Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

A new dynamic security defense system based on TCP_REPAIR and deep learning

Jianxun Tang[1], Mingsong Chen[2*], Haoyu Chen[3], Shenqi Zhao[2] and Yu Huang[2]

Abstract

Honeypot is an active defense mechanism, which attracts attackers to interact with virtual resources in the honeypot mainly by simulating real working scenarios and deploying decoy targets, so as to prevent real resources from being damaged and collect attackers' attack processes and analyze potential system vulnerabilities to proactively respond to similar attacks. Because of the existing honeypot system has defects such as the inability to deploy specific honeypots to induce attacks based on complex attacks, the inability to select the best honeypot for dynamic response based on honeypot deployment and maintenance costs during attack interactions, and insufficient ability to identify variants of known attack methods. Although hybrid honeypots can solve some of these problems by deploying low-interaction honeypots and high-interaction honeypots, they cannot really be applied to real production scenarios because of their slow TCP connection switching speed and inability to efficiently identify encrypted malicious traffic. In this paper, we propose a new dynamic security defense system based on the combination of TCP_REPAIR-based dynamic honeypot selection architecture and a deep learning-based intelligent firewall. The system accurately distributes encrypted or non-encrypted attack traffic and its variants through the intelligent firewall. The normal traffic is sent to the actual system, and the marked malicious traffic dynamically selects honeypots to respond according to the attack process. The experimental result indicated that the system can select honeypots for targeted responses according to the actual network situation quickly and dynamically and covertly, effectively improving the utilization rate of honeypot clusters as well as the ability to decoy.

Keywords Honeynet, TCP_REPAIR, Convolutional neural network, Software-defined networking, Encrypted traffic detection, Cyber security

Introduction

With the continuous development of the Internet, the security threats facing the Internet are also escalating. In the face of complex and changing attack methods, traditional network defense technologies such as firewalls, Intrusion Detection System (IDS), Intrusion Prevention System (IPS) and other passive defense methods [1] have difficulties in processing and responding to unknown and complex attack events such as encrypted malicious traffic rapidly. In order to cope with the current endless new network attacks and enhance the security of Internet information, the application of active defense technology has become an urgent need for today's network security defense technology.

Honeypot technology is an active defense technology that attracts attackers to conduct illegal applications by arranging decoy resources and then obtaining their attack process. Honeypot is essentially to arrange some bait hosts and network services to induce attackers to attack their exposed services, reproduce their attack process, analyze the attack flow and process, and apply them to the intrusion detection system to improve the network defense capability. Depending on their ability to interact

*Correspondence:
Mingsong Chen
cms@guet.edu.cn
[1] School of Ocean Engineering, Guilin University of Electronic Technology, Guilin, China
[2] School of Information and Communication, Guilin University of Electronic Technology, Guilin, China
[3] School of Electric Power Engineering, Guangxi Electrical Polytechnic Institute, Nanning, China

with attackers, honeypots can be classified into two types, High-Interaction Honeypots (HIH) and Low-Interaction Honeypots (LIH) [2]. High-interaction honeypots are usually built using real system environments, which can easily build complex honeypot environments with decoy functions; low-interaction honeypots are usually built using virtual simulation software, which isolates the real environment and is simple and less expensive to build, but it is easier to be detected by attackers because of the own virtual environment. The basic idea of hybrid honeypots [3] means that arranges a large number of low-interaction honeypots at the front-end and a small number of high-interaction honeypots at the back-end to achieve attack capture and analysis by migration of traffic. However, the traditional honeynet, due to its coarse-grained data control, leads to various deficiencies of hybrid honeypots such as data connection control migration.

Encrypted malicious traffic identification techniques mainly identify and classify encrypted traffic by analyzing the statistical and time series features of data streams, using statistical models such as machine learning algorithms (e.g., support vector machines, decision trees, random forests, and other algorithms) [4–7] and Gaussian mixture models [8]. Although the above methods can solve many problems that cannot be solved by port- and payload-based methods, there are still problems such as the inability to automatically extract and select features, the tendency of features to fail, and the need for continuous updating [9].

Based on the existing research work, this paper proposes a new dynamic security defense system based on TCP_REPAIR and deep learning, which can divide the traffic (including encrypted and unencrypted traffic) entering the system through an intelligent firewall. Therefore, normal traffic can access the real system normally and malicious traffic will be forwarded to the low-interaction honeypot automatically. When the low-interaction honeypot does not meet the attack depth, it will automatically switch to the high interaction honeypot. Normal traffic will continuously pass through the intelligent firewall in transit; once the malicious traffic is flagged in transit, it will automatically switch the connection to the high interaction honeypot. The hybrid honeypot with low interaction and high interaction honeypots collects the attack process to provide a new type of attack process for the system and improve the security performance of the network. The main contributions in the paper consist of major two parts. The first part is to add the malicious encrypted traffic identification based on deep learning in the active defense system. It can make up for the defect that the traditional intrusion detection system in the hybrid honeypot can only identify the non-encrypted traffic. Secondly, we use the TCP live migration module in the Linux underlying network communication. By saving the negotiation fields in the original connection in advance, the TCP_REPAIR technology is used to directly create a new TCP connection when the TCP connection is transferred. It can optimize the shorts in the traditional TCP connection switching process, which takes a long time during switching and is not hidden in the switching process. In this paper, we firstly summarize the current research on Software Defined Network (SDN), hybrid honeynets, and malicious traffic identification, secondly elaborate on the solution of this system, and finally design and implement the prototype system. The system is experimentally proven to have good performance. The rest of this paper is organized as follows: "Current status of research" section introduces the current research status of SDN, hybrid honeynet, and encrypted malicious traffic identification; "The proposed program" section proposes the architectural solution of this system and the specific implementation plan of each module; "Functionality review" section evaluates the indicators of the system through the overall usability and quantitative performance tests of the system; "Concluding remarks" section summarizes the problems of the system and future research directions through the test data of the system.

Current status of research

Software defined network

SDN [10] is a new network innovation architecture proposed by the Clean-Slate research group at Stanford University, which is an implementation of network virtualization. Its core technology, OpenFlow [11], provides a good platform for core network and application innovation by separating the control plane of network devices from the data plane, thus enabling flexible control of network traffic and making the network more intelligent.

SDN uses the idea of stratification to separate data from control. The overall architecture of SDN is divided into a data plane, control plane and application plane from south to north [12], as shown in Fig. 1. The data plane consists of switches and other network general-purpose devices that provide simple data forwarding functions, besides the SDN links are formed by different protocol rules between different network devices. The main function of the control plane which consists of the controller is to extract the matching function of route calculation from the traditional routers and let the controller perform the calculation uniformly; create forwarding rules for network traffic by mastering the global network information, and then distribute flow tables to the switches through the OpenFlow protocol [13]. The application plane mainly contains SDN-based applications, and users can deploy new applications through the northbound interface without understanding the underlying details, which provides great convenience for the installation of intrusion prevention systems.

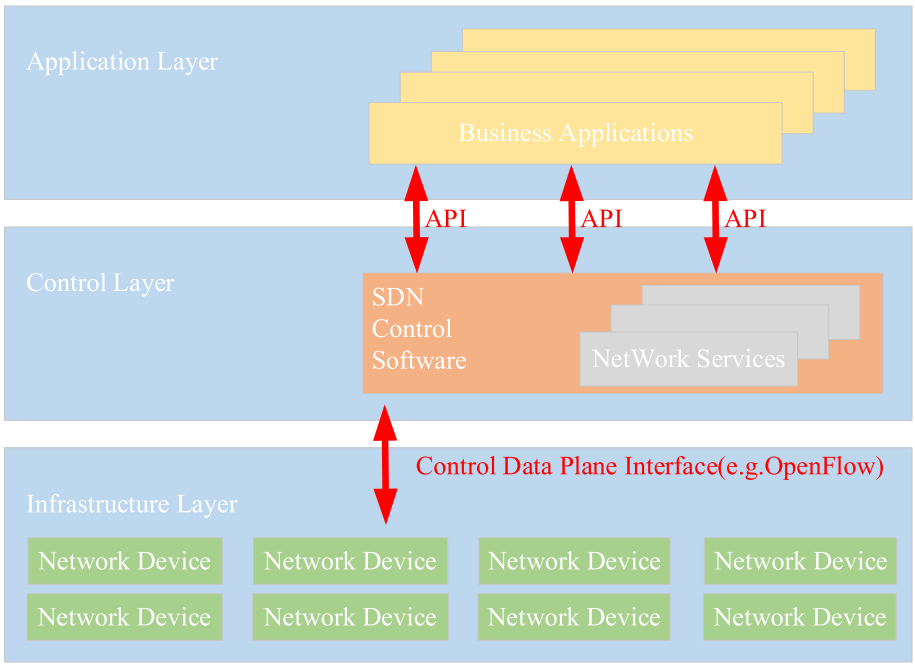

Fig. 1 SDN overall architecture diagram

SDN controllers have developed at a high rate in recent years and at this stage are divided into two main factions, one is the camp of network service providers such as operators, of which the main representatives are ONOS controllers [14] and Ryu controllers [15], and the other is the camp of network equipment vendors, of which the main representative is the OpenDayLight controller [16]. Since OpenDayLight controllers and ONOS controllers are mainly used for industrial applications and their architectures are more complex, scientific experiments usually use lightweight Ryu controllers. The current stage of the Ryu controller solution is mainly based on the OpenFlow protocol, where forwarding rules are first created in the controller and passed through the OpenFlow protocol to the data plane to the switch, which matches the destination address according to the flow table. The main features of the OpenFlow protocol have changed significantly from version 1.0 released in 2009 to version 1.5 in 2015, but the main enhancement is that version 1.5 natively supports TCP keyword matching [17], so it can intercept TCP packets (including three handshakes) and can play an important role in task of targeted data interception based on TCP message headers.

Hybrid honeynet

The hybrid honeynet is mainly divided into front-end and back-end, where the front-end usually arranges a large number of low-interaction honeypots and the back-end usually arranges a small number of high-interaction honeypots [18]; when the attacker's attack arrives, the attack answers firstly through the front-end; when the attack depth is too deep it will let the back-end answer through connection conversion to make the overall utilization of the hybrid secret network increase [19].

The core of the hybrid honeynet is mainly the method of connection switching compared with the traditional honeynet.

Artail [20] et al. use Honeyd's built-in proxy function to redirect traffic to a high interaction honeypot, but this method does not install traffic filtering, so the backend is occupied by a large number of invalid connections, increasing the burden on the whole system.

Fan [21] et al. proposed HoneyDOC architecture. The SDN store-and-forward strategy is used to switch TCP connections. During the connection switchover, the previously stored packets are replayed, and then the connection is established with the back-end honeypot. The scheme has excessive storage requirements because the massive storage of malicious traffic packets is too burdensome for the system when a large number of attacks arrive.

