Jelly Concepcion

Aspiring Cyber Defense Incident Responder | Future Chief Information Security Officer (CISO) | Passionate About Cybersecurity & Threat Detection.

Email: jellydizonconcepcion@gmail.com

GitHub: https://github.com/jellyconcepcion/blue-team-homelab

LinkedIn: https://www.linkedin.com/in/jellyconcepcion/

Introduction & Purpose

This guide documents **Splunk Project 02 – Detection Engineering**, a hands-on SOC-style project focused on building, validating, and operationalizing security detections using **Splunk Enterprise**.

The purpose of this project is to simulate how detections are developed in a real Security Operations Center (SOC), starting from **verified log ingestion** and progressing through **SPL query development, report creation, and dashboard visualization**.

All detections in this project are based on **real telemetry generated within the lab environment**, collected from:

- A Windows 11 endpoint (Windows Security Events and Sysmon telemetry)
- An Ubuntu Linux system (authentication and system logs)

Rather than relying on sample datasets, this project emphasizes **working with actual endpoint activity**, allowing realistic querying, tuning, and validation of detection logic.

Project Objectives

The primary objectives of this project are to:

- Develop SOC-relevant detections using **Splunk Processing Language (SPL)**
- Translate raw log data into **actionable security signals**

- Implement detections as **Splunk reports** with proper access controls
- Map detections to **MITRE ATT&CK techniques**
- Build a **SOC-style dashboard** for monitoring and investigation context
- Apply professional documentation practices used in detection engineering

---

Detection Scope

This project implements detections commonly used in enterprise SOC environments, including:

- Brute-force authentication attempts on Windows systems
- Privileged account and administrative activity monitoring
- Failed SSH login attempts on Linux systems
- Rare or anomalous process execution using Sysmon telemetry
- Cross-platform event correlation through a unified timeline view

These detections reflect **real-world attack techniques** and operational security concerns faced by SOC teams.

Professional Focus

This project aligns with **enterprise SOC workflows**, emphasizing:

- Log hygiene and visibility validation before detection logic
- Reproducible SPL queries and report definitions
- Clear separation between detections, dashboards, and documentation
- Evidence collection through screenshots and query results

- Analyst-oriented context to support triage and investigation

The outcome is a **SOC-ready detection engineering project** that demonstrates practical skills in Splunk, security monitoring, and incident analysis.

To proceed with the following steps, ensure that your virtual machines are already created.
If you do not yet have any VMs, you can follow the same step-by-step process for creating an **Ubuntu VM** using `ubuntu-24.04.3-live-server-amd64.iso` on **VirtualBox 7.1.12**, as documented in **LAB-Ubuntu-01-Setup.pdf**, available on my GitHub:
https://github.com/jellyconcepcion/blue-team-homelab/blob/main/01-lab-setup/01-ubuntu-siem/LAB-Ubuntu-01-Setup.pdf

The step-by-step guide for creating a **Windows 11 Pro VM** is also available on my GitHub:
https://github.com/jellyconcepcion/blue-team-homelab/blob/main/01-lab-setup/02-windows-endpoint/LAB-Windows-01-Setup.pdf

Note: In these guides, replace all references to **"Wazuh"** with **"Splunk"**, as the following steps use **Splunk Enterprise 10.0.2** instead of Wazuh.

Additionally, before proceeding, you must configure your virtual machines to forward logs and events to **Splunk Enterprise**.
This can be done by following the step-by-step guide provided in:
https://github.com/jellyconcepcion/blue-team-homelab/blob/main/02-splunk-projects/Splunk-Project-01-SOC-Homelab/docs/Splunk-Project-01-Guide.pdf

---

Splunk-Project-02 - Detection Engineering

Create 5 Core Saved Searches

Detection 1: Failed Authentication Attempts

Why it matters: Brute force, password spraying

Verify what fields actually exist

Run this **first**:

```
index=windows_logs EventCode=4625
```

```
| table _time host sourcetype *
| head 1
```

Your output:

```
Account_Name              Source_Network_Address
WIN11-LAB-01$             127.0.0.1
labuser                   127.0.0.1
```

This tells us:

Your field	What it really is
Account_Name	Windows username
Source_Network_Address	Source IP
127.0.0.1	Localhost (logins from the same machine)

Make it readable

Add a final formatting step so analysts can read it fast:

```
index=windows_logs EventCode=4625
| bucket _time span=5m
| stats count by _time host Account_Name Source_Network_Address
| where count >= 5
```

```
| sort -count

| table _time host Account_Name Source_Network_Address count
```

This is now:

- Clean
- Triage-ready
- SOC-usable

STEP-BY-STEP: Save your search as a REPORT (exact clicks)

You are here:

> Search & Reporting → Statistics tab
> Your query already ran and shows results

🔹 STEP 1 — Find the correct “Save As” button

Look at the **top of the search results pane**, slightly **above the table**.

You should see text like:

```
Save As | Create Table View | Close
```

👉 **Click `Save As`** (not Create Table View)

🔹 STEP 2 — Choose “Report”

After clicking **Save As**, a dropdown appears.

Click:

```
Save As → Report
```

A **Save Report dialog** will open.

🔹 STEP 3 — Fill in the report details (copy exactly)

Title

```
Detection – Windows Failed Logins (5 in 5 min)
```

Description

```
Detects potential brute-force or password-spraying activity by identifying

5 or more failed Windows logon attempts from the same source IP and account

within a 5-minute window.
```

Content

- Keep **Statistics** selected (correct for detections)

Editable Time Range

- Choose “No”.

🔹 STEP 4 — Set permissions (VERY IMPORTANT)

Click **Permissions** (or “Set Permissions”).

Set exactly:

- **Choose App (Not owner or all apps):** Search & Reporting
- Run as: Owner (not user)

- **Read:** Everyone
- **Write:** Admin

Click **Save**.

STEP 5 — Verify it saved correctly

After saving:

1. Go to **Reports** (top menu: Search | Reports | Alerts | Dashboards)
2. Find:

```
Detection – Windows Failed Logins (5 in 5 min)
```

3. Click it → confirm:
    - Query is correct
    - Results still populate

Screenshot to TAKE 📸

Now take your screenshot.

What must be visible

- Report title
- SPL query
- Results table

- Time range selector

Filename

`Detection-Failed-Logins.png`

STEP 6 — Document it (for your repo)

Add this to `Detection-1.md`:

```
### Detection: Windows Failed Authentication Attempts


Purpose:

Identify brute-force or password-spraying attacks against Windows endpoints.


Data Source:

Windows Security Event Log (EventCode 4625)


Detection Logic:

Counts failed login attempts per account and source IP within a 5-minute window.

Triggers when 5 or more failures occur.


SPL Query:

index=windows_logs EventCode=4625

| bucket _time span=5m
```

```
| stats count by _time host Account_Name Source_Network_Address

| where count >= 5

| sort -count

| table _time host host Account_Name Source_Network_Address
```

MITRE ATT&CK:

T1110 – Brute Force

Expected False Positives:

- Users mistyping passwords
- Automated login scripts

Analyst Actions:

- Validate source IP reputation
- Check for successful logons following failures
- Correlate with admin privilege events

SPL for Detection 2 – Privileged / Admin Logins

```
index=windows_logs EventCode=4672

| search NOT Account_Name="SYSTEM" AND NOT Account_Name="LOCAL
SERVICE" AND NOT Account_Name="NETWORK SERVICE"

| table _time host Account_Name
```

```
| sort -_time
```

What changed & why

- `NOT Account_Name="SYSTEM"` → removes routine SYSTEM logins
- `NOT Account_Name="LOCAL SERVICE"` and `NOT Account_Name="NETWORK SERVICE"` → removes other service accounts
- Now the result **only shows real users who got admin privileges**, which is what SOC analysts care about.

✅ This aligns with professional SOC practices: focusing on **anomalous or high-risk accounts** instead of expected system activity.

