Gina Cody School of Engineering & Computer Science

SOEN 321: Systems Security

Final Report

Prepared by:

Karina Sanchez-Duran (ID: 40189860)
Qian Yi Wang (ID: 40211303)
Paul Humennyj (ID: 40209588)
Vanessa DiPietrantonio (ID: 40189938)
Mohamad Mounir Yassin (ID: 40198854)
Yash Patel (ID: 40175454)

Presented to:

Dr. Mohammad Mannan

April 22, 2025

Table of Contents

I. Summary

- **Goal:** To analyze and compare the effects of various attack methods on three different machine learning algorithms and identify critical vulnerabilities and proposed countermeasures.
- Three credit card fraud detection machine learning models were implemented and trained with the "Credit Card Fraud" from Kaggle. The three models were as follows: a decision tree model, a random forest model and a gradient boosting model.
- Various attack methods were performed on these credit card fraud detection models. The attacks implemented are as follows: Poisoning attack, model extraction attack, evasion attack, membership inference attack, and DoS attack
- Our attack implementations were done using Python and Jupyter notebooks with powerful libraries such as:
    - scikit-learn for training and saving the models
    - joblib to save/load the trained models and scalars
    - pandas and NumPy for data manipulation
    - VS Code for testing and development
    - A python based slowloris repository [10] as the base code for the DoS attack
- Critical vulnerabilities identified:
    - Simple modifications to training data significantly degrade the performance of the baseline models
    - Surrogate models with excellent performance can be easily obtained with no knowledge of the inner workings of the system
    - No API rate limiting can lead to service interruption due to DoS attacks
    - Susceptible to privacy leaks
- Countermeasures & Future Directives
    - API rate limiting
    - Authentication system
    - Implementation of security algorithms to detect malicious inputs from attacker
    - Adversarial training
    - Explainability tools (e.g. SHAP and LIME)

II. Base Model Implementation

A. Dataset Description

The credit card fraud detection model is built using the "Credit Card Fraud" from Kaggle [1]. The dataset contains the following features:

1. **distance_from_home**: The distance from home where the transaction occurred
2. **distance_from_last_transaction**: Distance from the last transaction
3. **ratio_to_median_purchase_price**: Ratio of transaction amount to median purchase price
4. **repeat_retailer**: Binary indicator (0/1) if transaction is from a repeat retailer
5. **used_chip**: Binary indicator if chip was used in the transaction
6. **used_pin_number**: Binary indicator if PIN was used
7. **online_order**: Binary indicator if the transaction was made online
8. **fraud**: Target variable (0 = Not Fraud, 1 = Fraud)

The dataset was preprocessed to handle missing values and ensure consistent column naming. All numerical features were standardized using StandardScaler to normalize the data distribution.

B. Model Architecture

Three machine learning classifiers were implemented to detect fraudulent transactions: Decision Tree, Random Forest, and Gradient Boosting [2]. Each model follows a consistent pipeline with minor architectural differences.

1. Decision Tree Classifier

The Decision Tree Classifier was implemented using scikit-learn's DecisionTreeClassifier with default hyperparameters and a fixed random_state of 42 to ensure reproducibility. This model was chosen as the baseline due to its interpretability, low computational cost, and fast training speed, making it ideal for initial prototyping and comparison. All numerical features were standardized using StandardScaler to improve model performance. The trained model and the fitted scaler were saved as fraud_model_dt.pkl and scaler.pkl, respectively, for future inference and evaluation.

2. Random Forest Classifier

The Random Forest Classifier was developed as a stronger benchmark to improve predictive performance and generalization over the base model. Built using scikit-learn's RandomForestClassifier, it utilizes an ensemble of decision trees with the entropy criterion and the same random_state of 42. The ensemble approach allows the model to reduce overfitting and capture more complex decision boundaries. Like the decision tree model, feature scaling was performed using StandardScaler, and the trained model along with its scaler were saved as random_forest_model.pkl and rf_scaler.pkl.

3. Gradient Boosting Classifier

The Gradient Boosting Classifier was included as a third model due to its ability to sequentially build an ensemble of weak learners, each one correcting the errors of its predecessors. Implemented using scikit-learn's GradientBoostingClassifier with a fixed random_state of 42, this model is particularly effective for handling imbalanced classification problems such as fraud detection. Feature normalization was again applied using StandardScaler, and the resulting model and scaler were saved as fraud_model_gb.pkl and gb_scaler.pkl. Gradient Boosting typically provides robust performance, especially in scenarios with noisy data or subtle fraud patterns.

C. Training Methodology

The training process for all 3 models follows the standard machine learning training steps:

1. **Data Loading and Preprocessing:**
    a. Load the dataset from cc_data.csv
    b. Handle missing values by dropping rows with null values
    c. Standardize column names for consistency
2. **Feature Engineering:**
    a. Separate features (X) and target variable (y)
    b. Apply StandardScaler to normalize numerical features
    c. Save the scaler for consistent preprocessing during inference
3. **Data Splitting:**
    a. Split data into training (80%) and testing (20%) sets
    b. Use stratified sampling to maintain class distribution

    c. Set random_state=42 for reproducibility (seed the random generator)

4. **Model Training:**
    a. Initialize model
    b. Fit the model on training data
    c. Save the trained model and scaler for deployment / testing with the attack models

D. Baseline Performance Metrics

The model's performance is evaluated through a comprehensive set of metrics that provide insights into its effectiveness in detecting fraudulent transactions.

The evaluation process begins with a confusion matrix, which visually represents a model's predictions through a seaborn heatmap, showing the distribution of true positives, true negatives, false positives, and false negatives. This visual representation helps in quickly understanding the model's classification patterns. The core performance metrics include accuracy, precision, recall, F1-score, and ROC AUC. Accuracy measures the overall correctness of predictions, while precision focuses on the ratio of true positives to all positive predictions. Recall indicates the model's ability to identify actual fraud cases, and the F1-score provides a balanced measure by combining precision and recall. The ROC AUC score gives an overall measure of the model's ability to distinguish between fraud and non-fraud cases.

