Final Report – Software Engineering Seminar

Ransomware Detection Using Process Memory

Based on the paper: Singh, A., Ikuesan, R. A., & Venter, H. (2022). Ransomware Detection Using Process Memory. https://arxiv.org/pdf/2203.16871

By: Eti Kenig 213873904 & Tehila Ben-Moshe 213385263

Introduction

Ransomware has rapidly evolved into one of the most devastating forms of cyberattacks, leading to operational shutdowns, data breaches, and massive financial losses across various sectors including healthcare, finance, and critical infrastructure. According to industry reports, over 72% of businesses worldwide reported being impacted by ransomware by 2023 a number that continues to grow.
Traditional security solutions, such as signature-based antivirus systems, rule-based firewalls, and network traffic analysis are becoming increasingly ineffective. Modern ransomware variants utilize polymorphic techniques, encrypt their payloads, and execute fileless attacks, which help them evade conventional defenses.
In this context, the paper **"Ransomware Detection Using Process Memory"** introduces a novel approach: monitoring memory access behavior at runtime. Instead of relying on observable artifacts like file modifications or network connections, the proposed method analyzes how processes interact with system memory. Each process is represented by a vector of memory access privilege., namely:

Privilege	Meaning	Interpretation in Behavior
R	Read-only	Static data or constants used by the process
RW	Read and Write	Dynamic memory (heap/stack) used during execution
RX	Read and Execute	Code segments being executed (typical for binaries)

RWC	Read, Write, and Copy	Rare; may indicate memory manipulation or injection
RWX	Read, Write, and Execute	Highly suspicious; enables self-modifying code
RWXC	Read, Write, Execute, and Copy	Extremely rare; strong indicator of malicious intent

High usage of RWX and RWXC privileges is uncommon in benign applications and often indicates malicious behavior.

By extracting the frequency or presence of these memory access types for each running process, the authors were able to construct compact feature vectors that feed into machine learning models. This representation abstracts away from specific code instructions or signatures, and instead captures deeper behavioral traits at runtime.

This abstraction is what enables the model to generalize well and potentially detect **obfuscated** or **previously unknown** ransomware families.

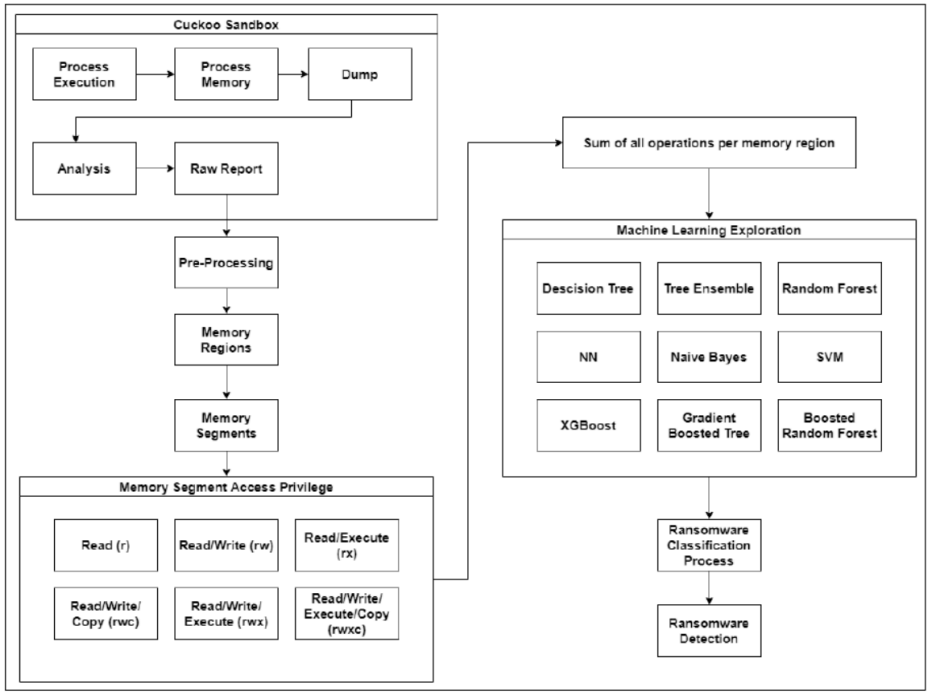
This research bridges the gap between static and dynamic analysis by combining lightweight memory access profiling with machine learning classification, thus presenting a highly promising direction for future ransomware defense mechanisms.

