# Secure Solution Report: Honeypot Deployment and Analysis

# Introduction

Project Directions

The primary aim of this project was to deploy and analyze a honeypot to study cyber-attacks and malicious activities in a controlled environment. This involved setting up a honeypot to attract attackers, capturing their behavior, and analyzing the collected data to understand the methodologies and tools used by cybercriminals. The chosen challenge was to figure out the most effective strategies for deploying and managing honeypots to maximize their utility in detecting and preventing cyber threats

## Analysis of the Problem

The problem/opportunity is based on researching and implementing the previously mentioned best strategies. What constitutes a honeypot, and in what areas do we have room to interpretate and make design decisions? Before we can truly begin implementing a honeypot, we need to know what type of honeypot we want to create, how the logging of malicious activity is done, what kind of front we want to create to make our webapp more believable, etc..  
  
With these discussion points in mind, we've come up with the following research questions:

#### Main Research Question:

What are the most effective strategies for deploying and managing honeypots to maximize their utility in detecting and preventing cyber threats?

#### Sub-questions:

What are the needed components and requirements for creating a honeypot?

What are the different types of honeypots, and how do they differ in deployment and management?

How do we monitor and analyze honeypot data to identify and respond to security threats?

What is a viable network infrastructure to successfully mimic a company?

What are design conventions for online trading platforms that can be implemented in a honeypot for a more believable application?

#### Research Strategies Used

Our project consists of researching strategies and design conventions, and then iterating our design to improve our honeypot based on research and stakeholder feedback.  
Therefore, working with the DOT framework in mind, we mainly make use of the **Library** & **Workshop** strategies.  
  
**Library**: Investigated existing research papers, articles, and case studies on honeypots and their effectiveness.

**Workshop:** Set up a honeypot in a controlled network environment to capture live attack data. Implement and update solution flexibly based on stakeholder requirements and remarks.

#### Research Methods Used

**Library:**

* Available product analysis
* Best good and bad practices
* Design pattern research
* Literature study

**Workshop:**

* Brainstorm
* Decomposition
* Code review
* Prototyping
* Requirements prioritization

## Design Considerations

#### Sub-question 1: What are the needed components and requirements for creating a honeypot?

Creating an effective honeypot involves several key components and requirements:

1. **Hardware and Software Resources**:
   * **Hardware**: Depending on the scale of the honeypot, this can range from a single machine to a network of devices.
   * **Software**: This includes operating systems, virtualization software, and specific honeypot software such as Honeyd, Cowrie, or Dionaea.
2. **Network Configuration**:
   * **Isolation**: The honeypot should be isolated from the production network to prevent any potential compromise from spreading.
   * **Network Interface**: Proper configuration of network interfaces to handle incoming and outgoing traffic effectively.
3. **Security Measures**:
   * **Firewalls**: To control and monitor traffic to and from the honeypot.
   * **Intrusion Detection Systems (IDS)**: To alert administrators of any suspicious activity.
4. **Logging and Monitoring Tools**:
   * **Logging**: Comprehensive logging mechanisms to capture all activities within the honeypot.
   * **Monitoring**: Real-time monitoring tools to track interactions and gather intelligence.
5. **Data Storage and Analysis**:
   * **Storage Solutions**: Secure storage for logs and captured data.
   * **Analysis Tools**: Tools for analyzing collected data to identify patterns and potential threats.
6. **Deception Techniques**:
   * **Realistic Services**: Offering services that mimic those of a real environment (e.g., web servers, SSH).
   * **Vulnerability Simulation**: Introducing deliberate vulnerabilities to attract attackers.

#### Sub-question 2: What are the different types of honeypots, and how do they differ in deployment and management?

Honeypots can be categorized based on their interaction level and purpose:

1. **Interaction Levels**:
   * **Low-Interaction Honeypots**: Simulate only a few services or parts of an operating system. They are easier to deploy and manage but provide limited data. Examples include Honeyd and Glastopf.
   * **High-Interaction Honeypots**: Offer a full operating system with real services. They are more complex to manage but provide comprehensive data. Examples include Honeynet and Qebek.
2. **Purpose**:
   * **Research Honeypots**: Used primarily for studying attack methods and gathering intelligence. They often have extensive logging and analysis capabilities.
   * **Production Honeypots**: Deployed within an organization's network to detect and mitigate real-time threats. They are configured to interact with the organization's security infrastructure.

**Deployment and Management Differences**:

* **Complexity**: High-interaction honeypots require more resources and careful management to prevent them from being used as a launchpad for attacks.
* **Maintenance**: Regular updates and patches are crucial for high-interaction honeypots, whereas low-interaction ones need fewer updates.
* **Data Volume**: High-interaction honeypots generate more data, requiring robust storage and analysis solutions.

#### Sub-question 3: How do we monitor and analyze honeypot data to identify and respond to security threats?

Monitoring and analyzing honeypot data involves several steps:

1. **Data Collection**:
   * **Logging Tools**: Tools like ELK stack (Elasticsearch, Logstash, Kibana) to collect and visualize logs.
   * **Packet Captures**: Tools like Wireshark or tcpdump to capture and analyze network traffic.
2. **Data Analysis**:
   * **Automated Analysis**: Using machine learning and pattern recognition to identify suspicious activities.
   * **Manual Analysis**: Security experts reviewing logs and traffic patterns to pinpoint specific threats.
3. **Threat Identification**:
   * **Indicators of Compromise (IoCs)**: Identifying IoCs such as unusual login attempts, exploitation of vulnerabilities, or data exfiltration.
   * **Behavioral Analysis**: Understanding attacker behavior to detect sophisticated threats.
4. **Response Mechanisms**:
   * **Alerting Systems**: Automated alerts triggered by suspicious activities.
   * **Incident Response**: Procedures for isolating affected systems, gathering forensic data, and mitigating threats.

