2024 the 9th International Conference on Big Data Analytics

MySQL-Pot: A LLM-Based Honeypot for MySQL Threat Protection

Yuqi Hu
Institute of Advanced Technology,
University of Science and Technology of China
Hefei, Anhui
huyuqiokok@mail.ustc.edu.cn

Siyu Cheng
School of Information Science and Technology,
University of Science and Technology of China
Institute of Artificial intelligence, Hefei Comprehensive National Science Center
Hefei, Anhui
syccc@mail.ustc.edu.cn

Yuanyi Ma
School of Information Science and Technology,
University of Science and Technology of China
Hefei, Anhui
yym672@mail.ustc.edu.cn

Shuangwu Chen
Institute of Artificial intelligence, Hefei Comprehensive National Science Center
School of Information Science and Technology,
University of Science and Technology of China
Hefei, Anhui
chensw@ustc.edu.cn

Fengrui Xiao
School of Information Science and Technology,
University of Science and Technology of China
Hefei, Anhui
franxfr@mail.ustc.edu.cn

Quan Zheng*
School of Information Science and Technology,
University of Science and Technology of China
Institute of Artificial intelligence, Hefei Comprehensive National Science Center
Hefei, Anhui
qzheng@ustc.edu.cn

2024 9th International Conference on Big Data Analytics (ICBDA) | 979-8-3503-5296-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICBDA61153.2024.10607309

***Abstract*—Traditional network defense techniques such as firewalls and intrusion detection systems [1] primarily involve passive defense, which often struggles to effectively counter the omnipresent and ever-evolving threat landscape. The advent of honeypot technology has introduced a paradigm shift in network defense, enabling a more proactive approach by attracting and deceiving attackers. Honeypots facilitate the study of attackers' motives and techniques, ultimately delaying or preventing destructive attacks and safeguarding real service resources. In this paper, we introduce a MySQL protocol simulation honeypot, which represents an intelligent interactive honeypot system capable of generating responses using a LLM (Large Language Model). With the help of LLM language understanding capabilities, the honeypot learns the behavior of SQL (Structured Query Language) requests and continues to engage with attackers, which provides optimal simulated responses to attack requests instead of erroneous ones. We conducted comparative experiments with an open-source MySQL database honeypot, and the results indicate that our interactive approach improves session length and response speed compared to existing interaction methods.**

Keywords—Honeypot, LLM, MySQL

I. Introduction

Since the birth of databases, they have been constantly under the threat of network attacks. With the continuous development of attack technologies, new forms of security threats emerge and evolve, while defense technologies struggle to keep pace with the changing landscape of security threats. This has led to a worsening security situation on the Internet. Verizon's Data Breach Investigations Report (DBIR) [2] analyzed a large number of data breach incidents and records, revealing that 92% of data breaches originated from databases (database services being vulnerable). Because of this, a deception based approach called a 'honeypot' is being used by network administrators to prevent database intrusion. A honeypot is an active defense technology proposed by the defending party to counteract the asymmetry in the cybersecurity landscape. It is defined as a type of security resource with no operational purpose, and its value lies in attracting malicious usage by adversaries [3]. Honeypot technology is essentially a deceptive technique against attackers. By deploying certain hosts, network services, or information as bait, it entices attackers to launch attacks, allowing for the capture and analysis of their activities. This enables an understanding of the tools and methods employed by attackers, as well as speculation about their intentions and motives.

Early honeypots were typically disguised as network services with vulnerabilities, responding to connection attempts from attackers. Lopez *et al.* [4] built a deployable virtual honeypot using common honeypot frameworks to simulate IoT attacks. Tabari *et al.* [5] crafted their first low-interaction honeypot, emulating an IP-enabled camera called "Honeycam". However, these virtual honeypots faced issues such as low interaction and being easily identified by attackers. To address these drawbacks, researchers have utilized real system environments to construct honeypots. Conti M *et al.* [6] introduced a high-interaction honeypot for industrial control systems, and Guan *et al.* [7] proposed an Adaptive High-Interaction Honeypot for IoT Devices through Reinforcement Learning. Nevertheless, using real systems comes with higher maintenance costs and poses certain risks to the protected resources, which raises security concerns.

