A NOVEL EXPLOIT TRAFFIC TRACEBACK METHOD BASED ON SESSION RELATIONSHIP

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

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China.

ABSTRACT

Vulnerability exploitation is the key to obtaining the control authority of the system, posing a significant threat to network security. Therefore, it is necessary to discover exploitation from traffic. The current methods usually only target a single stage with an incomplete causal relationship and depend on the payload content, causing attacker easily avoids detection by encrypting traffic and other means. To solve the above problems, we propose a traffic traceback method of vulnerability exploitation based on session relation. First, we construct the session relationship model using the session correlation of different stages during the exploit. Second, we build a session diagram based on historical traffic. Finally, we traverse the session diagram to find the traffic conforming to the session relationship model. Compared with Blatta, a method detecting early exploit traffic with RNN, the detection rate of our method is increased by 50%, independent of traffic encryption methods.

KEYWORDS

Exploit, Malicious Traffic Detection, Session Relationship, Traffic Analysis

1. INTRODUCTION

As software features continue to grow, so do the size of code and the number of security holes. Attackers can use vulnerabilities to steal information, upgrade permissions, or even directly control the operating system, posing a serious threat to network security. Therefore, it is necessary to propose an exploit detection method.

To discover exploit behaviour, researchers proposed a series of detection methods for exploits[1]-[12]. At present, there are two types of detection technologies for exploits: host-based detection method and traffic-based detection method. Host-based detection methods often require detection tools to be deployed on the host system and require high system permissions. The traffic-based detection method is characterized by a large amount of data, rich information, and ease of operation. It can be deployed in offline mode through port mirroring and only requires traffic analysis. Therefore, this paper focuses on traffic-based detection methods. As early as 2007, Polychronakis et al. [8] proposed a heuristic Detection method by using a complete processor simulator to detect polymorphic shellcode in Network Intrusion Detection System (NIDS). But unknown shellcode cannot be detected. Borders et al. [9] proposed Spector, a traffic load analysis engine, which uses symbolic execution technology to extract meaningful API calls in shellcode and generate the underlying disassembly code. In 2019, Kanemoto et al. [10] proposed an attack detection method based on code simulation technology, combined with IDS rules to detect whether remote shellcode attacks are successful. In 2020, Pratomo et al. [11] proposed Blatta, a method of detecting early exploit traffic by using a cyclic neural network. The main idea is to detect the first 400 bytes of the application layer and identify exploit traffic according to shellcode characteristic bytes. In addition, Pratomo et al. [12] proposed a low-rate attack detection method based on fine-grained network intelligence analysis, which detects attacks through unsupervised learning of application layer information. However, the above traffic-based exploit detection technologies are mostly based on the idea of load characteristic pattern matching. shellcode characteristic code in traffic is used to detect exploits, which can be easily avoided and cannot detect encrypted traffic.

In 2022, Trofti et al. proposed a traffic detection method for vulnerability exploitation tools. This approach treats a network flow as a weighted directed interaction between a pair of nodes and proposes a simple method that can be used to use connectivity graph features in unsupervised anomaly detection algorithms. However, this method can only detect the traffic in the late stage of vulnerability exploitation, and cannot construct a complete causal relationship.

To solve the above problems, we present a novel exploit traffic traceback method. First, we construct the session relationship model using the session correlation in different stages of vulnerability exploitation. Second, we build a session graph based on historical traffic. Finally, we traverse the session graph to find the traffic that fits the session relationship model.The main contributions are as follows:

(1) we propose a detection model based on session relationship, which can detect the whole process of vulnerability exploitation traffic when the target's IP address is known;
(2) we propose SRG-ETDetector, a traffic traceback method based on the session relationship. Unlike the existing methods, SRG-ETDetector is according to the idea of causal association, which can not only detect the early and late traffic of vulnerability exploitation simultaneously but also has nothing to do with the traffic encryption method.
(3) We simulate ten vulnerability exploitation, and the result shows that the detection rate of our method is increased by 50% compared with Blatta, using a recurrent neural network to detect early attack traffic, and the detection results are independent of the specific cause of the exploited vulnerability and the traffic encryption method.

We organize the rest of this paper as follows. Section 2 introduces the definitions and categories of vulnerability exploitation, shellcode, and their relationship. We propose a novel exploit traffic detection method SRG-ETDetector in Section 3. In Section 4, we evaluate the performance of ETDetecor in detecting vulnerability exploitation traffic compared with the state-of-the-art work. It concludes in Section 5.

2. BACKGROUND

We first introduce the concepts related to vulnerability exploitation and classify common vulnerability exploitation behaviour, and point out that our research objective is code execution vulnerability exploitation. Secondly, shellcode-related concepts are introduced and classified according to function. Finally, we introduce the concepts related to reverse shell and classify reverse shell methods.

2.1. Vulnerability Exploitation Definition and Classification

Vulnerability exploitation is the behaviour that an attacker can access or destroy system data under unauthorized circumstances by taking advantage of the lack of consideration in the design of computer protocols, software, or hardware, or the flaws in the system security policy. As we all know, attackers usually exploit the vulnerability to obtain the target device system control rights or to destroy the target device system availability. Specifically, it can be divided into the following five types.

(1) Code execution vulnerability exploitation. A code execution attack means that the attacker uses the vulnerability of the application program to execute the code that implements malicious functions by constructing special command strings.
(2) Authentication bypass vulnerability exploitation. An authentication bypass attack means that the attacker uses the vulnerabilities in the login and identity authentication process of the system to avoid the system authentication process, and logs on to the system as a legal user without authentication information such as account and password.
(3) Privilege escalation vulnerability exploitation. A privilege escalation attack means that the attacker uses the vulnerability to bypass the system Settings, perform file operations, and set system information and other operations with higher privileges.
(4) Data leakage vulnerability exploitation. Data leakage attack refers to the attacker using system vulnerabilities to view, copy, steal, modify, and other illegal operations on legal user data without permission.
(5) Denial of service vulnerability exploitation. Unlike the above four attacks, denial of service attacks do not aim to gain system permissions, but to prevent or weaken the authorized use of the network, system, or application by exhausting system resources such as CPU, memory, bandwidth, or disk space.

