**Wazuh - Search Query**

For this lesson, we will be using a live lab environment containing a Linux server running the **Wazuh SIEM / XDR** and **a Windows server** with the **Wazuh agent installed**.

Before getting started, we will need to connect the two systems and verify that they can communicate. We will begin the lab on the Wazuh server.

1. From the Machines tab, we will connect to the **Wazuh server**.
2. From the dock at the bottom of the screen, open a **Terminal Emulator** window.
3. At the command prompt, we will run command: **ip a** to display the network configuration for the Wazuh server. A screenshot of a computer

   AI-generated content may be incorrect.

Take note of the IP address. We will need this information shortly.

1. From the Machines tab, connect to the **Windows server**.
2. Start a **Windows PowerShell (Admin)**.
3. At the prompt, we will run **notepad C:\Windows\System32\drivers\etc\hosts** to open the etc/hosts file in Notepad.
4. In the etc/hosts file, replace **the IP Address for Wazuh Server** with the IP address we pulled from the \etc\hosts file. A computer screen shot of a computer program

   AI-generated content may be incorrect.
5. Now we will ping Wazuh to confirm we can reach the Wazuh server by name.
6. Then we will need to bounce the wazuh agent, run command: **net stop wazuh**.
7. Then run **net start wazuh** to re-start the Wazuh agent.
8. Now we will capture the IP Address of the windows server for later in the lab. Run command: **ipconfig**. A computer screen with white text

   AI-generated content may be incorrect.

**Part 1: Explore Events in Wazuh**

In the next steps, we will connect to Wazuh and explore the Security Events dashboard.

1. **Log in** to wazuh web application. A screenshot of a computer

   AI-generated content may be incorrect.
2. On the Wazuh dashboard, be sure we see **1 Total agents and 1 Active agents**. A screenshot of a computer

   AI-generated content may be incorrect.

3. On the Wazuh dashboard, click the **menu** and select **Discover**.

In the Discover dashboard, we will see our selected ***data source*** (a.k.a. "index pattern") is **wazhu-alerts-**\* . This data source **contains security alert data**, and it is the one we want to work with. Wazuh is ingesting logs from a single remote Windows server and the Wazuh (Ubuntu) server.

1. Under *Available fields*, scroll down to view all the fields we can use to construct search expressions. A screenshot of a computer

   AI-generated content may be incorrect.

Notice that many fields start with "**data.win**". **This data comes from the Wazuh agent installed on the Windows server.**

**A screenshot of a computer

AI-generated content may be incorrect.**

*The* ***two most prevalent data.win fields*** *are eventdata and system.*

**Part 2: Apply Search Expressions**

In this part of the lab, we will apply a series of search expressions to surface different alerts.

**Note:**

The wildcard (\*) denotes any field starting with "data.win". Wildcards are usable in multiple query languages, although their specific implementation may vary slightly.

1. In the Data Table, click the **> symbol** to expand the first Windows alert. A screenshot of a computer

   AI-generated content may be incorrect.

Under *Expanded Document*, note all the Windows-related fields recorded in this one alert.

In Wazuh, some of the more useful Windows search fields are shown below: A screenshot of a computer program

AI-generated content may be incorrect.

**Note:** Not every alert will have every field listed above.

Let's try to find Windows Events with an "**ERROR**" severity. As seen in the image above, the field for that is "**data.win.system.severityValue**".

1. In the Search Bar, replace the existing query with **data.win.system.severityValue:ERROR** and run to show **Windows Event log errors**.

**Note:** Whenever we make *any change* to the search query or the search time frame be sure to click **Refresh** or **Update** button.

Notice that Wazuh highlights the term "ERROR" in the Data Table. A screenshot of a chat

AI-generated content may be incorrect.

**Tip:**

Different SIEMs implement case sensitivity in different ways. When constructing your search query we should always verify whether the search we're running requires case sensitive definition of search terms or values.

Sometimes, we want to look for keywords, or just specific terms outside of the context of a specific field. For example, we may want to see if the alerts record any failures. For that, we can use a *simple*

search expression and wildcards.

1. In the Search Bar, we will replace the *entire* existing query with \****fail\****  **Update**  to show all records containing “fail”, “failing”, “failed”, “failure”, etc.

**Note**: If we see "Expand your time range" it just means that the query is not found (yet) in the current time range. We can wait a *few minutes.*

and click Refresh, or we can move on.

We may not see any failures in the data for the last 24 hours. Let's adjust the date and time to include more data.

1. Using the Date and Time control select **Last 24 Hours**, then click **A Day Ago**, then click **Absolute** and select **May 1, 2023** as the Start Date.
2. In the Date and Time control, click **Now**, then click **Absolute** and select **May 1, 2024** as the End Date, then click **Update**.

We should now see quite a few historical failure events.

1. In the Search Bar, let’s append **and @timestamp > "2023-11-21T00:00:00"** and **Update**. A screenshot of a computer

   AI-generated content may be incorrect.

We have added a new *conditional* argument in the query. This query asks for every record containing "fail" *after* midnight on November 21, 2023.

