HoneyLLM: A Large Language Model-Powered Medium-interaction Honeypot

Wenjun Fan[1,2], Zichen Yang[1,2], Yuanzhen Liu[1], Lang Qin[1] and Jia Liu[1,2].

[1] School of Advanced Technology, Xi'an Jiaotong-Liverpool University
[2] Department of Computer Science, University of Liverpool

August 2024

Outline

- Introduction
- Background
- Motivation
- Approach
- Results
- Conclusion

Introduction

Honeypot is a trap-like defense measure created to be probed, attacked and compromised in order to capture malicious behaviors.[1]

⇨ **Virtual Honeypot (focused on) /** Physical Honeypot

- Use minimal system resources and lack full system functionality, called low-interaction (LIH) or medium-interaction honeypots (MIH).
- Build on the genuine OS, called high-interaction honeypot (HIH).

Large Language Models (LLMs) are a subset of pre-trained language models (PLMs).

---

[1] *Lance Spitzner. Honeypots: Tracking Hackers. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.*

Motivation

Interaction capacity of honeypot is the level of the information system resource exposed to the network adversary.

Issues

- High-interaction honeypot (HIH) has to disclose the full operating system which leads to a higher security risk and is less scalable for deployment.
- Low-/medium-interaction honeypot (LIH/MIH) has a lower security risk while it can only capture network-level data, lacking system-level data.

Key point -> Balance between security and utility of honeypot

⇨ We propose an **LLM-powered MIH** for constituting SSH honeypot to **capture system-level data**, with **shell evaluation** to assess LLM response performance.

Background (Hacking with Shells)

- **Bind shell** -> the listener runs on the target node, allowing the adversary gain remote shell access.

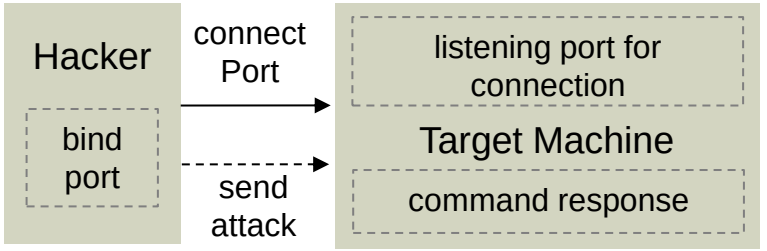

Fig.1 Bind shell connection.

- Reverse shell -> the listener runs on the adversary node, enabling the target node to connect back with shell.
  - ① adversary bypass the firewall restrictions
  - ② adversary does not face the NAT address translation.

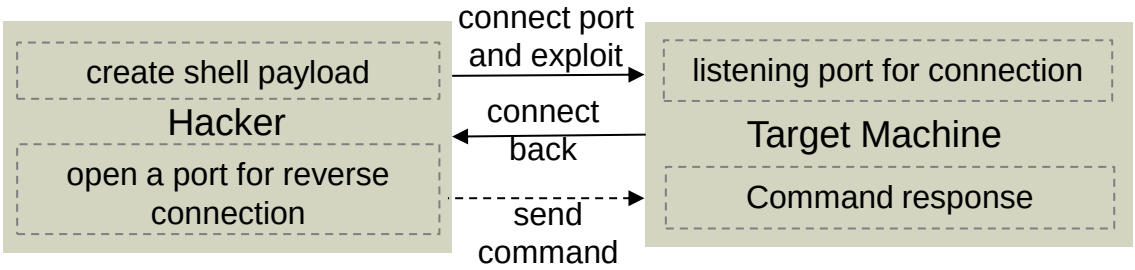

Fig.2 Reverse shell connection.

Methods (LLM-powered Honeypot System Design)

Overview

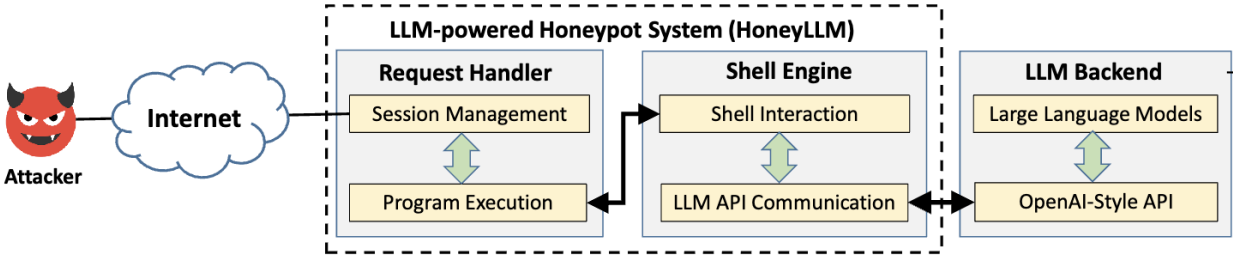

Fig.3 Overview of HoneyLLM.

