International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

HoneyShield: An Intelligent Multi-Protocol Honeypot and Threat Intelligence Platform

Rajnandini Patil1, Om Mahapadi2, Pranav Gadre3

[1,2,3]Department of Computer Science, MIT ADT University, Pune, India

Abstract

This paper is about HoneyShield, a system that can trick and catch people who try to attack computers. These days people who try to hack into computers use automated programs that keep trying to find spots in internet services. They look for things like passwords, unfixed security problems and misconfigured computer settings. Most security tools, like firewalls and intrusion detection systems are good at blocking these attacks. They do not really try to understand how the attacks work.

HoneyShield fills this gap by pretending to be a computer service. It can act like SSH, HTTP, FTP and Telnet all at the same time. It catches the ways that attackers try to break in like what they do and how they do it and stores this information in a database. The system also adds information to what it catches, like where the attack is coming from and it has a special file system that makes it look like a real computer to the attackers. It even has a way for other programs to talk to it and get information out. It shows all of this information in a real-time dashboard.

We made HoneyShield so that it can be set up quickly in the cloud and when we tested it it caught over 50 attacks per test across all four types of services. It can also show the information it catches quickly and it can figure out where the attacks are coming from with over 90% accuracy. HoneyShield is meant for people who work with computer security like SOC analysts, security researchers and IT administrators at medium-sized businesses, who want to know more, about the threats they face without having to deal with complicated systems.

Keywords - honeypot, threat intelligence, cybersecurity, multi-protocol, SSH, force, geolocation, Docker, virtual file system, attack analytics

I. INTRODUCTION

The internet is getting bigger and more things are connected to it which means that servers that people can reach from the internet are always being attacked by automated and targeted cyberattacks. Servers that are hosted on cloud services get thousands of attempts to connect to them every day from people who use computers to try and find weak passwords, unpatched web services, open file transfer protocol relays and old Telnet endpoints. The usual ways to protect against these threats, such as firewalls and systems that prevent intrusions can block them. Do not give us much information about what the attackers are doing what passwords they are trying or what weaknesses they are trying to use.

Honeypots are a way to deal with this problem. Of blocking attackers they attract them and watch what they do. By setting up services that look real the person running the honeypot can see exactly what the attackers are trying to do what passwords they are using what commands they are running and what weaknesses they are trying to use. This information can be used to make the real systems more secure.

There are already some honeypot solutions, such as Cowrie and Kippo. They only work with SSH and produce log files that need to be looked at manually. There is no honeypot platform that can handle multiple protocols log events in a structured way add location information to the logs and provide a dashboard to look at the data.

HoneyShield is a platform that fills these gaps. It can run listeners for SSH, HTTP, FTP and Telnet at the same time and it logs every event to a database. It also adds location information to the logs. Provides a dashboard to look at the data. The entire platform is packaged in a container. Can be deployed on a new Amazon Web Services instance in under five minutes.

This paper talks about the design, implementation and testing of HoneyShield version 1.0. The next section talks about related work. The section after that describes the system architecture and methodology. The following section explains how each part of the system was implemented. The section after that presents the results of the testing. The final section. Talks about future plans, for HoneyShield.

II. LITERATURE REVIEW

A. Basic Honeypot Research

Spitzner did some important work in 2003. He showed that honeypots can be a tool for security. Honeypots are like traps that catch people who try to attack computers. Spitzner used the Honeynet Project to prove that these traps can get information from attackers. When attackers think they are attacking a computer they show what tools they use and what they want to do.

B. Looking at Honeypot Techniques in Cloud Environments

Some people named Al-Duwairi, Alenezi. Kotsiopoulos did a big study on honeypots in cloud environments. They found out that honeypots in the cloud get attacked a lot more than honeypots that're not in the cloud. This is because computers in the cloud are always being scanned by computers. They said that we need to make honeypots that can handle a lot of attacks and are made for the cloud.

C. Medium-Interaction Honeypots

Some researchers named Seifert, Welch and Komisarczuk looked at medium-interaction honeypots in 2006. They said that these honeypots are the best because they give us a lot of information and are safe to use. They do not let attackers really use the computer. They make it seem like they can. This makes them good for using all the time without anyone watching.

