JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 25, No. 10, pp. 2889-2899, Oct. 2024

Check for updates

은닉 마르코프 모델 기반 고속 암호화 네트워크 트래픽 탐지 시스템

염 녕[1] · 추 영 열[2*]
[1]동명대학교 컴퓨터미디어공학과 석사과정
[2]동명대학교 컴퓨터공학과 교수

Fast Encrypted Network Traffic Detection System Based on Hidden Markov Models

Ning Yan[1] · Young-Yeol Choo[2*]

[1]Master's Course, Dept. of Computer and Media Engineering, Tongmyong University, Busan 48520, Korea
[2]Professor, Dept. of Computer Engineering, Tongmyong University, Busan 48520, Korea

[요 약]

순환신경망기반의 딥 패킷탐지 방법은 원시 패킷에서 특징을 자동으로 추출하고 응용 프로토콜을 식별할 수 있지만, 일정한 정확도를 유지하려면 많은 양의 훈련 데이터와 계산 리소스가 필요하고, 모델의 해석이 좋지 않아 실제 환경에 적용하기가 매우 어렵다. 이 문제를 해결하기 위해 히든 마르코프 모델을 기반으로 한 네트워크 트래픽 탐지 알고리즘을 제안한다. 암호화된 트래픽을 게놈 시퀀스로 간주하여 정상적인 네트워크 동작을 모델링하고 이상 징후를 탐지하는 새로운 알고리즘이 제안되어 암호화된 악성 트래픽을 효과적으로 식별할 수 있다. 제안한 시스템은 기존 알고리즘과 비교하여 95% 이상의 정확도 또한 메타데이터를 최대 MTU까지 패딩하고 난독화되거나 유효하지 않은 패킷을 삽입함으로써 패딩이 큰 부분 을 초과하더라도 50% 이상의 정확도를 보장 유지하며 난독화에 강하다는 것을 보였다. 본 연구는 암호화 트래픽 탐지에 대한 새로운 접근 방식을 다양한 분야에서도 적용될 것이다.

[Abstract]

Deep packet detection methods based on recurrent neural networks can automatically extract features from raw packets and identify application protocols. However, they require a large amount of training data and computational resources to maintain accuracy, and the interpretability of the model is poor, making its real-world applications challenging. To address this issue, we propose a network traffic detection algorithm based on hidden Markov models. By treating encrypted traffic as genome sequences, this new algorithm models normal network behavior and detects anomalies, effectively identifying encrypted malicious traffic. The proposed system achieves over 95% accuracy compared to existing algorithms. Additionally, it is robust against obfuscation techniques, such as padding metadata up to the maximum MTU and inserting obfuscated or invalid packets, maintaining accuracy above 50% even when padding occupies a significant portion. This research offers a novel approach to encrypted traffic detection, which is applicable across various fields.

색인어 : 암호화 기술, 보안소켓계층/전송계층보안, 심층 패킷 검사, 숨겨진 마르코프 모델, 네트워크 기술
Keyword : Encryption Technology, SSL/TLS, Deep Packet Inspection, Hidden Markov Model, Network Technologies

http://dx.doi.org/10.9728/dcs.2024.25.10.2889

Received 11 September 2024; **Revised** 21 October 2024
Accepted 21 October 2024

***Corresponding Author; Young-Yeol Choo**
Tel: ※ 개인정보 표시제한
E-mail: yychoo@tu.ac.kr

Ⅰ. Introduction

Encryption technology has boosted online security, but also created opportunities for malicious actors like advanced persistent threats (APTs)[1] to exploit encrypted traffic. The rise of APT attacks and limitations of traditional methods like deep packet inspection have highlighted the need for new detection techniques. Machine learning, particularly hidden markov models (HMMs), shows promise in this area. Inspired by genome sequence comparison, HMMs can model statistical characteristics of encrypted traffic to detect deviations from normal patterns, potentially revealing APT activity.

While hypertext transfer protocol secure (HTTPS) is the most common encryption protocol, it's also frequently used in phishing attacks and by botnets like Mirai[2]. This, along with the fact that botnets are evolving to be less predictable[3], makes it difficult to rely on specific protocol analysis for detection.

Traditional security software struggles to keep up with new virus strains and encrypting malware. This leaves government agencies and other critical assets vulnerable. There's a clear need for dynamic detection of malicious encrypted software. Existing research on network traffic detection mostly focuses on unencrypted traffic, which differs significantly from encrypted traffic due to changes in request headers, obfuscation techniques (SOCKS5 within HTTPS tunnels)[4], and multiple encryption layers.

Machine learning methods like C4.5[5], k-means, and support vector machine (SVM)[6] have been used for encrypted traffic detection, but they often rely on accurate data labeling. Semi-supervised learning approaches like clustering and HMMs offer alternatives, but still face challenges in identifying different types of malware. Some studies have achieved impressive results, like the random forest algorithm by Anderson will be show in this paper 4-2 part. But it only distinguishes between malware and benign traffic, not different malware types.