From the perspective of the attack chain [14], the relevance of the above five kinds of vulnerability exploitation is shown in Figure 1Illustration of the five vulnerability exploitation associations. As can be seen from Figure 2, a code execution attack is the most important prerequisite and necessary attack path of the other four attack behaviors, so it is the most important to study the vulnerability exploitation traffic detection method for the purpose of a code execution attack. Therefore, this paper focuses on code execution vulnerability exploitation as the research object.

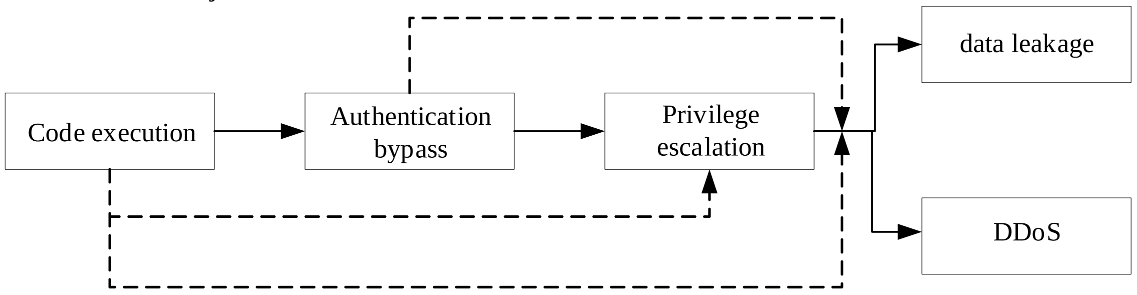

Figure 1. Illustration of the five vulnerability exploitation associations

2.2. Shellcode Definition and Classification

Shellcode refers to a code fragment with certain functions used in the code execution vulnerabilities exploitation [15], which can achieve various functions according to the needs of attackers, including modifying system Settings, starting shell interface, establishing remote sessions, etc.

Shellcode has diversified as the battle between attack and defence escalates. Cheng et al.[16]In terms of function, the shellcode is divided into four types: 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 widely used in actual exploit scenarios [17]. Therefore, we mainly analyse the malicious behaviour generated after the loading of the Reverse Shell (RS) class shellcode, namely, reverse shell.

2.3. Reverse ShellDefinition and Classification

Stipovic et al. [18] defined a reverse shell as a piece of malicious code, whose function is to 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 in real applications, but their essence is inseparable from network communication. Based on the types of communication protocols, reverse shells can be divided into 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 reverse shell based on TCP is more widely used in actual attack scenarios [19]. Therefore, this paper studies the reverse shell based on the TCP protocol.

According to whether the content of the traffic load is visible, reverse shells based on TCP protocol are divided into the encrypted shell and non-encrypted shell. According to the type of shell returned, it can be divided into system shell, and other advanced shells, such as Meterpreter[20]. Therefore, according to the different types of traffic encryption and the types of shell returned, reverse shells based on TCP protocol can be subdivided into four categories, namely non-encryption system shells, encryption system shells, non-encryption advanced shells, and encryption advanced shells.

3. METHOD

This chapter proposes a vulnerability exploitation detection model based on traffic behaviour. Firstly, the whole process of vulnerability exploitation was divided into stages according to the relevant actors of the attack payload, and the possible malicious behaviours and their corresponding traffic communication behaviour characteristics in each stage were analysed. Secondly, the malicious behaviour correlation between adjacent stages and the corresponding traffic session correlation characteristics were analysed. Then, based on the above characteristics, a vulnerability exploitation detection model based on traffic behaviour analysis was proposed. The model consists of two parts. The first part is responsible for locating the target IP address, and the second part detects the whole process traffic of vulnerability exploitation based on individual session characteristics and session relationships. Finally, we analyse the practical application of the model.

3.1. Vulnerability Exploitation Behaviours Analysis

In this section, according to the attack steps, we divide the whole exploit process into payload generation phase, payload delivery phase and payload execution phase, and analyse the behaviour of each phase respectively.

3.1.1. Behaviours Analysis during Payload Generation

Generating the payload is the first step in any exploit. In this phase, the actor is the attacker, and the included attack steps are shown in Figure 2. Firstly, the attacker collects vulnerability related information of the target device through vulnerability scanning and other methods, writes shellcode according to the vulnerability type and trigger principle, and writes code segments that need to be executed through the vulnerability exploitation according to the attack purpose, and combines them into the attack payload, namely payload. The payload is then encoded, encrypted, and so on. Typically, generating payloads is a local activity, meaning that there is no interaction between the attacker and the target device that is reflected in the traffic. Although the attacker has the action of scanning the target device for vulnerabilities in this stage, usually, this stage belongs to the period when the attacker unilaterally collects target intelligence, and does not involve the interaction behaviors directly related to vulnerability exploitation. Since the payload generation phase and the latter two phases are not necessarily tightly coupled in time, only the payload delivery phase and payload execution phase are considered for the time being when detecting the vulnerability exploitation traffic.

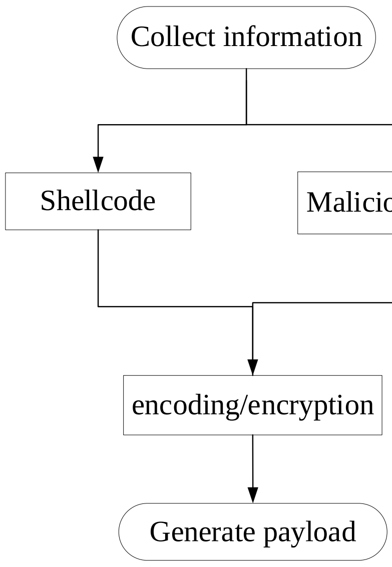

Figure 2. Illustration of attacker’s behaviours during payload generation

3.1.2. Behaviours Analysis during Payload Delivery

Once the payload is generated, we move on to the payload delivery phase. In this stage, the actor is also the attacker. The main goal of the attacker is to send the payload to the target device, which is mainly achieved by two ways.

(1) passive delivery, mainly represented by social engineering, which is characterized by various forms of carriers and strong stealth, but the attack effect depends on the target user's artificial choice and has uncertainty.
(2) Active delivery: the attacker has mastered the port opening status of the target device, then he actively establishes a session, and sends payload-bearing packets to the target device.

In this paper, we ignore the impact of social engineering, and focus on the case where the attacker actively accesses the port to deliver the attack payload. Its behaviour is shown in Figure 3.

