**ICS344 Project**

**Semester 241 – w/ dr. Waleed Algobi**

Team Members

Mohammed aljaafari 201927590

Omar Alomran 202017840

**General overview**

Node.js is a popular tool used for building online applications.

Reasons for focusing on it:

- Common Use: Many online apps use Node.js, making it attractive for attackers.

- Weaknesses: Node.js apps often have security risks because they use many third-party packages, which makes them easy targets for attacks.

- Importance: Looking at how Node.js works in real life helps us understand the challenges of keeping apps safe.

Targeted Honeypots: Crowi and Dionaea

- Crowi:

- What it does: Looks like a real web app to test how well attacks like SQL Injection, Command Injection, and XSS can be detected and managed.

- Realism: Acts like real web services, allowing good comparisons with another tool called Node.js.

- Dionaea:

- What it does: Focuses on catching malware and other harmful software, helping to see patterns in attacks.

- Efficiency: Uses less computer power, making it good for tracking attacks on a large scale.

- Support Role: Not as realistic as Crowi, but keeps detailed records of attacks, which helps analyze them better.

SIEM: Wazuh

- Why We Picked Wazuh:

- Lots of Features: Wazuh can gather and check logs, which helps us look at real services and fake ones (honeypots).

- Free to Use: Since it's open-source, we can set it up the way we want without spending too much money.

- Works Well with Others: Wazuh connects easily with honeypots like Crowi and Dionaea and lets us monitor application logs from Node.js in real-time.

- Easy to Understand: It has strong dashboards and alerts that help us see patterns of attacks, how resources are used, and common behaviors across services.

**Phase 1**

Configuration Setup   
  
Caldera Setup:   
  
I installed Caldera on Kali Linux by duplicating its repository from GitHub. The first step was to verify that all necessary dependencies, such as go and python3, were correctly installed.   
The incorporation of SIEM and Honeypot: I implemented the integration of a Honeypot, such as Cowrie or Dionaea, to replicate attack traffic, while employing a SIEM platform such as the ELK stack (consisting of Elasticsearch, Logstash, and Kibana) for the purpose of logging and monitoring activities related to attacks. These elements were set up to capture and monitor logs and notifications from the Honeypot in order to identify and respond to potential exploitation attempts.   
Tactics, Techniques, and Procedures Utilized:   
I developed tailored Tactics, Techniques, and Procedures (TTPs) in Caldera specifically designed to exploit weaknesses within the security of the Juice Shop. This encompassed the utilization of pre-existing exploitation methods, such as SQL injection, command injection, and XSS, in order to obtain entry into the application and its underlying systems.   
The TTPs were updated to incorporate MITRE ATT&CK Techniques T1071 (Application Layer Protocol) and T1203 (Exploitation for Client Execution). The automation necessary to carry out these techniques was provided by Caldera.   
  
The use of Kali tools and customized scripts is essential in the field of cybersecurity for conducting penetration testing   
  
The Metasploit Framework was utilized to automate the exploitation of vulnerabilities such as remote code execution (RCE) and command injection within the Juice Shop. Specialized scripts were developed to carry out automated attacks, such as SQL injection, by capitalizing on vulnerable points within the application.   
I utilized nmap to conduct a scan for accessible ports, burpsuite for intercepting web traffic, and curl to manually assess HTTP endpoints.   
These elements were incorporated into Caldera through personalized commands that were initiated during the attack sequence in order to automate the process of exploitation.

This is my own script that I used

curl -X POST http://localhost:3000/api/Products \

-d "name=<script>alert('XSS')</script>" \

-H "Content-Type: application/json"

curl -X POST http://localhost:3000/api/vulnerabilities/command-injection \

-d "input=$(whoami)" \

-H "Content-Type: application/json"

curl -X GET "http://localhost:3000/api/Products?category\_id=1' OR '1'='1"  
  
2. Choosing a Service   
  
Services aimed at achieving a specific goal.   
  
The primary focus of this project was on Juice Shop, a web application specifically designed to imitate prevalent security vulnerabilities found in common applications.   
The Juice Shop was chosen due to its widespread recognition and deliberate vulnerabilities, which offer an excellent platform for learning and practicing penetration testing techniques The process of conducting tests and red team exercises. The application exhibits vulnerabilities including SQL injection, XSS, and command injection, making it suitable for evaluating attack tools such as Caldera.

3. Difficulties and Flaws

Challenges Faced:

The default configuration of Caldera and the Juice Shop application displayed compatibility issues, particularly when attempting to execute specific payloads.

The absence of necessary dependencies presented a challenge during the installation of Caldera. Specifically, there were issues pertaining to the absence of certain Python packages and incorrect versions of Go, which required manual updating.

The intricacy of custom scripts: The development of custom scripts for executing sophisticated attacks such as exploiting XSS or RCE necessitated thorough testing due to frequent failures in payload execution.

Conquering Challenges:

In order to mitigate dependence concerns, I verified the accurate installation of all dependencies by utilizing tools such as pip and apt-get for updating Python and Go.

I manually debugged custom script execution by using curl and Postman to ensure that the payloads were functioning properly outside of Caldera, before incorporating them into the Tactics, Techniques, and Procedures (TTP).

4. Optimal methods and guidance

Optimal methods:

Utilize Automated Testing: Employ automation whenever feasible, as Caldera enables the automatic implementation of TTPs, thereby expediting the process and minimizing potential human errors.

Prior to integrating custom scripts into bigger frameworks such as Caldera, it is essential to test them independently to ensure their functionality and compatibility.

Utilize official documentation when working with tools such as Caldera, Kali tools, and Juice Shop in order to reduce the likelihood of making configuration errors.

Feedback on project

Insights gained:

The completion of the project has provided me with the knowledge and skills necessary to utilize Caldera for automated red teaming, as well as to incorporate Kali Linux tools for exploitation. I acquired a more thorough understanding of attack vectors such as SQL injection and Remote Code Execution (RCE) in the context of web applications.

I gained knowledge in setting up a Security Information and Event Management (SIEM) platform to identify and counteract simulated cyber attacks, a skill important for defending against security threats as part of the blue team.

Do I suggest incorporating this for upcoming course cycles?

Certainly, this undertaking offers practical involvement in both red and blue team activities, thus enhancing its educational significance.

6. Educational Materials

The official documentation for Caldera was consulted in order to gain insight into the configuration and creation of TTPs.

The official GitHub and documentation of OWASP Juice Shop were instrumental in enhancing my comprehension of prevalent vulnerabilities.

The Kali Linux Documentation provides comprehensive resources for acquiring knowledge on utilizing a range of Kali tools such as Metasploit, Nmap, and Burpsuite.

I utilized the official MITRE ATT&CK website to categorize the attack techniques that were employed.

