4. Experiments

4.1: Digital Forensics Using Python: Parsing Windows Event Logs:

Objective

The primary objective of Week 1 was to understand and automate the analysis of Windows Security Event Logs using Python. By focusing on log data instead of full disk images, I aimed to:

- Grasp foundational digital forensics principles through real-world data.
- Avoid the complexity of disk acquisition by working with pre-converted .evtx logs in CSV format.
- Build automation scripts using Python and pandas for log parsing and filtering.
- Gain fluency in Linux terminal tools and scripting for forensic analysis.
- Understand how log events can provide insights into system usage, account activity, and potential breaches.

Background

Windows operating systems generate and store detailed records of system and user activities in .evtx event log files. These logs serve as a critical source of truth in forensic investigations, allowing analysts to reconstruct user behaviour, trace attacks, detect privilege escalation, and identify anomalies.

Among the various log types, the **security.evtx** file is particularly vital because it records security-centric events, including:

- Successful and failed login attempts
- Account lockouts
- Privilege use and policy changes
- System shutdowns and restarts

However, working with .evtx files directly requires disk-level access or specialized viewers. To simplify the learning process, I used a .csv version of the security.evtx file, which preserved its structure and enabled easier data manipulation with Python and pandas.

Tools and Technologies Used

Tool/Tech	**Purpose**
Kali Linux	Operating system for cybersecurity tasks
Python 3.11+	Scripting and automation
pandas	Data parsing and analysis
venv	Virtual environment for Python dependencies
Terminal utilities	cat, grep, less, column for CLI analysis
Jupyter Notebook	Optional - for interactive exploration
Text Editors	nano, VS Code
Dataset	security.csv (converted from security.evtx)

Environment Setup

To ensure a clean and reproducible environment for scripting and data analysis, I followed these steps:

Step 1: System Preparation

```
sudo apt update && sudo apt install -y python3-pip git
```

```
pip3 install pandas jupyter
```

Step 2: Creating a Dedicated Workspace

```
mkdir -p ~/dfir/week1
```

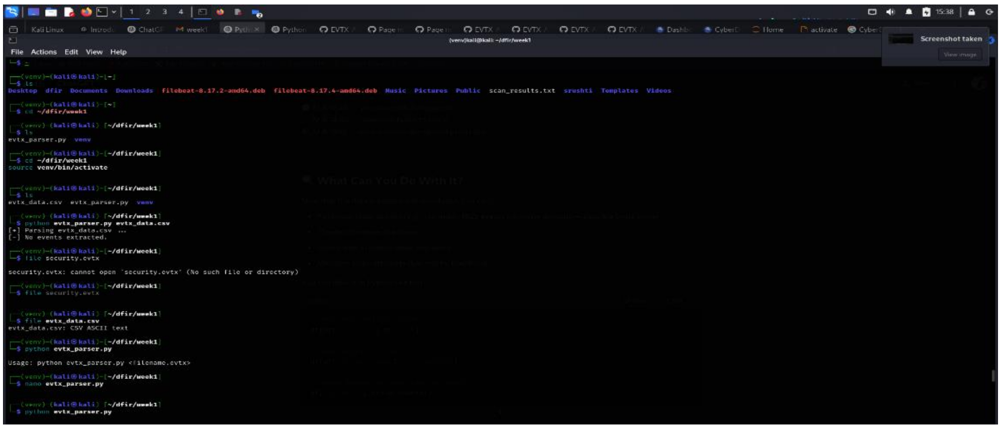
```
cd ~/dfir/week1
```

Step 3: Virtual Environment Setup

```
python3 -m venv venv
```

```
source venv/bin/activate
```

This isolated environment prevented dependency conflicts and ensured clean installations.



```
┌──(venv)─(kali㉿kali)-[~]
└─$ ls
Desktop  dfir  Documents  Downloads  filebeat-8.17.2-amd64.deb  filebeat-8.17.4-amd64.deb  Music  Pictures  Public  scan_results.txt  srushti  Templates  Videos

┌──(venv)─(kali㉿kali)-[~]
└─$ cd ~/dfir/week1

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ ls
evtx_parser.py  venv

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ cd ~/dfir/week1
source venv/bin/activate

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ ls
evtx_data.csv  evtx_parser.py  venv

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ python evtx_parser.py evtx_data.csv
[+] Parsing evtx_data.csv ...
[-] No events extracted.

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ file security.evtx

security.evtx: cannot open `security.evtx' (No such file or directory)

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ file security.evtx

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ file evtx_data.csv
evtx_data.csv: CSV ASCII text

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ python evtx_parser.py

Usage: python evtx_parser.py <filename.evtx>

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ nano evtx_parser.py

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ python evtx_parser.py
```

Step 4: Dataset Preparation

The CSV-formatted security.csv file (converted from .evtx) was placed inside the working directory for analysis.

