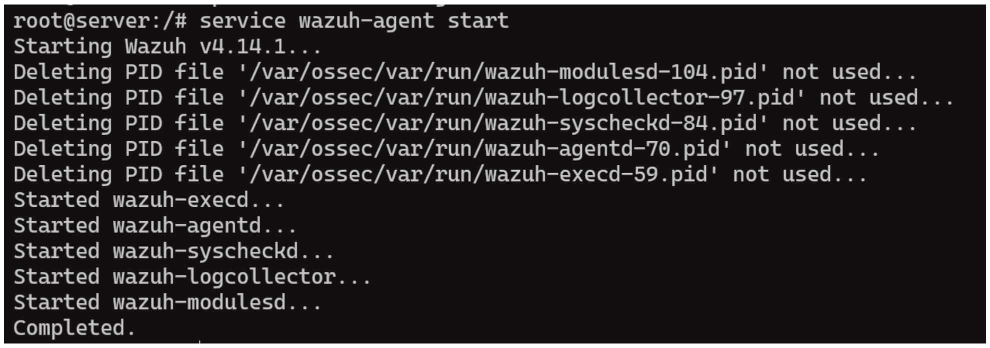
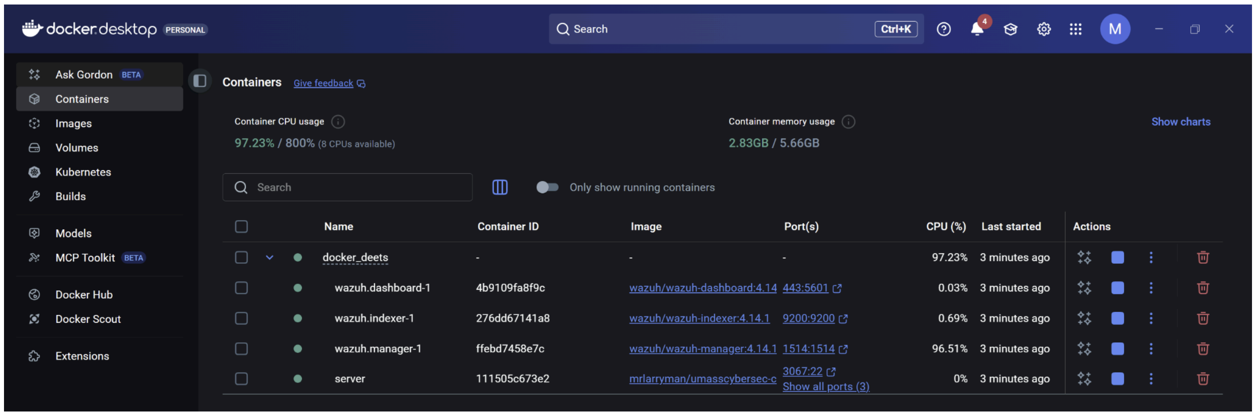
Docker

1. Turn on all the machines on docker using the UI
    a. OR go to C:\Users\Micha\Downloads\CCDC2025-2026Training-main\CCDC2025-2026Training-main\docker_deets and run this:
        i. docker-compose up -d
            1. The -d flag stands for detached mode. When you run docker-compose up without this flag, Docker keeps your terminal "attached" to the containers. This means you will see a live stream of all the logs from every container scrolling on your screen, and if you close the terminal or press Ctrl + C, all your containers will stop
2. Open terminal in there
3. To open a shell in the server machine (the one with the Wazuh Agent installed), use:
    a. docker exec -it server /bin/bash
4. To open a shell in the wazuh.manager-1 (the one with all the wazuh rules and configs):
    a. docker exec -it docker_deets-wazuh.manager-1 /bin/bash
5. First thing to do is turn on the Wazuh Agent on the server machine:
    a. docker exec -it server /bin/bash
    b. update-rc.d wazuh-agent defaults 95 10
    c. service wazuh-agent start
        i. 

```
root@server:/# service wazuh-agent start
Starting Wazuh v4.14.1...
Deleting PID file '/var/ossec/var/run/wazuh-modulesd-104.pid' not used...
Deleting PID file '/var/ossec/var/run/wazuh-logcollector-97.pid' not used...
Deleting PID file '/var/ossec/var/run/wazuh-syscheckd-84.pid' not used...
Deleting PID file '/var/ossec/var/run/wazuh-agentd-70.pid' not used...
Deleting PID file '/var/ossec/var/run/wazuh-execd-59.pid' not used...
Started wazuh-execd...
Started wazuh-agentd...
Started wazuh-syscheckd...
Started wazuh-logcollector...
Started wazuh-modulesd...
Completed.
```

    d. Check the Wazuh Overview on the UI to make sure the agent is on
6. To turn off all Docker, do this:
    a. docker compose stop

Docker Architecture Explained



Containers

Container CPU usage
97.23% / 800% (8 CPUs available)

