Machine Learning Agent

Timothy Jones: jone2541

5/9/2025

Abstract

Today, AI is growing into everyday life, from phones to smart devices, ovens, and large enterprises. With that notion, the complexity and dynamics of attacks are also increasing, to the point that individual devices require some form of machine learning(ML) or artificial intelligence (AI) to assist with their defense. My project aimed to provide a local agent that utilizes an ML/AL model to learn a baseline of good DNS network traffic on the device and respond to a Kaminsky DNS attack.

1 Introduction

1.1 Why an agent

It will soon be common for devices of all types to run local agents that collect, scan, report, and assess their security in real time. For the US government, this is called out in 'Zero-Trust'[3], where devices must be postured and evaluated in real time. To achieve real-time, ML/AI will provide dynamic security from the device itself and throughout the entire enterprise. As vendors and companies try to solve this problem, I intend to give a real example of how ML/AI could be utilized to train and detect attacks for my project.

1.2 Technologies used and why

Within the ML/AI community, it's no secret that Python or R is the most common language of choice. For my project, I chose Python because of the readily available libraries and my experience with them from CS529. To save the data required for training and to store alerting, I decided on SQLite. It is lightweight, scalable, interoperable with almost any device, and ideal for use on devices with resource constraints[2]. For the OS, I chose Ubuntu, and set it up like the DNS Cache poisoning lab in CS528[5].

1.3 Why DNS network traffic and the Kaminsky DNS attack

Compromising DNS of a local network offers attackers a wide range of exploits, and it is still common today. Although DNSSEC offers some protections, having clients learn or detect attacks adds a practical layer of protection. With the Kaminsky DNS attack, good DNS responses are overloaded with thousands of guessing attempts to the DNS server. This offers a great dataset, as clusters will naturally form from the many requests in a small time window, for the same fully qualified domain name[5].

2 Choosing an ML/AI model

2.1 Overview of model learning

Machine learning models are divided into three categories: supervised, unsupervised, and semi-supervised[6]. Supervised learning offers the greatest accuracy, but requires data to be labeled and large amounts of resources to train and test on the datasets. Unsupervised learning doesn't require data labeling, focuses on finding anomalies, and doesn't require a lot of resources. Semi-supervised learning includes labeled and unlabeled data, but relies on the performance of the labeled data for scoring and uses more resources than unsupervised learning.

2.2 Isolation Forest Model

For the agent, I chose the unsupervised model, Isolation Forest, because of the requirement to be light weight and run on resource-restricted devices like phones or internet of things devices. Isolation Forest learns from unlabeled data, identifies anomalies, and forms clusters based on fields from the dataset[4]. During a Kaminsky DNS Attack, it would most certainly form clusters related to TTL, response volume, and distinct DNS names.

3 Implementation and experiments

3.1 Database Schema

For the project, I found it was required to maintain two datasets. The first is saving the DNS traffic collected during 'good' traffic windows, and during 'attack' traffic windows. For each, I stored the DNS query, response code, transaction ID, response ID, TTL, and entropy of the DNS name. This allowed the model to form clusters around the DNS name frequencies, TTL, and responses in a given time frame.

```
dns_features (
        timestamp REAL PRIMARY KEY,
        query_name TEXT,
        query_type INTEGER,
        response_src_ip TEXT,
        response_txid INTEGER,
        response_code INTEGER,
        responses_in_window INTEGER,
        unique_src_ips_in_window INTEGER,
        unique_txids_in_window INTEGER,
        pkt_qname_entropy REAL,
        qname_entropy_mean_in_window REAL,
        qname_entropy_stddev_in_window REAL,
            label INTEGER DEFAULT 0
)
```

The second dataset to be stored was the detected attacks when the model is running. This provides the ability to not only have a log trace saved, but for additional system integration for later use.

```
alerts (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp REAL,
            alert_message TEXT,
            query_name TEXT,
            response_src_ip TEXT,
            response_txid INTEGER,
            responses_in_window INTEGER,
            unique_src_ips_in_window INTEGER,
            unique_txids_in_window INTEGER
        )
```