The metrics and confusion matrices for all models can be seen in Appendix A. All models have perfect or near perfect scores with random forest scoring the highest, followed by decision tree and lastly the gradient boosting model.

All these metrics are automatically calculated during the training process using the test set, evaluating both binary predictions and probability scores. The results are presented in a standardized format with four decimal precision, providing immediate feedback on the model's performance. This comprehensive evaluation approach ensures a thorough understanding of the model's effectiveness in detecting fraudulent transactions.

III. Security Testing Methodology

A. Methodology

To verify the security of our credit card fraud model, we used hands-on testing. Once we had trained the main model using decision trees (i.e. the decision tree, random forest and gradient boosting models), we built attack scripts to try out several real-world attacks. We each tried a different vulnerability, like small changes to inputs or determining whether the model would leak training data. We quantified how often these attacks fooled the model into getting fraud wrong.

Moreover, we created a surrogate model that mimicked the behavior of the base model by generating random inputs and observing the resulting outputs. This surrogate was then used in evasion and new surrogate models were also used in the membership inference attacks to assess the ease with which a black-box attacker could evade the system.

B. Tools and Libraries Used

Our implementation used basic but powerful Python libraries:

- scikit-learn for training and saving the decision tree model
- joblib to save /load the trained model and scalar
- pandas and NumPy for data manipulation
- VS Code for testing and development
- A python based slowloris repository [10] as the base code for the DoS attack

All attacks and tests were carried out using simple scripts to keep the project lightweight and easy to run.

IV. Attack Implementations & Results

A. Poisoning Attack

A poisoning attack is a type of white-box adversarial attack wherein the attacker has complete knowledge of the target model's architecture as well as inner workings and purposefully manipulates the training data of the machine learning model [3]-[4]. In a targeted poisoning attack, the attacker manipulates the training data in a specific way in order to let malicious inputs go undetected or to create new vulnerabilities in the model [4]. In a non-targeted poisoning attack manipulates the training data in order to degrade the overall performance of the model [4].

In this project, a non-targeted poisoning attack was implemented on all three models: the decision tree, the random forest and the gradient boosting. The feature importance analysis (seen in Appendix B) revealed that the top three most important features for all three models are the ratio to median purchase price, whether or not it was an online order and the distance from home [5]. Simple modifications to the training data of the top three most important features were made on all three models in order to degrade their performance more effectively.

Before the attack was implemented, the baseline performance metrics for accuracy, precision, recall, F1-score and ROC-AUC of all models were near perfect (as seen Appendix A). After the attack, the performance of all three models plummeted with a particularly negative impact on precision, recall and F1-score (as seen in Appendix C). The order of least to most affected models are as follows: random forest, decision tree and gradient boosting. The results indicate that a simple poisoning attack (i.e. basic manipulation of data) can cause significant performance degradation and go undetected when there are no countermeasures in place. The results also indicate that the random forest is the most robust model of the three.

B. Model Extraction Attack

In a black-box adversarial attack, the attacker only knows the inputs and output(s) of the machine learning model and does not know anything about the target model's architecture or inner workings [6]. This type of attack is much more representative of a real life attack because, in most cases, the attacker will not have access to the details of the model's implementation [6].

A common tool used to implement black-box adversarial attacks is to create a surrogate model based on the inputs/outputs of the target model (i.e. extract the model) [6]. Then, an attacker can analyze the surrogate model and generate malicious input that goes undetected or purposefully hinders the surrogate model's predictions. The concept of attack transferability is then applied [6]. Meaning that if malicious inputs successfully attack the surrogate model, then they can most likely successfully attack the target model.

In this project, a surrogate model was created for all three models. For each model, a brute-force attack was implemented in which 10 000 randomly generated inputs and subsequent outputs were used as data to create a surrogate machine learning model based on the target model. Through trial and error (i.e. modifications to the pseudo-random number generation algorithm used), each surrogate model achieved extremely high values for accuracy, precision, recall, F1-score, and ROC AUC (as seen in Appendix D). The surrogate models built in this part of the project were subsequently used to successfully implement several evasion attack strategies (as seen below). During the membership inference attack (see below), several new surrogate models with 10 000 rows of data were implemented as well.

C. Evasion Attacks

The evasion attacks aim to subtly modify fraudulent transactions to evade detection by the surrogate model [12]. The implementation consists of designing and evaluating three different evasion strategies, each simulating plausible real-world attacker behavior. These attacks replicate scenarios where an adversary, without access to the internal model architecture or parameters, attempts to craft inputs that the model misclassified as legitimate at test time [13]. The goal is to determine how easily fraud can bypass detection with minimal, realistic changes.

Each evasion strategy applies small perturbations to known fraudulent inputs and evaluates whether these manipulated samples are misclassified as non-fraudulent by the model. All strategies were applied only to transactions initially labeled as fraud (target = 1). These modified records were then passed through the same preprocessing pipeline and classification model used during training. Predictions were collected, and evasion success was defined as any instance where the modified fraud sample was predicted as non-fraud (target = 0).