Berthier [22] et al. proposed HoneyBrid architecture based on traditional hybrid honeynets, which mainly implements redirection through NAT address translation devices, and then processes the Seq, ACK, and window size of TCP and then replay the message to achieve

connection switching. Due to the use of NAT address translation, the switchover is perceived during the actual connection.

Encrypted malicious traffic classification

With the increasing awareness of Internet security and privacy protection and the widespread use of Hyper Text Transfer Protocol Secure (HTTPS), the encryption of Internet communication has become an unstoppable trend. Although traffic encryption makes information interaction more secure, encrypted traffic also poses a great challenge to Internet security (e.g., using encryption technology to spread illegal information). Traditional traffic classification is mainly based on port, payload and flow approaches. Among them, port-based approaches typically use the default port numbers of TCP and UDP protocols to infer the type of service or application. However, the use of port masquerading random and tunneling techniques can evade the detection of this method. Payload-based methods, also called deep packet parsing Deep Packet Inspection (DPI) techniques [23], mainly match the contents of packets, and the method cannot handle encrypted traffic. The flow-based method mainly uses statistical and timing characteristics of packets using machine learning algorithms for modeling and identification, where the statistical models used to identify encrypted traffic are mainly Gaussian mixture and other models.

Machine learning-based traffic classification methods mainly use machine learning algorithms to model the identification for the problem. Soleimani [24] et al. use machine learning methods for Obfs3, Obfs4 and ScrambleSuit traffic obfuscation plugins commonly used in Tor. It can quickly detect obfuscation-based plugins Tor traffic identification by using the information of the first 10–50 packets of each stream, but the detection direction of this method is too homogeneous. ACETO [25] et al. compared multiple deep learning algorithms with Random Forest (RF) based on different encrypted traffic datasets and showed that most of the deep learning algorithms outperformed the Random Forest algorithm.

Deep learning-based traffic classification methods achieve high accuracy in classifying traffic mainly by using traffic features (packet-level features, session features and statistical features), etc., through neural network models such as Convolutional Neural Networks (CNN), Recurrent Neural Network (RNN), and AutoEncoder (AE) [26]. WANG [27] et al. used a one-dimensional vector to represent each flow or session to train CNN models, and the results demonstrated that the accuracy using CNN was much higher than the C4.5 method using temporal and statistical features. CHEN [28] et al. transformed temporal data into two-dimensional images and trained CNNs using two convolutional layers, two pooling layers, and three fully connected layers, and the results showed that the method outperformed machine learning methods.

The proposed program

Overall architecture

Based on the existing research work, the proposed new defense system architecture is shown in Fig. 2.

As can be seen from the above figure, the initial traffic discrimination is performed by the intrusion detection system Snort in OVS when the traffic enters the system. If Snort detects that the traffic is normal and encrypted, the traffic is reviewed by the deep learning based malicious traffic detection module 1D-CNN to see if the traffic is malicious. If it is normal then it is sent to the normal host, otherwise it is malicious then it is sent down through the Ryu controller The flow table forwards the traffic to the low-interaction honeypot created by HFish. When the attack depth reaches a critical point, HFish activates the copy of the normal host of the timed snapshot as a high-interaction honeypot, while using a random fill form to overwrite the sensitive data, and then switches the connection to the high-interaction honeypot using the TCP_REPAIR Proxy.

Encrypted malicious traffic identification scheme

IDS is a network device or application that monitors network transmissions in real-time. The overview of traditional IDS traffic classification methods mainly includes four categories: (1) pre-attack port scanning; (2) detection of payloads containing attacks in the communication process; (3) statistical traffic characteristics;(4) behavioral analysis. Due to the development of random ports and port disguise and the popularity of traffic encryption, coupled with the low accuracy of the pre-attack port scanning method, and the detection of payloads in the communication process requires decryption of encrypted traffic, the traditional IDS for traffic classification methods are no longer applicable to current network communications. Traditional statistical methods are used to analyze the strategy of big data, which usually can not do real-time analysis, mainly through a large number of a priori knowledge in the traffic to match. Behavioral analysis methods are similar. These two approaches are summarized in two processes: feature extraction and feature matching. At this stage, the most advanced techniques for feature extraction training are machine learning and deep learning. Therefore, the use of machine learning or deep learning-based traffic classification can be applied to the current traffic encryption development status.

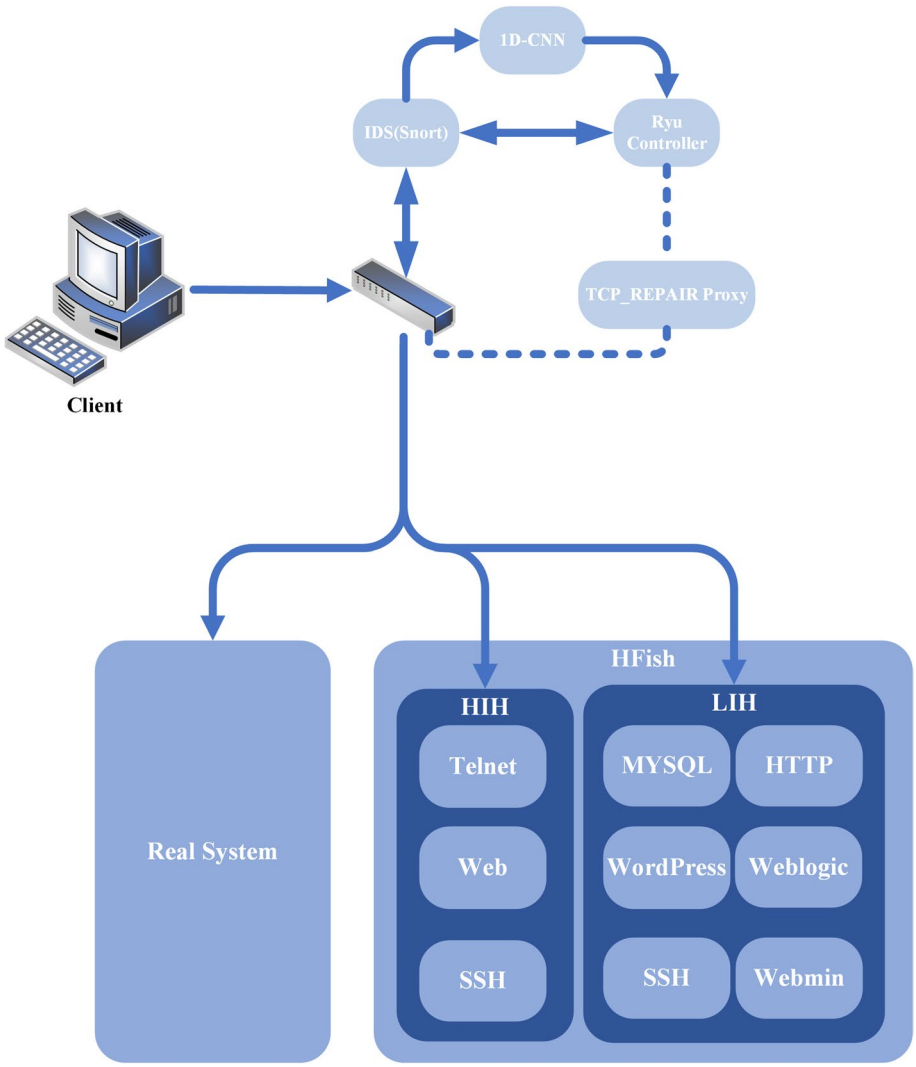

Fig. 2 Overall system architecture

Comparison of machine learning-based and deep learning traffic classification

Machine learning traffic classification is generally divided into four steps: firstly designed traffic features and packet features manually; secondly, followed by extraction and selection from the original traffic according to the manually designed features; thirdly designed classifiers such as decision trees manually; finally input from the extracted and selected features into the classifier to get the output of traffic classification. This process is essentially a divide-and-conquer idea, and the disadvantage of the divide-and-conquer method is that the locally optimal solution is not necessarily globally optimal.

Traffic classification by deep learning is generally divided into two steps: first input of raw data and then deep learning algorithms such as CNN and RNN are used to learn the original data features and train the deep learning model at the same time. Finally, the trained deep learning model outputs the classification results for the new input data [29]. Depending on whether traffic features are required and thus classified into supervised, semi-supervised and unsupervised learning [30].

Both hand-designed features and hand-designed classifiers in machine learning-based traffic classification require expert knowledge. Different feature designs and classifier designs can largely affect the final classification results. The method of feature extraction in deep learning is widely adaptable and more flexible in application.

The traffic classification module integrated in IDS needs to have end-to-end traffic classification function.

Although machine learning can be used to identify encrypted traffic, the Disadvantages mainly include the inability to automatically extract features, the tendency of features to fail, and the need to manually design features and classifiers, which is contrary to the need for immediate traffic classification in IDS. Deep learning-based traffic classification is selected based on the demand for greater extraction of encrypted traffic features to accurately identify malicious traffic.

1D-CNN model design

Deep learning is classified into unsupervised learning, semi-supervised learning. Because the accuracy of unsupervised learning classification is not high, and the acquisition of traffic characteristics required by supervised learning requires a lot of computing power, a semi-supervised learning method combining labeled data and unlabeled data is selected.

Since CNN is mainly suitable for data where features can appear anywhere, objects are not subject to translation and distortion, and have strong local relevance. The traffic fits exactly the above characteristics. Therefore, CNN is chosen as the deep learning algorithm for training. The commonly used CNNs include 1-dimensional CNN, 2-dimensional CNN, and 3-dimensional CNN. 1-dimensional CNN is mainly applicable to sequence data, 2-dimensional CNN is mainly applicable to image and audio data, and 3-dimensional CNN is mainly applicable to data such as video and volume images [31]. Although it has also been studied to classify traffic images by transforming the traffic into 2D images and then using 2D-CNN, 1D-CNN was chosen as the training algorithm based on the fact that the traffic itself is sequential data because the matrix computation of the images requires strong computer performance and is more time consuming to compute.

Based on the deep learn-based encryption traffic classification proposed by Wang [32] et al., the original model is improved according to the specific classification tasks in the system. The operation flow of the improved 1D-CNN model is shown in Fig. 3.

