SOC & Automation Report

Table of Contents

Overview of the Selected IOC

Introduction

SQL injection is one of the most common types of threats in the world. It is a common technique used to manipulate data in databases. Attackers use SQL injection to send malicious SQL statements to the database via web page input fields in order to retrieve, modify or delete information. This can allow attackers to gain unauthorized access to sensitive data or perform other malicious actions. Defending against SQL injection requires proper input validation and parameterized queries. This relies on developers using secure coding practices, implementing input validation frameworks, and scan for SQL injection flaws. Given its prevalence, organizations should prioritize SQL injection vulnerabilities in their vulnerability management programs. Additionally, monitoring for SQL injection attempts can further protect an organization by providing automated alerting and mitigation.

Key Characteristics

SQL injection attacks involve inserting malicious SQL statements into an entry field for execution by the underlying database. The attacker crafts the SQL statement to manipulate or extract data they should not have access to.

Common injection points are web forms, URL parameters, and HTTP headers. The injected query can read/write/modify/delete information in the database, execute admin commands, or potentially access OS files.

Setting up Wazuh and Creating a Rule

Installation and Configuration

- Following the steps outlined in the alternative installations documentation, wazuh was set up as a set of docker containers.
  - First clone the wazuh-docker git repository.
    - `git clone https://github.com/wazuh/wazuh-docker.git`
  - Next install docker and docker compose using our systems package manager (pacman for Arch and apt for debian).
    - `sudo pacman -S docker docker-compose`
  - Additionally increase max memory mapped areas.
    - `sudo sysctl -w vm.max_map_count=262144`
  - Navigate to the cloned repository and locate the single-node directory.
    - `cd wazuh-docker/single-node`
  - Use the provided certificate generator to generate the certificates for the wazuh containers.
    - `docker-compose -f generate-indexer-certs.yml run --rm generator`
  - Now the wazuh containers can be started with docker-compose.
    - `docker-compose up` or `docker-compose up -d` (to run in background)
  - Finally configure the wazuh manager using the web interface now accessible at https://localhost.

Rule Creation

- While exploring multiple custom rules to detect malicious domains a lack of experience and knowledge made testing these rules difficult. Here are some of the custom rules tested however it was difficult to find what may be causing the rule to fail to trigger. Testing multiple variations of these rules had no noticeable effect and lack of understanding made it difficult to troubleshoot.

```
  <rule id="100003" level="5">
    <category>syslog</category>
    <match>hydeoutent\.com</match>
    <description>IOC detected for domain hydeoutent.com</description>
  </rule>
<group name="network_traffic">
  <rule id="100002" level="5">
    <match>mainserver\.com</match>
    <description>Suspicious domain found:</description>
    <group>network_traffic, pentest</group>
  </rule>
</group>
```

- After writing the rule and accessing the wazuh manager multiple attempts were made to trigger the rule including, adding the malicious domain to the /etc/hosts file and attempting to curl the domain in order to trigger the detection. However the rule was not detected and after many hours of troubleshooting no rules were triggered including the default rules. In an effort to simplify the variables a pre-built rule was chosen and modifications were made to the test environment.