The following strategies were implemented:

- **Strategy A:** Decrease values of features most indicative of fraud.
    - Reduced feature 5 (used_chip) and feature 7 (online_order) by 1 if they were positive.
    - Results:
        - **Decision Tree**: 35 / 226 = **15.49%**
        - **Random Forest**: 27 / 149 = **18.12%**
        - **Gradient Boosting**: 52 / 230 = **22.61%**
    - Goal: Reduce fraud-indicative binary flags.
    - Binary fraud indicators are moderately sensitive to minimal manipulation. GB is most affected.
- **Strategy B:** Increase behavioral features that may resemble legitimate user behavior.
    - Increased feature 3 (ratio_to_median_purchase_price) and feature6 (used_pin_number) by 1
    - Results:
        - **Decision Tree**: 226 / 226 = **100.00%**
        - **Random Forest**: 149 / 149 = **100.00%**
        - **Gradient Boosting**: 230 / 230 = **100.00%**
    - Goal: Make fraudulent transactions appear more legitimate.
    - All models fail completely under this attack. Two-value increases fully bypass detection.
- **Strategy C:** Add randomized perturbations to all features.

- Each feature in a fraud sample was randomly incremented or decremented by 1.
- Results:
  - **Decision Tree: 169 / 226 = 74.78%**
  - **Random Forest: 182 / 226 = 80.53%**
  - **Gradient Boosting: 186 / 230 = 80.87%**
- Goal: Introduce generalized noise without targeting specific features.
- Even untargeted noise causes high misclassification. Ensemble models show serious weakness.

The effectiveness of each strategy was measured by the percentage of fraudulent samples misclassified as legitimate after modification. The table in Appendix E illustrates the results.

While the success rate for Strategy A was relatively low compared to other strategies, even these minimal modifications to two binary features led to incorrect classifications. This suggests the models place significant weight on these fraud indicators (chip usage and online ordering). Strategy B was completely successful for all three models, with 100% of modified fraudulent samples being classified as legitimate. Such a result is highly concerning and reveals a major vulnerability. Slightly increasing values that mimic legitimate behavior can entirely bypass detection, indicating an overly simplistic or under-regularized decision boundary. The randomized perturbation of Strategy C achieved a high evasion success rate among the three models, despite not targeting specific features. This result demonstrates that the model is broadly sensitive to noise. Even when an attacker modifies features arbitrarily, the models can misclassify fraud as legitimate, suggesting a lack of robustness to feature noise.

D. Membership Inference Attack

The goal of a membership inference attack is to find out if a given data sample is part of the target model's training dataset or not [16]. This attack was implemented in black box and white box settings. In a black box membership inference attack, the attacker only has access to the inputs and outputs of the model [15]. The attacker creates multiple shadow models that imitate the target model [15]. These shadow models are used to generate a training dataset for the attack model [15]. The attack model is used to predict whether a data sample is part of the training set or not. In a white box membership inference attack, the attacker has access to the internal details of the model, such as the intermediate calculations of the model [14]. The attacker is able to observe the model's behavior to determine if a data sample is part of the training dataset.

Our implementation of the black box membership inference attack consists of using two surrogate models to generate an attack dataset. An entry in the attack set is labeled 1 if it was part of the training set and 0 if it was part of the testing set of the surrogate model. The attack set is used to train the attack model which will infer whether a given data sample is in the training set.

For the white box membership inference attack, since the attacker has access to the details of the model [14], they would know that a decision tree classifier was being used. An attack dataset is generated in the same manner as from the black box attack. The attack dataset is used to find which leaf index the sample is at. Using the target model, the leaf indices of the training dataset are obtained. A data sample is part of the training dataset if its leaf index is in the leaf indices of the training dataset. Both methods were tested by using the actual training dataset and a synthetic one and seeing how many data samples are labeled correctly as member and nonmember. The results can be seen in Figure 22 in Appendix F.

The results show that for both black box and white box methods, all models were able to correctly predict members of the training set with an accuracy of over 90%. This indicates that the method may be overfitted to the training data, which is of concern since it indicates a higher level of risk of privacy leaks. As for predicting nonmembers, using the black box method, the attack model had an accuracy of over 95% for the decision tree model, while it was unable to even pass the 1% mark on the gradient boosting and random forest model. Using the white box method, the attack model was able to get an accuracy of 98% on both decision tree and gradient boosting models. These results indicate that although it is easy for the attack model to infer members of the training data, it is unable to infer nonmembers when attacking a random forest model. This means that if an attacker targets a random forest model, it is very likely that they will receive false-positive results.

E. DoS Attack

The chosen Denial of Service attack on the model was implemented as a Slowloris DoS attack [11], given this could be most practically performed using two personal computers over a local network with one being the attacker and the other the project host. The latter was set up to run the model wrapped by a basic Flask API with a "/predict" endpoint which the attacker could call. A Slowloris type attack involves the attacker sending partial HTTP requests to the server host, which establishes a connection, and tries to keep this connection alive by occasionally sending keep-alive headers to the server [11]. Keeping these connections alive slowly starves the host of resources, preventing it from completing actual requests.

We based the implementation of the attack on an existing slowloris python repo [10], whose code was modified to collect performance data and run a 5-minute attack. Multiple initial iterations of the attack were run with varying variables until the first run that caused a noticeable performance drop. Eventually, the following variables were identified as most responsible for the attack's effectiveness: the number of connections created by the attacker, how often the keep-alive headers are sent, and the number of bytes sent per connection alongside the keep-alive headers. With this in mind three tests were performed with varying byte sizes, this being the variable that had the greatest effect on the attack's effectiveness. The results are recorded in Appendix G.

Based on the results, we see that the error rate for the requests increases as we increase the number of bytes being sent. Curiously, the CPU usage slightly drops in the more aggressive Attack 3 compared to Attack 2, indicating that the error rate is likely more affected by memory usage. The processing time seems to decrease from Attack 1 to Attack 3, but this is misleading given it only reflects the successful responses of which there are fewer given the error rate. While the model survived and returned back to normal operating levels during Attack 1 and Attack 2, it crashed entirely near the end of Attack 3. Ultimately, the results show a big vulnerability in Flask API to slowloris attacks due to its lack of rate limiting which has the potential of crashing the application it is hosting.