#### Sub-question 4: What is a viable network infrastructure to successfully mimic a company?

To successfully mimic a company, a honeypot’s network infrastructure should include:

1. **Segmentation**:
   * **Multiple Network Segments**: Mimicking different departments (e.g., HR, Finance) with appropriate services and resources.
   * **Virtual Local Area Networks (VLANs)**: To segment traffic and manage interactions.
2. **Diverse Services**:
   * **Common Services**: Web servers, email servers, file servers, and databases commonly used in businesses.
   * **Custom Applications**: Specific applications that reflect the company's industry and operations.
3. **Network Topology**:
   * **Realistic Topology**: Including internal and external facing servers, DMZ, and internal network zones.
   * **Traffic Simulation**: Generating legitimate-looking traffic to and from the honeypot to mimic normal business operations.
4. **Security Measures**:
   * **Firewalls and IDS**: To monitor and control traffic while collecting intelligence.
   * **Deception Techniques**: Using decoy data and services to lure attackers.
5. **Integration with Security Tools**:
   * **SIEM Integration**: Feeding honeypot data into Security Information and Event Management (SIEM) systems for comprehensive analysis.
   * **Incident Response Tools**: Ensuring quick action on detected threats.

#### Sub-question 5: What are design conventions for online trading platforms that can be implemented in a honeypot for a more believable application?

To create a believable honeypot that mimics an online trading platform, consider the following design conventions:

1. **User Interface**:
   * **Professional Look and Feel**: Using modern UI/UX design principles with intuitive navigation.
   * **Responsive Design**: Ensuring the platform works well on both desktop and mobile devices.
2. **Core Features**:
   * **Real-time Data**: Displaying live market data and trading options.
   * **User Authentication**: Implementing robust login and authentication mechanisms.
   * **Trading Functions**: Providing functionalities for buying, selling, and managing portfolios.
3. **Security Features**:
   * **Encryption**: Using HTTPS and secure communication protocols.
   * **Two-Factor Authentication**: Adding an extra layer of security for user logins.
4. **Compliance and Legal Disclaimers**:
   * **Terms and Conditions**: Clear terms of service and privacy policies.
   * **Regulatory Compliance**: Simulating compliance with relevant financial regulations.
5. **Backend Systems**:
   * **Transaction Logging**: Detailed logs of all trading activities.
   * **Data Storage**: Secure databases for user and transaction data.
   * **Analytics and Reporting**: Tools for analyzing trading patterns and user behavior.
6. **Deception Elements**:
   * **Fake Users and Transactions**: Simulated user accounts and transactions to create realistic activity.
   * **Dynamic Content**: Regular updates and changes to reflect a live trading environment.

#### Research Conclusion about the Value of the Outcomes for the Development of the Security Solution

This research into honeypot strategies has provided crucial insights that enhance our approach to cybersecurity. Here's what we've learned:

**Building Effective Defenses**: We now understand the essential components needed to create honeypots. This includes the right hardware, software, and network setup to mimic real environments while keeping threats isolated.

**Tailored Defense Strategies**: We've categorized honeypots into simpler (low-interaction) and more complex (high-interaction) types. This helps organizations choose the right approach based on their security needs and resources.

**Effective Threat Detection**: By using tools like SPLUNK and WiAZUH, we can monitor and analyze honeypot data effectively. This allows us to spot potential threats early and respond faster to protect our systems.

**Practical Application**: Incorporating design ideas from sectors like online trading platforms makes honeypots more believable to attackers. This ensures we gather accurate insights into their methods.

In conclusion, investing in honeypot technology isn't just about defense; it's a proactive step towards stronger cybersecurity. By deploying honeypots strategically and learning from their insights, we can better protect our systems and stay ahead of cyber threats.

### Chosen Technology and How It Works Web Application and Its Functionality

The web application in this project is built using React for the frontend and Node.js for the backend. This technology stack offers a modern and efficient way to develop dynamic web applications. The webapp was built using:

* **Frontend (React)**: React, a JavaScript library, builds the user interface with reusable components and efficient state management, providing a responsive and interactive experience.
* **Backend (Node.js)**: Node.js, a JavaScript runtime, handles server-side operations, processes requests, and manages communication between the frontend and the database, making the application scalable and efficient.

### **Security Vulnerabilities**

The web application intentionally lacks certain security measures to expose it to SQL injection attacks, allowing attackers to exploit the system and provide valuable data on attack techniques. Here are the specific vulnerabilities:

1. **No Input Sanitization**: The application does not sanitize user inputs, meaning that any data entered by users is directly used in SQL queries. This lack of sanitization makes the application susceptible to various forms of attacks, including SQL injection.
2. **No Parameterized Queries**: The application does not use parameterized queries to interact with the database. Parameterized queries are crucial for preventing SQL injection attacks because they separate SQL code from data. Without them, attackers can inject malicious SQL code through user inputs, potentially compromising the database.

### **Purpose and Benefits**

By intentionally incorporating these vulnerabilities, the project aims to attract attackers and study their methods. The insights gained from these attacks can help in understanding the following:

* **Attacker Techniques**: Detailed logs of attacker interactions with the honeypot and the web app provide valuable information on how attackers exploit vulnerabilities, what tools they use, and what kind of data they target.
* **Improving Security Measures**: Analyzing the attacks helps in developing better security practices and defenses. The knowledge gained can be used to enhance input sanitization methods, implement parameterized queries, and adopt other security best practices to protect against similar attacks in real-world applications.

#### Cowrie

Cowrie is an open-source honeypot solution designed to simulate an interactive SSH and Telnet environment. It captures the activities of attackers attempting to breach these systems, providing valuable insights into their techniques and behaviors. By mimicking a real system, Cowrie logs all interaction details, including commands executed and files transferred. This data is crucial for understanding attack patterns and developing defensive measures.

**Purpose and Benefits:** Cowrie's primary purpose is to gather intelligence on attackers' methods and strategies by attracting and monitoring malicious activity. The benefits include:

* **Insight into Attack Tactics:** Detailed logs and session recordings offer insights into how attackers attempt to compromise systems, helping security teams understand vulnerabilities and improve defenses.
* **Enhanced Security Posture:** By studying attacker behavior, organizations can refine incident response plans, strengthen network configurations, and implement targeted security measures to better protect against real-world threats.

#### Splunk

Splunk is a powerful data analysis platform that excels in indexing and visualizing machine-generated data. It aggregates logs from various sources, including servers, network devices, and applications, providing real-time insights and alerts. Splunk's capabilities allow for deep searches, correlations, and reporting, making it an indispensable tool for monitoring security events and detecting anomalies. It helps organizations respond quickly to incidents and refine their security strategies.