- Request Handler (RH): handle malicious connection requests, forwarding and displaying commands/responses between the attacker and the shell.
- Shell Engine (SE): supply bogus responses using LLM to the attacker.

Methods (LLM-powered Honeypot System Design)

Request Handler (RH)

- Session Management

  It handles the transport-layer connection/ the application-layer session

  After the TCP handshake, the program provides application-layer responses, such as SSH session setup.

- Program Execution

  It prepares the environment, and then executes the command.

  In our case, the program execution relays the attacker's command to the shell engine and forwards the shell's bogus response back to the attacker.

Methods (LLM-powered Honeypot System Design)

Shell Engine (SE)

- Shell Interaction

  It shows initial shell prompt, collect user input, and display LLM output.

- LLM API Communication

  It is created for interacting with the corresponding LLM.

  API responses will be appended to history, which then will be sent to the shell interaction program.

Methods (LLM-powered Honeypot System Design)

LLM Backend (external component)

- LLMs
- OpenAI-Style API

API is only required to provide an OpenAI-compatible chat completion endpoint.

Large language model stand behind the the API.

Methods (LLM-powered Honeypot System Design)

Shell evaluation metrics

Table.1 Metric description.

Metric	Description
Role Consistency	It refers to the LLM's ability to stay in character as a shell.
Instruction Following	It represents the LLM's ability to follow instructions accurately.
Command Decomposition	It indicates the LLM's ability to process the compound commands.
Parameter Handling	It denotes the LLM's ability to handle command arguments.
Error Handling	It signifies the LLM's ability to identify and handle error scenarios.
Context Awareness	It pertains to the LLM's awareness of the shell environment.
Output Formatting	It shows the LLM's ability to generate output in the correct format.

We propose a set of shell evaluation metrics to evaluate the quality of each LLM for honeypot shell response, designated as "ShellEval".

Implement

- Deploy the prototype on a Raspberry Pi 5 (4GB, UK) running **64-bit Raspberry OS**, compiled with **Go 1.22.1**.
- The device has a static public **IPv4 address (31.205.7.193)** provided by the ISP.
- The *cli2ssh* listens on port 22 and is configured to execute *aish* with formatted environment variables for each session.
- The shell simulator, *aish*, interacts with the OpenAI API via OpenRouter, using primarily the **GPT-4** model for response generation.

Results

Comparison of Large Language Models

Table 2. The performance comparison of different LLMs.

Model	Hit-rate (ShellEval)	Latency (s)	Throughput (t/s)	Input-cost (USD/1M Tokens)	Output-cost (USD/1M Tokens)
Mistral 7B	31.25%	0.17	103.45	0.07	0.07
GPT-3.5 Turbo	50%	0.42	98.93	0.50	1.50
GPT-4 Turbo	68.75%	1.29	30.27	10.00	30.00
GPT-4o	75%	0.81	81.89	5.00	15.00
Claude 3 Opus	68.75%	3.46	19.63	15.00	75.00
Gemini 1.5 Pro	56.25%	1.51	49.84	7.00	21.00

① Claude 3 Opus and GPT-4 series have the best hit rate, followed by Gemini 1.5 Pro and GPT-3.5 Turbo. Mistral 7B performs the worst.

② Claude 3 Opus has higher latency and lower throughput, making it less suitable for honeypot shell simulation.

③ Mistral 7B is unstable, and GPT-4 Turbo is too costly.

We have opted for GPT-4 Turbo as the cornerstone of our prototype.

Results

End-to-end Session Performance

⇨ The experiment performs evaluation by "executing time ssh localhost echo hi" 10 times and calculating the average real time.

① Time Costs: OpenSSH averages 0.1748s, Cowrie 0.1380s, and our honeypot 1.8849s.
② Our honeypot is slower than OpenSSH and Cowrie, likely due to Go implementation, additional shell simulator execution, and reliance on a remote LLM server.
③ Cowrie's fake SSH service outperforms real OpenSSH, likely due to its lightweight nature.

