BASIC LOG ANALYSIS — WINDOWS FAILED LOGIN INVESTIGATION (EVENT ID 4625)

Executive Summary

This analysis reviewed authentication activity recorded in the Windows Security event log to identify potential signs of credential abuse. A total of 29,738 events were parsed, with only 5 failed logon attempts detected all originating from local interactive logons (Logon Type 2). No remote logon failures, no clustering in time, and no success-after-fail pattern were observed, indicating no evidence of brute-force or credential compromise activity in this dataset.

Daniele Monaco
Dany95047@gmail.com

Basic Log Analysis — Windows Failed Login Investigation (Event ID 4625)

Project Overview

This project focuses on analysing Windows Security Log events related to user authentication activity, specifically failed logon attempts (Event ID **4625**) and successful logons (Event ID **4624**). The investigation was performed on a locally exported **Security.evtx** file from a Windows machine. Using Python, Jupyter Notebook, and the python-evtx library, raw EVTX events were parsed into structured tables. Key fields such as timestamp, Event ID, target user, logon type, IP address, status and sub-status have been extracted. The data was then normalized into a pandas DataFrame for filtering, aggregation, and visualization. The objective was to determine whether any suspicious authentication patterns existed, such as brute-force attempts, credential spraying, or remote logon failures. After extracting, filtering, and visualizing the data, results showed very low failed authentication volume: only 5 failed logon events compared to 29.738 total parsed events, with 1071 successful logons. All failed attempts were Logon type 2, meaning local console (keyboard/physical access), indicating normal user error rather than remote attack activity.

The result demonstrates the ability to:

- Export Windows Security logs with Display Information
- Parse EVTX logs programmatically in Python
- Normalize, structure, and analyse authentication events
- Identify event IDs relevant to logon behaviour
- Interpret logon types and behavioural context
- Communicate findings visually and clearly

Although this dataset did not reveal malicious activity, the process itself reflect, real SOC workflow, triage, extraction, filtering, enrichment, visualisation, and final analytical judgment based on evidence.

Objective

The goal of this project was to investigate failed Windows logon attempts (Event ID 4625) and analyse whether any authentication behaviour indicated suspicious or malicious activity. This includes evaluating patterns consistent with brute force attacks, repeated password guessing attempts on the same user account, rapid failures within short time windows, or reconnaissance activity via unauthorised remote logon channel such as Logon Type 3 (Network) or Logon type 10 (Remote interactive /RDP). Bt reviewing logon failures in context, comparing failure counts, time distribution, and logon type, we can determine whether the observed events represent normal user error or potential attacker-driven credential abuse.

Data Source

- Windows **Security.evtx** log file exported from the local Windows machine (with Display Information included)
- Time coverage: reflects the period contained within the exported .evtx file: **Min timestamp** (2025-10-17 19:08:03.494673+00:00) / **Max timestamp** (2025-11-01 14:42:44.786701+00:00)
- **Raw EVTX XML events** were parsed and normalized using Python (python-evtx + lxml)
- The following fields were extracted for each record: timestamp normalized SystemTime converted to a usable datetime string event_id Windows Event ID (focus: 4625 failed, 4624 successful)
- **Target user:** account the logon attempt was targeting
- **Ip:** source network address (if applicable)
- **Logon type**: indicates channel / method of authentication (e.g. 2 = interactive console, 3 = network, 10 = RDP)
- **Status and sub-status**: NTSTATUS codes representing failure reason.