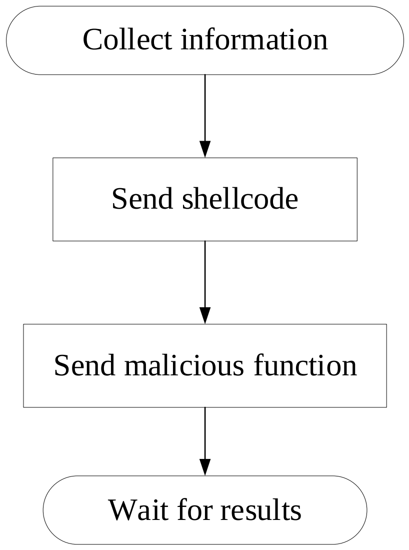

Figure 3. Illustration of attacker's behaviours during payload delivery

Firstly, the attacker will obtain the port opening information of the target device through port scanning and other means to prepare for the later establishment of network session connection. After gathering enough intelligence, the attacker sends packets containing shellcode and exploit code to the target device in turn. It is important to note that the shellcode and the attack code are not necessarily sent together.

Taking the SMB remote code execution vulnerability CVE-2017-7494 as an example, there is a sequence in which packets are sent.

Firstly, the attacker always initiated TCP connection actively, accessed the 445 port of the target device, determined the version number of SMB protocol, and then anonymously logged in. Through path traversal and other methods, the physical path of the shared directory of Samba server was exploded, and whether the dynamic link library file containing malicious code could be written was explored. Finally, the SMB protocol vulnerability function *is_known_pipename()* does not check the special characters in the name of the anonymous pipe to load the dynamic link library file using the name, so as to run malicious code to achieve the purpose of the attack. And the packet containing malicious code may not be delivered in one time. Generally, for exploit tools such as Metasploit, there are three types of attack payloads: single, stager, and stage, depending on how they are delivered and run. The single is transmitted to the target device at one time and can run independently. stager, on the other hand, is a transporter and is only responsible for establishing a new connection between the target device and the attacker so that subsequent attack payloads, called stages, can be delivered to the target device. The Stage is the actual malicious code used to accomplish the purpose of the attack. The advantage of this method is that it allows the attacker to use a smaller attack payload to load the actual malicious code, and makes the communication mechanism and the final execution phase independent of each other, so there is no need to copy the code, which can avoid the detection of anti-virus software on the target device and improve the probability of successful execution of malicious code.

In terms of traffic, this phase may involve multiple sessions. First, the initial session is initiated by the attacker to deliver shellcode and the first part of the attack payload to the target device.

After the vulnerability is successfully triggered by the shellcode, the first part of the attack payload is run, usually by creating a new session with the goal of transmitting further attack payloads.

3.1.3. Behaviours Analysis During Payload Execution

After the payload is delivered to the target device, the payload execution phase begins. The main purpose of the attacker in this phase is to wait for the execution result of the attack code, and the main body of the behaviour is converted from the attacker to the target device. The flow of its behaviour is shown in Figure 4.

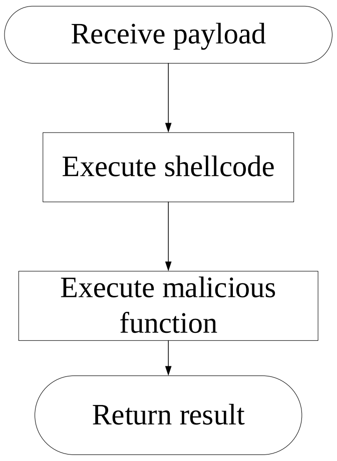

Figure 4. Illustration of target's behaviours during payload delivery

This phase is the most important step of vulnerability exploitation and determines whether a vulnerability exploitation is successful or not. In the payload delivery phase, after the attacker initiates the first connection to the target device and sends the first attack payload, the target device receives the packet containing shellcode and parses its contents. If the vulnerability is successfully triggered, the malicious code following the shellcode is executed. For code execution exploits, the first piece of malicious code after the shellcode usually establishes a new connection between the target device and the attacker. In the Intranet environment, the new connection is usually made by the way of the target device connecting back, which is easy to break through the firewall and other protection products. In the case of segmented attack payloads, the malicious code sent along with the shellcode is called a stager, or transporter, and is responsible for connecting back to the attacker, establishing a new connection, and delivering the actual code for the attack, the stage, to the target device. When the last stage is run on the target device, the execution result is returned to the attacker, which represents the successful completion of the exploit. Combined with the analysis of the attacker's behaviours in the delivery phase of payload in Section 3.1.2, it is not difficult to see that the delivery phase and the execution phase of payload are only different in the behaviour subject and have a certain temporal relationship. However, the behaviour of the attacker transmitting the payload in segments and the behaviour of the target device running the payload are tightly coupled, and the two phases occur alternately until the last part of the malicious code is executed. Therefore, it is also tightly coupled on traffic sessions.

In terms of traffic, similar to the payload delivery phase, this phase may also involve more than one session. First, the initial session is initiated by the attacker to deliver shellcode and the first part of the attack payload to the target device. After the vulnerability is successfully triggered by the shellcode, the first part of the attack payload is run, usually by creating a new session with the goal of transmitting further attack payloads. The specific characteristics of the different sessions in this phase and their relationship analysis are given in Section 3.2.

3.2. Session Relationship Analysis of Vulnerability Exploitation

According to the two conditions "whether the final instructions are executed on the target machine immediately after the vulnerability is triggered" *("final instructions"refers to a series of operations performed by the attacker to achieve persistence in subsequent stages, such as downloading trojans, backdoors to the target machine, etc.)* and"whether the target needs to communicate with other terminals besides the attacker ", The session relationship of the last two attack stages of vulnerability exploitation is roughly divided into four categories, and "execute download Trojan command" is taken as an example to illustrate, where the attacker is A, the target device is B, and other devices are C.

Considering the reverse shell type shellcode, which is mainly studied in this paper, the possible session combinations for vulnerability exploitation are shown in Table 1.

For the first and fourth cases, since only one session is involved and multiple phases are included in the session, whether the session is suspicious traffic can be determined by analysing the data flow direction segmentation, packet length segmentation, etc.

