applied sciences

MDPI

Article

An Exploit Traffic Detection Method Based on Reverse Shell

Yajing Liu, Ruijie Cai *, Xiaokang Yin and Shengli Liu

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China
* Correspondence: wsxcrj@163.com

Abstract: As the most crucial link in the network kill chain, exploiting a vulnerability is viewed as one of the most popular attack vectors to get the control authority of the system, which is dangerous for legal users. Therefore, an effective exploit traffic detection method is urgent. However, current methods are almost based on pattern matching, invalid for encrypted traffic. To address this problem, we propose a reverse shell-based exploit traffic detection method, ETDetector. Our key insight is that the reverse shell attack often coexists with vulnerability exploitation as one of the most popular exploit behaviors. So, we first extract the fusion information feature from original features, such as the packet delay sequence, as input of a decision tree model to identify reverse shell traffic in the shellcode execution stage. Then, we trace suspicious traffic in the shellcode delivery stage by reconstructing the session relationship of the two stages above. Compared with Blatta, using a cyclic neural network to detect early exploit traffic, the detection rate of ETDetector is increased by 50% and valid for encrypted exploit traffic. In addition, we propose a traffic stratification method based on a bisecting K-means algorithm, which can intuitively show the traffic communication behavior and improve the interpretability of ETDetector.

Keywords: exploit; reverse shell; malicious traffic detection; behavior features

check for updates

Citation: Liu, Y.; Cai, R.; Yin, X.; Liu, S. An Exploit Traffic Detection Method Based on Reverse Shell. *Appl. Sci.* **2023**, *13*, 7161. https://doi.org/10.3390/app13127161

Academic Editor: Luis Javier Garcia Villalba

Received: 27 February 2023
Revised: 10 June 2023
Accepted: 11 June 2023
Published: 15 June 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Security vulnerability has become a crucial threat to cyberspace. Attackers can remotely exploit the vulnerability to steal information, upgrade permissions, or even directly control the operating system. For example, the Federal Bureau of Investigation (FBI) and the Cybersecurity and Infrastructure Security Agency (CISA) released a joint report that an unknown threat group had compromised a federal civil administration organization and deployed the XMRig cryptocurrency mining malware in November 2022. The attacker exploited a VMware Horizon server that had not patched the Log4Shell (CVE-2021-44228) remote code execution vulnerability and successfully compromised the federated network. Therefore, how to detect vulnerability exploitation behavior in time becomes an urgent problem to be solved.

To discover vulnerability exploitation behavior, researchers have proposed a series of detection methods [1–12]. They can be divided into two categories: host-based detection methods and traffic-based detection methods. Host-based detection methods often require complex deployment and high system permissions. Traffic-based detection methods are more convenient, with a large amount of data, rich information, and ease of operation. They can work offline through port mirroring and only require traffic analysis. Therefore, we focus on traffic-based detection methods.

To detect an exploit from traffic, Polychronakis et al. [8] propose a heuristic detection method using a complete processor simulator to detect polymorphic shellcode in a network intrusion detection system (NIDS). However, it cannot detect unknown shellcode. Borders et al. [9] propose Spector, a traffic payload analysis engine, which uses symbolic execution technology to extract meaningful API calls in shellcode and generate the underlying disassembly code. Kanemoto et al. [10] propose an attack detection method based on code simulation, combined with IDS rules to detect whether remote shellcode attacks

are successful. Pratomo et al. [11] propose Blatta, a method of detecting early exploit traffic using a cyclic neural network. The main idea is to analyze the first 400 bytes of the application layer and identify vulnerability exploitation traffic according to shellcode characteristic bytes. In addition, Pratomo et al. [12] propose a low-rate attack detection method based on fine-grained network intelligence analysis, which detects attacks through unsupervised learning of application layer information. However, these traffic-based exploit detection technologies are almost based on load characteristic pattern matching, identifying shellcode with specific strings, which can be easily evaded and are invalid for encrypted traffic.

To overcome the above problems, we propose a novel exploit traffic detection method, ETDetector. We observe that the reverse shell attack often coexists with vulnerability exploitation and is among the most popular exploits. So, ETDetector starts from the inevitable relationship between the exploit and its result. We first extract the fusion information feature from original features, such as the packet delay sequence, as input of a decision tree model to identify reverse shell traffic in the shellcode execution stage. When the proportion of reverse shell traffic decreases to 1/300, the *F1-score* and *F2-score* of the fusion information feature are only lower by about 0.05 than the original features, whereas the average time of model training and detection is shortened by 90%. Then, we trace suspicious traffic in the shellcode delivery stage by reconstructing the session relationship of the two stages above. In ten vulnerability exploitation experiments, our method detects nine of them, and the effect is not affected by the specific types of vulnerabilities and traffic encryption methods. The main contributions are as follows:

- We develop a reverse shell traffic detection method based on thought time. We extract the fusion information feature from original features, such as the packet delay sequence, as input of a decision tree model to identify reverse shell traffic in the shellcode execution stage.
- We propose ETDetector, a novel exploit detection method. Different from the existing methods, we take the reverse shell, one of the most popular exploit behaviors, as the entry point to detect exploits traffic. We first identify reverse shell traffic in the shellcode execution stage. Then, we trace suspicious traffic in the shellcode delivery stage by reconstructing the session relationship of the two stages above.
- We design a traffic stratification method based on a bisecting K-means algorithm, which can intuitively show the traffic communication behavior and improve the interpretability of ETDetector.
- We simulate ten vulnerability exploitation experiments to evaluate the effect of ETDetector. It proves that ETDetector detects nine of them, and the result is not affected by the specific category of vulnerabilities and traffic encryption methods.

The rest of this paper is organized as follows. Section 2 introduces the definitions and categories of shellcode and reverse shell and their relationship. We introduce the proposed method ETDetector in Section 3. In Section 4, we evaluate the performance of ETDetector in detecting vulnerability exploitation traffic compared with the state-of-the-art work. It is concluded in Section 5.

2. Background

2.1. Shellcode

Shellcode refers to the code fragment with certain functions used in the exploitation of vulnerability [13]. Shellcode has all kinds of functions according to the various needs of attackers, including modifying the remote login authentication mode of the system, starting a command interface, initiating a remote connection, etc.

As we all know, shellcode has diversified as the battle between attack and defense escalates. Cheng et al. [14] classify shellcode into four types according to functions, including execute command (EC), bind shell (BS), reverse shell (RS), and executable download and execute (ED). Due to Intranet isolation and other reasons, the reverse shell is the most popular in actual exploit scenarios [15]. Therefore, this paper mainly studies the

malicious behavior generated after the reverse shell (RS) class shellcode is loaded, namely, reverse shell.

2.2. Reverse Shell

In the Cyber Kill Chain model [16], the reverse shell is the last stage in vulnerability exploitation and the first stage in the post-penetration. From the view of causality, the reverse shell is the result of the shellcode executed successfully, and the shellcode is the cause of the reverse shell. Therefore, detecting reverse shell behavior is a shred of crucial evidence for identifying vulnerability exploitation. This section mainly analyzes the definition and classification of the reverse shell and models its behavior from the traffic level.

2.2.1. Reverse Shell Definition and Classification

Stipovic et al. [17] define a reverse shell as a piece of malicious code that can establish a TCP connection from the controlled end to the control terminal and download the payload to achieve permission upgrade.

There are many ways to implement reverse shell attacks, but their essence is inseparable from network communication. According to different types of network protocols, reverse shells can be categorized as UDP-based reverse shells, ICMP-based reverse shells, TCP-based reverse shells, and HTTP/HTTPS-based reverse shells. Since the network intrusion detection system (NIDS) usually intercepts UDP and ICMP packets, the reverse shell based on HTTP/HTTPS is mostly non-real-time interaction, so the TCP-based reverse shells are more widely used in actual attack scenarios [18]. Therefore, this paper studies TCP-based reverse shells.

According to whether the plaintext of the payload is visible, TCP-based reverse shells include encrypted shells and non-encrypted shells. According to the type of shell returned, it can be divided into system shells and advanced shells, such as Meterpreter [19]. In conclusion, TCP-based reverse shells can be subdivided into four categories, namely non-encryption system shells, encryption system shells, non-encryption advanced shells, and encryption advanced shells.