`.md` File for Detection 2

File path:

```
02-splunk-projects/Splunk-Project-01-SOC-Homelab/docs/detections/detection-02-privileged-logins/detection-02-privileged-logins.md
```

```
# Detection: Privileged / Admin Logins (Excluding System Accounts)

**Purpose:**

Detect potential privilege escalation by identifying non-system accounts obtaining administrative privileges.

**Data Source:**

Windows Security Event Log (EventCode 4672)
```

Detection Logic:

- Counts logins by non-standard admin accounts

- Filters out routine SYSTEM, LOCAL SERVICE, and NETWORK SERVICE logins

- Triggers when any unusual admin account logs in

SPL Query:

```
index=windows_logs EventCode=4672
 | search NOT Account_Name="SYSTEM" AND NOT Account_Name="LOCAL SERVICE" AND NOT Account_Name="NETWORK SERVICE"
 | table _time host Account_Name
 | sort -_time
```

MITRE ATT&CK:

- T1078 – Valid Accounts

- T1548 – Abuse Elevation Control Mechanism

Expected False Positives:

- Admin accounts logging in during routine maintenance windows

- Scheduled automation using privileged accounts

Analyst Actions:

- Validate unusual admin account logins

```
- Correlate with failed login attempts or suspicious activity

- Confirm legitimacy with endpoint owner or IT team
```

Updated Report Details for Splunk UI

- **Title:**

```
Detection – Privileged Admin Logins (Non-System)
```

- **Description:**

```
Detects potential privilege escalation by identifying logins by
non-system accounts with administrative privileges while ignoring
routine SYSTEM, LOCAL SERVICE, and NETWORK SERVICE logins.
```

- **Permissions:**
    - App: Search & Reporting
    - Read: Everyone
    - Write: Admin
- **Run As:**
    - User (so it doesn't require admin rights for routine viewing)

✅ Professional Impact

- Focuses on actionable events, not system noise
- Aligns with SOC standard: **look for anomalies, correlate across endpoints**
- Prepares you for real SOC detection and reporting

Take screenshot of SPL query + results → `Detection-Admin-Logins.png`

Detection 3

Why `user` and `src_ip` are empty

`user` and `src_ip` are empty because Splunk did NOT automatically extract them from the raw Linux SSH logs.

Splunk only auto-extracts fields when:

- **The sourcetype has built-in field extractions OR**
- **You explicitly define them via `props.conf + transforms.conf` OR**
- **You extract them at search time using SPL (like `rex`)**

Right now:

- **The events exist ✅**
- **The logs are forwarded correctly ✅**
- **But Splunk doesn't "know" what part of the text is the username or source IP ❌**

So if you only run this without rex:

```
| stats count by host user src_ip
```

Splunk says:

> **"I see events, but `user` and `src_ip` fields are empty → nothing to group by."**

That's why you'll get 0 rows even though events exist.

Why we use `rex` (what `rex` means)

What is `rex`?

`rex` = Regular Expression extraction

Think of it as:

> **"Hey Splunk, go inside the raw log text and manually pull out fields using patterns."**

It is a search-time field extractor.

✔️ **Professional SOC analysts use `rex` all the time**
✔️ **It does not modify data on disk**
✔️ **It's perfect for labs and detection logic**

Understanding the SPL query

Here's the query again:

```
index=security_logs sourcetype=linux_secure ("Failed password" OR "Invalid user")
```

```
| rex "Failed password for (invalid user )?(?<user>\S+)"
| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"
| stats count by host user src_ip
| where count >= 3
| sort -count
```

First `rex`: extracting the SSH username

```
| rex "Failed password for (invalid user )?(?<user>\S+)"
```

What is happening here?

This pattern matches log lines like:

```
Failed password for invalid user wronguser from
192.168.56.10

Failed password for labadmin from 192.168.56.10
```

Breaking down the regex:

`(invalid user )?`

- `invalid user` → **literal text**
- `()` → **grouping**

- `?` → optional

So this matches:

- ✔️ `invalid user wronguser`
- ✔️ `labadmin` (without “invalid user”)

`(?<user>\S+)`

This is the most important part.

- `?<user>` → name the field `user`
- `\S+` → one or more non-space characters

So Splunk extracts:

- `wronguser`
- `labadmin`

and stores them in a field called `user`

Second `rex`: extracting the source IP

```
| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"
```

This matches:

`from 192.168.56.10`

Regex breakdown:

`(?<src_ip>...)`

- **Creates a field named `src_ip`**

`\d+`

- **`\d` = digit (0–9)**
- **`+` = one or more digits**

`\.`

- **Literal dot (`.` is special in regex, so we escape it)**

So:

`\d+\.\d+\.\d+\.\d+`

Matches:

`192.168.56.10`

Now Splunk knows the source IP, even though it wasn't auto-extracted.

