Security Event and Information Manager(SIEM) with Wazuh

By: Josh Murdock

What is a SIEM?

- Log management
  - Collects log data from various sources (servers, endpoints, firewalls, etc.)

- Real time threat detection
  - Uses predefined rules, correlation, and analytics to detect suspicious behavior or security incidents

- Alerting and Notification
  - Triggers alerts when specific events or patterns are detected

- Forensic and incident investigation
  - Stores historical logs and provides search tools for investigating past incidents

- Dashboard and visualization
  - Offers a web interface to monitor events, alerts, and system health in real time.

- Active response capabilities
  - Can execute automated responses (e.g., block IPs) when a threat is detected.

Siem Options

- Splunk: High scalability, advanced analytics, premium pricing.
- IBM QRadar: Strong threat intelligence, ideal for large enterprises.
- Elastic Security (ELK Stack): Open source, flexible, good for custom setups.
- Wazuh: Open source, lightweight
- Microsoft Sentinel: Cloud-native, easy Azure integration, good for hybrid environments.
- LogRhythm: All-in-one security platform, strong compliance tools.
- AlienVault (now AT&T Cybersecurity): Good threat intelligence, beginner-friendly.

Why SIEM?

- Cyber threats are increasing in frequency and complexity
  (e.g., brute-force SSH attacks, malware infections, port scans)
- Traditional security tools (like firewalls or antivirus) often detect threats after damage is done
- Lack of centralized visibility into logs and system behavior delays threat response
- Manual log analysis is time-consuming and prone to human error
- Coordinated attacks across multiple systems are hard to detect without correlation tools

Wazuh

- Wazuh is a free and open-source SIEM platform
- Unlimited amount of customizability to fit your needs
- In the context of security software, open source is super important for code auditability

Wazuh Architecture

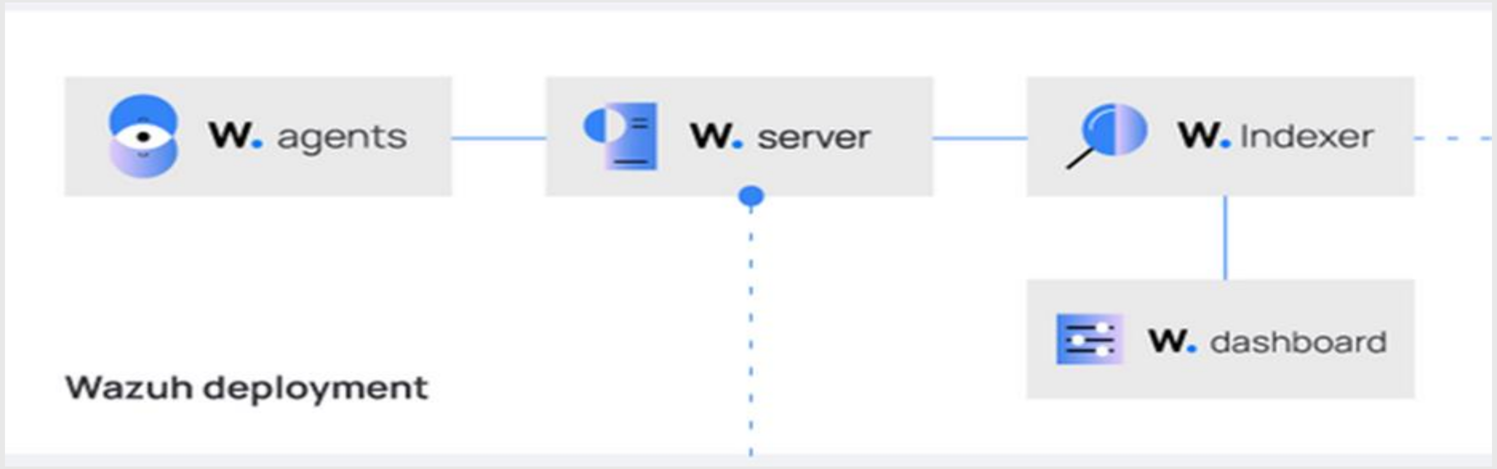
Dashboard
- Web-based interface for interacting with Wazuh
- Lets users filter, search, and monitor security events in real time

