European Journal of Engineering and Technology Research
Vol 8 | Issue 5 | October 2023
ISSN 2736-576X

RESEARCH ARTICLE

Classifying Memory Based Injections using Machine Learning

Doddagadduvalli Prasanna Amogh[1,*], Boraiah Ramesh[1], Rajanahally Jayakumar Bhuvan[1], Prasad Yash Vardhan[2], and Anil Apekshith[1]

ABSTRACT

This research paper explores the application of machine learning techniques to classify memory-based injection attacks. By leveraging process list data, the study focuses on distinguishing between injected and non-injected processes. Through feature engineering and training a machine learning model, the research aims to enable accurate identification of memory injection, aiding in proactive threat detection and mitigating the risk of malicious activities in computer systems.

Keywords: Dynamic link libraries, K-Nearest Neighbors, Memory injection, Random Forest.

Submitted: June 28, 2023
Published: October 30, 2023

10.24018/ejeng.2023.8.5.3077

[1] *Malnad College of Engineering, India.*
[2] *National Institute of Technology, India.*

* ***Corresponding Author:***
e-mail: amoghdp6@gmail.com

1. Introduction

With the rapid advancement of cyber threats, detecting and mitigating sophisticated malware attacks has become a critical challenge for modern security systems. One prevalent technique employed by attackers is memory injection, which involves the malicious insertion of code into a running process's memory space. Memory injection attacks evade traditional signature-based detection methods and exploit the trust between processes, making them particularly challenging to detect.

1.1. *How to Find Malicious Code or Malware?*

1) *Malicious code running in its own process*
2) *Malicious code injected into an actual legitimate process*

When a machine gets infected never turn off the memory, the data will be lost since the malware tries to hide itself and is running on the volatile memory. So, when setting up the malware lab, we will isolate the instance from the host machine and take an image of the memory instance by saving the whole .raw file for the live system and the .vmem file from the Virtual Machine, we can use WinPmem tool to extract the raw file which will later be used to analyze with Volatility tool on Linux machine/OSx.

Inside a running process, we can find whether the process is accessing a file, registry key, DLLs, etc. Networking connections show the live connections from the Virtual machine to the outside world, it can also be the Host machine, so it is recommended to isolate the test environment.

Memory-based injection attacks pose a significant threat to computer systems, when it comes to memory injections, the attacks are mainly focused on volatile memory. It lives in a computer's RAM memory. The main attackers are trying to avoid files being written to disk is that most of the anti-virus software/security software programs concentrate on scanning for malicious files and artifacts written on the disk, on the other hand, memory-based attacks are engineered to bypass a system's security software and also some sophisticated metamorphic malware/viruses remain unnoticed because the best way to remain in the dark is to just evade the security system or act like the system.

Machine learning algorithms have emerged as powerful tools for enhancing malware detection capabilities. ML techniques leverage the power of data analysis and pattern recognition to identify and classify malicious activities. The application of ML in memory injection detection holds great promise in providing proactive defense mechanisms against these elusive attacks making it a viable option in an ever-evolving malware ecosystem.

Our research paper focuses on the detection of memory injection attacks using ML techniques. The primary objective is to explore and evaluate the effectiveness of two ML algorithms in detecting and classifying processes that are injected from those that are not injected. Using ML methods, we aim to develop robust and proactive approaches to improve the security of systems against these sophisticated memory injection attacks.

The paper will present a comprehensive review of existing literature and state-of-the-art techniques related to memory injection detection and ML-based malware detection in general. We will examine different ML algorithms, such as random forests, and K-nearest neighbors, in the context of memory injection detection. Furthermore, we will explore the relevant features and memory access patterns that can be utilized for effective feature engineering and model training.

To evaluate the performance of the proposed ML-based memory injection detection approach, we will employ real-world datasets containing both normal and injected process instances. We will compare and analyze the accuracy, precision, recall, and F1 scores achieved by the two different ML algorithms. Hopefully, the findings from this research will contribute to the advancement of memory injection detection techniques and provide valuable insights into the effectiveness of ML algorithms for enhancing system security. By identifying and mitigating memory injection attacks in real time, organizations can fortify their defenses and proactively safeguard critical data and resources against evolving cyber threats.

2. Literature Survey

2.1. *An Architectural Approach to Preventing Code Injection Attacks*

This research paper [1] proposes a systematic approach to mitigate code injection attacks. The research focuses on designing and implementing a secure architectural framework that includes security measures such as input validation, secure coding practices, and runtime monitoring. By adopting this architectural approach, organizations can strengthen their systems against code injection vulnerabilities and significantly reduce the risk of successful attacks, enhancing overall system security. his approach offers proactive defense against vulnerabilities and strengthens overall system resilience. However, successful implementation relies on careful adherence to the recommended security measures, considering the system's specific architecture and constraints. Additionally, the impact on system performance and overhead should be considered when adopting the proposed approach.