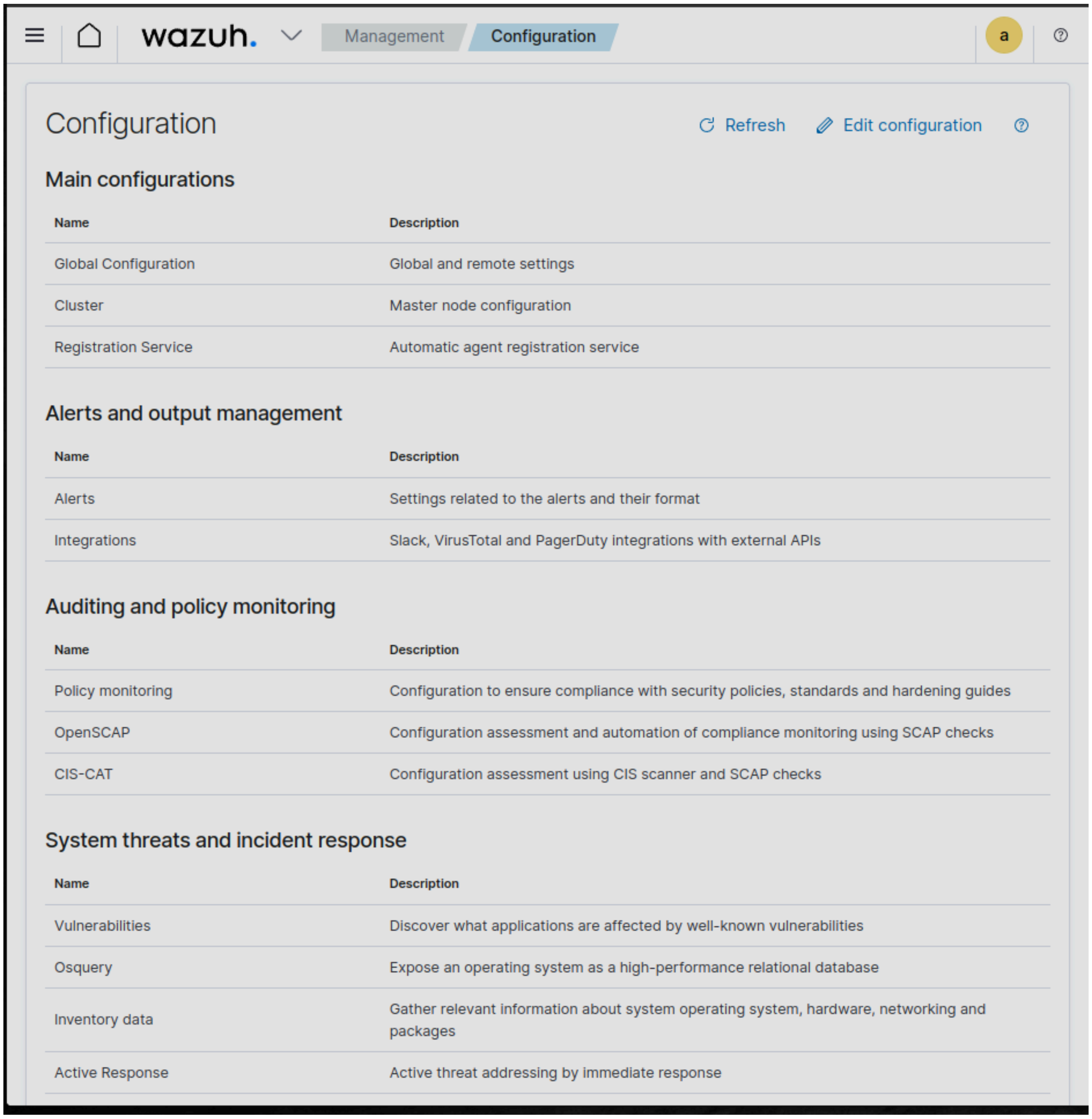
- This lead us to the approach of this report in an attempt to demonstrate a working shuffler.io integration and container monitoring.
- This rule triggers when a URL containing SQL keywords is detected signaling an injection attempt.

```
  <rule id="31164" level="6">
    <if_sid>31100</if_sid>
    <url>=%27|select%2B|insert%2B|%2Bfrom%2B|%2Bwhere%2B|%2Bunion%2B</url>
    <description>SQL injection attempt.</description>
    <mitre>
      <id>T1055</id>
      <id>T1190</id>
    </mitre>

<group>attack,sqlinjection,attack,pci_dss_6.5,pci_dss_11.4,pci_dss_6.5.1,gdpr_IV_35.7.d,nist_800_53_SA.11,nist_800_53_SI.4,tsc_CC6.6,tsc_CC7.1,tsc_CC8.1,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
  </rule>
```

Shuffler.io Configuration for Automated Response

Integration with Wazuh

- Navigate to the Management/configuration page and edit the configuration file.



Configuration

Main configurations

Name	Description
Global Configuration	Global and remote settings
Cluster	Master node configuration
Registration Service	Automatic agent registration service

Alerts and output management

Name	Description
Alerts	Settings related to the alerts and their format
Integrations	Slack, VirusTotal and PagerDuty integrations with external APIs