V. Discussion of Results

A. Critical Vulnerabilities Identified

The results indicate that a lack of training data validation and strict access control can easily allow attackers to perform white-box adversarial attacks undetected. Furthermore, the results of the evasion attacks revealed that minimal changes to input data could lead to serious misclassification of data. This suggests a lack of robustness to noise on the part of all three models. However, out of all the models, the random forest was found to be the least affected by the attacks and is, thus, the most robust and most advisable model to use out of all three. The membership inference attacks also suggest that the models are susceptible to privacy leaks.

The lack of API rate limiting allows anybody, including attackers, to make unlimited requests and overwhelm the system leading to serious service disruption. The results further indicate that, through some very basic brute-force of the input data, a surrogate model with perfect metrics could be easily achieved. Therefore, attackers can easily derive these high-performing surrogate models to later use to develop other attacks/malicious input effectively through the concept of attack transferability, making all three machine learning models in this report susceptible to a simple black-box adversarial attack.

B. Risk Assessment

If any of the attacks described in this report are successfully implemented in a real-world credit card fraud detection model, the consequences could be disastrous. If an attacker can successfully degrade the performance of the target model (even temporarily), many fraudulent transactions can go undetected and many non-fraudulent transactions can be incorrectly flagged. In the case where a fraudulent credit card transaction goes undetected, banking systems who rely on such models can be left unaware of the scam for months, resulting in customers having their money stolen for potentially long periods of time. The results in this report also show that a model that is susceptible to membership-inference attacks could result in privacy leaks resulting in confidential information (i.e. credit card information) being exposed. Finally, a lack of API rate limiting can cause a denial of service on part of the model, leaving time for an attacker to perform a number of attacks undetected while the system is down.

C. Proposed Countermeasures

To minimize the risk of white-box adversarial attacks, one should enforce strict access control of the model's inner workings and introduce automated data validation techniques to ensure the training data meets a predefined standard [7].

To minimize the risk of black-box adversarial attacks such as evasion attack, one should make use of adversarial training and feature importance analysis to make the machine learning model more robust to noise and, thus, minimize the risk and efficacy of black-box adversarial attacks. One should also use a robust machine learning model such as random forest [8]. Furthermore, one should always use a secure and thorough multi-factor authentication system to minimize the risk of an attacker gaining API access and provide non-repudiation in case of an attack event [9].

One should also employ API rate limiting to prevent DoS attacks and brute-force attacks [9]. Additionally, one can implement an algorithm to attempt to detect illogical inputs with the goal of identifying them as malicious input in order to be able to identify a potential attacker quickly and, if need be, take legal action against them [9].

VI. Real-World Applications

A. Industry Impact

The vulnerabilities revealed in this project have important implications for industries that rely on machine learning for fraud detection, including banking, fintech, e-commerce, and insurance. Our results show that even simple attacks, such as evasion or poisoning, can significantly reduce a model's effectiveness, allowing fraudulent transactions to go undetected. The black-box attack further demonstrated that attackers do not need internal knowledge of the model to compromise its integrity.

Additionally, the membership inference attack raises serious privacy concerns. Models that unintentionally reveal whether a sample was part of the training data risk violating data protection regulations such as GDPR. This highlights the need for stronger privacy-preserving techniques and more secure deployment practices in real-world systems.

B. Future Research Directions

Moving forward, there are several areas worth exploring to improve the robustness and reliability of machine learning models used in fraud detection. Adversarial training, where models are exposed to manipulated inputs during training, may help increase their resistance to attacks. Incorporating differential privacy techniques could reduce the risk of membership inference by adding noise to sensitive data or predictions.

Explainability tools like SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) can provide insights into how much the model depends on certain features, helping developers adjust overly sensitive decision boundaries. Building ensemble models that combine multiple classifiers may also improve robustness by reducing the chance that a single weak model can be exploited. Finally, implementing real-time monitoring systems to detect unusual input patterns could serve as an additional line of defense.

These strategies, along with a stronger focus on ethical and legal implications, are essential for developing fraud detection systems that are not only accurate but also secure and responsible for real-world deployment.

VII. Conclusion

Having performed a thorough and varied series of attacks on three models trained on the "Credit Card Fraud" dataset, we have uncovered numerous critical vulnerabilities that require addressing if these models were to be deployed in production. Specifically, the models suffered strong degradation in performance due to poisoning attacks and were easily replicated as surrogates through model extraction tests. Additionally, the evasion attacks performed reveal that the model can be easily tricked and is susceptible to privacy leaks through membership inference attacks. Finally, the Flask API wrapping the model does not offer any built-in rate limiting protection from DoS attacks, which were effective in disrupting service. Overall, the three models in their current form are not suitably protected from the different attacks they are likely to encounter in the real world and must first be made more secure using the discussed countermeasures. Machine learning is a viable solution to many real-world problems and the analysis done in this report highlights the importance of security in its implementation. For future work, one should consider safety measures such as adversarial training, the use of robust models, strict access control to code and real-time monitoring.

VIII. References

[1] “Credit Card Fraud,” *www.kaggle.com*. https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud (accessed Jan. 31, 2025)

[2] scikit-learn, “1. Supervised learning — scikit-learn 0.22 documentation,” *Scikit-learn.org*, 2019. https://scikit-learn.org/stable/supervised_learning.html

[3] D. E. Lee, “White-Box Adversarial Attacks in AI - Dr. Ernesto Lee - Medium,” *Medium*, May 05, 2024. https://drlee.io/white-box-adversarial-attacks-in-ai-d1832c3a643f (accessed Mar. 13, 2025).