2.2. *DeepLog: Anomaly Detection and Diagnosis from SystemLogs through Deep Learning*

This research paper [2] introduces DeepLog, a system that leverages deep learning techniques for anomaly detection and diagnosis from system logs. By analyzing log data using deep learning models, DeepLog can effectively identify abnormal patterns and provide insights into potential system issues. This approach offers a promising solution for improving system monitoring and troubleshooting, enabling organizations to proactively address anomalies and enhance system reliability and performance. It offers advantages such as automatic learning of complex patterns, real-time insights for prompt troubleshooting, and the ability to handle high-dimensional and unstructured log data. However, limitations include the dependence on quality training data, resource-intensive computations, and the interpretability of deep learning models. Overall, DeepLog presents a valuable approach for enhancing system monitoring and troubleshooting, but careful consideration of these limitations is necessary for effective implementation.

2.3. *Ten Process Injection Techniques: A Technical Survey of Common and Trending Process Injection Techniques*

This survey blog [3] provides an overview of ten common and trending process injection techniques used in the field of cybersecurity. It aims to offer technical insights into various methods employed for injecting code into processes, enhancing the understanding of process injection vulnerabilities and their potential mitigation. Some commonly used techniques include DLL Injection, which allows arbitrary code to be injected stealthily but can be detected through DLL loading activities. Code Cave Injection utilizes unused memory regions for flexibility but requires finding suitable code caves. APC Injection leverages asynchronous procedure calls for evasion but can be hindered by APC monitoring. AtomBombing enables high-privileged code execution, bypassing security mechanisms, but demands administrative privileges. Process Doppelganging manipulates process images to evade detection achieve stealth¨ but can be detected by process creation monitoring. TLS Callback Injection offers covert injection through TLS callbacks but may not be applicable in all processes. Reflective DLL Injection avoids file-based detection by loading DLLs from memory but relies on existing malicious code. Process Hollowing replaces a target process's memory space to evade detection but requires administrative privileges and may introduce stability issues. Early Bird Injection exploits process initialization but relies on precise timing. Thread Execution Hijacking redirects thread execution flow for flexibility but demands in-depth knowledge of the target process. This survey provides insights into the strengths and weaknesses of these process injection techniques, aiding in understanding their effectiveness and potential mitigation.

2.4. *CODDLE: Code-Injection Detection with Deep Learning*

This research paper [4] proposes a deep learning-based approach for detecting code injection attacks. By leveraging deep learning techniques, CODDLE aims to identify and classify instances of code injection in real time. The method demonstrates promising results in accurately detecting code injection attacks, offering improved security against such threats. However, the effectiveness of CODDLE may depend on the availability and quality of training data, as well as the complexity of the code injection techniques employed. Further research and experimentation are necessary to validate and refine the performance of CODDLE in different real-world scenarios. While CODDLE focuses specifically on code injection attacks, DeepLog aims to detect anomalies from system logs. CODDLE utilizes deep learning models to identify and classify instances of code injection in real time, offering improved security against such attacks. On the other hand, DeepLog uses deep learning algorithms to analyze system logs and identify abnormal patterns indicative of anomalous behavior or potential security breaches. Both approaches demonstrate the potential of deep learning in detecting and diagnosing security threats. They rely on large amounts of training data to train their models effectively. However, CODDLE's effectiveness may be influenced by the availability and quality of training data for code injection attacks, while DeepLog's performance may depend on the comprehensiveness and accuracy of the system logs.

2.5. *A Basic Malware Analysis Process Based on FireEyeEcosystem*

The paper [5] explores the detection of malicious processes based on memory injection techniques using machine learning. The authors address the increasing threat of memory injection-based attacks, which involve injecting malicious code into the memory space of legitimate processes. Such attacks can evade traditional antivirus systems and pose a significant challenge to security analysts. The paper focuses on developing a machine learning-based approach for identifying and classifying processes affected by memory injection.

The study utilizes a dataset of process information obtained from a controlled environment. It includes both normal processes and processes infected with various memory injection techniques. The authors employ several feature extraction techniques to capture meaningful information from the process data, including the number of modules, imported functions, and memory regions.

They evaluate the performance of different machine learning algorithms, such as Decision Trees, Random Forests, K-nearest neighbor (KNN), and Gradient Boosting, using metrics like accuracy, precision, recall, and F1-score. Additionally, they compare the results with traditional signature-based antivirus systems to assess the effectiveness of the proposed approach.

The findings indicate that machine learning algorithms, particularly Random Forests, and KNN, outperform signature based methods in terms of detection accuracy. The models achieve high precision and recall rates, demonstrating their capability to identify malicious processes affected by memory injection techniques. The authors highlight the significance of feature engineering in improving classification performance.

The paper concludes that machine learning techniques show promise in detecting memory injection-based attacks. The proposed approach provides a proactive and dynamic defense mechanism against these types of threats. However, the authors acknowledge the need for further research to handle the evolving nature of attacks and the challenges of real-time detection.

2.6. *A Dynamic Windows Malware Detection and Prediction Method Based on Contextual Understanding of API Call Sequence*

The authors of this paper [6] address the challenge of detecting memory injection attacks, which involve injecting malicious code into the memory space of legitimate processes. These attacks are designed to evade traditional security measures and can be used to carry out various malicious activities. The paper focuses on utilizing machine learning techniques to detect and classify memory injection attacks effectively.

The study proposes a framework that combines static and dynamic analysis to extract features from the process memory. Static analysis involves examining the characteristics of the code, while dynamic analysis monitors the behavior of the process during runtime. The authors extract features such as the entropy of memory regions, system call sequences, and API function calls to capture the behavior of the process.