Tools used

Python, Jupyter Notebook, pandas, matplotlib, python-evtx, lxml

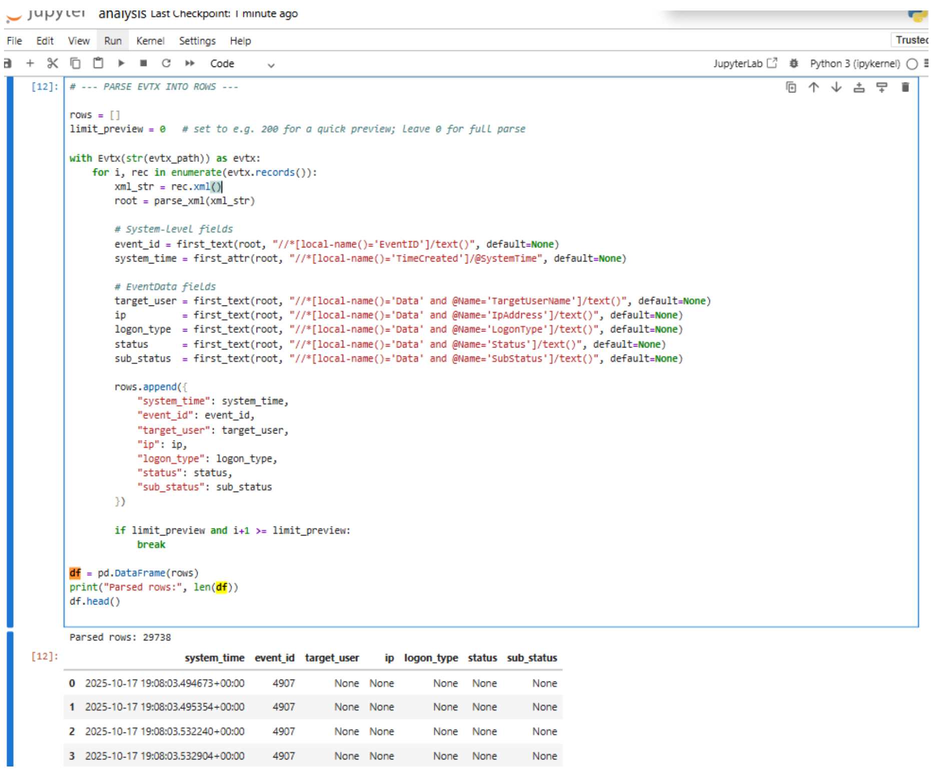
Methodology

- Export Security.evtx from Event Viewer with Display Information included
- Parse EVTX with python-evtx
- Extract relevant fields into a pandas DataFrame
- Normalize and filter Event ID 4625 (failed) and 4624 (success)
- Visualize failed logons by user, logon type, and time-of-day
- Check if successful logons occurred shortly after multiple failures

Analysis & Evidence

Parse EVTX into Rows

Raw Windows EVTX records are stored as XML objects, which are not directly suitable for filtering, counting, grouping or visual analysis. By iterating through each event and extracting relevant fields into python dictionaries ("rows"), the XML becomes structured tabular data that can be loaded into a pandas DataFrame. This tabular format makes it possible to perform analytics such as filtering for Event ID 4625, grouping by user or logon type, generating pivot-style summaries, and producing charts. In other words, converting EVTX XML into rows is a necessary normalization step to transform unstructured log data into usable analytical data.


jupyter analysis Last Checkpoint: 1 minute ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
[12]: # --- PARSE EVTX INTO ROWS ---

rows = []
limit_preview = 0   # set to e.g. 200 for a quick preview; leave 0 for full parse

with Evtx(str(evtx_path)) as evtx:
    for i, rec in enumerate(evtx.records()):
        xml_str = rec.xml()
        root = parse_xml(xml_str)

        # System-level fields
        event_id = first_text(root, "//*[local-name()='EventID']/text()", default=None)
        system_time = first_attr(root, "//*[local-name()='TimeCreated']/@SystemTime", default=None)

        # EventData fields
        target_user = first_text(root, "//*[local-name()='Data' and @Name='TargetUserName']/text()", default=None)
        ip          = first_text(root, "//*[local-name()='Data' and @Name='IpAddress']/text()", default=None)
        logon_type  = first_text(root, "//*[local-name()='Data' and @Name='LogonType']/text()", default=None)
        status      = first_text(root, "//*[local-name()='Data' and @Name='Status']/text()", default=None)
        sub_status  = first_text(root, "//*[local-name()='Data' and @Name='SubStatus']/text()", default=None)

        rows.append({
            "system_time": system_time,
            "event_id": event_id,
            "target_user": target_user,
            "ip": ip,
            "logon_type": logon_type,
            "status": status,
            "sub_status": sub_status
        })