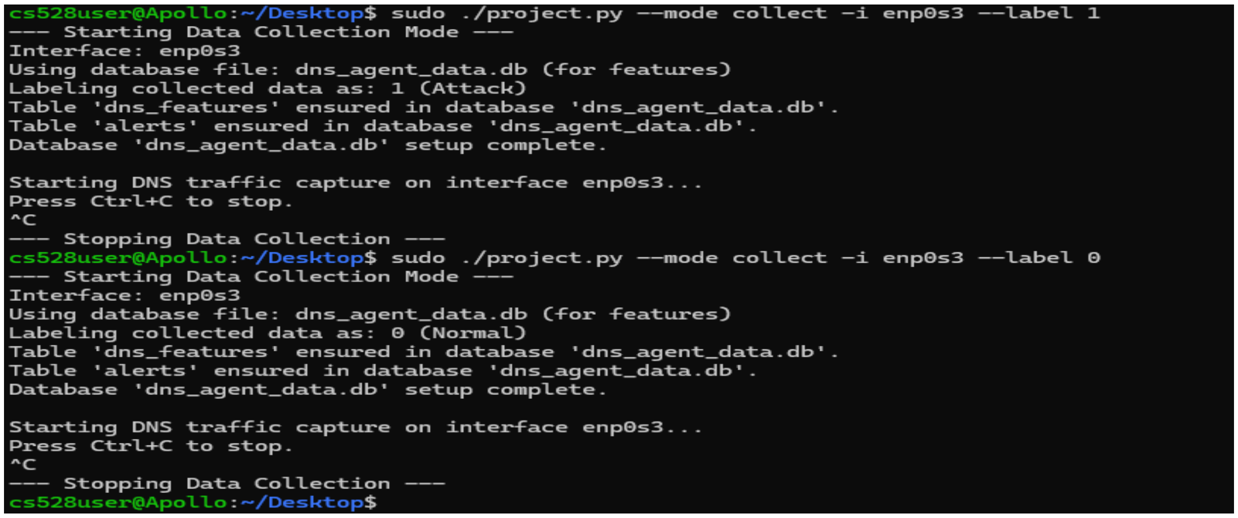
3.2 Dataset collection

For dataset collection, the user or service needs to provide the ‘collect’ flag, the given interface name or it will use the default, and the labeling of whether the data is ‘0’ for normal traffic, or ‘1’ for attacking. During data collection, the agent program will listen on the given interface, read the packets received, normalize them, and store them within the SQLite Database. For my project, I utilized a list of good domains that were requested during the ‘0’ flag collection, and when the ‘1’ flag was used, I had a separate machine use the Kaminsky DNS attack against the DNS server. The dataset contained attack and good ‘white listed’ domain response data.

Full collect mode is as follows:

- Mode: Collect
- Interface: Give the program the interface you wish to listen on
- DB File (Optional, Use Default): Give the program a Sqlite DB to use
- Log File (Optional, Use Default): : Give the program a Sqlite DB to use

- Label: For collection mode, provides labeling functionality.
- Use 0 for labeling good traffic, 1 for attackers



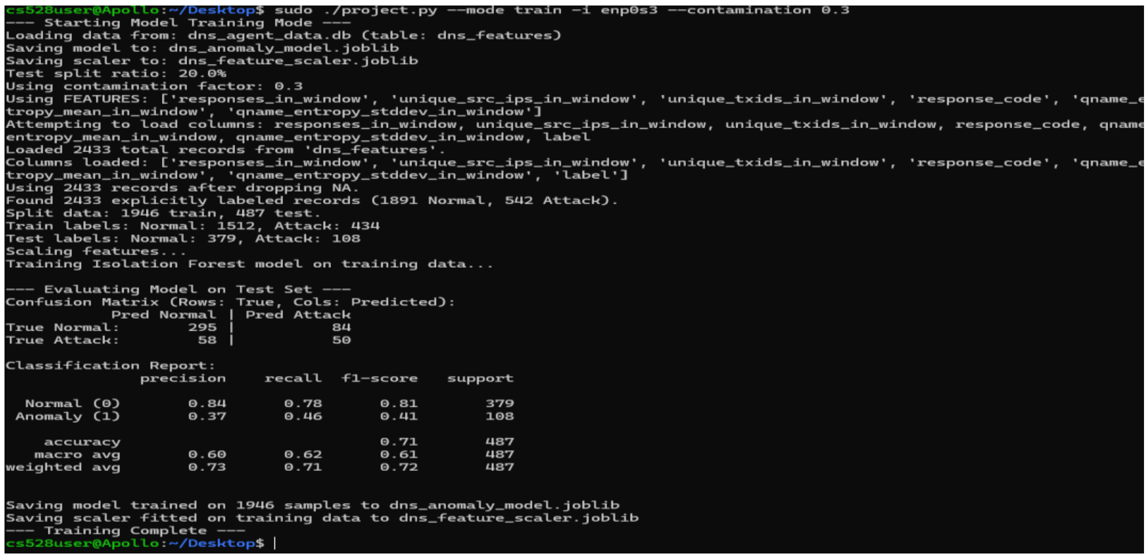
```
cs528user@Apollo:~/Desktop$ sudo ./project.py --mode collect -i enp0s3 --label 1
--- Starting Data Collection Mode ---
Interface: enp0s3
Using database file: dns_agent_data.db (for features)
Labeling collected data as: 1 (Attack)
Table 'dns_features' ensured in database 'dns_agent_data.db'.
Table 'alerts' ensured in database 'dns_agent_data.db'.
Database 'dns_agent_data.db' setup complete.

Starting DNS traffic capture on interface enp0s3...
Press Ctrl+C to stop.
^C
--- Stopping Data Collection ---
cs528user@Apollo:~/Desktop$ sudo ./project.py --mode collect -i enp0s3 --label 0
--- Starting Data Collection Mode ---
Interface: enp0s3
Using database file: dns_agent_data.db (for features)
Labeling collected data as: 0 (Normal)
Table 'dns_features' ensured in database 'dns_agent_data.db'.
Table 'alerts' ensured in database 'dns_agent_data.db'.
Database 'dns_agent_data.db' setup complete.

Starting DNS traffic capture on interface enp0s3...
Press Ctrl+C to stop.
^C
--- Stopping Data Collection ---
cs528user@Apollo:~/Desktop$
```

3.3 Model training

For dataset training, the user or service needs to provide the ‘train’ flag, the given interface name or it will use the default, and the contamination value within the range of 0.0-0.5. During training mode, the dataset collected prior is split into a 80/20 dataset of 80% for training purposes, and 20% for testing. The features trained on are TTL, frequency, responses, response codes, transaction IDs, and source IPs. Once training is finished, the model and scalar information are stored in the file system for future use.