Agent
- Installed on monitored endpoints (e.g., servers, workstations)
- Sends collected data securely to the Wazuh Manager

Manager
- Core processing engine of Wazuh
- Matches logs against rule sets to detect threats

Indexer
- Stores log data and security events in a searchable format
- Handles data indexing, querying, and storage optimization

[Agent] → [Manager] → [Indexer] → [Dashboard] Logs



Detecting port scanning

- Port scanning is when someone systematically sends messages to different ports on a computer or server to find out:

  - Which ports are open (accepting connections)

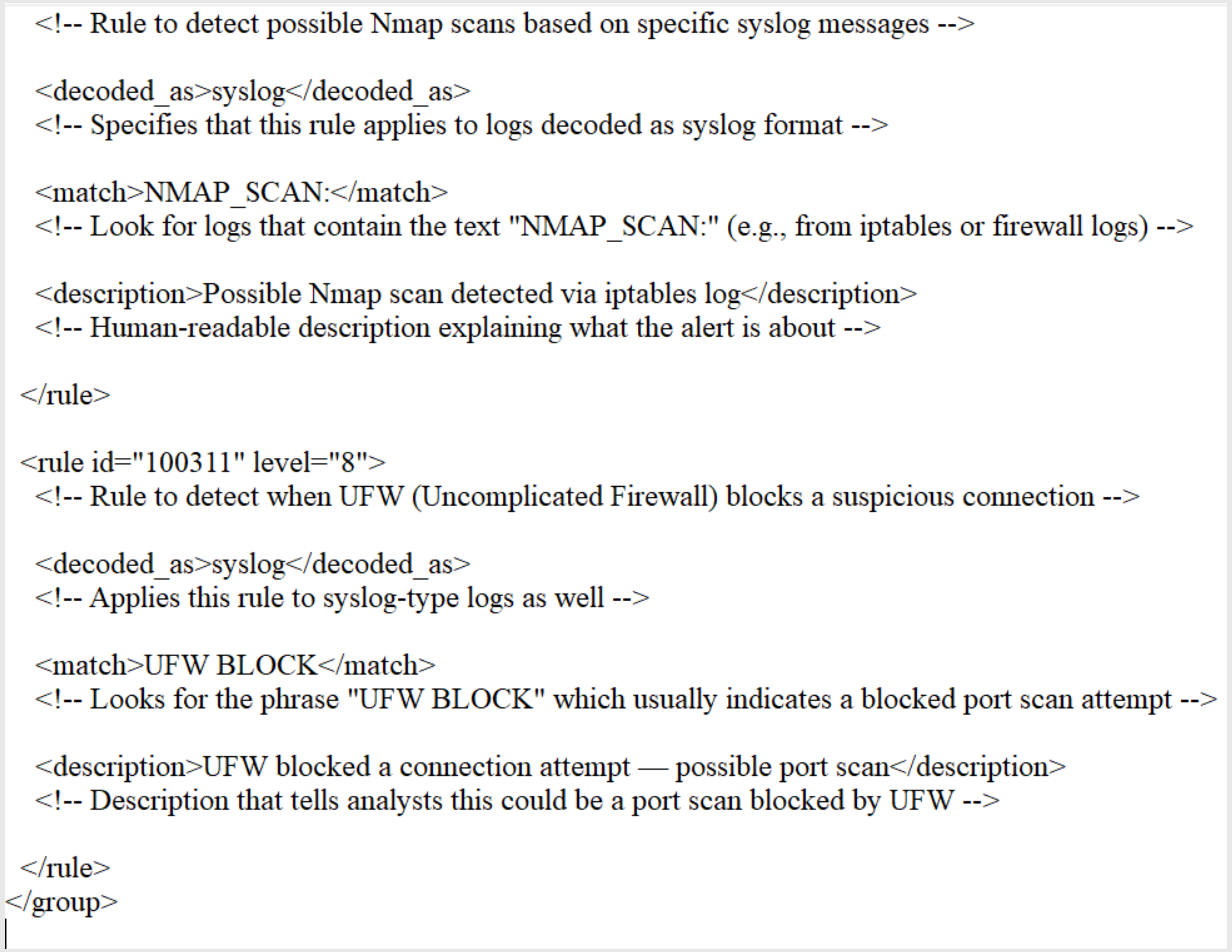
  - Which ports are closed (rejecting connections)