This paper proposes an HMM-based approach for detecting encrypted traffic of packet-swapped malware, drawing inspiration from gene sequence detection in biology. The approach aims to address the challenges of detecting evolving malware and achieving high accuracy in real-world data. It also leverages Go and the AVX512 instruction set for improved efficiency[7],[8].

Ⅱ. Related works

2-1 Hidden Markov Model

Hidden markov model[9] is a statistical model for describing a Markov process with unknown parameters. Its core principles are: (1) the next state depends only on the current state, and (2) the output depends only on the current state. Originally used in biological sequence analysis, HMM has been applied in various fields like speech recognition and text classification.

In a simple example with letters a and b, HMM starts with an initial state, transitions to new states based on probabilities, and emits a symbol (like the letter “a”) based on emission probabilities associated with the current state. This process repeats until a final state, resulting in a sequence of hidden states and observed symbols.

The “hidden” in HMM refers to the fact that only the symbol sequence is directly observable, while the hidden state sequence follows a first-order Markov chain.

2-2 Profile HMM

As shown in Fig. 1, Profile HMM[10] extends HMM with two states: Insert, allowing insertion of states between any two states, and Delete, enabling state deletion.

Profile hidden markov models (Profile HMMs) enhance traditional HMMs by incorporating Insert and Delete states to better handle sequence variations like insertions or deletions often found in biological or network data. Unlike HMMs, Profile HMMs leverage multiple sequence alignments to record positional information of states, enabling them to model specific locations where insertions or deletions are likely to occur. Profile HMMs offer two key advantages over HMMs: utilizing positional information of observed sequences and allowing null transitions to match sequences with insertions or deletions.

In Fig. 1, Insert (I) states allow symbol insertion

before Match (M) or Delete (D) states, while Delete (D) states allow symbol deletion. The model transitions between states based on symbol matches and mismatches.

This paper applies profile HMMs to network security by constructing encrypted traffic as gene sequences and using homologous gene search algorithms to detect malicious attacks within the encrypted traffic.

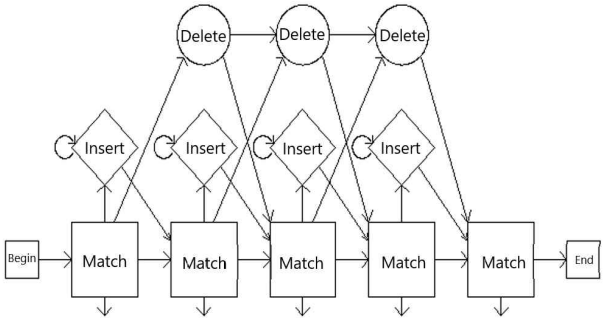

Fig. 1. Overview of the profile HMM

2-3 Problems with Encrypted Traffic Identification using HMM

Existing HMM-based methods for encrypted traffic analysis often fall short due to their reliance on simplified models and limited state representation. To address this, we propose a series of enhancements:

1) Introduce a two-dimensional feature combining packet length and message type during the handshake process to expand the number of Markov states. Incorporate a second-order Markov model to capture more complex dependencies in traffic patterns.

2) Utilize packet sizes during data transmission to construct an HMM model, refining emission probabilities by considering the correlation between adjacent packet sizes.

These enhancements aim to improve the accuracy and effectiveness of HMM-based encrypted traffic analysis by providing a more comprehensive and discriminative model.

2-4 The Markov Chain Model based on the Handshaking Process

The secure sockets layer/transport layer security (SSL/TLS)[11] handshake process, a series of message exchanges to establish a secure connection, can be modeled as a Markov chain. Each state represents a handshake stage, with transitions governed by protocol rules, and observed message features as emissions. This approach is protocol agnostic, adaptable to various SSL/TLS versions, and feature-rich, capturing diverse application behaviors.

As show in Fig. 2 and Table 1, our HMMS structured the message content of the protocol according to the different intervals of the TLS handshake. Shows the contents of the protocol messages at different times of the TLS handshake. This model can be used for encrypted traffic classification, anomaly detection, and protocol fingerprinting. However, challenges like state space complexity and adversarial evasion need to be addressed.

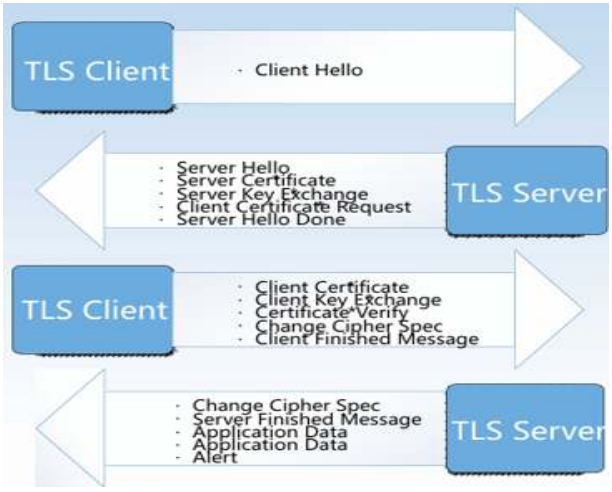

Fig. 2. Markov chain model based on and shaking process

Table 1. Markov chain model based on handshaking process

Plain Text	SSL/TLS Handshake Characteristics
20	Change Cipher Spec
21	Alert
22:02	Server Hello
22:11	Certificate
22:12	Server Key Exchange
22:13	Certificate Request
22:14	Server Hello Done
22:17	Encrypted Handshake Message
22:18	New Session Ticket
22:20	Finished
23	Application Data