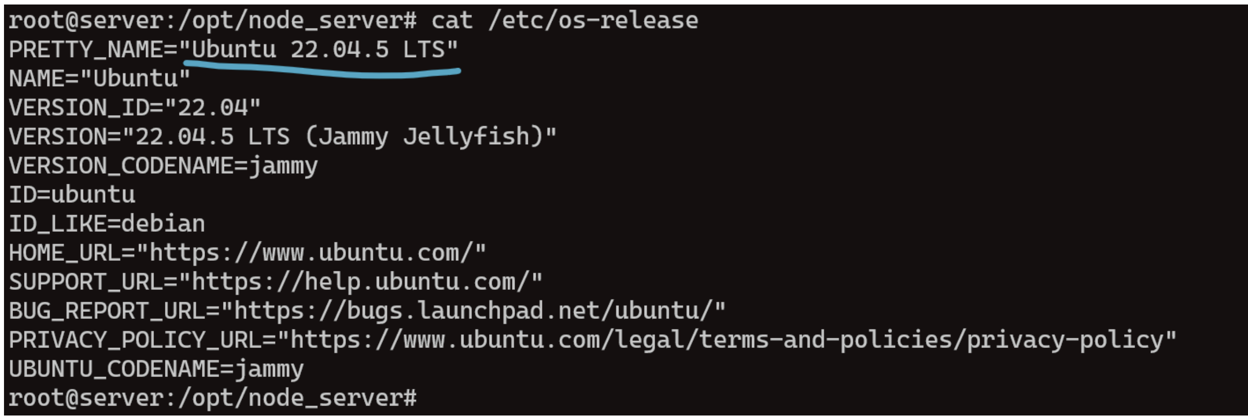
Container memory usage
2.83GB / 5.66GB

Name	Container ID	Image	Port(s)	CPU (%)	Last started
docker_deets	-	-	-	97.23%	3 minutes ago
wazuh.dashboard-1	4b9109fa8f9c	wazuh/wazuh-dashboard:4.14	443:5601	0.03%	3 minutes ago
wazuh.indexer-1	276dd67141a8	wazuh/wazuh-indexer:4.14.1	9200:9200	0.69%	3 minutes ago
wazuh.manager-1	ffebd7458e7c	wazuh/wazuh-manager:4.14.1	1514:1514	96.51%	3 minutes ago
server	111505c673e2	mrlarryman/umasscybersec-c	3067:22 Show all ports (3)	0%	3 minutes ago

- Docker_deets is the project name
- Containers:
    - **wazuh.indexer-1 (The Database)**
        - A high-performance search engine (based on OpenSearch) that stores all the security alerts. This is the "Elasticsearch" component mentioned in the diagram.
    - **wazuh.manager-1 (The Brains)**
        - The core engine. It receives logs, decodes them (JSON, Syslog, etc.), runs them against security rules, and decides if an alert should be generated.
    - **wazuh.dashboard-1 (The Interface)**
        - The web UI you log into. It doesn't store data; it just queries the **Indexer** to show you charts and alerts.
    - **server (Agent) (the eyes)**
        - This is a container (or separate VM) where the **Wazuh Agent** is installed. It watches the local system and ships logs to the **Manager**.
- Images:
    - Each of these containers have a single image which is uses. You can look at the docker compose to see that. All the Wazuh stuff have standard images that it uses, and only the "server" container uses a custom image that was made specifically for CCDC training

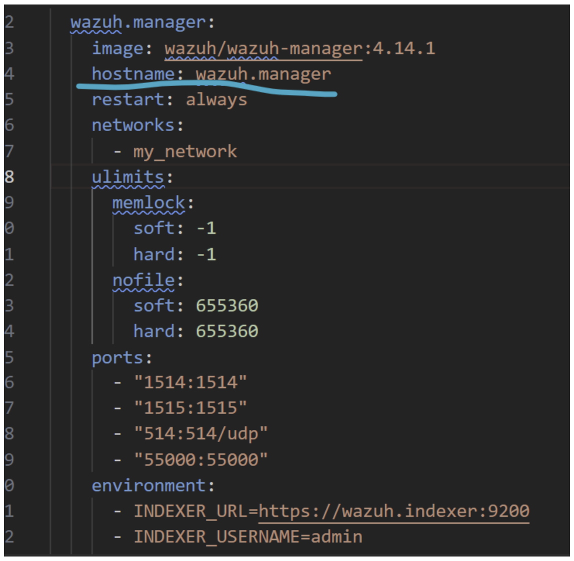
Set Up

1. So, we had to set up Wazuh. I downloaded it using the guide linked in the guide: https://documentation.wazuh.com/current/installation-guide/wazuh-agent/wazuh-agent-package-linux.html

2. One thing that I had to check was my Linux distro and version. This is because older distros downloaded differently according the guide
    a. cat /etc/os-release
    b. 

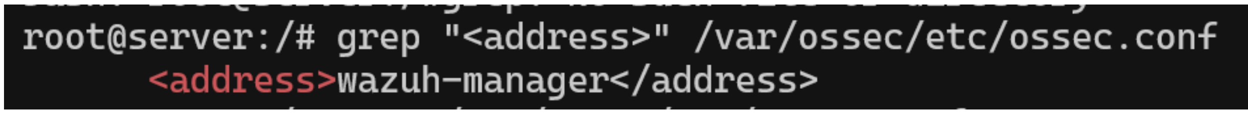
```
root@server:/opt/node_server# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.5 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
root@server:/opt/node_server#
```

3. After running the command above, I see that the server is running Ubuntu 22 so we should be good to run the up to date commands. The command below are how I got the most up to date packages for wazuh by adding the Wazuh repository
    a. docker exec -it server /bin/bash
    b. apt update
    c. apt-get install gnupg apt-transport-https
    d. curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | gpg --no-default-keyring --keyring gnupg-ring:/usr/share/keyrings/wazuh.gpg --import && chmod 644 /usr/share/keyrings/wazuh.gpg
    e. echo "deb [signed-by=/usr/share/keyrings/wazuh.gpg] https://packages.wazuh.com/4.x/apt/ stable main" | tee -a /etc/apt/sources.list.d/wazuh.list
    f. apt-get update
4. Now, I have to actually install wazuh and they provided a command for that
    a. WAZUH_MANAGER="**wazuh.manager**" apt-get install wazuh-agent
    b. To find the hostname for the manager, I looked at the Docker Compose