  - What services (like SSH, web servers, FTP) are running on those ports

- I configured Wazuh to monitor its agents to detect when a potential attacker is scanning it

Detecting port scanning

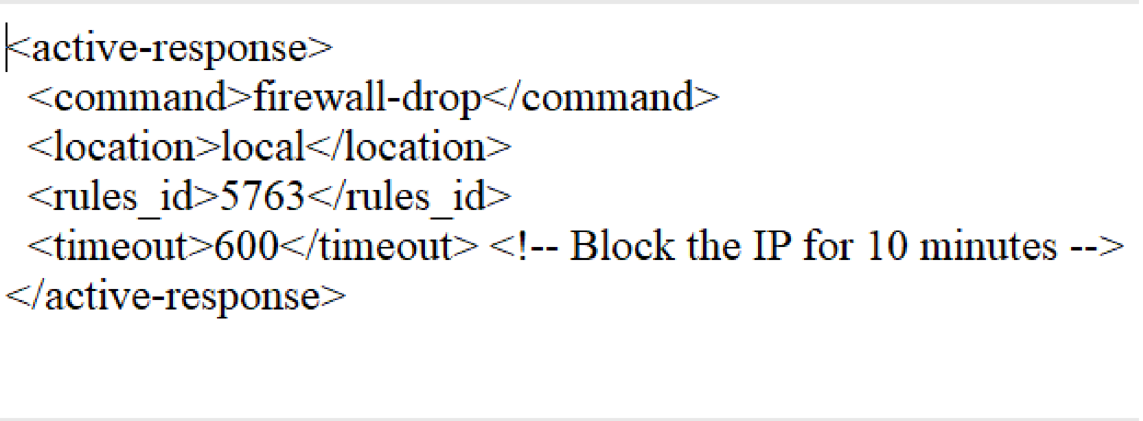
- Configuration in rules file



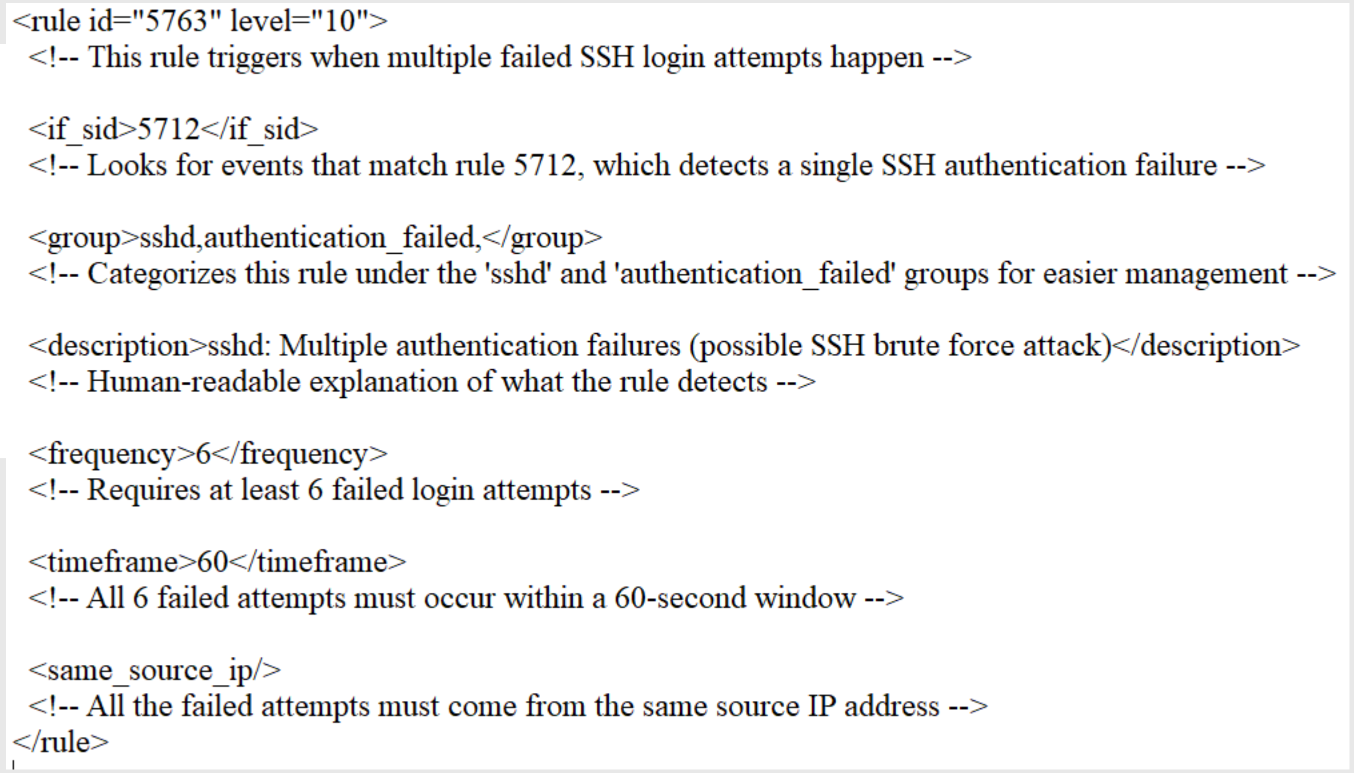
```
  <!-- Rule to detect possible Nmap scans based on specific syslog messages -->

  <decoded_as>syslog</decoded_as>
  <!-- Specifies that this rule applies to logs decoded as syslog format -->

  <match>NMAP_SCAN:</match>
  <!-- Look for logs that contain the text "NMAP_SCAN:" (e.g., from iptables or firewall logs) -->

  <description>Possible Nmap scan detected via iptables log</description>
  <!-- Human-readable description explaining what the alert is about -->

 </rule>

 <rule id="100311" level="8">
  <!-- Rule to detect when UFW (Uncomplicated Firewall) blocks a suspicious connection -->

  <decoded_as>syslog</decoded_as>
  <!-- Applies this rule to syslog-type logs as well -->

  <match>UFW BLOCK</match>
  <!-- Looks for the phrase "UFW BLOCK" which usually indicates a blocked port scan attempt -->

  <description>UFW blocked a connection attempt — possible port scan</description>
  <!-- Description that tells analysts this could be a port scan blocked by UFW -->

 </rule>
</group>
```