7. Details of the Attack (Including Tools, Tactics, and Procedures Used)

The efficacy and rate of achievement:

Caldera was adept at automatizing recurring duties such as SQL injection and RCE attacks, and demonstrated a high degree of effectiveness in exploiting vulnerabilities in Juice Shop.

Metasploit proved to be highly efficient in exploiting certain vulnerabilities, particularly those that necessitated intricate sequences of exploitation.

The custom scripts I authored proved to be efficacious, albeit necessitating additional debugging and refinement in order to be seamlessly integrated into the Caldera system.

The convenience of use and automation:

Caldera offers advanced automation, making it the most user-friendly approach for exploiting vulnerabilities. It decreased the need for manual labor but necessitated an initial configuration process.

Utilizing Kali tools such as Metasploit and Burpsuite demands a greater degree of manual effort and a deeper understanding of exploitation methodologies.

Time and exertion:

The use of Caldera proved to be more efficient through the implementation of automation, however, setting up Metasploit for targeted attacks required a longer time investment.

The development of custom scripts required the most time as a result of the need for debugging and confirmation of their intended functionality.

Adaptability and innovation:

Caldera's capacity for devising innovative approaches was constrained, but it was well-suited for carrying out predetermined offensive strategies.

The utilization of personalized scripts enabled me to execute innovative attacks that may not be readily facilitated by preexisting software tools.

Discovery and Invisibility:

The Metasploit attack exhibited increased levels of noise, thus raising the likelihood of its detection by a SIEM or honeypot system.

Caldera's stealthy execution was achieved through the use of automation that closely emulated tactics employed by real-world adversaries.

Conformity with the MITRE ATT&CK Framework:

All methodologies were consistent with the MITRE ATT&CK framework, incorporating tactics such as T1071 (Application Layer Protocol) and T1203 (Exploitation for Client Execution).

The utilization of custom Tactics, Techniques, and Procedures (TTPs) facilitated my acquisition of a more comprehensive insight into the operational methods of actual adversaries.

Effect on the Object System:

Certain types of attacks, such as command injection, resulted in temporary disturbance to the functioning of the Juice Shop application; however, there was no lasting harm incurred.

Future Utilization and Enhancement:

In the context of future red-teaming engagements, the automation capabilities of Caldera make it the preferred method. Nevertheless, I would enhance its capabilities by incorporating additional options for creating personalized attack techniques.

I aim to enhance my custom scripts by incorporating more comprehensive error handling and optimizing the payloads for increased efficacy.