2-5 The profile HMM Employed in Conjunction with Encrypted Traffic

Inspired by Profile HMMs, this section integrates packet-level features into a model for encrypted traffic analysis. Both traffic sequences and gene sequences evolve over time, and key subsequence within them can reveal their overall classification.

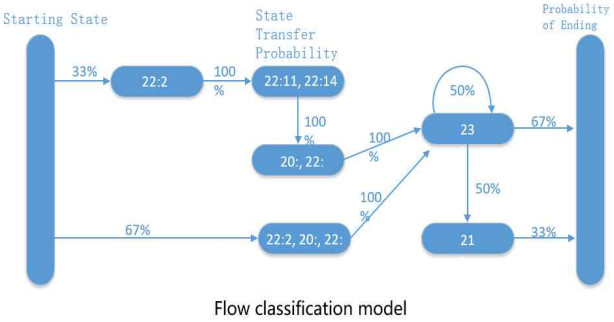

Fig. 3. Markov chain model implementation example

Similar to how core gene fragments in homologous gene sequences can be traced back to their gene families, key subsequence in malicious traffic retain attack characteristics. This paper aims to analyze these “key genes” to detect malicious attacks.

To account for packet sequence variations within the same protocol, the model includes Insert and Delete states for each position. Insert states represent duplicate or retransmitted packets, while Delete states signify packet loss.

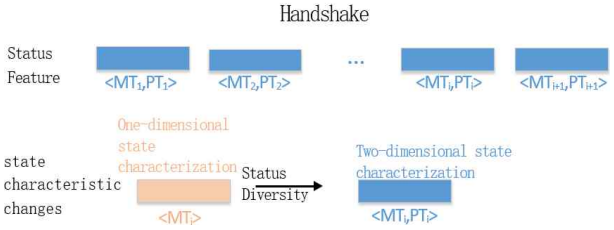

Fig. 4. Markov chain state characteristics diversification

Each chain in the Profile HMM is associated with a specific packet, and states emit symbols with position-specific probability distributions. Fig. 3 and Fig. 4 illustrate the routine modeling process for encrypted traffic detection based on profile hidden Markov models (HMMs). Slicing and dicing the communication process during a TLS handshake allows us to understand exactly what is going on at each step of the TLS communication process. This removes unnecessary packets thus retaining exactly the data we need.

Ⅲ. HMM based Flow Processing

3-1 Core Work

The objective of this study is to enhance the hidden markov model based on the data transmission process as show in Fig. 5. ADPT is a packet length of application data. A DPT Represents the length of an application packet and can be viewed as a state in the data transmission process. state transfer Represents the process of transferring the packet length from one state to another. Launch probability Optimization It may refer to some optimization algorithm that adjusts the probability of state transfer to better fit the actual network traffic data.

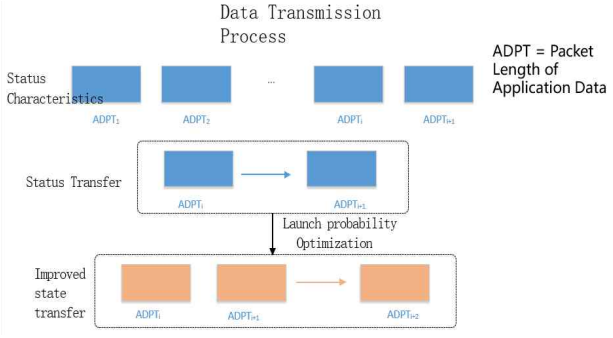

Fig. 5. Improvement of HMM model based on data transmission process

Malicious traffic often has a distinct temporal signature, which attackers try to disguise through delayed transmission and data obfuscation[12]. However, these tactics also introduce overheads that create hidden patterns, exploitable for detection.

We capture and filter traffic to obtain packet sequences, then symbolize them to reduce observable states in the HMM, enhancing robustness. This allows us to create a model with a single gene sequence representing all sequences from the same malicious sample. Multiple sequences can be used as training samples for better efficacy. Detection involves transforming the test sample into a symbolic sequence and comparing it against known sequences.

3-2 HMM Launch Probability Optimization: A Specific Description

If a detection model performs whole-sequence matching on pre-established malicious samples and sequences awaiting detection, it may misidentify non-key subsequence and miss actual key subsequence, making it ineffective against attacks with obfuscated packets. However, the proposed model segments the sequence into individual subsequence, allowing it to match key subsequence regardless of the order in which the attacker sends packets.