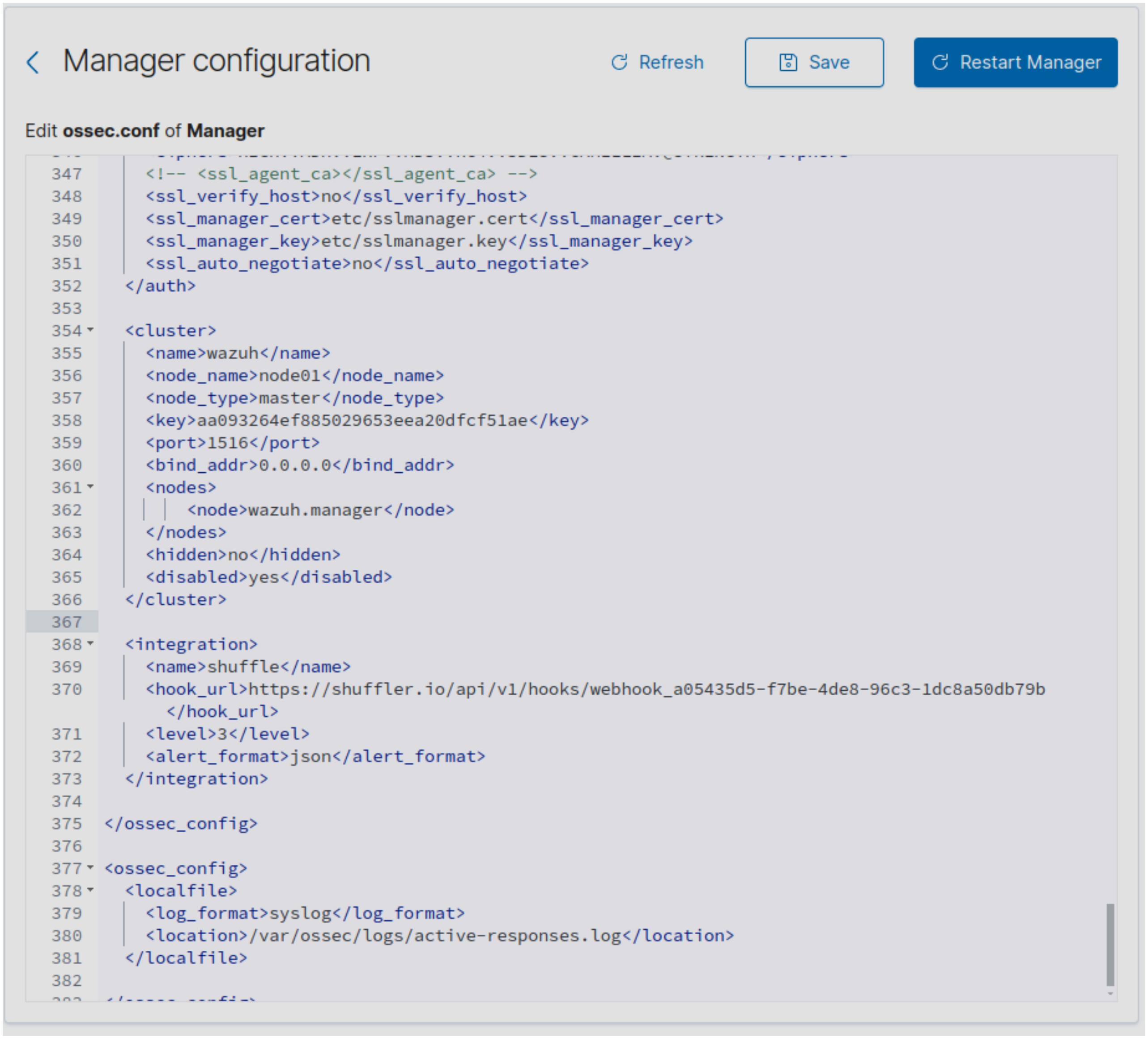
Auditing and policy monitoring

Name	Description
Policy monitoring	Configuration to ensure compliance with security policies, standards and hardening guides
OpenSCAP	Configuration assessment and automation of compliance monitoring using SCAP checks
CIS-CAT	Configuration assessment using CIS scanner and SCAP checks

System threats and incident response

Name	Description
Vulnerabilities	Discover what applications are affected by well-known vulnerabilities
Osquery	Expose an operating system as a high-performance relational database
Inventory data	Gather relevant information about system operating system, hardware, networking and packages
Active Response	Active threat addressing by immediate response

- Add the shuffler.io integration to the ossec.conf file.



```
    <!-- <ssl_agent_ca></ssl_agent_ca> -->
    <ssl_verify_host>no</ssl_verify_host>
    <ssl_manager_cert>etc/sslmanager.cert</ssl_manager_cert>
    <ssl_manager_key>etc/sslmanager.key</ssl_manager_key>
    <ssl_auto_negotiate>no</ssl_auto_negotiate>
  </auth>

  <cluster>
    <name>wazuh</name>
    <node_name>node01</node_name>
    <node_type>master</node_type>
    <key>aa093264ef885029653eea20dfcf51ae</key>
    <port>1516</port>
    <bind_addr>0.0.0.0</bind_addr>
    <nodes>
        <node>wazuh.manager</node>
    </nodes>
    <hidden>no</hidden>
    <disabled>yes</disabled>
  </cluster>

  <integration>
    <name>shuffle</name>
    <hook_url>https://shuffler.io/api/v1/hooks/webhook_a05435d5-f7be-4de8-96c3-1dc8a50db79b
      </hook_url>
    <level>3</level>
    <alert_format>json</alert_format>
  </integration>

</ossec_config>

<ossec_config>
  <localfile>
    <log_format>syslog</log_format>
    <location>/var/ossec/logs/active-responses.log</location>
  </localfile>

```

> *NOTE:* Utilizing the Wazuh web dashboard we were able to add the shuffler.io integration to the ossec.conf file. When using the dashboard to restart the manager everything behaved normally. However if the Docker containers are restarted the changes are lost. May be the result of missing volume mounts or conflicts with running agent on host.