Why `stats` works AFTER `rex`

```
| stats count by host user src_ip
```

Now:

- `host` → **already exists**
- `user` → **extracted by first** `rex`
- `src_ip` → **extracted by second** `rex`

So Splunk can finally:

- **Group events properly**
- **Count failures per user + IP + host**

This is SOC-grade detection logic.

SPL to save as report

```
index=security_logs sourcetype=linux_secure ("Failed password" OR
"Invalid user")

| rex "Failed password for (invalid user )?(?<user>\S+)"

| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"

| stats count by host user src_ip

| where count >= 3

| sort -count
```

Report title

Detection – Linux SSH Failed Logins

Description

```
Detects repeated SSH authentication failures against Linux hosts

that may indicate brute-force or credential-guessing activity.
```

What to screenshot

📸 **Detection-SSH-Failures.png**

- SPL visible
- Statistics tab
- host, user, src_ip, count shown

What to write in the MD file

MD File (`detection-03-ssh-failures.md`):

```
# Detection: Linux SSH Failed Authentication Attempts


## Purpose

Detect repeated SSH authentication failures against Linux systems that
may indicate brute-force or credential-guessing attacks.
```

Data Source

Ubuntu Linux security logs (`/var/log/auth.log`) collected and forwarded to Splunk using **Splunk Universal Forwarder 10.0.2**.

Detection Logic

Counts failed SSH login attempts grouped by host, user, and source IP address.

Triggers when **three (3) or more failed authentication attempts** are observed within the selected time range.

SPL Query

```spl
index=security_logs sourcetype=linux_secure ("Failed password" OR "Invalid user")
| rex "Failed password for (invalid user )?(?<user>\S+)"
| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"
| stats count by host user src_ip
| where count >= 3
| sort -count
```

MITRE ATT&CK Mapping

T1110 – Brute Force

Expected False Positives

Users mistyping passwords

Automated scripts or monitoring tools

```
Misconfigured services attempting authentication
```

```
Analyst Actions

Validate the source IP reputation and ownership

Check for successful logins following repeated failures

Correlate with other authentication or privilege-related events

Assess whether the activity aligns with normal administrative behavior
```

```
Lab Notes

In this lab environment, the source IP may appear identical to the
host IP due to NAT and single-host virtualization.

 In production environments, this detection typically reveals external
attacker IP addresses targeting SSH services.
```

Detection 4 — Rare Process Execution (Sysmon / Windows)

Correct Sysmon input

On Windows 11 UF, open notepad as admin then locate and update `C:\Program Files\SplunkUniversalForwarder\etc\system\local\inputs.conf` to:

```
[WinEventLog://Security]
```

```
disabled = 0

index = windows_logs


[WinEventLog://System]

disabled = 0

index = windows_logs


[WinEventLog://Application]

disabled = 0

index = windows_logs


[WinEventLog://Microsoft-Windows-Sysmon/Operational]

disabled = 0

index = windows_logs

sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational

renderXml = true
```

Force-register Sysmon channel

Windows sometimes doesn't expose the channel cleanly.

Run as Administrator Powershell on Windows 11:

```
wevtutil sl Microsoft-Windows-Sysmon/Operational /e:true
```

Add Splunk UF to Event Log Readers

Run **Admin PowerShell**:

```
net localgroup "Event Log Readers" "NT AUTHORITY\SYSTEM" /add
```

STOP Splunk UF

Admin PowerShell:

```
net stop SplunkForwarder
```

Create the Sysmon folder manually (so Splunk UF can store a checkpoint)

```
New-Item -Path "C:\Program
Files\SplunkUniversalForwarder\var\lib\splunk\modinputs\WinEventLog\Mi
crosoft-Windows-Sysmon" -ItemType Directory
```

✅ Now Splunk has a proper place to track the channel.