This enables detection of malicious samples despite obfuscation:

$$B = \{b \downarrow im\} \tag{1}$$

Where "B" is the number of Markov states and "b" and "im" are the packet length and message type of the handshake process, respectively.

The probability of a successful launch has been enhanced:

$$B = \{b \downarrow im(o \downarrow t-1)\} \tag{2}$$

Specifically:

$$b \downarrow im(o \downarrow t-1) = P(v \downarrow m / q \downarrow i, o \downarrow t-1) \tag{3}$$

The probability of a successful firing is enhanced when the last observed feature state is $o \downarrow t-1$ and the hidden state is $q \downarrow i$. This probability is dependent on the value of the observation feature.

3-3 Block Diagram of the Overall System Structure

Before training each HMM, the training set undergoes multiple order alignment, aligning homologous parts of sequences by inserting special symbols. This facilitates the discovery of common subset sequences, making the model easier to train and enhancing the hit rate of analogous subsequence, preventing the model from being trapped in local optima. The full structure is as show in Fig. 6.

① Preprocessing: Capture SSL/TLS traffic: Collect a large amount of SSL/TLS traffic data. Traffic Statistics and Feature Extraction: Perform statistical analysis of the captured traffic to extract attributes that characterize the traffic, such as packet length, interval time, protocol type, etc.

② Multi-order Alignment: Before training each HMM model, the sequences in the training set are multi-order aligned. By inserting special symbols, the homologous parts of different sequences are aligned so that common subsequences can be found. Significance of Alignment: The purpose of alignment is to improve the efficiency and accuracy of model training. Through the alignment, it can make the model easier to learn the commonality between sequences, so as to improve the hit rate of similar subsequences and avoid the model falling into the local optimum.

③ Model Training: Multiple HMM models: For each feature subset, train one HMM model, which can effectively model sequence data, capturing transfer probabilities and firing probabilities between sequences.

④ Combining Markov models: In addition to HMM models, Markov models are also combined. Markov models capture the transfer relationships between neighboring states in a sequence.

⑤ Multiple classifiers: Multiple trained HMM models and Markov models are used as base classifiers.

⑥ Weighted Integration: Integrate the base classifiers by assigning different weights to each of them to form a weighted integrated classifier.

⑦ Final Classification: The weighted integrated classifier classifies the new test samples and outputs the final classification result.

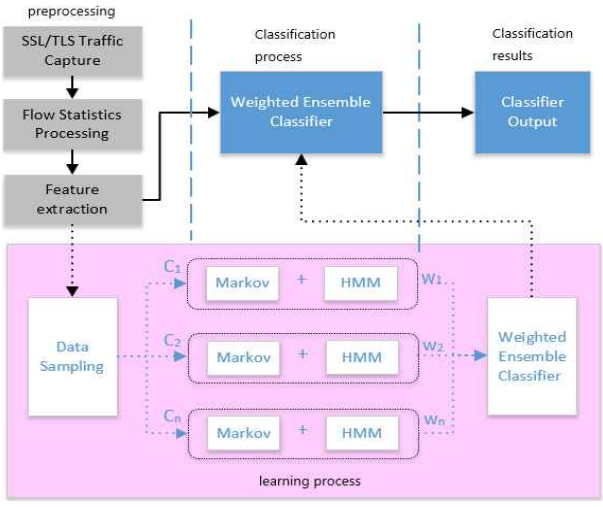

Fig. 6. Overall architecture of weighted ensemble classifier

3-4 Data Preparation

This experiment employs the open-source dataset USTC-TFC2016[12] and SSL Blacklist[11], in addition to traffic obtained following authorization on a public network to assess the efficacy of the model.

The USTC-TFC2016 dataset, created by the University of Science and Technology of China, is a benchmark for encrypted traffic classification and analysis. It contains real-world encrypted traffic traces with labeled application types, making it suitable for supervised learning. This dataset captures the nuances of real-world traffic, including variations in packet sizes, timings, and content. However, it's important to note that encrypted traffic patterns are dynamic and adversaries can use evasion techniques, highlighting the need for continuous model updates and adaptation.

The SSL Blacklist (SSL BL) dataset collects malicious SSL-related certificates, domain names, and other data. In this study, Suricata's rule sets and the SSL BL dataset are used to identify malware, resulting in 1,605 malicious samples across 16 malware families and as show in Table 2.

Table 2. Virus family schematic

Malware Family	Sample Size	Sample Flow Data Volume
IcedID	201	8251
CobInt	33	4687
Gozi	185	7671
OrcusRAT	153	6972
Adwind	52	4937
AsyncRAT	47	5640
Ostap	137	6839
TA505	38	4468
FindPOS	177	7490
TinyNuke	115	6003
ServHelper	25	3528
PsiXBot	94	5586
AZORult	61	5309
PredatorStealer	103	7195
NetWire	47	4115
PandaZeus	137	6898

3-5 Data Processing and Modelling

To analyze malware behavior, packet data is symbolized into a matchable sequence. First, malicious sample data is serialized using MapReduce to aggregate encrypted traffic within a timeframe.