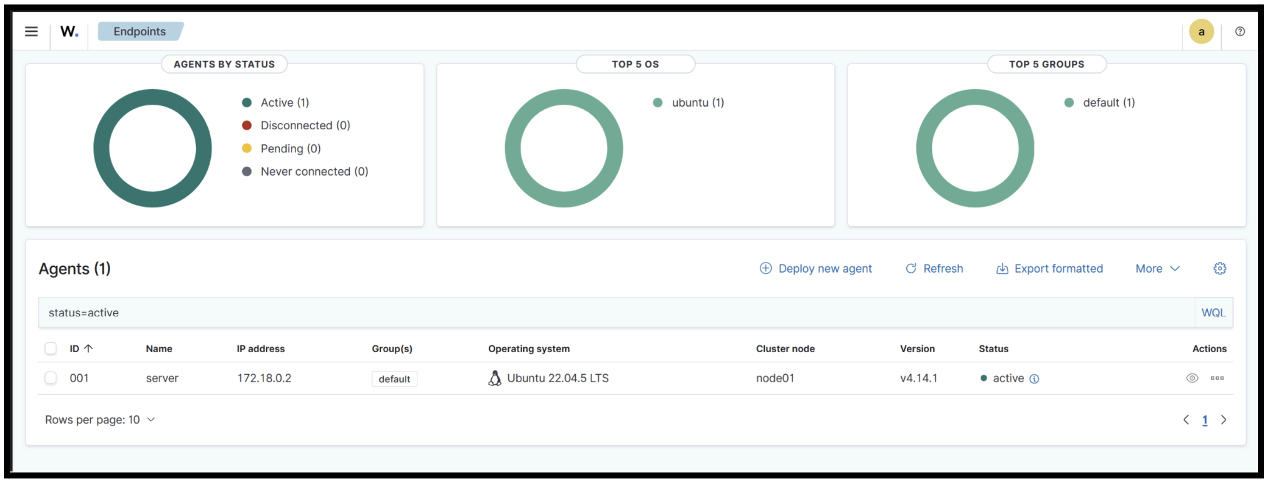
i.

```yaml
  wazuh.manager:
    image: wazuh/wazuh-manager:4.14.1
    hostname: wazuh.manager
    restart: always
    networks:
      - my_network
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 655360
        hard: 655360
    ports:
      - "1514:1514"
      - "1515:1515"
      - "514:514/udp"
      - "55000:55000"
    environment:
      - INDEXER_URL=https://wazuh.indexer:9200
      - INDEXER_USERNAME=admin
```

5. And then to start the Wazuh agent, the guide showed us how to do that using systemctl but it didn't work, as hinted by the CCDC training material. So I did some research, and I found out how to start is using the "service" command just like I did with SSH. service is an older tool.
    a. update-rc.d wazuh-agent defaults 95 10
    b. service wazuh-agent start
6. Ok. And then around here, is when my Wazuh Manager Container continued to restart every like 10 seconds, which made the dashboard unusable since the API was going down every 10 seconds. I had zero clue what was going on, and tried so many different things. I tried re-downloading everything, giving more resources to WSL, giving more resource to each container by editing the docker compose file, and so much more. I tried making new SSL certificates as the instructions hints, but that didn't work.
7. After **2 hours** of continuous debugging and no progress, I decided to try and delete all the old certificates. I deleted all of them, and tried making certificates fresh.
    a. docker-compose -f generate-indexer-certs.yml run --rm generator
8. And then it worked!! Well, the API worked. Now I had to connect it to the agent. Since it said that the agent was connected, even though the console made it seem like the Agent was deployed
9. So, after debugging that problem for a while, my best guess was that possibly the value used in the WAZAH_MANAGER value was wrong. As a reminder, I did WAZUH_MANAGER="**wazuh.manager**". So after some digging online, I found that we should find the appropriate WAZUH_MANAGER value by using this command:
    a. grep "<address>" /var/ossec/etc/ossec.conf
    b. 

```
root@server:/# grep "<address>" /var/ossec/etc/ossec.conf
    <address>wazuh-manager</address>
```

10. And so as you can see, the hostname wasn't wazuh.manager but rather wazuh-manager! So, I changed that line in the address field and restarted the agent
    a. service wazuh-agent restart

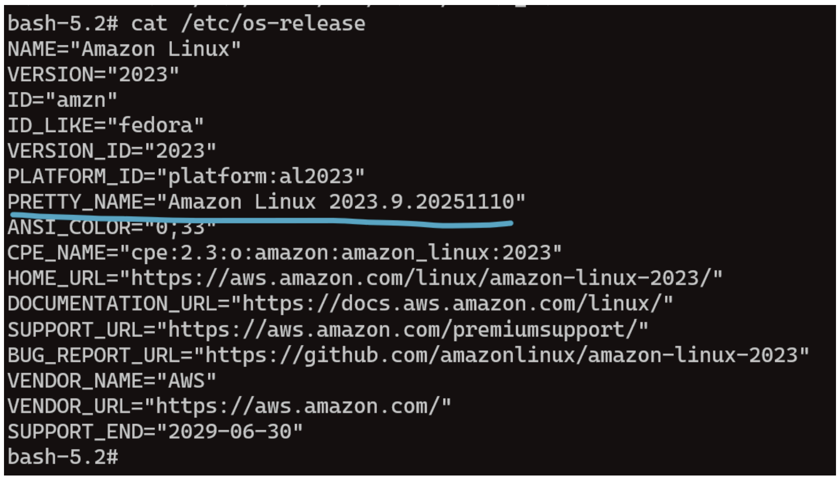
11. 