Grant the UF service access to the Sysmon channel

Open **Admin PowerShell** and run:

```
net localgroup "Event Log Readers" "NT AUTHORITY\LOCAL SERVICE" /add
```

Confirm the UF service account via PowerShell

Open **Admin PowerShell** and run:

```
Get-Service -Name SplunkForwarder | Select-Object Name, StartType,
Status, @{Name="LogOnAs";Expression={(Get-WmiObject Win32_Service
-Filter "Name='SplunkForwarder'").StartName}}
```

- Look at the `LogOnAs` column.
- It will show something like:
    - `NT SERVICE\SplunkForwarder`

Grant NT SERVICE\SplunkForwarder access to Sysmon Operational

Find the SID of NT SERVICE\SplunkForwarder

Run this in **Admin PowerShell**:

```
$obj = New-Object System.Security.Principal.NTAccount("NT
SERVICE\SplunkForwarder")

$obj.Translate([System.Security.Principal.SecurityIdentifier]).Value
```

- This will output something like:

```
S-1-5-80-972488765-139171986-783781252-3188962990-3730692313
```

- Copy this **exact SID**.

Update the ACL using the correct SID

Once you have the SID, replace it in the `wevtutil` command:

```
wevtutil sl Microsoft-Windows-Sysmon/Operational
/ca:"O:BAG:SYD:(A;;0xf0007;;;SY)(A;;0x7;;;BA)(A;;0x1;;;BO)(A;;0x1;;;SO
)(A;;0x1;;;S-1-5-32-573)(A;;0x1;;;S-1-5-80-972488765-139171986-7837812
52-3188962990-3730692313))"
```

- Replace `<SID>` with the SID from step 2.

Restart Splunk UF

```
net start SplunkForwarder
```

Field extraction using `rex` in your SPL

You can extract `Image`, `CommandLine`, and `User` directly from the raw XML with `rex`:

```
index=windows_logs
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"

| rex field=_raw "<Data Name='Image'>(?<Image>[^<]+)</Data>"

| rex field=_raw "<Data
Name='CommandLine'>(?<CommandLine>[^<]*)</Data>"

| rex field=_raw "<Data Name='User'>(?<User>[^<]*)</Data>"

| table _time host Image CommandLine User

| sort -_time
```

Explanation:

- `?<Image>` → creates a new field called `Image` from the XML `<Data Name='Image'>…</Data>`.

- `[^<]+` → grabs everything until the next `<` (end of the tag).
- Same logic applies to `CommandLine` and `User`.

After running this SPL, you **should see the executable path, command line, and user account properly populated**.

Screenshot: `Detection-Rare-Process.png`

MD File (`detection-04-rare-process.md`):

```
# Detection: Rare Process Execution

**Purpose:**
Identify processes that execute rarely on Windows endpoints, which may indicate malware, suspicious activity, or abnormal user behavior.

**Data Source:**
Windows Sysmon Event Logs

**Detection Logic:**
- Extracts the process executable path (`Image`), full command line (`CommandLine`), and the user (`User`) from Sysmon event logs.
- Counts occurrences of each executable (`Image`) to find rare or unusual executions.
- Analysts investigate processes that appear infrequently or unexpectedly.

**SPL Query:**
```spl
index=windows_logs
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
| rex field=_raw "<Data Name='Image'>(?<Image>[^<]+)</Data>"
| rex field=_raw "<Data
Name='CommandLine'>(?<CommandLine>[^<]*)</Data>"
| rex field=_raw "<Data Name='User'>(?<User>[^<]*)</Data>"
| table _time host Image CommandLine User
| sort -_time
```
```

MITRE ATT&CK:

- T1059 – Command and Scripting Interpreter

Expected False Positives:

- One-time admin scripts
- Software updates
- Scheduled maintenance tasks

Analyst Actions:

- Verify executable location and file hash
- Compare against whitelist and known software
- Correlate with network connections, logon activity, and process ancestry

Report Title: Detection – Rare Process Execution (Sysmon / Windows)
Description: This detection identifies Windows processes that execute rarely, using Sysmon logs to capture process path, command line arguments, and executing user. Rare or unusual executions may indicate malware, misuse, or abnormal administrative activity. Analysts should investigate low-frequency processes while accounting for known legitimate one-off executions.