They evaluate the performance of various machine learning algorithms, including Random Forest, and K-Nearest neighbor (KNN). The evaluation is based on different datasets that contain both normal and injected processes. The metrics used for evaluation include accuracy, precision, recall, and F1-score.

The results demonstrate the effectiveness of the proposed approach in detecting memory injection attacks. The machine learning models achieve high accuracy rates and outperform traditional signature-based methods. The authors highlight the importance of feature selection and the combination of static and dynamic analysis for accurate detection.

3. Objectives

3.1. *Data Collection*

In this step, we collect data from a local Windows machine, as the primary target of memory injection attacks are Windows systems. We create a new virtual machine and then populate it with some basic applications that might be targeted for memory injection attacks. Then we use a shell script to collect the process list data which includes the memory used by the process and the DLL files loaded by it. This is first done for a clean process where the memory injection is not done and then the same thing is done again for the injected process. This is how data is collected from a single system.

3.2. *Data Preprocessing*

In the dataset preprocessing phase, the raw data will be transformed into a suitable format for machine learning analysis. This involves removing irrelevant data, handling missing values and outliers, and applying feature engineering techniques to extract meaningful features. Additionally, data normalization or scaling methods will be used to ensure comparable scales across features. The preprocessing step aims to prepare the dataset for accurate classification of memory injection techniques using machine learning algorithms.

3.3. *Model Selection*

After preprocessing the dataset, the next step is to select a suitable machine learning algorithm for the task of classifying memory injection techniques. Two commonly used algorithms for classification tasks are Random Forest and K-Nearest neighbor (KNN).

Random Forest is an ensemble learning algorithm that combines multiple decision trees to enhance classification accuracy. It works by constructing a multitude of decision trees and aggregating their predictions to make the final classification. Random Forest is robust to noise and can handle high dimensional data effectively.

On the other hand, K-Nearest Neighbors (KNN) can be a suitable model for memory injection classification due to its ability to capture non-linear relationships and create localized decision boundaries. In the context of memory injection detection, where complex patterns may exist, KNN's flexibility in adapting to local data characteristics can be advantageous. Additionally, KNN provides interpretability by associating predictions with neighboring instances, aiding in understanding classification decisions. Its easy implementation and robustness to irrelevant features make it a convenient choice for quickly building a baseline model without extensive feature engineering. The choice between Random Forest and KNN depends on the specific characteristics of the dataset and the requirements of the classification task. Random Forest is suitable when there are multiple features and potential interactions among them. It is also useful when dealing with noisy or missing data. KNN, on the other hand, is effective in handling high-dimensional data and can generalize well even with a smaller dataset as the need for a neighbor can be as less as 1.

3.4. Feature Engineering

Extract meaningful features from the process list that can help the machine learning algorithm distinguish between injected and non-injected processes. The way this will be done in this project will be by using the used memory part of the extracted data list and the loaded DLL files. Both will be used to derive a meaningful relation and thus help the machine learning model to distinguish between injected and non-injected processes.

3.5. Model Evaluation

During the model evaluation phase, the performance of the trained model is assessed using metrics such as accuracy, precision, recall, and F1-score. Accuracy measures the overall correctness of the model's predictions, while precision and recall evaluate its ability to correctly classify positive instances and detect true positives, respectively. The F1 score combines precision and recall into a single measure to provide a balanced evaluation. By analyzing these metrics, researchers can gain insights into the model's strengths and weaknesses, identify areas for improvement, and assess its suitability for classifying memory injection techniques.

4. Machine Learning Method

4.1. Creating a Safe and Effective Analysis Environment

Malware is software that is explicitly designed to perform some tasks without the knowledge of the user using the system. This means that it is generally a bad idea to let malware run on the same PC on which you send e-mails to your friends, do your online banking, and write papers for security conferences. One way to solve this problem is to isolate the environment where the malware would be executed. Isolation should be both on the storage level and network level. At the storage level, the malware could have the potential to spread itself to other partitions without our knowledge, and at the network level, it might try and connect to the internet, and try and access the host machine from which the environment is supposed to be isolated. Although the malware is written by us and is unable to carry out above said actions, it is always better to be vigilant when dealing with malware. These machines should have a standardized software build that can easily be restored from a backup image after some piece of malware has finished destroying the system. One such solution would be to use virtual machines to isolate the execution environment. There are several software products that can be used to create virtual machines. Virtual Box from Oracle is currently our favorite for malware analysis by virtue of its ability to create a tree of snapshots that capture system state at various times and are free to use. These snapshots can be used to easily revert to a previous system state where the malware was not injected at all and try and redo the whole injection if things go south.

We have used a virtual box to create 2 new isolated environments. We made sure that both these systems were deployed on an external hard disk rather than an internal one. The first system was a Windows machine with Windows 10 64-bit installed, and around 50 GB of storage allocated for its entire use. This is the target machine where the Memory injection will happen. The second system was a 64-bit Kali Linux with 40 GB of allocated memory. This is the machine that serves the requests generated by the Windows machine, if any, and acts as the attacking machine. We create an isolated network containing only these two systems and that is not connected to the internet. For this, we created a new Virtual Box Host-only Ethernet adapter with the Kali Linux server serving all the requests sent by the Windows machine. This helps in isolating the environment from the host machine network and thus from the internet. To send back responses for the requests sent by the Windows machine we use INetSim, which basically responds to every request with a fake response page. This makes sure that the malware will get some response instead of getting nothing, which might cause the malware to stop executing. Basically, this enables the malware to think that it is still connected to the internet and continue its execution as designed whilst keeping the network isolated as well (see Fig. 1).