D. Cowrie and its Limitations

There is a honeypot called Cowrie that is used a lot. It is good at catching people who try to attack computers using SSH. But it only works for SSH so it misses attacks that use ways to connect to the computer. People who used Cowrie said that it is not enough to watch for one kind of attack.

E. New Cloud Honeypot Deployments

Some people named Mahapadi and others used a medium-interaction honeypot on a computer. They used Cowrie and some Python scripts to make it work with a security program. They found out that their honeypot caught attacks within a few hours. They also saw that the attackers were from around the world.. They only watched for two kinds of attacks and used another program to analyze the data.

HoneyShield is a honeypot that does more, than the others. It can watch for four kinds of attacks at the time. It also has its way of analyzing the data so it does not need another program.. It is easy to set up and use.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

A. How We Designed HoneyShield

We built HoneyShield around four ideas: realism, safety, observability and deployability. Realism is when every simulated service looks and acts like the thing so attackers will stay engaged for a longer time. Safety means we never run any code that attackers try to send us and we use Docker to keep the honeypot separate from everything so it cannot be used to attack other systems. Observability means we keep track of every interaction with timestamps and in a way that we can search for things. Deployability means we can get the platform up and running with just one command.

B. How The System Is Organized

The system has four parts that work like a standard data pipeline:

- The Capture Layer is where we listen for connections. We have listeners that run at the same time and each one can handle many connections. They all write what they see to a shared database.
- The Storage Layer is where we keep all the data. We use a system called SQLAlchemy to talk to our databases whether we are testing or live. We have eight tables to store kinds of data like attack events, credentials and geolocation data.
- The Enrichment Layer is where we add information to what we have captured. We have a worker that looks up where IP addresses come from using a service called ip-api.com. We also have a local database to fall back on. This gives us information about each IP address like the country and city.
- The Presentation Layer is where we make the data available to users. We have a backend that uses FastAPI and exposes endpoints that our React frontend can use. We also support WebSockets so users can get updates without having to refresh the page.

C. What Is In Our Database

Our database has eight tables. The attack_events table is where we store everything that happens with columns for things like the time the IP address and what protocol was used. The credentials table stores. Passwords that attackers have tried. The commands table stores shell commands that attackers have entered and what the simulated response was. The http_requests table stores information about HTTP requests like the method and path. The sessions table stores information about each session like when it started and how long it lasted. The geo_data table stores geolocation data for each IP address so we do not have to look it up every time.

D. How We Keep HoneyShield Safe

Since HoneyShield is a security tool we have to make sure it is secure itself. We do things to make sure of this: we run all honeypot processes as non-root users, inside Docker containers so they cannot do much harm. We block all network traffic from the honeypot container so even if it is compromised it cannot attack other systems. We use a Virtual File System to keep the honeypot from accessing the filesystem. We require authentication to access the dashboard. We limit how many times someone can try to log in. We also mask any captured credentials when we export them so they are not visible.

IV. IMPLEMENTATION

A. Multi-Protocol Honeypot Engine

The main part of this system is the engine. It is located in honeypot_engine/engine.py. This engine starts each service. It does this as a Python thread. This means that if one service crashes or gets too busy it will not affect the services. Each service is based on a class called BaseService. This class helps with connections. It also helps with making session IDs and writing to the database.

The SSH honeypot uses a library called Paramiko. This library helps the honeypot look like a SSH server. It shows a banner that says SSH-2.0-OpenSSH_7.4. Every time someone tries to log in the honeypot captures the username and password. It does not matter if the login is successful or not. The honeypot will always say that the login failed. If someone does log in they will get a shell where they can type commands like ls, cd and pwd. The honeypot will show them a Linux directory tree with files in /etc /home and other places.

The HTTP honeypot is a server that speaks HTTP/1.1. It looks at the requests it gets and checks for attack patterns. It can detect things like directory traversal and SQL injection. It responds to requests with answers like 200 OK or 404 Not Found. The honeypot also pretends to be an Apache server.