In order to address the issues of insufficient interaction in early virtual honeypots and the security shortcomings of honeypots using real systems, this paper proposes a virtual interactive honeypot specifically designed for databases. This honeypot involves simulating the commonly used MySQL network protocol in database systems. When attackers attempt to interact with the database, it utilizes historical database interaction traffic to return fake fingerprint response packets. This approach resolves the security concerns by using historical traffic that does not involve real protected resources within the database. The MySQL protocol simulation employs large-scale language modeling techniques, enabling real-time simulation of MySQL request-response flows. This allows for the provision of efficient customized responses for each interaction initiated by the attacker, thereby enhancing the honeypot system's deceptive capabilities and addressing the issue of insufficient interaction in honeypots.

The contributions of this article are as follows:

(1). We propose a method to enhance the interactivity of honeypot systems using LLM. With this approach, even in the case of unknown requests, we can collect more attack interaction data, facilitating defensive decision-making.

(2). We propose a method that utilizes historical traffic as a foundation, extracting valid payloads (database name, table name, column name, row data) from attacker commands and incorporating them into historical traffic. This aims to construct fake data packets, thereby enhancing the security of the honeypot.

(3). A honeypot system for database services is implemented and interacts with the real database service of the client, and the experiment shows that the system can respond to client requests in real time.

The rest of the paper is structured as follows: Chapter 2 presents related works on MySQL honeypots. Chapter 3 introduces the honeypot system based on language model design proposed in this paper. Chapter 4 compares the tools we designed with other open-source honeypot systems, and the fifth chapter summarizes and looks ahead.

II. Related Work

Low-interaction honeypots use a flexible software framework that supports the creation of low-interaction simulation honeypots within their framework protocol. These honeypots are highly scalable and can simulate services by developing corresponding service modules that can be added to the honeypot. This type of honeypot includes systems like Honeyd [8], Deception Toolkit (DTK) [9], and Nepenthes [10] [11]. Honeyd is a low-interaction honeypot designed for Unix platforms used for attack detection. Honeyd directly uses the Nmap fingerprint database as the basis for generating operating system fingerprints, allowing it to simulate a wide variety of operating systems. Users can configure the operating system in the configuration file, where each template corresponds to a specific operating system, and each system can have multiple implementation details, including open ports, port behaviors, and more. Deception Toolkit (DTK) is a low-interaction honeypot tool for Unix platforms. DTK uses Perl scripts to simulate real system services and, by manipulating Perl scripts, can easily simulate various system service vulnerabilities. Nepenthes inherits Honeyd's network protocol stack simulation mechanism and framework structure. However, unlike previous honeypot systems that attempt to simulate the entire network service interaction process, Nepenthes is designed to only simulate the vulnerable parts of network services. It uses "Shellcode heuristic recognition" and "emulation execution technology" to discover penetration attacks targeting network service security vulnerabilities, extract download links for actively spreading malicious code, and further capture samples. The advantages of this type of honeypot are strong scalability, relatively simple service module development, easy deployment and less risk, but because it is a low-interaction honeypot, the use of simulation services, so there must be low interaction, limited capture of attack data, and easy to be identified.

The use of high-interaction honeypots in real systems can provide complete interaction functionality. For example, Cenys *et al*. [12] and others have designed a high-interaction database honeypot system based on honeytoken technology for Oracle. Guarnizo *et al*. [13] have proposed a scalable and high-interaction honeypot platform for IoT devices to study attack vectors. Sochor *et al*. [14] use a web server honeypot to respond to requests on port 80 and monitor using vulnerable web systems. The advantage of such honeypots is that they use real hardware and software systems as their operational environment, significantly increasing the level of interaction in attacks. Using real hardware and software devices to lure attackers has certain advantages in low-energy scenarios. However, this type of honeypot has a higher risk of being hijacked by attackers in order to launch further attacks [15].

III. System Design

To combine the advantages of low-interaction honeypots, which require minimal resource usage and simple deployment, with the advantages of high-interaction honeypots, which have high deception levels and strong interaction capabilities. We propose a system that minimizes the respective drawbacks of two types of honeypots and leverages artificial intelligence technology to enhance the interaction performance of the honeypot system with unknown requests to a certain extent.

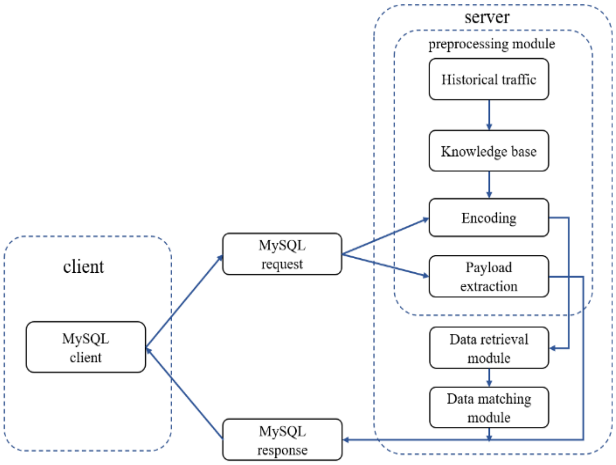


Fig. 1. System design

As shown in Fig. 1, our primary task is to handle MySQL requests sent by clients with a real MySQL database. We achieve this by simulating a server that sends crafted fake response data, allowing communication with the actual database.