For each packet, its size and direction are extracted (3000 possibilities). To avoid excessive symbols and poor model performance, a grouped symbolization approach divides possibilities into 3000/capacity groups.

After symbolizing individual packets, all packets are processed. Data streams exceeding a set length are truncated, while shorter ones are padded. This completes preprocessing as show in Fig. 7.

```
1  Seq = {}, Out = −1, Size = 0;
2  for packet p ∈ P_uncry do
3      if Size < L then
4          if p is an outgoing packet then
5              p → length = p → length × Out;
6          else
7              p → length = p → length × In;
8          Seq = Seq + Symbolize(p → length);
9          Size = Size + 1;
10 while Seq → length < L do
11     Seq = Seq+"-";
12     Seq → length = Seq → length + 1;
13 return Seq
```

Fig. 7. The encrypted traffic sequence symbolization algorithm

In the algorithm, Symbolize (p→length) signifies that, in accordance with the capacity that has been established, the packet of a given length is transformed into the symbol of the corresponding group. In the algorithm, Symbolize (p→length) signifies the conversion of a packet of length into a symbol of the corresponding group, contingent upon the capacity that has been set. To illustrate, a packet of one byte in size is assigned to the range + 0, + 1, and so forth.

The capacity group is designated as + 1, + 2, + 3, and so forth, with the group identifier assigned in the order of aa, ab, ac, and so on. In the event that the capacity is equal to 10, the corresponding group is designated as “fl.” This implies that the packet has successfully traversed the specified group.

Conversely, if the capacity is equal to 10, the corresponding packet is designated as “fl.” This signifies that the packet has been successfully mapped

to the character “fl” following the process of symbolization.

3-6 AVX512 Accelerated Go Reasoning

Advanced vector extensions 512 (AVX512), Intel's latest single instrument multiple data (SIMD) instruction set, can process 512 bits of data per cycle, offering potential acceleration for vector computations in Go-based deep learning systems. However, Go compiler doesn't automatically generate SIMD instructions. While c2goasm converts Intel assembly to Go assembly, it doesn't handle AVX512 instructions.

To utilize AVX512 in Go, we developed autogoassmb, a toolkit compiling C to Go assembly. It directly converts C source code to Go assembly, eliminating the need for manual compilation and function definition. This enhances functionality compared to c2goasm, enabling AVX512 usage in Go.

Detailed thoughts are as follows:

1) Coding Machine Code

Go assembly provides three instructions for representing binary machine code:

BYTE represents one byte of binary data. WORD represents two bytes of binary data. LONG means four bytes of binary data. If the instruction machine code length is exactly a multiple of two, for example as show in Fig. 8.

```
498:   41 83 e1 fc                   and    $0xfffffffc,%r9d
```

Fig. 8. ASM CODE from Go

Can be converted to as show in Fig. 9.

```
LONG $0xfce18341
```

Fig. 9. ASM CODE optimization from Go

But if the length is not a multiple of two, for example as show in Fig. 10.

```
4b2:   62 f1 7c 48 10 47 01           vmovups 0x40(%rdi),%zmm0
```

Fig. 10. ASM CODE not enough from Go

It would require a combination of three instructions to represent as show in Fig. 11. Note that the byte order of the instruction encoding and the byte order of the objdump output are reversed.

```
LONG $0x487cf162; WORD $0x4710; BYTE $0x01 // vmovups  64(%rdi), %zmm0
```

Fig. 11. ASM CODE not enough optimization from Go

2) Function Definitions and Arguments

In C assembly, if a function has no more than 6 arguments, then the arguments are saved in registers and passed to the function in the order of %rdi, %rsi, %rdx, %rcx, %r8, and %r9. However, in Go assembly, the arguments of the function are placed in memory starting from the address saved in the FP registers, which requires us to move the arguments from the memory to the registers.

The “_mm512_mul_to” function has four arguments, so it is necessary to move the four arguments well before the function starts as show in Fig. 12.

The function definition consists of three parts: the TEXT keyword, the name starting with - and ending with (SB), and finally the parameter memory size of 32 bytes. There is no information about the number of parameters in assembly, you need to get it from the C function definition.

```
TEXT ·_mm512_mul_to(SB), $0-32
  MOVQ a+0(FP), DI
  MOVQ b+8(FP), SI
  MOVQ c+16(FP), DX
  MOVQ n+24(FP), CX
```