Agents (1)

status=active

ID	Name	IP address	Group(s)	Operating system	Cluster node	Version	Status	Actions
001	server	172.18.0.2	default	Ubuntu 22.04.5 LTS	node01	v4.14.1	active	

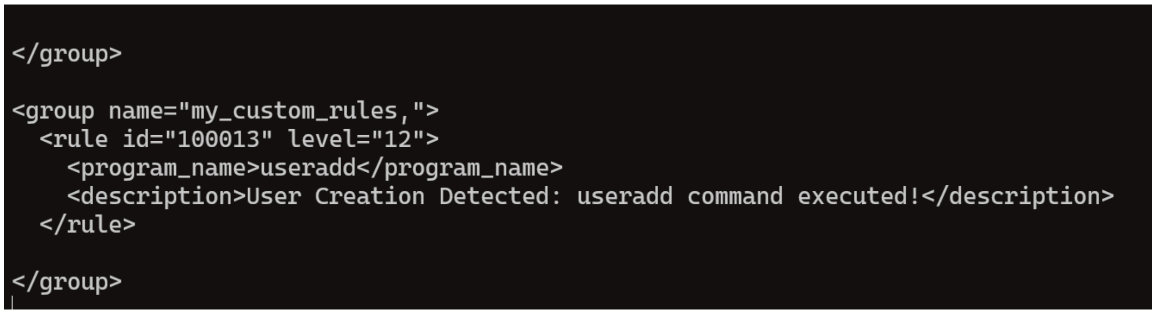
    a. My agent is finally up!



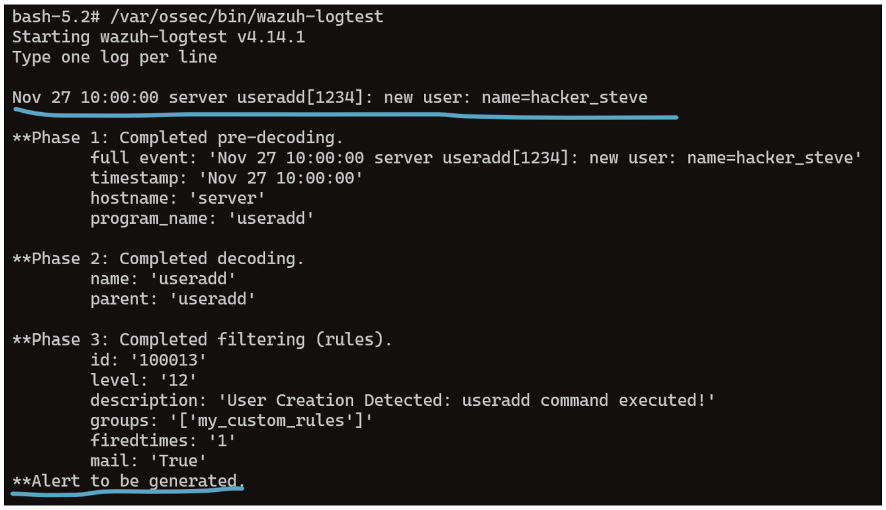
```
bash-5.2# cat /etc/os-release
NAME="Amazon Linux"
VERSION="2023"
ID="amzn"
ID_LIKE="fedora"
VERSION_ID="2023"
PLATFORM_ID="platform:al2023"
PRETTY_NAME="Amazon Linux 2023.9.20251110"
ANSI_COLOR="0;33"
CPE_NAME="cpe:2.3:o:amazon:amazon_linux:2023"
HOME_URL="https://aws.amazon.com/linux/amazon-linux-2023/"
DOCUMENTATION_URL="https://docs.aws.amazon.com/linux/"
SUPPORT_URL="https://aws.amazon.com/premiumsupport/"
BUG_REPORT_URL="https://github.com/amazonlinux/amazon-linux-2023"
VENDOR_NAME="AWS"
VENDOR_URL="https://aws.amazon.com/"
SUPPORT_END="2029-06-30"
bash-5.2#
```

Implementing Rules

12. Now it's time to do the extra credit, and implement some rules. This one actually took a while, since I was having problems actually triggering the rules. And then I did some research, and there's a custom tool that they have to simulate whether an event was triggered based on logs.

13. So, the rule I created was to see if the useradd command used. This is because as a penetration tester, I know one common strategy is persistence. This means after getting initial access (ex. Reverse shell), you will want to create a permanent back door, and one method is to create a new user so you can continuously log back instead of having re-do the exploit again to get a reverse shell.
14. So to create a new rule, I had to log in to this:
    a. docker exec -it docker_deets-wazuh.manager-1 /bin/bash
    b. nano /var/ossec/etc/rules/local_rules.xml
        i. 

```
</group>

<group name="my_custom_rules,">
  <rule id="100013" level="12">
    <program_name>useradd</program_name>
    <description>User Creation Detected: useradd command executed!</description>
  </rule>

</group>
```

15. And then to test it out, you have to run this custom script (/var/ossec/bin/wazuh-logtest) and it will ask for one line as input and you put in an example log to see if it would trigger an event and mine does:
    a. 

```
bash-5.2# /var/ossec/bin/wazuh-logtest
Starting wazuh-logtest v4.14.1
Type one log per line

Nov 27 10:00:00 server useradd[1234]: new user: name=hacker_steve

**Phase 1: Completed pre-decoding.
        full event: 'Nov 27 10:00:00 server useradd[1234]: new user: name=hacker_steve'
        timestamp: 'Nov 27 10:00:00'
        hostname: 'server'
        program_name: 'useradd'

**Phase 2: Completed decoding.
        name: 'useradd'
        parent: 'useradd'

**Phase 3: Completed filtering (rules).
        id: '100013'
        level: '12'
        description: 'User Creation Detected: useradd command executed!'
        groups: '['my_custom_rules']'
        firedtimes: '1'
        mail: 'True'
**Alert to be generated.
```