SSH Brute-Force Detection & Active Response

- Configuration to detect SSH brute-force attacks by monitoring failed login attempts



```
<active-response>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>5763</rules_id>
  <timeout>600</timeout> <!-- Block the IP for 10 minutes -->
</active-response>
```

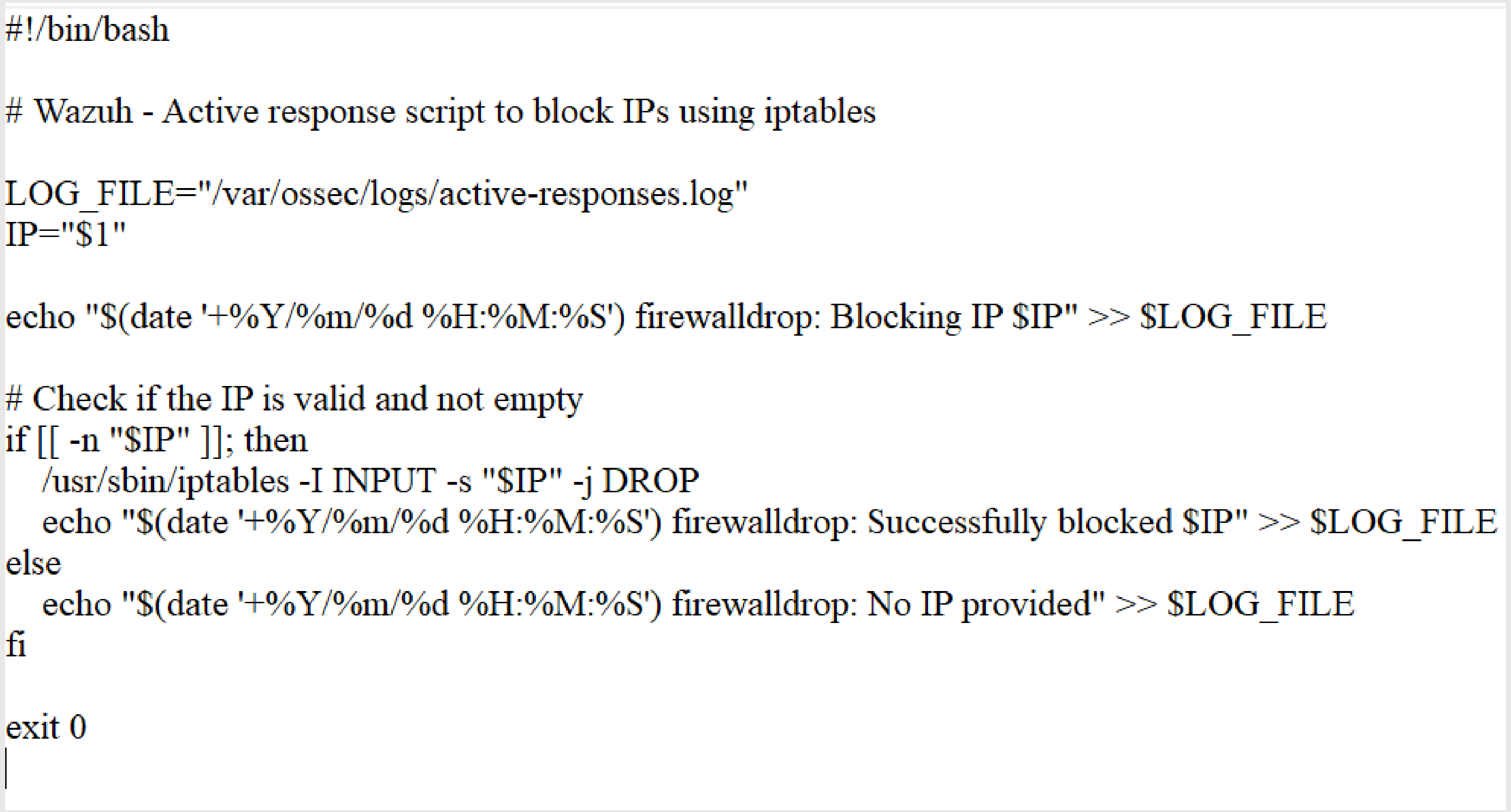


```
<rule id="5763" level="10">
  <!-- This rule triggers when multiple failed SSH login attempts happen -->

  <if_sid>5712</if_sid>
  <!-- Looks for events that match rule 5712, which detects a single SSH authentication failure -->

  <group>sshd,authentication_failed,</group>
  <!-- Categorizes this rule under the 'sshd' and 'authentication_failed' groups for easier management -->

  <description>sshd: Multiple authentication failures (possible SSH brute force attack)</description>
  <!-- Human-readable explanation of what the rule detects -->

  <frequency>6</frequency>
  <!-- Requires at least 6 failed login attempts -->

  <timeframe>60</timeframe>
  <!-- All 6 failed attempts must occur within a 60-second window -->

  <same_source_ip/>
  <!-- All the failed attempts must come from the same source IP address -->
</rule>
```

[Agent] ⤳ Sends logs → [Manager]
[Manager] ⤳ Detects brute force with Rule 5763 ⤳ Sends active response command
[Agent] ⤳ Executes firewall-drop → Blocks attacker

SSH Brute-Force Detection & Active Response

- When the threshold is exceeded, Wazuh triggers an Active Response running a firewall drop bash script



```
#!/bin/bash

# Wazuh - Active response script to block IPs using iptables

LOG_FILE="/var/ossec/logs/active-responses.log"
IP="$1"

echo "$(date '+%Y/%m/%d %H:%M:%S') firewalldrop: Blocking IP $IP" >> $LOG_FILE

# Check if the IP is valid and not empty
if [[ -n "$IP" ]]; then
    /usr/sbin/iptables -I INPUT -s "$IP" -j DROP
    echo "$(date '+%Y/%m/%d %H:%M:%S') firewalldrop: Successfully blocked $IP" >> $LOG_FILE
else
    echo "$(date '+%Y/%m/%d %H:%M:%S') firewalldrop: No IP provided" >> $LOG_FILE
fi

exit 0
```