**Purpose and Benefits:** Splunk serves the purpose of centralizing and analyzing logs to enhance security monitoring and operational intelligence. The benefits include:

* **Real-time Visibility:** Splunk enables real-time monitoring of security events across the entire infrastructure, facilitating early detection and response to potential threats.
* **Comprehensive Analysis:** By correlating data from different sources, Splunk provides holistic insights into security incidents, facilitating root cause analysis and proactive threat mitigation.
* **Operational Efficiency:** Automated alerting and reporting streamline incident response processes, improving overall security posture and reducing time to resolution.

#### Wazuh

Wazuh is an open-source security monitoring platform that integrates with the Elastic Stack for comprehensive threat detection and response. It collects and analyzes data from various endpoints, providing real-time visibility into potential security issues. Wazuh's capabilities include log analysis, file integrity monitoring, vulnerability detection, and compliance management. By continuously monitoring and analyzing system behavior, Wazuh helps in identifying and mitigating threats effectively.

**Purpose and Benefits:** Wazuh aims to enhance security monitoring and incident response capabilities across diverse environments. Its benefits include:

* **Multi-platform Support:** Wazuh can monitor and analyze security data from endpoints, servers, and cloud environments, offering centralized visibility and control.
* **Threat Detection:** Advanced correlation rules and anomaly detection capabilities enable early detection of suspicious activities and potential security breaches.
* **Compliance and Risk Management:** Wazuh helps organizations maintain compliance with security standards and regulations through continuous monitoring and auditing.

#### Suricata

Suricata is an advanced, open-source network threat detection engine capable of performing real-time intrusion detection (IDS), intrusion prevention (IPS), and network security monitoring (NSM). It inspects network traffic using a combination of signature, protocol, and anomaly-based inspection methods. Suricata generates detailed logs and alerts for network events, helping security teams identify and respond to suspicious activities. Its ability to handle high-throughput environments makes it a valuable asset in securing network infrastructures.

**Purpose and Benefits:** Suricata aims to protect network infrastructures by detecting and preventing malicious activities. Its benefits include:

* **High Performance:** Suricata's multi-threaded architecture and efficient packet processing enable real-time monitoring and analysis of network traffic without impacting performance.
* **Customizable Rulesets:** Organizations can create and deploy custom detection rules tailored to their specific network environments and security requirements.
* **Threat Intelligence Integration:** Integration with threat intelligence feeds enhances detection capabilities by leveraging up-to-date information on emerging threats and attack vectors.

#### Apache2 Configured as a Reverse Proxy

Apache HTTP Server (Apache2) configured as a reverse proxy acts as an intermediary between clients and backend servers. It receives requests from clients and forwards them to the appropriate backend server, masking the identity and characteristics of the origin server. This setup enhances security by hiding backend server details and enabling centralized access control and monitoring. Apache2's reverse proxy capabilities are essential for load balancing, caching, and protecting backend servers from direct exposure to the internet, thereby improving overall system performance and security posture.

**Purpose and Benefits:** The purpose of Apache2 as a reverse proxy is to improve security, scalability, and performance of web applications and services. Its benefits include:

* **Security Enhancement:** Apache2 hides backend server details, reducing the attack surface and protecting against direct exploits.
* **Load Balancing:** Distributes incoming client requests across multiple backend servers, optimizing resource utilization and improving responsiveness.
* **Centralized Access Control:** Enables uniform access policies and authentication mechanisms for multiple backend services, enhancing overall security management and compliance.

#### Security Aspects Involved

**Containment and Isolation**

Network segmentation: Honeypots should be isolated from the rest of the network to prevent attackers from escaping using the honeypot's internal network as a gateway to attack other unintended systems.

Firewall rules: Strict firewall rules should be implemented to control traffic from honeypot to honeypot. They are severely restricting communication and only allowing necessary communication.

Logging and Monitoring: Real-time detailed monitoring is necessary for capturing attack data. Logs should be securely collected and always stored

Data analysis: When it comes to viewing said logs there should be tools to look at the attacks attempted that can review the attack in detail (e.g.: time of the attack) Backup and Recovery: Regular backups should be made of the honeypot itself and configuration

involved to enable quick recovery in case of a system compromise or failure.Build and Implement Documentation for the Secure Solution  
  
**Explanation of How the Web App Was Made**

The web application backend is built using Node.js with Express, utilizing MySQL for the database, JWT for authentication, and follows a modular architecture for better maintainability.

#### **Step 1: Setting Up the Project**

1. **Initialize the Project**:
   * Create a new Node.js project and initialize it with npm init.
   * Install necessary dependencies: express, cors, mysql2, md5, jsonwebtoken, and dotenv.
2. **Project Structure**:
   * Create a project structure with folders for routes, services, middleware, and configuration files.

#### **Step 2: Setting Up the Server**

1. **Main Server File (main.js)**:
   * Import required modules: express, cors, ./db, ./routes/auth, and ./middleware/authMiddleware.
   * Create an Express app and configure it to use JSON and CORS middleware.
   * Connect to the MySQL database using db.connect().
   * Define routes: Unprotected /auth routes and a protected /hi route.
   * Add error handling for 404 and other server errors.
   * Start the server on the specified port.

#### **Step 3: Database Configuration**

1. **Database Connection (db.js)**:
   * Use mysql2 to create a connection to the MySQL database.
   * Export the database connection for use in other parts of the application.

#### **Step 4: Authentication Service**

1. **AuthService (authService.js)**:
   * Import necessary modules: md5, ../db, jsonwebtoken, and dotenv.
   * Define login and register functions:
     + **Login**: Validate user credentials, generate a JWT token on successful login.
     + **Register**: Check for existing users, hash the password using MD5, and insert a new user into the database.
   * Export the functions for use in routes.

#### **Step 5: Authentication Routes**

1. **Auth Routes (routes/auth.js)**:
   * Import necessary modules: express and ../services/authService.
   * Define validation functions for email and required fields.
   * Create POST routes for /register and /login:
     + **Register**: Validate input, call the register function, and handle errors.
     + **Login**: Validate input, call the login function, and handle errors.
   * Export the router for use in the main server file.