2.2.2. Reverse Shell Communication Behavior Model

The network communication process of TCP-based reverse shells usually includes three stages: connection establishment, reverse shell, and command interaction. At the same stage, traffic of different types of reverse shells may show various characteristics. For example, in the connection establishment phase, a non-encrypted reverse shell only needs the three-way handshake, while an encrypted reverse shell additionally needs an encryption protocol negotiation process. After establishing the connection, a system shell directly enters the command interaction stage, when the target machine returns the system shell information. while an advanced shell, such as a Meterpreter shell, usually delivers the payload in segments first and then returns an advanced shell after it runs on the target machine.

With Metasploit penetration testing tools [20] in the Linux/x64/Meterpreter/reverse_tcp module as an example, as Figure 1 shows, in the connection establishment stage, the target machine establishes the TCP connection through the three-way handshake. Then, it enters the shell return stage. Since the Meterpreter belongs to staged attack payloads, attackers first send a small piece of payload to the target, named stage0, and then upload stage1 through stage0. stage1 and other payloads gradually hand over the control of the shell and finally return a multi-functional Meterpreter shell to the attacker. The process is similar to the "a little horse introduces a big horse" in web penetration. Finally, in the command interaction stage, the attacker sends the command to the control terminal, and the target machine receives and returns the result to the attacker. There is a relatively long interval between the packet carrying a command and the next packet due to the thought time indispensable for the attacker to decide what to enter.

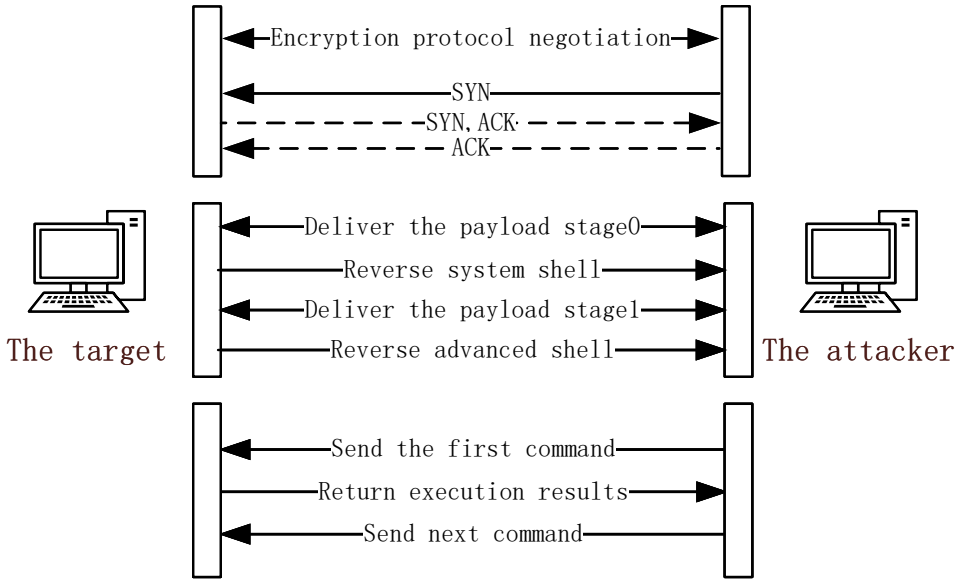


Figure 1. The network communication process of a typical TCP-based reverse shell.

3. Methodology

In this section, we first divide the stages of vulnerability exploitation from the perspective of the network attack chain [16] and then analyze the differences between vulnerability exploitation traffic and normal traffic. On this basis, we propose ETDetector, a novel exploit traffic detection method, and introduce the framework, specific modules, and workflow of ETDetector.

3.1. Traffic Feature Analysis

From the perspective of the network attack chain [16], vulnerability exploitation includes the shellcode generation stage, shellcode delivery stage, and shellcode execution stage. Generally speaking, attacks are performed locally in the first phase, while the others are more visible in traffic. In the delivery phase of the shellcode, the control terminal sends shellcode packets to the controlled end, which is similar to file uploading behavior, so it is not easy to distinguish. In the execution stage of the shellcode, because the control terminal will launch a reverse shell after the shellcode is loaded, the traffic has visible features and is easy to identify. Logically, the two stages have a temporal and spatial correlation. Therefore, this section first analyzes the traffic features of the execution phase and then illustrates the relationships of the two-phase sessions according to the spatiotemporal correlation.

3.1.1. Original Session Features

According to the reverse shell behavior model described in Section 2.2.2, reverse shell connections are always initiated by the controlled terminal. Moreover, in the command control stage, the attacker mainly sends the command and the target returns the information, and its behavior has visible interaction. We only choose valid packets in sessions to avoid interference from TCP heartbeat packets. Relevant definitions are as follows.

Definition 1. *Valid packet is the first SYN packet of a TCP three-way handshake in a session or other packet carrying a payload of more than 1 byte.*

We use three original session features to compare with our method, including packet delay sequence, packet direction sequence, and packet length sequence, which are available according to the quintuple, time stamp, and transmission layer load length of valid packets. The details are as follows:

(1) Packet delay sequence. The delay of a packet in a session means the time interval between the current packet and the last valid packet. The time interval sequence is the packet delay sequence. When reverse shell attacks occur, the attacker needs to take some time to think about what to do before performing operations, which causes the packet containing the attack command to have a relatively long interval with the next packet. Generally speaking, normal traffic will not appear in this situation. Therefore, this feature can obviously reflect the attacker's thinking time, and we can regard it as a crucial signal for detecting reverse shell attack behavior;
(2) Packet direction sequence. We pronounce the sequence of valid packets transmitted in a session as the sequence of packet directions. The direction of the first SYN packet in the three-way handshake is "0". The reverse packet direction is marked as "1". In addition to packet delay, packet interaction direction sequence is another important feature that can reflect the interaction of reverse shell attack behavior. Generally speaking, in the early stage of rebound shell attacks, the target initiates a TCP connection and returns shell-related information to prepare for subsequent interactions. After a successful reverse shell, the attacker sends the command, the target executes the command, and returns the result. Then, it enters the shell return stage. Taking the Metasploit reverse shell module as an example, the attacker usually sends packets bearing the Trojan to the target machine. When Meterpreter completely takes over the control of the target machine, the target sends the packet carrying the Meterpreter shell information to the control terminal, and the packet transmission direction is "0". The last stage is command interaction. The control terminal sends packets carrying attack commands, the direction of which is "1". The target machine sends packets with the response message, the direction of which is "0". Because the attacker usually waits for the target machine to return the message before executing a new command, there are generally only packets in the direction of "0" between the two packets in the direction of "1";
(3) Packet length sequence. We regard a valid packet length sequence in a session as a packet length sequence. Packet length means transport layer payload length. After returning the shell, the attacker sends the command, and the target machine returns the command execution result. Generally, the length of a system command is within a confined range, but the command execution result is not limited. Usually, one of the primary purposes for the attacker to execute commands after the successful reverse shell attack is to obtain information about the target. Therefore, from the perspective of data flow direction, there is data leakage. This behavior is reflected in the traffic by the shorter payload length of the packet carrying the command and the longer payload length of the packet carrying the response message.

3.1.2. Fusion Information Features

The original session features extracted in Section 3.1.1, such as packet delay sequence, packet direction sequence, and packet length sequence, are relatively simple; however, it may be impossible to extract the information thoroughly. Therefore, in this section, based on the concept of delay packet, information fusion is carried out on the above features from the two dimensions of time and space. The fused feature vector is taken as the model input.

The delay packet in a TCP session is defined as follows.