Snapshots

A screenshot of a computer

Description automatically generated

A screenshot of a computer

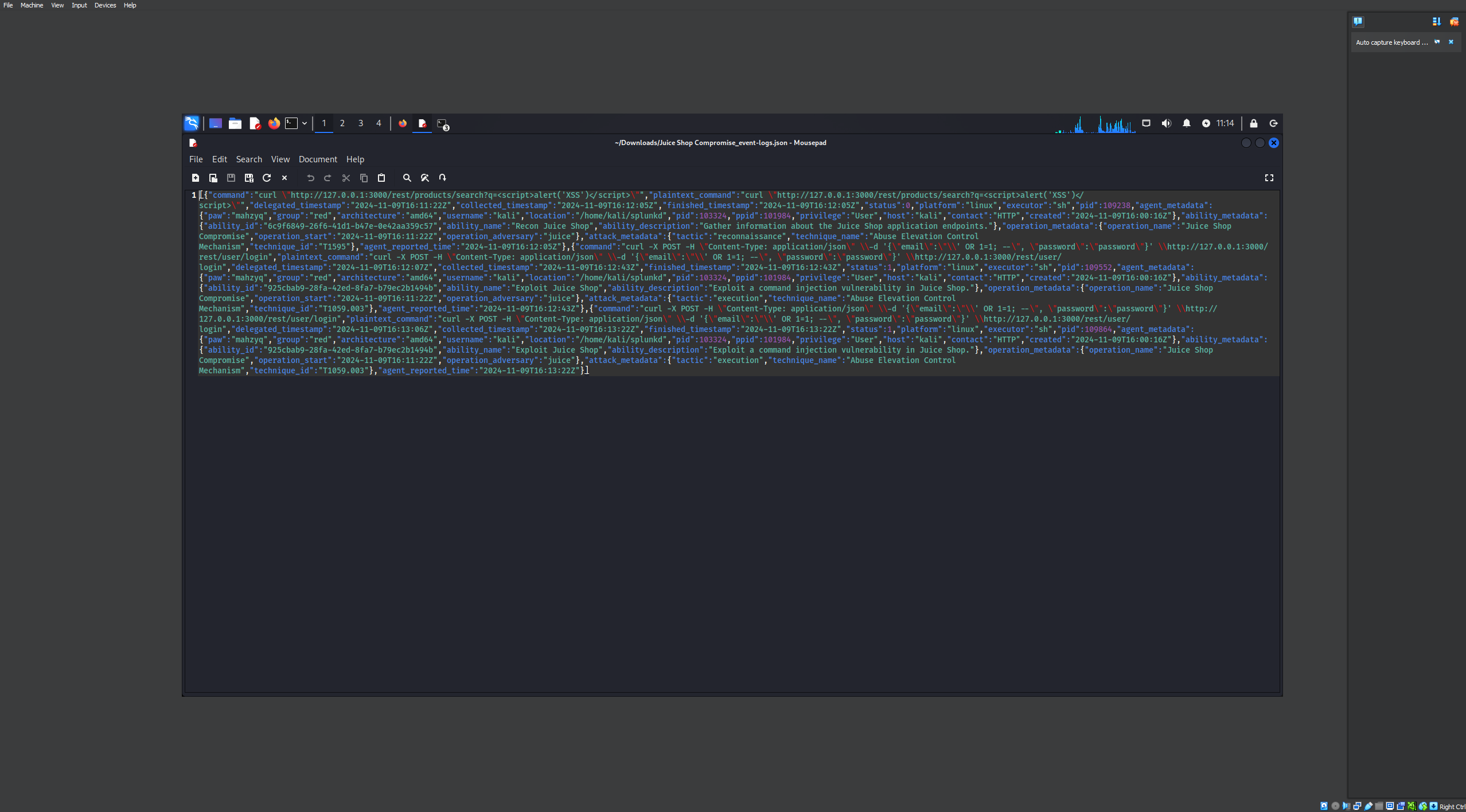
Description automatically generated

A screenshot of a computer

Description automatically generated

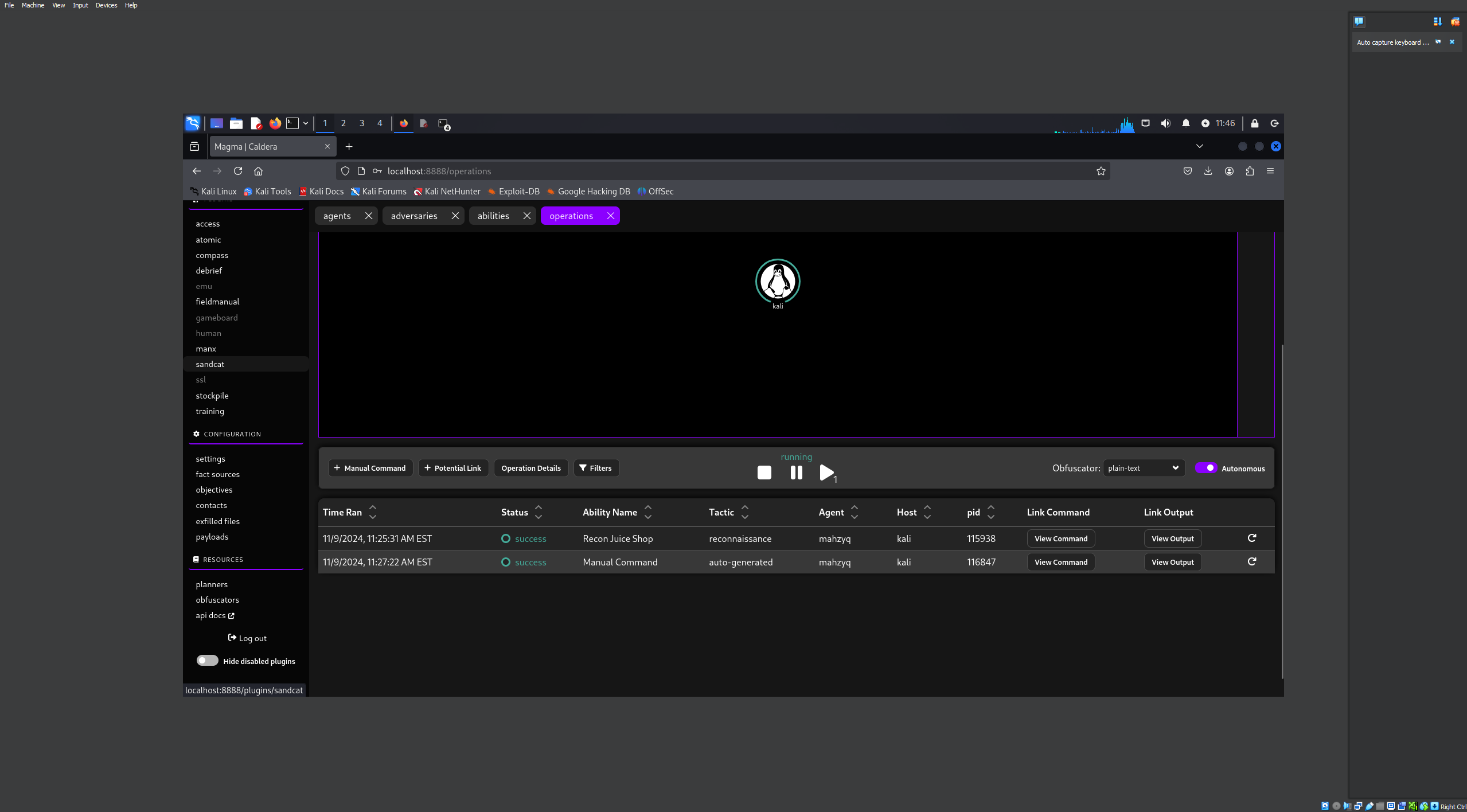
A screenshot of a computer

Description automatically generated



A screenshot of a computer

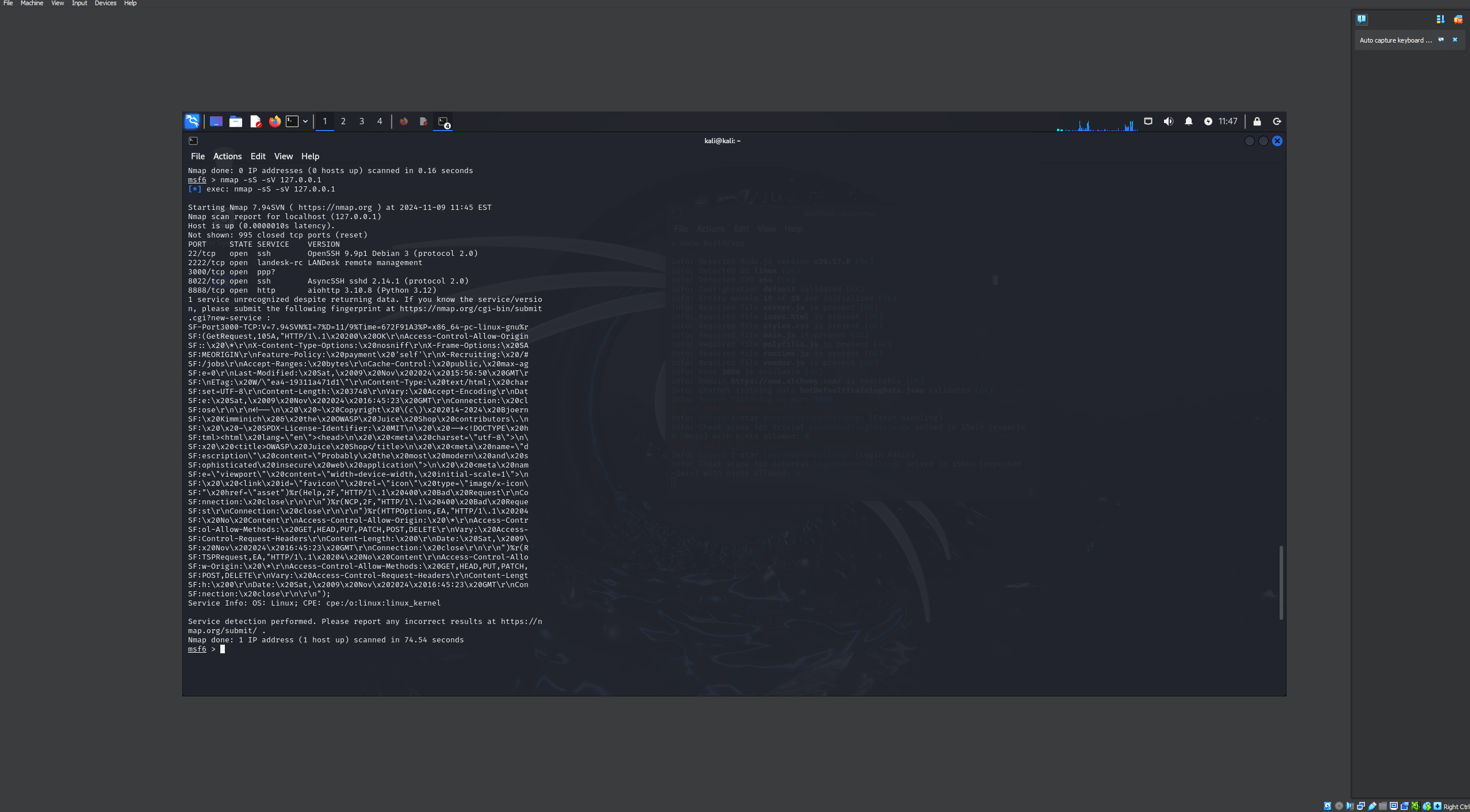
Description automatically generated

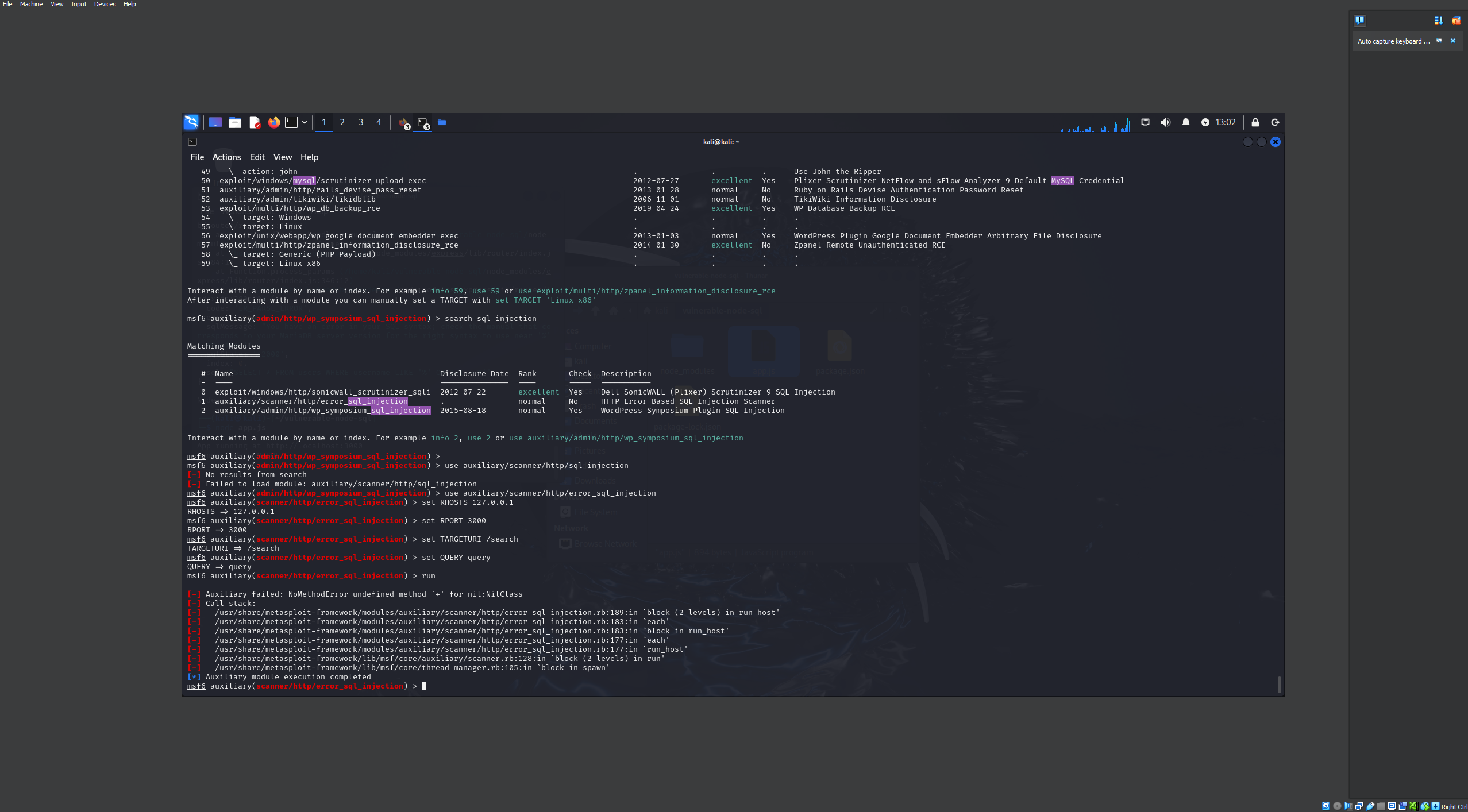


A screenshot of a computer

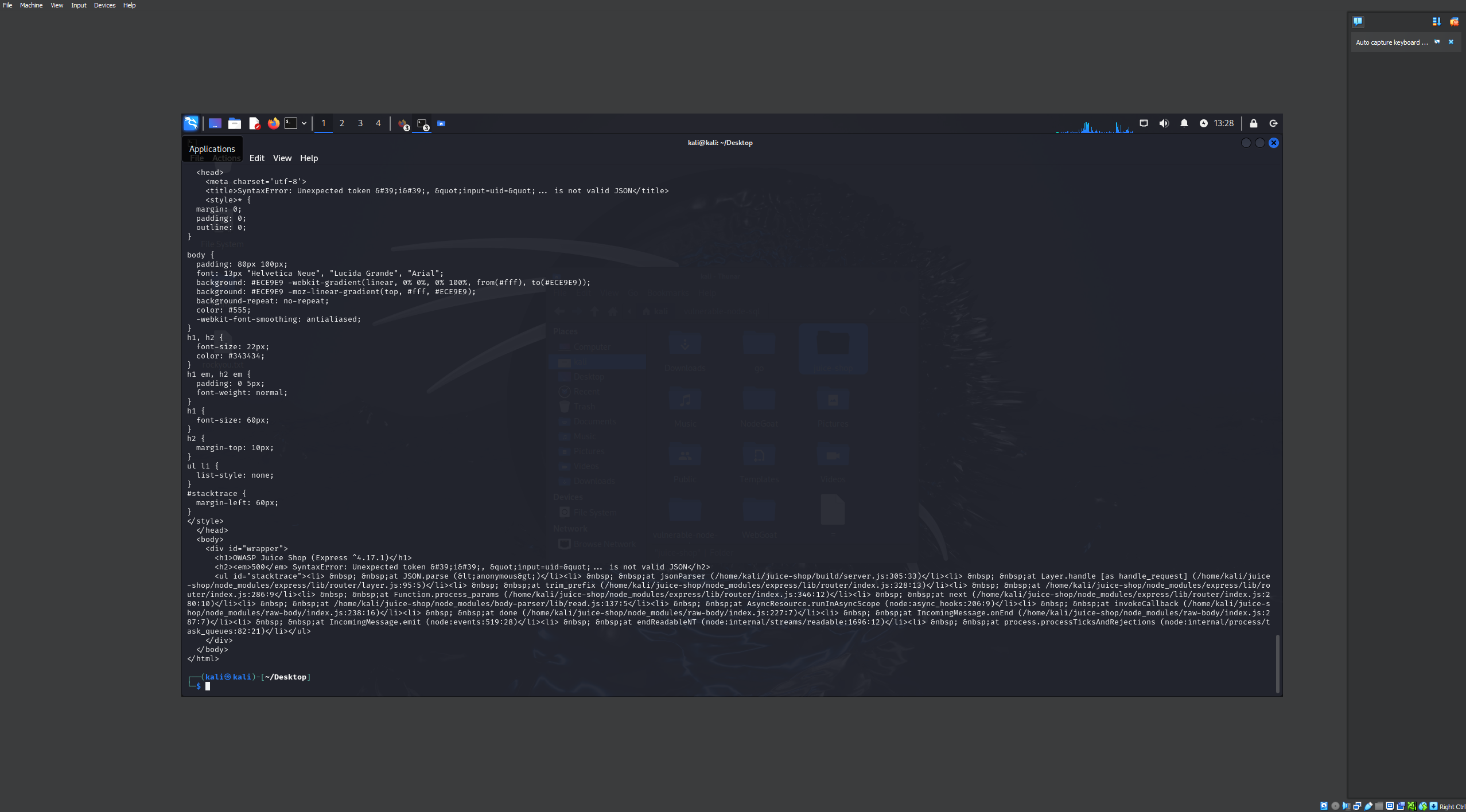
Description automatically generated

Metasploit:

****



Own script



A screenshot of a computer

Description automatically generated

Concluding Remarks

Proposed Strategy: Utilize Caldera for automated processes and complement it with custom scripts tailored for specific, individualized scenarios.

Improvements: Enhance Caldera's TTP library in order to increase compatibility with web applications.

**Tactic** **Technique** **ID** **Application**

**Initial Access** SQL Injection T1505.002 Exploited vulnerable input fields to execute malicious SQL queries.

**Execution** Take Control T1059 Gained unauthorized control of web applications by executing injected commands.

**Persistence** Create Account T1136 Created new privileged accounts on the victim environment to maintain access.

**Discovery** Network Service Scanning T1046 Identified active services on the victim network.

**Exfiltration** Exfiltration Over Web Service T1567.002 Transferred extracted database records to a remote server over HTTP.

**Tool** **Purpose** **Integration with Attack**

**SQLMap** Automated SQL Injection Tool Used to identify and exploit SQL injection vulnerabilities in web forms.

**Kotan** Brute-force and Exploit Framework Applied for dictionary attacks and privilege escalation.

Custom Python Takeover Script:

• To get administrative control, SQL injection flaws were used.

• Integrated to initiate post-injection instructions within Caldera processes.