However, in the case of multiple sessions, it is necessary to locate the traffic sessions in the whole process of vulnerability exploitation through the characteristics of session node association, session establishment time sequence, session establishment direction, session data flow direction, packet direction sequence, packet payload length and size distribution, etc. The comparison table contains 12 session relationships in the case of multiple sessions.

Table 1. 12 possible session combinations for vulnerability exploitation

Index	Session number	Session relationship	Session1 function	Session2function	Session3 function	Reverse shell
1	1	A-B	Send&executepayload	-	-	√
2	2	A-B, B-A	Sendpayload	Executepayload	-	×
3	2	A-B, B-C	Sendpayload	Executepayload	-	×
4	1	A-B	Send&executepayload，reverseshell	-	-	√，A-B
5	2	A-B, B-A	Sendpayload，reverseshell	Executepayload	-	√，A-B
6	2	A-B, B-C	Sendpayload，reverseshell	Executepayload	-	√，A-B
7	2	A-B, B-A	Sendpayload	Reverseshell&execute payload	-	√，B-A
8	2	A-B, B-C	Sendpayload	Reverseshell&execute payload	-	√，B-C
9	3	A-B, B-A, B-A	Sendpayload	Reverseshell	Executepayload	√，B-A
10	3	A-B, B-A, B-C	Sendpayload	Reverseshell	Executepayload	√，B-A

11	3	A-B, B-C, B-A	Sendpayload	Reverseshell	Executepayload	√，B-C
12	3	A-B, B-C, B-D	Sendpayload	Reverseshell	Executepayload	√，B-C

3.3. An Exploit Traffic Detection Model Based on Session Relationship

According to the content of Section 3.1 and 3.2, combined with the idea of "abductive causation by effect", we propose a novel exploit traffic detection model based on traffic behaviour analysis. Define that in time T, the set of all IP addresses appearing is I, and the set of all ports is P. The session relation is constructed as a graph $G = (V, E, F)$, where V is the node set and the elements in the node set $vi = (ip\_i, port\_i, flag)$, $ip\_i \in I, port\_i \in P, flag \in \{1,0\}$.The $flag$ is the flag bit; the default value is 0. If in the preliminary detection, node vi is the controlled target device, that is, the target machine, then the flag bit $flag = 1$. E is the edge set, and the elements in the edge set $e = (vi, vj)$. F is the session feature set, and the elements in the session feature set $fi = (i, dir, p\_num, p\_dir\_list, p\_bytes\_rate, p\_len\_list)$, where i is the session sequence number, dir is the session setup direction, p_num is the number of data packets, and fi is the number of data packets. p_dir_list represents the packet direction sequence, $p_bytes_rate$represents the upload/download ratio, and p_len_list represents the packet length sequence.

The detection model is shown in Figure 5. Firstly, the traffic passing through the port in time T is captured, and then the traffic is cut and stored in the form of sessions to filter out empty ACK packets. Finally, the infected target device is located by detecting malicious traffic sessions during the execution of payload. Then, the timestamp of the first SYN packet of the malicious traffic session "triple handshake" during the payload execution phase is extracted, and the historical traffic before this time is backtracked. According to the session relationship described in Section 3.2, the traffic sessions in the whole process of vulnerability exploitation are located through the characteristics such as the association of session nodes, the time sequence of session establishment, the direction of session establishment, the direction of session data flow, the sequence of packet direction, and the distribution of packet payload length and size, and the integrated alarm information is output.

Different from the existing detection methods, the detection model proposed in this paper is based on the idea of "causal association", and takes the result of vulnerability exploitation as the breakthrough point. It can not only detect the malicious traffic generated in the late stage of vulnerability exploitation, but also detect the traffic in the whole process of vulnerability exploitation according to the session relationship. Since the detection basis of this model is the session relationship characteristics of different stages of attack, as long as the specific analysis of each stage of attack behaviour is carried out, the session relationship is summarized, and on this basis the current session relationship graph model is extended, theoretically any kind of multi-stage attack traffic can be detected.

In practical applications, considering that one of the main purposes of code execution vulnerability exploitation is to obtain system control rights, and due to the restrictions of security devices such as firewalls, reverse shell has become a common control means for attackers, so detecting reverse shell traffic can be used as an entry point to locate target machines. Namely, the first part can be implemented by the reverse shell traffic detection method. The second part includes session classification, session relation graph construction and suspicious session localization. The session classification is carried out according to the segmentation of the session data stream, which can be realized by combining the sequence segmentation algorithm with the clustering algorithm. Session relationship graph construction and suspicious session localization can be implemented by graph models.

