Build a Honeypot to Trap Hackers & Bots in Python

Rudra Kadel

March 8, 2025

Abstract

This document details the development of a Python-based SSH honeypot system designed to attract, monitor, and analyze malicious access attempts. The honeypot simulates a vulnerable SSH server environment, logs authentication attempts, and captures commands executed by attackers. Through this project, we gain insights into attack patterns, common intrusion techniques, and can develop more effective defensive strategies.

Contents

1 Introduction

A honeypot is a cybersecurity mechanism designed to detect, deflect, and analyze unauthorized access attempts. By creating a deliberately vulnerable system, security professionals can observe attack patterns, understand threat actor behavior, and enhance overall security strategies. This project aims to build a basic honeypot using Python to log and analyze unauthorized access attempts.

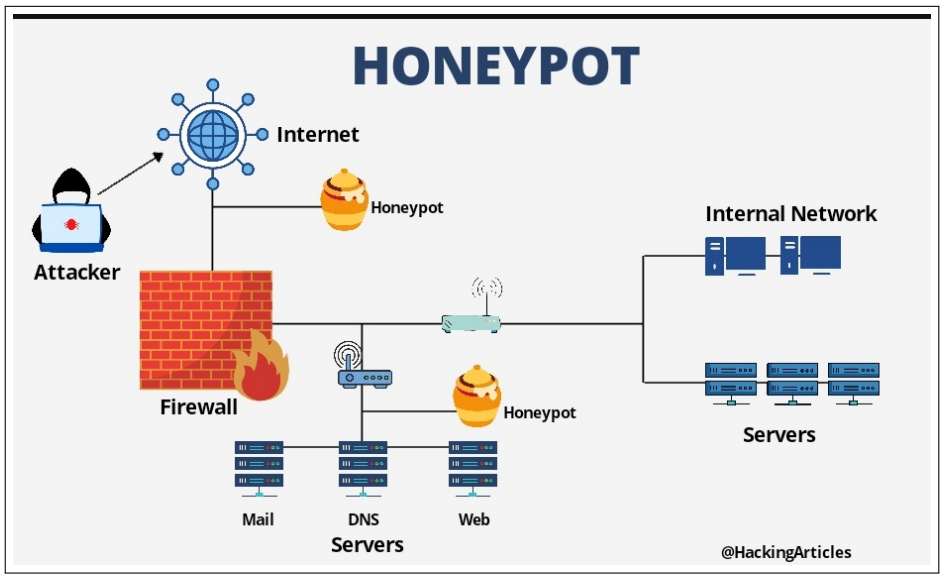
1.1 Project Objectives

- Create a simulated vulnerable SSH server to attract attackers
- Capture and log authentication attempts including credentials
- Record all commands executed by attackers in the system
- Analyze attack patterns and intrusion techniques
- Develop insights for improving security measures

2 How Honeypots Work

Honeypots are intentionally vulnerable systems that mimic real-world servers or services to attract cyber attackers. These systems allow security professionals to:

- Identify attack techniques used by malicious actors.
- Log unauthorized access attempts for forensic analysis.
- Enhance defensive security mechanisms by studying real-world threats.
- Divert attacks away from production systems.
- Gather intelligence on emerging threat vectors.



Types of Honeypots

- **Low-interaction honeypots:** Simulate only basic services with limited functionality
- **Medium-interaction honeypots:** Offer more sophisticated emulation of services
- **High-interaction honeypots:** Provide fully functional systems for deeper attacker engagement

Our SSH honeypot falls between low and medium interaction, offering enough functionality to engage attackers while maintaining control and security.

3 Project Requirements

- WSL Ubuntu (for running SSH services)
- VS Code (for development and editing)
- GitHub (for version control)
- Python Libraries: Paramiko, Socket, Logging, Threading
- SSH key generation tools
- Log analysis capabilities

4 Project Flow

1. Start an SSH server using Paramiko that listens for incoming connections.
2. Capture login attempts and log username-password pairs.
3. Emulate a simple shell to capture commands executed by the attacker.
4. Store logs securely for analysis.
5. Process and analyze collected data to identify patterns.

5 Architecture

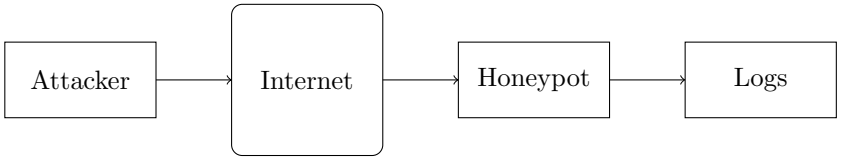


Figure 1: Basic architecture of the SSH honeypot system

6 Code Breakdown

The project consists of various components that work together to simulate an SSH honeypot:

6.1 Server Class

- Implements the SSH authentication mechanism.
- Logs every authentication attempt (username and password) to a log file.
- Always returns authentication failure to keep attackers engaged.

```
class SSHServer(paramiko.ServerInterface):
    def __init__(self, client_ip):
        self.client_ip = client_ip
        self.event = threading.Event()

    def check_auth_password(self, username, password):
        # Log the authentication attempt
        logger.info(f"{self.client_ip} attempted login with
    username: {username}, password: {password}")
        return paramiko.AUTH_FAILED
```

Listing 1: Example Server Class Code

6.2 Emulated Shell Function

- Provides a fake command-line interface to deceive attackers.
- Logs every command executed by the attacker.
- Returns simulated outputs for common Linux commands.

```
def handle_shell(chan, client_ip):
    chan.send("Welcome to Ubuntu 20.04.3 LTS\r\n")
    chan.send("$ ")

    while True:
        command = ""
        while not command.endswith("\r"):
            data = chan.recv(1024)
            if not data:
                break
            chan.send(data)
            command += data.decode("utf-8")

        command = command.rstrip("\r")

        # Log the command
        cmd_logger.info(f"{client_ip} executed command: {command}")

        # Process the command and return fake output
        if command == "ls":
            chan.send("\r\npassword_config  users.db  backup.tar.gz
    \r\n")
        elif command.startswith("cat"):
            chan.send("\r\nUSER=admin\r\nPASSWORD=admin123\r\n")
        else:
            chan.send("\r\nCommand not found\r\n")

        chan.send("$ ")
```

Listing 2: Example Shell Emulation Code

6.3 Client Handling Function

- Handles new client connections and initiates the SSH handshake.
- Sends a fake welcome banner to the attacker.
- Creates a thread for each session to handle concurrent connections.

6.4 Main Honeypot Function

- Binds a socket to listen for SSH connection attempts.
- Spawns new threads for each connection.
- Manages server resources and ensures stability.

7 Special Features

7.1 Tarpit Mode

The honeypot includes an optional tarpit mode which deliberately slows down responses to waste attackers' time and resources:

```
def tarpit_delay():
    """Add random delays to waste attacker time"""
    time.sleep(random.uniform(0.5, 3.0))

# Use in command processing
if TARPIT_MODE:
    tarpit_delay()
    chan.send("\r\nProcessing...\r\n")
    tarpit_delay()
```

Listing 3: Example Tarpit Implementation

7.2 Deceptive Content

The honeypot includes fake sensitive files to entice attackers:

- Fake password configuration files
- Dummy user databases
- Misleading system information

8 Data Analysis

8.1 Credential Analysis

The honeypot collects usernames and passwords which can be analyzed to identify:

- Most common username/password combinations
- Password complexity patterns
- Brute force attack patterns

8.2 Command Analysis

By analyzing executed commands, we can determine:

- Common post-exploitation activities
- Malware deployment techniques
- Lateral movement attempts
- Data exfiltration methods

9 Security Considerations

Running a honeypot involves inherent risks that must be addressed:

9.1 Isolation

The honeypot should be properly isolated to prevent attackers from using it as a pivot point:

- Run in a separate network segment
- Use containerization or virtualization
- Implement strict firewall rules

9.2 Legal Considerations

Operating honeypots may have legal implications:

- Ensure compliance with local and international laws
- Consider privacy regulations for data collection
- Document authorization for honeypot operation
- Be aware of entrapment concerns

9.3 Maintenance and Monitoring

- Regularly inspect honeypot logs for potential compromises
- Update and patch the honeypot system
- Monitor resource usage to prevent denial of service

10 Future Enhancements

- Integration with threat intelligence platforms
- Automated alerting system for specific attack patterns
- Enhanced shell emulation with more realistic responses
- Honeytokens implementation to track data access
- Machine learning for attack classification
- Multi-service honeypot (FTP, HTTP, etc.)

11 Conclusion

The SSH honeypot project provides valuable insights into attacker behavior while serving as an early warning system for new attack techniques. By carefully analyzing the collected data, security professionals can improve defensive measures and stay ahead of emerging threats. This project demonstrates the value of deception technology in modern cybersecurity strategies.

12 References

- Mentor: Grant Collins (YouTube Video)
- Project Code: GitHub Repository
- Paramiko Documentation: http://docs.paramiko.org/
- Blog on Creating a Honeypot by Justin H. Hooper Medium Blog

13 Appendix: Installation and Usage

13.1 Environment Setup

```
# Create virtual environment
python -m venv honeypot_env
source honeypot_env/bin/activate

# Install dependencies
pip install -r requirements.txt

# Generate SSH keys
ssh-keygen -t rsa -b 2048 -f static/server.key -N ""
```

Listing 4: Setup Commands

13.2 Running the Honeypot

```
# Start the honeypot on port 2222
python ssh_honeypot.py

# In another terminal (for testing)
ssh -p 2222 admin@127.0.0.1
```

Listing 5: Running the Honeypot