16. And as you can see from the bottom, an event was triggered!
17. The resource I used was this:
    a. https://documentation.wazuh.com/current/user-manual/ruleset/rules/custom.html#custom-rules
    b. https://documentation.wazuh.com/current/user-manual/ruleset/decoders/custom.html

Note: if I want to create a new rule folder do this:

- # 1. Give ownership to the Wazuh user
- chown wazuh:wazuh /var/ossec/etc/rules/**competition_rules.xml**

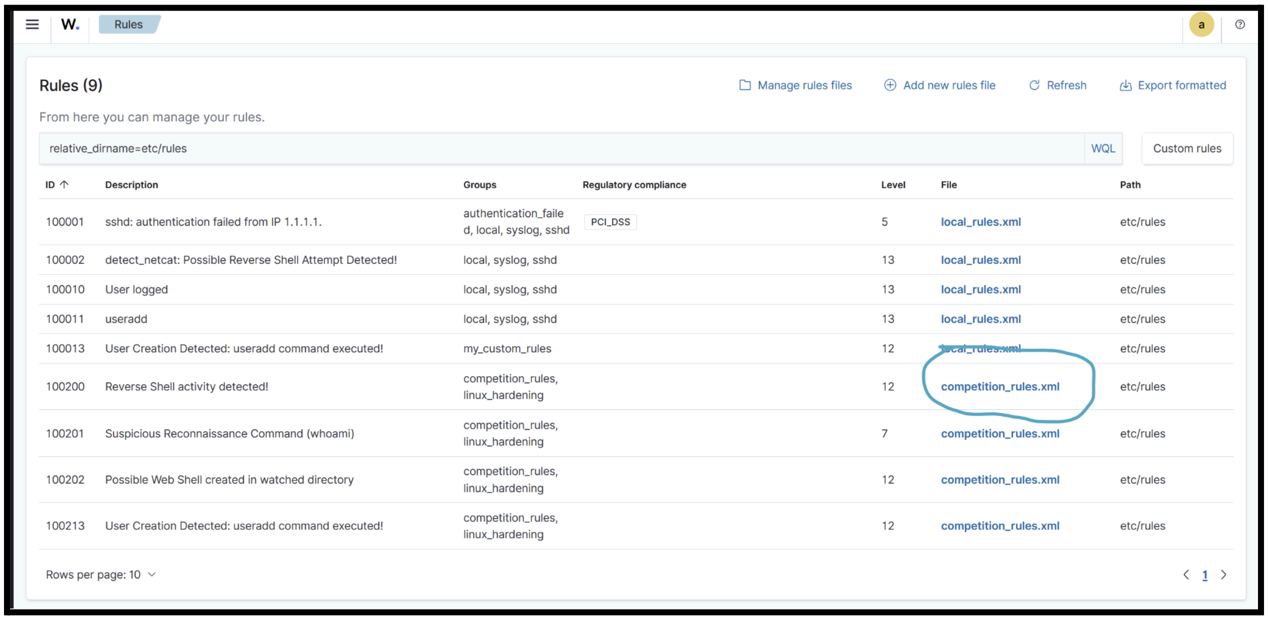
- # 2. Give read permissions
- chmod 660 /var/ossec/etc/rules/**competition_rules.xml**

Example rule set:

```
<group name="linux, competition,">

 <rule id="100100" level="12">
  <decoded_as>auditd</decoded_as>
  <match>nc -e|nc.traditional -e|ncat -e|bash -i >&|python -c 'import socket|socat exec</match>
  <description>Reverse Shell activity detected!</description>
 </rule>

 <rule id="100101" level="7">
  <decoded_as>auditd</decoded_as>
  <match>exe="/usr/bin/whoami"|exe="/usr/bin/id"</match>
  <description>Suspicious Reconnaissance Command (whoami/id)</description>
 </rule>

 <rule id="100102" level="12">
  <if_group>syscheck</if_group>
  <match>.php|.jsp|.phtml|.php5</match>
  <description>Possible Web Shell created in watched directory</description>
 </rule>

 <rule id="100103" level="10">
  <decoded_as>auditd</decoded_as>
  <match>rm .bash_history|ln -sf /dev/null .bash_history|truncate -s 0 .bash_history</match>
  <description>User attempted to clear command history</description>
 </rule>

 <rule id="100104" level="10">
  <if_group>syscheck</if_group>
  <match>/etc/cron</match>
  <description>Cron job configuration modified (Persistence risk)</description>
 </rule>
```

</group>

How to test for rules better



Rules (9)