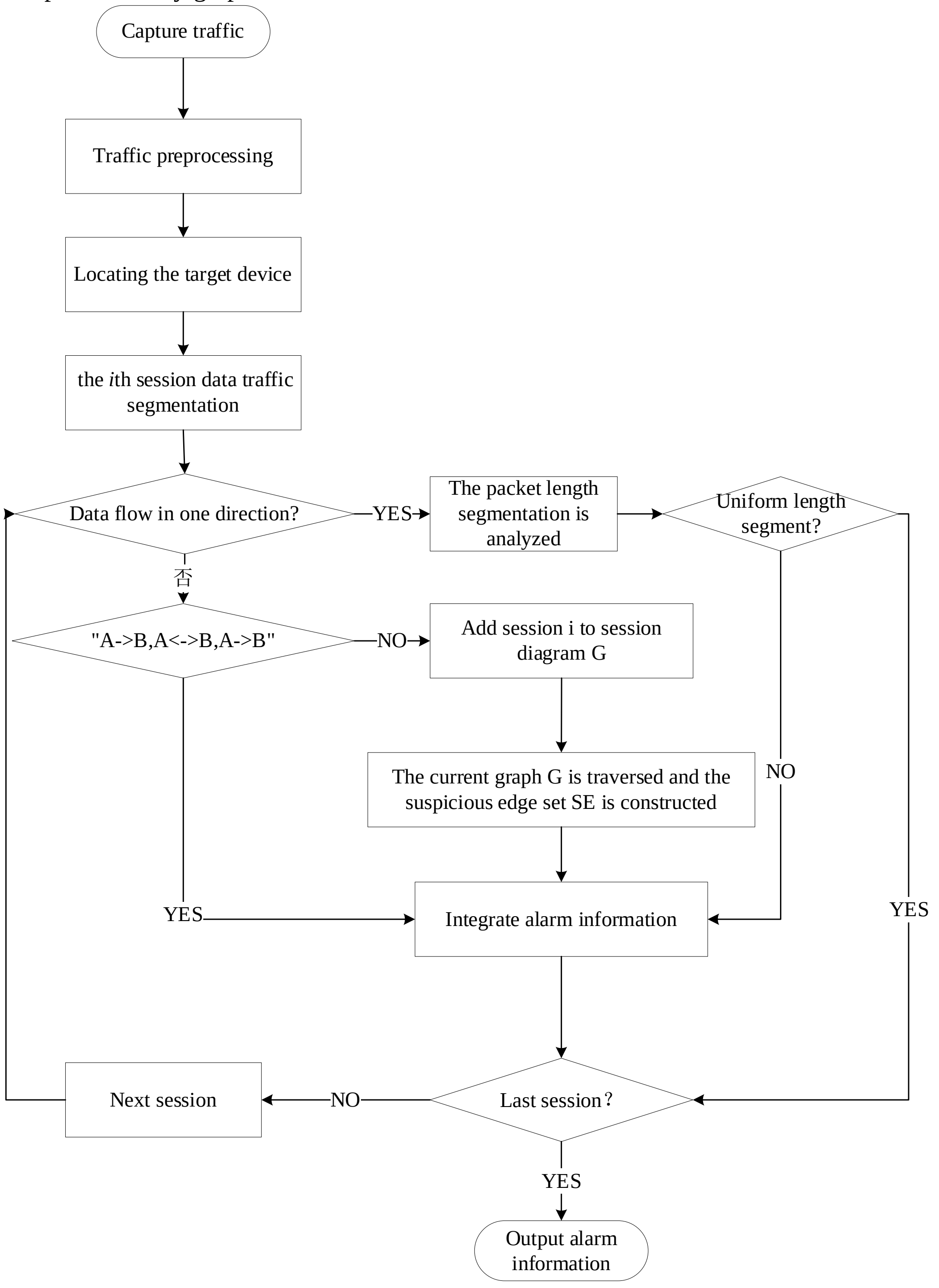
Capture traffic
Traffic preprocessing
Locating the target device
the *i*th session data traffic segmentation
Data flow in one direction?
YES
The packet length segmentation is analyzed
Uniform length segment?
否
"A->B,A<->B,A->B"
NO
Add session i to session diagram G
The current graph G is traversed and the suspicious edge set SE is constructed
NO
YES
Integrate alarm information
YES
Next session
NO
Last session？
YES
Output alarm information

Figure 5. Vulnerability exploitation traffic detection model based on traffic behaviour

3.4. Framework of SRG-ETDetection

We propose a code execution vulnerability exploitation detection method based on session relations, SRG-ETDetetcion, and the overall framework is shown inFigure 6, which includes four modules including data preprocessing, reverse shell detection, session classification, and vulnerability exploitation traffic detection. This section describes the functions and implementation details of each module in detail.

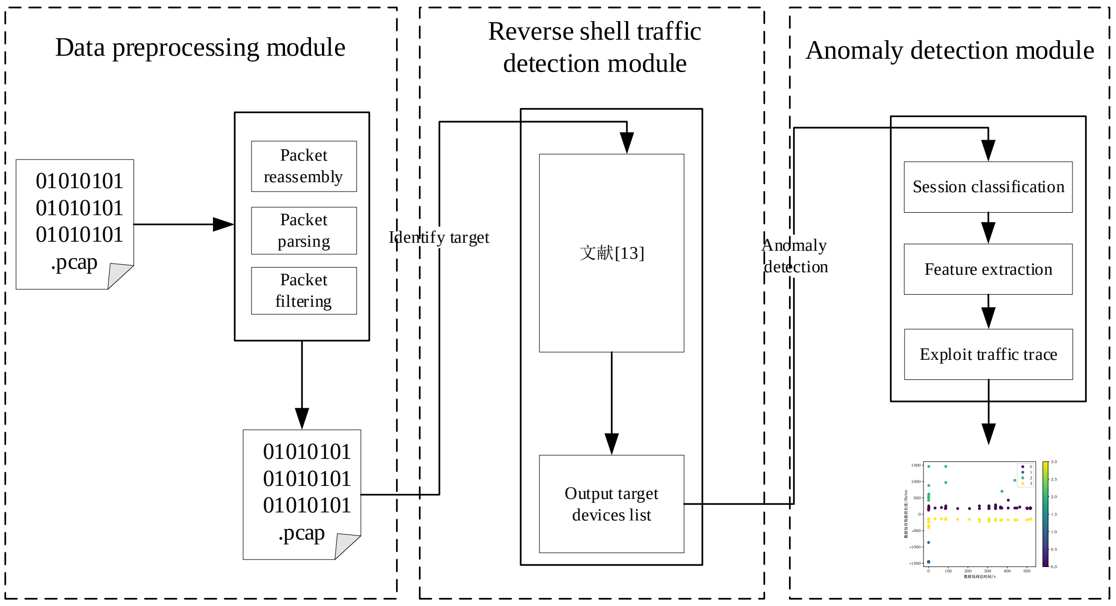

Figure 6. The framework of SRG-ETDection

(1). Data preprocessing module

The specific workflow of this module is shown in Figure 24. Firstly, the collected raw traffic was parsed by the traffic processing tool SplitCap.exe, and the TCP data bidirectional flow was merged into a complete communication session according to the five-tuple < source IP, source port, destination IP, destination port, protocol >. Then, the first SYN packet and other packets whose payload length is greater than 0 in the TCP connection establishment phase are filtered out, and the information of each packet such as quintuple, timestamp and payload length is extracted. On this basis, the packet interval sequence, packet length sequence and packet direction sequence of each session are obtained, and the related work is implemented by Algorithm 1. See Table for the pseudocode of Algorithm 1.

Table 3. Data preprocessing module Algorithm 1 pseudocode

```
Algorithm 1：
Input: session_j
Output: delt_j, len_j, dir_j

for packet_i; packet_i in session_j; i++:
if packet_i.TCP.SYN=0
&packet_i.TCP.len=0:
        pass;
else:
        delt_j.append(packet_i_delt);
        len_j.append(packet_i_len);
        dir_j.append(packet_i_dir).
```

(2). Reverse shell detection module

The purpose of detecting reverse shell traffic is to confirm the target device, so it is independent of the specific method of detecting reverse shells. Due to the detection effect, deployment method, detection efficiency and other reasons, we choose Metasploit reverse shell traffic detection method proposed in literature [13] to realize reverse shell traffic detection, and then locate the target device.