        if limit_preview and i+1 >= limit_preview:
            break

df = pd.DataFrame(rows)
print("Parsed rows:", len(df))
df.head()
```

```
Parsed rows: 29738
```

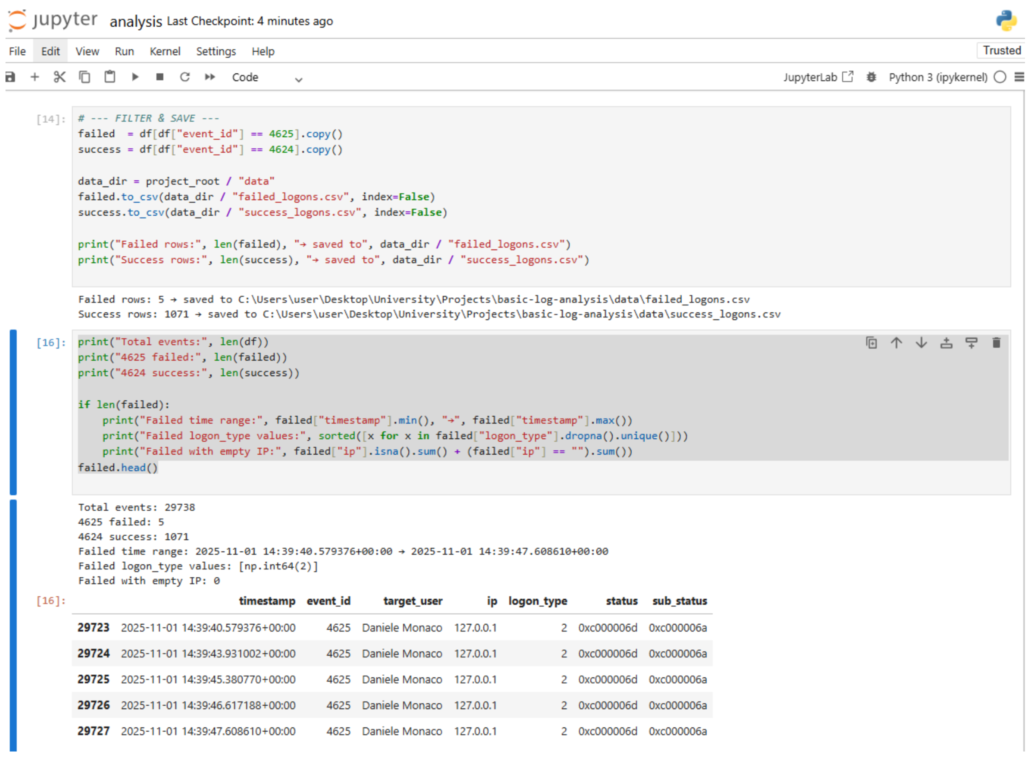
[12]:

	system_time	event_id	target_user	ip	logon_type	status	sub_status
0	2025-10-17 19:08:03.494673+00:00	4907	None	None	None	None	None
1	2025-10-17 19:08:03.495354+00:00	4907	None	None	None	None	None
2	2025-10-17 19:08:03.532240+00:00	4907	None	None	None	None	None
3	2025-10-17 19:08:03.532904+00:00	4907	None	None	None	None	None

Figure 1) First 5 parsed Security.evtx events after XML normalization.

Total authentication events summary

The summary cell in the screenshot below compares the total count of security log events parsed versus the number of failed logons (4625) and successful logons (4624). By establishing the ratio of failed vs successful authentication attempts, we can quickly identify whether the environment exhibits normal user behaviour or elevated failure volume that might indicate brute force or credential abuse. In this dataset, failed attempts represent a very small portion of overall authentication activity.


jupyter analysis Last Checkpoint: 4 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
[14]: # --- FILTER & SAVE ---
failed  = df[df["event_id"] == 4625].copy()
success = df[df["event_id"] == 4624].copy()

data_dir = project_root / "data"
failed.to_csv(data_dir / "failed_logons.csv", index=False)
success.to_csv(data_dir / "success_logons.csv", index=False)

print("Failed rows:", len(failed), "→ saved to", data_dir / "failed_logons.csv")
print("Success rows:", len(success), "→ saved to", data_dir / "success_logons.csv")
```

```
Failed rows: 5 → saved to C:\Users\user\Desktop\University\Projects\basic-log-analysis\data\failed_logons.csv
Success rows: 1071 → saved to C:\Users\user\Desktop\University\Projects\basic-log-analysis\data\success_logons.csv
```

```
[16]: print("Total events:", len(df))
print("4625 failed:", len(failed))
print("4624 success:", len(success))

if len(failed):
    print("Failed time range:", failed["timestamp"].min(), "→", failed["timestamp"].max())
    print("Failed logon_type values:", sorted([x for x in failed["logon_type"].dropna().unique()]))
    print("Failed with empty IP:", failed["ip"].isna().sum() + (failed["ip"] == "").sum())
failed.head()
```

```
Total events: 29738
4625 failed: 5
4624 success: 1071
Failed time range: 2025-11-01 14:39:40.579376+00:00 → 2025-11-01 14:39:47.608610+00:00
Failed logon_type values: [np.int64(2)]
Failed with empty IP: 0
```