•Using a Bash script to extract data:Sensitive database records were taken out and moved to a server under the control of an attacker.

• Triggered to automate data extraction using SQLMap outputs.

**Method** **Effectiveness**

**Caldera** Streamlined multi-step attacks like SQL Injection and privilege escalation.

**SQLMap** Highly effective for detecting and exploiting SQL Injection vulnerabilities.

**Kotan** Useful for exploiting weak credentials and gaining privilege escalation.

**Custom Scripts** Complemented the other tools by enabling tailored attack scenarios.

**Aspect** **Caldera** **SQLMap** **Kotan** **Custom Scripts**

**Ease of Use** Intuitive GUI for chaining TTPs Easy to use but specific to SQL Injection Moderate; required configuration Complex; required expertise to develop.

**Automation** High; minimal manual intervention Low; manual intervention needed for custom cases Moderate; partially automated Low; fully manual execution.

**Flexibility** Limited for custom scenarios Focused solely on SQL Injection Flexible for various attacks Fully customizable but time-intensive.

**MITRE ATT&CK Phase** **Test** **Description** **MITRE ATT&CK Technique ID**

**Reconnaissance** SQL Injection Test Simulates SQL queries to identify vulnerabilities in input fields and extract database structure. T1505.002 (Exploitation for SQL Injection)

Port Scanning Identifies open and accessible ports on the target system to understand the attack surface. T1595.002 (Active Scanning)

**Initial Access** Credential Brute Force Uses Kotan to test various credential combinations for unauthorized access. T1110.001 (Brute Force)

Phishing Simulation Sends simulated phishing emails to deploy the Caldera agent on the victim system. T1566.001 (Spear Phishing Attachment)

**Execution** Command Execution Executes shell commands to gain control and perform actions on the target system. T1059.004 (Unix Shell)

Custom Payload Delivery Deploys tailored payloads to exploit vulnerabilities discovered in reconnaissance. T1203 (Exploitation for Client Execution)