Definition 2. *Delay packet (delayed packet) is a packet the delay ratio of a packet in a session to the duration of the whole session (two decimal places reserved) is greater than 0. Otherwise, it is a non-delay packet (non-delayed packet).*

Based on the original session features extracted in Section 3.1.1, the fusion information feature vector is available. It contains five flag bits, and each sub-features are as follows:

(1) The number distribution sequence of delay packets within 3 min before reverse flow. Three flag bits represent the number distribution of delay packets in the first three

minutes. The number "0" indicates that no reverse delay packet exists in the time range, and "1" represents that the reverse delay packet exists in the time range;

(2) Whether the packet length sequences of the delayed packet and non-delayed packet obey the same distribution. First, we extract the packet length sequences of the session. The directions are symbolized by positive and negative values. Then, we decide whether the two packet length sequences obeyed the same distribution (whether the p-value was less than 0.05, which indicated that the distribution was inconsistent) by the Kolmogorov–Smirnov test (K-S test). "0" means to obey the same distribution, and "1" means contrarily;

(3) The difference between the packet direction sequence of the delayed packet and the packet direction change sequence. Firstly, the packet direction sequences of all delayed packets and non-delayed packets were extracted, respectively, with "0" representing forward and "1" representing reverse. Then, the sequence of packet direction change is calculated, respectively. "0" means the direction of adjacent packets remains unchanged, and "1" means the change. Next, we calculate the Hamming distance between the delay packet direction sequence and the packet direction change sequence, and its ratio to its sequence length. The feature is represented by a bit of a flag bit, and the threshold is set as 0.58. If the ratio is greater than 0.58, the flag bit is "1"; otherwise, it is "0". For details about the meaning of the threshold setting, see the following.

3.1.3. Analysis of Two-Stage Session Association Features of Vulnerability Exploitation

Logically, the shellcode delivery phase and shellcode execution phase have a causal relationship, causing two-stage sessions to have a temporal and spatial correlation. Therefore, this section mainly analyzes the spatiotemporal association features of two-phase sessions and explains how to preliminarily filter out suspicious shellcode delivery phase traffic based on traffic features in Section 3.1.2.

(1) According to the reverse shell session features described in Section 3.1.2, we can identify the IP address of the suspicious target machine. As shellcode delivery sessions must be binding with the target, the first step is to filter out all the packets that have the target's IP address as the recipient;

(2) From the chronological analysis, the shellcode execution phase always occurs after the shellcode delivery phase, which means the message carrying the shellcode must appear before the first SYN message of the reverse shell session three-way handshake. The interval between the execution of the shellcode and the establishment of the reverse shell connection is usually less than 0.1s. Considering the delay caused by network transmission problems, we filter out all the packets based on timestamp and further analyze the suspicious session.

3.1.4. Session Feature Analysis in Shellcode Delivery Phase

In Section 3.1.3, we detect suspicious sessions in the shellcode delivery stage according to the spatiotemporal correlation features. Considering that the target's benign programs are still running when the vulnerability exploitation occurs, the exploit traffic tends to blend with a mass scale of legal traffic. This section analyzes the session characteristics of the shellcode delivery phase from four aspects, including the direction of TCP session connection establishment, data flow direction, number of response packets, and packet length distribution, to identify the exploit traffic in the shellcode delivery stage.

(1) Direction of establishing TCP connection sessions. In the shellcode delivery phase, the attacker usually already gathers service information of the target, so it is reasonable to establish a TCP connection. On the contrary, a legal client sends the first SYN message regardless of whether users browse web pages or transfer files. Therefore, the direction of the TCP connection session establishment can be crucial to determining the exploit traffic. Of course, attackers also use social engineering to induce users to click malicious links and actively establish TCP connections with malicious servers to

obtain shellcodes in the real world. To specify the scope of this paper, we suppose the attacker always initiates a TCP session. If the target receives the first SYN message in a session, step into (2). Otherwise, it is considered benign traffic;

(2) The direction of data flow. In the shellcode delivery phase, the attacker sends the shellcode to the target, and the target usually only sends an ACK message. Therefore, data flow is mainly from the attacker to the target. We set the timestamp of the first SYN message in the three-way shakehand of the reverse shell session as the cut-off point. Suppose the data flow to the target in the last interaction before this time point and step into (3). Otherwise, it is considered benign traffic;

(3) Number of the response message. Through the analysis of exploit traffic, we infer that after the attacker sends shellcode packets, the target either does not send a response message or sends multiple response packets. Legal programs of the target always send one or more response packets after receiving the request from the client, apart from receiving a FIN or RST message to disconnect the connection. Therefore, we use the number of response messages from the target in the last interaction before the cut-off time for the verdict. Provided the target sends no response with the connection lasting, the session is a suspicious shellcode delivery session. If there are multi-response messages, step into (4). Otherwise, it is considered benign traffic;

(4) Packet length distribution. After the successful execution of shellcode delivery, the target may return some content containing relevant information about itself, similar to the response message information sent by the legal programs, but payload length distributions are different. The former message is related to the specific command sent by the attacker, causing different payload lengths. Generally, the latter is fragmented as maximum payload length to improve transmission efficiency. Therefore the number of bytes in each fragment is the same except for the last one. Here we infer the type of a session based on whether the payload length of response packet fragmentation is uniform. If the payload length of response packet fragmentation conforms to the general slicing discipline, it is benign traffic. Otherwise, it is a suspicious shellcode delivery session.

3.2. Overview of ETDetector

The overall framework of ETDetector is shown in Figure 2, including three stages: data preprocessing, feature extraction, and anomaly detection.

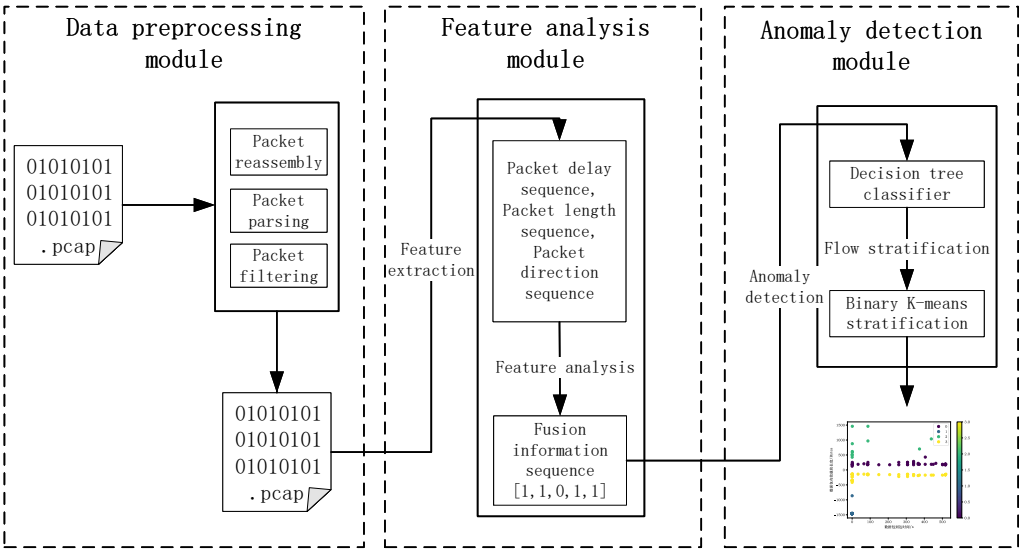


Figure 2. Overview of ETDetector.

3.2.1. Data Preprocessing

This module includes packet recombination, packet analysis, and packet filtering. Packet reassembly makes TCP packets separated and reassembled by the bi-direction flow. Packet parsing mainly analyzes the captured traffic layer by layer according to network protocols and generates readable network packet details for the feature extraction stage. Packet filtering filters out ACK packets without payloads based on the SYN and ACK flag bits and payload length of TCP packets to improve analysis efficiency.

3.2.2. Feature Extraction

The feature extraction module extracts the quintuple, time stamp, and payload length of each packet in the session, saves it as an a.CSV file, and calculates the original session features in Section 3.1.1. Then, according to the method described in Section 3.1.2, we calculate an information fusion vector from the two dimensions of time and space as the input of the machine learning model.

3.2.3. Abnormal Detection

The section illustrates how to detect vulnerability exploitation traffic based on the features in Section 3.1.

First, we classify the reverse shell traffic and benign traffic based on machine learning. Considering the low-dimensional feature vector proposed in Section 3.1.2 and limited vulnerability exploitation traffic samples, we prefer the model suitable for few-shot learning based on an unbalanced dataset. As we all know, support vector machine, KNN, and logistic regression algorithms are inappropriate on the unbalanced training set. Random forest and gradient-lifting decision trees are preferred for high-dimensional features classification. The naive Bayes algorithm requires prior probability calculation. The decision tree model has good interpretability, is suitable for small-scale datasets, and has excellent efficiency. Therefore, we select the decision tree model as a classifier to detect the reverse shell traffic.