#### **Step 6: Authentication Middleware**

1. **JWT Authentication Middleware (middleware/authMiddleware.js)**:
   * Import jsonwebtoken and dotenv.
   * Define authenticateJWT middleware:
     + Extract the token from the Authorization header.
     + Verify the token using the JWT secret.
     + Attach the user information to the request object on successful verification.
     + Handle unauthorized access.
   * Export the middleware for use in protected routes.

#### **Step 7: Environment Variables**

1. **Environment Configuration (.env file)**:
   * Store sensitive information like the JWT secret and database credentials in a .env file.
   * Use dotenv to load these variables into the application.

### **Explanation of How the Front-End Web App Was Made**

The front-end of the web application is built using React, with React Router for routing, Axios for API calls, and a context-based state management for authentication. Below is a step-by-step explanation of how it was created:

#### **Step 1: Setting Up the Project**

1. **Initialize the Project**:
   * Create a new React project using Create React App or a similar setup.
   * Install necessary dependencies: react-router-dom, axios, tailwindcss for styling.
2. **Project Structure**:
   * Create a project structure with folders for components, pages, context, and API.

#### **Step 2: Routing and Main Application**

1. **Main Application File (App.jsx)**:
   * Import necessary modules: react-router-dom, AuthProvider, Nav, and pages (Home, Prices, Login, Register).
   * Wrap the application in AuthProvider for context-based state management.
   * Define routes using Routes and Route from react-router-dom.

Step 3: Authentication Context

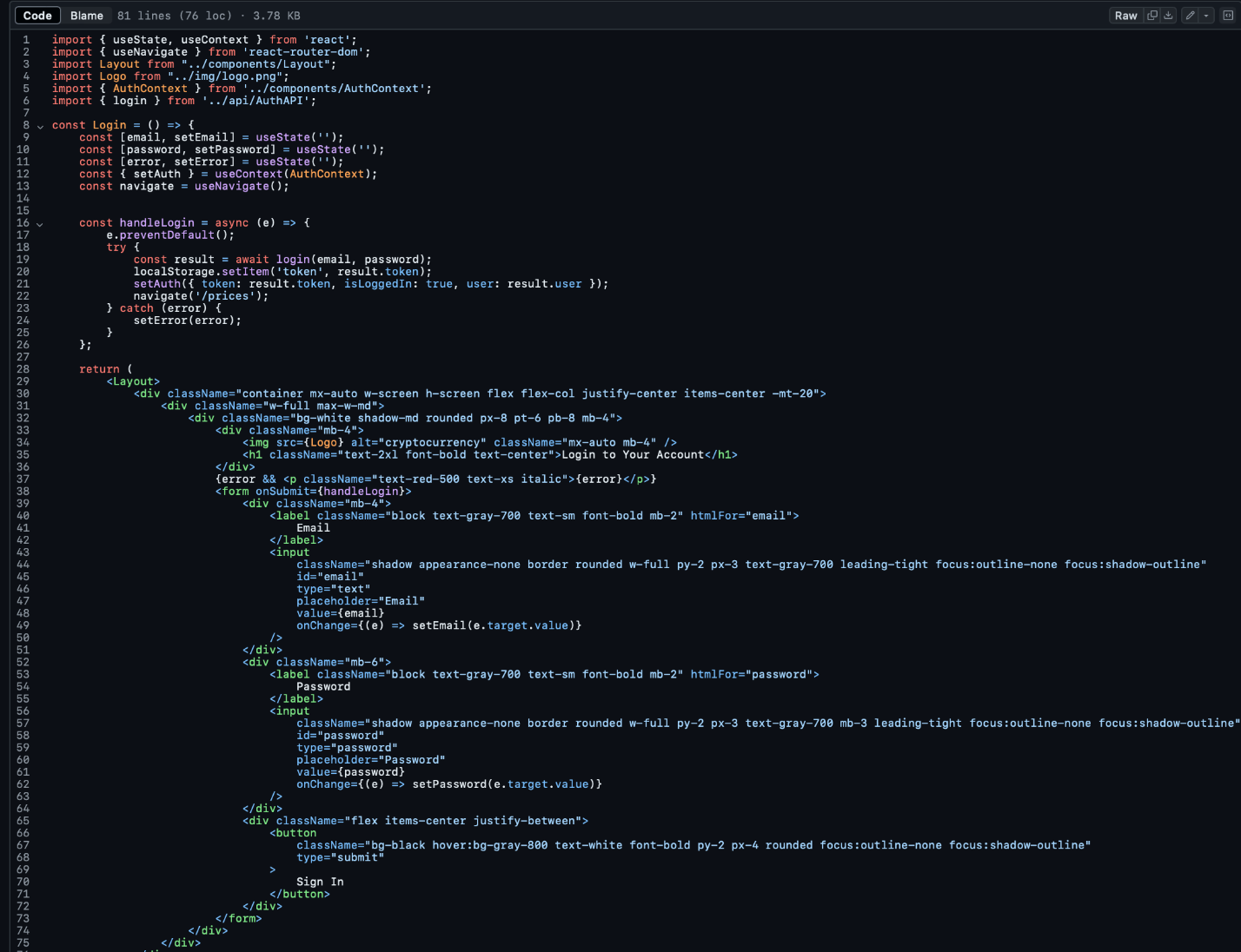
* + Create a context for managing authentication state.
  + Provide methods for setting and getting authentication data.

#### **Step 4: API Interaction**

1. **Auth API (AuthAPI.jsx)**:
   * Create functions to handle login and registration API calls using Axios.
   * Store the token and user information in local storage upon successful login or registration.