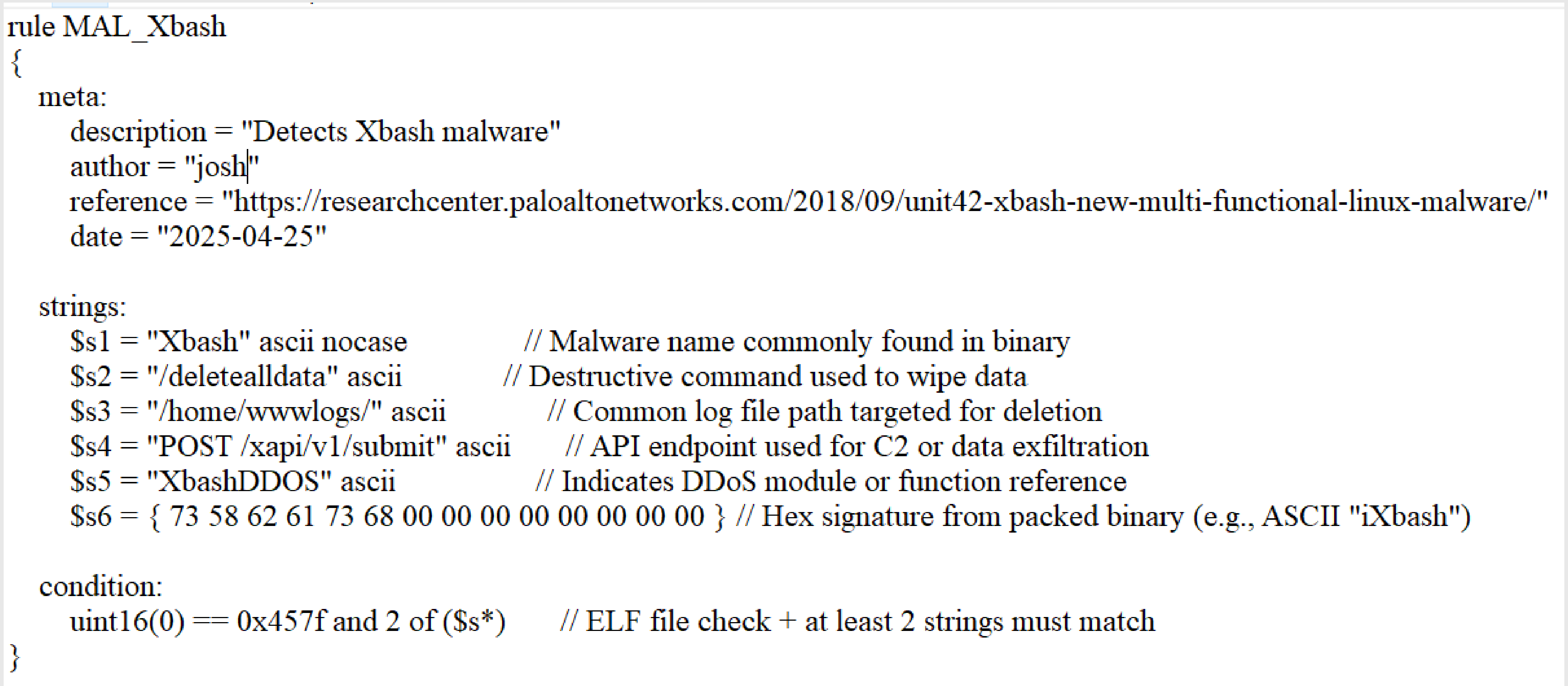
Building on top of Wazuh with YARA

- YARA is another open-source tool
- It is used to identify and classify malware based on patterns in files (string signatures, byte sequences, etc.)
- Wazuh can integrate YARA rules to scan files and directories in real-time or on a schedule
- Free to use platforms like Valhalla have over 20,000 YARA rules to detect malware
- When a YARA rule matches a suspicious file, Wazuh can:

  - Generate an alert
  - Trigger an Active Response (e.g., delete the file or quarantine it)
  - Send rule info to ChatGPT or other tools for analysis

YARA rule for xbash malware

- Xbash: Hybrid Malware: Combines features of a botnet, ransomware, cryptominer, and worm → All-in-one attack tool.