4.2. The Malware

Although there were many malwares available that got the job done, we wrote our own malware that simply does what is needed by us and nothing else. We used C++ to write the malware with just a main function, as there is not much the malware will be doing. First, the malware opens the process that needs to be injected. This process can be written into the script where the script loads all the currently available processes on the machine using the 'CreateToolhelp32Snapshot', from which the required process could be selected. We on the other hand used Process Hacker 2 in order to get the process ID of the process and manually type it into the malware's codebase. This was done to ensure that the malware is fully under our control and does not go rogue. Also using Process Hacker 2 we can continuously monitor the behavior of the process and see the effect of the memory injection in real-time. Next, the malware allocates the

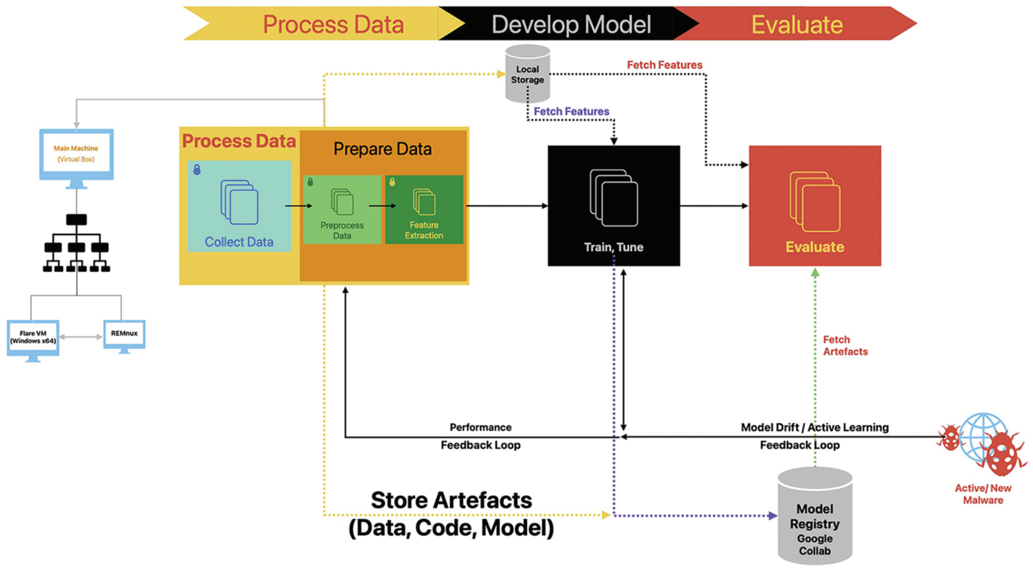

Fig. 1. High-level design.

memory to the process using the 'VirtualAllocEx' function, this allocates memory within the target process. You can increase the allocated memory by specifying a larger size. Once the memory is allocated to the process the malware copies the desired payload into the memory allocated. This is done using the 'WriteProcessMemory' function. The payload will be injected into the target process and executed. The payload in our case was simply a shell code for a reverse TCP connection to the Kali Linux machine. This shell code was generated using the Metasploit Framework. We also used 'LoadLibrary' to load more DLLs and adjust the process memory. Finally, we execute the payload shell code using the 'CreateRemoteThread' function, which creates a thread to execute the code. Once all of the mentioned steps are done, we release all the resources and terminate the thread. This is then converted into an executable and executed on the target Windows machine.

4.3. Data Extraction

Data for the machine learning models to learn and extract a detection pattern is to be collected from a Windows machine. We wrote a Windows power shell script that extracts and prints all the currently running processes along with their allocated memory, process ID, and loaded DLL files. We also added a target value named memory injection which is initially set to False for all the processes. First, the script is executed with the processes such as Notepad, Chrome, Word, and Powerpoint open, these processes will act as targets for our memory injection. Then the malware is executed with the process ID of the target processes and the shell script is run once again to gather the same parameters of the same processes. But in this case, we label the memory injection value to be True as these processes have undergone process injection. This is done through around 7 processes, and this is how data is collected from a single system.

4.4. Process Data

Processing the data is an important part of machine learning as the model should specifically feed only the data that it needs to analyze. Preprocessing our data requires several steps. In the first step, we remove all the unnecessary words that are in the collected data. This was due to the structure of our PowerShell code. In the next step of preprocessing, we add the word NULL to all the processes that do not have any of the DLL files loaded. This is just good practice. The loaded DLL files are a continuous list of paths to the DLL file being executed separated by a white space. This data is of no use so we clean it up and make a list with each of the loaded DLLs forming an element in the list. This list is used to find the percentage of DLLs loaded by the process, which is calculated by dividing the number of DLLs loaded by each process by the total number of DLLs loaded by the machine in during that session. This parameter can also be replaced by the number of DLLs loaded by the process directly, but we found taking the ratio a better and more normalized feature than just the number of DLLs. Finally, we convert data into a comma-separated variable file with the order being process name, process ID, memory usage, percentage of loaded DLLs, and injection technique. Once the processing of data is done, we can move on to the next step.