Automated Response Setup

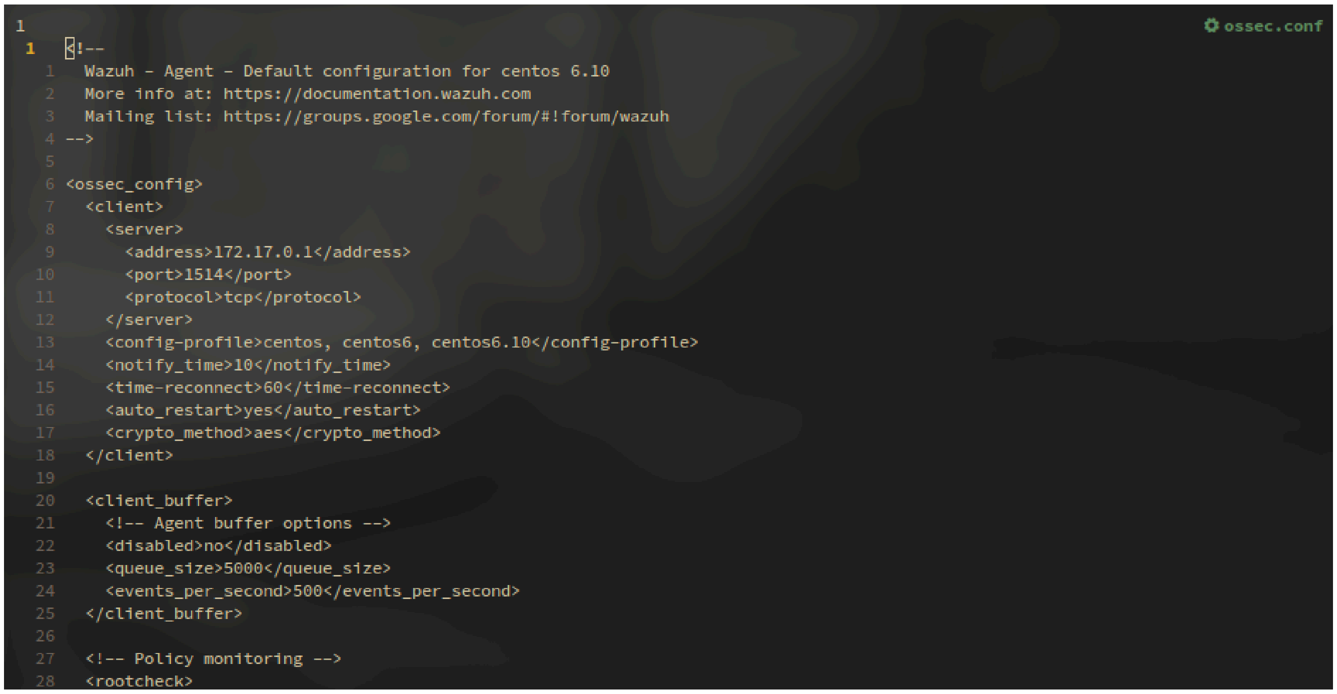
Setup a web hook for shuffler.io to listen for the response from the Wazuh server and specified the call attribute $exec in order to accept execution arguments. Use this web hook URI in the shuffle integration in the wazuh server. Then check the runs to see if there is a response when the Wazuh server restarted or an alert was triggered.

Results and Observations from Threat Simulation

Threat Simulation Scenario

Describe the specific threat scenario simulated for testing the integration of Wazuh and Shuffler.io.

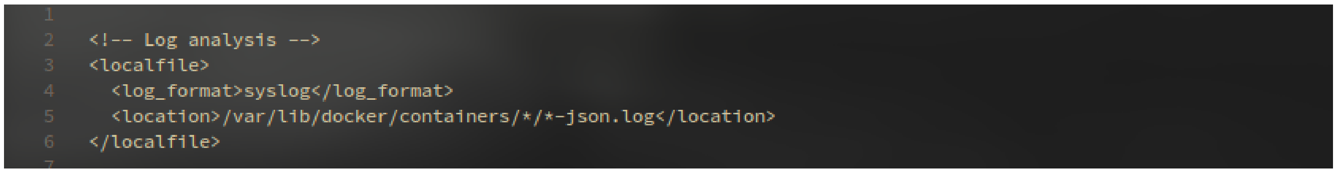
- In order to create a threat simulation install the wazuh agent on host computer utilizing the Arch User Repository (AUR)
- `yay -S wazuh-agent`
- Next the agent is configured to use the wazuh manager by modifying the ossec.conf file to include the wazuh manager ip address.
- `sudo vim /var/ossec/etc/ossec.conf`