As can be seen from the above figure, the improved 1D-CNN model mainly includes two parts, namely, the

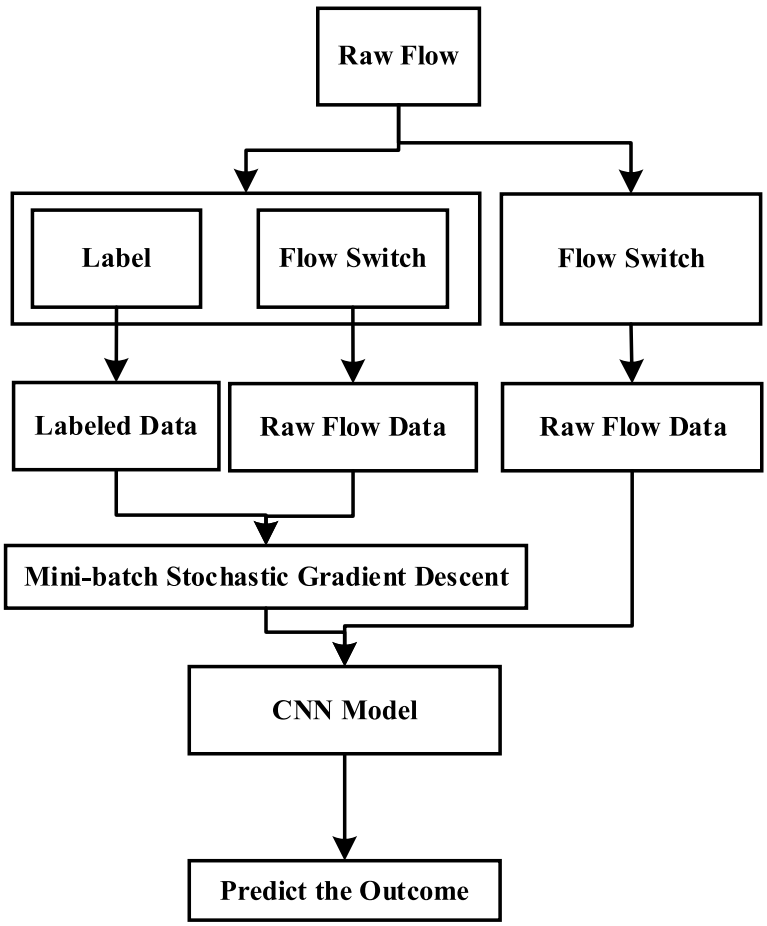

Fig. 3 1D-CNN operation flow

preprocessing module and the training output module. The preprocessing module generates labeled and unlabeled data by processing the original dataset, where part of the unlabeled data is used for testing and part for training. Since the model itself training process will produce errors, the Mini-batch Stochastic Gradient Descent (MBSGD) method is selected for model optimization. The final processed data and the originally reserved unlabeled data are used as the input of the 1D-CNN model, and then the model outputs the prediction results through training. The overall architecture of the CNN model is shown in Table 1.

Custom software defined networking SDN

Custom SDN consists of two main modules: Forwarding Decision Engine, and Redirected Forwarding Engine (TCP_REPAIR Proxy). It mainly contains traffic identification, filtering and TCP switching.

Forwarding decision engine

The forwarding decision engine is mainly composed of the intrusion detection system Snort, 1D-CNN and Ryu controller. It mainly consists of four forms of work. The first forms of work: To begin with, the traffic arrives at Snort, and Snort detects the port and payload of the traffic. If the port is an open port of the system and the payload is normal, it is marked as 0. If the traffic is encrypted, input it to 1D-CNN module for testing again. If the traffic is normal after 1D-CNN testing, mark 0(at this time, the traffic is marked as 00), then it is determined that this traffic is normal traffic. The traffic is forwarded directly to the normal host by OVS through the Ryu controller downstream flow table, and the payload of the subsequent traffic does not have any abnormalities. The second one: Based on the first one, if Snort or 1D-CNN detects an exception in the subsequent payload (that is, the traffic is marked as 1), the traffic will be directly forwarded to the high interaction honeypot of the corresponding service. The third one: The traffic reaches Snort firstly, and Snort detects the port and payload of the traffic. If the port is an open port of the system and the payload is normal, it is marked as 0. Then it is input to the 1D-CNN module for testing again. After the 1D-CNN test, the abnormal traffic is marked as 1 (at this time, the traffic is marked as 01), and the traffic is forwarded to the low interaction honeypot corresponding to the request port. The fourth one: On the basis of the third, if the payload of the attack process is determined by the action in Snort's msg to require forwarding to the high interaction honeypot, TCP_REPAIR is started to transfer the traffic originally connected to the low interaction honeypot to the high interaction honeypot. The specific flow is shown in Fig. 4.

The more important aspects of the system are the connection selection algorithm, and the Snort alarm rules.

Snort has three main modes: sniffer, packet logger, and network intrusion detection system. Sniffer mode simply grabs packets from the network and displays them on the terminal; packet logger mode saves packets to disk; network intrusion detection mode is the most complex, with high configurability. It allows Snort to analyze network traffic and react according to user-defined rules.

Due to the needs of the system, therefore, choose the Network Intrusion Detection System (NIDS) mode, which can respond by analyzing network traffic.

Snort alarm rules are as follows

Content 1: Snort alert rules

```
alter protocol source-ip source-port packet flow direction (->,<-,<>) destination-ip destination-
port(logto: "rule name"; msg: "alarm message"; sid:sequence number;priority:sequence
number;content : "Search for a specific pattern string in the packet's payload weight." ; )
```

Examples:

alter tcp any any - > 10.37.23.59/18 22 (logto: “ssh”; msg: “HIH”;)

TCP traffic accessing port 22 of network 10.37.23.59/18 triggers this warning and issues a warning rule with the name ssh and the content HIH.

Because of the warning rules, the design creates four actions in the msg warning content, Real_System,1D_CNN, LIH, HIH, which represent forwarding to normal host, forwarding to 1D_CNN, forwarding to low interaction honeypots, and forwarding to high interaction honeypots, respectively. Forwarding to a specific honeypot requires the Ryu controller to send down the flow table to the switch to

Table 1 CNN model parameters

Layer	Operation	Input	Kernel	Stride	Pad	Activation Function	Pooling Type	Pooling Size/ Stride/Padding	Output
1	CONV	784*1	25*1	1	12	ReLU	Max	3*1/3/1	262*32
2	CONV	262*32	25*1	1	12	ReLU	Max	3*1/3/1	88*64
3	Full Connect	88*64	–	–	–	–	–	–	1024
4	Full Connect	1024	–	–	–	–	–	–	2
5	Softmax	2	–	–	–	–	–	–	2

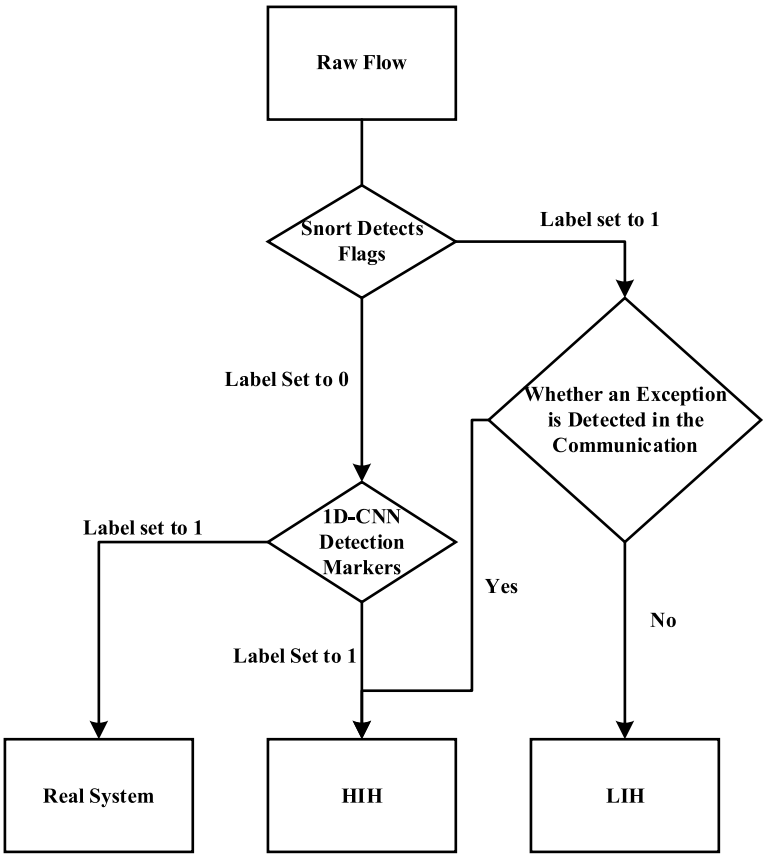

Fig. 4 Traffic forwarding flow chart

complete, so the Ryu controller completes the forwarding after using Snort's socket transmission alarm.

Pseudocode for connecting the selection engine in the controller.

Data: Payload pkt

Result: Decision

1 Decision-Making Method:

2 If payload then

3 Payload packets are sent to Snort

4 Snort detects payload packets using rules and generates alter msg messages

5 If msg=="1D_CNN" then

6 Encryption flow was sent to 1D CNN

7 End if

8 Snort sends alarm information to the Ryu controller

9 End if

Algorithm 1. Connection Selection Engine Algorithm

Redirected forwarding engine (TCP_REPAIR proxy)

The normal TCP connection process is shown in Fig. 5. Note that above the orange line are the TCP three-way handshakes, and below the orange line are the data exchanges after the TCP connection is established.

In a hybrid honeynet system, low-interaction honeypots mainly work at the early stage of the attack, that is the network segment information scanning phase. During the scanning probe phase, the attacker usually does not establish a full TCP connection with the target host in order to avoid the scanning behavior being recorded. Even if a TCP full connection scan is used, no subsequent data exchange occurs with the target host. Therefore, during a typical TCP connection, the arrival of the first PSH packet after three TCP handshakes can be used as a signal for connection transfer. The 1D_CNN of the smart firewall in this system detects encrypted malicious traffic as a signal for connection transfer to occur, and the steps are similar.