4.5. Develop Model

To implement the machine learning model to classify memory-based injection attacks using methods like Random Forest (RF) and K-Nearest Neighbors (KNN), the following algorithm is proposed.

Firstly, the data set that is formed after the preprocessing of data is used. The data set is then divided into positives and negatives. Positives contain samples that are True for memory injection and Negatives contain samples that are False for memory injection. This is basically done to have a small number of samples of processes that have been injected for testing into training due to a limited number of process data collected that were injected. Both the positives and negatives are divided into testing and training sets. The set of training sets from both positives and negatives form the final training set and similarly group of testing sets from both positives and negatives form the final testing set for the model. Feature engineering techniques are applied to extract meaningful information from the dataset, enhancing the quality of input features. Specially the names of processes were factorized in order to pass them as input for the models. Next, feature selection methods are employed to identify the most relevant features for classification. Techniques like correlation analysis, information gain, or recursive feature elimination are utilized to select the optimal subset of features that contribute most to the classification task (Fig. 2).

Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve classification accuracy. It is well-suited for handling high-dimensional data with complex interactions among features. In the context of memory injection classification, Random Forest can effectively capture patterns and relationships between various process attributes and the corresponding injection techniques. Its ability to handle noisy and missing data makes it a robust choice for this task. Although we do not have any missing data, we can say the data is noisy. Most of the data that is collected simply is useless and the Random Forest Classifier should correlate for interactions using the very little data that is present in the process dataset collected.

On the other hand, the KNN classifier is a powerful machine-learning method that excels in both classification and regression tasks. One of the key strengths of KNN lies in its simplicity and intuitive nature. It is a non-parametric algorithm, meaning it does not assume any specific data distribution and learns directly from the training examples. This flexibility allows KNN to capture complex and nonlinear relationships in the data. Additionally, KNN is considered a lazy learning algorithm, as it does not require an explicit training phase. Instead, it stores all training instances and makes predictions based on the k nearest neighbors to a given test point. This ability to leverage the local structure of the data makes KNN robust to outliers and noise. Moreover, KNN does not make strong assumptions about the data, allowing it to work well with both numerical and categorical features. Its simplicity, flexibility, and ability to handle various data types make KNN a popular choice in many real-world applications. KNN makes perfect sense for our research as the malware injected into all the processes is the same and thus will increase all the process values almost equally. This gives the classifier enough data or neighbors to classify a process into infected or noninfected easily.

Both Random Forest and KNN have their own advantages and considerations. Random Forest is known for its ability to handle complex interactions and noisy data, making it suitable when there are multiple features and potential dependencies in the dataset. KNN, on the other hand, includes its ability to capture non-linear relationships, create localized decision boundaries, and provide Interpretability.

In the classification of memory injection techniques, Random Forest can capture the relationships between various process attributes and injection methods, while KNN can create effective groups of neighbors to separate different injection techniques based on the given features. The choice between the two algorithms depends on the specific characteristics of the dataset and the desired trade-offs in terms of interpretability, computational efficiency, and performance. Subsequently, separate models are trained using RF and KNN algorithms on the training dataset. Hyperparameters of the models, such as the number of trees in RF or the number of neighbors in KNN, are configured to ensure optimal performance. The models are then evaluated using the testing dataset, employing evaluation metrics such as accuracy, precision, recall, and F1 score. Techniques like cross-validation may be applied to validate the models' robustness and prevent overfitting. As it is a major concern in Random Forest Classifier.

Based on the evaluation metrics, the performance of the RF and KNN models is compared, and the model with higher accuracy and better classification performance for memory-based injection attacks is selected.

By following this precise algorithm, we can effectively implement the machine learning part of their research paper, providing a systematic and reliable approach to classify memory-based injection attacks using Random Forest and K-Nearest neighbor. The algorithm can be changed accordingly.

4.6. *Evaluate*

Model evaluation techniques for KNN and Random Forest include train-test split, evaluation metrics such as accuracy and F1 score, feature importance analysis, stratified sampling, and cross-validation. These techniques enable the assessment of model performance, the identification of important features, and the understanding of the model's strengths and weaknesses. When evaluating the performance of machine learning models like K-Nearest Neighbors (KNN) and Random Forest, several techniques are commonly used. For KNN, one effective approach is to split the dataset into training and testing sets. The model is trained on the training set and then evaluated on the testing set to assess its performance. Evaluation metrics such as accuracy, precision, recall, and F1 score can be calculated to measure the model's classification performance.

Similarly, Random Forest models can be evaluated using the train-test split technique. The dataset is divided into training and testing sets, and the Random Forest model is trained on the training set. Subsequently, the model's predictions are compared to the actual values in the testing set to calculate evaluation metrics. Additionally, Random Forest provides a measure of feature importance, which can be examined to understand the relative contribution of different features in the model's predictions.

Other techniques for evaluating the models include stratified sampling, where the dataset is split while maintaining the distribution of target classes, and cross-validation, which involves dividing the dataset into multiple subsets called folds. These techniques provide more robust estimates of the model's performance by evaluating it on different subsets of the data. However, this cannot be tested in our dataset as the available data is very low and will yield high accuracy, but which will be mostly for the False memory injection label as that comprises most of the data.