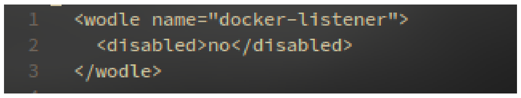
ossec.conf

```
<!--
  Wazuh - Agent - Default configuration for centos 6.10
  More info at: https://documentation.wazuh.com
  Mailing list: https://groups.google.com/forum/#!forum/wazuh
-->

<ossec_config>
  <client>
    <server>
      <address>172.17.0.1</address>
      <port>1514</port>
      <protocol>tcp</protocol>
    </server>
    <config-profile>centos, centos6, centos6.10</config-profile>
    <notify_time>10</notify_time>
    <time-reconnect>60</time-reconnect>
    <auto_restart>yes</auto_restart>
    <crypto_method>aes</crypto_method>
  </client>

  <client_buffer>
    <!-- Agent buffer options -->
    <disabled>no</disabled>
    <queue_size>5000</queue_size>
    <events_per_second>500</events_per_second>
  </client_buffer>

  <!-- Policy monitoring -->
  <rootcheck>
```

- Add the docker logs to the ossec.conf file and enable the docker-listener

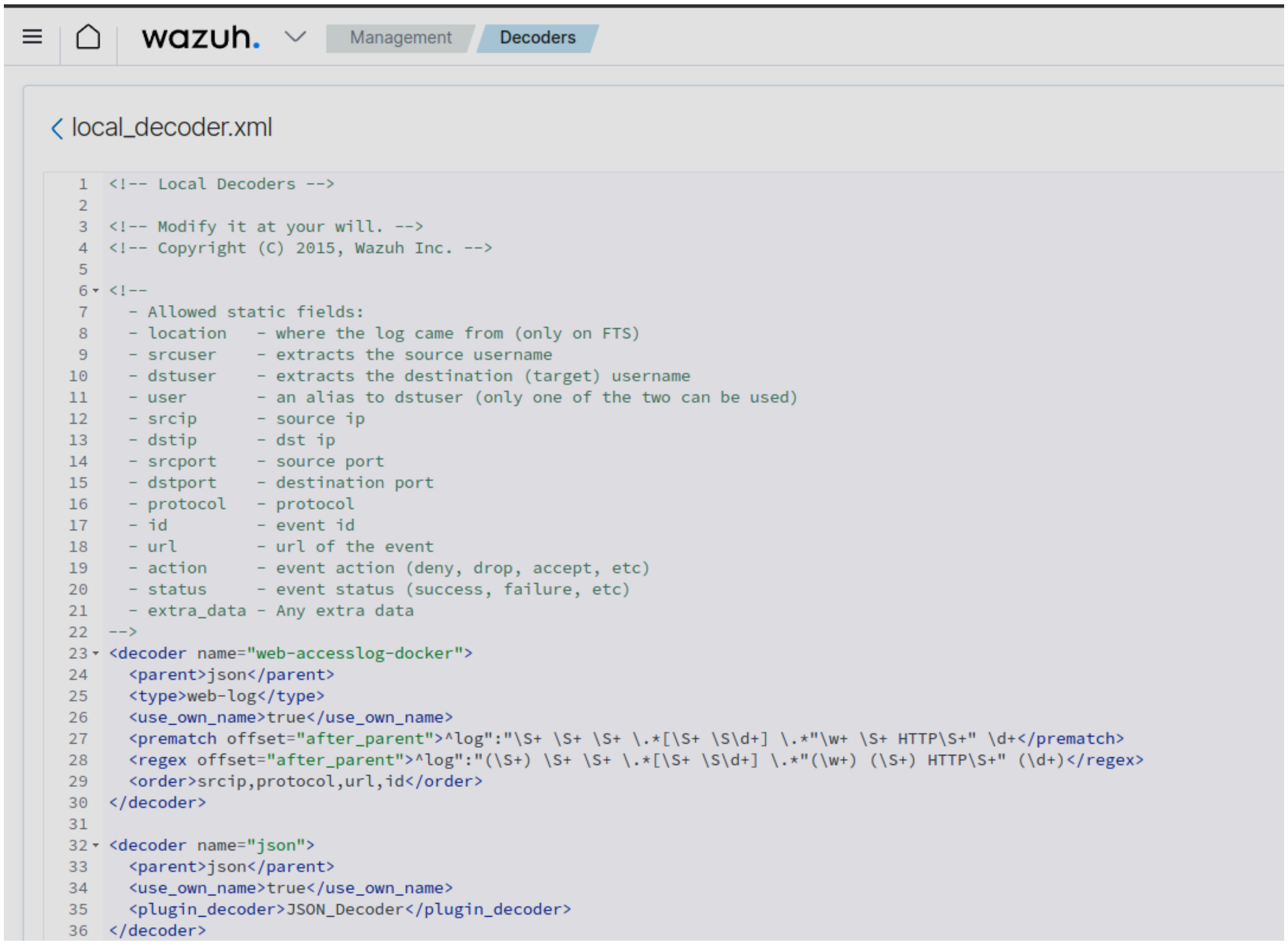


```

  <!-- Log analysis -->
  <localfile>
    <log_format>syslog</log_format>
    <location>/var/lib/docker/containers/*/*-json.log</location>
  </localfile>
```