This module is mainly inspired from TCP connection switchover method proposed by Cunha [33] et al. to perform connection hot migration at any stage of a

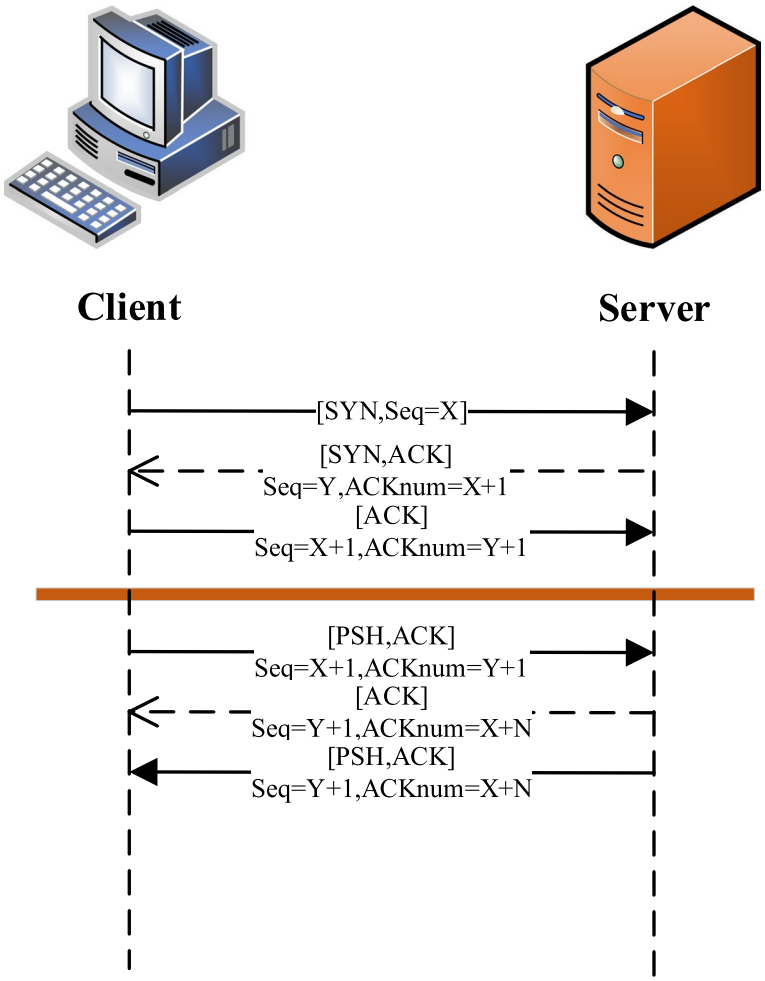

Fig. 5 Flow chart of TCP connection and data exchange

TCP connection through TCP_REPAIR mode in Linux, which greatly reduces the time loss and throughput loss of the switching process through the underlying Linux communication. The specific connection transfer process is shown in Fig. 6. The typical TCP connection communication process is shown above the orange line, and the data connection transfer process is shown below the orange line.

The steps are described in detail below.

Step 1: The attacker wants to establish a TCP full connection with the real system via TCP three times handshake. At this point, since the attacker has not yet made a real attack, the Ryu controller will first intercept the three times handshake messages between the attacker and the real system by sending flow rules matching the relevant TCP keywords to the switch via OpenFlow 1.5, recording the TCP negotiation parameters in them, and then the Ryu controller sends the flow table to the switch, which sends the connection messages to the real system. All TCP messages during the three handshakes are sent to Snort for detection, and the alert message msg is sent to the Ryu controller for determination. This step successfully isolates the large amount of scanned data in the network.

Step 2: After the Ryu controller receives the PSH packet, it first extracts the key fields such as sequence number, answer number and identification number from the packet and saves the connection information. When the Ryu controller receives an alert message msg "HIH" from the IDS or when the encrypted traffic is marked as "1" by 1D-CNN, the Ryu controller initiates a TCP connection switch. First of all the Ryu controller sends Flow-Mod, which temporarily takes over the connection from the attacker to the real system, and enters the attacker's message into the Ryu controller via Packet-In to prevent the real system from replying (to avoid further data leakage). Then the TCP_REPAIR Proxy

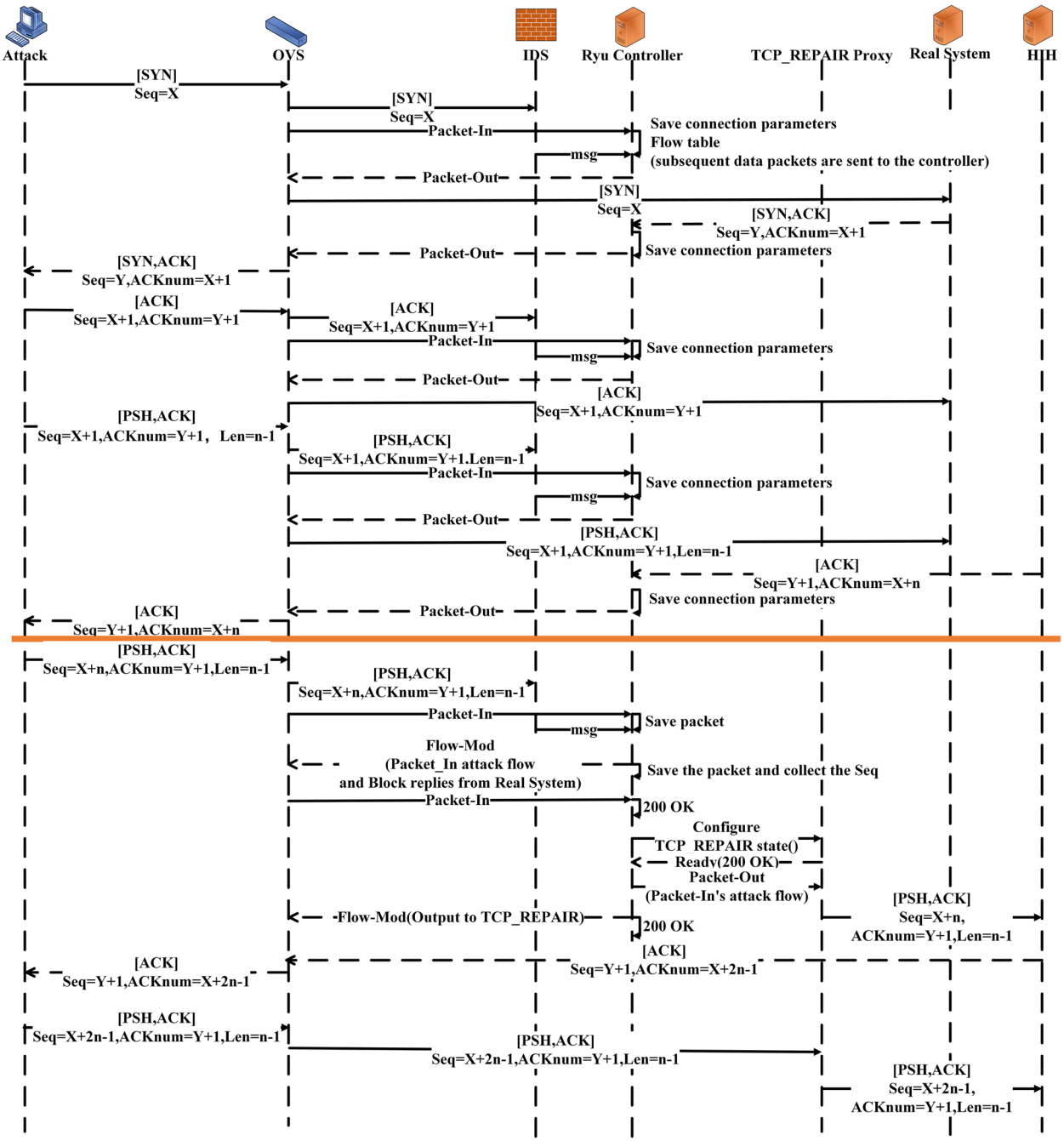

Fig. 6 Detailed flow of TCP switching

creates a socket and a new regular socket to connect to the high interaction honeypot. When the Ryu controller issues a "200 OK" to confirm that the system is OK, it uses the TCP_REPAIR socket and the original TCP_REPAIR sockets saved by the Ryu controller with the parameters of the real system. When the Ryu controller receives the "200 OK" confirmation from the TCP_REPAIR Proxy that the system is ready, the Ryu controller temporarily connects the attacker to the real system. When the Ryu controller receives the "200 OK" confirmation from TCP_REPAIR Proxy that it is ready, it passes the packets of the attacker and the real system temporarily managed by the Ryu controller to TCP_REPAIR Proxy via Packet-Out. TCP_REPAIR Proxy to submit the subsequent messages of this connection to the high interaction honeypot through the proxy to complete the connection transfer.

Data: Migrate Data

Result: True or False

1 Live Migration Method:

2 If Socket is not in the REPAIR then

3 Socket Switch to the REPAIR Mode

4 End if

5 Configuring Port Reuse

6 Send an ACK message (message content: negotiation data of the active TCP) to the socket to be migrated with the Seq number of the latest confirmation

7 Active TCP Data Backup Restore

8 End

Algorithm 2. TCP_REPAIR Socket Live Migration Algorithm

During connection transfer, the controller needs to save the negotiation parameters of the three TCP handshakes of the original active connection when implementing connection transfer through the sockets created by TCP_REPAIR. The following are the negotiation parameters that need to be intercepted by OpenFlow 1.5 during the TCP handshake phase.

Window size

This field is for the receiver to inform the sender of the number of bytes it can currently receive.

TCP Optional

TCP options are located at the end of the TCP header, and only those options that will appear in [SYN] and [SYN, ACK] messages are analyzed here.

(1) Maximum Segment Size (MSS). Both sides of the TCP connection declare the MSS value in their respective SYN packets and choose the smaller of the two values as the negotiated value at the end of the handshake. The MSS is the most important of the available options and the default size is 1460. Since this value is affected by other forwarding devices on the chain, it needs to be set comprehensively considering the actual situation. To avoid packet loss, enable the path MTU discovery mechanism for high interaction honeypots and set the firewall to allow ICMP message segmentation messages to pass. Alternatively, a simpler approach can be taken by setting the MSS to a smaller value of 1380 to accommodate most link conditions.

(2) Timestamp. The timestamp is used to calculate the round-trip delay. The sender puts the current timestamp into TSval when sending the message, and the receiver puts it into TSecr when acknowledging the message. If the TCP communication parties negotiate the use of a timestamp, this option is included in each packet. As a consequence, the controller can determine it through the PSH message. This value is modified in a similar way to the sequence number and answer number.

(3) Other options, such as SACK, window expansion factor, etc. Usually, since these options are rarely used in the SYN packets between the client and the server and cannot be inferred from the PSH packets, the proposed scheme in this paper does not use these optional options.

Other fields

The source and destination ports can be modified to the SDN switch to be modified according to the mapping between the high and low interaction honeypots. The fields such as checksum value, flag bit, and the emergency pointer can be set accordingly.

Identifier in the IP packet header

The initial value is randomly generated and incremented by 1 for client-side SYN packets, ACK packets and the first PSH packet, while the value is 0 for server-side SYN/ACK packets. This field does not need to be modified after connection transfer.

Functionality review

System testing

Test environment construction

In this section, we will build the test environment according to the scheme proposed in "The proposed program" section. The test system is selected from OpenFlow protocol version 1.5, and the controller framework is selected from Ryu, a lightweight Python-based framework. One OVS is deployed for traffic filtering and connection transfer. The test environment is shown in Fig. 7.