Full train mode is as follows:

- Mode: Train
- Interface: Give the program the interface you wish to listen on
- DB File (Optional, Use Default): Give the program a Sqlite DB to use
- Model(Optional, Use Default): Give the program a model to use
- Scaler(Optional, Use Default): Give the program a scalar to use
- Log File (Optional, Use Default): : Give the program a Sqlite DB to use
- Test Split (Optional, Use Default): : Give the program a non standard training/test split
- Contamination: For Isolated Forest Training



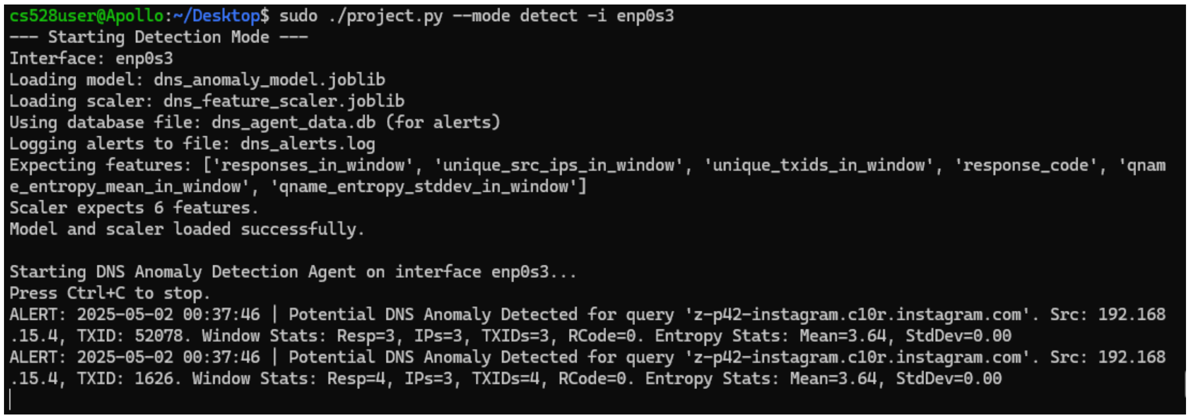
```
cs528user@Apollo:~/Desktop$ sudo ./project.py --mode train -i enp0s3 --contamination 0.3
--- Starting Model Training Mode ---
Loading data from: dns_agent_data.db (table: dns_features)
Saving model to: dns_anomaly_model.joblib
Saving scaler to: dns_feature_scaler.joblib
Test split ratio: 20.0%
Using contamination factor: 0.3
Using FEATURES: ['responses_in_window', 'unique_src_ips_in_window', 'unique_txids_in_window', 'response_code', 'qname_e
tropy_mean_in_window', 'qname_entropy_stddev_in_window']
Attempting to load columns: responses_in_window, unique_src_ips_in_window, unique_txids_in_window, response_code, qname
entropy_mean_in_window, qname_entropy_stddev_in_window, label
Loaded 2433 total records from 'dns_features'.
Columns loaded: ['responses_in_window', 'unique_src_ips_in_window', 'unique_txids_in_window', 'response_code', 'qname_e
tropy_mean_in_window', 'qname_entropy_stddev_in_window', 'label']
Using 2433 records after dropping NA.
Found 2433 explicitly labeled records (1891 Normal, 542 Attack).
Split data: 1946 train, 487 test.
Train labels: Normal: 1512, Attack: 434
Test labels: Normal: 379, Attack: 108
Scaling features...
Training Isolation Forest model on training data...

--- Evaluating Model on Test Set ---
Confusion Matrix (Rows: True, Cols: Predicted):
           Pred Normal | Pred Attack
True Normal:       295 |          84
True Attack:        58 |          50

Classification Report:
              precision    recall  f1-score   support

  Normal (0)       0.84      0.78      0.81       379
 Anomaly (1)       0.37      0.46      0.41       108

    accuracy                           0.71       487
   macro avg       0.60      0.62      0.61       487
weighted avg       0.73      0.71      0.72       487


Saving model trained on 1946 samples to dns_anomaly_model.joblib
Saving scaler fitted on training data to dns_feature_scaler.joblib
--- Training Complete ---
cs528user@Apollo:~/Desktop$ |
```

3.4 Model Detect

For model detect mode, the user or service needs to provide the 'detect' flag, and the given interface name or it will use the default. During detect mode, the model evaluates every DNS packet that is received, and predicts whether it is an anomaly or not. If it is, then its recorded as an attack within the database, and in the file system within the dns_alerts.log.

Full train mode is as follows:

- Interface: Give the program the interface you wish to listen on
- DB File (Optional, Use Default): Give the program a Sqlite DB to use
- Model(Optional, Use Default): Give the program a model to use
- Scaler(Optional, Use Default): Give the program a scalar to use
- Log File (Optional, Use Default): : Give the program a Sqlite DB to use



```
cs528user@Apollo:~/Desktop$ sudo ./project.py --mode detect -i enp0s3
--- Starting Detection Mode ---
Interface: enp0s3
Loading model: dns_anomaly_model.joblib
Loading scaler: dns_feature_scaler.joblib
Using database file: dns_agent_data.db (for alerts)
Logging alerts to file: dns_alerts.log
Expecting features: ['responses_in_window', 'unique_src_ips_in_window', 'unique_txids_in_window', 'response_code', 'qnam
e_entropy_mean_in_window', 'qname_entropy_stddev_in_window']
Scaler expects 6 features.
Model and scaler loaded successfully.

Starting DNS Anomaly Detection Agent on interface enp0s3...
Press Ctrl+C to stop.
ALERT: 2025-05-02 00:37:46 | Potential DNS Anomaly Detected for query 'z-p42-instagram.c10r.instagram.com'. Src: 192.168
.15.4, TXID: 52078. Window Stats: Resp=3, IPs=3, TXIDs=3, RCode=0. Entropy Stats: Mean=3.64, StdDev=0.00
ALERT: 2025-05-02 00:37:46 | Potential DNS Anomaly Detected for query 'z-p42-instagram.c10r.instagram.com'. Src: 192.168
.15.4, TXID: 1626. Window Stats: Resp=4, IPs=3, TXIDs=4, RCode=0. Entropy Stats: Mean=3.64, StdDev=0.00
|
```