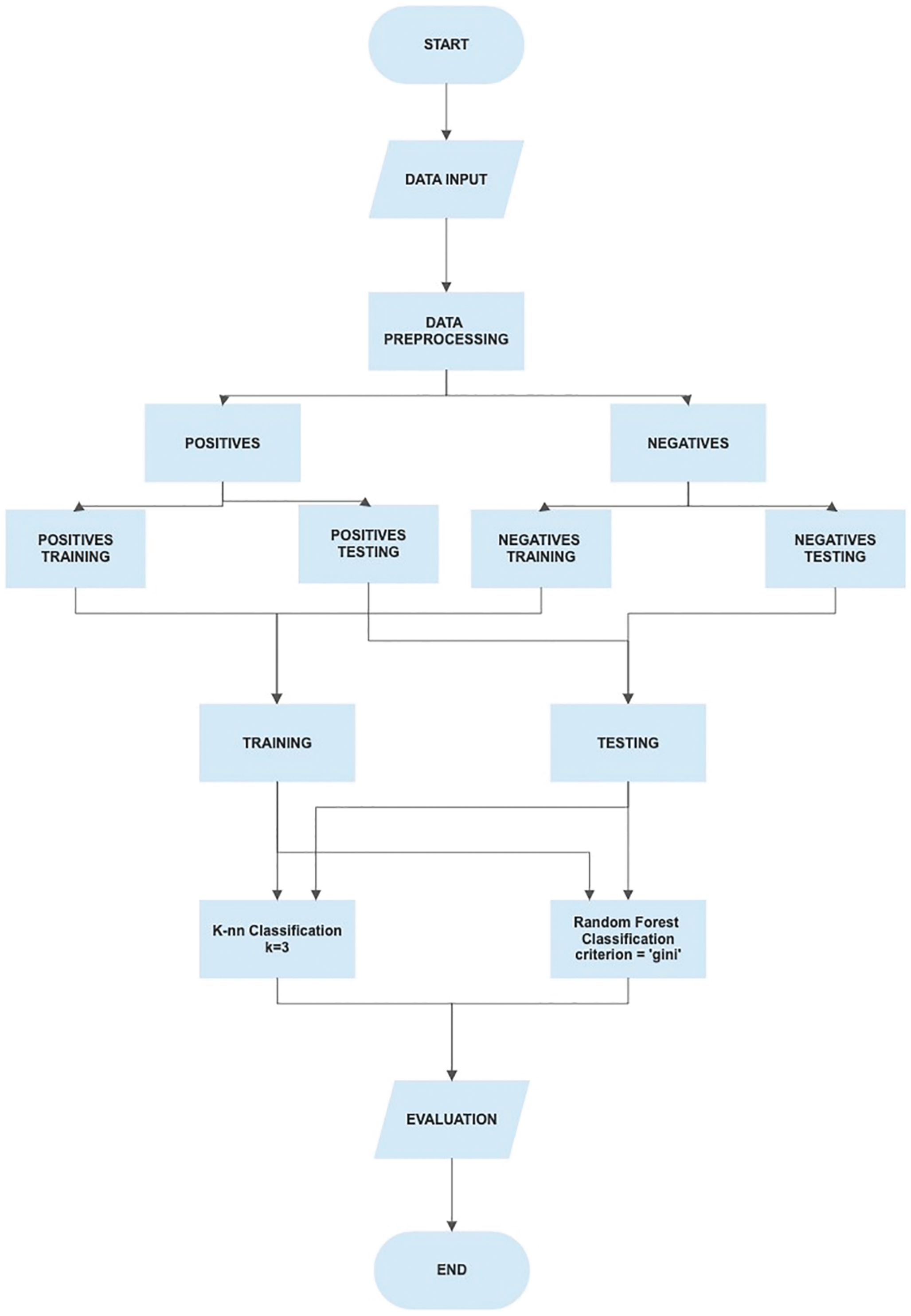

Fig. 2. Flow chart of the machine learning process.

5. Results and Discussion

The inputs for the machine learning models were the process name, memory used or allocated for the process, and the percentage of DLL files that were used by that process. This is calculated by first taking the total number of uniquely loaded DLL files by all the processes and taking the ratio of DLL files loaded by the current process and the total number of DLL files loaded in the current Windows session. Then there is the target value which says whether the process is injected or not. We gave the injected processes 1 and the ones that were not injected 0. This was done to facilitate the model to perform better. The process names were also factorized to denote them using numbers instead of text. Each of the processes was given a unique number which will be kept the same every time the process is in the process list. Using much more sophisticated scripts and libraries preprocessing was done so that the data could be used by the model to gain some valuable information.