In this context, the **">" symbol means “after”**, but **it can also mean "greater than" for numerical comparisons**. Conversely, the **"<" symbol means “before” or "less than"**. The **"@" symbol is used to identify a field created by the SIEM itself when it ingested the event**. This is not a field from within the event itself.

We should see over a thousand hits for *fail* in this time range. Something interesting generated all these failures, but we will revisit this event later in the lab.

Let's shift our focus and look for successful Windows login events.

1. In the Search Bar, replace the existing query with **data.win.system.eventID:4624** and **Update** to search for all successful Windows logins in our date and time range. A screenshot of a computer

   AI-generated content may be incorrect.

We should see hits for Windows Event ID 4624.

1. In the Search Bar, add **and data.win.eventdata.targetUserName:administrator** to the existing query and click Update.

In this query, we ask for all events with ID 4624, whose user name is "administrator”. A screenshot of a web page

AI-generated content may be incorrect.

**Note:** Again, if we see "Expand wer time range" it simply means that no events match wer search query in the time window specified. It does not necessarily mean our query is wrong. A screenshot of a computer

AI-generated content may be incorrect.

1. In the Search Bar, let’s change the value for data.win.system.targetUserName to **SYSTEM** and **Update**.

Are there any hits?

Are there currently *any* successful Windows logins that **are *not* SYSTEM**? We can use the Boolean **not** to find out.

1. In the Search Bar, add **not** after "and" and Update. A screenshot of a web page

   AI-generated content may be incorrect.

We did not receive any hits, only the "Expand we time range" message.

1. Change the Date back to **Last 24 hours**.
2. Change the query to **data.win.system.eventID:4624 and data.win.eventdata.targetUserName:cybrary**
3. Open a **terminal** window, then run **remmina &** to launch the Remmina RDP client. A screenshot of a computer

   AI-generated content may be incorrect.
4. In the Remmina Remote Desktop Client window, type ***the IP address of the Windows server*** in the address bar at the top, RDP to the Windows server.

When prompted, select **Yes** to accept the certificate.

1. At the authentication screen, enter the following credentials, to log in. A screenshot of a login box

   AI-generated content may be incorrect.A screenshot of a computer

   AI-generated content may be incorrect.

Username: **cybrary**

Password: **CLABL3arnersRock!1#$**

1. In Firefox, return to the **Wazuh Discovery dashboard** and **Reload** to refresh the page. A screenshot of a computer

   AI-generated content may be incorrect.

We should now see at least one alert for the successful cybrary user login.

1. In the Data Table, expand an **alert**, then scroll down to the **rule.level** field. A screenshot of a computer

   AI-generated content may be incorrect.

*Note the****rule.level for the alert****.*

When Wazuh ingests a log and creates an alert, it ranks the severity of the alert on a scale of 1 to 13, with 13 being the most serious. This ranking is stored in the rule.level field for every alert.

We noticed that the rule.level for a simple remote login is higher than expected. Why does a simple remote login justify even a moderate severity? The answer is found in this alert's "rule.description" field in the Data Table.

Remote logins can be part of a longer cyber kill chain (tactics chained together to compromise a network). It's important to note that a single remote connection is not a de facto security issue. As an analyst, we must correlate this alert with other alerts and events to build a picture of what *may* be an attack or breach of policy.

Let's go back to our historical data and change our DQL query to find SIEM alerts with a rule.level greater than 5. A screenshot of a computer

AI-generated content may be incorrect.

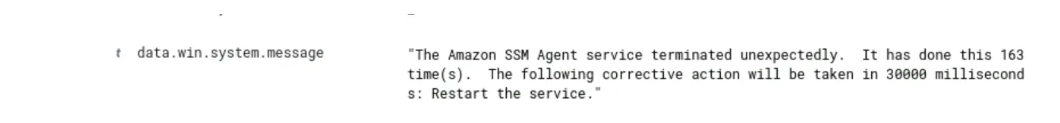
We should see several hundred alerts.

1. Try **increasing** the alert level to find the most severe alerts in the data. A screenshot of a web page

   AI-generated content may be incorrect.

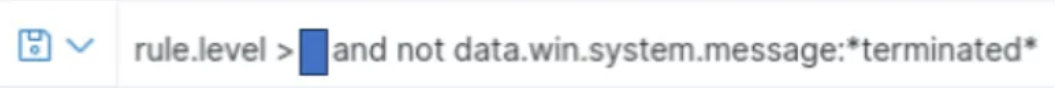
*We found that the****highest rule.level is 10.***

1. When we have found the highest rule.level, look through the **alerts** in the Data Table.

We will see that many of them deal with a buggy **Amazon SSM Agent**. Take note of this alert's "data.win.system.message" field. 

As seen earlier, "data.win.system.message" is the actual text of a Windows Event.

We can filter out these particular alerts using wildcards and a Boolean.