(3). Vulnerability exploitation traffic detection module

The module includes session classification sub-module, feature extraction sub-module, and anomaly detection sub-module.

Among them, the session classification sub-module is responsible for dividing the sessions into three categories: normal traffic, suspicious single session and suspicious multiple session according to the data flow and packet length segmentation, which are stored in different folders respectively. It is convenient for subsequent feature extraction and anomaly detection. The key of the session classification sub-module is to segment the data packets in the session according to the arrival time, that is, to segment the time series of the session data packets, which is realized by Algorithm 2, and the relevant pseudocode is shown in Table .

Table 4. Session classification module Algorithm 2 pseudocode

```
Algorithm 2：
Input:time serieT， max_error
Output: Seg_TS
Fori=1; i<length(T);i++:
Seg_TS=concat(Seg_TS, create_seg-moment(T[i:i+1]));
Fori=1; i<length(Seg_TS);i++:
Merge_cost(i)=calculate_error([merge(Seg_TS(i). Seg_TS(i+1))])
  While(min(merge_cost)<max_error):
     index=min(merge_cost);
Seg_TS(index)=merge(Seg_TS(index),Seg_TS(index+1));
delete(Seg_TS(index+1));
     Merge_cost(index-1)=calculate_error(merge(Seg_TS(index-1),Seg_TS(index)));
     Merge_cost(index)=calculate_error(merge(Seg_TS(index),Seg_TS(index+1))).
```

The feature extraction sub-module extracts session features and builds a session relationship graph, which is stored in matrix form for anomaly detection.

The anomaly detection sub-module is responsible for traversing the session relationship graph constructed by the feature extraction module. According to the 12 session combination situations summarized in Section 3.2, it traces back the traffic at the early stage of vulnerability exploitation, integrates all the current suspicious session information, and uses the bi-selecting K-means algorithm to hierarchical process the traffic and visualize the attack process. To visually show the traffic behaviour characteristics of the whole process of vulnerability exploitation.

4. Experiments and Results

This chapter first introduces the general framework of our method and describes the functions and implementation details of all modules in detail. Then, we simulated ten vulnerability exploitation experiments. Compared with Blatta, a method based on a recurrent neural network to detect early exploit traffic. Results show that the detection rate of the proposed method is greatly improved, and the detection effect is not affected by the specific causes of vulnerabilities exploited and traffic encryption methods.

4.1. Data Set

Since the vulnerability exploitation traffic in the public data set usually only consists of shellcode delivery phase sessions, and the detection of our method relies on shellcode execution results, namely reverse shell session detection, it cannot be verified by exploit traffic in the public data set.

Therefore, in the end, we selected the traffic generated by the artificial reverse shell experiments to train the decision tree model and conducted ten code execution loophole exploit reverse shell experiments using the Metasploit penetration test framework. We collect the traffic generated in the process for tests and select the laboratory exit traffic as the background traffic. SeeTable for related vulnerability numbers and reverse shell methods. Among them, the traffic data using the Meterpreter module in Metasploit has a unique encapsulation format, which cannot directly obtain meaningful information in the payload content. Therefore, in the verification experiment, regardless of whether the traffic uses SSH and other encryption protocols, the traffic is considered encrypted traffic.

Table 5. Description of experimental data set

index	scenario	Network behaviors	Traffic type and quantity
1	Normal users surfing the Internet	Browse the web SSH login Transfer file Remote desktop	Background traffic (38512)
2	Unencrypted reverse shell	Metasploit modules unencrypted system shell	Unencrypted reverse shell traffic (77)
3	Encrypted reverse shell	Metasploit module encryption system shell, Meterpreter advanced shell	Encrypted reverse shell traffic (39)

4.2. Comparison Method

The questions answered in this section are:

RQ1: Can SRG-ETDetector detect both the shellcode delivery phase and shellcode exploitation phase traffic?
RQ2: Are the results of SRG-ETDetector interpretable?

In order to answer RQ1, experiment 1 is conducted in this paper. Firstly, the attacker is simulated to exploit the vulnerability ten times of code execution and reverse the malicious behaviour of the shell. Then, the traffic is collected and mixed with the normal Internet access traffic of users in the data set in Section 4.1. When selecting the comparison method, considering that the detection objective of this paper is to discover code execution vulnerability exploitation behaviour through traffic, among the existing detection methods described in the introduction, the ones that are closest to the detection objective of this paper are literature [11], and literature [11]is also the most recent. Therefore, the detection results of code execution vulnerability exploit traffic were compared with the results of the literature[11].

In order to answer RQ2, experiment 2 is conducted in this paper. Based on the bi-selecting K-means algorithm, hierarchical analysis is carried out on exploit using two-stage traffic, and visual processing is carried out on exploit using two-stage session traffic respectively, so as to visually show the behaviour characteristics of traffic transmission in the process of code execution vulnerability exploit, so as to prove the interpretability of test results.

4.3. Evaluation Criteria

In this section, the detection rate of encrypted traffic, non-encrypted traffic, and all vulnerability exploitation traffic is used to evaluate the performance of our method. The higher the detection rate, the lower the probability of code execution vulnerability exploitation behaviour 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 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 $M\_noncrypted$. The three detection rates are defined as follows.

$$rate\_crypted = \frac{M\_crypted}{N\_crypted} \qquad (1)$$

$$rate\_noncrypted = \frac{M\_noncrypted}{N\_noncrypted} \qquad (2)$$

$$rate\_total = \frac{M}{N} \qquad (3)$$

4.4. Results Analysis

4.4.1. Comparison of Detection Results of Ten Code Execution Vulnerabilities Exploitation Reverse Shell Traffic

Experiments 1 is to compare the effect of code execution vulnerabilities detection in ten actual application scenarios. The vulnerability types used in the experiment include remote command execution vulnerability and buffer overflow vulnerability and the reverse shell includes both encrypted and non-encrypted types. To verify the detection effect of the method in this paper on encrypted traffic, four kinds of reverse shell encrypted traffic are included in the experiment. The implementation methods include encrypting "one-sentence reverse shell" traffic through OpenSSL, encrypting traffic through SSL and RC4-related modules in Metasploit, and the traffic of the Meterpreter session shell is also invisible in plain text. In different periods of the day, the target machine that normally communicates with the network is used for code execution vulnerabilities to make it reverse shell. At the same time, Wireshark is run to collect all the traffic passing through the target machine during the experiment for detection, as shown in Table 4. To exclude the influence of the target operating system type on the detection effect, the two target operating systems used in the experiment were Windows XP and Kali 2017. Finally, the results of the detection of code execution vulnerabilities using reverse shell traffic are shown in Table .