To achieve the required functionality, this paper focuses on the design of three main modules on the server side, namely the preprocessing module, data retrieval module, and data matching module. The preprocessing module primarily extracts historical traffic from the database to form a knowledge base comprising request-response pairs. It targets the extraction of payload (database name, table name, column name, row data) from MySQL requests inputted by attackers. The attackers' SQL sentences are converted into word vectors, which are then passed to the retrieval module. The data retrieval module is mainly responsible for retrieving similar sentences from the knowledge base based on the SQL sentences passed from the preprocessing module. The data matching module is designed to filter and sort SQL requests processed by the retrieval module using LLM. It selects the response corresponding to the top-ranked request and combines it with the extracted payload to generate the final response data.

A. Preprocessing Module

The historical access traffic from the database is extracted to create request-response pairs, forming a knowledge base denoted as $\boldsymbol{K}$, with the request-response pairs within this knowledge base represented as $\boldsymbol{k_n}=(\boldsymbol{k_n^{rq}},\boldsymbol{k_n^{rs}})$, $n \in \{0,1,\ldots,|\boldsymbol{K}|-1\}$. Additionally, a corpus $\boldsymbol{V}=\{v_0,v_1,\ldots,v_m\}$, $m=\{0,1,\ldots,|\boldsymbol{V}|-1\}$ is constructed from all the words present in the database. To ensure the authenticity of response data in the interactions between the client and the server, it is imperative to accurately acquire or generate the payload within the response packet.

In the preprocessing module, to extract the payload from the received string based on the MySQL protocol, we need to tokenize the input sentence according to the information for each field in the MySQL protocol. We define the input sentence as $\boldsymbol{W}=\{w_0,w_1,..,w_{n-1}\}$, where w represents the word after the participle and n represents the number of words in the input sentence. The parser scans the input sentence character by character from left to right, splitting it into tokens based on regular expressions as specified in TABLE I. Each token is assigned a type and value, for example, [{type: DML, value: select}, {type: Whitespace, value: }, {type: Identifier, value: columns}, ...]. Each word is associated with its corresponding type. Each tokens represent different parts of the SQL query, such as the select clause, from clause, where clause, etc. We extract column names from the payload based on DML types like 'select' and Keyword type like 'as.' Table names are extracted from the payload using Keyword type 'from' and Identifier type. Additionally, we extract column names and their corresponding row data using Assignment type and Identifier type. we use $\boldsymbol{P}^{db}$, $\boldsymbol{P}^{tb}$, $\boldsymbol{P}^{cl}$, $\boldsymbol{P}^{rd}$ to represent database names, table names, column names and row data in the payload.

Before entering the retrieval module, the words in the sentence W need to be encoded. With a specific corpus $\boldsymbol{V}$, for each word after word segmentation, the word has a corresponding position in the corpus and we take this position information as the encoding information of the word, represented as $x_i \in \{0,..,|\boldsymbol{V}|-1\}$, $i \in \{0,1,\ldots,n-1\}$.

TABLE I. REGULAR EXTRACTION

<table>
<tr><th>Types</th><th>Examples</th><th>Example of Regular Expressions</th></tr>
<tr><td>Comment</td><td>–</td><td>–|)#\+.*?(\r\n|\r|\n|$</td></tr>
<tr><td>Assignment</td><td>var:=val</td><td>:=</td></tr>
<tr><td>Operator</td><td>+</td><td>[+/@#%ˆ&|ˆ-]+</td></tr>
<tr><td>Comparison</td><td>=, ></td><td>[<>=~!]+</td></tr>
<tr><td>Text</td><td>\r\n</td><td>“.*?”</td></tr>
<tr><td>Whitespace</td><td></td><td>\s</td></tr>
<tr><td>Newline</td><td>\r</td><td>\r\n|\r|\n</td></tr>
<tr><td>Punctuation</td><td>; : ( ) [ ] , .</td><td>[;:()\ [\ ],\ . . .]</td></tr>
<tr><td>Keyword</td><td>from, GROUP BY</td><td>CASE|IN|V ALUES|USING|FROM|GROUP\s+BY</td></tr>
<tr><td>DDL</td><td>CREATE, ALTER</td><td>CREATE(\s+OR\s+REPLACE)?\b</td></tr>
<tr><td>DML</td><td>SELECT, UPDATE</td><td>SELECT|INSERT|UPDATE|DELETE</td></tr>
<tr><td>Name</td><td>c</td><td>(?<![\w\])])(\ [[ˆ\ ]\ []+\ ])</td></tr>
<tr><td>Placeholder</td><td>?, *</td><td>?, *</td></tr>
<tr><td>Identifier</td><td>database, table, column</td><td>\w</td></tr>
<tr><td>Wildcard</td><td>*</td><td>*</td></tr>
</table>