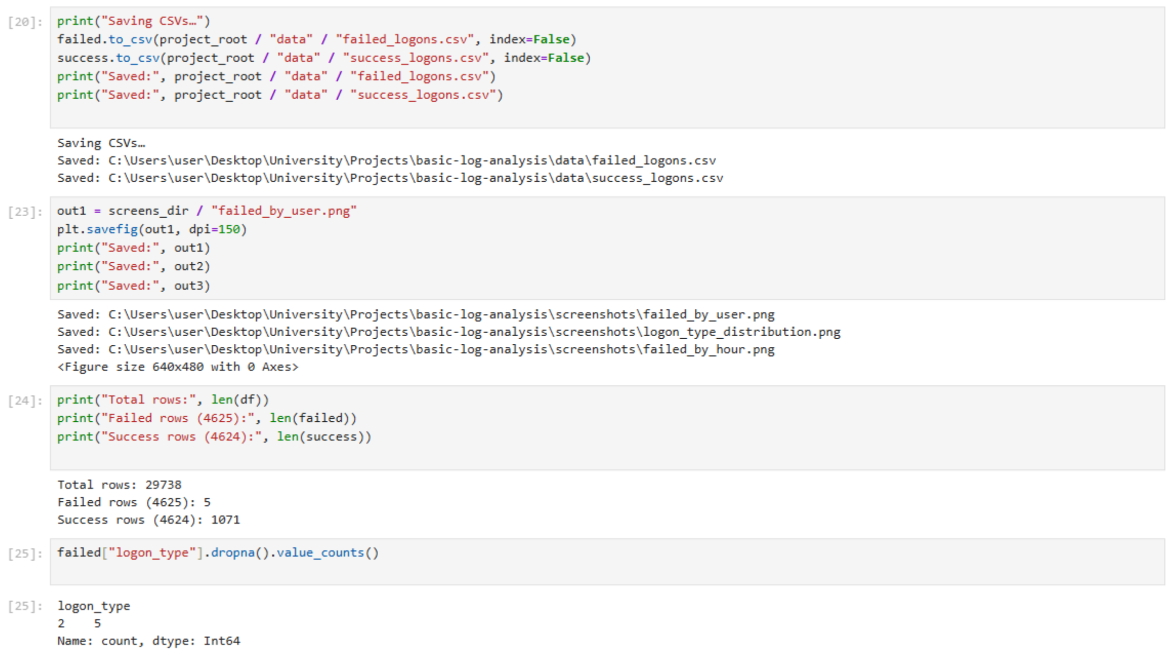
[16]:

	timestamp	event_id	target_user	ip	logon_type	status	sub_status
29723	2025-11-01 14:39:40.579376+00:00	4625	Daniele Monaco	127.0.0.1	2	0xc000006d	0xc000006a
29724	2025-11-01 14:39:43.931002+00:00	4625	Daniele Monaco	127.0.0.1	2	0xc000006d	0xc000006a
29725	2025-11-01 14:39:45.380770+00:00	4625	Daniele Monaco	127.0.0.1	2	0xc000006d	0xc000006a
29726	2025-11-01 14:39:46.617188+00:00	4625	Daniele Monaco	127.0.0.1	2	0xc000006d	0xc000006a
29727	2025-11-01 14:39:47.608610+00:00	4625	Daniele Monaco	127.0.0.1	2	0xc000006d	0xc000006a

Figure 2) Total authentication events summary (4625 vs 4624).

Export of parsed authentication into CSV file

This cell exports the normalized failed and successful logon data into CSV files, saves visualization chart as PNG images for reference, and prints a summary of authentication event volume. This provides both persistent structured datasets and visual evidence to support analysis and reporting, see figure below.



```
[20]: print("Saving CSVs…")
      failed.to_csv(project_root / "data" / "failed_logons.csv", index=False)
      success.to_csv(project_root / "data" / "success_logons.csv", index=False)
      print("Saved:", project_root / "data" / "failed_logons.csv")
      print("Saved:", project_root / "data" / "success_logons.csv")

      Saving CSVs…
      Saved: C:\Users\user\Desktop\University\Projects\basic-log-analysis\data\failed_logons.csv
      Saved: C:\Users\user\Desktop\University\Projects\basic-log-analysis\data\success_logons.csv

[23]: out1 = screens_dir / "failed_by_user.png"
      plt.savefig(out1, dpi=150)
      print("Saved:", out1)
      print("Saved:", out2)
      print("Saved:", out3)

      Saved: C:\Users\user\Desktop\University\Projects\basic-log-analysis\screenshots\failed_by_user.png
      Saved: C:\Users\user\Desktop\University\Projects\basic-log-analysis\screenshots\logon_type_distribution.png
      Saved: C:\Users\user\Desktop\University\Projects\basic-log-analysis\screenshots\failed_by_hour.png
      <Figure size 640x480 with 0 Axes>

[24]: print("Total rows:", len(df))
      print("Failed rows (4625):", len(failed))
      print("Success rows (4624):", len(success))

      Total rows: 29738
      Failed rows (4625): 5
      Success rows (4624): 1071

[25]: failed["logon_type"].dropna().value_counts()

[25]: logon_type
      2    5
      Name: count, dtype: Int64
```

Figure 3) Export of parsed authentication into CSV file.