As discussed earlier we first implemented the K-Nearest neighbor Classifier. Once the data was preprocessed it was fit into the model from sklearn, with the number of neighbors as 3. The process name, memory usage, and

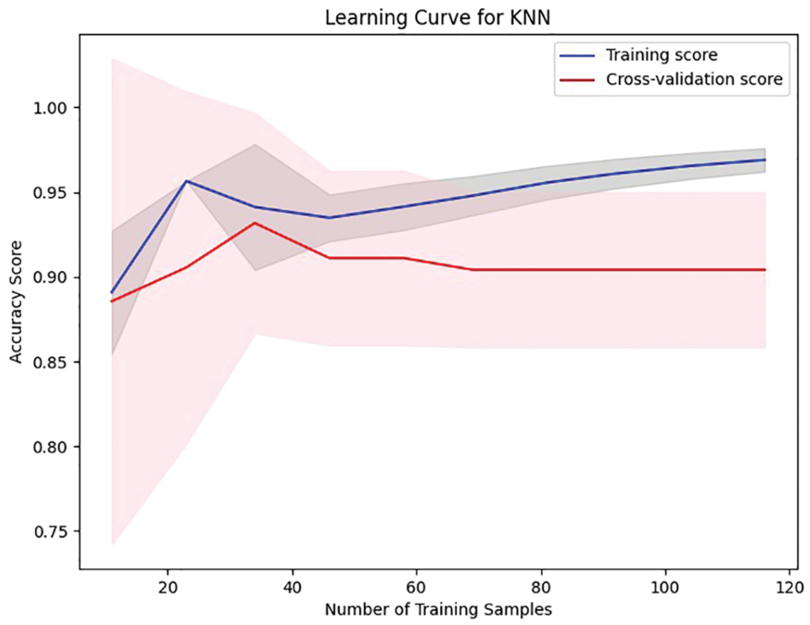

Fig. 3. Learning curve of KNN classifier.

percentage of loaded DLLs form the training data for the classifier, and the memory injection with 0 and 1 acts as the target value. Once fit the model was tested using the test set.

This resulted in a model that had an accuracy score of around 90 percent. Although the accuracy score is not a great indicator of how well the model performs having such high accuracy is expected in machine learning models. Precision represents the proportion of correctly classified positive instances out of all instances predicted as positive. A precision score of 71 percent indicates that the model achieved a reasonable level of accuracy in correctly identifying positive instances. Looking at the confusion matrix showed us that out of the 7 injected processes, the model was able to correctly identify 5 as infected. The recall score, also known as sensitivity or true positive rate, was also around 71 percent. Recall measures the ability of the model to correctly identify positive instances out of all actual positive instances. The F1 score, which combines precision and recall into a single metric, was also around 71 percent. The F1 score is a measure of the model's balance between precision and recall. A score of 71 percent implies that the model achieved a reasonable trade-off between precision and recall, resulting in a robust overall performance. The learning curve indicated in Fig. 3 shows the similarity between training score and cross-validation score.

Next, we implemented the Random Forest Classifier to compare the results with that of the KNN classifier. While implementing Random Forest Classifier the same dataset was used with the criterion being 'gini', max depth being 10, minimum samples split being 4, and random state in effect. The model was from sklearn and had the same training and target values. Once the model was fit it was cross verified using the test set. The Random Forest had an accuracy of around 91 percent it was more successful in identifying Injected processes correctly. This high accuracy indicates that the model accurately classified 92 percent of the instances in the dataset. When looking into the confusion matrix RF classifier had correctly identified 6 out of 7 infected processes at one of its best random states. But had also falsely labeled 4 of the process that was not injected as injected. As seen in Fig. 4, the training accuracy is quite high but the same cannot be inferred from the validation accuracy.

Random Forest Classifier also correctly mapped the contribution on each of the Feature vectors as shown in Fig. 5. This helped in feature selection for the KNN classifier. According to the classifier, the feature that contributed significantly to the classification was Memory usage which should be obvious as that is what is being manipulated by the malware. Next, we have the percent of DLLs loaded, the malware was coded in such a way that it could call and load for more libraries if needed so this contributing around 0.25 to the classification method is valid. The least relatively important feature is the process name.

The results shown above are average results of running the model around 15 times. This was the most general result shown by both the models. Although the results are shockingly similar in both the KNN classifier and the Random Forest classifier, the latter usually had a higher F1 score, precision, and recall in many cases with the highest being 6 identified as infected out of 7 infected processes but KNN failed to reach the same results. KNN classifier was trained by keeping the value of the neighbor as 5 and 1. With $k = 5$ the results are sub-par and $k = 1$ is not usually advised as it could easily misinterpret the results, even with such a low value the results were not as good as with that of $k = 3$. Looking at the results we can clearly say that a Random Forest classifier can easily predict whether a process is injected or not and doing that with such small data to work with is truly impressive. At the same time, the KNN classifier has done a better job of classifying the same processes. As we collect more data and reiterate the models accordingly, we believe that KNN would only get better in forming groups of infected processes and thus improving the overall process. At the same time, the