3.5 Model Tuning and Evaluation

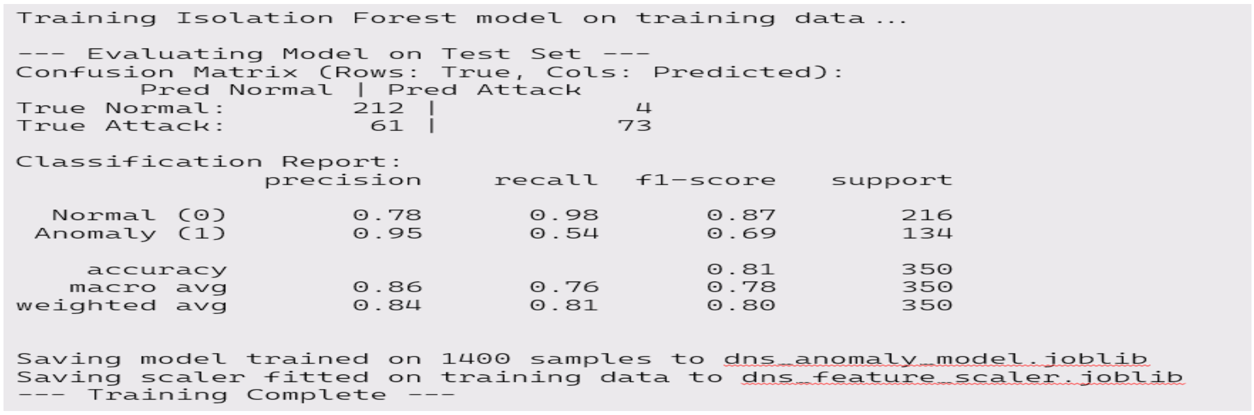
- With Isolation Forest, the most effective setting to tune the formation of clusters is ‘contamination’, by default, it's set to ‘auto’. The feature is used as a threshold for the expected number of anomalies. With the Kaminsky DNS attack, we can assume there will be a higher-than-normal ratio of anomalies, as most traffic will be DNS response attempts.
  The first revision of features I trained on was:
  'responses_in_window',
- 'unique_src_ips_in_window',
- 'unique_txids_in_window',
- 'response_code',
- 'response_ttl',
- 'ttl_mean_in_window',
- 'ttl_stddev_in_window',
- 'ttl_min_in_window',
- 'ttl_max_in_window’

With contamination set to ‘auto’:

Normal Traffic Detection: High precision and recall.

Anomaly Detection: High precision, but low recall of 54%

Usable model, as it detects 50% of attacks with over 90% precision.



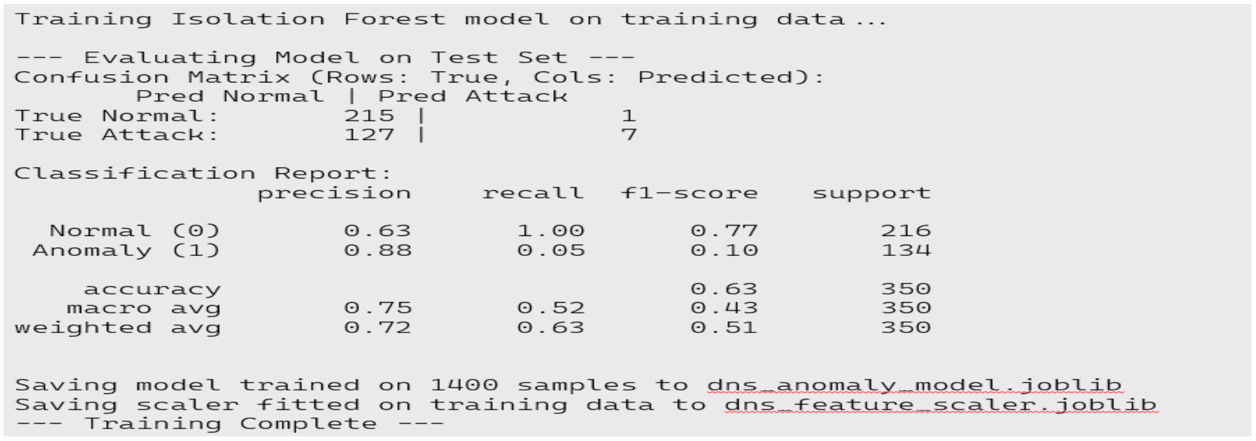
```
Training Isolation Forest model on training data ...

--- Evaluating Model on Test Set ---
Confusion Matrix (Rows: True, Cols: Predicted):
       Pred Normal | Pred Attack
True Normal:        212 |           4
True Attack:         61 |          73

Classification Report:
              precision    recall  f1-score   support

  Normal (0)       0.78      0.98      0.87       216
 Anomaly (1)       0.95      0.54      0.69       134

    accuracy                           0.81       350
   macro avg       0.86      0.76      0.78       350
weighted avg       0.84      0.81      0.80       350


Saving model trained on 1400 samples to dns_anomaly_model.joblib
Saving scaler fitted on training data to dns_feature_scaler.joblib
--- Training Complete ---
```

With contamination set to ‘0.1’:

Normal Traffic Detection: Low precision of 63% and 100% recall.

Anomaly Detection: High precision, but low recall of 5%

Unusable model, as it misses 95% of attacks.