The test system includes: (1) the external network where the attacker is located (10.37.52.0/24); (2) The internal network where the real system and the hybrid honeynet system are located (10.37.23.0/24). The hybrid honeynet includes a low-interaction honeypot in the front and a high-interaction honeypot in the back; (3) Controller Ryu (because Ryu is lightweight, easy to build and easy to test), in which OpenFlow is selected version 1.5 (support TCP keyword flow matching); (4) SDN supporting OVS switch connecting the two networks.

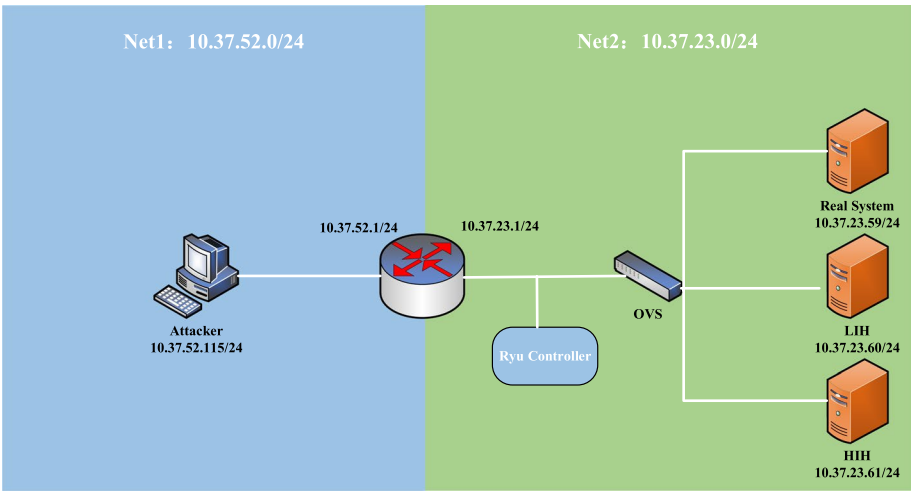

Fig. 7 Test environment framework

For these test configurations use the following Snort rules, as shown in Content 2.

Content 2: Some of the Snort rules

```
alter tcp any any -> any 21(msg:"MIH";sid:1000002;priority:2;)
…
alter tcp any any -> any 25(msg:"HIH";sid:1000006;priority:2;)
…
alter tcp any any ->any 8800(msg:"Real_System";sid:1000009;priority:2;)
…
alter tcp any any -> any any(msg:"1D_CNN";sid:1000012;priority:0;)
...
```

Through the above rules for the first step of the intrusion detection system IDS traffic filtering classification, the msg message will be passed to the SDN-enabled OVS according to the flow table rule matching for traffic forwarding work.

The flow table entries on the OVS are shown in Content 3.

Content 3: OVS partial flow table entries

```
cookie=0x0 , duration=7.361s , table=0 , n_packets=0 , n_bytes=0 , priority=100,ct_state=-trk,tcp a
ctions=ct(table=0)
cookie=0x0 , duration=7.347s , table=0 , n_packets=0 , n_bytes=0 , priority=100,ct_state=+new-rel
+trk,tcp,tp_dst=21 actions=ct(comnit,alg=ftp) , resubnit(,1)
cookie=0x0 , duration=7.331s , table=0 , n_packets=0 , n_bytes=0 , priority=100,ct_state=+new+re
l+trk,tcp actions=output: "s1-eth1"
cookie=0x0,duration=7.320s, table=0,n_packets=0 , n_bytes=0, priority=5o,ct_state=+new+trk,tcp ac
tions=ct(commit),resubnit( ,1)
cookie=0x0 , duration=7.310s , table=0 , n_packets=0 , n_bytes=0 , priority=50,ct_state=+est+trk ,
tcp actions=resubmit( ,1)
```

Simulation test

In order to verify the availability of the system, the Netcat tool was used to simulate a client machine and three servers (a real server, a low-interaction server, and a high-interaction server). The reasons for using the Netcat tool include two main aspects. The first reason is that Netcat can use "ncat" instructions to encrypt data transmission in addition to the regular "nc" instructions using plaintext, which allows testing the effectiveness of the machine's identification against malicious encrypted traffic. The second reason is that Netcat automatically disconnects by default, when a TCP session is established and does not allow recovery.

Experiment 1: Verify the problems that Netcat has with normal connection switching.

Test 1: Netcat environment basic test, mainly tests whether the server does not allow reconnection after a TCP session is disconnected.

First, connect the client to the real server, and use SDN to simply transfer the traffic to the high interaction server. Then use the client to send data to the real server, but the result is that the real server does not receive the data. Finally use the client to send data to the high interaction honeypot, but the high interaction honeypot does not receive the data, either.

Test 2: Based on Test 1, then configure the TCP traffic steering rules.

The experiment assumes that the attacker tries to restore the connection, and as a result, the client and the highly interactive honeypot can make a normal connection.

Test 3: Connect to the real system on the basis of test 2, when the real system is also not allowed to connect.

Test 4: Connect again on the basis of test 3, when the high interaction honeypot also does not allow connection.

Experiment 2: Verify the use of SDN controllers to control traffic for traffic forwarding and resolve normal connection switching.

First, a connection between the client and the real system is established using Netcat. Unlike experiment 1, where the switch is made using SDN instead of simply steering, at which point the TCP session data is confirmed and the TCP_REPAIR proxy is configured to the correct value. As a result, when the TCP_REPAIR proxy is configured and new sockets are instantiated, the high interaction server can be seen. During the handover process, the client did not receive any errors. Further simulation of the attack interaction reveals that the client and the high interaction honeypot can send and receive data to and from each other. Re-run the experiment again using the low interaction honeypot to the high interaction honeypot to determine again the feasibility of this connection switch.

Experiment 3: Test system connectivity using encrypted traffic.

First, create a connection between the client and the server using Netcat. Then use ncat to transfer ssl encrypted traffic in the client and check the output results. In consequence, the traffic is normal and enters the server without any problem. At last, we use ncat to pass the self-started Trojan data in the client and encrypt it with ssl, and we can see that the sub-Trojan data is received in the high interaction honeypot.

Real environment testing

Use the test system set up in "Test environment construction" section. Use Wireshark to listen to the connection network connection of each host to observe the connection changes between the front and back end, as shown in Fig. 8 shows the traffic exchange graph before and after the switch using TCP_REPAIR. Where the start time is the time when listening began using Wireshark.

The first half of the above figure shows the initial TCP connection established between the attacker and the front end, so the attacker needs to establish a connection with the front end through three handshakes. Then interact with the front end, when Snort sends "HIH" will automatically trigger Ryu to send down the flow table to transfer the traffic from the front end to the back end. At the same time, TCP_ REPAIR mode starts, as shown in the bottom half of Fig. 8. First, the controller will send an ACK message with the sequence number using the sequence number of the last byte of data confirmed by the last ACK from the opposite end, and use the original saved window size of the three-way handshake negotiation with LIH. And HIH will return the confirmed ACK message, negotiate the connection parameters. Then perform a backup of data, sequence number, TCP options, timestamp, TCP window Restore, exit TCP_ REPAIR mode, hot migration is complete, disconnect the original attacker from the LIH, and subsequent interactions use the attacker with the HIH. Here we should note that the time shown by Wireshark is relative to the first packet captured by the network interface it listens to during the test (independent of the time the connection was initiated).

Performance tests

1D-CNN performance test

To evaluate the SDN-based security defense system's ability to identify encrypted malicious traffic, traffic redirection, and system stress resistance. The performance of the 1D-CNN module for identifying encrypted malicious traffic is tested using three main metrics: accuracy, precision, and recall.

The equations are respectively.

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

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

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

In the above formula, TP represents the number of samples in which both predicted and true values are positive. FP represents the number of samples with negative predicted values and positive real values. FN represents the number of samples whose predicted value is positive and whose true value is negative, while TN represents the number of samples whose predicted and true values are both negative.

The experiments for performance testing of the 1D-CNN module for identifying encrypted malicious traffic are set up with three main comparison experiments, namely malicious encrypted traffic identification; regular encrypted traffic classification; and encrypted traffic classification. Since the two most common traffic representations are session and flow [4]. The session is a unit of traffic divided according to the 5-tuple of source IP, source port, destination IP, destination port, and transport level protocol (5-tuple). A flow is very similar to a session, except that it contains traffic in only one direction and the source IP/ port and destination IP / port are not interchangeable. Also, since normal traffic is usually divided into encrypted and non-encrypted, the ISCX dataset is used for the test to combine encrypted