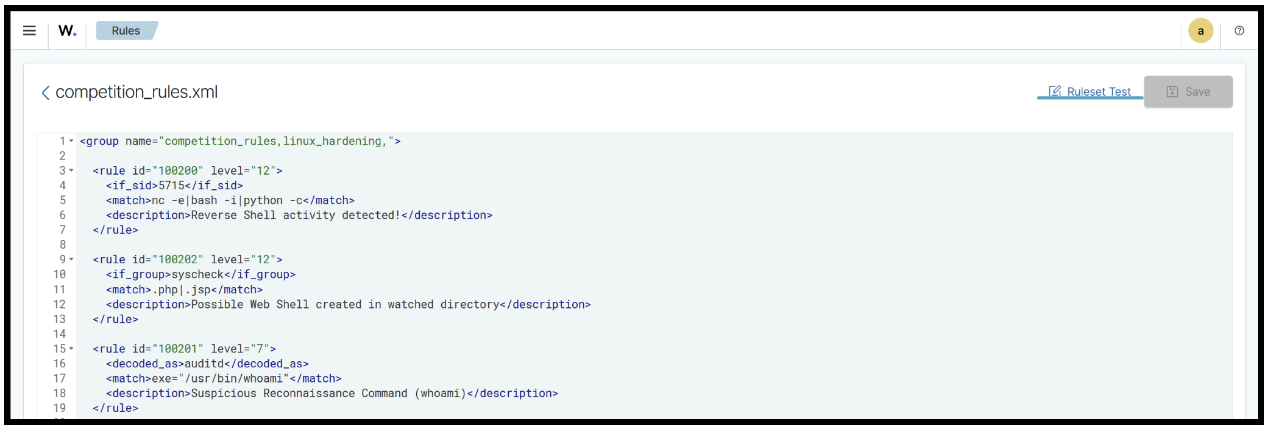
From here you can manage your rules.

relative_dirname=etc/rules

ID	Description	Groups	Regulatory compliance	Level	File	Path
100001	sshd: authentication failed from IP 1.1.1.1.	authentication_failed, local, syslog, sshd	PCI_DSS	5	local_rules.xml	etc/rules
100002	detect_netcat: Possible Reverse Shell Attempt Detected!	local, syslog, sshd		13	local_rules.xml	etc/rules
100010	User logged	local, syslog, sshd		13	local_rules.xml	etc/rules
100011	useradd	local, syslog, sshd		13	local_rules.xml	etc/rules
100013	User Creation Detected: useradd command executed!	my_custom_rules		12	local_rules.xml	etc/rules
100200	Reverse Shell activity detected!	competition_rules, linux_hardening		12	competition_rules.xml	etc/rules
100201	Suspicious Reconnaissance Command (whoami)	competition_rules, linux_hardening		7	competition_rules.xml	etc/rules
100202	Possible Web Shell created in watched directory	competition_rules, linux_hardening		12	competition_rules.xml	etc/rules
100213	User Creation Detected: useradd command executed!	competition_rules, linux_hardening		12	competition_rules.xml	etc/rules

Rows per page: 10

- Click on the file name on the right



competition_rules.xml

```
<group name="competition_rules,linux_hardening,">

  <rule id="100200" level="12">
    <if_sid>5715</if_sid>
    <match>nc -e|bash -i|python -c</match>
    <description>Reverse Shell activity detected!</description>
  </rule>

  <rule id="100202" level="12">
    <if_group>syscheck</if_group>
    <match>.php|.jsp</match>
    <description>Possible Web Shell created in watched directory</description>
  </rule>

  <rule id="100201" level="7">
    <decoded_as>auditd</decoded_as>
    <match>exe="/usr/bin/whoami"</match>
    <description>Suspicious Reconnaissance Command (whoami)</description>
  </rule>
```

- Click on "Ruleset Test" on the top right

- Then ask gemini to generate some examples of logs

Falco and Wazuh

https://keefeere.me/blog/falco_wazuh/

So Falco is made to be used to watch containers. This is because containers often die every like 10 minutes and there are so many fast things going on so a typical wazuh agent won't suffice. You need special technology like Falco. I'm actually monitoring the "server" container using two different methods: wazuh-agent (installed on server) and Falco. But as I'll explain below, the point of Falco is to have a more optimized way to monitor containers without wazuh-agent. So in an ideal world, we wouldn't have a wazuh-agent here since we are monitoring a container (server). We want to just use Falco when monitoring containers.

Here's a link to Gemini chat which explained everything and helped me follow article and modify it from Kubernetes to Docker: https://gemini.google.com/share/754d8bc394e8

Here's a link to Gemini chat which explained how Docker and Kubernetes work and how containers are cloud-native: https://gemini.google.com/share/bff2a0ffc4bc

So how does Falco work? Falco is a separate container that is meant to be on the same host as the containers that you want to monitor. And so that's why we needed to add a Falco container to my docker set up. And unlike wazuh-agent, Falco gets log data from the KERNEL level. This is because the container that I want to monitor (server) is using the same kernel as Falco since they are on the same host. That's how Docker works. And so since they use the same kernel, Falco can see all the changes being made on the server container (the container I'm monitoring) and that's how it sees when files are changes and such. So in other words, you don't need to install any Falco software bloat on the containers you want to monitor since it's all kernel level monitoring. But for wazuh-agents, you would need to manually install wazuh-agent on each container which could not scale well and the point of containers is to be able to scale LARGE.

The Problem with Agents in Containers
In a perfect "Cloud Native" world, containers are supposed to be lean and fast.

- Bloat: Installing a Wazuh agent inside every container (Web server, Database, Redis, etc.) makes your images huge and uses a lot of extra RAM/CPU.
- Maintenance: If you have 50 containers, you now have 50 agents to update, 50 sets of keys to manage, and 50 extra points of failure.
- Blind Spots: An agent inside a container is restricted by that container's boundaries. If a hacker performs a "container breakout" to attack the host, the agent inside the container might not even see it happening.