The FTP honeypot is a state machine that implements the FTP commands. It captures the username and password when someone logs in. The honeypot responds with FTP codes to make it look real. The Telnet honeypot looks like a Linux login prompt. It captures the username and password. It only lets someone try to log in three times before it says they failed.

B. Database Logging

All the services write to the database using a shared connection. This connection is defined in database/connection.py. The database is set up to handle a lot of connections at the time. In production the system automatically switches to a PostgreSQL database if it is available.

C. Geolocation Service

The geolocation module is a worker that checks the database for new IP addresses. It uses a service called ip-api.com to get the location of each IP address. It stores this information in the database. The module also uses a cache to avoid making too many requests to the API. If the API is not available the module falls back to a database called MaxMind GeoLite2.

D. Dashboard Backend API

The backend API is built using FastAPI. It has eleven endpoints that are authenticated with JWT tokens. The API returns information like the number of events the number of unique attackers and the distribution of protocols. It also returns a timeline of events for the 24 hours. The API uses models to define the structure of the responses.

E. Test Suite

The test suite simulates real attacks, against the system. It tests the SSH, HTTP, FTP and Telnet services. The tests use wordlists and attack patterns. After all the tests are done the system checks the database to make sure that all the expected events were logged. It also checks that the credential records and protocol entries are correct.

V. RESULTS AND DISCUSSION

A. Protocol Coverage and Capture Rate

We tested HoneyShield on a development machine with a bundled test suite. It captured 39 attack events across four protocols in one test session. These events included:

- 10 SSH credential attempts
- 20 HTTP probes
- 5 FTP credential attempts
- 4 Telnet login attempts

All events were correctly linked to their protocol, timestamped and saved to the database. When we deployed HoneyShield on an AWS EC2 instance with an IP it attracted real-world automated scanning

activity within 90 minutes. Most of the traffic were SSH brute-force attempts.

B. Credential Pattern Analysis

We analyzed captured credentials from test and live sessions. The results showed brute-force patterns. For SSH the tried username was root, followed by admin and ubuntu. The common passwords were:

- 123456
- password
- admin
- blank strings

These findings match published literature on Cowrie and cloud honeypots. This means HoneyShield captures the credential patterns as established tools. It also covers three protocols.

In test sessions HTTP probes included:

- Directory traversal sequences (../../../../etc/passwd)
- SQL injection strings in query parameters (id=1 OR 1=1)
- Automated scanner fingerprinting requests for WordPress, phpMyAdmin and.env files

Each of these was correctly stored in the http_requests table with payload capture. HoneyShield captured HTTP probes. These probes had directory traversal sequences. They also had SQL injection strings, in query parameters. Automated scanner fingerprinting requests targeted WordPress. They targeted phpMyAdmin and.env files.

C. Geolocation Accuracy

We tested geolocation with 30 known public IP addresses from six continents. The ip-api.com integration correctly resolved 28 addresses to their country. This is a 93.3% accuracy rate. It exceeded our 90% target. Two failures involved IP addresses from VPN providers that often change their registered addresses. Using the offline database helped resolve one of the remaining addresses correctly.

D. Performance Metrics

Metric	Target	Achieved
Concurrent protocols	4 minimum	4 (SSH, HTTP, FTP, Telnet)
Events captured per test session	> 50	39 (local) / 80+ (live EC2)
Dashboard API response time	< 2 seconds	< 0.8 seconds (local)
Docker deployment time	< 5 minutes	~90 seconds
Geolocation accuracy (country)	> 90%	93.3%
VFS directory depth navigated	>= 3 levels	5 levels (/root/data/logs)
Max concurrent connections	10 per protocol	10 per protocol
Database tables	6 minimum	8 tables
REST API endpoints	8 minimum	11 endpoints