The real-time size of the corpus is $|\boldsymbol{V}|$. For word in the input sentence that is not present in the original corpus, its position in the corpus is set to $|\boldsymbol{V}|$, and it is added to the position in the corpus. Subsequently, the sentence encoding $\boldsymbol{X}=\{x_0,x_1,\ldots,x_{n-1}\}$ is inputted into the retrieval module. We use $\boldsymbol{X_n^{rq}}$, $n \in \{0,1,\ldots,|\boldsymbol{K}|-1\}$ to represent the encoding request in the knowledge base. The response corresponding to the request is encoded as $\boldsymbol{X_n^{rs}}$, $n \in \{0,1,\ldots,|\boldsymbol{K}|-1\}$.

Given that the sentence's order contains some elements of the semantic relationship, it is imperative to incorporate positional information into the input content to enhance the understanding of the sentence's semantics. We use the position of words in the sentence like $\{0,1,\ldots,n-1\}$ as the positional encoding.

B. Data Retrieval Module

Given the vast amount of data in the knowledge base, we adopt a combined approach using the text-based BM25 [16] algorithm and the semantics-based Word2Vec [17] to swiftly and accurately filter out similar requests. The use of the BM25 algorithm effectively captures text features, enhancing our search efficiency. However, it has limitations in understanding semantic relationships, especially in the case of short texts, resulting in lower recognition rates. Combining Word2Vec with the BM25 algorithm effectively addresses these limitations. This approach allows us to quickly filter out the top M most similar requests from the knowledge base.

For the input request $\boldsymbol{X}$, in the preprocessing stage, the sentence encoding $\boldsymbol{X}$ are fed into the retrieval module. The retrieval module scans all the knowledge base request messages, identifying those that contain the word vectors. It then proceeds to calculate the number of occurrences of each word vector in the request messages. However, due to the large volume of data in the dataset, such a traversal-based retrieval structure is inefficient and cannot meet the requirements. Therefore, this paper adopts the BM25 model based on the TF-IDF algorithm [14]. The BM25 formula is primarily composed of three components: the relevance between word x_i, $i \in \{0,1,\ldots,n-1\}$ and the set of all requests in the knowledge base request encoding, the similarity between the word x_i and request $\boldsymbol{X}^{rq}_n$, $n \in \{0,1,\ldots,|\boldsymbol{K}|-1\}$ and the weight of each word in $\boldsymbol{X}$. By calculating the product of the three combinations mentioned above, we can retrieve request sentences represented as $\boldsymbol{X}^{rq}_n$, $n \in \{rank0,rank1,..,rankY-1\}$ that are akin to user requests from the knowledge base request encoding.

For the input sentence encoding $\boldsymbol{X}$, after training the Word2vec model, we obtain the word vector corresponding to each word. These word vectors are then summed together, resulting in the creation of the input sentence vector, denoted as $\boldsymbol{A_X}$. Finally, we obtain Y request sentences, each generating a sentence vector $\boldsymbol{B}_{\boldsymbol{X}^{rq}_n}$, $n \in \{rank0,rank1,..,rank(Y-1)\}$. We employ the cosine similarity formula (1) to calculate the similarity values between the input statement vector $\boldsymbol{A_X}$ and $\boldsymbol{B}_{\boldsymbol{X}^{rq}_n}$.