[4] IBM, “Data poisoning,” *Ibm.com*, Dec. 10, 2024. https://www.ibm.com/think/topics/data-poisoning (accessed Mar. 13, 2025)

[5] T. Shin, “Understanding Feature Importance in Machine Learning | Built In,” *builtin.com*, Nov. 07, 2023. https://builtin.com/data-science/feature-importance (accessed Mar. 11, 2025)

[6] W. Schroeder, “Learning Machine Learning Part 3: Attacking Black Box Models,” *Medium*, May 04, 2022. https://posts.specterops.io/learning-machine-learning-part-3-attacking-black-box-models-3efffc256909 (accessed Mar. 11, 2025)

[7] I. Limited, “Preventing Data Poisoning in AI,” *blogs.infosys.com*. https://blogs.infosys.com/digital-experience/emerging-technologies/preventing-data-poisoning-in-ai.html (accessed Apr. 12, 2025)

[8] A. Yadav, “Random Forest vs Decision Tree | Medium,” *Medium*, Aug. 11, 2024. https://medium.com/@amit25173/random-forest-vs-decision-tree-42b75aca4159 (accessed Apr. 12, 2025)

[9] SEARCH-LAB Ltd, “Model Extraction Attacks: An Emerging Threat to AI Systems,” *Scademy.ai*, Apr. 26, 2023. https://www.scademy.ai/post/model-extraction-attacks-an-emerging-threat-to-ai-systems (accessed Apr. 12, 2025)

[10] G. Yaltirakli, “slowloris”, Github, Apr. 30, 2023. https://github.com/gkbrk/slowloris (accessed Apr. 17, 2025)

[11] “Slowloris ddos attack,” Cloudflare, https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/ (accessed Apr. 17, 2025).

[12] IBM, “Evasion attack risk for AI,” *Ibm.com*, Feb. 7, 2025. https://blogs.infosys.com/digital-experience/emerging-technologies/preventing-data-poisoning-in-ai.html (accessed Apr. 17, 2025)

[13] Nightfall AI, “Evasion Attacks: AI Security 101,” *nightfall.ai*, https://www.nightfall.ai/ai-security-101/evasion-attacks (accessed Apr. 17, 2025)

[14] Y. Pang, T. Wang, X. Kang, M. Huai, and Y. Zhang, ‘White-box Membership Inference Attacks against Diffusion Models’, Nov. 21, 2024, arXiv: arXiv:2308.06405. doi: 10.48550/arXiv.2308.06405. (accessed Mar. 17, 2025)

[15] S. R. Shuvo, ‘Membership Inference Attacks on Machine Learning Models: Analysis and Mitigation’. (accessed Mar. 17, 2025)

[16] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, ‘Membership Inference Attacks Against Machine Learning Models’, in 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA: IEEE, May 2017, pp. 3–18. doi: 10.1109/SP.2017.41. (accessed Mar. 17, 2025)

Appendix A: Performance of Baseline Models

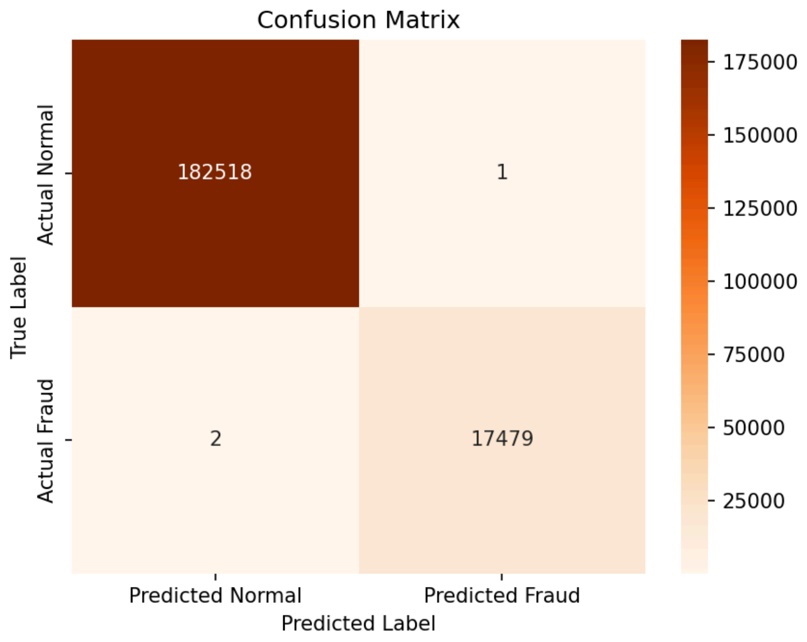

Figure 1: Confusion Matrix for Decision Tree

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
1.0000	0.9999	0.9999	0.9999	0.9999