Fig. 12. ASM CODE not enough optimization from Go

```
TEXT ·_mm512_mul_to(SB), $0-32
 MOVQ a+0(FP), DI
 MOVQ b+8(FP), SI
 MOVQ c+16(FP), DX
 MOVQ n+24(FP), CX
 BYTE $0x55                    // pushq %rbp
 WORD $0x8948; BYTE $0xe5 // movq %rsp, %rbp
 LONG $0xf8e48348              // andq $-8, %rsp
 LONG $0x0f498d4c              // leaq 15(%rcx), %r9
 WORD $0x8548; BYTE $0xc9 // testq %rcx, %rcx
 LONG $0xc9490f4c              // cmovnsq %rcx, %r9
 LONG $0x04e9c149              // shrq $4, %r9
 WORD $0x8944; BYTE $0xc8 // movl %r9d, %eax
 WORD $0xe0c1; BYTE $0x04 // shll $4, %eax
 WORD $0xc129                  // subl %eax, %ecx
 WORD $0x8545; BYTE $0xc9 // testl %r9d, %r9d
 JLE  LBB3_6
 LONG $0xff418d41              // leal -1(%r9), %eax
 WORD $0x8945; BYTE $0xc8 // movl %r9d, %r8d
 LONG $0x03e08341              // andl $3, %r8d
 WORD $0xf883; BYTE $0x03 // cmpl $3, %eax
 JB   LBB3_4
 LONG $0xfce18341              // andl $-4, %r9d
 WORD $0xf741; BYTE $0xd9 // negl %r9d
```

Fig. 13. Final assembly code in Go

3) Redirection Jump Instruction

In x86 jump instructions jump to absolute addresses and direct coding of jump instructions will not work. Therefore, jump instructions need to be converted to jump instructions in Go assembly. Converting labels: labels in Go assemblies cannot start with to start with, so it is necessary to remove the removed; Conversion commands: Go assembly jump commands are uppercase. After the above three-step process, you end up with Go assembly code as show in Fig. 13.

3-7 Evaluation Methodology

A tenfold cross-validation approach is employed, whereby each malicious sample's generated flow is subjected to randomization and subsequently partitioned into ten equal segments, each integrated into a distinct, segmented dataset. In each training iteration, one of the datasets is designated as the validation set, while the remaining nine serve as the training set for the detection model. Furthermore, during the training phase, 100 symbolic sequences are randomly selected from each sample for model generation. This is done with the intention of augmenting the uncertainty in the training phase and challenging the model's generalization capabilities. This process of alternating between training and validation sets is repeated ten times, with the final detection rate calculated as the average of these ten iterations.

To ascertain the impact of various parameters and experimental conditions on the results, we have designed multiple sets of comparative experiments. The experiments vary the sequence length (L) and assess the impact of utilizing multiple sequence alignments to preprocess the mapped symbolic sequences of data packets. Furthermore, we examine the efficacy of our method for detecting unencrypted traffic and compare it with established methods to assess the applicability of our approach.

To provide a comprehensive evaluation of the experimental outcomes, the following metrics are employed: accuracy, precision, recall, F-measure (a harmonic mean of precision and recall), and the area under the curve (AUC) of the receiver operating characteristic (ROC). The following labels are assigned:

True positive (TP) samples are those that are actual positives and predicted as such. False negative (FN) refers to samples that are actual positives but have been predicted as negatives. False positive (FP) refers to samples that are actual negatives but predicted as positives. True negative (TN) refers to samples that are actual negatives and predicted as negatives.

The aforementioned metrics can be computed from these labels.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{6}$$

$$\text{Acuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{TN} + \text{FP}} \tag{7}$$

$$\text{Ecificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{8}$$

It is important to note that in the experimental measurement of F-measure, the parameter α is set to 1, which corresponds to the actual composite evaluation index, F1. Furthermore, the accuracy value should be obtained by calculating the area under the ROC curve.

Ⅳ. Experiments

4-1 System Environments

In order to make the experimental results more precise, we tested them in numerous real environments, and because of the huge amount of data, we had to use very powerful experimental environments as show in Table 3.

Table 3. Testing environments

Resources	Specification
CPU	• Intel Xeon 9480×2
Language	• A100×2
OS	• Ubuntu 22.04 • OpenWrt 23.05
Libraries	• Python 3.6 • GCC 8.5 • Tensorflow 1.13.1 • Pytorch 2.0

4-2 Comparison of the Effectiveness of Algorithms

To evaluate our method against established machine learning approaches, we use the complete dataset for accuracy assessment. As some algorithms are designed for binary classification, we categorize labels as malicious or benign to distinguish between malware and normal traffic. We selected three common classification algorithms: random forest (RF)[15], support vector machine (SVM)[13], and multilayer perceptron (MLP)[14].

RF is an ensemble learning method based on decision trees, combining predictions of multiple trees to improve classification and address overfitting. SVM finds a hyperplane in a high-dimensional feature space to separate the two categories, maximizing the distance between them. MLP is a feed-forward artificial neural network with multiple layers, using a nonlinear activation function and back propagation to minimize loss and build a classifier model.

In order to facilitate the understanding, we use 12 software commonly used in the real world to analyse the data, as shown in Fig. 14, the average accuracy rate obtained by the ten-fold cross-validation method is 93%, and the experimental results prove that the multi-terminal alignment strategy proposed in the third part of this paper has a more obvious enhancement for traffic detection, but the multi- terminal alignment is not too friendly in terms of computational power, and it will very much occupy the CPU resources when the data sequences are too long.