Even in a massive Kubernetes Cluster spanning hundreds of physical machines, the fundamental logic remains the same: Falco and the containers it watches must be on the same host (node) to share a kernel
Here is how Kubernetes handles this differently than your single-machine Docker setup:

- The DaemonSet Solution: Kubernetes uses a controller called a DaemonSet. Its sole job is to ensure that one copy of the Falco pod runs on every single machine (node) in your cluster.
- Local Guarding: If you have 10 nodes, you have 10 Falco "guards." The Falco pod on "Node A" only watches the containers running on "Node A" because those are the only ones sharing its local kernel.
- Centralized Reporting: While there are many "guards" spread across different machines, they all send their alerts back to the same central Wazuh Manager. This gives you a "single pane of glass" view of your entire cluster's security.

Ok so basically, the article uses Kubernetes, but we have a Wazuh set up in Docker so we can just edit the Docker Compose file to make it so that the Falco container can communicate with the other containers. Please look at the Gemini chat link above for a more in-depth explanation of the process to set this up.

Agents on Windows

https://documentation.wazuh.com/current/installation-guide/wazuh-agent/wazuh-agent-package-windows.html

1. Download the Windows installer to start the installation process.

2. Deploy Wazuh Agent
Choose one of the command shell alternatives to deploy the Wazuh agent on your endpoint. Run the command below and replace the **WAZUH_MANAGER** value with your Wazuh manager IP address or hostname. Ensure the downloaded Wazuh agent installation file is in your working directory.

Using CMD: wazuh-agent-4.14.1-1.msi /q WAZUH_MANAGER="**10.0.0.2**"

Using Powershell: .\wazuh-agent-4.14.1-1.msi /q WAZUH_MANAGER="**10.0.0.2**"

For additional deployment options such as agent name, agent group, and registration password, see the Deployment variables for Windows section.

Start the Wazuh agent from the GUI or by running:

Using CMD: `NET START WazuhSvc`

Using PowerShell: `Start-Service wazuhsvc`

By default, all agent files are stored in `C:\Program Files (x86)\ossec-agent` after the installation.

The installation process is now complete and the Wazuh agent is successfully installed and configured.

Note: Alternatively, if you want to install an agent without enrolling it, omit the deployment variables. To learn more about the different enrollment methods, see the Wazuh agent enrollment section.

Script:

```
# Check for Admin Privileges
if (-NOT
([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsIn
Role([Security.Principal.WindowsBuiltInRole] "Administrator")) {
   Write-Warning "Please run this script as an Administrator."
   break
}

$msiUrl = "https://packages.wazuh.com/4.x/windows/wazuh-agent-4.14.1-1.msi"
$msiPath = "$env:TEMP\wazuh-agent-4.14.1-1.msi"
$managerIp = "10.0.0.2"

Write-Host "Downloading Wazuh Agent..." -ForegroundColor Cyan
Invoke-WebRequest -Uri $msiUrl -OutFile $msiPath

Write-Host "Installing Wazuh Agent..." -ForegroundColor Cyan
# This runs the installer using the path we defined above
Start-Process msiexec.exe -ArgumentList "/i `"$msiPath`" /q
WAZUH_MANAGER=`"$managerIp`"" -Wait
```

```
Write-Host "Starting Wazuh Service..." -ForegroundColor Cyan
# Reverted to the original command from your guide
Start-Service wazuhsvc

Write-Host "Installation and Configuration Complete!" -ForegroundColor Green
```

Agents on Active Directory

Official Documentation:

- LDAP Integration
- Detecting AD Attacks Part 1
- Detecting AD Attacks Part 2
- Installing Wazuh Agent via GPO

Installation Script for Agents on AD Machines (DC and Workstations):

```
# Check for Admin Privileges (Kept from your original)
if (-NOT
([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsIn
Role([Security.Principal.WindowsBuiltInRole] "Administrator")) {
   Write-Warning "Please run this script as an Administrator."
   break
}

# 1. SET VARIABLES (Inspired by your style)
$managerIp = "10.0.0.2"
$tempDir   = "$env:TEMP\WazuhInstall"
New-Item -ItemType Directory -Force -Path $tempDir | Out-Null

$wazuhMsiUrl = "https://packages.wazuh.com/4.x/windows/wazuh-agent-4.14.1-1.msi"
$sysmonUrl   = "https://download.sysinternals.com/files/Sysmon.zip"
```

```
$sysmonConf =
"https://raw.githubusercontent.com/SwiftOnSecurity/sysmon-config/master/sysmonconfig-export
.xml"

# 2. DOWNLOAD EVERYTHING (Using your Invoke-WebRequest method)
Write-Host "Downloading Wazuh and Sysmon..." -ForegroundColor Cyan
Invoke-WebRequest -Uri $wazuhMsiUrl -OutFile "$tempDir\wazuh-agent.msi"
Invoke-WebRequest -Uri $sysmonUrl -OutFile "$tempDir\Sysmon.zip"
Invoke-WebRequest -Uri $sysmonConf -OutFile "$tempDir\sysmonconfig.xml"

# 3. INSTALL WAZUH AGENT (Your trusted command)
Write-Host "Installing Wazuh Agent..." -ForegroundColor Cyan
Start-Process msiexec.exe -ArgumentList "/i `"$tempDir\wazuh-agent.msi`" /q
WAZUH_MANAGER=`"$managerIp`"" -Wait

# 4. INSTALL SYSMON (Needed for Part 2 of the AD articles)
Write-Host "Installing Sysmon..." -ForegroundColor Cyan
Expand-Archive -Path "$tempDir\Sysmon.zip" -DestinationPath "$tempDir\Sysmon" -Force
Start-Process "$tempDir\Sysmon\Sysmon64.exe" -ArgumentList "-accepteula -i
`"$tempDir\sysmonconfig.xml`"" -Wait

# 5. ENABLE AD AUDIT POLICIES (Needed for Part 1 of the AD articles)
Write-Host "Enabling Advanced Auditing..." -ForegroundColor Cyan
auditpol /set /subcategory:"Directory Service Access" /success:enable
auditpol /set /subcategory:"Kerberos Service Ticket Operations" /success:enable
auditpol /set /subcategory:"Logon" /success:enable /failure:enable

# 6. CONFIGURE WAZUH TO READ SYSMON
Write-Host "Updating Wazuh Config..." -ForegroundColor Cyan
$OssecConf = "C:\Program Files (x86)\ossec-agent\ossec.conf"
$SysmonBlock = @"
  <localfile>
    <location>Microsoft-Windows-Sysmon/Operational</location>
    <log_format>eventchannel</log_format>
  </localfile>
"@
# Inserts the Sysmon reading block into the ossec.conf file
(Get-Content $OssecConf) -replace '</ossec_config>', "$SysmonBlock`n</ossec_config>" |
Set-Content $OssecConf
```

```
# 7. START SERVICES (Your preferred command)
Write-Host "Starting Wazuh Service..." -ForegroundColor Cyan
Restart-Service wazuhsvc  # Use Restart to ensure it picks up the new config

Write-Host "AD Hardening & Monitoring Complete!" -ForegroundColor Green
```