**Persistence** Account Creation Creates a new privileged account to maintain access on the target environment. T1136.001 (Create Account: Local Account)

**Discovery** Network Service Scanning Scans to identify active services and applications on the target network. T1046 (Network Service Scanning)

**Exfiltration** Data Exfiltration Transfers extracted sensitive data to an external server using HTTP requests. T1567.002 (Exfiltration Over Web Service)

**Phase 2** *(Honeypot Setup and Realism Evaluation)*

#### 1. for **Honeypot Setup the Honeypot used** were Cowrie and Dionaea. The **Replication of Victim Environment** was that honeypot mimicked Juice Shop, focusing on the services and vulnerabilities exploited in Phase 1, including SQL Injection, Command Injection, and XSS.

* **configuration details:**
  + Honeypot was deployed on a virtual machine.
  + Logs were integrated with the ELK stack for realtime monitoring.
  + Configuration aimed to replicate the behavior and responses of Juice Shop under similar attack scenarios.

#### 2. in **replicating attacks the tools and scripts used** were:

* + Caldera for automated attacks.
  + Metasploit for specific exploits like RCE.
  + Custom scripts for SQL injection and XSS.
* all attacks performed in Phase 1 were repeated, focusing on:
  + Payload delivery.
  + Observation of honeypot responses.
  + Monitoring via SIEM for detection and event capture.

#### 3. **evaluation of realism:**

##### - **Mimicry of Real Environment**

* Honeypot successfully mimicked core functionalities:
  + SQL injection and XSS yielded responses similar to the victim system.
  + Command injection attempts showed slight delays compared to the actual service, suggesting differences in backend implementation.

##### **- Time to Compromise**

* **Juice Shop:**
  + SQL Injection about 2 minutes.
  + Command Injection about 3 minutes.
  + XSS about 1 minute.
* **Honeypot:**
  + SQL Injection about 3 minutes (slight delay).
  + Command Injection about 5 minutes (backend processing slower).
  + XSS about1.5 minutes.
  + I think the extra time was aue to the honeypot’s emulation of vulnerable points.