Figure 2: Metrics for Decision Tree

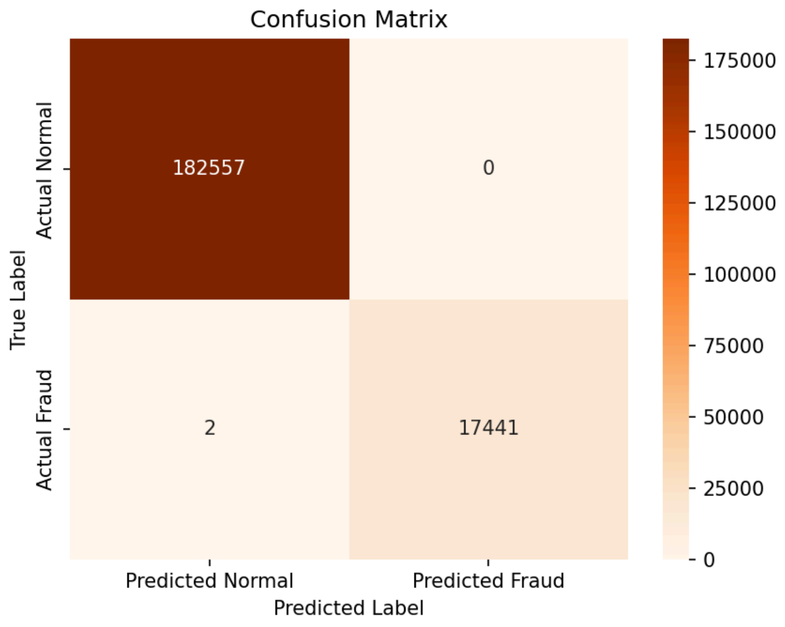

Figure 3: Confusion Matrix for Random Forest Model

Accuracy	Precision	Recall	F1-Score	ROC AUC
1.0000	1.0000	0.9999	0.9999	1.0000

Figure 4: Metrics for Random Forest

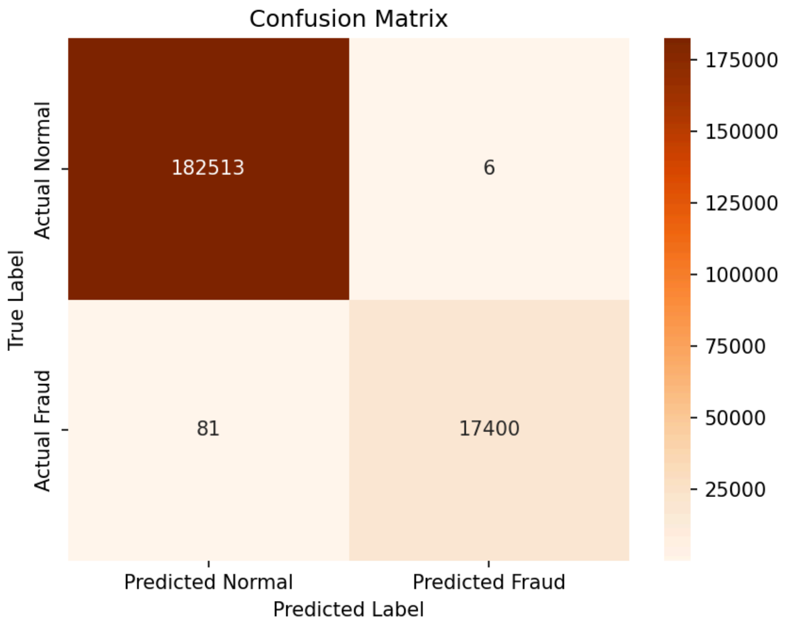

Figure 4: Confusion Matrix for Gradient Boosting

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
0.9996	0.9997	0.9954	0.9975	1.0000

Figure 5: Metrics for Random Forest

Appendix B: Feature Importances

Feature	Importance
ratio_to_median_purchase_price	0.422632
online_order	0.257501
distance_from_home	0.109354
used_pin_number	0.104819
used_chip	0.067798
distance_from_last_transaction	0.034779
repeat_retailer	0.003118

Figure 6: Feature Importance of Decision Tree Model

Feature	Importance
ratio_to_median_purchase_price	0.508624
online_order	0.150922
distance_from_home	0.150084
distance_from_last_transaction	0.074821
used_pin_number	0.059942
used_chip	0.048088
repeat_retailer	0.007519

Figure 7: Feature Importance of Random Forest Model

Feature	**Importance**
ratio_to_median_purchase_price	0.448025
online_order	0.237333
distance_from_home	0.122155
used_pin_number	0.086132
used_chip	0.051826
distance_from_last_transaction	0.050933
repeat_retailer	0.003596

Figure 8: Feature Importance of Gradient Boosting Model

Appendix C: Performance of Models after Poisoning Attack

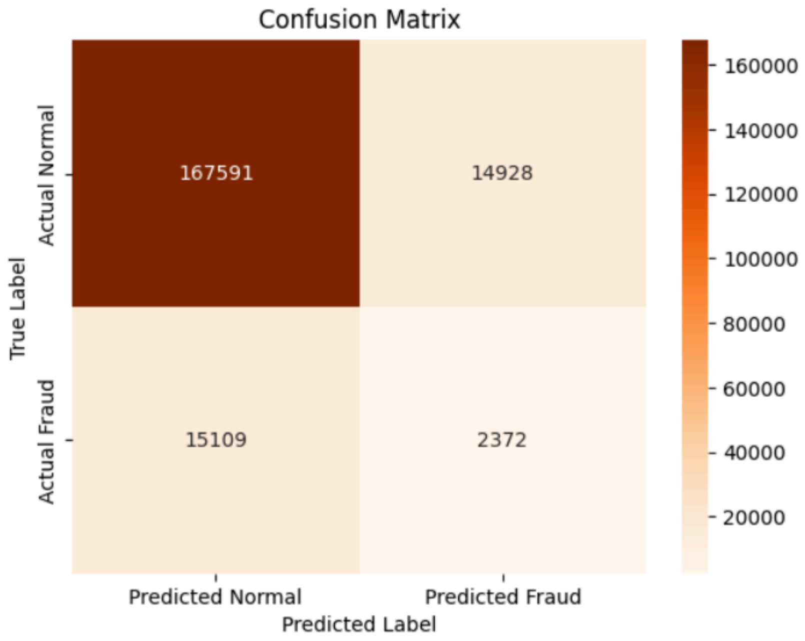

Figure 9: Confusion Matrix for Decision Tree after Poisoning Attack

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
0.8498	0.1371	0.1357	0.1364	0.5270