Table 6. Comparison of detection results of ten vulnerability exploitation attacks

index	Vulnerability types and CVE number, and the specific ways to get a reverse shell	encrypted	Whether the exploit traffic is detected	
			Blatta[11]	SRG-ETDetetcor
1	Remote code executionvulnerability,CVE-2021-22205, Bash	×	√	√
2	Buffer overflow vulnerability,MS17-010,Python	×	√	√
3	Remote code executionvulnerability,CVE-2020-17530, Perl+OpenSSL	√	√	√
4	Remote code executionvulnerability,CVE-2019-15107,linux/x64/shell/reverse_tcp	×	√	√
5	Buffer overflow vulnerability, MS17-010, windows/shell_reverse_tcp	×	√	√
6	Buffer overflow vulnerability, CVE-2017-7494, python/shell_reverse_tcp_ssl	√	×	√
7	Remote code executionvulnerability,CVE-2019-0708, windows/Meterpreter/reverse_tcp	√	×	√
8	Buffer overflow vulnerability, CVE-2017-7494, linux/x64/Meterpreter/reverse_tcp	√	×	√
9	Remote code executionvulnerability,CVE-2019-0708, windows/Meterpreter/reverse_http	√	×	×
10	Remote code executionvulnerability,CVE-2019-0708, windows/Meterpreter/reverse_tcp_rc4	√	×	√

The experimental results show that only 5 exploits of code execution were detected in literature [11], while the method in this paper 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 literature [11] and the method in this paper, and there is no significant difference in the rate_noncryped index. However, for the 6 times of code execution vulnerability exploitation under traffic encryption conditions, only 1 time was detected in literature [11], while the method in this paper detected 5 times, and rate_crypted increased by about 66.7%. This is because the method in literature [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, the method in this paper 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 , the ninth vulnerability exploitation was not detected by either method. Through the analysis, the time code execution exploit reverse module is Windows/Meterpreter shell attack/reverse_http. The basic function implemented by this module is similar to the Windows/Meterpreter/reverse_tcp module, that is, the Windows target machine returns the Meterpreter session shell, but the former implements communication based on TCP protocol, while the latter implements communication based on HTTP protocol. In terms of traffic, the former is a TCP long connection initiated by the target machine. The latter is multiple HTTP short connections initiated by the target machine. The reverse shell detection method adopted in literature [13]is suitable for reverse shell traffic at the transport layer based on TCP protocol. Reverse shell traffic based on HTTP protocol communication is fundamentally different from it in session, so it cannot be used to 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 with 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 reverse shell against other protocols, to realize the detection of vulnerability exploitation traffic.

4.4.2. The Vulnerability Exploits the Traffic Stratification Results

Experiment 2 is to stratify the detected code execution vulnerability exploitation traffic using the bisecting K-means algorithm and visualize the code execution vulnerability exploitation traffic, thus further proving the interpretability of the classifier. The preset number of clusters in the bi-selecting K-means algorithm is set to 4, and the two-phase traffic of buffer overflow vulnerability CVE-2017-7494 reverse shell is stratified, and the results are shown in Figure 7and Figure 8respectively.

Figure 7shows the traffic stratification results of the shellcode delivery phase. In the shellcode delivery phase, the main purpose of the attacker is to transmit the shellcode to the target. The data packets sent by the attacker are mostly concentrated in the early stage of the session, the packet length is large, and the data flows to the target. Correspond to the blue, green, and purple clusters in Figure 7. At this time, the target passively receives the shellcode, so there are only response packets. Response packet payloads are typically shorter compared to shellcode, corresponding to the yellow clusters in Figure 7.

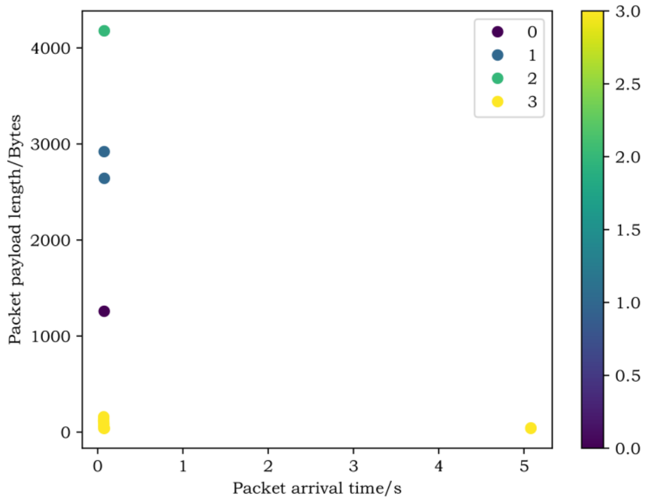

Figure 7. The traffic stratification results in the shellcode delivery phase

Figure 8 shows the hierarchical result of reverse shell traffic for the python/shell_reverse_tcp_ssl module. During the shellcode execution phase, the target returns the encryption shell. In the session establishment phase, the data packets of both sides interact frequently, and the length of the data packets is symmetric because the encryption protocol is negotiated. This process is very transient and corresponds to the yellow cluster in Figure 8. After the reverse shell is successful, the attacker usually executes the command to obtain the target information, and the overall data flow to the attack side. Usually, the packets carrying commands are short in length and relatively scattered. However, the target machine needs to return the corresponding execution results according to different commands, so the length of the data packets carrying the execution results is different, corresponding to the green cluster, blue cluster, and purple cluster above the X-axis in Figure 8.