#### **Step 5: Login and Registration Pages ( only Login as example, since they are the same )**

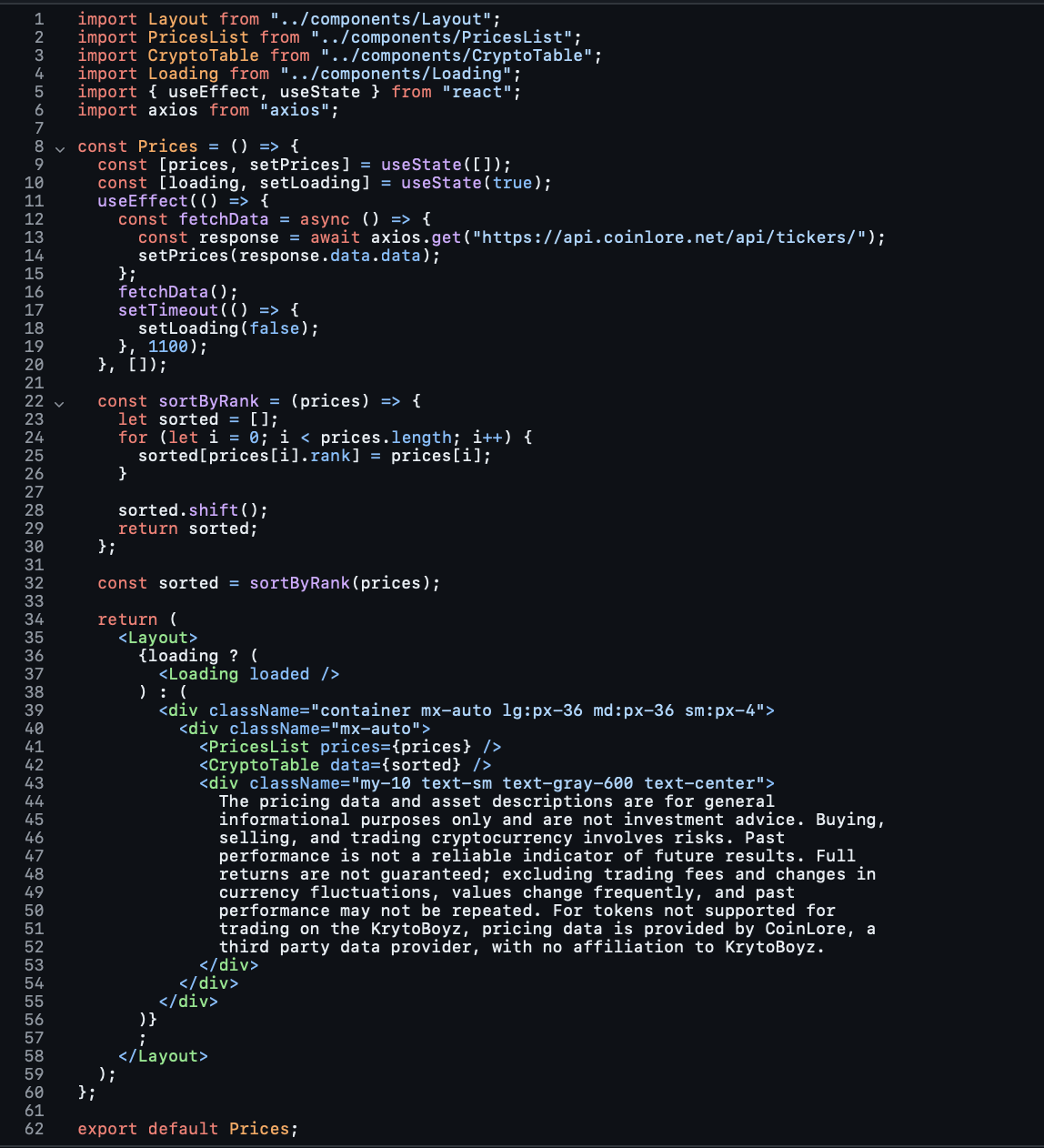
1. **Login Page (Login.jsx)**:
   * Create a form for user login.
   * Use context to set authentication state upon successful login.
   * Handle form submission and error display.



1. **Register Page (Register.jsx)**:
   * Similar to the login page, create a form for user registration.
   * Use context to set authentication state upon successful registration.

#### **Step 6: Protected Routes and Loading State**

1. **Prices Page (Prices.jsx)**:
   * Fetch cryptocurrency prices from an API.
   * Use state to manage loading and data.
   * Display prices using components like PricesList and CryptoTable.



We just do a simple sort on the prices in order to have them by rank ( ex. Bitcoin first )  
  
Reflection

I would say the hardest part of making this web application is making it unsecure. Most of the libraries are foolproof and they stop some behavior that should not be allowed (like sqli ), so we had to get clever and use outdated libraries.

Conclusion

By following these steps, the web application achieves a modular structure with clear separation of concerns. This makes it easier to maintain and extend in the future. Each part of the application handles specific responsibilities, ensuring a clean and efficient codebase.

### Installation of Cowrie:

#### Prerequisites

* **Linux OS**: Cowrie runs best on Linux. Ubuntu or Debian-based distributions are recommended.
* **Python**: Cowrie requires Python 3.6 or later.
* **Root Access**: You need administrative privileges to install packages.

#### 2. Installation

##### Step 1: Install Dependencies

Ensure your system is up-to-date and has necessary packages:

sudo apt update

sudo apt upgrade

sudo apt install python3 python3-pip virtualenv python3-virtualenv git

##### Step 2: Create a Virtual Environment

It's good practice to isolate Cowrie’s dependencies using a virtual environment:

mkdir cowrie

cd cowrie

python3 -m virtualenv cowrie-env

source cowrie-env/bin/activate

##### Step 3: Clone Cowrie from GitHub

Clone the Cowrie repository from GitHub:

git clone http://github.com/cowrie/cowrie

cd cowrie

##### Step 4: Install Cowrie

Install Cowrie and its dependencies:

pip install -r requirements.txt

#### 3. Configuration

##### Step 1: Customize the Filesystem

You may want to customize fake filesystem structures. Modify etc/userdb.example for user credentials, etc/issue for SSH banner, etc.

#### 4. Testing

Verify that Cowrie is running by attempting to connect to it via SSH or Telnet on the specified port. Check logs (var/log/cowrie/\*) for captured events.