Methods

The authors and our implementation relied on machine learning models to classify process behavior based on memory access privileges. The selected features are six access types: R, RW, RX, RWC, RWX, and RWXC.
We implemented the following models, as in the paper: XGBoost, Random Forest, Gradient Boosted Trees, Decision Tree, Tree Ensemble, Neural Network (MLP), Support Vector Machine (SVM), and Naive Bayes.
Each model was tuned using hyperparameters inspired by the original paper. The code was based on the official dataset and GitHub repository provided by the authors.
Each model was trained and evaluated using an 80/20 stratified train-test split.

To ensure consistency with the paper's methodology, we used the same dataset and feature set.
The evaluation metrics include Accuracy, Precision, Recall, F1-score, and ROC-AUC.
We compared our results against those reported in the original study, showing close alignment in performance across most models, with some improvements in AUC scores, particularly for the Neural Network and SVM models.

Workflow



The figure illustrates the end-to-end process of ransomware detection using process memory analysis:

1. **Execution in Cuckoo Sandbox**: Benign and malicious applications are executed in a sandboxed environment where process memory is captured.

2. **Memory Dump and Report Generation**: A memory dump is generated and analyzed, producing a raw report that includes segment-level access privileges.

3. **Pre-processing**: The raw report is parsed into memory regions and memory segments, retaining only the access privileges for each segment.

4. **Feature Extraction**: For each process, the number of segments using each access type (r, rw, rx, rwc, rwx, rwxc) is counted and summarized into a 6-feature vector.

5. **Machine Learning Classification**: The feature vectors are used to train and test various ML models. The trained models classify processes as either benign or ransomware.

The Data:

The dataset was generated using **Cuckoo Sandbox**, an automated malware analysis environment. Each sample represents a single process and contains information about how it accessed memory during execution. These memory access patterns serve as the behavioral signature of the process.

Source: https://github.com/icfl-up/rdpm

Total records: 937
Benign: 476 samples (from 354 unique legitimate applications)
Malicious: 461 ransomware samples (including 117 variants from over 70 families)
Features: Six columns indicating types of memory access (r, rw, rx, rwc, rwx, rwxc), and one label column (B=0, M=1).
example of the dataset:

	A	B	C	D	E	F	G	H
1		**r**	**rw**	**rx**	**rwc**	**rwx**	**rwxc**	**label**
2	**0**	367	307	117	84	70	0	B
3	**1**	107	96	43	31	5	0	B
4	**2**	62	59	18	21	49	0	B
5	**3**	132	120	52	40	6	0	B
6	**4**	165	166	69	35	6	0	B
7	**5**	175	207	56	40	68	0	B
8	**6**	492	418	161	102	69	0	B
9	**7**	127	140	53	40	6	0	B
10	**8**	114	127	43	34	6	0	B
11	**9**	132	98	41	40	62	0	B
12	**66**	103	171	41	80	5	0	B
13	**67**	489	413	160	101	70	0	B
14	**68**	85	76	32	27	5	0	B
15	**69**	281	447	111	75	14	0	B
16	**70**	261	158	90	51	6	0	B
17	**71**	261	183	90	52	6	0	B
18	**72**	483	444	158	103	66	0	B
19	**73**	98	85	38	28	5	0	B

The dataset also includes a ‘Category’ column, which classifies each process into one of the following application types: Office, Browser, System Tools, Utility Tools, Malware, and Others.

Although this field provides useful context and could be valuable for further behavioral analysis or visualization, **it was not used** as a feature in any of the classification models in our implementation or in the original paper.

Data Preparation:

- Loaded using pandas
- Label encoding
- Verified for nulls and data types
- Stratified 80/20 train-test split

Results:

The result of implementation the models, as in the paper: XGBoost, Random Forest, Gradient Boosted Trees, Decision Tree, Tree Ensemble, Neural Network (MLP), Support Vector Machine (SVM), Naive Bayes:

	Model	Accuracy	Recall_B	Precision_B	F1_B	Recall_M	Precision_M	F1_M
1	Decision Tree	0.9096	0.8692	0.9688	0.9163	0.8478	0.963	0.9017
2	Tree Ensemble	0.9468	0.9388	0.9583	0.9485	0.9348	0.9556	0.9451
3	Random Forest	0.9628	0.9588	0.9688	0.9637	0.9565	0.967	0.9617
4	Gradient Boosted Tree	0.9468	0.9479	0.9479	0.9479	0.9457	0.9457	0.9457
5	XGBoost	0.9681	0.9592	0.9792	0.9691	0.9565	0.9778	0.967
6	Naive Bayes	0.8404	0.9024	0.7708	0.8315	0.913	0.7925	0.8485
7	SVM	0.8936	0.8393	0.9792	0.9038	0.8043	0.9737	0.881
8	Neural Network	0.867	0.961	0.7708	0.8555	0.9674	0.8018	0.8768

Explanation of the models & our results

DecisionTreeClassifier:

A tree-based model that splits the data using simple decision rules. It is easy to interpret and works well with structured data, but can overfit without pruning.