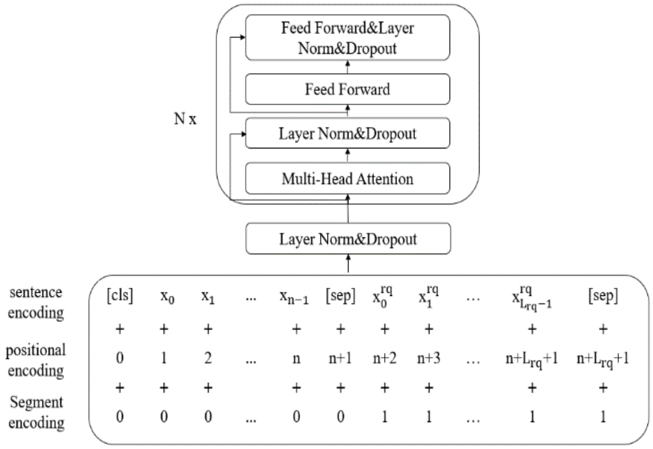
$$Similarity = cos(\theta) = \frac{A_X B_{X^{rq}_n}}{||A_X|| \, ||B_{X^{rq}_n}||} \tag{1}$$

The M ($M<Y$) highest similarity values are selected, forming the candidate set of the similar requests, denoted as $\boldsymbol{X}^{rq}_n$, $n \in \{rank0,rank1,..,rank(M-1)\}$.

C. Data Matching Module

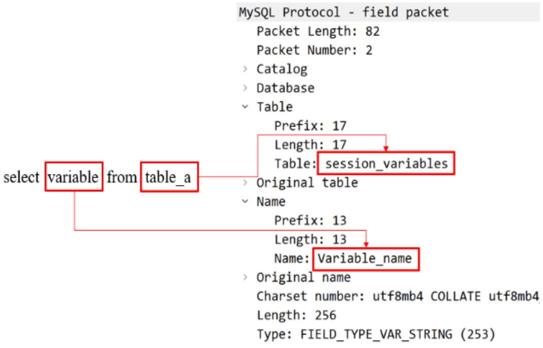
For request packets that do not appear in the knowledge base, the matching module utilizes an enhanced Bert model to analyze the user's input request. It calculates the standard request packet that closely resembles the attacker's request packet, sorts the results, and selects the most similar standard request packet. This selected standard request packet is then used to retrieve the associated response packet, which can be modified as necessary.

The input to the Bert model is a pair of sentences ($\boldsymbol{X}$ and $\boldsymbol{X}^{rq}_n$, $n \in \{rank0,rank1,..,rank(M-1)\}$) that can be represented in a sequence. We use L_{rq} to represent the number of words in $\boldsymbol{X}^{rq}_n$. As shown in Fig. 2, we input segment encoding, sentence encoding, and positional encoding into the model. At the same time, we scale down the original model to reduce the number of neurons, resulting in a significant reduction in the amount of machine operation.



sentence encoding	[cls]	x_0	x_1	...	x_{n-1}	[sep]	x^{rq}_0	x^{rq}_1	...	$x^{rq}_{L_{rq}-1}$	[sep]
	+	+	+		+	+	+	+		+	+
positional encoding	0	1	2	...	n	n+1	n+2	n+3	...	$n+L_{rq}+1$	$n+L_{rq}+1$
	+	+	+		+	+	+	+		+	+
Segment encoding	0	0	0	...	0	0	1	1	...	1	1

Fig. 2. Model structure.



```
MySQL Protocol - field packet
  Packet Length: 82
  Packet Number: 2
› Catalog
› Database
⌄ Table
    Prefix: 17
    Length: 17
    Table: session_variables
› Original table
⌄ Name
    Prefix: 13
    Length: 13
    Name: Variable_name
› Original name
  Charset number: utf8mb4 COLLATE utf8mb4
  Length: 256
  Type: FIELD_TYPE_VAR_STRING (253)
```

Fig. 3. Response generation

In this paper, for the task of text similarity matching, we use binary cross-entropy loss to measure the difference between the model's predicted labels and the actual labels. Assuming the model's predicted score is denoted as P (a scalar), and the true label is denoted as Q (0 or 1, representing non-match or match). The specific loss function is as follows:

$$Loss = -[Q * log(P) + (1 - Q) * log(1 - P)] \tag{2}$$

For received SQL statement packets, even in the case of unknown requests, the language model can find the most similar request message $\boldsymbol{X}^{rq}_{best}$. However, this response message may not be identical to the payload sent by the user's message.

For example, when the user request message is "select variable from table_a", the model returns the most similar standard request as "select Variable_name from session_variables". In this case, the corresponding response column in the response dataset is "Variable_name", which does not match the user's desired "variable" column data. Therefore, as shown in Fig. 3, it is necessary to construct the response message for this request, based on the payload parsed from the SQL statement and the knowledge base responses obtained during the matching phase. This is done to return the content most relevant to the attacker's request, thereby achieving the goal of deceiving the attacker.