Then, we trace suspicious traffic in the shellcode delivery stage by reconstructing the session relationship features in Section 3.1.3, and confirm the shellcode delivery phase traffic according to the session features of the shellcode delivery phase in Section 3.1.4.

Finally, we use a bisecting K-means algorithm to stratify the detected vulnerability exploitation traffic to intuitively show the traffic communication behavior and improve the interpretability of ETDetector.

4. Evaluation

We verify the effectiveness of the proposed method by answering the following two research questions in this section:

- RQ1: Can the decision tree model identify reverse shell traffic at whether it is encrypted?
- RQ2: Can ETDetector detect the traffic of the shellcode delivery stage based on reverse shell traffic?

4.1. Experimental Setup

4.1.1. Data Collection

Since open datasets usually merely consist of shellcode delivery phase traffic, ETDetector is based on shellcode execution results, namely, reverse shell session detection, and it cannot be verified by exploit traffic in open datasets.

Therefore, we generate the traffic by artificial reverse shell experiment to train the decision tree model and implement ten vulnerability exploitation experiments using the Metasploit penetration testing framework to gather test traffic. Moreover, we collect the laboratory exit traffic as the background traffic. Normal online behaviors include browsing the web, SSH login, transferring files, and remote desktop login. Unencrypted reverse shell traffic is generated by the Metasploit module unencrypted system shell, while encrypted reverse shell traffic is generated by the Metasploit module encryption system shell and

Meterpreter advanced shell. Table 1 shows a detailed description of our dataset. It is worth mentioning that the Meterpreter traffic has a dedicated encapsulation format, which contains no plain text. Therefore, in the verification experiment, regardless of whether the traffic is encrypted, Meterpreter traffic is considered encrypted traffic.

Table 1. The detailed description of our dataset.

Scene	Traffic Type (Number)	Use
Get online legally	Background traffic (38,512)	18% for training and validation (6847) and 82% for testing in Experiments 1 and 2 (31,665)
Non-encrypted reverse shell	Non-encrypted reverse shell traffic (77)	72% for training and validation (6847) and 28% for testing in Experiments 1 and 2 (21)
Encrypted reverse shell	Encrypted reverse shell traffic (39)	Used for testing in Experiment 2
Realistic attack scenario	Non-encrypted reverse shell traffic (5); encrypted reverse shell traffic (5)	Used for testing in Experiment 3

4.1.2. Evaluate Metrics

We evaluate the performance of the decision tree classifier according to *Accuracy*, *Precision*, *Recall*, *F*1-*score*, and *F*2-*score*. To illustrate the meaning of the above assessment indexes, we define four cases and five assessment indexes of the binary model as follows.

- TP (TRUE positive): The current traffic is predicted to be reverse shell traffic, and it is reverse shell traffic.
- FN (FALSE negative): The current traffic is predicted to be normal, but it is a reverse shell traffic.
- FP (FALSE positive): The current traffic is predicted to be reverse shell traffic, but it is normal traffic.
- TN (TRUE negative): The current traffic is predicted to be normal, and it is normal traffic.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1\text{-}score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

$$F2\text{-}score = \frac{5 * Precision * Recall}{4 * Precision + Recall} \tag{5}$$

According to Equations (2) and (3), the *Accuracy* is the proportion of correctly predicted reverse shell traffic in the predicted reverse shell traffic, and the *Recall* is the proportion of correctly predicted reverse shell traffic in the total reverse shell traffic. The former represents the false positive rate, and the latter relates to the false negative rate.

From the view of practice, we aim to detect malicious traffic as much as possible, so the false negative rate and *Recall* can better reflect the effectiveness of the detection model.

According to Equations (4) and (5), compared with *F1-score*, *F2-score* increases the weight of the *Recall* and can better represent the overall detection effect of the model.

In addition, we use the detection rate of encrypted traffic, non-encrypted traffic, and all vulnerability exploitation traffic to evaluate the performance of our method. The higher the detection rate, the lower the probability of code execution vulnerability exploitation behavior escaping detection, that is, the better the detection method effect.

Within the defined time T, the total number of occurrences of code execution vulnerability exploitation d behavior is N, where the number of occurrences under traffic encryption condition is denoted as *N-crypted*, and the number of occurrences without encryption is denoted as *N-noncrypted*. The number of code execution vulnerability exploitation behaviors detected is M, where the number of encrypted traffic is denoted as *M-crypted* and the number of non-encrypted traffic is denoted as *M-noncrypted*.

- M: The number of code execution vulnerability exploitation behaviors detected in the defined time T.
- MC: The number of encrypted traffic detected in the defined time T.
- MN: The number of non-encrypted traffic detected in the defined time T.
- N: The total number of occurrences of code execution vulnerability exploitation behavior in the defined time T.
- NC: The number of occurrences under traffic encryption conditions in the defined time T.
- NN: The number of occurrences without encryption in the defined time T.

$$rate{-}crypted = \frac{MC}{NC} \tag{6}$$

$$rate{-}noncrypted = \frac{MN}{NN} \tag{7}$$

$$rate{-}total = \frac{M}{N} \tag{8}$$

4.2. Experimental Design Scheme

To answer RQ1, we design a comparative experiment (Experiment 1 and Experiment 2) using different feature sequences with three traffic mixed ratios. To answer RQ2, we conduct Experiments 3 and 4. Experiment 3 simulated ten vulnerability exploitation attacks and compared the detection effect of ETDetector with Blatta [11], which uses a cyclic neural network to detect early exploit traffic. In Experiment 4, we implement a hierarchical analysis and visualization of the exploit traffic to enhance the interpretability of the method.

4.2.1. Comparison of Detection Results of Mixed Traffic in Three Proportions

First of all, we will prove the validity of the above three sub-features and illustrate their practical significance.

(1) The number distribution sequence of delay packets in the reverse flow within the first three minutes. Figure 3 shows that, compared with white traffic, there are more delay packets of reverse direction in reverse shell traffic. Generally speaking, once the attackers get a successful reverse shell, they should execute commands immediately, causing a series of delay packets due to the thought time. Moreover, the attackers usually wait for the command execution result before sending new commands. Therefore, the transmission rate of the delay packets is limited, and the distribution is relatively dispersed within a certain period. On the contrary, the interaction of benign programs is almost automatic, making delay packets rare. Even if there are delay packets, they should be intense in short intervals due to the sending rate of programs. Therefore, the number of delay packets in each interval can effectively distinguish the traffic generated by human and automatic operation, reducing the possibility of false positives;

(2) Whether the packet length sequences of the delayed packet and non-delayed packet obey the same distribution. Figure 4 shows the K-S test results of the packet length sequence of the delayed packet and non-delayed packet, where the K-S test p-value corresponding to the red dashed line is 0.05. It shows that the consistency of packet length sequence distribution of reverse shell traffic sessions is below 0.05, meaning they do not obey the same distribution, while 98% of benign traffic sessions are on the contrary. As we all know, packets sent by benign programs fit the fragment rule to achieve maximum transmission efficiency. However, it is not the same in reverse shell traffic. Delay packets bearing commands have a short packet payload, and non-delayed packets delivering shellcode have more bytes, causing a relatively mass difference in packet length distribution. Here we judge whether the packet length sequences in bi-direction flows obey the same distribution based on the K-S test. Because it is one of the most popular non-parametric methods, and it is sensitive to the difference in the position and shape parameters of the empirical distribution function of samples. When the p-value is less than 0.05, we deny the null hypothesis, meaning the two packet length sequences do not obey the same distribution;

(3) The difference between the packet direction sequence of the delayed packet and the packet direction change sequence. Figure 5 shows the ratio of the Hamming distance to the packet sequence length. Compared with benign traffic, the delay packet direction sequence of the reverse shell traffic is more different from its packet direction change sequence. As commands are always from the attacker to the target in reverse shell interaction, delay packets with commands always belong to the reverse flow, and the direction changes infrequently. We suppose that the direction change sequence starts with "0", and if the direction of the packet is different from the last one, note "1", else "0". Therefore, the direction change sequence of delay packets in reverse shell sessions consists of merely "0". According to Section 3.1.1, "1" appears continuously in the direction sequence of the reverse shell traffic delay packet. We infer the distribution of the above sequence is symmetrical, and the Hamming distance is the best way to describe the difference between the two sequences. Considering the length of the packet sequence may cause interference, we use the ratio of Hamming distance to sequence length as the judgment basis. Figure 5 shows the ratio of reverse shell traffic is all greater than 0.58, while 99% of benign traffic is on the contrary. Therefore, we can detect reverse shell traffic easily with the threshold of 0.58.