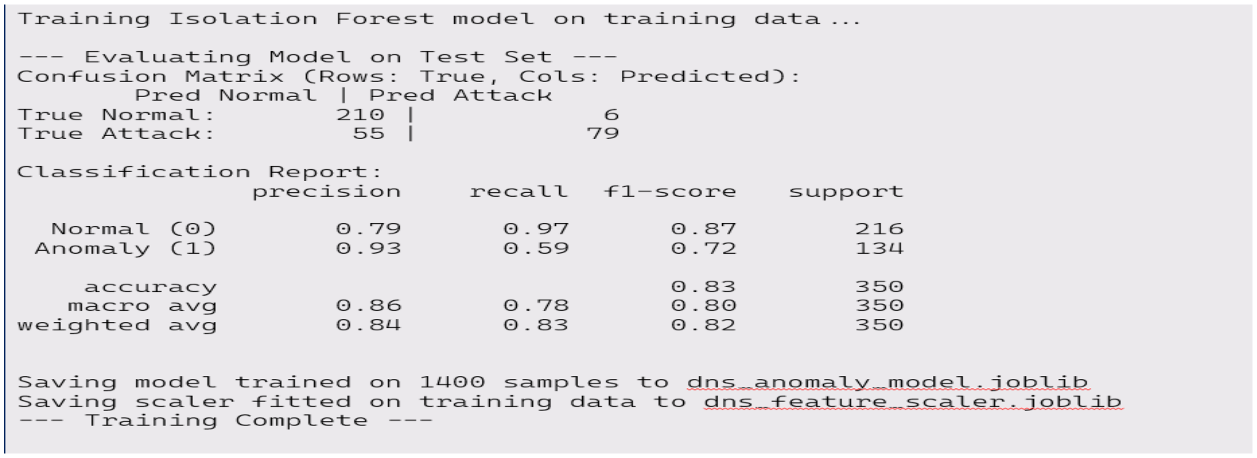
```
Training Isolation Forest model on training data...

--- Evaluating Model on Test Set ---
Confusion Matrix (Rows: True, Cols: Predicted):
        Pred Normal | Pred Attack
True Normal:        215 |            1
True Attack:        127 |            7

Classification Report:
              precision    recall  f1-score   support

  Normal (0)       0.63      1.00      0.77       216
 Anomaly (1)       0.88      0.05      0.10       134

    accuracy                           0.63       350
   macro avg       0.75      0.52      0.43       350
weighted avg       0.72      0.63      0.51       350


Saving model trained on 1400 samples to dns_anomaly_model.joblib
Saving scaler fitted on training data to dns_feature_scaler.joblib
--- Training Complete ---
```

With contamination set to ‘0.3’:

Normal Traffic Detection: Low precision of 79% and 93% recall.

Anomaly Detection: High precision of 93%, but low recall of 59%.

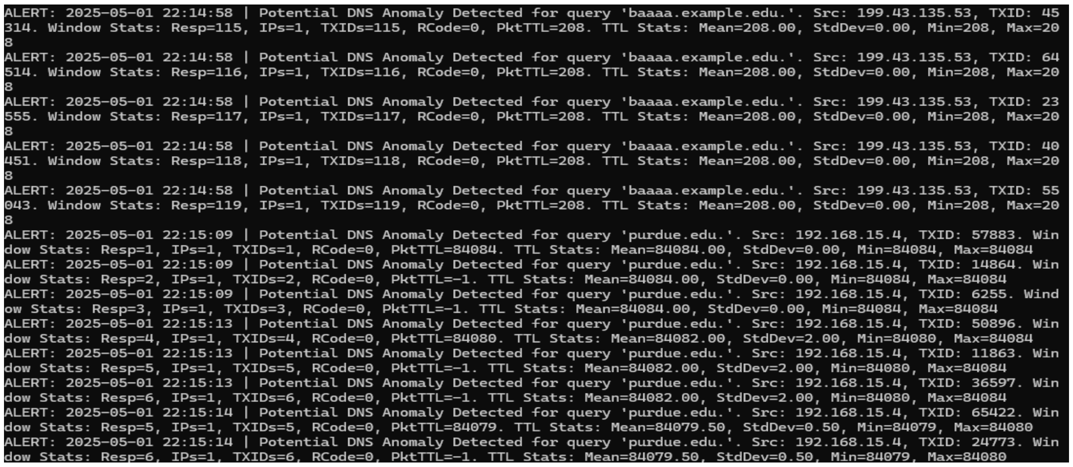
Usable model, as its slightly better than ‘auto’ with 59% attack detection rate.



```
Training Isolation Forest model on training data...

--- Evaluating Model on Test Set ---
Confusion Matrix (Rows: True, Cols: Predicted):
        Pred Normal | Pred Attack
True Normal:       210 |           6
True Attack:        55 |          79

Classification Report:
              precision    recall  f1-score   support

  Normal (0)       0.79      0.97      0.87       216
 Anomaly (1)       0.93      0.59      0.72       134

    accuracy                           0.83       350
   macro avg       0.86      0.78      0.80       350
weighted avg       0.84      0.83      0.82       350


Saving model trained on 1400 samples to dns_anomaly_model.joblib
Saving scaler fitted on training data to dns_feature_scaler.joblib
--- Training Complete ---
```

These data points led me to begin testing with the model with ‘0.3’, but what I found was that it would ‘detect’ every DNS request as an attack... even learned ‘good’ traffic. From the dataset, I found that it was creating clusters based on TTL, where ‘good’ traffic was falling within the 80000- 85000ms range, and bad ' traffic always settled in the 200ms range.

Effectively, any request would be flagged as an attack.