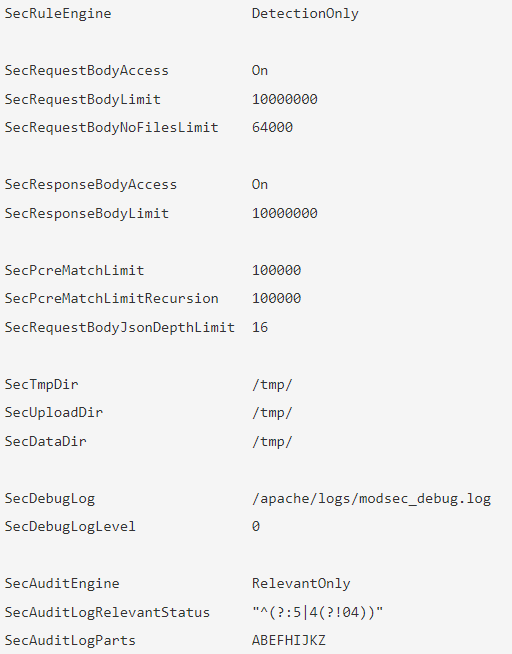
#### 5. Advanced Configuration

For advanced customization and features (like email alerting, logging to external databases), refer to Cowrie's documentation and modify etc/cowrie.cfg accordingly.

#### 6. Security Considerations

* **Firewall**: Ensure ports used by Cowrie are not exposed to unauthorized access.
* **Regular Updates**: Keep Cowrie and its dependencies up-to-date to mitigate security risks.

Or Run the installation script from ansible



1. SecRuleEngine (DetectionOnly)

- Change this to “DetectionOnly”, so that it doesn't stop any attacks, but only detects logs and.

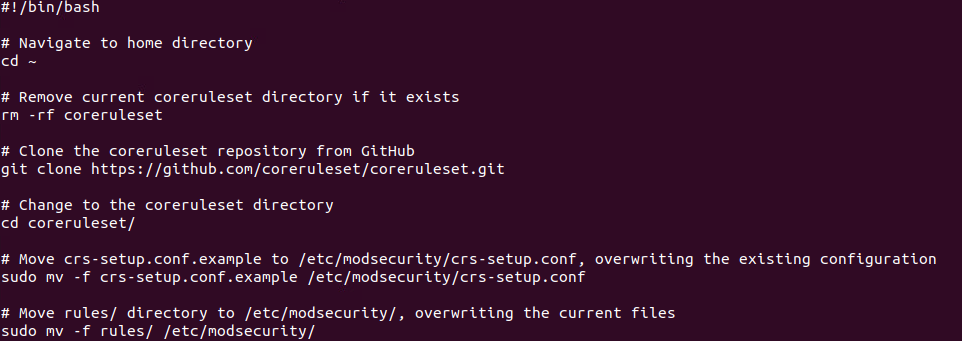
1. SecAuditEngine (RelevantOnly)

- Change this to “RelevantOnly”, so that it doesn't send all the logs but only the relevant logs included in the core ruleset.

1. SecAuditLogParts (ABEFHIJKZ)

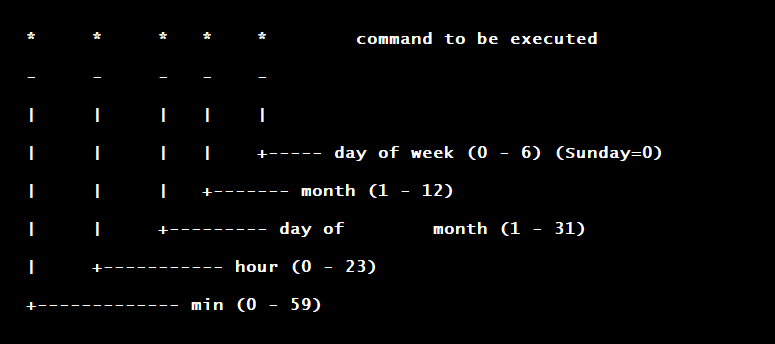
- We use “ABEFHIJKZ”, this doesn't send every single detail about the attack, this however is possible by adding ‘C’ and 'D’ to the already used alphabetic letters.

Cron Tab:

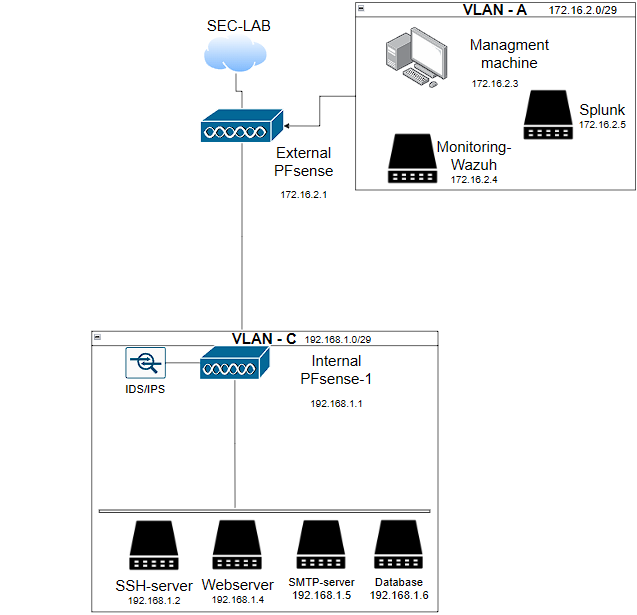


This script removes the current core ruleset because you'll get errors or prompts to overwrite it if it already exists. Then we clone the newest core ruleset and move the ruleset to the existing Mod security Web Application Firewall configuration. We then also move the rules to the Web Application Firewall.

We then create a new crontab that runs every day at 00:00 at midnight.

How crontab configuration works:  


**Network Configuration:**

For the honeypot we’ve setup our network as pictured below

The internal network receives the internet connection through the external pfsense. Any outside requests will have to go through that network.

Vlan-A is configured for 172.16.2.1 with a prefix of 29 (255.255.255.248)

Both pfsenses webconfigurators are configured behind TCP port 6615 and are being forwarded so they can only be managed from the management machine and nowhere else

On both pfsenses suricata is actively running a set of custom rules but also some default rules imported from a snort set.

