542, doi: 10.1109/ICACRS55517.2022.10029209.