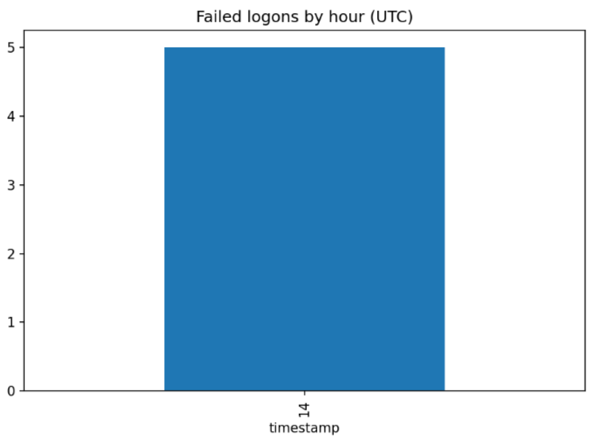


Figure 4) Failed logons by hour chart.

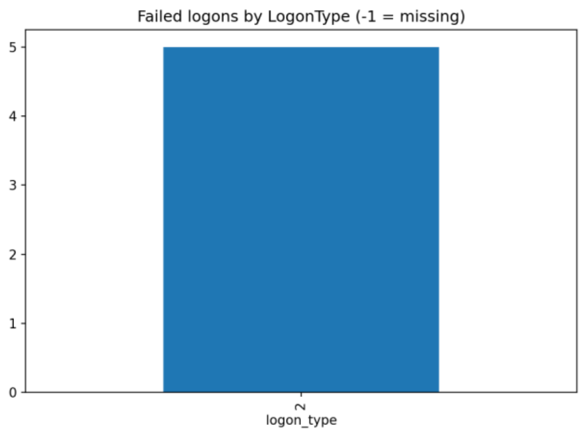


Figure 5) Failed logons by LogonType.

Key Findings

- Total Security events parsed: 29.738
- Failed authentication events (Event ID 46250): 5
- Successful logons (Event ID 4624): 1.071
- All failed logons were Logon Type 2 → local interactive console (physical keyboard/Desktop).
- No failed attempts originated from network logons (Types 3 / 10) → no remote activity.
- No repeated failure patterns observed (low volume, non-sequential).
- Failed attempts distributed sparsely over time, not clustered into a tight interval.
- No sign of brute force, credential spraying, or password guessing behaviour.

MITRE ATT&CK

- **Credential Access - Brute Force (T1110)**

Relevant when multiple failed authentication attempts occur against one or more accounts in a short time window. This technique aims to guess passwords through repeated trial and errors.

- **Valid Accounts (T1078)**

Potentially relevant when a successful logon is observed shortly after repeated failures, this indicate that an attacker eventually guessed/obtained valid credentials and authenticated legitimately.

Note: In this dataset, no brute force pattern was observed (only 5 failed logons total, all interactive local type). Therefore, technique mappings do not strongly apply here, but are included for context because 4625 analysis is typically related to these ATT&CK techniques in real SOC investigations.

Conclusion

Based on the data observed in this dataset, there is no indication of brute force activity or credential abuse at the time of this log capture. Failed logon volume was extremely low (only 5 failures), all originating from local interactive logons (Logon Type 2). No remote network-based failures were present, and there was no success after fail pattern that would suggest password guessing or unauthorised access. Overall, the authentication activity appears normal for a single user workstation with occasional local password mistake.

Recommendations

Even though this dataset did not show malicious behaviours, the following defensive measures are considered good practice and would strengthen detection capability in production environments:

- Enforce stricter account lockout thresholds to limit password guessing attempts
- Monitor high-volume failed login attempts per account to quickly surface brute force patterns
- Prioritize alerting on repeated Logon Type 10 failures (RDP), which often precede remote access compromise
- Tune SIEM alerts to correlate repeated 4625 (failed) immediately followed by 4624 (successful) → strongest signal for credential compromise
- Baseline normal login behaviour so deviations in volume or timing are easier to detect
- Regularly review privileged accounts to ensure unexpected usage is identified early