Detection 5 — Combined Timeline / Correlation View

SPL Query:

```
(index=windows_logs OR index=security_logs)

| rex field=_raw "<Data Name='Image'>(?<Image>[^<]+)</Data>"
| rex field=_raw "<Data
Name='CommandLine'>(?<CommandLine>[^<]*)</Data>"
| rex field=_raw "<Data Name='User'>(?<win_user>[^<]*)</Data>"

| rex "Failed password for (invalid user )?(?<linux_user>\S+)"
| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"

| eval user=coalesce(Account_Name, win_user, linux_user)
| eval source_ip=coalesce(Source_Network_Address, src_ip)

| table _time host user EventCode Image CommandLine source_ip
| sort -_time
```

Why this is the *right* professional approach

1️⃣ `rex` extracts raw data

Splunk **does not auto-parse everything**, especially:

- Sysmon XML
- Linux auth logs

2️⃣ `eval coalesce()` normalizes fields

Creates a **single field name** SOC analysts can reason about.

3️⃣ Timeline stays clean

Even if:

- Linux has no `EventCode`

- Windows has no `src_ip`

Splunk simply leaves those cells blank (correct behavior).

📸 Screenshot Requirements

Filename:

`Detection-Combined-Timeline.png`

Updated `detection-05-combined-timeline.md`

```
# Detection: Combined Timeline Correlation

**Purpose:**
Provide a unified timeline of Windows and Linux security events to
support investigation and incident triage.

**Data Source:**
• Windows Security Event Logs
• Windows Sysmon Logs
• Linux Authentication Logs (`/var/log/auth.log`)

**Detection Logic:**
Normalizes user and source IP fields across Windows and Linux logs to
present a single chronological event timeline.
This view enables analysts to correlate authentication events, process
execution, and suspicious activity across hosts.

**Field Availability Notes:**
Not all events contain `source_ip` or `EventCode` fields.
This is expected behavior depending on the log type:
- Local system and service events do not generate network source IPs
- Linux authentication logs only include IP addresses for remote (SSH)
activity
```

```
Missing fields indicate normal log behavior, not detection failure.

**SPL Query:**
(index=windows_logs OR index=security_logs)
| rex field=_raw "<Data Name='Image'>(?<Image>[^<]+)</Data>"
| rex field=_raw "<Data
Name='CommandLine'>(?<CommandLine>[^<]*)</Data>"
| rex field=_raw "<Data Name='User'>(?<win_user>[^<]*)</Data>"
| rex "Failed password for (invalid user )?(?<linux_user>\S+)"
| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"
| eval user=coalesce(Account_Name, win_user, linux_user)
| eval source_ip=coalesce(Source_Network_Address, src_ip)
| table _time host user EventCode Image CommandLine source_ip
| sort -_time

**MITRE ATT&CK:**
Multiple techniques depending on correlated activity

**Expected False Positives:**
• Normal system operations
• Scheduled tasks and maintenance activity

**Analyst Actions:**
• Identify suspicious sequences of events
• Correlate failed logins with privileged access
• Use as starting point for incident investigation
```

Saving as a Splunk Report

After the search succeeds:

Save As → Report

Report Title

```
Detection – Combined Timeline Correlation
```

Description

```
Provides a unified timeline of Windows and Linux security events by normalizing
user and source IP fields to support SOC investigation and event correlation.
Note: Some events may not contain source IPs or EventCodes depending on log type.
```

1️⃣ What does `eval` mean in SPL?

Yes — `eval` literally means "evaluate."

In Splunk:

> `eval` is used to **create, modify, or normalize fields** using logic or functions.

Think of it as:

- "Create a new field"
- "Fix inconsistent field names"
- "Combine multiple fields into one"

Simple example

```
| eval status="failed"
```

➡ Creates a new field called `status` with value `failed` for every event.

2️⃣ What is `coalesce()`?

`coalesce()` is a **function used inside `eval`**.

Definition (SOC-friendly)

`coalesce()` returns the **first non-empty value** from a list of fields.