Pfsense-External Nat

A screenshot of a computer

Description automatically generated

1. Port forwarding for the webserver nodejs application
2. Port forwarding for telnet connections from outside to internally (cowrie)
3. Port forwarding for SMTP connections from outside to internally (cowrie)
4. Port forwarding for HTTP connections from outside to internally (cowrie)
5. Port forwarding for ssh connections from outside to internally (cowrie)
6. Port forwarding for actual SSH connection into the machine (not the honeypot)

Pfsense-Internal Nat

A screenshot of a computer

Description automatically generated

1. Firewall rule to port forward communication from outside to the website api endpoint
2. Portfowarding the pfsenses webconfigurator
3. Redirecting traffic from outside to our webserver
4. Redirecting ssh traffic from port 22 to 2222
5. Our actual ssh connection from port 2400

Pfsense-Internal LAN

A screenshot of a computer

Description automatically generated

1. Allow communication wazuh on port 1515
2. Allow communication wazuh on port 1514
3. Blocking firewall ICMP ping
4. Blocking firewall access for the lan side
5. Blocking internal pfsense pinging
6. Blocking external pfsense pinging

Pfsense-Internal + Pfsense-External suricata rules NMAP+SQL INJECTION+ path traversal attempts

A screenshot of a computer

Description automatically generated

#### Monitoring Setup:

1. Stack Forwarding:
   * Wazuh
   * Mod Security/WAF
   * Splunk

Firstly, wazuh must be installed on a machine that will be used for the purpose of managing all the agents. An all-in-one installation must be performed. Before the installation you’ll need to configure the wazuh config file to change the indexer/server/dashboard ip

A screenshot of a computer

Description automatically generated

The wazuh installation should be run as below to install all the features

A black background with white text

Description automatically generated

After the installation agents must be installed onto the machines you wish to monitor. This is quite straightforward as the wazuh dashboard clearly explains on how to install agents.

Setting up WAF/modsec can be done following the guide here: <https://www.inmotionhosting.com/support/server/apache/install-modsecurity-apache-module/>

Logs can be forwarded by adding the machine into a group in wazuh management > groups and edit the group configuration to forward the logs from /var/log/apache2/modsec\_audit.log

In order to install the wazuh agent on pfsense you’ll need to enable the FreeBSD package

Start by editing the file nano /usr/local/etc/pkg/repos/pfSense.conf and setting FREEBSD to yes

Set the repo also to yes nano /usr/local/etc/pkg/repos/FreeBSD.conf

Proceed with the following commands

Pkg update && Pkg search wazuh agent && pkg install wazuh-agent-x.xx.x

Following installation of the agent, you'll see some output on configuring your agent.

Now, edit the /var/ossec/etc/ossec.conf file and point it to your Wazuh manager:

Now, start the Wazuh agent and enable start at boot by creating a syslink <https://docs.netgate.com/pfsense/en/latest/development/boot-commands.html?ref=benheater.com#shell-script-option>

Now pfsense does ship logs but it does not ship Suricata logs for IDS/IPS monitoring yet. Go into pfsense and go to services/Suricata and edit the interface that you want to forward and enable the following like below and restart the configuratorA screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generatedGo into wazuh manager and go into management > groups and create a new group and name it pfsense and add in the pfsense agent into that group. Edit the group configuration and add the following snippet. Suricata logs should start to showup

Example screenshot of sql injection suricata log being forwarded to wazuh

A screenshot of a computer

Description automatically generated

When it comes to splunk you’ll need to go signup and make an enterprise account on splunk.com. This will grant you access to the download of splunk. Run through the splunk installation and configure your credentials.

A screenshot of a computer

Description automatically generated

After configuring your credentials go to your machines ip:port:8000 to access splunk

A screenshot of a computer

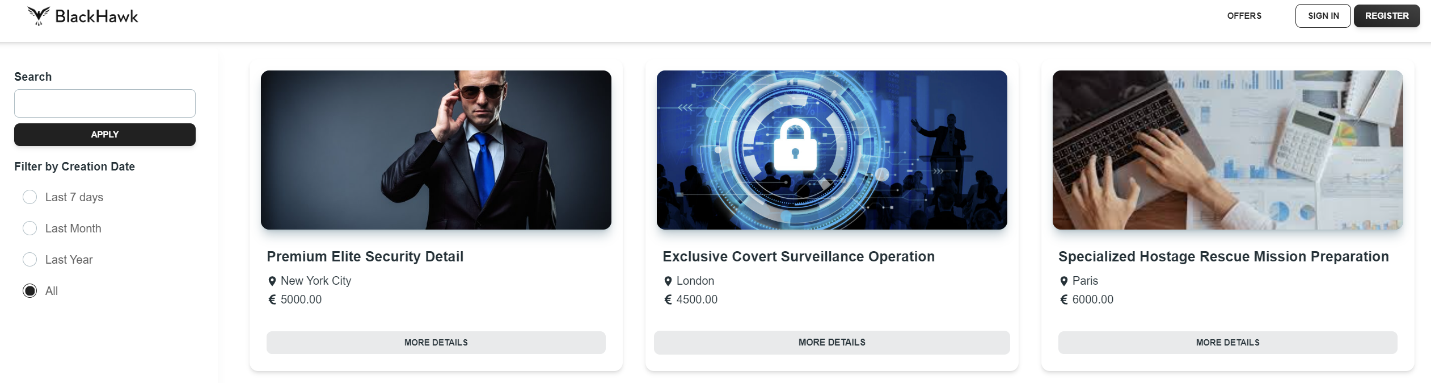
Description automatically generated

Download the splunk forwarder on machines that you wish to monitor. After installing the forwarder. Go to splunk and go to forwarding and receiving and add a new listener port on 9997. Then call the binary /opt/splunkforwarder/bin/splunk add forward-server “splunkip” to forward the logs to the splunk server. You can also configure them to be sent to a specific source.