1. Add **and not data.win.system.message:*terminated*** to the existing query to filter out these alerts. 

We are asking for the most severe alerts in this data set that do not have "terminated" in the Windows Event message. **We are using a Boolean “not” to clear out the noise.**

Explore the remaining alerts in the Data Table. We will find that the vast majority of what we have left contains a "data.win.system.severityValue" of **AUDIT\_FAILURE**, a "data.win.system.message" stating **"An account failed to log on"**

and a "data.win.system.eventID" of **4625**. A screenshot of a computer

AI-generated content may be incorrect.

In other words, there are a lot of login failures. An analyst should always investigate multiple login failures when they happen within a short time frame.

**Part 3: Format Search Output as a Table**

Viewing search results without any additional formatting can be difficult. By default, the output of a search shows the entirety of each record along with field names. With larger log records, this is tough to read. Fortunately, we can narrow the output to specific fields and have them displayed in a table for an easier view.

For example, instead of this: A screenshot of a computer

AI-generated content may be incorrect.

We could see something like this: A screenshot of a computer

AI-generated content may be incorrect.

As we can see, the example output formatted as a table is easier to read. To reconfigure our search output in table format, we simply need to select fields to include as columns. Let's give it a try.

1. In the output of our most recent search, scroll down and review the **Available fields** listed on the left side of the UI in Discover.
2. Hover wer cursor over any **field of interest**, then click the **+ symbol** that appears next to the field name. A close up of a screen

   AI-generated content may be incorrect.
3. We will repeat the step above with **two more fields** of our choice to construct a table.

Sometimes we will add a field to our output display that we later want to remove. Let's give this a try.

A close-up of a black square

AI-generated content may be incorrect.

1. Hover your cursor over the **third column header** in wer table, then click the **X symbol** to remove a column from wer table.

The columns we may need to display will vary based upon the analysis questions we are going to answer. When working in a SIEM, we will often find ourselves adding and removing columns to our display between different tasks.

**Summary**

In this lesson, we gained some initial experience crafting and running searches in a SIEM. We will build on these techniques in later courses in this series and throughout our career.

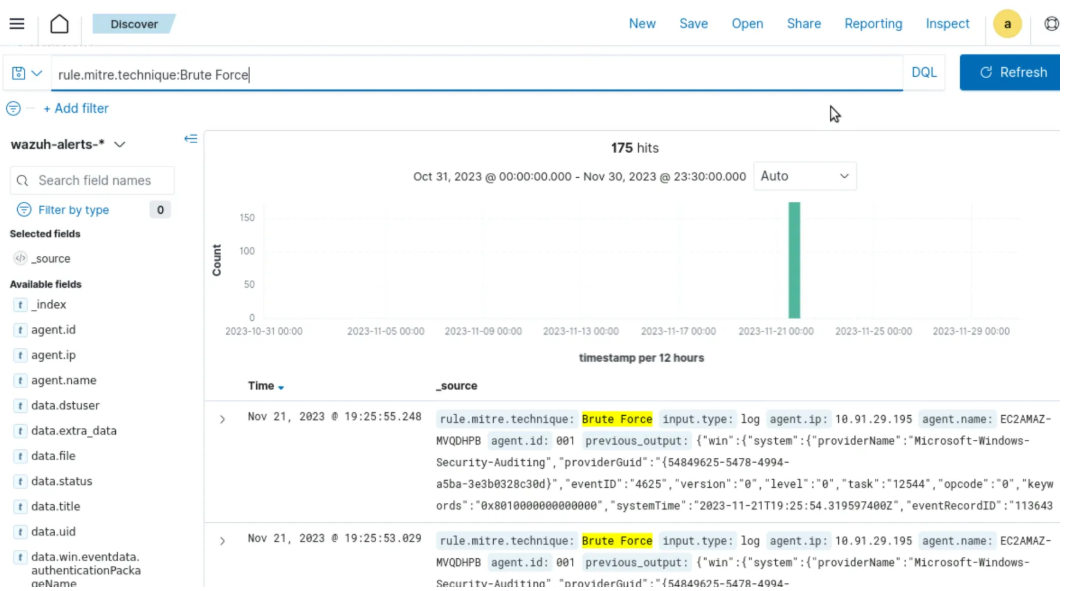
**CHALLENGE EXERCISE:**

For this challenge, clear all queries and be sure the date and time are set to “Last 1 year”. Use the following search expression to focus on the month of November 2023:

**@timestamp > "2023-11"**

On 11-21-2023, three user accounts were *brute-force* attacked. If we do not know that term, it means that an automated tool was used to try and guess the user's password.

1. What three accounts were attacked?

The approach was initially based on a query of all rule.mitre.technique:Brute Force within November 2023. There were 175 alerts for Brute Force. We then reviewed the logs and selected instances randomly along the way. We found three different users Cybrary, Guest, and Administrator. Adding AND data.win.eventdata.targetUserName:<username> for each user and totaling the alerts between them equals 175. This confirmed that those were the three users (Cybrary, Guest, and Administrator) affected by the attack. A screenshot of a computer

AI-generated content may be incorrect.

1. Which user account had the most alerts related to the brute-force attack? User: administrator had the most alerts.