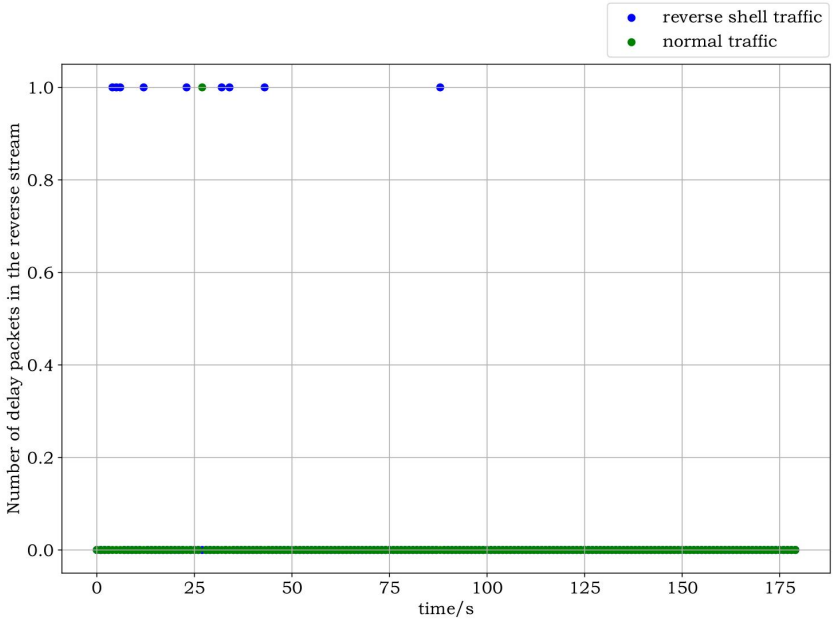


Figure 3. Comparison of the number distribution of delay packets in the reverse flow.

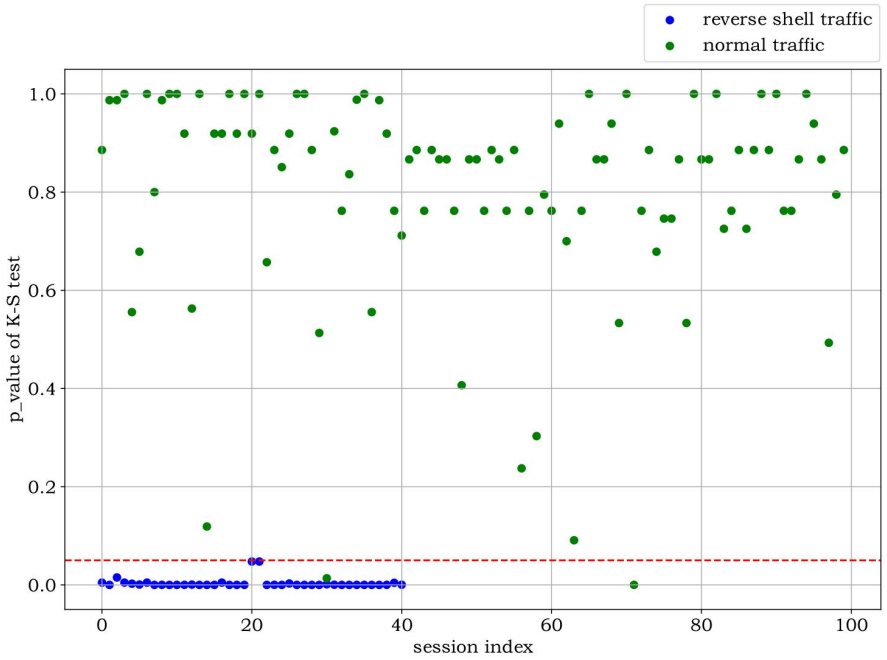


Figure 4. Comparison of K-S test results of bi-direction packet length sequences.

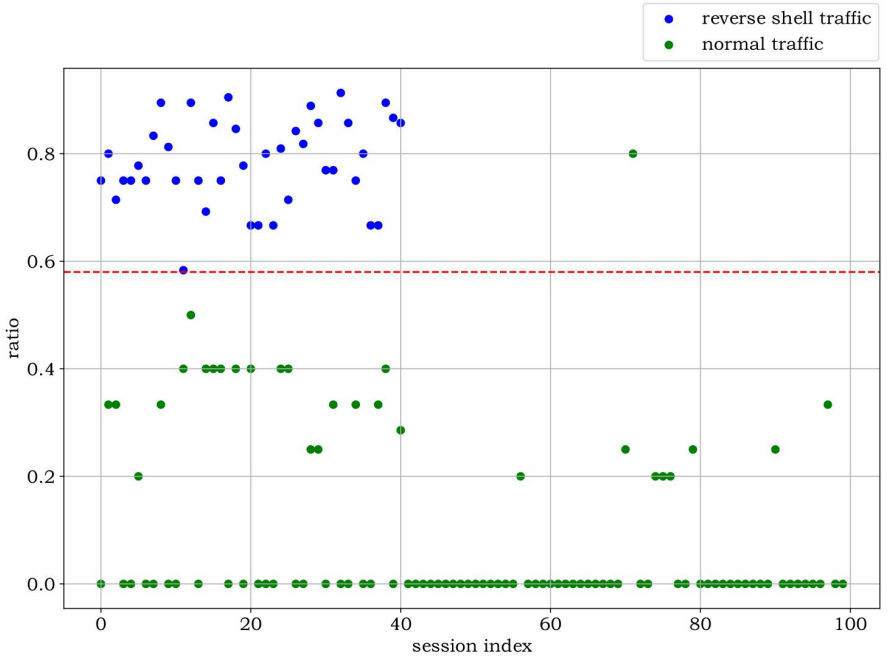


Figure 5. Comparison of the Hamming test results between the delay packet direction sequence and the change sequence of packet direction.

The following experiments compare the results of the decision tree model with three proportions of reverse shell traffic.

Table 2 shows that the *Accuracy*, *Recall*, *F*1-*score*, and *F*2-*score* of the model fluctuate slightly with the proportion of benign traffic increasing. As the ratio of reverse shell traffic to benign traffic reach 1:300, the *Accuracy* remains above 75%, and the *Recall* is 72%. The values of the *F*1-*score* and *F*2-*score* are still over 0.7, indicating that the fusion information feature vector not only can accurately detect reverse shell traffic but also has a relatively stable detection effect.

Because our method is on the essence that the reverse shell behavior is different from the online behavior of legal users, the reverse shell session can be accurately detected even when the reverse shell traffic in real attack scenarios is rare.

Table 2. The detection results of the decision tree model with three proportions of reverse shell traffic.

Ratio	Accuracy	Precision	Recall	*F1-Score*	*F2-Score*
1:50	0.995	**0.942**	**0.82**	**0.876**	**0.841**
1:100	0.995	0.835	0.671	0.703	0.679
1:300	**0.998**	0.785	0.720	0.717	0.715

4.2.2. Comparison of Detection Results before and after Feature Fusion

The purpose of Experiment 2 is to verify that the feature fusion method proposed in Section 3.1.2 improves the detection effect and robustness compared with the original features in Section 3.1.1.

As we can see in Figures 6–8, with the increase in the proportion of benign traffic, the results of the fusion feature sequences are relatively stable compared with original features. We infer that session features become diverse, causing it difficult to identify reverse shell traffic by original features merely. However, no matter how the proportion of traffic changes, the fusion information feature always conforms to the principle of reverse shell attack, so the detection effect is relatively stable.