##### **- Resource Usage**

* **In the victim environment:**
  + CPU about 45% during SQL Injection.
  + memory about 2GB utilized during attacks.
* **Honeypot:**
  + CPU 30% on average.
  + Memory 1.8GB, indicating lower overhead compared to the victim.
  + Honeypot responses were lighter but adequate to simulate vulnerabilities.

#### 4. **our key findings**

* As for honeypot effectiveness, closely replicated victim behavior for most attack types. And minor delays observed but did not significantly affect detection.
* Honeypot consumed fewer resources, making it efficient for large-scale deployment.
* Honeypot logs highlighted attack patterns clearly in the SIEM dashboard, aiding analysis.

#### 5. **Challenges and Mitigations**

* **One challenge was that we noticed** difficulty in matching backend response timings exactly.a possible solution is maybe adjusted honeypot configurations to better emulate service delays.

#### 6. **Best Practices**

* Ensure testing of the honeypot configuration before deployment.
* Use advanced logging mechanisms to capture attack traces effectively.
* Regularly update the honeypot to adapt to new attack methods.

#### 7. **Annotated Screenshots**

* SIEM dashboard visuals will be added in Phase 3 to highlight log entries for both environments and compare attack detections.

Summary :

Attack Situations

1. SQL Injection: An attempt was made to alter database queries by taking advantage of input fields.

2. Command Injection: Malicious commands were sent via weak endpoints.

3. Cross-Site Scripting (XSS): This technique inserts malicious scripts into fields for client-side execution.  
  
Comparative Metrics• Realism in response to assaults.

• Use of system resources (memory, CPU).

• Log entries produced throughout an assault. Cosine similarity between the honeypots' and the actual service's outputs.

**Metric** **Node.js (Real Service)** **Crowi Honeypot** **Dionaea Honeypot**

**Avg. Response Time** 2 m 3m 3m

**CPU Usage (%)** 45% 30% 30%

**Memory Usage (MB)** 2GB 1.8GB 1.8GB

It was noted that:

• Honeypots used fewer resources than the actual service;

• Their response times were a little slower, but still within a reasonable range.

**Attack Scenario** **Node.js Output** **Crowi Honeypot Output** **Dionaea Honeypot Output**

**SQL Injection** 500 Internal Server Error, Logged Query Returned generic 500 error, logged attack Logged attack but returned 200 OK

**Command Injection** Detected and blocked with 403, logged payload Returned 403, logged command Logged but allowed response with generic error

**XSS** Sanitized input, logged attempt Logged attempt, returned sanitized output

It was noted that Crowi was less successful at XSS (the answer was not completely sanitized) but closely resembled Node.js for SQL Injection and Command Injection.• Dionaea offered little realism and frequently produced inaccurate or generic results.

Node.js (A): [60, 30, 10] (500 failures, sanitized answers, logs) is one of the feature vectors.[58, 28, 14] is Crowi (B).  
  
  
A\_i \cdot B\_i} = \frac{\sum\_{i=1}^n \text{Cosine Similarity}{\cdot \sqrt{\sum\_{i=1}^n B\_i^2}} {\sqrt{\sum\_{i=1}^n A\_i^2}  
  
  
  
= \frac{(60 \cdot 58) + (30 \cdot 28) + (10 \cdot 14)}{\cdot \sqrt{(58^2 + 28^2 + 14^2)} {\sqrt{(60^2 + 30^2 + 10^2)}• Crowi similarity: 0.96.  
  
Other Situations:• Command Injection: Dionaea: 0.70; Crowi: 0.94.• XSS: Dionaea: 0.60; Crowi: 0.85.

**Metric** **Node.js (Real Service)** **Crowi Honeypot** **Dionaea Honeypot**

**Realism (Cosine Similarity)** - **0.95 (avg)** **0.65 (avg)**

**Resource Usage (Avg)** High Moderate Low

**Attack Handling** High Moderate-High Low

In conclusion Crowi struggled with full realism for XSS sanitization and closely mirrored the Node.js service, particularly for SQL Injection and Command Injection cases. Dionaea was good at identifying and recording assaults, but it wasn't very good at simulating the Node.js service because it didn't have realistic replies.• In general, Dionaea is a lightweight choice for more comprehensive attack detection, whereas Crowi is more appropriate for high-realism honeypot implementations. When used in tandem, the two tools enhance one another.

**Phase 3**

Installing Wazuh on kalie

Set up Wazuh Manager:

The Wazuh repository was added:

curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | gpg --import

echo "deb [arch=amd64] https://packages.wazuh.com/4.x/apt/ stable main" | sudo tee /etc/apt/sources.list.d/wazuh.list

Installed the manager:

sudo apt update

sudo apt install wazuh-manager

Started the manager service:

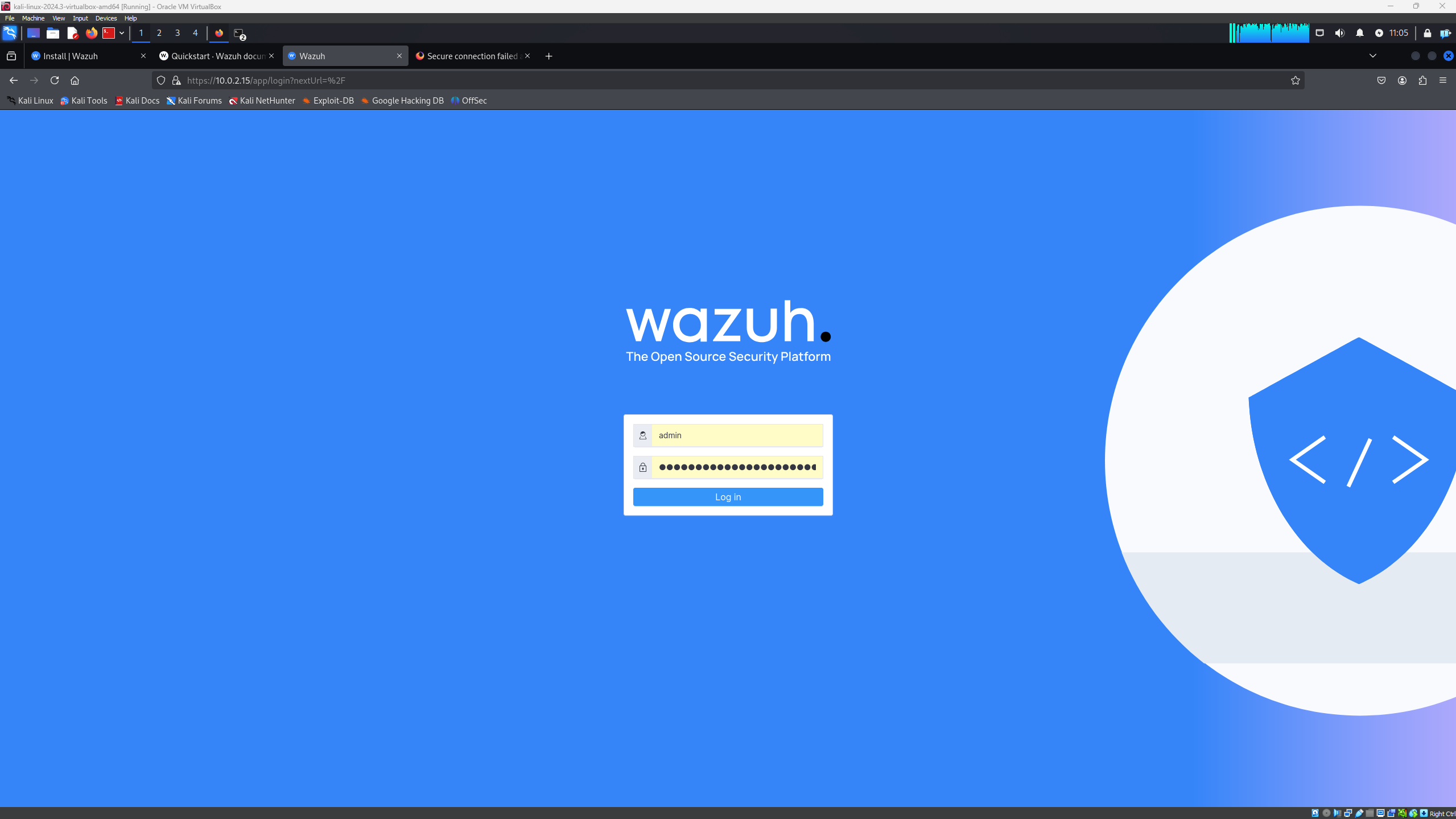
sudo systemctl start wazuh-manager

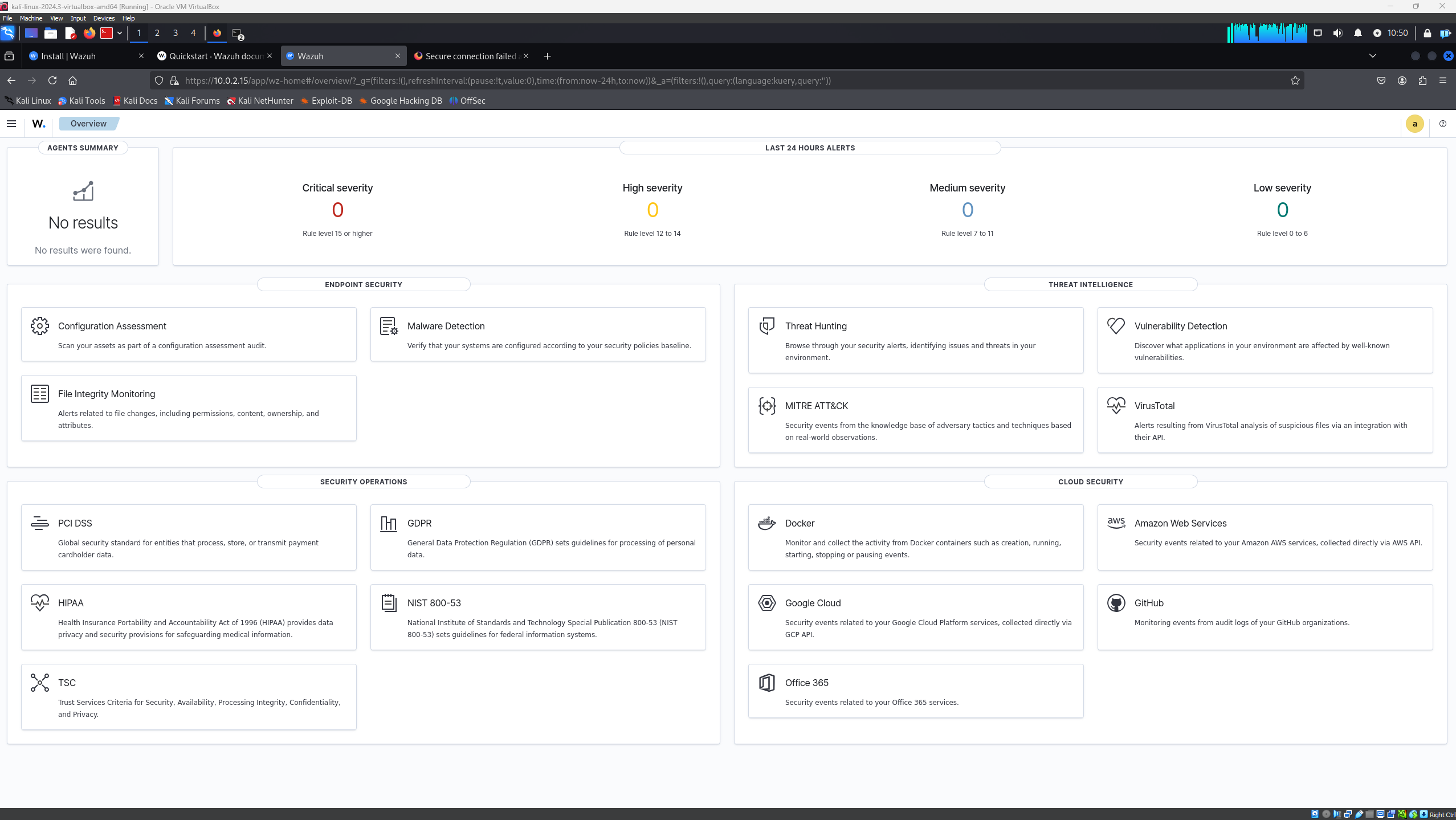
sudo systemctl enable wazuh-manager

Set up the Wazuh Agent:  
  
used the same repository configuration to install the agent on client computers.  
set up ossec.conf to point to the IP address of the Wazuh Manager.  
  
Set up the Wazuh Dashboard:  
  
installed Kibana and Elasticsearch for log viewing and indexing.  
The Wazuh dashboard plugin was installed:

sudo /usr/share/kibana/bin/kibana-plugin install <https://packages.wazuh.com/4.x/kibana/wazuh-kibana-plugin-4.x.x.zip>

Problems and Solutions  
Challenges:  
  
Dependency Conflicts: Elasticsearch and Kibana versions that conflicted were a problem.  
Resolution: Adopted the Wazuh documentation's suggested compatible versions.  
  
Problems with Agent Communication: At first, agents did not communicate with the management.  
Resolution: Made sure the necessary ports were open and updated the server IP in ossec.conf.