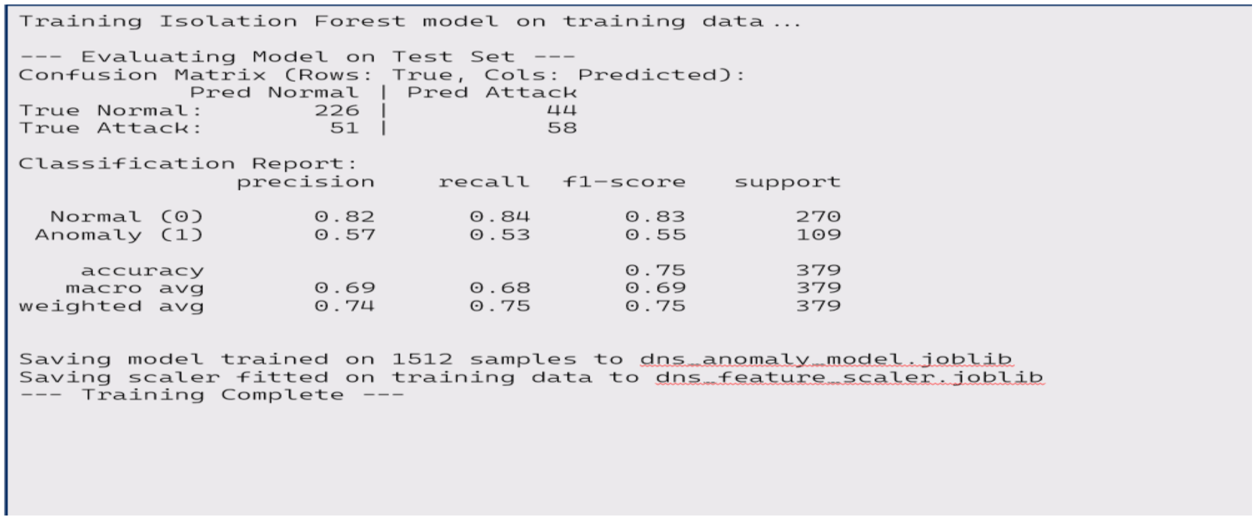


```
ALERT: 2025-05-01 22:14:58 | Potential DNS Anomaly Detected for query 'baaaa.example.edu.'. Src: 199.43.135.53, TXID: 45
314. Window Stats: Resp=115, IPs=1, TXIDs=115, RCode=0, PktTTL=208. TTL Stats: Mean=208.00, StdDev=0.00, Min=208, Max=20
8
ALERT: 2025-05-01 22:14:58 | Potential DNS Anomaly Detected for query 'baaaa.example.edu.'. Src: 199.43.135.53, TXID: 64
514. Window Stats: Resp=116, IPs=1, TXIDs=116, RCode=0, PktTTL=208. TTL Stats: Mean=208.00, StdDev=0.00, Min=208, Max=20
8
ALERT: 2025-05-01 22:14:58 | Potential DNS Anomaly Detected for query 'baaaa.example.edu.'. Src: 199.43.135.53, TXID: 23
555. Window Stats: Resp=117, IPs=1, TXIDs=117, RCode=0, PktTTL=208. TTL Stats: Mean=208.00, StdDev=0.00, Min=208, Max=20
8
ALERT: 2025-05-01 22:14:58 | Potential DNS Anomaly Detected for query 'baaaa.example.edu.'. Src: 199.43.135.53, TXID: 40
451. Window Stats: Resp=118, IPs=1, TXIDs=118, RCode=0, PktTTL=208. TTL Stats: Mean=208.00, StdDev=0.00, Min=208, Max=20
8
ALERT: 2025-05-01 22:14:58 | Potential DNS Anomaly Detected for query 'baaaa.example.edu.'. Src: 199.43.135.53, TXID: 55
043. Window Stats: Resp=119, IPs=1, TXIDs=119, RCode=0, PktTTL=208. TTL Stats: Mean=208.00, StdDev=0.00, Min=208, Max=20
8
ALERT: 2025-05-01 22:15:09 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 57883. Win
dow Stats: Resp=1, IPs=1, TXIDs=1, RCode=0, PktTTL=84084. TTL Stats: Mean=84084.00, StdDev=0.00, Min=84084, Max=84084
ALERT: 2025-05-01 22:15:09 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 14864. Win
dow Stats: Resp=2, IPs=1, TXIDs=2, RCode=0, PktTTL=-1. TTL Stats: Mean=84084.00, StdDev=0.00, Min=84084, Max=84084
ALERT: 2025-05-01 22:15:09 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 6255. Wind
ow Stats: Resp=3, IPs=1, TXIDs=3, RCode=0, PktTTL=-1. TTL Stats: Mean=84084.00, StdDev=0.00, Min=84084, Max=84084
ALERT: 2025-05-01 22:15:13 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 50896. Win
dow Stats: Resp=4, IPs=1, TXIDs=4, RCode=0, PktTTL=84080. TTL Stats: Mean=84082.00, StdDev=2.00, Min=84080, Max=84084
ALERT: 2025-05-01 22:15:13 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 11863. Win
dow Stats: Resp=5, IPs=1, TXIDs=5, RCode=0, PktTTL=-1. TTL Stats: Mean=84082.00, StdDev=2.00, Min=84080, Max=84084
ALERT: 2025-05-01 22:15:13 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 36597. Win
dow Stats: Resp=6, IPs=1, TXIDs=6, RCode=0, PktTTL=-1. TTL Stats: Mean=84082.00, StdDev=2.00, Min=84080, Max=84084
ALERT: 2025-05-01 22:15:14 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 65422. Win
dow Stats: Resp=5, IPs=1, TXIDs=5, RCode=0, PktTTL=84079. TTL Stats: Mean=84079.50, StdDev=0.50, Min=84079, Max=84080
ALERT: 2025-05-01 22:15:14 | Potential DNS Anomaly Detected for query 'purdue.edu.'. Src: 192.168.15.4, TXID: 24773. Win
dow Stats: Resp=6, IPs=1, TXIDs=6, RCode=0, PktTTL=-1. TTL Stats: Mean=84079.50, StdDev=0.50, Min=84079, Max=84080
```

However, to remove TTL, I needed another feature for the model to train on, and decided on the frequency of the DNS query name. The feature list then became: 'responses_in_window',

- 'unique_src_ips_in_window',
- 'unique_txids_in_window',
- 'response_code',
- 'qname_entropy_mean_in_window',
- 'qname_entropy_stddev_in_window'

With the feature list updated to include statistics of each query name, it allowed my model to detect and learn anomalies when they had high entropy. In the case of the Kaminsky DNS attack, the frequency of the same DNS name response indicates an attack. So, the dataset being comprised of 'good' traffic effectively becomes a white list of domains compared to the DNS name requests during an attack.