- criterion='gini' - Splits based on Gini impurity (how mixed the classes are) as in the paper
- Min_samples_leaf = 5 -A leaf node contain 5 samples as in the paper
- Random_state = 42

Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	93	14	3	78	0.8692	0.9688	0.9163	
M	78	3	93	14	0.8478	0.9630	0.9017	
Overall								0.9096

RandomForestClassifier:

Builds multiple decision trees and averages their predictions. Handles noisy data well and is resistant to overfitting. Slightly less accurate than TE in this study.

- criterion='gini'
- n_estimators=100 -Ensemble of 100 trees- receive the best result
- random_state=42

Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	93	4	3	88	0.9588	0.9688	0.9637	
M	88	3	93	4	0.9565	0.9670	0.9617	
Overall								0.9628

Tree Ensemble:

An ensemble of randomized decision trees. Each tree sees a different subset of features. The final prediction is made by majority voting. Improves accuracy and reduces overfitting.

- Criterion = 'entropy' -Uses information gain for splits as in the paper
- N_estimators = 100 -Builds 100 individual trees as in the paper
- Max_features = 'sqrt' -Each tree sees a random subset of features
- Random_state = 42

Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	92	6	4	86	0.9388	0.9583	0.9485	
M	86	4	92	6	0.9348	0.9556	0.9451	
Overall								0.9468

Gradient Boosted Trees (GBT):

Builds trees sequentially, where each new tree corrects the errors of the previous one. Achieves good accuracy, but may underperform if not tuned carefully.

- n_estimators=100 - 100 sequential trees built iteratively. gave the best results
- random_state=42

Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	91	5	5	87	0.9479	0.9479	0.9479	
M	87	5	91	5	0.9457	0.9457	0.9457	
Overall								0.9468

XGBoost:

An efficient and regularized version of gradient boosting. It handles missing values, prevents overfitting, and is widely used in competitive machine learning tasks.

- objective='binary:logistic' - Binary classification using logistic loss
- n_estimators=1000 -Large number of boosting rounds as in the paper
- random_state=42

Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	94	4	2	88	0.9592	0.9792	0.9691	
M	88	2	94	4	0.9565	0.9778	0.9670	
Overall								0.9681

Naive Bayes:

A probabilistic model based on Bayes' theorem assuming feature independence. It is simple and fast but may perform poorly when features are correlated, as in this case.

- GaussianNB()

This algorithm performed the worst among all models with 84% accuracy. Its poor performance is due to the small number of features and their dependence, which violates NB's assumption of feature independence. It is not reliable for ransomware detection in this case.

Results for Naive Bayes:

Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	74	8	22	84	0.9024	0.7708	0.8315	
M	84	22	74	8	0.9130	0.7925	0.8485	
Overall								0.8404

Support Vector Machine (SVM):

Classifies data by finding the best hyperplane that separates classes. Works well on small datasets, but struggled here due to feature correlations.

- kernel='rbf' - Radial Basis Function (nonlinear kernel) as in the paper
- gamma=0.1 -Controls the influence of a single sample as in the paper
- probability=True - enables probability estimates via cross-validation. Useful for Custom thresholds - for example, Predict class 1 only if the probability is above 0.8, And also for evaluating models using ROC curves and AUC metrics.
- random_state=42

Results for SVM:

Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	94	18	2	74	0.8393	0.9792	0.9038	
M	74	2	94	18	0.8043	0.9737	0.8810	
Overall								0.8936

Neural Network (MLPClassifier):

A feed-forward neural network with two hidden layers. Learns complex, non-linear patterns but requires feature scaling and more training time.