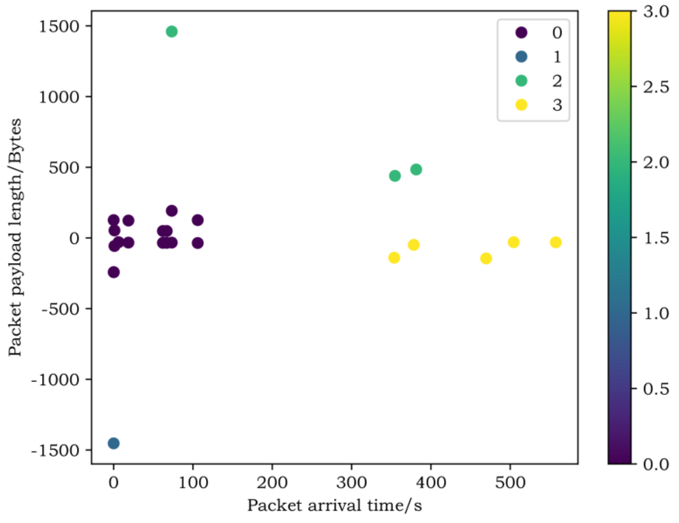

Figure 8. The traffic stratification results in the shellcode execution phase

5. CONCLUSIONS

This paper proposes SRG-ETDetector, an exploit traffic detection method based on session relationship graph analysis. Compared with the existing exploit traffic detection methods based on pattern matching of the payload, SRG-ETDetector can accurately detect the exploit session according to the effect of shellcode execution, independent of the specific cause of vulnerability and traffic encryption methods. It is a pity that our method is currently applicable to detecting code execution vulnerability exploits with traffic for the whole phase. We plan to analyse other exploit behaviours to expand the application scope of our method and realize real-time detection in the future.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] D. Kong, D. Tian, Q. Pan, P. Liu, and D. Wu, “Semantic aware attribution analysis of remote exploits,” Security and Communication Networks, vol. 6, no. 7, pp. 818–832, 2012.

[2] J. Wu, A. Arrott, and F. C. Colon Osorio, “Protection against remote code execution exploits of popular applications in windows,” 2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE), 2014.

[3] P. Parrend, J. Navarro, F. Guigou, A. Deruyver, and P. Collet, “Foundations and applications of Artificial Intelligence for Zero-day and multi-step attack detection,” EURASIP Journal on Information Security, vol. 2018, no. 1, 2018.

[4] Homoliak, M. Teknös, M. Ochoa, D. Breitenbacher, S. Hosseini, and P. Hanacek, “Improving network intrusion detection classifiers by non-payload-based exploit-independent obfuscations: An adversarial approach,” ICST Transactions on Security and Safety, vol. 5, no. 17, p. 156245, 2019.

[5] L. Chen, S. Sultana, and R. Sahita, “HeNet: A deep learning approach on Intel® processor trace for effective exploit detection,” 2018 IEEE Security and Privacy Workshops (SPW), 2018.

[6] S. Biswas, M. M. H. K. Sajal, T. Afrin, T. Bhuiyan & M. M. Hassan1, (2018) "A study on remote code execution vulnerability in web applications", International Conference on Cyber Security and Computer Science (ICONCS’18), 2018.

[7] F. M. Mokbal, W. Dan, A. Imran, L. Jiuchuan, F. Akhtar, and W. Xiaoxi, “MLPXSS: An integrated XSS-Based Attack Detection Scheme in web applications using multilayer perceptron technique,” IEEE Access, vol. 7, pp. 100567–100580, 2019.

[8] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos, “Network-level polymorphic shellcode detection using emulation,” Journal in Computer Virology, vol. 2, no. 4, pp. 257–274, 2006.

[9] K. Borders, A. Prakash, and M. Zielinski, “Spector: Automatically analyzing Shell Code,” Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), 2007.

[10] Y. Kanemoto, K. Aoki, M. Iwamura, J. Miyoshi, D. Kotani, H. Takakura, and Y. Okabe, “Detecting successful attacks from ids alerts based on emulation of Remote Shellcodes,” 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), 2019.

[11] B. A. Pratomo, P. Burnap, and G. Theodorakopoulos, “Blatta: Early exploit detection on network traffic with recurrent neural networks,” Security and Communication Networks, vol. 2020, pp. 1–15, 2020.

[12] B. Pratomo, “Low-rate attack detection with intelligent fine-grained network analysis,” dissertation, 2020.

[13] P. Irofti, A. Patrascu, and A. I. Hiji, “Unsupervised abnormal traffic detection through topological flow analysis,” 2022 14th International Conference on Communications (COMM), 2022.

[14] T. Yadav and A. M. Rao, “Technical aspects of cyber kill chain,” Communications in Computer and Information Science, pp. 438–452, 2015.

[15] J. C. Foster, M. Price, and S. McClure, Sockets, Shellcode, porting &amp; coding: Reverse engineering exploits and tool coding for security professionals. Rockland: Syngress Pub., 2005.

[16] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, “Evasion techniques: Sneaking through your intrusion detection/prevention systems,” IEEE Communications Surveys &amp; Tutorials, vol. 14, no. 4, pp. 1011–1020, 2012.
[17] H. A. Noman, Q. Al-Maatouk, and S. A. Noman, “Design and implementation of a security analysis tool that detects and eliminates code caves in windows applications,” 2021 International Conference on Data Analytics for Business and Industry (ICDABI), 2021.
[18] I. Stipovic , “Antiforensic techniques deployed by custom developed malware in evading anti-virus detection,” https://arxiv.org/abs/1906.10625, 2019.
[19] C. Leka, C. Ntantogian, S. Karagiannis, E. Magkos, and V. S. Verykios, “A comparative analysis of VirusTotal and desktop antivirus detection capabilities,” 2022 13th International Conference on Information, Intelligence, Systems &amp; Applications (IISA), 2022.
[20] M. Denis, C. Zena, and T. Hayajneh, “Penetration testing: Concepts, attack methods, and defense strategies,” 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT), 2016.

AUTHORS

Yajing Liu (1998-), female, master candidate, Cyberspace Security Academy, Information Engineering University. The research fields include malicious traffic detection and machine learning.

Ruijie Cai (1990-), male, master, lecturer of Cyberspace Security Academy, Information Engineering University. The research fields include network security, binary code analysis, and vulnerability mining.

Xiaokang Yin (1993-), male, Ph.D., Cyberspace Security Academy, Information Engineering University. The research fields include network security, binary code analysis, and machine learning.

Shengli Liu (1973-), male, Ph.D., professor and doctoral supervisor of Cyberspace Security Academy, Information Engineering University. The research fields include network attack detection and network infrastructure security.