Figure 10: Performance Metrics for Decision Tree after Poisoning Attack

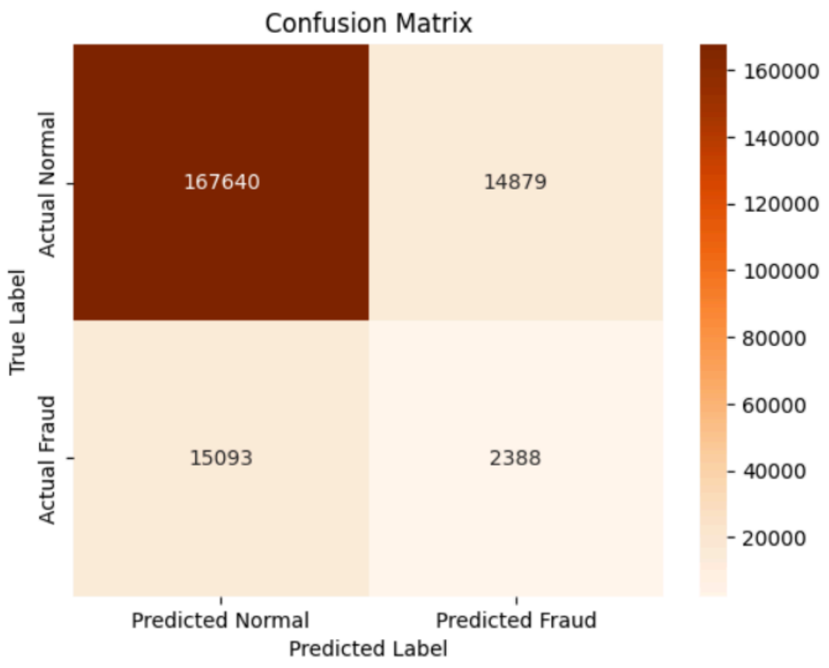

Figure 11: Confusion Matrix for Random Forest after Poisoning Attack

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
0.8501	0.1383	0.1366	0.1374	0.5558

Figure 12: Performance Metrics for Random Forest after Poisoning Attack

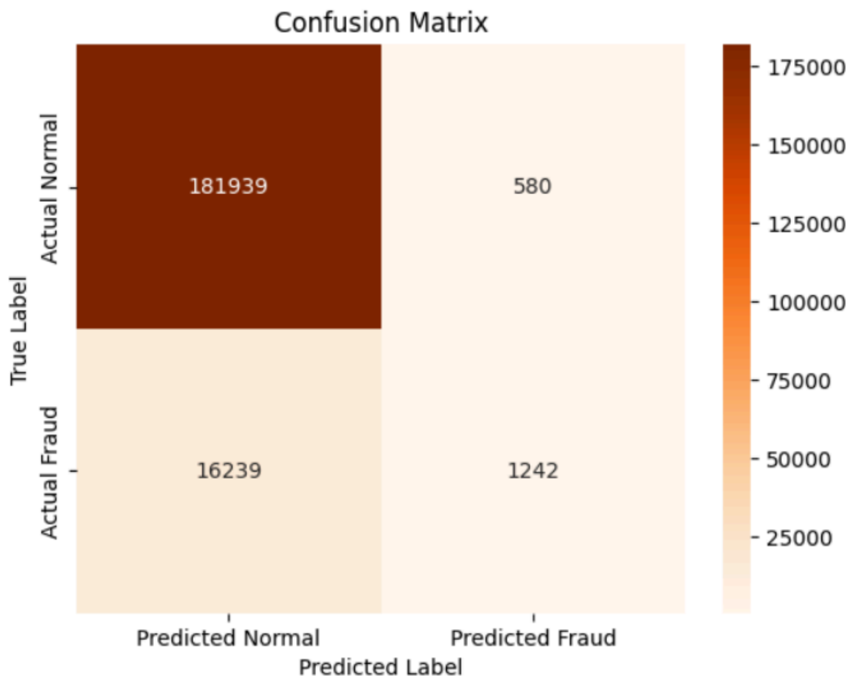

Figure 13: Confusion Matrix for Gradient Boosting after Poisoning Attack

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
0.9159	0.6817	0.0710	0.1287	0.6233

Figure 14: Performance Metrics for Gradient Boosting after Poisoning Attack

Appendix D: Performance of Surrogate Models

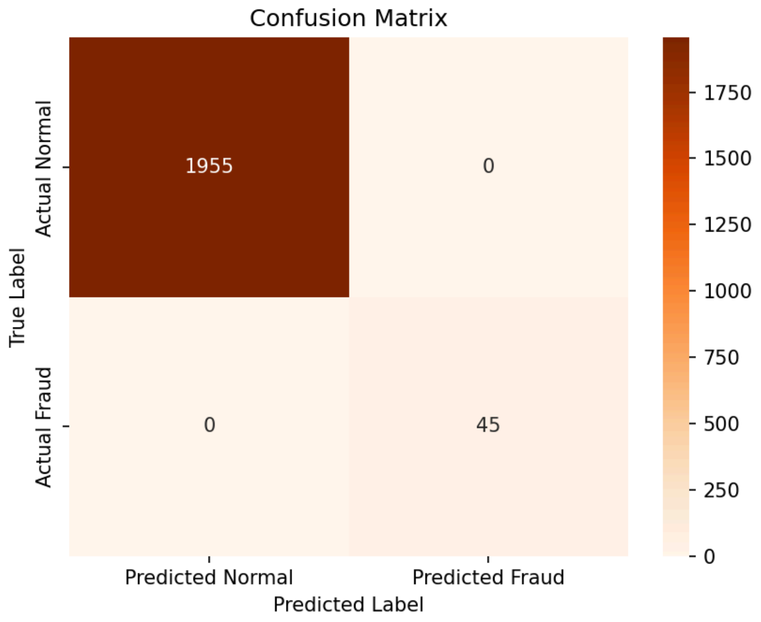

Figure 15: Confusion Matrix for Decision Tree Surrogate Model

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
1.0000	1.0000	1.0000	1.0000	1.0000