In the response message generation phase, if the request message $\boldsymbol{X}$ completely matches a request $\boldsymbol{X}^{rq}_{best}$ from the knowledge base, the decoded response data of $\boldsymbol{X}^{rs}_{best}$ is directly sent to the attacker. If the request message does not completely match any message in the knowledge base, We fill the response data based on the four types of payloads.

By extracting the valid payloads from the decoded response message based on MySQL protocol fields, we use l_{db} ,l_{tb}, l_{cl}, l_{rd} to represent the respective counts of database names, table names, column names and row data in the payload. If $l_{db} = l_{tb} = l_{cl} = l_{rd} = 0$, the decoded response data of $\boldsymbol{X}^{rs}_{best}$ is directly sent to the attacker.

For each database name in response data , We randomly select a database name from $\boldsymbol{P}^{db}$ for replacement. Similarly, For each table name in response data, We randomly select a table name from $\boldsymbol{P}^{tb}$ for replacement.

If $|\boldsymbol{P}^{cl}| > l_{cl}$, In the MySQL protocol packet, each column requires a field packet layer to represent it. Therefore, we need to add some field packet layers to ensure that the packet contains all the column names. We replace the column names in the first l_{cl} field packets of $\boldsymbol{P}^{cl}$ with the corresponding payloads. As for the remaining $|\boldsymbol{P}^{cl}|$-l_{cl} payloads, we choose to replicate $|\boldsymbol{P}^{cl}|$-l_{cl} field packet layers containing the l_{cl}-th payload located in the last field packet layer, modify the column names corresponding to $\boldsymbol{P}^{cl}$, and add $|\boldsymbol{P}^{cl}|$-l_{cl} modified field packet layers to the response.

If $|\boldsymbol{P}^{cl}|<l_{cl}$, after we replace the column names in the first $|\boldsymbol{P}^{cl}|$ field packets of $\boldsymbol{P}^{cl}$ with the corresponding payloads. we need to remove l_{cl}-$|\boldsymbol{P}^{cl}|$ field packet layers to ensure that there are no redundant column names.

After the modification of the first three payloads, we finally proceed with the modification of the row data. Through the analysis of the MySQL protocol , we identified that the response message's row data is primarily stored in row packet layers. For each row packet layer, there are $|\boldsymbol{P}^{cl}|$ rows of data and we modify the first $|\boldsymbol{P}^{rd}|$ rows to correspond to $p_n^{rd} \in \boldsymbol{P}^{rd}$, $n \in \{0,1,\ldots,|\boldsymbol{P}^{rd}|\text{-}1\}$.

Based on the modification methods for the four types of payloads, the response message is altered to change the payload information, providing attackers with the data they need, ultimately maximizing the deception of the attacker's objectives.

IV. Experimental Results and Analysis

A. Experimental Setup

The data used in the experiment was sourced from historical traffic in various domains such as education, finance, and movies, captured during the attack processes of external attackers on databases. The captured packets were further processed, and the cleaned data was ultimately saved in pcap file format. The entire pcap file comprises over 5000 packet-related message information regarding requests and responses for attacker commands during the attack. Dataset is divided into the training set, the validation set and the testing set according to the ratio of 8 : 1 : 1.

Our model has 12 layers, with a hidden state size of 768 for each layer. In pre-training, the batch size is 2 and the total steps is 10,000. We set the learning rate is 0.0001, and the ratio of warmup is 0.1. We fine-tune with the Adam optimizer, where the learning rate is set to 0.001. All the experiments are implemented with Pytorch 1.8.0, conducted with NVIDIA GRID A100 PCIe.

B. Experimental Metrics

In this section, we will describe the performance of the system based on the attack data obtained in the actual deployment. To evaluate the performance of each system, we use the following criteria:

1) Accuracy of the response. We use accuracy to determine whether a response is generated properly.

2) Session length. We send a request to the client, the honeypot returns a response, and the communication is terminated, defining its session length to 1. The longer the session [18], the more likely an attacker is to perceive the honeypot as a natural system.