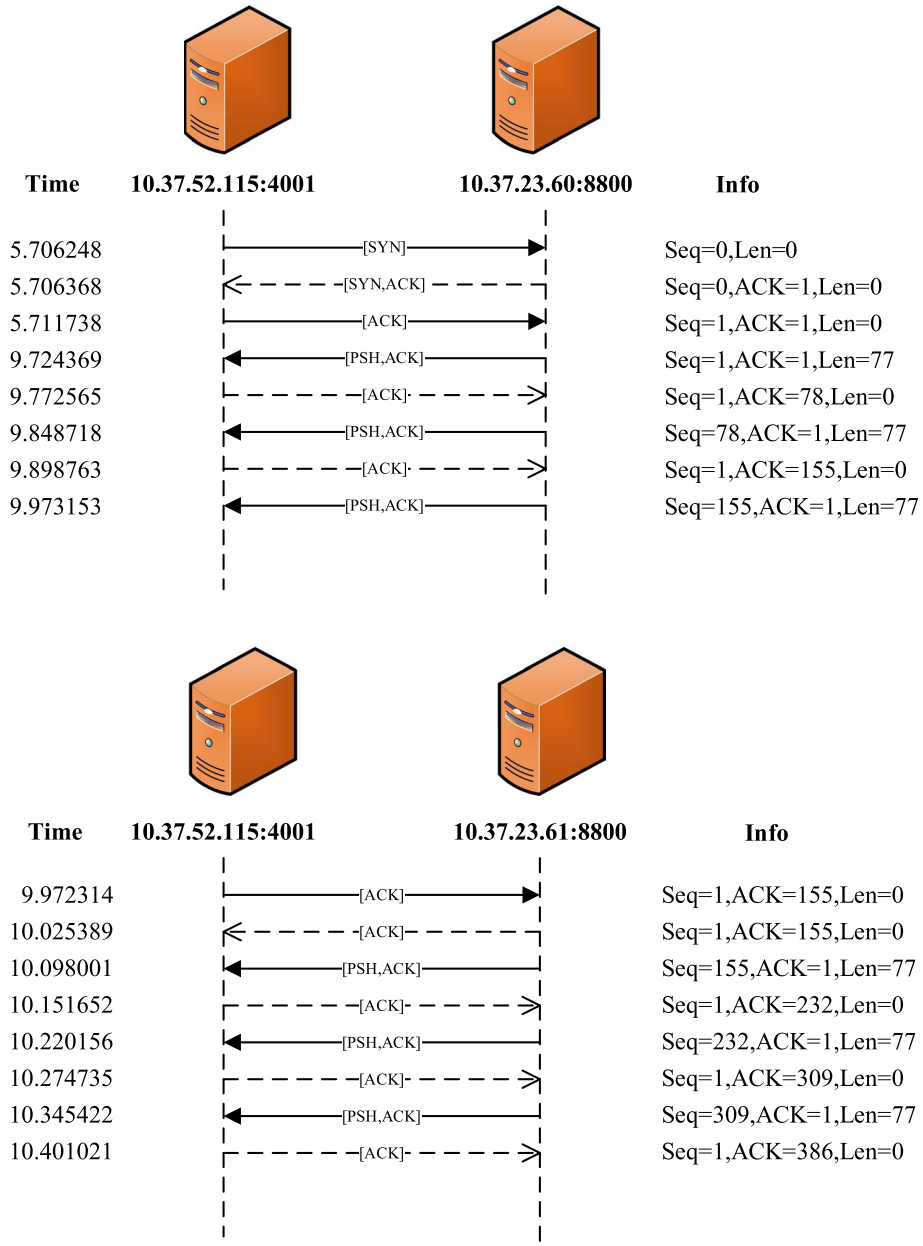

Fig. 8 Traffic exchange diagram before and after TCP_REPAIR switching

and non-encrypted traffic for the experiment, which matches the real working scenario. Details of the comparison experiments are shown in Table 2.

The accuracy of the three experimental sessions with streams is shown in Fig. 9.

As can be seen from the figure, if only identifying whether the encrypted or non-encrypted traffic is malicious traffic, the accuracy of its recognition Session is nearly 1.8% higher than Flow, However, if the classification is divided into Experiment 2 and Experiment 3 that are accurate to each type, the recognition accuracy will be greatly reduced. Fortunately, this does not have much impact on this system, because this system only needs to carry out whether the encrypted traffic has malicious components The accuracy of 1D-CNN recognition meets the experimental requirements.

In order to further verify the fine-grained classification effect of the model, the exact experimental precision and recall of Experiment 3 are shown in Tables 3 and 4.

Table 2 1D-CNN comparison experiment table

Experiment	Content	Category	Traffic Type	Whether Encryption	Dataset Size
1	Malicious encrypted traffic identification	2-class	Session	No	26,921
				Yes	12,525
			Flow	No	32,422
				Yes	17,926
2	Regular encrypted traffic classification	6-class	Session	No	26,921
				Yes	12,525
			Flow	No	32,422
				Yes	17,926
3	Encrypted traffic classification	12-class	Session	No	26,921
				Yes	12,525
			Flow	No	32,422
				Yes	17,926

The ISCX dataset classifies data into 12 categories, including 6 categories of regular encrypted traffic (VPN-Email, VPN-Chat, VPN-Streaming, VPN-File Transfer, VPN-VoIP, VPN-P2P) and 6 categories of protocol-encapsulated traffic (Email, Chat, Streaming, File Transfer, VoIP, P2P). In Experiment 1, there were 14,000 malicious traffic and 12,921 normal traffic in the unencrypted data set of Session traffic. There are 7000 malicious traffic and 5525 normal traffic in the encrypted data set. Flow Traffic There are 17,000 malicious traffic and 15,422 normal traffic in the non-encrypted data set. There were 9000 malicious traffic and 8926 normal traffic in the encrypted data set. The above malicious traffic is generated by randomly adding some malicious codes to the traffic data

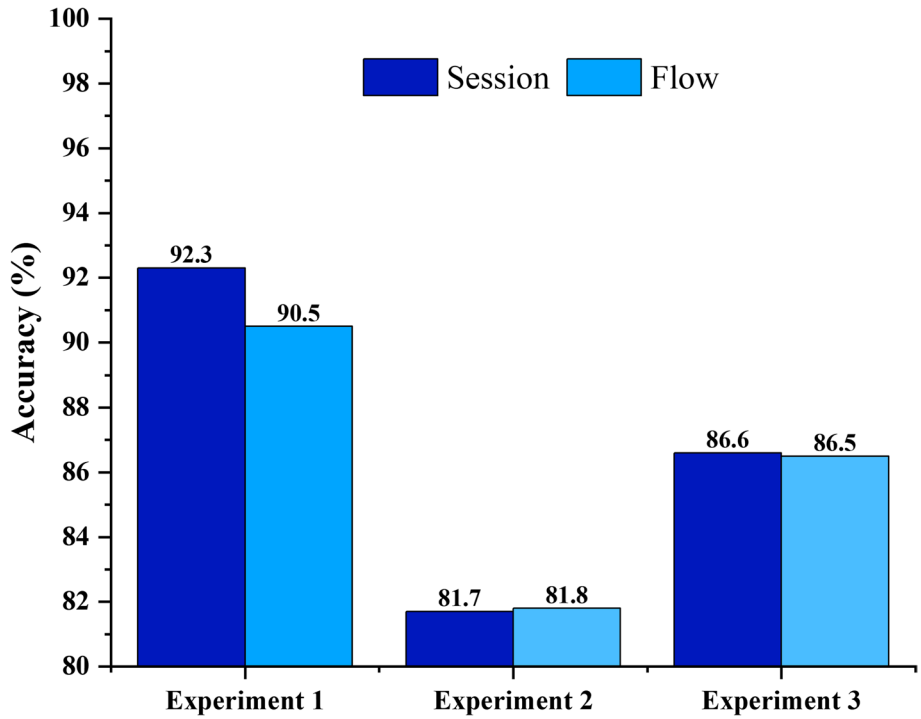

Fig. 9 Accuracy of three experimental sessions with streams

It can be seen from the above table that 1D-CNN has strong feature extraction ability in fine-grained classification of encrypted traffic. The Precision of all categories except VPN-Email was higher than 90%. The feature extraction capability of unencrypted traffic is lower than that of encrypted traffic, but the Precision of all traffic except Email, Chat, and FT is 90%. This shows that the model has strong feature recognition ability for both encrypted and unencrypted traffic.

Table 3 Accurate experimental precision and recall for the non-encrypted category of Experiment 3

Indicator	Classification					
	Email	Chat	Stream	FT	VoIP	P2P
Precision	69.5	71.3	95.7	82.6	95.3	98.2
Recall	72.8	69.8	95.6	94.8	83.9	98.5

Table 4 Accurate experimental precision and recall for the encryption category of Experiment 3

Indicator	Classification					
	VPN-Email	VPN-Chat	VPN-Stream	VPN-FT	VPN-VoIP	VPN-P2P
Precision	79.7	97.3	91.2	92.7	99.8	92.1
Recall	99.7	96.3	85.6	91.3	98.7	97.6

Table 5 Comparison of precision of encrypted and non-encrypted traffic identification for C4.5 and 1D-CNN

Experiment	Classification			
	C4.5(Non-VPN)	1D-CNN(Non-VPN)	C4.5(VPN)	1D-CNN(VPN)
1	90.6	99.7	89	98.6
2	89	85.5	84	94.9
3	84.3	85.8	78.2	92

Table 6 Comparison of recall of encrypted and non-encrypted traffic identification for C4.5 and 1D-CNN

Experiment	Classification			
	C4.5(Non-VPN)	1D-CNN(Non-VPN)	C4.5(VPN)	1D-CNN(VPN)
1	88.8	98.8	92	98.7
2	85.5	85.8	87.6	97.3
3	79.3	85.9	81.3	95.2

In order to verify the difference between the performance of 1D-CNN and the more popular current encrypted traffic identification models, the current more advanced traffic classification model C4.5 [34] was therefore compared. The performance of Precision and Recall of the two models was compared in three experimental settings using the ISCX dataset for encrypted and non-encrypted traffic, respectively, as shown in Tables 5 and 6.

As can be seen from the table, classification accuracy of the 1D-CNN model is far superior to the traffic recognition and classification algorithm C4.5 based on machine learning in three experiments. The main reason is that the feature extraction ability of deep learning is far higher than that of machine learning algorithm, which can adapt to various classification environments. Furthermore, it can quickly discriminate whether the flow is malicious traffic or not for the flow except for the training phase which is losser in computational performance when the model training is completed.

Overall system performance testing

In order to evaluate the traffic conversion capability and system stress tolerance of the system, therefore multiple experiments were designed to make a comprehensive comparison with HoneyBrid, HoneyDOC and Honeyd methods. The experiment of comparison mainly included latency, identification traffic accuracy and throughput. Although the experiments considered other parameters such as CPU load and memory occupancy, these parameters had a low impact on the experimental results due to the same experimental environment. In short these metrics are not particularly emphasized in the quantitative evaluation below. The benchmark test is represented as the performance of regular communication using ordinary TCP sockets without the redirection feature.

The first is delayed evaluation, the client and server built by the test system can effectively measure the round trip time (RTT). Because the sending and receiving paths are symmetrical, the delay uses half of the RTT. The attackers use TCP sockets to send the specified timestamp to the server, and the attacker calculates RTT after the server responds. The calculation method is half of the difference between echo time and the corresponding system clock when the message is sent. The test experiment is repeated 10,000 times. The experimental results are chosen to be in the 96% confidence interval of the value, and the comparative experimental results are shown in Fig. 10 (round-trip time comparison graph for connection redirection).