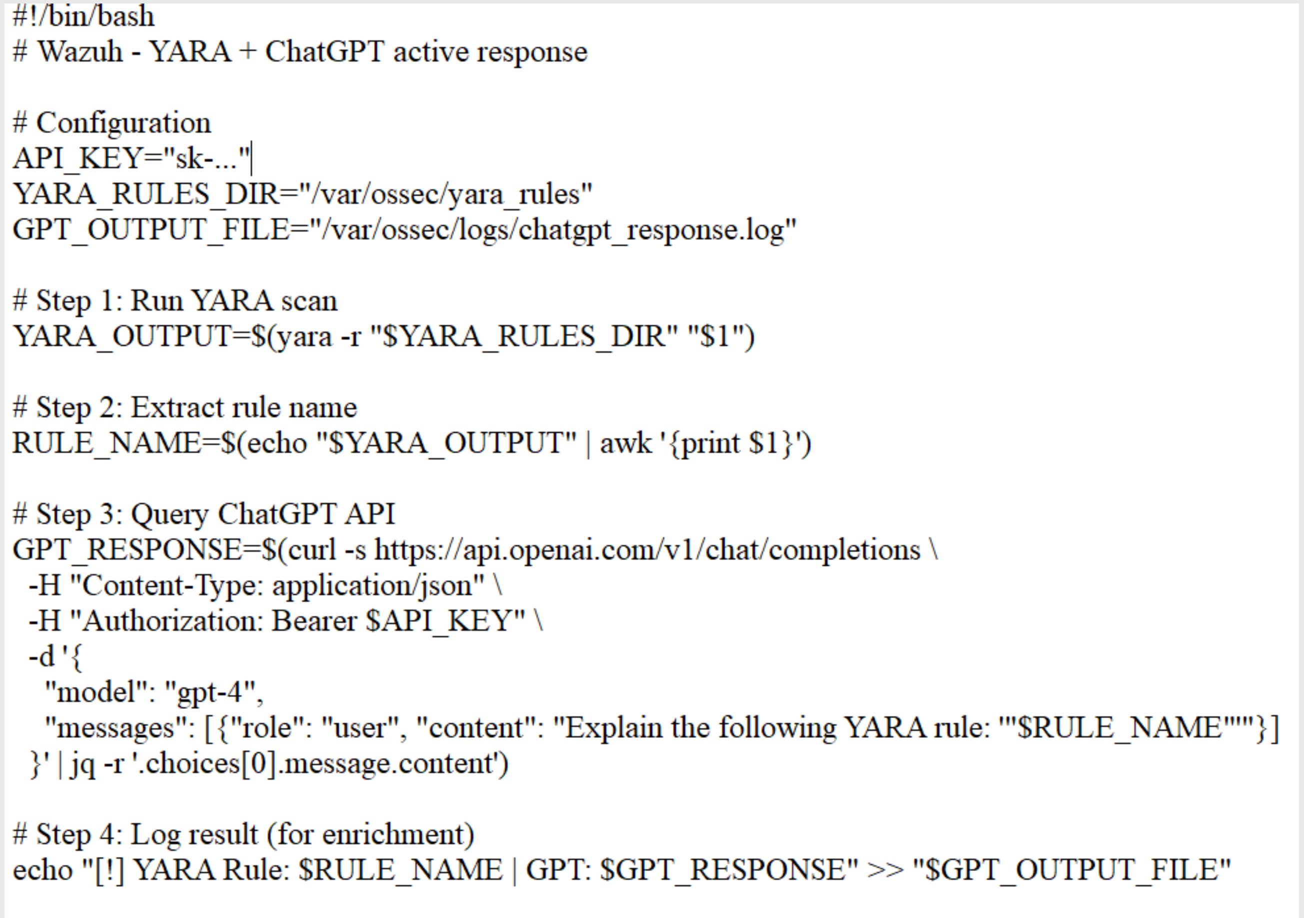
```
rule MAL_Xbash
{
   meta:
      description = "Detects Xbash malware"
      author = "josh"
      reference = "https://researchcenter.paloaltonetworks.com/2018/09/unit42-xbash-new-multi-functional-linux-malware/"
      date = "2025-04-25"

   strings:
      $s1 = "Xbash" ascii nocase              // Malware name commonly found in binary
      $s2 = "/deletealldata" ascii             // Destructive command used to wipe data
      $s3 = "/home/wwwlogs/" ascii              // Common log file path targeted for deletion
      $s4 = "POST /xapi/v1/submit" ascii       // API endpoint used for C2 or data exfiltration
      $s5 = "XbashDDOS" ascii                   // Indicates DDoS module or function reference
      $s6 = { 73 58 62 61 73 68 00 00 00 00 00 00 00 00 } // Hex signature from packed binary (e.g., ASCII "iXbash")

   condition:
      uint16(0) == 0x457f and 2 of ($s*)      // ELF file check + at least 2 strings must match
}
```