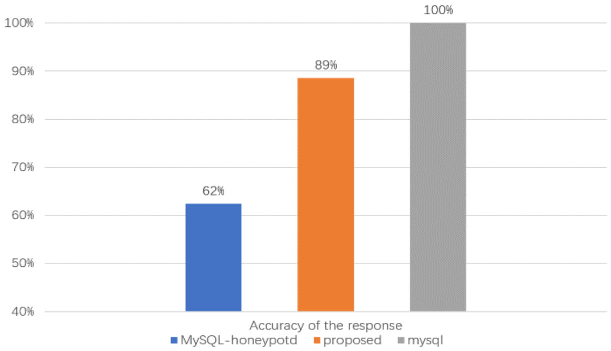


Fig. 4. Comparison of accuracy

3) Speed of response. We believe that the faster the honeypot responds, the more intelligent the honeypot is, the closer it is to the real system.

C. Accuracy of the Response

In response to the rule-based low-interaction honeypot MySQL-honeypotd [19], our proposed honeypot, and a real database service, we sent 200 identical request data to each. To assess the reasonableness of response generation, we calculated the Jaccard coefficient between the responses generated by the response of two honeypots and the response from the real database service. We set a threshold of 0.6, considering responses as correct if the coefficient is greater than 0.6, otherwise, it is considered an incorrect response. Additionally, if the honeypot fails to respond to a request, we also consider the generated response as incorrect. From Fig. 4, it can be observed that the accuracy of responses generated by our proposed honeypot is much higher than that of open-source honeypots.

D. Session Length

For comparison, in addition to our smart interactive honeypot, we have prepared a rule-based low-interaction honeypot (MySQL-honeypotd). We also prepare a simple interactive honeypot that responds to Response OK packets for all MySQL requests. Fig. 5 illustrates the session lengths of our honeypot and compared honeypots over a 30-day observation period. It's important to note that the number of client machines accessing the honeypots varies by location, so for clarity, the results are presented as percentages. Our honeypot exhibits a higher percentage of extended interactions compared to other honeypots, especially interactions with a duration exceeding 7 times.

In order to determine the reasons for the continuation of communication, we used SQLMap [20] on our proposed honeypot and the mysql-honeypotd honeypot. This tool tests whether the system responds to various existing attacks. As a result, our honeypots responded to attacks ranging from one to four attempts, whereas the rule-based honeypot did not respond to any attacks. In other words, from the attacker's perspective,

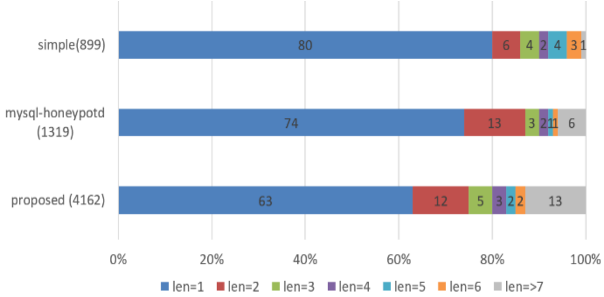


Fig. 5. Comparison in the percentage of session length with clients between honeypots. The number after the honeypot represents "total number of requests received"

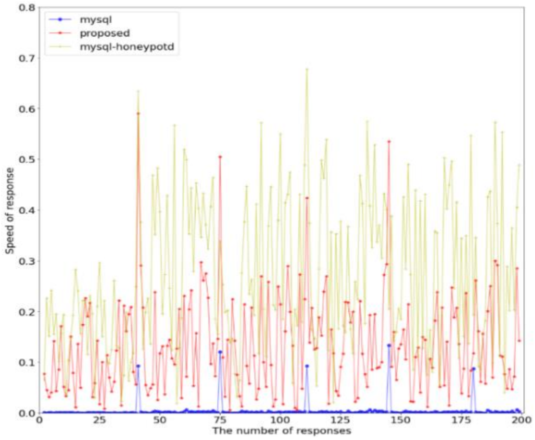


Fig. 6. Comparison of response speeds

our honeypots appear to be vulnerable systems that react to attacks, allowing communication to proceed.

E. *Speed of Response*

To evaluate the execution speed of our honeypot, we compare our intelligent interaction honeypot with a real MySQL server and mysql-honeypotd. We send 200 identical SQL requests to the MySQL service, and for each request their response time is shown in Fig. 6. The real MySQL server response fastest, all within 0.5 seconds. In addition, the honeypot proposed in this paper takes less time to respond than the mysql-honeypotd honeypot in most cases.

V. Conclusions and Future Works

This paper proposes a framework for intelligent interaction honeypot using language models to observe attacks on database systems and evaluate the performance of their responses. The experimental results show that the session length of the low-interaction honeypot based on rule matching performs well, but it cannot respond to novel and unknown attacks. The word embedding method based on semantic knowledge learning is more flexible and can better handle this situation. The proposed honeypot session length and response speed are higher than those of open source honeypots, indicating that the model has better generalization ability. However, the model is limited to MySQL-based database systems and does not learn enough SQL dialects due to the small sample size. In the future, we will implement a common honeypot framework and try to reduce the size of the model while increasing the speed of inspection.