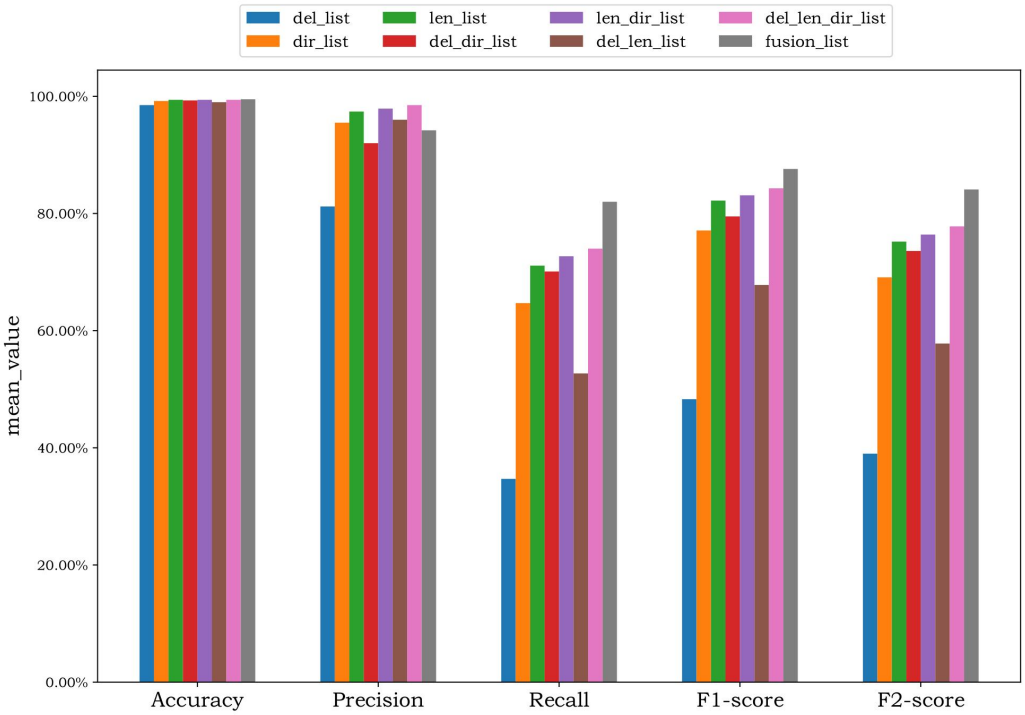


Figure 6. Comparison of detection results before and after feature fusion with the ratio 1:50.

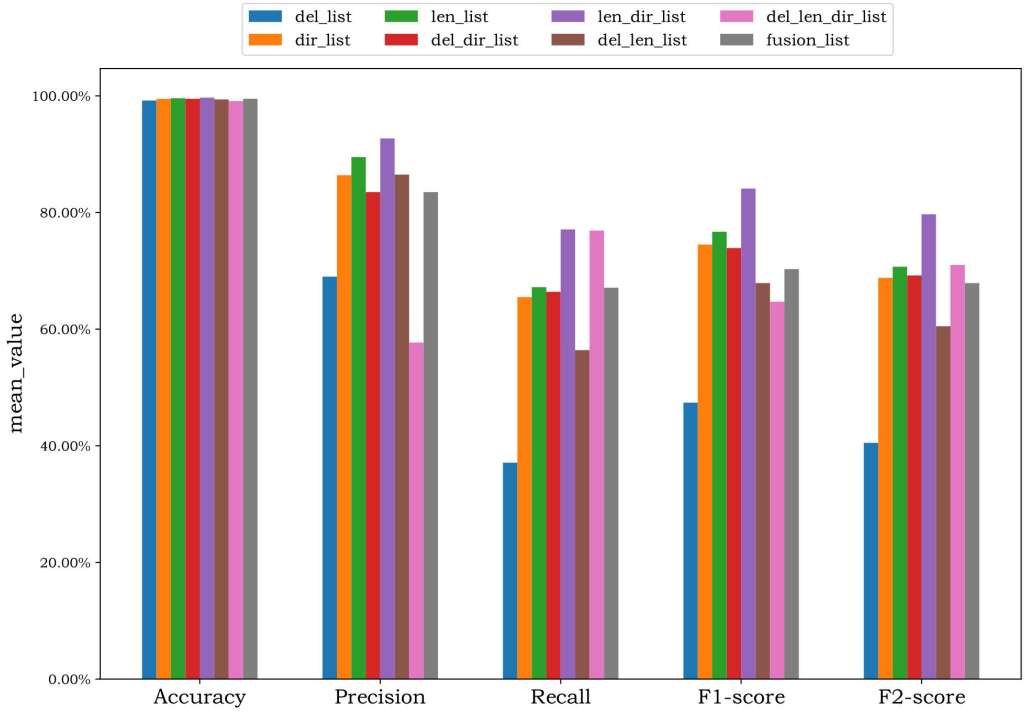


Figure 7. Comparison of detection results before and after feature fusion with the ratio 1:100.

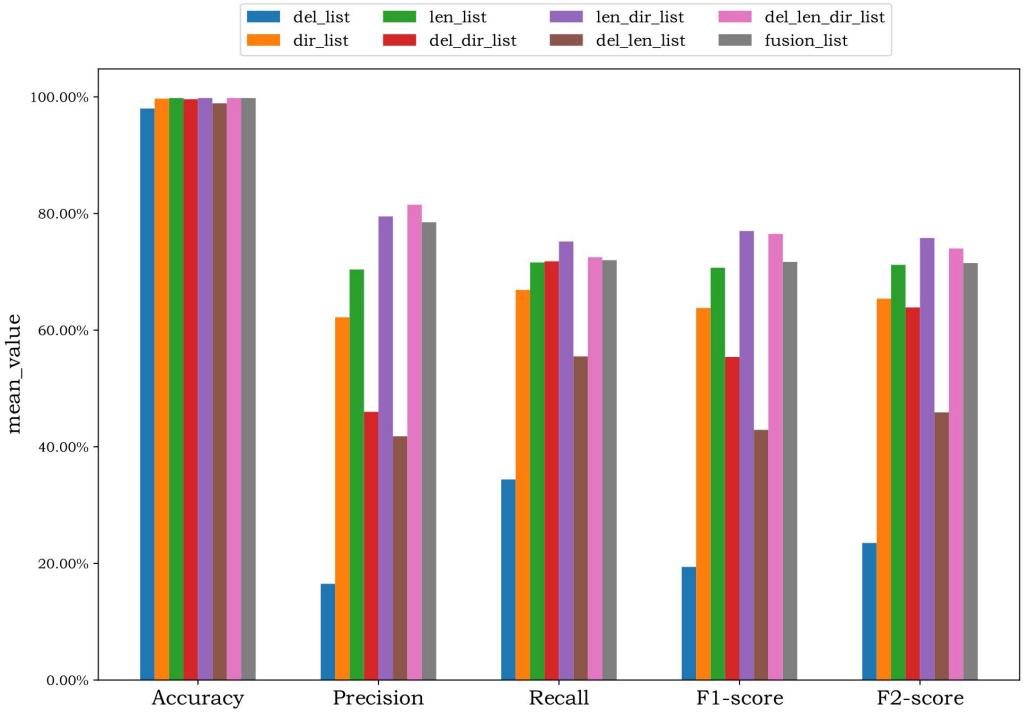


Figure 8. Comparison of detection results before and after feature fusion with the ratio 1:300.

In addition, with the increase in flow scale, the time required for model training and detection of original features increased significantly. On the contrary, the fusion information feature remains stable, as shown in Table 3 for details.

When the proportion is 1:300, the average required time of the fusion information feature group is only 1.379 s. The best group "packet length + packet direction + packet delay" sequence needs 13.992 s, 10.1 times longer, but its *F1-score* and *F2-score* are only

0.06 and 0.04 higher than the former, respectively. The fusion information feature is more practical for reverse shell traffic detection in large-scale traffic scenarios.

Table 3. Comparison of detection results before and after feature fusion.