### Test Results for the Secure Solutions (Red versus Blue)

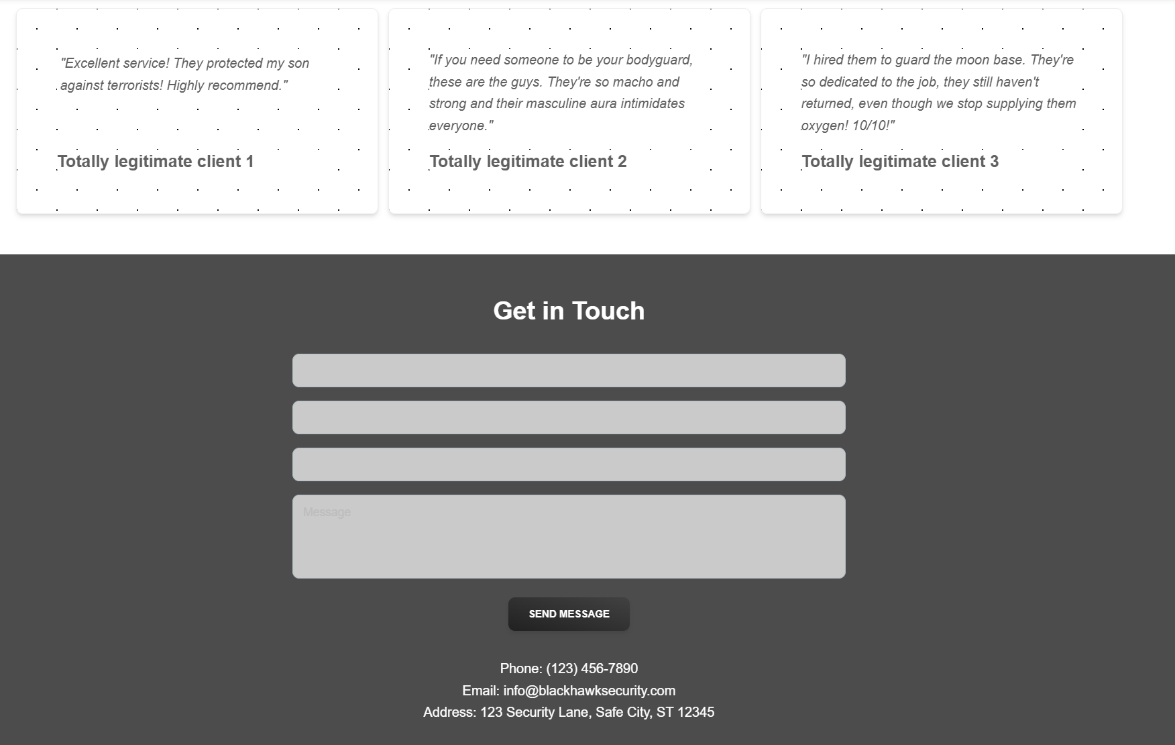
**SQL Injection (SQLi):**

We checked out most of their website and where we possibly could use SQL injection. We then found this page:



We could search for specific offers. But whatever we tried, nothing worked. The moment we used an OR statement, it wouldn't work. It was only searching for the exact wording of the offer.

**Cross-Site Scripting (XSS):**

When searching the website, we found a page, where we possibly could use XSS:

We were curious if they had any filters, but as expected they filter everything. From empty spaces to every single special character we could possibly use. We also tried to search inside the source code if they have anything sanitized about errors and found the following:



They sanitize every single possibility when using errors. This way we could use errors but nothing to trigger it.

**Path Traversal:**

Path traversal attacks, which could allow unauthorized access to server files and directories, were also tested. We tested every single possibility but like the XSS sanitation, the website also sanitized everything for path traversal such as empty spaces and special characters:  


**Scanning and Enumeration:**

We also tried scanning and enumerating, to identify any possible security gaps or find information about the system they use. But everything was blocked by Microsoft's Web Application Firewall (WAF). Every single scan came back empty at a very quick rate. They even blocked WAF scans with software such as Wafw00f. Now we see how important a WAF can be. With just the use of that WAF we couldn't find any single point of security threat or even information about the software or hardware they were using.

**Conclusion Red vs Blue:**

The team that made BlackHawkSec.nl did an extremely well job in securing their environment. While they didn't do everything knowingly, such as blocking WAF scans, the way they set up their environment and combined their infrastructure to Azure and doing everything securely is a feat. It was also interesting to see what a perfectly protected system would look like. I have never had any problems with scans, but this time every single scan was blocked. It was a good lesson from an attack and defense perspective.

### Conclusions Towards the Original Goals and Problem that Had to Be Solved

The project successfully showed us detecting and analyzing cyber-attacks with the usage of Honeypots. By employing a honeypot, we were able to gather detailed information on attackers' tactics, providing insights into potential vulnerabilities. This data can help better our network security by informing better defensive strategies and improving overall cyber resilience.

Our main research question was: What are the most effective strategies for deploying and managing honeypots to maximize their utility in detecting and preventing cyber threats? To address this, we explored the components and requirements for creating a honeypot, the different types of honeypots and the methods for monitoring and analyzing honeypot data to identify and respond to security threats. Additionally, we investigated how to design a network infrastructure to mimic a company.

A honeypot requires hardware and software resources, including operating systems, virtualization tools, and specific honeypot software like Cowrie, or Dionaea. Network configuration is crucial, this involves isolating it from the production network and correct setup of network interfaces. Security measures such as firewalls, intrusion detection systems (IDS), and logging and monitoring tools are vital. Data storage and analysis solutions are necessary for securing logs and analyzing data to identify patterns and threats.

We explored different types of honeypots, including low-interaction honeypots that simulate limited services. Low-interaction honeypots are easier to manage but provide limited data, while high-interaction honeypots are more complex but provide comprehensive data.

Our research strategies included both library and workshop approaches. We investigated existing research papers and case studies and set up a honeypot in a controlled environment to capture live attack data. This combined approach helped us finding out the right strategy. In the time that our honeypot was live (3 weeks) we’ve gotten many results from it. We were able to see if humans or robots were attacking us and what they were trying. We were able to properly monitor everything they did and secured ourselves against any threats. So we believe that we have found the most effective strategy to deploy and manage a honeypot successfully and being able to detect and prevent cyberattacks.

### References

<http://www.it-docs.net/ddata/792.pdf>

<http://www.citi.umich.edu/techreports/reports/citi-tr-03-1.pdf>

<https://cowrie.readthedocs.io/en/latest/README.html>

<https://documentation.wazuh.com/current/index.html>

<https://docs.splunk.com/Documentation>

<https://www.honeynet.org/>

<https://community.splunk.com/t5/All-Apps-and-Add-ons/How-to-deploy-a-HoneyPot-using-Splunk/m-p/484261>

<https://github.com/paralax/awesome-honeypots>

<https://owasp.org/www-project-top-ten/>