References

[1] Gunay Abdiyeva-Aliyeva and Mehran Hematyar, "Statistic Approached Dynamically Detecting Security Threats and Updating a Signature-Based Intrusion Detection System's Database in NGN," Journal of Advances i n Information Technology, Vol. 13, No. 5, pp. 524-529, October 2022.

[2] verizon. 2012 data breach investigations report. http://www.verizonenterprise.com/security/blog/.

[3] Cheng J R, Yin J P, Liu Y, et al. Advances in the honeypot and honeynet technologies[J]. Journal of Computer Research and Development, 2008, 45(1): 375-378.

[4] A. Acien, A. Nieto, G. Fernandez, and J. Lopez, "A comprehensive methodology for deploying iot honeypots," 15th Int. Conf. Trust Priv. Secur. Digit. Bus. Trust. 2018, vol. 11033 LNCS, no. TrustBus, pp. 229–243, 2018, doi: 10.1007/978-3-319-98385-1_16.

[5] B. Wang, Y. Dou, Y. Sang, Y. Zhang, and J. Huang, "IoTCMal: Towards A Hybrid IoT Honeypot for Capturing and Analyzing Malware," IEEE Int. Conf. Commun., vol. 2020-June, no. November, 2020, doi: 10.1109/ICC40277.2020.9149314.

[6] Conti M, Trolese F, Turrin F. Icspot: A high-interaction honeypot for industrial control systems[C]//2022 International Symposium on Networks, Computers and Communications (ISNCC). IEEE, 2022: 1-4.

[7] Guan C, Liu H, Cao G, et al. HoneyIoT: Adaptive High-Interaction Honeypot for IoT Devices Through Reinforcement Learning[J]. arXiv preprint arXiv:2305.06430, 2023.

[8] Ren J, Zhang C, Hao Q. A theoretical method to evaluate honeynet potency[J]. Future Generation Computer Systems, 2021, 116: 76-85.

[9] Ashenden D, Black R, Reid I, et al. Design thinking for cyber deception[J]. 2021.

[10] Ahmad W, Arsalan M, Nawaz S, et al. Detection and Analysis of Active Attacks using Honeypot[J]. International Journal of Computer Applications, 975: 8887.

[11] Sanjeev Kumar, Rakesh Sehgal, Paramdeep Singh, and Ankit Chaudhary, "Nepenthes Honeypots Based Botnet Detection," Journal of Advances in Information Technology, Vol. 3, No. 4, pp. 215-221, November, 2012.doi:10.4304/jait.3.4.215-221.

[12] Zhang L, Thing V L L. Three decades of deception techniques in active cyber defense-retrospect and outlook[J]. Computers & Security, 2021, 106: 102288.

[13] Guarnizo j, Tambe a, Bhunia s, et al. SIPHON: Towards scalable high-Interation physical honeypots[C]. The 3rd ACM Workshop on Cyber-Physical System Security, New York, USA, 2017:57–68. doi:10.1145/3055186.3055192.

[14] Buzzio-Garcia J. Creation of a High-Interaction Honeypot System based -on Docker containers[C]. 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4). IEEE, 2021: 146-151.

[15] Chowdhary V, Tongaonkar A, Chiueh T. Towards automatic learning of valid services for honeypots[J]. Controls, Automation of Communication Systems (ICCACS2004), 2004: 246.

[16] Askari A, Abolghasemi A, Pasi G, et al. Injecting the BM25 Score as Text Improves BERT-Based Re-rankers[C]//European Conference on Information Retrieval. Cham: Springer Nature Switzerland, 2023: 66-83.

[17] Dharma E M, Gaol F L, Warnars H, et al. The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification[J]. J Theor Appl Inf Technol, 2022, 100(2): 31.

[18] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar:Towards an intelligent-interaction honeypot for iot devices. Black Hat, pages 1–11, 2017.

[19] mysql-honeypotd. https://github.com/sjinks/mysql-honeypotd.

[20] O. Ojagbule, H. Wimmer and R. J. Haddad. Vulnerability Analysis of Content Management Systems to SQL Injection Using SQLMAP. SoutheastCon 2018, St. Petersburg, FL, USA, 2018, pp. 1-7, doi: 10.1109/SECON.2018.8479130.