Detecting SQL Injection Attacks with Wazuh: A Hands-On Walkthrough

Introduction

SQL Injection (SQLi) remains one of the most common and dangerous web application vulnerabilities. It allows attackers to manipulate backend SQL queries through malicious input, potentially bypassing authentication or exfiltrating sensitive data.

In this article, we'll walk through how to use **Wazuh**, a free and open-source SIEM/XDR platform, to detect SQL injection attempts by monitoring Apache web server logs. By the end, you'll see how Wazuh flags suspicious requests, maps them to MITRE ATT&CK techniques, and helps security teams respond.

Lab Setup

For this demonstration, I built a simple lab environment:

- **Wazuh Manager:** Running in VMware on a dedicated server
- **Agent:** Windows 10 host (`DESKTOP-GSG0OEO`) with Apache24 installed
- **Monitored Logs:** Apache `access.log` and `error.log` configured in Wazuh agent
- **Test Page:** A deliberately vulnerable PHP page (`auth.php`) designed to log SQL queries for testing

This setup simulates a common enterprise environment where endpoint logs are forwarded to Wazuh for centralized monitoring and detection.

Step 1: Create a Vulnerable Test Page

To simulate SQL injection, I deployed a small PHP page under Apache's `htdocs` directory.

```
<?php
// auth.php (deliberately vulnerable for lab only)
$user = $_GET['username'] ?? '';
$pass = $_GET['password'] ?? '';

$query = "SELECT * FROM users WHERE username = '$user'
AND password = '$pass'";

// log query to Apache error log for Wazuh detection
error_log("[lab] Executed query: $query");

echo "<h3>Executed query:</h3><pre>$query</pre>";
?>
```

This script takes input from URL parameters and builds a raw SQL statement without sanitization—classic SQLi territory.

Step 2: Simulate a SQL Injection Attempt

Next, I simulated a common SQLi attack by sending a crafted request to the vulnerable page:

```
http://localhost:8080/auth.php?username=admin' OR
'1'='1&password=123
```

The malicious payload (`' OR '1'='1`) attempts to always return true for the username check, bypassing authentication.

Apache logs this request in `access.log`, while the vulnerable PHP page writes the full query to `error.log`.

Step 3: Configure Wazuh to Monitor Apache Logs

On the Windows agent, I configured the Wazuh agent to collect Apache logs. In `ossec.conf`, the following block was added:

```
<localfile>
  <log_format>apache</log_format>
  <location>C:\Apache24\logs\access.log</location>
</localfile>

<localfile>
  <log_format>syslog</log_format>
  <location>C:\Apache24\logs\error.log</location>
</localfile>
```

This ensures all web requests and logged queries are shipped to the Wazuh Manager for analysis.

Step 4: Detection in Wazuh

After executing the SQLi payload, Wazuh generated an alert. Here's the actual alert snippet from the Wazuh dashboard:

```
rule.id: 31164
rule.description: SQL injection attempt.
rule.groups: web, accesslog, attack, sqlinjection
rule.level: 6
timestamp: Aug 28, 2025 @ 16:34:09.198
agent.name: DESKTOP-GSG0OEO
location: C:\Apache24\logs\access.log
full_log:
::1 - - [28/Aug/2025:16:34:08 +0530]
"GET /auth.php?username=admin' OR '1'='1&password=123
HTTP/1.1" 200 529
```

Wazuh correctly identified the injection pattern in the URL (`OR '1'='1`) and triggered **Rule ID 31164**, marking it as an SQL injection attempt.

Step 5: MITRE ATT&CK Mapping

The alert was automatically mapped to MITRE ATT&CK techniques:

- **T1190 – Exploit Public-Facing Application**
- **T1055 – Process Injection** (mapped in metadata, though less relevant here)

This helps analysts understand the broader adversary behavior behind the detection.

Step 6: Observations

- Wazuh successfully identified the malicious request and raised an alert with severity level 6.

- The rule was categorized under **web, attack, sqlinjection**, ensuring SOC teams can easily triage and filter SQLi events.

- The detection validates Wazuh's ability to analyze raw Apache logs and recognize common attack patterns.

Step 7: Extending the Use Case

While this demo focused on a simple SQLi payload, the same approach can detect more complex patterns. To extend the detection:

- **Enable ModSecurity with OWASP CRS** for richer SQLi detection and forward its audit logs to Wazuh.

- **Create custom Wazuh rules** for specific SQL keywords or query structures (e.g., `UNION SELECT`, `-- comment`).

- **Enable Active Response** to automatically block the attacking IP using a firewall rule.

Conclusion

With just a vulnerable test page and Apache logs, we demonstrated how Wazuh can effectively detect SQL injection attempts. This use case highlights the importance of log monitoring and rule tuning in identifying early attack behaviors.

By integrating Wazuh into your security stack, you can catch common web attacks like SQLi before they escalate into breaches, and extend detection with frameworks like ModSecurity and MITRE ATT&CK mapping for richer context.