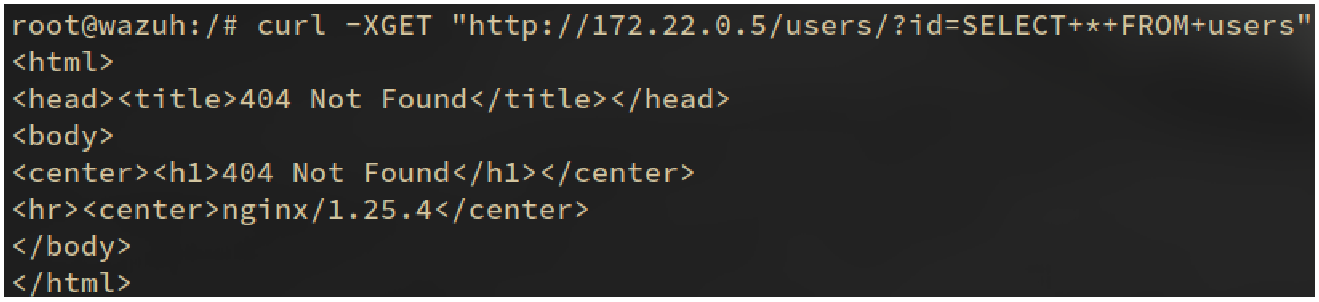
```
<wodle name="docker-listener">
  <disabled>no</disabled>
</wodle>
```

- On the wazuh server add a custom decoder to parse the docker logs and create a json file.



```xml
<!-- Local Decoders -->

<!-- Modify it at your will. -->
<!-- Copyright (C) 2015, Wazuh Inc. -->

<!--
  - Allowed static fields:
  - location   - where the log came from (only on FTS)
  - srcuser    - extracts the source username
  - dstuser    - extracts the destination (target) username
  - user       - an alias to dstuser (only one of the two can be used)
  - srcip      - source ip
  - dstip      - dst ip
  - srcport    - source port
  - dstport    - destination port
  - protocol   - protocol
  - id         - event id
  - url        - url of the event
  - action     - event action (deny, drop, accept, etc)
  - status     - event status (success, failure, etc)
  - extra_data - Any extra data
-->
<decoder name="web-accesslog-docker">
  <parent>json</parent>
  <type>web-log</type>
  <use_own_name>true</use_own_name>
  <prematch offset="after_parent">^log":"\S+ \S+ \S+ \.*[\S+ \S\d+] \.*"\w+ \S+ HTTP\S+" \d+</prematch>
  <regex offset="after_parent">^log":"(\S+) \S+ \S+ \.*[\S+ \S\d+] \.*"(\w+) (\S+) HTTP\S+" (\d+)</regex>
  <order>srcip,protocol,url,id</order>
</decoder>

<decoder name="json">
  <parent>json</parent>
  <use_own_name>true</use_own_name>
  <plugin_decoder>JSON_Decoder</plugin_decoder>
</decoder>
```