Syntax

```
coalesce(field1, field2, field3)
```

Splunk checks them **left to right**:

- If `field1` exists → use it
- Else if `field2` exists → use it
- Else if `field3` exists → use it
- If all are empty → result is null

3️⃣ Why SOCs use `coalesce` (IMPORTANT)

Different log sources use **different field names**:

Platform	Username Field
Windows Security Log	`Account_Name`
Sysmon	`User`
Linux auth.log	extracted `linux_user`

Without normalization:

- Dashboards break
- Correlation fails
- Investigations take longer

4️⃣ This line explained (very important)

```
| eval user=coalesce(Account_Name, win_user, linux_user)
```

What it does:

Creates **one unified field** called `user`.

Logic:

1. Use `Account_Name` if it exists (Windows)
2. Else use `win_user` (Sysmon XML)
3. Else use `linux_user` (SSH logs)

✔ Result:
One field that works across Windows + Linux

That's **SOC-grade field normalization**.

5️⃣ What is `Source_Network_Address`?

Short answer:

It's a **native Windows Security Event field**.

Long answer:

`Source_Network_Address` appears in events like:

- **4625** – Failed logon
- **4624** – Successful logon

It represents:

> The **IP address where the login attempt came from**

Examples:

- `192.168.56.1`
- `10.0.0.5`
- `127.0.0.1` (local system)

6️⃣ Why you need this line

```
| eval source_ip=coalesce(Source_Network_Address, src_ip)
```

Because:

- Windows logs use `Source_Network_Address`
- Linux SSH logs include IPs in text → extracted as `src_ip`

This creates **one consistent field**:

`source_ip`

That allows:

- Correlation
- Dashboards
- Brute force detection
- Investigation timelines

7️⃣ Why some events have `source_ip = -` or empty

This is **EXPECTED and NORMAL**.

Why?

Some events are:

- Local system events
- Service starts
- Scheduled tasks
- Kernel events

These **do not originate from a network**, so:

- No source IP exists
- Field remains empty or -

✔ That is **not an error**
✔ That is **correct SOC behavior**

Create the Dashboard (Step-by-Step)

Step A — Open Dashboard Builder

1. Go to **Search & Reporting**
2. Click **Dashboards**
3. Click **Create New Dashboard**

Step B — Dashboard Metadata (USE THESE EXACT VALUES)

Dashboard Title

SOC Security Monitoring Dashboard – Windows & Linux

Dashboard Description

This dashboard provides a centralized SOC view of critical Windows and Linux
security detections, including failed authentication attempts, privileged
logins, SSH brute-force activity, rare process execution, and a combined
cross-platform event timeline to support security monitoring and investigation.

Permissions

- ✅ Shared in App

Dashboard Type

- ✅ Classic Dashboards

👉 Click **Create Dashboard**

Add Panels (Exact Setup for Each Detection)

🔴 Panel 1 — Windows Failed Logins (Brute Force)

Add Panel

- Click **Add Panel**

- Choose **New from Search**

Panel Settings

Panel Title

```
Windows Failed Logins (Brute Force Detection)
```

Panel Description

```
Displays accounts with 5 or more failed Windows logon attempts from
the same
source IP within a 5-minute window.
```

Panel Search (PASTE EXACTLY):

```
index=windows_logs EventCode=4625
| bucket _time span=5m
| stats count by _time host Account_Name Source_Network_Address
| where count >= 5
| sort -count
| table _time host Account_Name Source_Network_Address count
```

Visualization

- Statistics Table

👉 Save panel

🟠 Panel 2 — Privileged Admin Logins

Panel Title

```
Privileged Admin Logins (Non-System Accounts)
```

Panel Description

```
Identifies privileged logins by non-system Windows accounts that may
indicate
privilege escalation or misuse of administrative credentials.
```

Panel Search

```
index=windows_logs EventCode=4672
| search NOT Account_Name="SYSTEM" AND NOT Account_Name="LOCAL
SERVICE" AND NOT Account_Name="NETWORK SERVICE"
| table _time host Account_Name
| sort -_time
```

Visualization

- Statistics Table

👉 Save panel

🟡 Panel 3 — Linux SSH Failed Logins

Panel Title

```
Linux SSH Failed Logins
```

Panel Description

```
Shows repeated SSH authentication failures on Linux hosts that may
indicate
brute-force or credential-guessing activity.
```