Feature Vector Type	Mean Time Required/s		
	1:50	**1:100**	**1:300**
Packet delay sequence	3.614	4.384	7.868
Packet direction sequence	3.392	4.120	7.105
Packet length sequence	3.505	4.292	7.335
Packet delay + packet direction sequence	3.587	4.364	7.901
Packet length + packet direction sequence	3.601	4.185	7.368
Packet delay + packet length sequence	6.522	7.716	14.36
Packet delay + packet length + packet direction sequence	6.424	7.943	13.992
Fusion information feature vector	**0.863**	**0.949**	**1.379**

4.2.3. Comparison of Detection Results of Exploit Traffic

Experiment 3 compares the detection ability of ETDetector and Blatta, the most popular method to date.

The experimental results show that only 5 exploits of code execution were detected in Reference [11], while ETDetector detected 9 exploits, and rate-total increased by 40%. For the four exploits under the condition that the traffic is not encrypted, the corresponding traffic can be detected in Reference [11] and ETDetector and there is no significant difference in the rate-encrypted index. However, for the six times of code execution vulnerability exploitation under traffic encryption conditions, only one time was detected in Reference [11], while ETDetector was detected five times, and rate-encrypted increased by about 66.7%. This is because the method in Reference [11] detects based on traffic load. In the case of unencrypted traffic, plaintext features of shellcode in the load can be extracted; however, in the case of encrypted traffic, the content of the payload is invisible, and effective information cannot be extracted effectively, so encrypted traffic cannot be detected. However, ETDetector extracts traffic behavior characteristics independent of load content. Even under the condition of traffic encryption, as long as the TCP protocol is used, the traffic communication behavior characteristics conform to the law, and it can be effectively detected.

According to Table 4, the ninth reverse shell attack avoids both methods. Through analysis, the related module is Windows/Meterpreter/reverse_http, which communicates by HTTP sessions initiated by the target machine, intrinsically different from TCP-based reverse shells, so the fusion information feature in Section 3.1.2 cannot detect it. Since the ultimate purpose of detecting reverse shell traffic in this paper is to locate the target machine, there is no restriction on the specific methods to achieve reverse shell traffic detection. Therefore, for the reverse shell traffic implemented by HTTP/HTTPS protocol, supplementary experiments were carried out by combining some existing vulnerabilities and using tool traffic detection methods. The results show that as long as the reverse shell traffic can be detected and the target machine can detect the whole process of vulnerability exploitation traffic according to the session relationship, there is no significant impact on the detection results. Therefore, in practical applications, accurate positioning of the target machine can be realized in combination with the detection methods of the reverse shell against other protocols, to realize the detection of vulnerability exploitation traffic.

Table 4. Comparison of detection results of exploit traffic.

Vulnerability Type, CVE, and Shell Type	Encrypted	Reference [11]	Ours
Remote code execution vulnerability, CVE-2021-22205, bash	✗	✓	✓
Buffer overflow vulnerability, MS17-010, Python	✗	✓	✓
Remote code execution vulnerability, CVE-2020-17530, Perl+OpenSSL	✓	✓	✓
Remote code execution vulnerability, CVE-2019-15107, linux/x64/shell/reverse_tcp	✗	✓	✓
Buffer overflow vulnerability, MS17-010, windows/shell_reverse_tcp	✗	✓	✓
Buffer overflow vulnerability, CVE-2017-7494, python/shell_reverse_tcp_ssl	✓	✗	✓
Remote code execution vulnerability, CVE-2019-0708, windows/Meterpreter/reverse_tcp	✓	✗	✓
Buffer overflow vulnerability, CVE-2017-7494, linux/x64/Meterpreter/reverse_tcp	✓	✗	✓
Remote code execution vulnerability, CVE-2019-0708, windows/Meterpreter/reverse_http	✓	✗	✗
Remote code execution vulnerability, CVE-2019-0708, windows/Meterpreter/reverse_tcp_rc4	✓	✗	✓

4.2.4. The Stratification Results of Detected Exploit Traffic

Experience 4 uses a bisecting K-means algorithm to stratify the detected exploit traffic and visualize the exploit traffic, thus further proving the interpretability of ETDetector. The preset cluster number in the bisecting K-means algorithm is four, and the traffic of the buffer overflow vulnerability CVE-2017-7494 reverse shell at two stages is stratified.

Figure 9 shows the traffic stratification results in the shellcode delivery phase. In this stage, the attacker aims to deliver the malicious shellcode to the target. The packets bearing on the shellcode are intense at the beginning of the session, and the payload is long and diverse. Meanwhile, the target usually passively receives the shellcode and only delivers the ACK message without a valid payload. Therefore, all clusters are above the axis of X equals 0 in Figure 9, and the value of the Y-axis is diverse.

Figure 10 shows the traffic stratification results in the shellcode execution phase, and the reverse shell attack is implemented by the python/shell_reverse_tcp_ssl module. In the shellcode execution phase, the target returns the encrypted shell. Initially, the attacker delivers the malicious payload to the target, corresponding to the blue cluster below the axis of X equals 0. in Figure 10. Then, the encryption protocol negotiates, bi-direction packets interact frequently and have symmetrical lengths. This process is very short-lived and corresponds to the purple cluster in Figure 10. After the shell is returned successfully, the attacker usually executes commands to obtain information about the target. In general, the data flows to the attacker. Typically, packets carrying commands are short in length and relatively dispersed; however, the target machine needs to return corresponding execution results according to different commands, so the packets carrying execution results have different lengths, corresponding to the green cluster and yellow cluster on either side of the axis of X equals 0 in Figure 10.

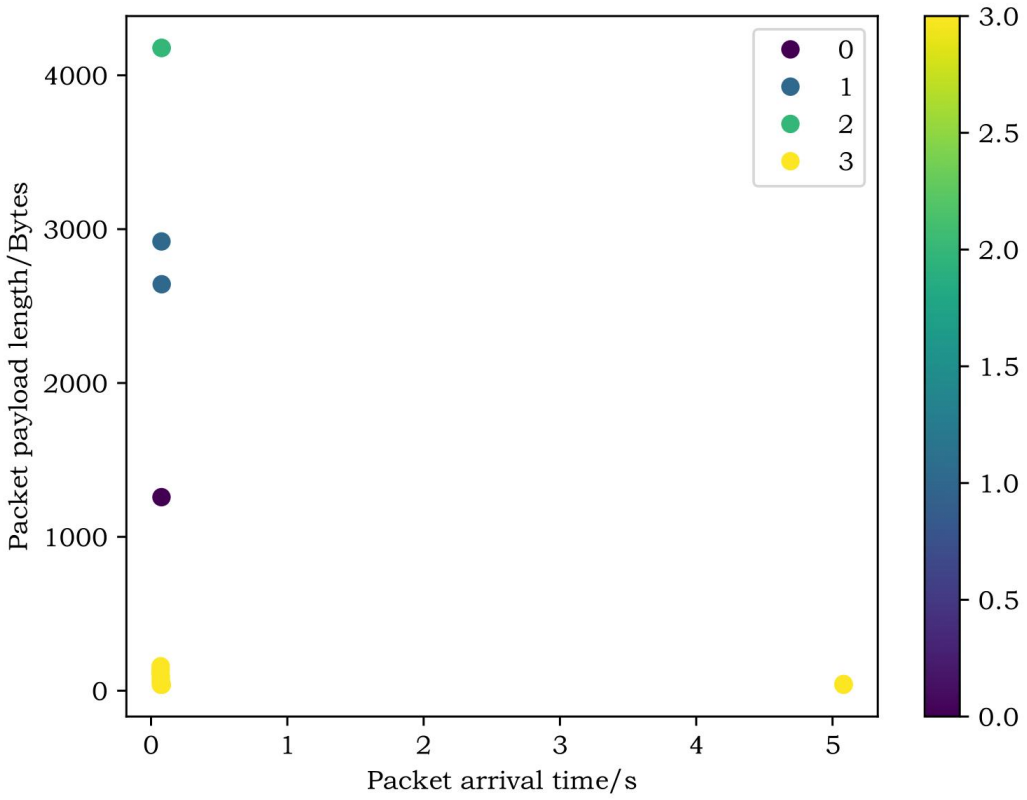


Figure 9. The traffic stratification results in the shellcode delivery phase.