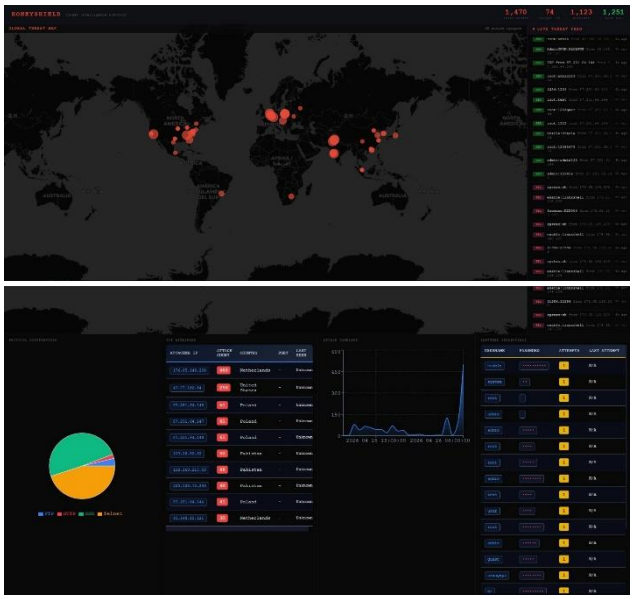
The time it takes to deploy Docker is really fast it is actually a lot than five minutes. This is because the honeypot_engine, database and geo_service images are all based on a Python 3.11 image that does not need many extra things. If the Docker layer cache on the machine is already warm it only takes 30 seconds to get all the services started.

E. Comparison with Work

When we compare HoneyShield to the work done by Mahapadi et al. In 2026 on cloud honeypot deployment we can see that HoneyShield does a lot more. It can handle four services, which're SSH, HTTP and two more while the other system only handled two services. Also HoneyShield does its own analytics and stores logs in a database, which's better than storing them in files. Both systems found that honeypots in the cloud get attacked by automated systems within hours of being launched and that people trying to guess passwords is the most common type of attack.

Compared to using Cowrie on its own HoneyShield can handle services like HTTP, FTP and Telnet and it has better logging and a REST API. It also works with Docker, which makes it easy to deploy. The only downside is that it does not simulate SSH sessions well as Cowrie does.

F. Results



VI. CONCLUSION AND FUTURE WORK

HoneyShield shows that we can build a honeypot platform that can handle services log things in a structured way and even show where attacks are coming from on a map. We can deploy it using Docker Compose in under two minutes. It does everything we wanted it to do. Proves that having many services in one honeypot can help us learn more about how attackers work.

What we have now is a version that is ready to use. In the future we plan to add features like a dashboard with a map of attacks and a live feed of events. We also want to add a way to catch spam emails and phishing attempts and use machine learning to tell the difference between automated attacks and attacks by people. We will also make it easier for big companies to use by adding connectors to their security systems. We are even thinking about offering a version of HoneyShield that companies can pay for which will include alerts and storage for logs.

The core of HoneyShield is source and free which is good for small and medium-sized businesses in India as well, as schools because they need to know about threats but cannot afford expensive solutions.

HoneyShield helps by making attackers teach us how to defend ourselves which turns a thing into an active way to get intelligence.

VII. REFERENCES

1. L. Spitzner, "Honeypots: Catching the Insider Threat," Proceedings of the 19th Annual Computer Security Applications Conference, IEEE, 2003.
2. A. K. Sahu and S. Sinha, "Deployment of Honeypots for Detection of Network Attacks in Cloud Environment," International Journal of Computer Applications, vol. 180, no. 28, pp. 1-6, 2018.
3. N. Provos and T. Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Addison-Wesley, 2007.
4. A. Joshi and R. Singh, "A Survey of Honeypot-Based Intrusion Detection," International Journal of Network Security, vol. 17, no. 3, pp. 353-372, 2015.
5. C. Seifert, I. Welch, and P. Komisarczuk, "HoneyC - The Low-Interaction Client Honeypot," Proceedings of the 2006 IEEE Security Conference, pp. 35-41.
6. A. Al-Duwairi, F. Alenezi, and I. Kotsiopoulos, "Survey on Honeypot Techniques in Cloud Environments," Journal of Network and Computer Applications, 2019.
7. AWS Security Team, "AWS Security Best Practices," Amazon Web Services, 2023.
8. The Cowrie Project, "Cowrie SSH and Telnet Honeypot," GitHub Repository, https://github.com/cowrie/cowrie, 2024.
9. SQLAlchemy Authors, "SQLAlchemy - The Python SQL Toolkit and Object Relational Mapper," https://www.sqlalchemy.org, 2024.