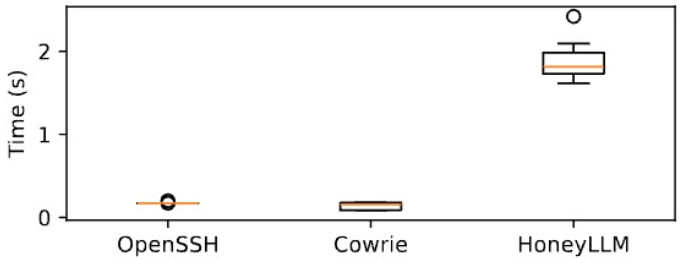

Fig 4. The comparison of time required for a simple session.

Results (Analyses on Real-world Attacks Captured)

Top Attack Sources Captured

① Top IPs: 170.64.211.62 had the most accesses (3,519), followed by 170.64.190.158 (3,315), both from the US.
② Other significant IPs include 118.26.194.190 (1,638) from China and 45.125.66.43 (1,466) from Hong Kong.
③ Lowest Access: 170.64.144.244 had the lowest (933).
④ All US-based IPs in the top 10 are from DigitalOcean, a cloud VM provider.

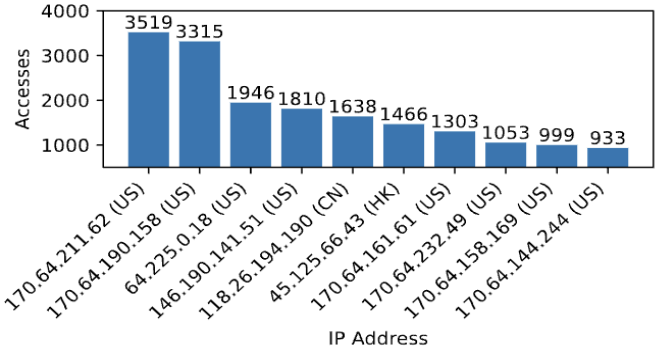

Fig. 5 The most popular attack source IP addresses.

Results (Analyses on Real-world Attacks Captured)

Attack Connection Duration Observed

- Cowrie: (1) Connection durations in the range of seconds, with more than 50% lasting just 1 sec. (2) Around 15% of connections last more than 100 sec.
- Amun: (1) Captures connection durations as short as 10^{-8} sec. (2) About 40% of connections last less than 1 sec, 20% last more than 10 sec, 5% exceed 100 seconds.
- LLM-powered MIH: (1) Approximately 95% of connections last more than 1 sec, with 87% lasting between 1 and 10 sec. (2) The longest recorded connection lasted 952 sec, longer than any captured by Cowrie or Amun.

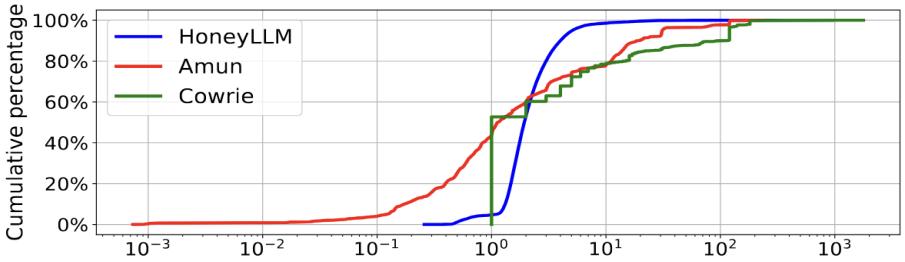

Fig. 6 The comparison of attack session duration using different honeypots.

Results (Analyses on Real-world Attacks Captured)

Popular Attacking Commands Observed

- HoneyLLM captured common commands for system info gathering, backdoor installation, and GPU detection.
- In reenacting 25 common commands, **HoneyLLM** performed best, successfully executing 21; **Cowrie** executed 8; **Amun** performed worst, often returning "command not found" errors.

Table 3. The captured popular commands.