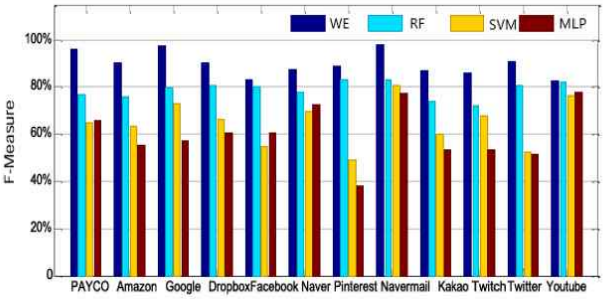

Fig. 14. Comparison with three algorithms

Therefore, in this experiment, we fix the data length to 500 for testing, and according to the experimental results, we prove that this method has strong effectiveness. The detection model, by considering the intrinsic connection between subsequence, can match corresponding key subsequence regardless of the packet order, effectively capturing malicious traffic hidden within obfuscated packets.

4-3 About Circumventing Detection

Our biology-inspired approach is protocol-independent and doesn't rely on metadata, making it applicable to any encrypted traffic detection. Experiments demonstrate its feasibility, effectively handling random sequence population due to high packet length dependence and evasion techniques. We also observed numerous duplicate data streams in real data, mainly from MTU. Experiments with filling metadata to maximum MTU length and inserting obfuscated or invalid packets show the algorithm maintains high accuracy (over 65%) even with more than half filling, demonstrating strong anti-evasion capabilities as show in Fig. 15.

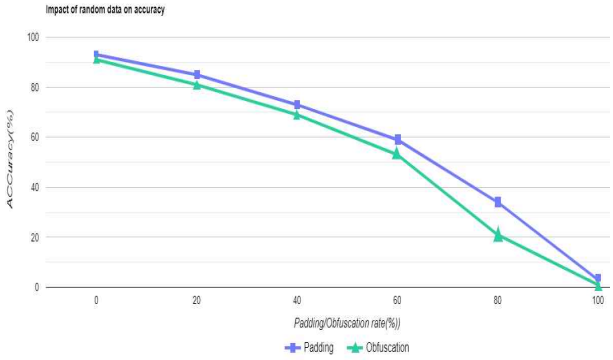

Fig. 15. Impact of random data on accuracy

4-4 Effectiveness of Multi-Algorithm Avoidance Detection

Anti obfuscation is an important topic, modern hackers commonly transmit malicious traffic along with normal traffic to avoid detection by firewalls, here we replace the normal traffic with randomized data. The use of randomly populated data is a better way to demonstrate the robustness of our system to traffic detection, as presented in Articles 4-2 and 4-3, where we compare it with “RF”, “SVM”, and “MLP” in terms of accuracy, and also with several algorithms in terms of obfuscation resistance. Next, we also compare the anti-confusion properties of several algorithms.

As shown in Fig. 16, without exception, none of the other three algorithms are able to detect effectively

with more than 40% padding, and conjecturally, the performance of the RF algorithms depends heavily on the quality of the input features. When obfuscation attacks introduce a lot of noise or change the feature distribution, the performance of the RF algorithm is significantly affected. Whereas algorithms such as SVMs usually establish linear decision boundaries, obfuscation attacks can be targeted to generate antagonistic samples that are very close to legitimate samples in the feature space, thus deceiving the classifier. And although neural networks such as MLP have strong nonlinear representation, they are prone to fall into local minima, resulting in models that are less robust to adversarial samples. And our algorithm effectively circumvents the shortcomings of other algorithms, so it maintains an effective detection rate.

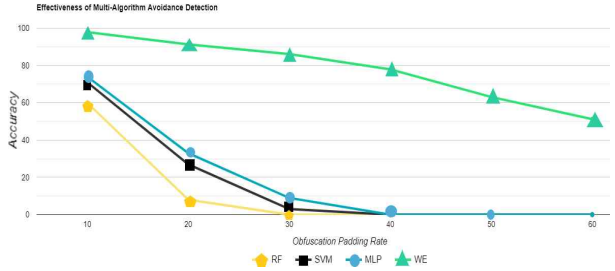

Fig. 16. Effectiveness of multi-algorithm avoidance detection

4-5 Go AVX512 Performance Comparison

Finally, we compare the performance of the non-vectorised and vectorised functions. Firstly, the vectorised function takes significantly less time than the non-vectorised function, especially when the vector length is 128; Then, AVX512 has a slight performance improvement over AVX2. The comparison as show in Fig. 17.

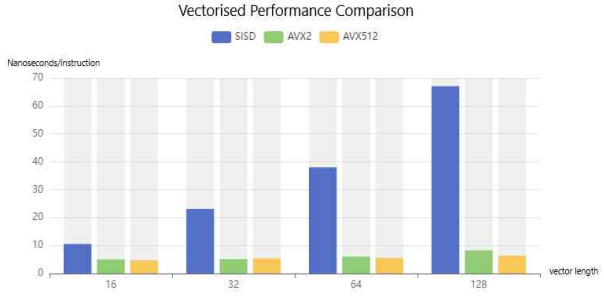

Fig. 17. Performance comparison of vectorised implementations

V. Conclusion

This paper proposes a novel approach for encrypted traffic analysis inspired by biological sequence analysis and profile hidden markov models (HMMs). By modeling packet-level features as gene sequences and identifying key subsequence, the method effectively detects malicious traffic within encrypted communications. The approach is protocol-independent, adaptable to evolving malware, and demonstrates strong resilience against evasion techniques. Experimental results on real-world datasets validate its high accuracy and potential for practical application in network security.