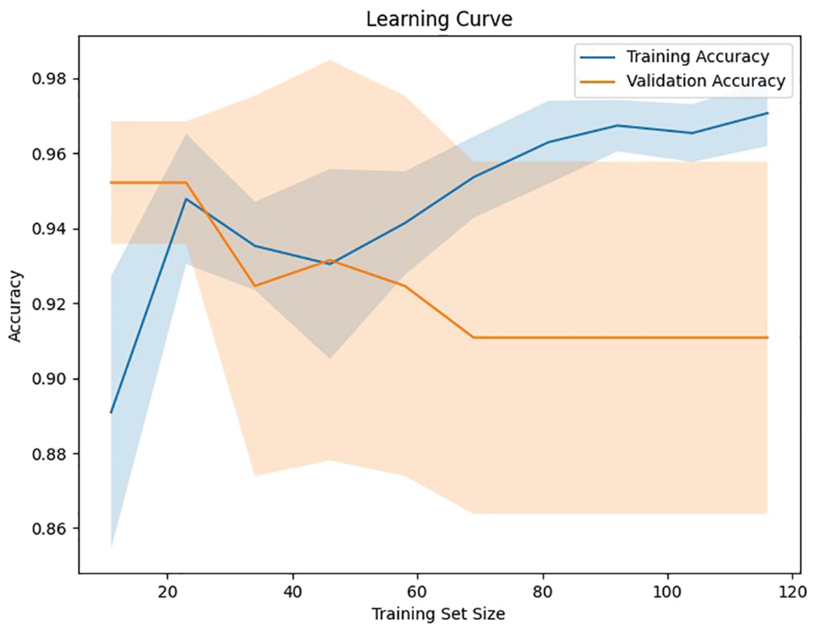

Fig. 4. Learning curve of random forest classifier.

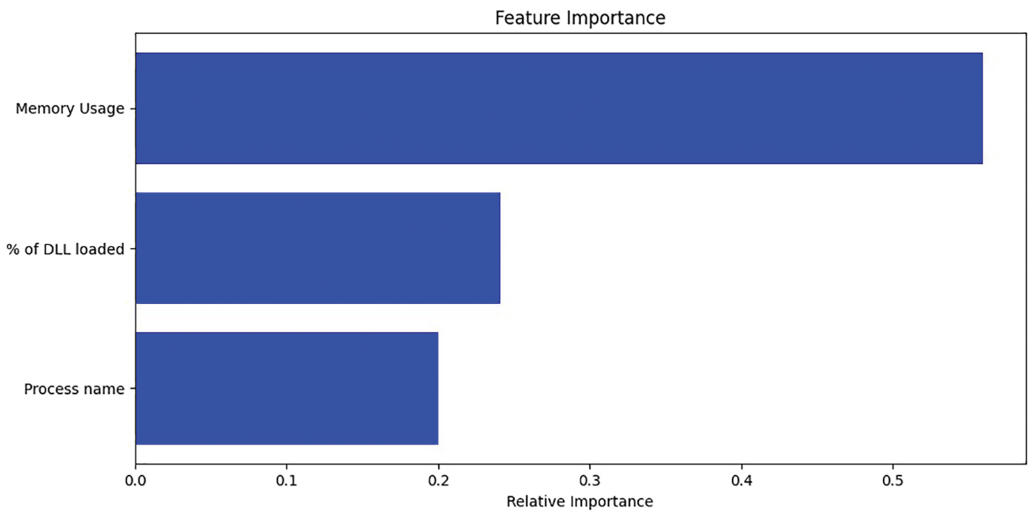

Fig. 5. Feature importance graph for random forest.

Random Forest classifier would branch out more in case of more data and its performance would only deteriorate from here on.

6. Conclusion

In conclusion, this research paper focused on the task of classifying memory-based injection in processes using machine learning techniques, specifically Random Forest (RF) and K-Nearest neighbor Classifier. The features used for classification were memory usage and the ratio of DLLs loaded. The study demonstrated the effectiveness of RF and KNN in accurately determining whether a process is injected or not. By employing these Machine Learning algorithms and leveraging feature engineering techniques, the research aimed to enhance the detection and prevention of memory-based injection attacks. The results showed that the trained RF and KNN models achieved high accuracy in classifying the injection status of processes based on the provided features. The findings of this study contribute to the development of robust intrusion detection systems capable of identifying memory-based injection attacks in real time. The utilization of RF and KNN algorithms, along with the selected features, offers a reliable approach for detecting malicious code injection and protecting computer systems from potential vulnerabilities.

The findings in this paper aren't the most effective way to detect malware and mitigate it, but they let a researcher know where one should be looking for malware and these findings will be used in our future works to improve the accuracy by identifying false positives, and false negatives accurately. Further research can be conducted to explore the performance of other machine learning algorithms and additional feature engineering techniques in improving the accuracy and efficiency of memory-based injection classification. The continuous advancement of machine learning and feature engineering methods will aid in fortifying the security of computer systems against sophisticated attacks.

References

[1] Riley R, Jiang X, Xu D. An architectural approach to preventing code injection attacks. *IEEE Trans Dependable Secure Comput*. 2010;7(4):351–65.

[2] Du M, Li F, Zheng G, Srikumar V. Deeplog: anomaly detection and diagnosis from system logs through deep learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298, Dallas, Texas, USA, 2017, October

[3] Hosseini A. *Ten Process Injection Techniques: a Technical Survey of Common and Trending Process Injection Techniques*. Endpoint Security Blog; 2017. [Online], Available: https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process.

[4] Abaimov S, Bianchi G. CODDLE: code-injection detection with deep learning. *IEEE Access*. 2019;7:128617–27.

[5] Le DT, Dinh DT, Nguyen QLT, Tran LT. A basic malware analysis process based on FireEye ecosystem. *Webology*. 2022;19(2):1011–1018, 1025, 1026, 1032. (ISSN: 1735-188X).

[6] Amer E, Zelinka I. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Comput & Secure.* 2020;92(2020):101760.