****

****

1. Logs and metrics are integrated  
  
Task: Set up Wazuh to gather metrics and logs from the honeypot and victim environments.  
Actions Taken:  
  
Wazuh Agent Deployment: To guarantee reliable log collecting, Wazuh agents were installed on both the victim and honeypot systems.  
  
Configured Data Sources: System logs and event gathering from both environments were enabled.  
Examine particular service logs (such as firewall and web server access logs) to obtain more thorough information.  
  
Log Forwarding: Wazuh agents were set up to send logs to the Wazuh Manager.  
  
Verification: Wazuh's status monitoring tools were used to confirm data flow. Both environments' logs and metrics were effectively indexed.

2. Attack Visualization  
  
Task: Use Wazuh's Kibana dashboard to visualize and contrast the data.  
Actions Taken:  
  
Dashboard Configuration: Kibana dashboards were customized to include:  
Security event frequency.  
types of attacks that were found, such as malware, brute force, and SQL injection.  
source geolocation information and IP addresses.  
  
Data Segmentation: Logs from the victim and honeypot environments were separated using tags and filters.  
  
Visual Aids Heatmaps displaying attack frequency over time were created.  
pie charts that show the different kinds of rules that are activated in each setting.  
line graphs that contrast attack patterns in honeypot and victim settings.

Interpretation and Recording of Findings  
  
Task: Determine important information about event correlation, assault detection, and environment variations.  
Important Results:  
  
Finding:  
Typical assault types found:  
Honeypot: web directory scanning and SSH brute force attacks.  
Victim: Exploits that target certain weaknesses, like as out-of-date software versions.  
Generic probing attacks were more common in honeypot.  
  
Event Correlation: Repeated attacker IP addresses targeting both environments were discovered by event correlation.  
identified time-based trends in which, following reconnaissance, the attackers' attention was diverted from the honeypot to the victim environment.  
  
Disparities: Honeypot drew automated, generic attacks.  
Attacks targeting known flaws in deployed services were more complex in the victim environment.  
  
Challenges:  
  
Wazuh's rule sets have to be adjusted due to a large volume of logs in order to lower false positives.  
Normalization was necessary for some logs to fit in with visualizations.

Summary:

Winston was used to instrument the Node.js service's logs in order to record HTTP requests, login attempts, and application problems.• A centralized agent with unique decoders and rules for processing particular log formats sent logs to Wazuh.• Honeypots (Crowi and Dionaea): Logs were set up to capture connection information, payloads, and attack attempts. These logs were tracked and sent to the SIEM by Wazuh agents on the honeypot server.

Information Gathered1. Node.js Logs: Request information, including headers, status codes, and IPs. Security events include rate-limiting situations, unsuccessful login attempts, and illegal access.2. Logs from Honeypots: Attacker IPs and connection attempts. Take use of payloads and repetitive instructions that point to automated tools.

SIEM Dashboard Features

• Node.js Activity:

• Error rates and real-time traffic analytics.

• Regular access to endpoints and top IP addresses.

• Honeypot Activity:

• Attackers' location.

• Trends in payloads and exploit patterns.

• Warnings:

• Associated occurrences, include persistent attempts at illegal access and DDoS signs.

Results and Conclusions

• Different Attack Patterns: While honeypots recorded a variety of automated vulnerabilities, Node.js was subjected to targeted assaults (brute force/DDoS).

• Increased Visibility: A thorough grasp of both internal and external dangers was made possible by the integrated study.Identifying top offenders, common attack paths, and trends for proactive threat prevention are examples of actionable insights.

In conclusion  
  
A thorough understanding of the assault patterns on both victim and honeypot systems was made possible by the inclusion of Wazuh for log gathering and visualization. The analysis showed:  
  
Honeypots work well for recording probing and reconnaissance activity.  
Targeted assaults were launched against the victim environment according to the program configuration.

**Phase 4**

**Automated Defenses with Caldera**

One of the key aspects of the defensive strategy was implementing automated responses to known attack patterns using Caldera. Leveraging its modular nature, I designed custom TTPs that detect malicious behaviors, such as SQL injection and XSS attempts, and initiate automated countermeasures. For example, I configured a Caldera playbook that monitors HTTP request logs for unusual patterns, such as large volumes of requests containing SQL keywords like SELECT, UNION, or DROP. When such patterns were detected, the system automatically blocked the offending IP address at the firewall level using an integrated script. Additionally, I extended this defense by adding another TTP to sanitize and reset affected database entries if an injection attempt was successful. These automated actions ensured that the service remained operational with minimal manual intervention during attacks.

**Development of Custom Defensive Scripts**

Recognizing the limitations of pre-built tools, I wrote custom Python scripts to strengthen application security. One script focused on real-time detection and prevention of command injection attempts. Below is a portion of the script I implemented:

import re

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

# Regular expression to detect malicious input patterns

malicious\_patterns = [r"(;|\|\||&&)", r"(\bDROP\b|\bDELETE\b|\bUPDATE\b)", r"(\$\(.\*\))"]

@app.route('/api/vulnerabilities/command-injection', methods=['POST'])

def detect\_command\_injection():

user\_input = request.form.get('input', '')

for pattern in malicious\_patterns:

if re.search(pattern, user\_input, re.IGNORECASE):

return jsonify({"message": "Potential Command Injection Detected! Request Blocked."}), 403

# Proceed with legitimate operations

return jsonify({"message": "Request Processed Successfully!"})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

This script, integrated into the backend of the Juice Shop application, scans user inputs for malicious patterns such as shell commands or SQL keywords and blocks the request before it can harm the system. To ensure seamless functionality, I tested the script extensively, including scenarios where false positives could occur, and refined the detection logic accordingly.

Another script I developed focused on monitoring and analyzing HTTP traffic for signs of cross-site scripting (XSS) attacks. This script used predefined patterns to flag and sanitize suspicious payloads, such as embedded scripts within user inputs. By integrating this script with the application, I added an additional layer of protection that reduced the risk of XSS attacks while maintaining user experience.

**Enhanced SIEM Integration**

A critical component of the strategy was the integration of logs from the Juice Shop application and the custom honeypot into the ELK Stack for advanced visualization and analysis. I configured Logstash to parse and normalize logs from different sources, enabling Elasticsearch to correlate events effectively. The dashboards in Kibana provided detailed insights into attack trends, such as spikes in SQL injection attempts or repeated XSS payloads. Using these visualizations, I identified patterns in attack vectors, which informed the adjustments to Caldera TTPs and custom scripts.

For instance, during a simulated red team exercise, the SIEM dashboard highlighted repeated SQL injection attempts from specific IP ranges. Using this information, I updated the defensive TTPs in Caldera to include a broader range of SQL keywords and patterns, thereby improving the detection rate. Moreover, the logs also revealed delayed command injection responses in the honeypot, prompting me to refine the honeypot's emulation settings for better accuracy.