Building on top of Wazuh with YARA and Chat-GPT

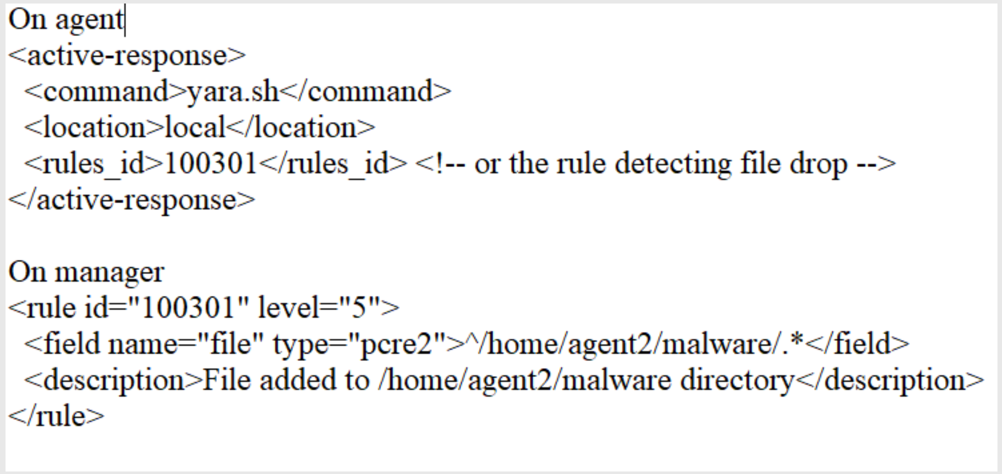
- Once a YARA rule matches a known piece of malware, I configured Wazuh to send it to Chat GPT for further analysis
- With all the different types of malware out there its hard to memorize them all and AI models can help us understand faster a potential threat

Building on top of Wazuh with YARA and Chat-GPT

Configurations



```bash
#!/bin/bash
# Wazuh - YARA + ChatGPT active response

# Configuration
API_KEY="sk-..."
YARA_RULES_DIR="/var/ossec/yara_rules"
GPT_OUTPUT_FILE="/var/ossec/logs/chatgpt_response.log"

# Step 1: Run YARA scan
YARA_OUTPUT=$(yara -r "$YARA_RULES_DIR" "$1")

# Step 2: Extract rule name
RULE_NAME=$(echo "$YARA_OUTPUT" | awk '{print $1}')

# Step 3: Query ChatGPT API
GPT_RESPONSE=$(curl -s https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $API_KEY" \
  -d '{
    "model": "gpt-4",
    "messages": [{"role": "user", "content": "Explain the following YARA rule: '"$RULE_NAME"'"}]
  }' | jq -r '.choices[0].message.content')

# Step 4: Log result (for enrichment)
echo "[!] YARA Rule: $RULE_NAME | GPT: $GPT_RESPONSE" >> "$GPT_OUTPUT_FILE"
```



On agent

```xml
<active-response>
  <command>yara.sh</command>
  <location>local</location>
  <rules_id>100301</rules_id> <!-- or the rule detecting file drop -->
</active-response>
```

On manager

```xml
<rule id="100301" level="5">
  <field name="file" type="pcre2">^/home/agent2/malware/.*</field>
  <description>File added to /home/agent2/malware directory</description>
</rule>
```

Demonstration Video

https://youtu.be/7J-VBG9ozDE

Next steps

- Integrate a local LLM to replace ChatGPT API to keep analysis offline
- Expand to detect other attack types (SQLi, reverse shells, etc)
- Separate Wazuh components manager, indexer, and dashboard to different machines for better security
- Publish a guide or Github repo for other to easily replicate and contribute