The experimental results shown above show that the latency of normal TCP socket communication is

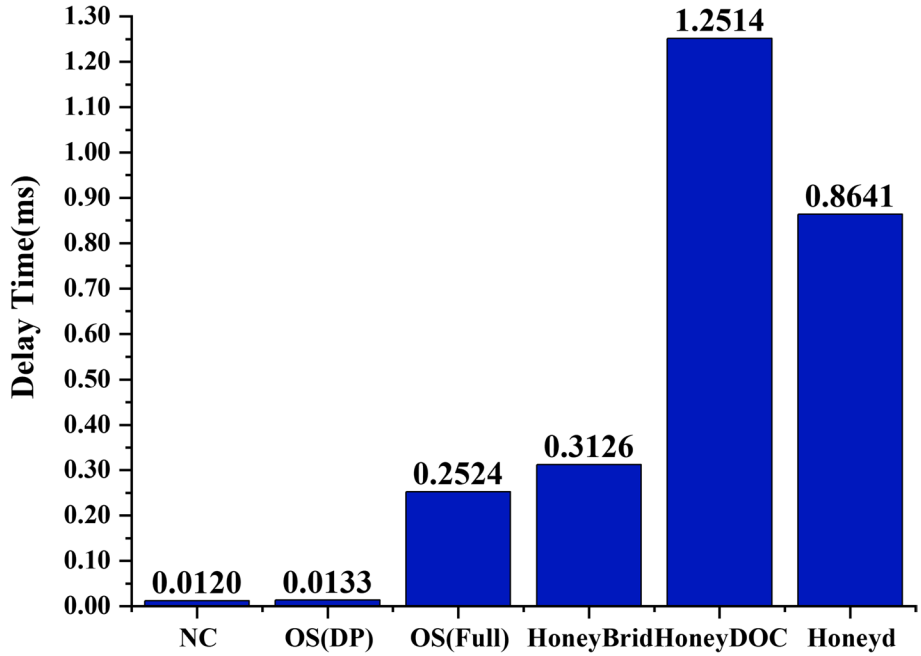

Fig. 10 System latency comparison

0.012049 ms, while the latency of our system excluding the 1D-CNN module is 0.013268 ms which is close to the latency of normal TCP socket communication. Because of the hot migration technique of TCP_REPAIR in the Linux kernel for connection transfer, which does not require re Instead of performing three handshakes, the original TCP connection parameters can be negotiated by sending an ACK packet with the confirmed maximum sequence number, thus redirecting the TCP connection. The latency of the complete system with the 1D-CNN module is 0.252356 ms, which is mainly due to the complexity of identifying the traffic as malicious traffic in the 1D-CNN, but still has an advantage over other mainstream TCP switching models. Due to the use of NAT address translation, the connection switch is achieved by replaying the message after modifying TCP parameters, so the intermediate processing needs to go through TCP three handshakes again. Thus, the latency is greatly increased compared to this system. HoneyDOC uses SDN controller to perform the worst traffic conversion, with an average time close to 1.251356 ms. This is mainly because HoneyDOC first needs IDS to send an alarm, and then replays TCP messages through the redirection module in the SDN controller's northbound interface to negotiate TCP parameters, so as to achieve connection switching. Since most of the switching processes are implemented by SDN controller and software modules, the time delay will be greatly increased compared with the first two methods. The delay of Honeyd method is 0.864102 ms. The main reason is that its method of switching traffic to the highly interactive honeypot mainly uses the built-in agent function of Honeyd software. The switching process is still operated by user software, so the delay is greatly increased.

In order to further test the latency of the system, a simple SMTP test is designed to test the delay from the first payload to the back-end honeypot in the concurrent state, The SMTP server is installed in the honeypot and the SMTP client script was sent by simulating an attacker. The content of the script is shown in Content 4. This test is run automatically on the normal system, this system, HoneyBrid, HoneyDOC, Honeyd system, and the script was run automatically at a rate of 10 connections per second, of which 1 s was randomly selected to test the delay between the first payload packet of the 10 connections and the arrival of the honeypot The details are shown in the table (Table 7).

Content 4: SMTP Client Script

```
HELO  test \n
MAIL FROM:<test@test.test.test> \n
PCRT TO:<root@localhost> \n
DATA \n
Test \n
```

Next, the system's ability to identify malicious traffic is tested. We designed 1000 of the four forms of SMTP

scripts to run on a single attack host to complete the test. The script contents are 250 non-encrypted normal scripts, 250 non-encrypted malicious scripts, 250 encrypted normal scripts, and 250 encrypted malicious scripts. The Content of each script is approximately equal with Content 3 above. After the script runs properly, the non-encrypted SMTP connects to port 25, and the encrypted SMTP connects to port 465. Compared with normal traffic, malicious traffic only adds malicious code in the DATA part. Recognition accuracy is represented by the ratio of the sum of connections correctly transferred to the real system and the honeypot of high interaction to all connections. The results of the comparative experiments are shown in the Fig. 11.

As shown in the figure above, the precision of HoneyBrid model and Honeyd model in identifying malicious traffic (both encrypted and unencrypted) is 76.3% and 78.4%, respectively. The main reason is that the traffic filtering of the two models is completed by the original intrusion detection system. The traditional intrusion detection system mainly detects whether the traffic has malicious components by matching the keywords, which is not suitable for encrypting the malicious traffic. The precision of HoneyDOC and the system without 1D-CNN module to identify malicious traffic is 78.8% and 79.1%, respectively. Compared with HoneyBrid model and Honeyd model, there is a partial improvement, but the improvement is not significant. The main reason is that although the two models add more traffic rules to the original intrusion detection system to identify malicious traffic, they can only identify non-encrypted malicious traffic to a greater extent and is not still effective on encrypted malicious traffic. In the paper, the accuracy of the traffic direction of the system with full module form is 92.6%, which is greatly improved compared with the traditional intrusion detection system. The main reason is the powerful feature extraction ability of 1D-CNN, which can effectively identify malicious components in encrypted traffic, and greatly improve the traffic identification ability of the system. Finally, the throughput is tested by transferring large files that have been prepared by the client, and the client calculates the time it takes for the system to transfer the large files. The system throughput is the file size divided by the transfer time. To reduce the error, all tests were repeated 200 times with a confidence interval of 96%. The results of the comparison experiments are shown in Fig. 12.

The throughput of Normal Communication shown above is 744.5Mbit/s. The throughput of the system with 1D-CNN removed is 711.6Mbit/s, with a loss rate of 4.42%. The principal reason is that its TCP connection switching mainly depends on the conversion of the Linux kernel. Therefore, the loss is mainly caused by the software level such as IDS and SDN controller. The throughput of the full version of the system is 583.7Mbit/s, with a loss of 21.59%.Largely because the 1D-CNN in the traffic detection takes up channel resources, so the throughput is reduced. 50.8Mbit/s of HoneyBrid's throughput, with a loss of 93.18%, is chiefly due to the replay of TCP packets in the NAT address translation process The throughput of HoneyDOC is 18.2Mbit/s, with a loss of 97.55%, mainly because the system is mainly realized by TCP packet replay through the TCP switch module in the northbound interface of the SDN controller, and its front IDS also leads to a significant reduction in throughput. The throughput of Honeyd is 45.7 Mbit/s, and the loss is 93.86%. Although the throughput of Honeyd is higher than HoneyDOC, its overall performance is not as good as HoneyDOC. Primarily because Honeyd does not have traffic filtering, so it is easily flooded by invalid traffic.

Table 7 Table of latency information for 10 connections in 1 s

10 Connection Serial Numbers for 1 Second	Normal Communication(s)	Our System Dropout 1D-CNN(s)	HoneyBrid(s)	HoneyDOC(s)	Honeyd(s)
1	0.002	0.009	0.208	0.049	0.225
2	0.008	0.009	0.209	0.209	0.225
3	0.008	0.013	0.209	0.225	0.229
4	0.008	0.017	0.209	0.250	0.241
5	0.010	0.017	0.209	0.324	0.263
6	0.012	0.017	0.209	0.325	0.274
7	0.014	0.019	0.209	0.325	0.291
8	0.015	0.023	0.209	0.350	0.325
9	0.017	0.023	0.223	0.368	0.371
10	0.017	0.027	0.231	0.400	0.384

The experimental results again demonstrate that the TCP_REPAIR-based connection switching latency in this system is much lower than using TCP replay and NAT address translation

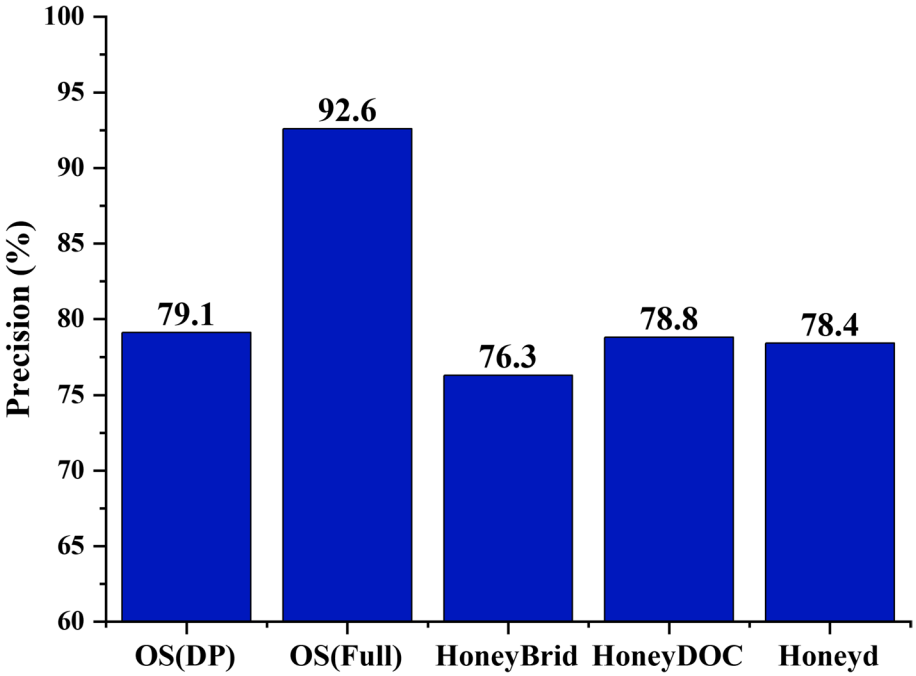

Fig. 11 Comparison of system identification traffic precision

Experimental results show that the system performs well for all types of network requirements and is more stealthy compared to software replay of TCP packets due to its conversion using Linux kernel features.

