Adaptive Encryption System Based on Machine Learning for Attack Detection

1 Introduction

In the context of growing cyber threats, particularly Distributed Denial of Service (DDoS) attacks, securing network traffic has become critical. This project presents an **Adaptive Encryption System** that dynamically adjusts its encryption strategy based on real-time traffic analysis using a **Support Vector Machine (SVM)** model. This system aims to strengthen security by adapting the encryption complexity in response to detected threats.

2 Project Objectives

- **Attack Detection:** Use an SVM model to classify network traffic as either benign or malicious.
- **Adaptive Encryption:** Dynamically select and configure encryption algorithms based on threat detection.
- **Secure Data Transmission:** Apply strong encryption mechanisms to safeguard sensitive information.

3 Methodology

3.1 Data Preparation and Feature Selection

Relevant features were selected to improve model performance:

```
columns_to_select = ['Protocol', 'Fwd Packet Length Min', 'Fwd
    Packet Length Mean',
                     'Flow Packets/s', 'Fwd Packets/s', 'Packet
    Length Min',
                     'Packet Length Mean', 'Down/Up Ratio', 'Avg
    Packet Size',
                     'Avg Fwd Segment Size']
X_selected = df[columns_to_select]
```

Listing 1: Feature Selection

3.2 Model Training (SVM)

The SVM model with an RBF kernel was trained to detect DDoS attacks.

```python
from sklearn.svm import SVC
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train, y_train)
```

Listing 2: SVM Training

3.3 Dynamic Key Generation

A stronger key is generated when a threat is detected:

```python
def generate_dynamic_aes_key(prediction):
    if prediction == 1:  # Attack detected
        entropy = get_random_bytes(32)
        return hashlib.sha512(entropy + b'attack').digest()[:32]
    else:
        entropy = get_random_bytes(16)
        return hashlib.sha256(entropy + b'benign').digest()[:32]
```

Listing 3: Dynamic AES Key Generation

3.4 Adaptive Hybrid Encryption

AES encryption with RSA for secure key exchange:

```python
def encrypt_data(data, aes_key, rsa_public_key):
    cipher_aes = AES.new(aes_key, AES.MODE_CBC)
    ciphertext = cipher_aes.encrypt(pad(data.encode(), AES.
    block_size))
    hmac_digest = hmac.new(aes_key, ciphertext, hashlib.sha256).
    digest()
    rsa_key = RSA.import_key(rsa_public_key)
    cipher_rsa = PKCS1_OAEP.new(rsa_key)
    encrypted_aes_key = cipher_rsa.encrypt(aes_key)
    return encrypted_aes_key + cipher_aes.iv + hmac_digest +
    ciphertext
```

Listing 4: Hybrid Encryption with AES and RSA

4 Results and Evaluation

4.1 Model Performance

The model was evaluated using a classification report:

```python
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Listing 5: Model Evaluation

4.2 Encryption Adaptability

- **Benign Traffic:** Fast encryption using AES-128.
- **Malicious Traffic:** Enhanced encryption using AES-256.

5 Advantages of the System

- **Dynamic Security:** Real-time adaptation to threats.
- **Resource Optimization:** Lightweight encryption for benign traffic.
- **Enhanced Protection:** Strong encryption for detected threats.

6 Future Improvements

- Integration with Intrusion Detection Systems (IDS)
- Real-time deployment in production environments
- Incorporation of anomaly detection algorithms
- Currently, our classification model categorizes network traffic into two classes: **attack** or **benign**. However, a significant improvement would be to refine this classification based on the **type of attack** (e.g., amplification DDoS, SYN flood, etc.). This approach would allow for a more targeted response based on the specific nature of the attack, thus enhancing **security** in a more focused manner.

  Additionally, this more detailed classification could also improve the system's **performance**. For example, lighter protection mechanisms could be applied for less complex attacks, optimizing system resources. On the other hand, for more sophisticated attacks, the system could dynamically adjust the encryption strategy in real-time to provide stronger security while maintaining optimal performance.

  This **dynamic classification** approach would not only improve detection accuracy but also allow for a more efficient and fine-tuned protection mechanism, enhancing both security and performance.

7 Conclusion

This project demonstrates how machine learning enhances cybersecurity by dynamically adjusting encryption strategies based on threat detection.

8 References

- Scikit-learn documentation: `https://scikit-learn.org/`
- PyCryptodome documentation: `https://pycryptodome.readthedocs.io/`