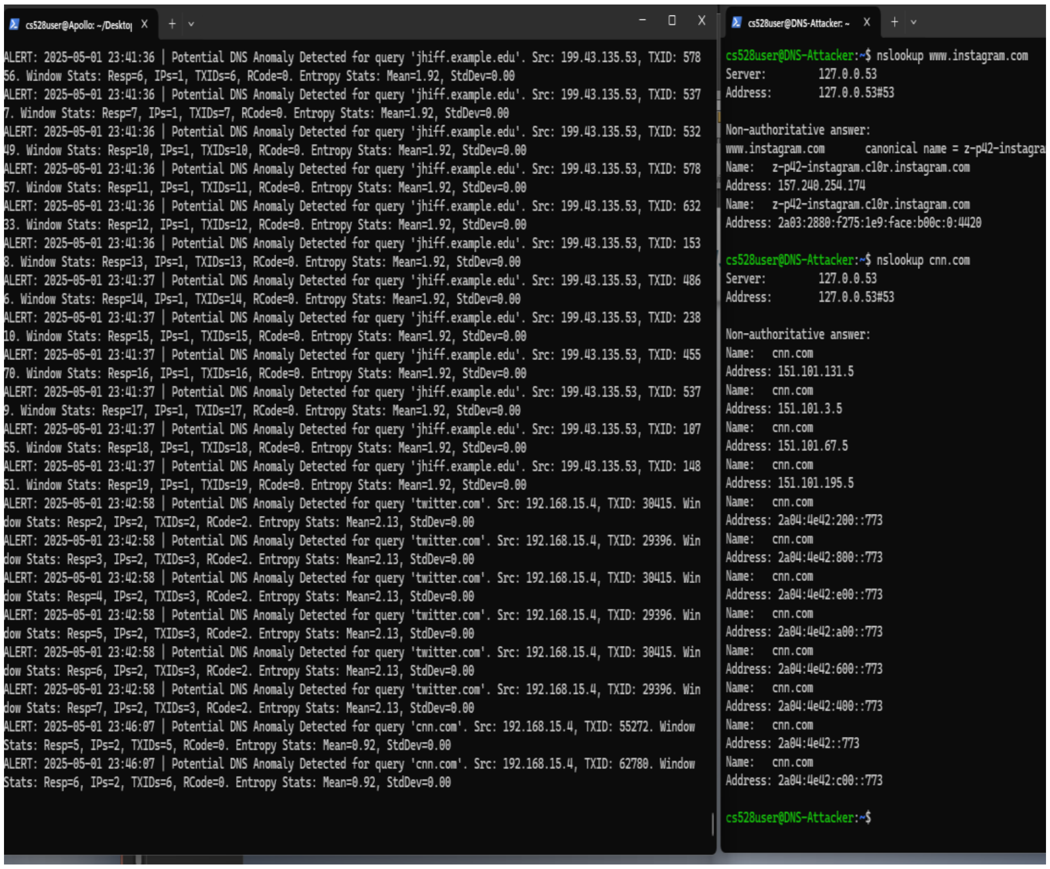
```
Training Isolation Forest model on training data...

--- Evaluating Model on Test Set ---
Confusion Matrix (Rows: True, Cols: Predicted):
           Pred Normal | Pred Attack
True Normal:       226 |          44
True Attack:        51 |          58

Classification Report:
              precision    recall  f1-score   support

  Normal (0)       0.82      0.84      0.83       270
 Anomaly (1)       0.57      0.53      0.55       109

    accuracy                           0.75       379
   macro avg       0.69      0.68      0.69       379
weighted avg       0.74      0.75      0.75       379

Saving model trained on 1512 samples to dns_anomaly_model.joblib
Saving scaler fitted on training data to dns_feature_scaler.joblib
--- Training Complete ---
```

With the new training features, we found that had similar training results as with TTL.
With contamination set to ‘0.3’:
Normal Traffic Detection: Low precision of 82% and 84% recall.
Anomaly Detection: High precision of 57%, but low recall of 53%.
Usable model, but it performs on average 55% of the time.