Panel Search

```
index=security_logs sourcetype=linux_secure ("Failed password" OR
"Invalid user")
| rex "Failed password for (invalid user )?(?<user>\S+)"
| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"
```

```
| stats count by host user src_ip
| where count >= 3
| sort -count
```

Visualization

- Statistics Table

👉 Save panel

🟣 Panel 4 — Rare Process Execution (Sysmon)

Panel Title

```
Rare Process Execution (Windows Sysmon)
```

Panel Description

```
Displays process executions captured by Sysmon, highlighting
potentially rare
or unusual processes for further investigation.
```

Panel Search

```
index=windows_logs
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
| rex field=_raw "<Data Name='Image'>(?<Image>[^<]+)</Data>"
| rex field=_raw "<Data
Name='CommandLine'>(?<CommandLine>[^<]*)</Data>"
| rex field=_raw "<Data Name='User'>(?<User>[^<]*)</Data>"
| table _time host Image CommandLine User
| sort -_time
```

Visualization

- Statistics Table

👉 Save panel

🔵 Panel 5 — Combined Timeline (Correlation View)

Panel Title

```
Combined Windows & Linux Security Timeline
```

Panel Description

```
Provides a unified, time-ordered view of Windows and Linux security
events
to support SOC investigation and correlation.
```

Panel Search

```
(index=windows_logs OR index=security_logs)
| rex field=_raw "<Data Name='Image'>(?<Image>[^<]+)</Data>"
| rex field=_raw "<Data
Name='CommandLine'>(?<CommandLine>[^<]*)</Data>"
| rex field=_raw "<Data Name='User'>(?<win_user>[^<]*)</Data>"
| rex "Failed password for (invalid user )?(?<linux_user>\S+)"
| rex "from (?<src_ip>\d+\.\d+\.\d+\.\d+)"
| eval user=coalesce(Account_Name, win_user, linux_user)
| eval source_ip=coalesce(Source_Network_Address, src_ip)
| table _time host user EventCode Image CommandLine source_ip
| sort -_time
```

Visualization

- Statistics Table

👉 Save panel

Take Individual Panel Screenshots

- Open your dashboard.
- For each panel:
    1. Scroll the dashboard so the panel is fully visible.
    2. Include the **panel title**, **the data rows**, and optionally the **search bar at the top** for context.
    3. Take a screenshot.

Example Filenames:

Panel	Screenshot Filename
Windows Failed Logins	`Dashboard-Panel-Windows-Failed-Logins.png`
Privileged Admin Logins	`Dashboard-Panel-Privileged-Admin-Logins.png`
Linux SSH Failed Logins	`Dashboard-Panel-Linux-SSH-Failed-Logins.png`
Rare Process Execution	`Dashboard-Panel-Rare-Process-Execution.png`
Combined Timeline	`Dashboard-Panel-Combined-Timeline.png`

> Tip: Make sure the **column headers** (_time, host, user, EventCode, etc.) are visible in the screenshot.

Full Dashboard Overview Screenshot

One dashboard screenshot:

- Zoom out your browser until you can **see 2–3 panels stacked**.
- Screenshot that partial view.

- Filename: `Dashboard-Overview.png`

- Note:
  "Full dashboard includes 5 panels. Individual panels are captured in separate screenshots for clarity."

README.md for the dashboards folder:

```
# SOC Security Monitoring Dashboard

This dashboard consolidates five core security detections across
Windows and Linux systems into a single SOC monitoring view.
It enables analysts to quickly identify authentication attacks,
privilege escalation, suspicious process execution, and correlate
events across platforms during investigations.

## Included Detections

| Panel | Purpose |
|-------|---------|
| Windows Failed Logins | Detect brute-force and password-spraying
attempts |
| Privileged Admin Logins | Identify non-system accounts with admin
privileges |
| Linux SSH Failed Logins | Detect repeated SSH login failures on
Linux hosts |
| Rare Process Execution | Highlight unusual Windows processes via
Sysmon |
| Combined Timeline Correlation | Unified, time-ordered view for
cross-platform correlation |

All panels use **Statistics Table visualization** for clear SOC-style
analysis.
Individual panel screenshots are provided in this folder.
```