References

[1] O. Yaacoubi, "The Rise of Encrypted Malware," *Network Security*, Vol. 2019, No. 5, pp. 6-9, May 2019. https://doi.org/10.1016/S1353-4858(19)30059-5

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, ... and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of the 26th USENIX Security Symposium*, Vancouver, Canada, pp. 1093-1110, August 2017.

[3] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection," in *Proceedings of the 17th USENIX Security Symposium*, San Jose: CA, pp. 139-154, July-August 2008.

[4] R. Bortolameotti, T. van Ede, M. Caselli, M. H. Everts, P. Hartel, R. Hofstede, ... and A. Peter, "DECANTeR: DEteCtion of Anomalous outbouNd HTTP TRaffic by Passive Application Fingerprinting," in *Proceedings of the 33rd Annual computer security applications Conference (ACSAC '17)*, Orlando: FL, pp. 373-386, December 2017. https://doi.org/10.1145/3134600.3134605

[5] R. Bar-Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime Classification for Encrypted Traffic," in *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA 2010)*, Ischia Island, Italy, pp. 373-385, May 2010. https://doi.org/10.1007/978-3-642-13193-6_32

[6] B. Anderson and D. McGrew, "Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity," in *Proceedings of the 23rd ACM SIGKDD International Conference on*

Knowledge Discovery and Data Mining (KDD '17), Halifax, Canada, pp. 1723-1732, August 2017. https://doi.org/10.1145/3097983.3098163

[7] M. Andrawos and M. Helmich, *Cloud Native Programming with Golang: Develop Microservice-Based High Performance Web Apps for the Cloud with Go*, Birmingham, UK: Packt Publishing, 2017.

[8] J. M. Cebrian, L. Natvig, and M. Jahre, "Scalability Analysis of AVX-512 Extensions," *The Journal of Supercomputing*, Vol. 76, No. 3, pp. 2082-2097, March 2020. https://doi.org/10.1007/s11227-019-02840-7

[9] S. R. Eddy, "Hidden Markov Models," *Current Opinion in Structural Biology*, Vol. 6, No. 3, pp. 361-365, June 1996. https://doi.org/10.1016/S0959-440X(96)80056-X

[10] T. J. Wheeler and S. R. Eddy, "nhmmer: DNA Homology Search with Profile HMMs," *Bioinformatics*, Vol. 29, No. 19, pp. 2487-2489, October 2013. https://doi.org/10.1093/bioinformatics/btt403

[11] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password Interception in a SSL/TLS Channel," in *Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO 2003)*, Santa Barbara: CA, pp. 583-599, August 2003. https://doi.org/10.1007/978-3-540-45146-4_34

[12] B. Wang, Y. Su, M. Zhang, and J. Nie, "A Deep Hierarchical Network for Packet-Level Malicious Traffic Detection," *IEEE Access*, Vol. 8, pp. 201728-201740, 2020. https://doi.org/10.1109/ACCESS.2020.3035967

[13] H. Wang and D. Hu, "Comparison of SVM and LS-SVM for Regression," in *Proceedings of 2005 International Conference on Neural Networks and Brain*, Beijing, China, pp. 279-283, October 2005. https://doi.org/10.1109/ICNNB.2005.1614615

[14] H. Taud and J. F. Mas, Multilayer Perceptron (MLP), in *Geomatic Approaches for Modeling Land Change Scenarios*, Cham, Switzerland: Springer, ch. 27, pp. 451-455, 2018. https://doi.org/10.1007/978-3-319-60801-3_27

[15] S. J. Rigatti, "Random Forest," *Journal of Insurance Medicine*, Vol. 47, No. 1, pp. 31-39, January 2017. https://doi.org/10.17849/insm-47-01-31-39.1

염녕(Ning Yan)

2023년 2월 : 동명대학교 컴퓨터공학과 학사 (공학사)

2023년 3월~현 재: 동명대학교 컴퓨터미디어공학과 석사과정

※관심분야 : 네트워크 보안, IoT, 무선망 기술, 통신기술

추영열(Young-Yeol Choo)

1988년 2월 : 서울대학교 제어계측공학과 (공학석사)

2002년 2월 : 포항공대 컴퓨터공학과 (공학박사)

1988년~2002년: 포스코 기술연구소 책임연구원

2005년~2005년: 독일 Fraunhofer IESE Visiting Scientist

2002년~현 재: 동명대학교 컴퓨터공학과 교수

※관심분야 : 컴퓨터 네트워크, IoT, 네트워크 보안, 실시간 시스템