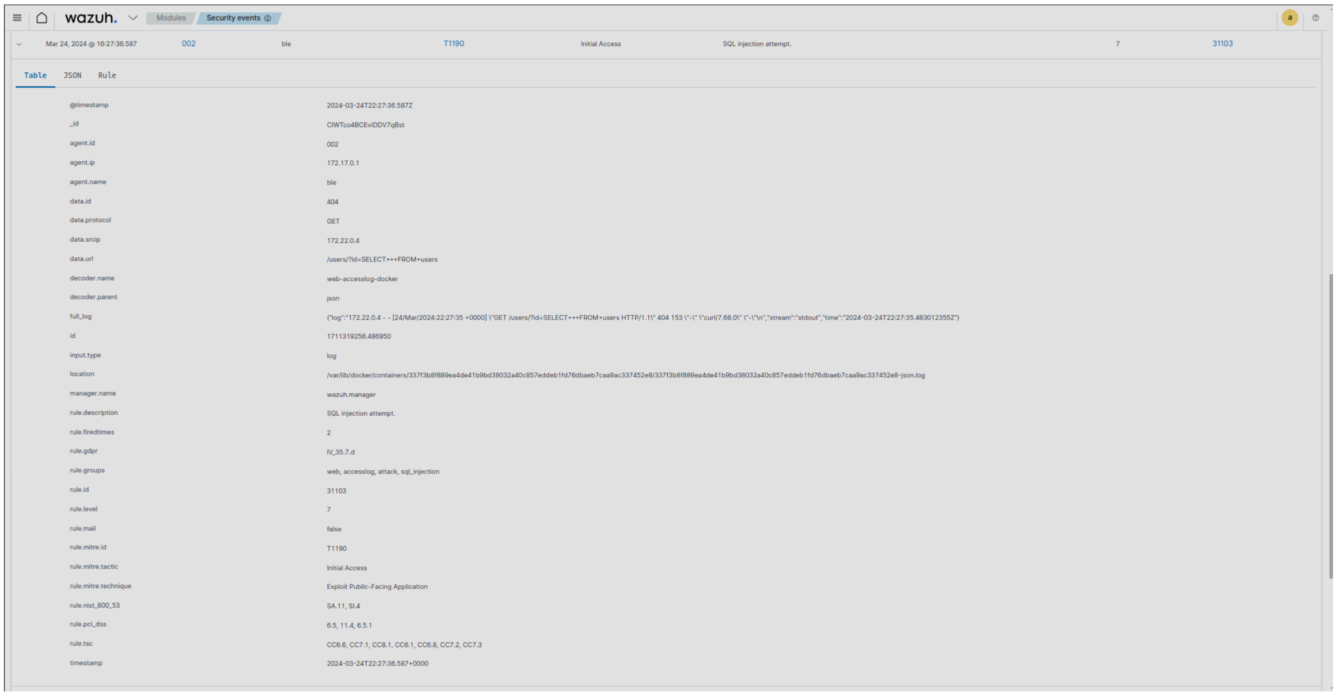
- Now create a new docker container of a base nginx image and deploy it on the wazuh servers network
  - `docker run --name test --network single-node_default -p 80:80 -d nginx`
- Obtain the ip address of the web server with the following command
  - `docker inspect test | grep IPAddress`
- Next access the wazuh manager shell
  - `docker exec -it single-node-wazuh.manager-1 bash`
- From here attempt an SQL Injection on the test web server
  - `curl -XGET "http://172.22.0.5/users/?id=SELECT+*+FROM+users"`



```
root@wazuh:/# curl -XGET "http://172.22.0.5/users/?id=SELECT+*+FROM+users"
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.25.4</center>
</body>
</html>
```

Detection and Response

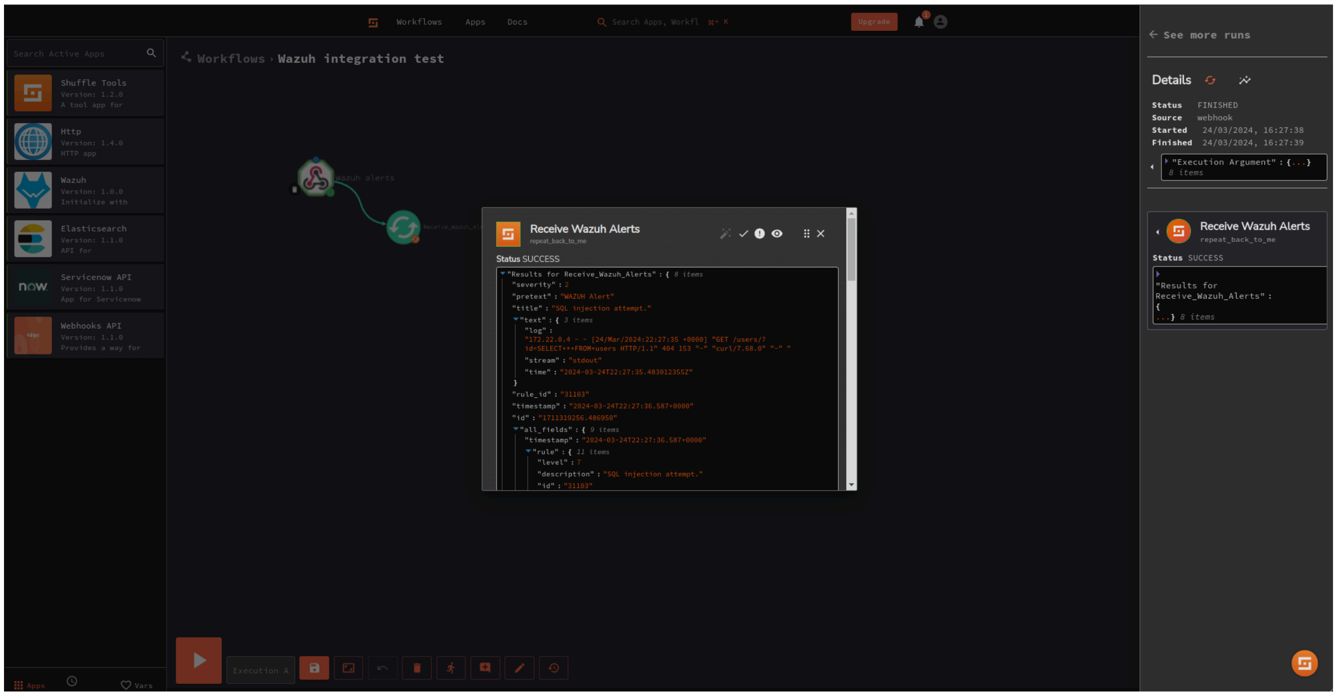
- The server responds with 404 as there is no sql configured however an alert is raised in the wazuh dashboard.



```
Mar 24, 2024 @ 16:27:36.587   002   ble   T1190   Initial Access   SQL injection attempt.   7   31103
```