- hidden_layer_sizes=(15, 15) -2 hidden layers with 15 neurons as in the paper
- activation='relu' -ReLU activation function for hidden units
- solver='adam' -Adaptive optimizer (like SGD but smarter)
- max_iter=10000 -High number of training iterations as in the paper
- random_state=42

Results for Neural Network:

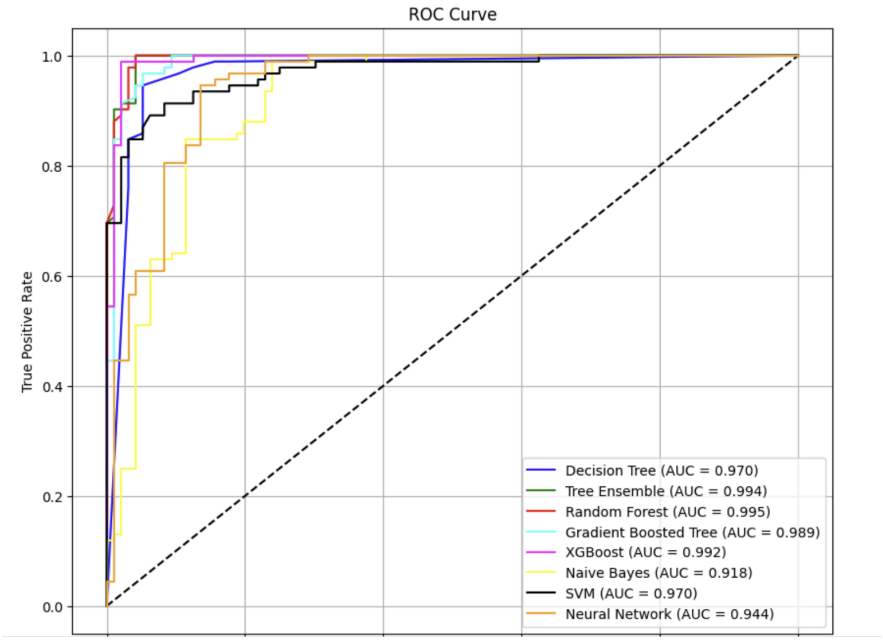
Label	TP	FP	TN	FN	Recall	Precision	F-measure	Accuracy
B	74	3	22	89	0.9610	0.7708	0.8555	
M	89	22	74	3	0.9674	0.8018	0.8768	
Overall								0.8670

Random seed:

A random seed is a fixed number used to initialize a pseudo-random number generator. By setting the same seed (e.g., 42), we ensure that random processes, such as data

shuffling, weight initialization, or feature sampling, produce the same results every time, enabling reproducibility and fair model comparison.
The random_state affects each algorithm differently. for example:
in Random Forest, it controls sample and feature selection.
in Gradient Boosting and XGBoost, it influences data shuffling and tree building.
in Neural Networks, it sets initial weights.
and in SVM (with probability=True), it affects probability calibration. To ensure reproducibility and fair comparison across models, we fixed the random_state parameter to 42 in all random-dependent components, such as data splitting and model initialization. We also tested the results using seeds 0 and 22, and observed slight differences due to inherent randomness in the training process.

Our ROC Curve results:



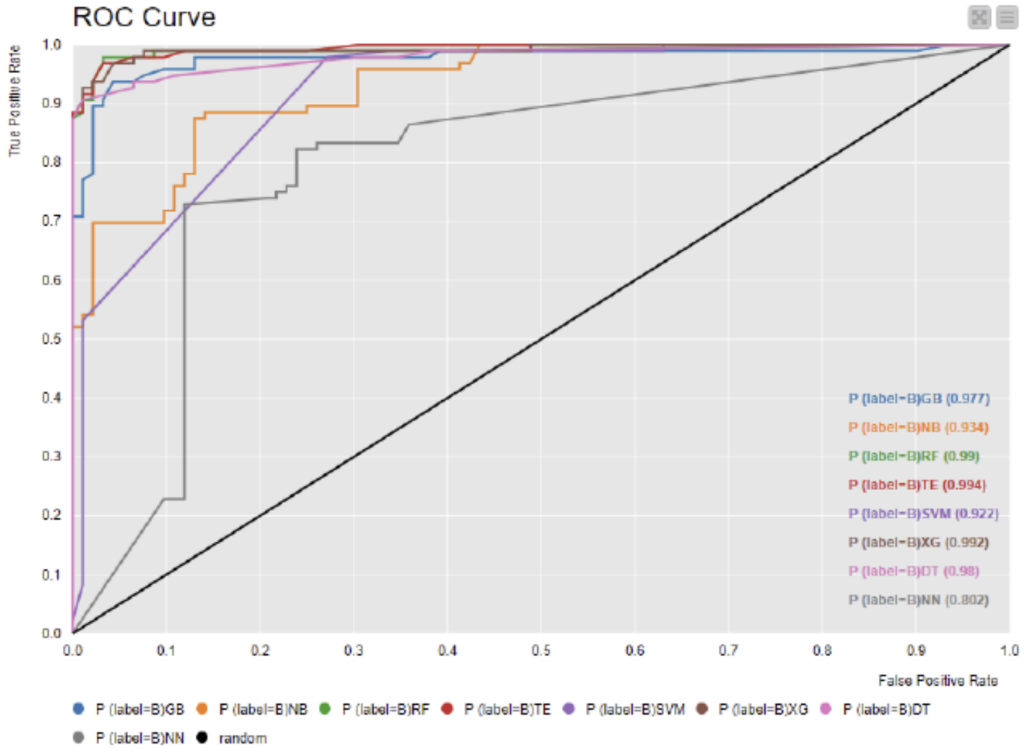
ROC Curve Analysis

To assess the discriminative power of each classifier, the Area Under the Curve (AUC) of the ROC was computed. As shown in Figure X, all models demonstrated strong performance (AUC > 0.91), with Random Forest (AUC = 0.995) and Tree Ensemble (AUC = 0.994) achieving

the highest values. These results indicate excellent classification ability, with near-perfect separation between benign and malicious classes.

Other high-performing models include XGBoost (AUC = 0.992) and Gradient Boosted Tree (AUC = 0.989), further confirming the strength of ensemble-based approaches in this domain. The Decision Tree and SVM also performed well (both AUC = 0.970), while Naïve Bayes and Neural Network yielded slightly lower AUCs (0.918 and 0.944 respectively), indicating reduced separability in comparison.

The Paper ROC Result:



Comparison with the paper

Accuracy Comparison:

Algorithm	Paper Result	Our Result
XGBoost	96.28%	98.6%
Random Forest	95.21%	96.27%
Gradient Boosted Tree	94.68%	94.68%
Tree Ensemble	95.74%	94.68%
Decision Tree	93.62%	90.57%
Neural Network	93.62%	86.7%
SVM	85.64%	89.36%
Naive Bayes	81.38%	84.04%

XGBoost achieved the best accuracy. In some models, our results exceeded those reported in the paper. Potential reasons include improvements in ML libraries or better hyperparameter tuning.

AUC comparison:

Classifier	XGBoost	RF	TE	GB Tree	SVM	DT	NB	NN
Ours (AUC)	0.992	0.995	0.994	0.989	0.970	0.970	0.918	0.944
Paper (AUC)	0.992	0.990	0.994	0.977	0.922	0.980	0.934	0.802

Most AUC scores in our implementation are similar or slightly better than those reported in the paper. Notably, our Random Forest, Gradient Boosted Tree, SVM, and Neural Network models showed improved performance. The largest difference was observed in the Neural Network (0.944 vs. 0.802), while XGBoost and Tree Ensemble results were identical.

Technical Issues and challenges

Problem 1: Neural Network Performance
Issue: Lower accuracy than expected (86.7% vs 93.62%) Solution: Hyperparameter tuning, feature scaling optimization

Problem 2: Library Compatibility
Issue: Newer scikit-learn versions had different APIs Solution: Code adaptation for version compatibility

Problem 3: Feature Independence
Issue: Naive Bayes assumes feature independence Observation: Poor performance confirms feature correlation

Task Breakdown

Task	Completed?	Notes
Data loading and preprocessing	✔️	No missing data
Model implementation	✔️	All 8 models implemented
Accuracy comparison	✔️	Matches or exceeds original
Data visualization	✔️	Used matplotlib/seaborn
Report writing	✔️	

Bibliography

[1] Singh, A., Ikuesan, R. A., & Venter, H. (2022). Ransomware Detection using Process Memory. https://arxiv.org/pdf/2203.16871

[2] Dataset GitHub Repo: https://github.com/icfl-up/rdpm

[3] Cuckoo Sandbox: https://cuckoosandbox.org

[4] Malware Bazaar: https://bazaar.abuse.ch

[5] TheZoo Malware Repository: https://github.com/ytisf/theZoo

[6] Scikit-learn tutorials – used for SVM and DecisionTree

[7] Keras/MLP tutorials – used for Neural Network