Script to add AD Rules to Wazuh Manager

Run this script on the Wazuh Manager

```
#!/bin/bash

# Check for root privileges
if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root (use sudo)"
   exit 1
fi

RULES_FILE="/var/ossec/etc/rules/local_rules.xml"

echo "Adding Active Directory detection rules to $RULES_FILE..."

# Append the rules in the specific order requested
cat <<EOF >> $RULES_FILE

<group name="security_event, windows,">

  <rule id="110001" level="12">
    <if_sid>60103</if_sid>
    <field name="win.system.eventID">^4662$</field>
    <field name="win.eventdata.properties"
type="pcre2">{1131f6aa-9c07-11d1-f79f-00c04fc2dcd2}|{19195a5b-6da0-11d0-afd3-00c04fd9
30c9}</field>
    <options>no_full_log</options>
    <description>Directory Service Access. Possible DCSync attack</description>
  </rule>

 <rule id="110009" level="0">
```

```
    <if_sid>60103</if_sid>
    <field name="win.system.eventID">^4662$</field>
    <field name="win.eventdata.properties"
type="pcre2">{1131f6aa-9c07-11d1-f79f-00c04fc2dcd2}|{19195a5b-6da0-11d0-afd3-00c04fd9
30c9}</field>
    <field name="win.eventdata.SubjectUserName" type="pcre2">\\$$</field>
    <options>no_full_log</options>
    <description>Ignore all Directory Service Access that is originated from a machine account
containing $</description>
  </rule>

  <rule id="110002" level="12">
    <if_sid>60103</if_sid>
    <field name="win.system.eventID">^4769$</field>
    <field name="win.eventdata.TicketOptions" type="pcre2">0x40810000</field>
    <field name="win.eventdata.TicketEncryptionType" type="pcre2">0x17</field>
    <options>no_full_log</options>
    <description>Possible Keberoasting attack</description>
  </rule>

  <rule id="110003" level="12">
    <if_sid>60103</if_sid>
    <field name="win.system.eventID">^4624$</field>
    <field name="win.eventdata.LogonGuid"
type="pcre2">{00000000-0000-0000-0000-000000000000}</field>
    <field name="win.eventdata.logonType" type="pcre2">3</field>
    <options>no_full_log</options>
    <description>Possible Golden Ticket attack</description>
  </rule>

  <rule id="110004" level="12">
    <if_sid>61600</if_sid>
    <field name="win.system.eventID" type="pcre2">17|18</field>
    <field name="win.eventdata.PipeName" type="pcre2">\\\\PSEXESVC</field>
    <options>no_full_log</options>
    <description>PsExec service launched for possible lateral movement within the
domain</description>
  </rule>

  <rule id="110006" level="12">
```

```
    <if_group>sysmon_event1</if_group>
    <field name="win.eventdata.commandLine" type="pcre2">NTDSUTIL</field>
    <description>Possible NTDS.dit file extraction using ntdsutil.exe</description>
  </rule>

  <rule id="110007" level="12">
    <if_sid>60103</if_sid>
    <field name="win.system.eventID">^4624$</field>
    <field name="win.eventdata.LogonProcessName" type="pcre2">seclogo</field>
    <field name="win.eventdata.LogonType" type="pcre2">9</field>
    <field name="win.eventdata.AuthenticationPackageName" type="pcre2">Negotiate</field>
    <field name="win.eventdata.LogonGuid"
type="pcre2">{00000000-0000-0000-0000-000000000000}</field>
    <options>no_full_log</options>
    <description>Possible Pass the hash attack</description>
  </rule>

  <rule id="110008" level="12">
    <if_sid>61612</if_sid>
    <field name="win.eventdata.TargetImage"
type="pcre2">(?i)\\\\\\\\system32\\\\\\\\lsass.exe</field>
    <field name="win.eventdata.GrantedAccess" type="pcre2">(?i)0x1010</field>
    <description>Possible credential dumping using mimikatz</description>
  </rule>

</group>
EOF

echo "Restarting Wazuh Manager to apply changes..."
systemctl restart wazuh-manager

echo "Done!"
```

How to download Wazuh Indexer, Server, and Dashboard

```
curl -sO https://packages.wazuh.com/4.14/wazuh-install.sh && sudo bash ./wazuh-install.sh -a
```