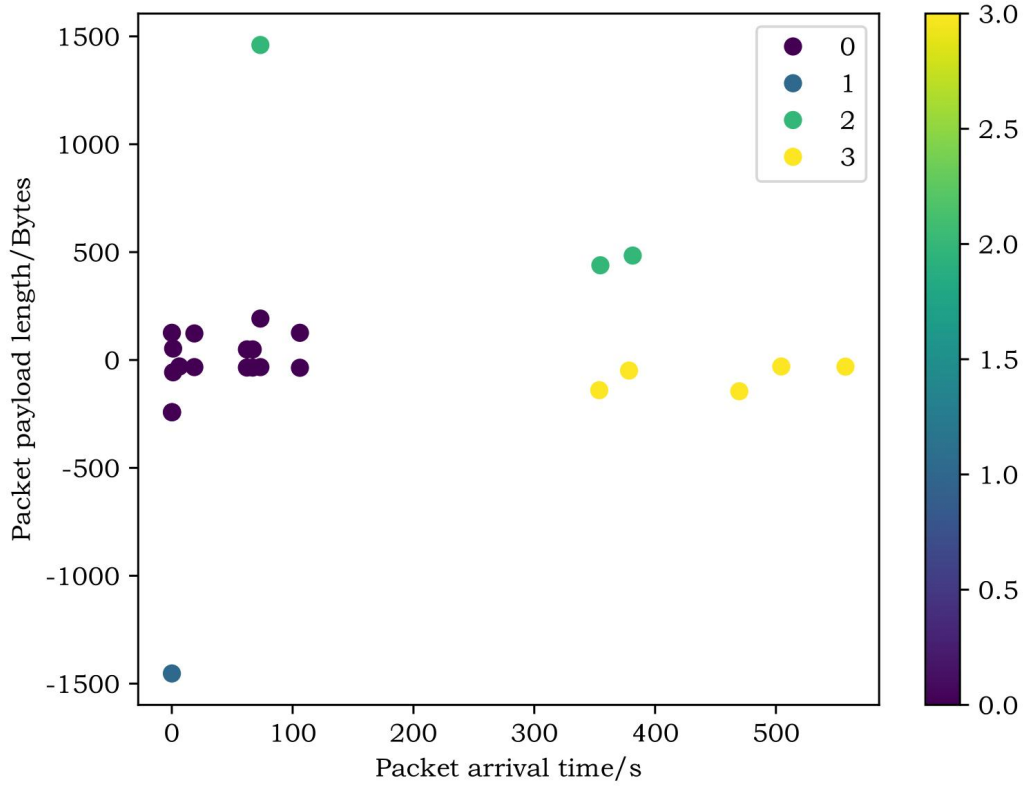


Figure 10. The traffic stratification results in the shellcode execution phase.

5. Conclusions

We propose ETDetector, a method of exploit traffic detection based on reverse shell behavior analysis. Compared with Blatta, one of the popular methods to detect the exploit

traffic, ETDetectorhas improved the detection rate with encrypted traffic based on the reverse shell. We plan to research other exploitation behaviors in the future.

Author Contributions: Conceptualization, Y.L.; data curation, R.C.; methodology, Y.L.; software, Y.L. and S.L.; formal analysis, X.Y.; writing—original draft preparation, Y.L.; writing—review and editing, Y.L. and X.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Foundation Strengthening Key Project of Science & Technology Commission (2019-JCJQ-ZD-113).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The calculated data presented in this work are unavailable due to privacy and ethical restrictions.

Acknowledgments: The author would like to thank the anonymous reviewers for their valuable comments on our paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kong , D.; Tian, D.; Pan, Q.; Liu, P.; Wu, D. Semantic aware attribution analysis of remote exploits. *Secur. Commun. Netw.* **2013**, *6*, 818–832. [CrossRef]
2. Wu, J.; Arrott, A.; Osorio, F.C.C. Protection against remote code execution exploits of popular applications in Windows. In Proceedings of the IEEE 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE), Fajardo, PR, USA, 28–30 October 2014.
3. Parrend, P.; Navarro, J.; Guigou, F.; Deruyver, A.; Collet, P. Foundations and applications of artificial Intelligence for zero-day and multi-step attack detection. *EURASIP J. Inf. Secur.* **2018**, *2018*, 4. [CrossRef]
4. Homoliak, I.; Teknos, M.; Ochoa, M.; Breitenbacher, D.; Hosseini, S.; Hanacek, P. Improving Network Intrusion Detection Classifiers by Non-payload-Based Exploit-Independent Obfuscations: An Adversarial Approach. *EAI Endorsed Trans. Secur. Saf.* **2018**, *5*, 156245. [CrossRef]
5. Chen, L.; Sultana, S.; Sahita, R. Henet: A deep learning approach on intel® processor trace for effective exploit detection. In Proceedings of the IEEE Security and Privacy Workshops (SPW), SP Workshops 2018, San Francisco, CA, USA, 24 May 2018.
6. Biswas, S.; Sohel, M.; Sajal, M.M.; Afrin, T.; Bhuiyan, T.; Hassan, M.M. A study on remote code execution vulnerability in web applications. In Proceedings of the International Conference on Cyber Security and Computer Science (ICONCS 2018), Karabuk, Turkey, 18–20 October 2018.
7. Mokbal, F.M.; Dan, W.; Imran, A.; Jiuchuan, L.; Akhtar, F.; Xiaoxi, W. MLPXSS: An integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique. *IEEE Access* **2019**, *7*, 100567–100580. [CrossRef]
8. Polychronakis, M.; Anagnostakis, K.G.; Markatos, E.P. Network-level polymorphic shellcode detection using emulation. *J. Comput. Virol.* **2007**, *2*, 257–274. [CrossRef]
9. Borders, K.; Prakash, A.; Zielinski, M. Spector: Automatically analyzing shell code. In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, USA, 10–14 December 2007.
10. Kanemoto, Y.; Aoki, K.; Iwamura, M.; Miyoshi, J.; Kotani, D.; Takakura, H.; Okabe, Y. Detecting successful attacks from IDS alerts based on an emulation of remote shellcodes. In Proceedings of the IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019.
11. Pratomo, B.A.; Burnap, P.; Theodorakopoulos, G. Blatta: Early exploit detection on network traffic with recurrent neural networks. *Secur. Commun. Netw.* **2020**, *2020*, 8826038. [CrossRef]
12. Pratomo, B. Low-Rate Attack Detection with Intelligent Fine-Grained Network Analysis, Ph.D. Thesis, Cardiff University, Cardiff, UK, 2020.
13. Foster, J.C. *Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals*; Elsevier: Amsterdam, The Netherlands, 2005.
14. Cheng, T.H.; Lin, Y.D.; Lai, Y.C.; Lin, P.C. Evasion techniques: Sneaking through your intrusion detection/prevention systems. *IEEE Commun. Surv. Tutor.* **2011**, *14*, 1011–1020. [CrossRef]
15. Noman, H.A.; Al-Maatouk, Q.; Noman, S.A. Design and Implementation of a Security Analysis Tool that Detects and Eliminates Code Caves in Windows Applications. In Proceedings of the International Conference on Data Analytics for Business and Industry, Virtual, 25–26 October 2021.
16. Yadav, T.; Rao, A.M. Technical aspects of the cyber kill chain. In Proceedings of the International Symposium on Security in Computing and Communication, Kochi, India, 10–13 August 2015.
17. Stipovic, I. Antiforensic techniques deployed by custom developed malware in evading anti-virus detection. *arXiv* **2019**, arXiv:1906.10625.

18. Leka, C.; Ntantogian, C.; Karagiannis, S.; Magkos, E.; Verykios, V.S. A Comparative Analysis of VirusTotal and Desktop Antivirus Detection Capabilities. In Proceedings of the 2022 13th International Conference on Information, Intelligence, Systems & Applications, IISA 2022, Corfu, Greece, 18–20 July 2022.
19. Denis, M.; Zena, C.; Hayajneh, T. Penetration testing: Concepts, attack methods, and defense strategies. In Proceedings of the IEEE Long Island Systems, Applications, and Technology Conference, LISAT 2016, Farmingdale, New York, NY, USA, 29 April 2016.
20. Rahalkar, S.; Jaswal, N. *The Complete Metasploit Guide: Explore Effective Penetration Testing Techniques with Metasploit*; Packt Publishing Ltd.: Birmingham, UK, 2019.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.