Figure 16: Decision Tree Model Performance Metrics

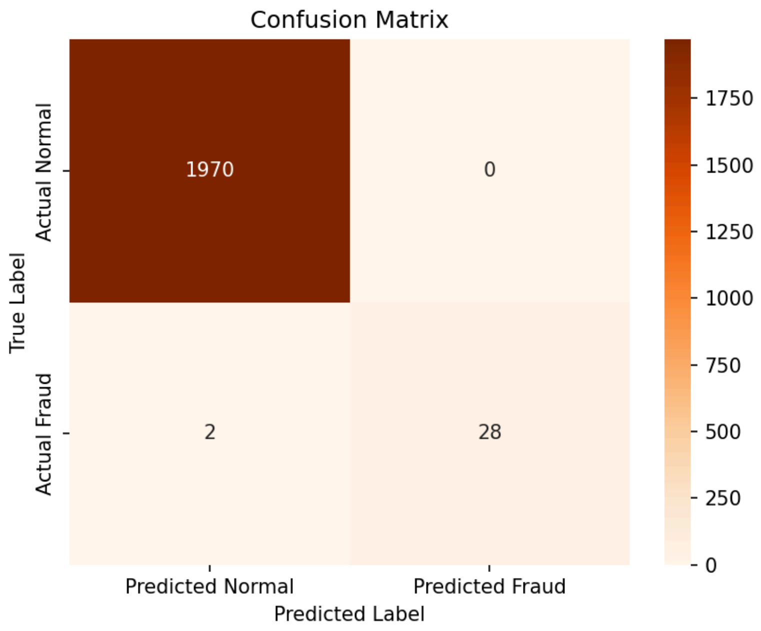

Figure 17: Confusion Matrix for Random Forest Surrogate Model

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
0.9990	1.0000	0.9333	0.9655	1.0000

Figure 18: Metrics for Random Forest Surrogate Model

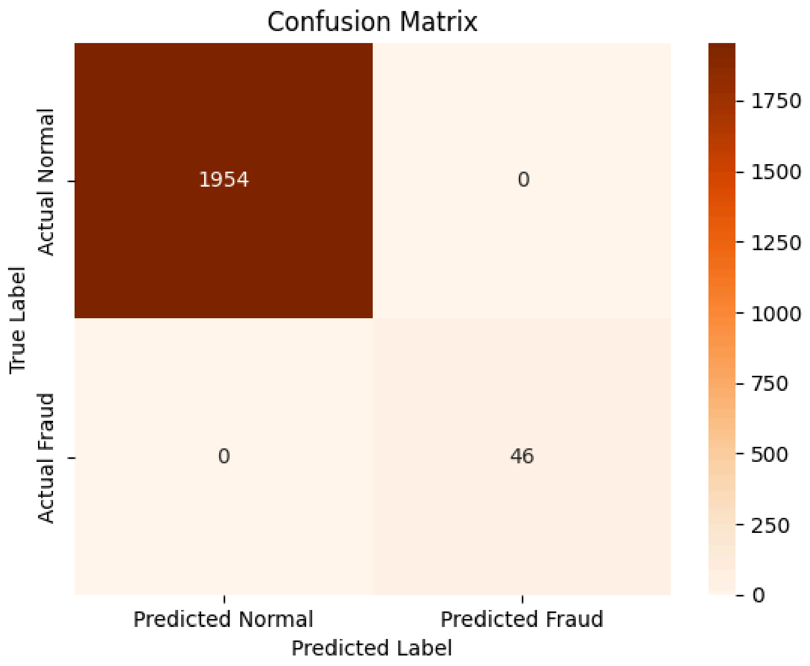

Figure 19: Confusion Matrix for Gradient Boosting Surrogate Model

Accuracy	**Precision**	**Recall**	**F1-Score**	**ROC AUC**
1.0000	1.0000	1.0000	1.0000	1.0000

Figure 20: Metrics for Gradient Boosting Surrogate Model

Appendix E: Evasion Attack Effectiveness by Strategy

Strategy	Model	Total Fraud Samples	Successful Evasions	Success Rate (%)
A	Decision Tree	226	35	15.49
	Random Forest	149	27	18.12
	Gradient Boosting	230	52	22.61
B	Decision Tree	226	226	100
	Random Forest	149	149	100
	Gradient Boosting	230	230	100
C	Decision Tree	226	169	74.78
	Random Forest	149	182	80.53
	Gradient Boosting	230	186	80.87

Figure 21: Evasion Attack Effectiveness across Strategies and Models

Appendix F: Membership Inference Attack Performance

Method	**Model**	**Accuracy for Inferring Members**	**Accuracy for Inferring Nonmembers**
Black Box	Decision Tree	0.91176	0.98517
	Gradient Boosting	0.98288	0.00205
	Random Forest	0.91176	0.00139
White Box	Decision Tree	0.92667	0.98987
	Gradient Boosting	0.98916	0.98933
	Random Forest	0.92598	0.00450

Figure 22: Accuracy of black box and white box methods

Appendix G: Slowloris DoS Attack Effectiveness

Average/Relative	**Idle**		**Attack 1**		**Attack 2**		**Attack 3**	
CPU usage	1.82%	1x	17.87%	9.8x	44.64%	24.5x	44.23%	24.3x
Memory usage	22.42%	1x	45.99%	2.1x	72.14%	3.2x	83.57%	3.7x
Response Time	0.0214s	1x	1.87s	87.2x	2.31s	108x	2.43s	113.5x
Processing Time	0.00425s	1x	0.779s	1832x	0.608s	1430x	0.206s	483x
Error Rate	0%		25%		63.3%		77.9%	

Note: By the end of Attack 3, the memory usage reached 100% and the model crashed.

Figure 23: Slowloris DoS Attack Effectiveness