Concluding remarks

Due to the rise of encrypted communication, it brings security to communication on the one hand, but also creates a great security problem for the interception and capture of attacks. At the same time, with the continuous

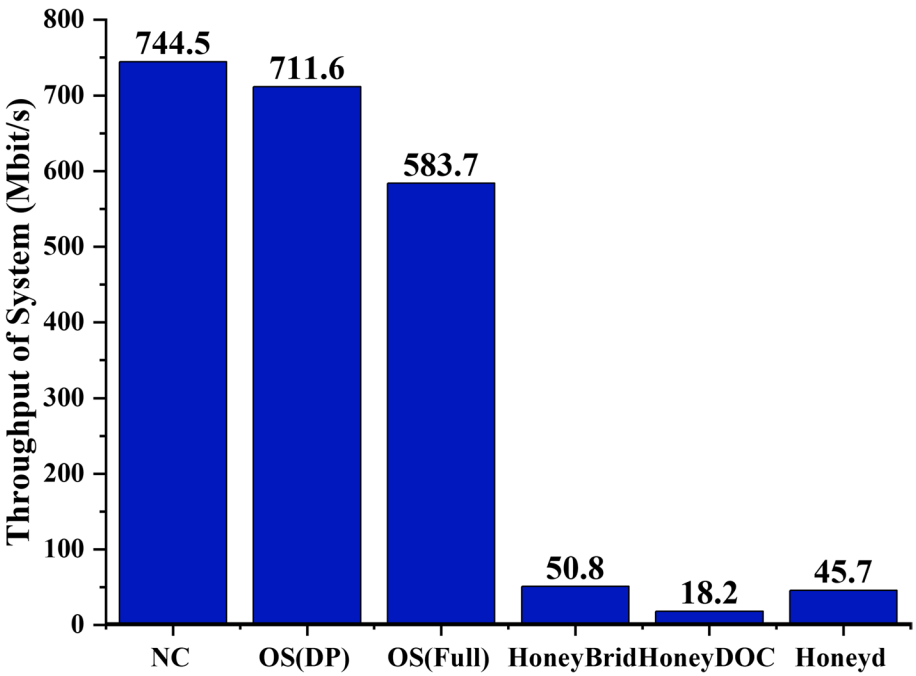

Fig. 12 System Throughput Comparison

updating of network attacks, it also poses a great challenge to the security of real systems. Therefore honeypots that can triage and actively capture attack process analysis based on identifying attack traffic have great development value. As an active defense tool, honeypots are also evolving according to the current development of network attacks. Although the traditional honeypot can focus on a specific attack, it has little effect when encountering other attacks. Thus, the invention of the hybrid honeypot solves this problem. The biggest problem of hybrid honeypots is the switching algorithm between low interaction honeypots and high interaction honeypots. Although traditional NAT address translation can also achieve this effect, it has defects both in speed and security.

In this paper, we propose an intelligent defense system based on TCP_REPAIR and deep learning. Traffic classification and identification by traditional intrusion detection system and deep learning-based traffic analysis model can accurately verify the security of ordinary traffic and encrypted traffic. Abandoning the use of traditional NAT address translation for hybrid honeypot connection transfer, connection transfer is performed using the TCP_REPAIR socket design agent in the Linux kernel, allowing for fast and seamless transfer to another endpoint at any stage of an active connection to continue the current session. And the performance loss of our solution is mainly in the process of creating sockets, using OpenFlow 1.5 to match TCP keywords and intelligent firewalls to identify malicious traffic, with almost identical performance to that of regular sockets. Thanks to TCP_REPAIR's reliance on the underlying Linux communication results.

The usability and efficiency of the method are confirmed by comparing the experimental data results, while the method can remain invisible during active TCP connection switching. A part of the efficiency problem is detected to arise mainly in the detection of 1D-CNN, principally because the virtual scenes currently tested are relatively simple. The future research direction is mainly the detection speed improvement of 1D-CNN, which makes the performance of the whole system adaptable to large-scale use.

Abbreviations

IDS	Intrusion Detection System
IPS	Intrusion Prevention System
HIH	High-Interaction Honeypot
LIH	Low-Interaction Honeypot
SDN	Software Defined Network
HTTPS	Hyper Text Transfer Protocol Secure
RF	Random Forest
MBSGD	Mini-Batch Stochastic Gradient Descent
NIDS	Network Intrusion Detection System
MSS	Maximum Segment Size
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
DPI	Deep Packet Inspection
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Network
AE	AutoEncoder

Authors' contributions

Jianxun Tang: Conceptualization (lead); formal analysis (lead); investigation (lead); methodology (lead); software (lead); writing-original draft preparation (lead). Haoyu Chen: Software (equal); data curation (lead); writing-original draft (equal). Shenqi Zhao: Software (equal); data curation (equal); writing-original draft (equal). Yu Huang: Software (equal); validation (lead). Mingsong Chen: Conceptualization (supporting); funding acquisition (supporting); resources (supporting); supervision (lead); writing-review and editing (lead). The author(s) read and approved the final manuscript.

Funding

This study was supported in part by the National Natural Science Foundation of China (91836301), the Special Program of Guangxi Science and Technology Base and Talents under Grant (AD21220098) and the Innovation Project of Guangxi Graduate Education (YCSW2022289).

Availability of data and materials

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Competing interests

The authors report no conflict of interest.

Received: 25 August 2022 Accepted: 7 December 2022
Published online: 14 February 2023

References

1. Shi J, Chen M, Jiao J (2022) Thoughts on the application of low-interactive honeypot based on raspberry pi in public security actual combat, LIHRP. In: Sun X, Zhang X, Xia Z, Bertino E (eds) Artificial intelligence and security. ICAIS 2022. Lecture notes in computer science, vol 13340. Springer, Cham
2. Anwar AH, Leslie NO, Kamhoua CA (2021) Honeypot allocation for cyber deception in internet of battlefield things systems. In: MILCOM 2021–2021 IEEE military communications conference (MILCOM), pp 1005–1010
3. Tang J, Zhou F (2021) Design and implementation of high-performance web vulnerability scanner based on Python intelligent crawler. In: 2021 International Conference on Computer Information Science and Artificial Intelligence (CISAI), pp 765–769
4. Dainotti A, Pescape A, Claffy KC (2012) Issues and future directions in traffic classification. IEEE Netw 26(1):35–40
5. Sun GL, Xue Y, Dong Y et al (2010) A novel hybrid method for effectively classifying encrypted traffic. In: 2010 IEEE global telecommunications conference
6. Velan P, Čermák M, Čeleda P et al (2015) A survey of methods for encrypted traffic classification and analysis. Int J Netw Manag 25(5):355–374
7. Arndt DJ, Zincir-Heywood AN (2011) A comparison of three machine learning techniques for encrypted network traffic analysis. In: 2011 IEEE symposium on computational intelligence for security and defense applications (CISDA)
8. Yao Z, Ge J, Wu Y et al (2020) Encrypted traffic classification based on Gaussian mixture models and hidden Markov models. J Netw Comput Appl 166:102711
9. He K, Chen X, Xie S et al (2022) Masked autoencoders are scalable vision learners. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 16000–16009

10. Li R, Zheng M, Bai D, Chen Z (2021) SDN based intelligent Honeynet network model design and verification. In: 2021 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE), pp 59–64. https://doi.org/10.1109/MLISE54096.2021.00019
11. Wazirali R, Ahmad R, Alhiyari S (2021) SDN-openflow topology discovery: an overview of performance issues. Appl Sci 11(15):6999
12. Ray PP, Kumar N (2021) SDN/NFV architectures for edge-cloud oriented IoT: a systematic review. Comput Commun 169:129–153
13. Khorsandroo S, Sánchez AG, Tosun AS et al (2021) Hybrid SDN evolution: a comprehensive survey of the state-of-the-art. Comput Netw 192:107981
14. Vieira JL, Ferreira VC, Bastos IV et al (2021) THANOS: Teleprotection holistic application for ONOS controller. In: 2021 IFIP/IEEE international symposium on integrated network management (IM). IEEE, pp 818–823
15. Babbar H, Rani S (2021) "Performance evaluation of qos metrics in software defined networking using ryu controller." IOP conference series: materials science and engineering, vol 1022. No. 1. IOP Publishing. https://doi.org/10.1088/1757-899X/1022/1/012024
16. Parhandhito N, Negara RM, Dewanta F (2021) "Comparison of High Availability Performance on OpenDaylight with Corosync Pacemaker and OpenDaylight SDN Controller Platform Clustering," 2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS), pp 66–71. https://doi.org/10.1109/IoTaIS50849.2021.9359696
17. Yan B, Liu Q, Shen JL et al (2022) Flowlet-level multipath routing based on graph neural network in OpenFlow-based SDN. Futur Gener Comput Syst 134:140–153
18. Wang J, Yang H, Fan C (2021) A SDN dynamic honeypot with multi-phase attack response. Netinfo Security 21(1):27–40
19. Jiahui L, Yijun W, Zhi X (2021) TCP connection handover mechanism for hybrid Honeynet based on connection state and SDN. Commun Technol 54(2):444–450
20. Artail H, Safa H, Sraj M, Kuwatly I, Al-Masri Z (2006) A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks. Comput. 25(4):274–288
21. Fan W, Du Z, Smith-Creasey M, Fernandez D (2019) HoneyDOC: an efficient honeypot architecture enabling all-round design. In: IEEE journal on selected areas in communications
22. Berthier R, Cukier M (2008) Honeybrid: a hybrid honeypot architecture. In: USENIX security symposium, vol 2008
23. El-Maghraby RT, Abd Elazim NM, Bahaa-Eldin AM (2017) "A survey on deep packet inspection," 2017 12th International Conference on Computer Engineering and Systems (ICCES), pp 188–197. https://doi.org/10.1109/ICCES.2017.8275301
24. Soleimani MHM, Mansoorizadeh M, Nassiri M (2018) Real-time identification of three Tor pluggable transports using machine learning techniques. J Supercomput 74(10):4910–4927
25. Aceto G, Ciuonzo D, Montieri A et al (2019) Mobile encrypted traffic classification using deep learning:experimental evaluation, lessons learned, and challenges. IEEE Trans Netw Serv Manag 16(2):445–458
26. Yang L, Finamore A, Jun F et al (2021) Deep learning and zero-day traffic classification: lessons learned from a commercial-grade dataset. IEEE Trans Netw Serv Manag 18(4):4103–4118
27. Wang W, Zhu M, Wang J et al (2017) End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: 2017 IEEE international conference on intelligence and security informatics
28. Chen Z, He K, Li J et al (2017) Seq2img:a sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In: 2017 IEEE international conference on big data
29. Salman O, Elhajj IH, Kayssi A et al (2021) Data representation for CNN based internet traffic classification: a comparative study. Multimed Tools Appl 80(11):16951–16977
30. Sadeghzadeh AM, Shiravi S, Jalili R (2021) Adversarial network traffic: towards evaluating the robustness of deep-learning-based network traffic classification. IEEE Trans Netw Serv Manag 18(2):1962–1976
31. Shin HC, Roth HR, Gao M et al (2016) Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE Trans Med Imaging 35(5):1285–1298
32. Wang W, Zhu M, Wang J, Zeng X, Yang Z (2017) End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: 2017 IEEE international conference on intelligence and security informatics (ISI), pp 43–48. https://doi.org/10.1109/ISI.2017.8004872
33. Cunha VA, Corujo D, Barraca JP, Aguiar RL (2020) Using Linux TCP connection repair for mid-session endpoint handover: a security enhancement use-case. In: 2020 IEEE conference on network function virtualization and software defined networks (NFV-SDN), pp 174–180
34. Draper-Gil G, Lashkari AH, Mamun MSI, Ghorbani AA (2016) Characterization of encrypted and VPN traffic using time-related features. In: In proceedings of the 2nd international conference on information systems security and privacy (ICISSP), pp 407–414

Publisher's Note