4 Conclusion

4.1 Results of the demo

With the changes to include the frequency of DNS names to be trained on, detecting anomalies improved in the practical cases. Consistently, domain names that were new or had a high entropy would get flagged by the agent. While ‘whitelisted’ domains were not.
An example of this: www.instagram.com was in the training set of good domains, while cnn.com wasn’t included. The left terminal is the Apollo DNS server with the agent running, and the right terminal is the attacker.



```
ALERT: 2025-05-01 23:41:36 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 578
56. Window Stats: Resp=6, IPs=1, TXIDs=6, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:36 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 537
7. Window Stats: Resp=7, IPs=1, TXIDs=7, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:36 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 532
49. Window Stats: Resp=10, IPs=1, TXIDs=10, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:36 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 578
57. Window Stats: Resp=11, IPs=1, TXIDs=11, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:36 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 632
33. Window Stats: Resp=12, IPs=1, TXIDs=12, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:36 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 153
8. Window Stats: Resp=13, IPs=1, TXIDs=13, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:37 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 486
6. Window Stats: Resp=14, IPs=1, TXIDs=14, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:37 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 238
10. Window Stats: Resp=15, IPs=1, TXIDs=15, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:37 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 455
70. Window Stats: Resp=16, IPs=1, TXIDs=16, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:37 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 537
9. Window Stats: Resp=17, IPs=1, TXIDs=17, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:37 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 107
55. Window Stats: Resp=18, IPs=1, TXIDs=18, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:41:37 | Potential DNS Anomaly Detected for query 'jhiff.example.edu'. Src: 199.43.135.53, TXID: 148
51. Window Stats: Resp=19, IPs=1, TXIDs=19, RCode=0. Entropy Stats: Mean=1.92, StdDev=0.00
ALERT: 2025-05-01 23:42:58 | Potential DNS Anomaly Detected for query 'twitter.com'. Src: 192.168.15.4, TXID: 30415. Win
dow Stats: Resp=2, IPs=2, TXIDs=2, RCode=2. Entropy Stats: Mean=2.13, StdDev=0.00
ALERT: 2025-05-01 23:42:58 | Potential DNS Anomaly Detected for query 'twitter.com'. Src: 192.168.15.4, TXID: 29396. Win
dow Stats: Resp=3, IPs=2, TXIDs=3, RCode=2. Entropy Stats: Mean=2.13, StdDev=0.00
ALERT: 2025-05-01 23:42:58 | Potential DNS Anomaly Detected for query 'twitter.com'. Src: 192.168.15.4, TXID: 30415. Win
dow Stats: Resp=4, IPs=2, TXIDs=3, RCode=2. Entropy Stats: Mean=2.13, StdDev=0.00
ALERT: 2025-05-01 23:42:58 | Potential DNS Anomaly Detected for query 'twitter.com'. Src: 192.168.15.4, TXID: 29396. Win
dow Stats: Resp=5, IPs=2, TXIDs=3, RCode=2. Entropy Stats: Mean=2.13, StdDev=0.00
ALERT: 2025-05-01 23:42:58 | Potential DNS Anomaly Detected for query 'twitter.com'. Src: 192.168.15.4, TXID: 30415. Win
dow Stats: Resp=6, IPs=2, TXIDs=3, RCode=2. Entropy Stats: Mean=2.13, StdDev=0.00
ALERT: 2025-05-01 23:42:58 | Potential DNS Anomaly Detected for query 'twitter.com'. Src: 192.168.15.4, TXID: 29396. Win
dow Stats: Resp=7, IPs=2, TXIDs=3, RCode=2. Entropy Stats: Mean=2.13, StdDev=0.00
ALERT: 2025-05-01 23:46:07 | Potential DNS Anomaly Detected for query 'cnn.com'. Src: 192.168.15.4, TXID: 55272. Window
Stats: Resp=5, IPs=2, TXIDs=5, RCode=0. Entropy Stats: Mean=0.92, StdDev=0.00
ALERT: 2025-05-01 23:46:07 | Potential DNS Anomaly Detected for query 'cnn.com'. Src: 192.168.15.4, TXID: 62780. Window
Stats: Resp=6, IPs=2, TXIDs=6, RCode=0. Entropy Stats: Mean=0.92, StdDev=0.00
```

```
cs528user@DNS-Attacker:~$ nslookup www.instagram.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
www.instagram.com       canonical name = z-p42-instagra
Name:   z-p42-instagram.c10r.instagram.com
Address: 157.240.254.174
Name:   z-p42-instagram.c10r.instagram.com
Address: 2a03:2880:f275:1e9:face:b00c:0:4420

cs528user@DNS-Attacker:~$ nslookup cnn.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   cnn.com
Address: 151.101.131.5
Name:   cnn.com
Address: 151.101.3.5
Name:   cnn.com
Address: 151.101.67.5
Name:   cnn.com
Address: 151.101.195.5
Name:   cnn.com
Address: 2a04:4e42:200::773
Name:   cnn.com
Address: 2a04:4e42:800::773
Name:   cnn.com
Address: 2a04:4e42:e00::773
Name:   cnn.com
Address: 2a04:4e42:a00::773
Name:   cnn.com
Address: 2a04:4e42:600::773
Name:   cnn.com
Address: 2a04:4e42:400::773
Name:   cnn.com
Address: 2a04:4e42::773
Name:   cnn.com
Address: 2a04:4e42:c00::773

cs528user@DNS-Attacker:~$
```

4.2 Final analysis

By detecting Kaminsky DNS attacks occurring and flagging domain requests not in the whitelisted domains, while ignoring the whitelisted domains. I believe that my project has reached most of my goals. The agent collects data from the interfaces and stores it within the SQLite database. It trains on the collected data from the database and saves it locally. Finally, the agent listens on the interface, utilizes the Isolation Forest model to detect attacks, and stores the results in the database and log file.

Although my project's agent is successful, the practical issue is how much accuracy an unsupervised model can provide. If data labeling was involved, combining it with frequency would drive the model's accuracy further, but that would require switching to a KNN-style model. In conclusion, my model provides an example of how ML/AL could be used in the future to fight against cyberattacks.

4.3 Future Steps

As my agent dumps the detected attacks into a log file and database, other systems may utilize that information for additional security purposes. An extended detection and response(XDR) or security information and event management(SIEM) system could be utilized to fill the role of an auto-response system in an enterprise setting. In my current employment, this would involve using the agent's data to provide dynamic responses to stop attacks.

5 Reference

5.1 [1] https://www.sqlite.org/whentouse.html

5.2 [2] Gaffney, Kevin P., Prammer, Martin, Brasfield, Larry, Hipp, D. Richard, Kennedy, Dan, and Patel, Jignesh M. SQLite: past, present, and future. Retrieved from https://par.nsf.gov/biblio/10390276. Proceedings of the VLDB Endowment 15.12 Web. doi:10.14778/3554821.3554842.

5.3 [3] https://dodcio.defense.gov/Portals/0/Documents/Library/ZeroTrustOverlays.pdf

5.4 [4]Falahi, T., Nasserddine, G., Younis, J. (2023). Detecting Data Outliers with Machine Learning. In Al-Salam Journal for Engineering and Technology.

5.5 [5]Lecture notes & labs of CS528

5.6 https://en.wikipedia.org/wiki/Machine_learning