Index	Command	Count
1	uname -s -v -n -r -m	2036
2	cd ~; chattr -ia .ssh; lockr -ia .ssh	2008
3	cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa ..."">>.ssh/authorized_keys && chmod -R go= ~/.ssh && cd ~	2002
4	uptime | grep -ohe 'up .*' | sed 's/,//g' | awk '{ print $2" "$3 }'	1349
5	lscpu | egrep "Model name:" | cut -d ' ' -f 14-	1347
6	curl ipinfo.io/org	1347
7	nproc	1344
8	echo -e "ok"	824
9	lspci | egrep VGA && lspci | grep 3D	712
10	uname -m	703
11	uptime -p	633
12	lspci | grep VGA | cut -f5- -d ' '	612
13	lspci | grep VGA -c	591
14	nvidia-smi -q | grep "Product Name" | head -n 1 | awk '{print $4, $5, $6, $7, $8, $9, $10, $11}'	575
15	lspci | grep "3D controller" | cut -f5- -d ' '	565
16	nvidia-smi -q | grep "Product Name" | awk '{print $4, $5, $6, $7, $8, $9, $10, $11}' | grep . -c	559
17	ip r | grep -Eo '[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}/[0-9]{1,2}'	552
18	if command -v lspci &>/dev/null; then lspci | egrep VGA | grep NVIDIA | awk '{print $5}' |wc -l; else nvidia-smi -q | grep "Product Name" | awk '{print $4, $5, $6, $7, $8, $9, $10, $11}' |wc -l; fi	354
19	ip r | grep -Eo '[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}/[0-9]{1,2}'	354
20	cat /etc/passwd |grep -v nologin |grep -v false |grep -v sync |grep -v halt|grep -v shutdown|cut -d: -f1	354
21	echo -e "Kplm041@\nKplm041@" | passwd ;history -c;history -c ;history -c ;cd /tmp/ ; mkdir .est1; cd .est1 ;rm -rf * .* ; curl -O 91.92.254.166:8181/.dcplm || wget 91.92.254.166:8181/.dcplm; chmod +x .dcplm; ./.dcplm;rm -rf .dcplm;./.system3d;history -c; rm -rf ~/.ash_hisotry	343
22	lspci | egrep VGA && lspci | grep 3D	283
23	nvidia-smi -q | grep "Product Name" | awk '{print $4, $5, $6, $7, $8, $9, $10, $11}' | wc -l | head -c 1	283
24	nvidia-smi -q | grep "Product Name"	283
25	lspci | egrep VGA | grep Radeon | wc -l | head -c 1	283

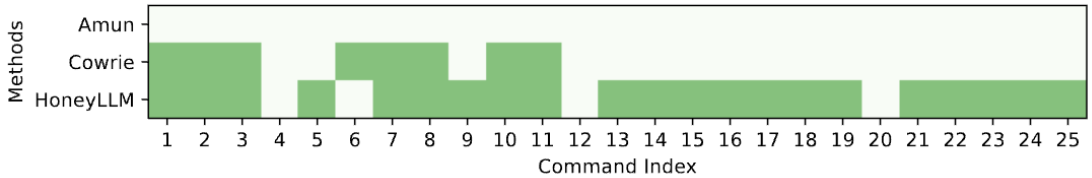

Fig 7. Command success comparison across different honeypots.

Results (Analyses on Real-world Attacks Captured)

Popular Credentials

- - "***345gs5662d34***" is used as both a username and a password by attackers.
  - "***3245gs5662d34***" is used as password, not be common weak password in real-world.
  Common Credentials Used by Attackers => detect **whether the target is a honeypot** instead of a real production server.

- Popular public keys used for creating backdoors occur in multiples of 20.

Table 4. The popular captured public key.

Public Key	Count
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCcPjOVNHzQQYmUsyGOf...	200
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDMtPQsyaqw+aw2IVa/8...	20
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDW7bXyg1qQpzOijaWpT...	20
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDIHOZv7Y48fd8PViQw8...	20
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAABAEAwsFzinSlj2egX2w1f3FOU...	20
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCBBNG9ZWFubdzlVhtet...	20
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC8vlPpc3X7NgX49pTAO...	20
ssh-ed25519 AAAAC3NzaC11ZDI1NTE5AAAAILZMR3AsV6mzndFLFF/og...	20
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDTpdG+f24ZLGM1XY2PT...	20
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDevQyCt06VKbdFcD680...	20

Conclusion

Our research demonstrates the potential of using large language models to enhance **medium-interaction honeypots**. **HoneyLLM** ‘s ability to provide authentic system interactions has proven effective in capturing detailed attacker behavior and increasing engagement duration.

Future Work

1. Optimize performance and reduce response times.
2. Conduct more extensive real-world testing to evaluate HoneyLLM.
3. Improve the honeypot implementation for supporting more shortcomings such as reverse shell.

Thank you!

If you have any questions, please contanct me anytime.
J.Liu154@liverpool.ac.uk