@timestamp	2024-03-24T22:27:36.587Z
_id	CIWTco4BCEviDDV7qBst
agent.id	002
agent.ip	172.17.0.1
agent.name	ble
data.id	404
data.protocol	GET
data.srcip	172.22.0.4
data.url	/users/?id=SELECT+++FROM+users
decoder.name	web-accesslog-docker
decoder.parent	json
full_log	{"log":"172.22.0.4 - - [24/Mar/2024:22:27:35 +0000] \"GET /users/?id=SELECT+++FROM+users HTTP/1.1\" 404 153 \"-\" \"curl/7.68.0\" \"-\"\n","stream":"stdout","time":"2024-03-24T22:27:35.483012355Z"}
id	1711319256.486950
input.type	log
location	/var/lib/docker/containers/337f3b8f889ea4de41b9bd38032a40c857eddeb1fd76dbaeb7caa9ac337452e8/337f3b8f889ea4de41b9bd38032a40c857eddeb1fd76dbaeb7caa9ac337452e8-json.log
manager.name	wazuh.manager
rule.description	SQL injection attempt.
rule.firedtimes	2
rule.gdpr	IV_35.7.d
rule.groups	web, accesslog, attack, sql_injection
rule.id	31103
rule.level	7
rule.mail	false
rule.mitre.id	T1190
rule.mitre.tactic	Initial Access
rule.mitre.technique	Exploit Public-Facing Application
rule.nist_800_53	SA.11, SI.4
rule.pci_dss	6.5, 11.4, 6.5.1
rule.tsc	CC6.6, CC7.1, CC8.1, CC6.1, CC6.8, CC7.2, CC7.3
timestamp	2024-03-24T22:27:36.587+0000

- Also check out the shuffler.io integration working as it forwards the alert from the webhook



Receive Wazuh Alerts
repeat_back_to_me
Status SUCCESS

```
"Results for Receive_Wazuh_Alerts" : { 8 items
  "severity" : 2
  "pretext" : "WAZUH Alert"
  "title" : "SQL injection attempt."
  "text" : { 3 items
    "log" :
    "172.22.0.4 - - [24/Mar/2024:22:27:35 +0000] "GET /users/?
    id=SELECT+++FROM+users HTTP/1.1" 404 153 "-" "curl/7.68.0" "-" "
    "stream" : "stdout"
    "time" : "2024-03-24T22:27:35.483012355Z"
  }
  "rule_id" : "31103"
  "timestamp" : "2024-03-24T22:27:36.587+0000"
  "id" : "1711319256.486950"
  "all_fields" : { 9 items
    "timestamp" : "2024-03-24T22:27:36.587+0000"
    "rule" : { 11 items
      "level" : 7
      "description" : "SQL injection attempt."
      "id" : "31103"
```

See more runs

Details

```
Status    FINISHED
Source    webhook
Started   24/03/2024, 16:27:38
Finished  24/03/2024, 16:27:39
"Execution Argument" : {...} 8 items
```

Receive Wazuh Alerts
repeat_back_to_me
Status SUCCESS

```
"Results for Receive_Wazuh_Alerts" : { ...} 8 items
```

Observations and Improvements

We faced many challenges in the simulation and spent a majority of the time troubleshooting and assisting our colleges. The initial setup with docker went smoothly and we were confident that configuring rules and integrations would be relatively. Once we had identified the IOC using alienvault we attempted to write a rule that detected if a malicious domain was being accessed. When it came to injecting the IOC to simulate the treat we lacked the confidence to do so without compromising our host machine. We attempted to use the wazuh agent to inject the IOC but it was not working as expected and our additional research did not provide any fruitful insights.

As a last resort we configured container monitoring and deployed a basic web server. We were able to then simulate an SQL injection on the web server which finally resulted in an alert. These challenges inhibited our ability to further investigate our shuffler workflow and we were only able to complete a basic proof of concept.

Conclusion

This remains an area for further research and improvement. Though the project we identified critical gaps in understanding of the tools and their various components. The automation pipeline also requires further investigation to understand how to optimize responses and integrate more services. There were important lessons learned in how to setup a test environment and further research could help in setting up a fully functional home lab. While the tools are powerful they require a deeper understanding to be used effectively.