Part 2 – Writing the Python Parser Script

A script was created to load, filter, and save relevant event data:

```
nano evtx_parser.py
```

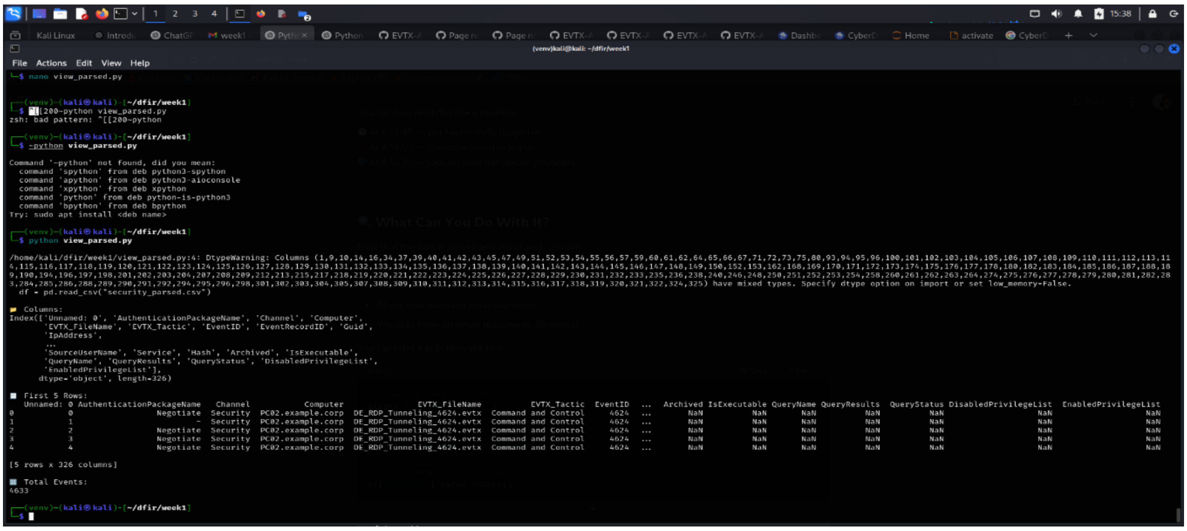
Script: evtx_parser.py

```
import pandas as pd

# Load CSV into DataFrame

df = pd.read_csv("security.csv", low_memory=False)

# Display column names for inspection

print("Available columns:", df.columns)

# Select key forensic-relevant columns

parsed_df = df[["TimeCreated", "EventID", "AccountName", "Message"]]

# Save refined output

parsed_df.to_csv("security_parsed.csv", index=False)

print("Parsing complete. Output saved as security_parsed.csv")
```

Execution

```
python evtx_parser.py
```

The script executed successfully and generated security_parsed.csv.



```
└─$ nano view_parsed.py

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ ^[[200~python view_parsed.py
zsh: bad pattern: ^[[200~python

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ ~python view_parsed.py

Command '~python' not found, did you mean:
  command 'spython' from deb python3-spython
  command 'apython' from deb python3-aioconsole
  command 'xpython' from deb xpython
  command 'python' from deb python-is-python3
  command 'bpython' from deb bpython
Try: sudo apt install <deb name>

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ python view_parsed.py

/home/kali/dfir/week1/view_parsed.py:4: DtypeWarning: Columns (1,9,10,14,16,34,37,39,40,41,42,43,45,47,49,51,52,53,54,55,56,57,59,60,61,62,64,65,66,67,71,72,73,75,80,93,94,95,96,100,101,102,103,104,105,106,107,108,109,110,111,112,113,11
4,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,152,153,162,168,169,170,171,172,173,174,175,176,177,178,180,182,183,184,185,186,187,188,18
9,190,194,196,197,198,201,202,203,204,207,208,209,212,213,215,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,235,236,238,240,246,248,250,251,252,253,254,258,260,261,262,263,264,274,275,276,277,278,279,280,281,282,28
3,284,285,286,288,289,290,291,292,294,295,296,298,301,302,303,304,305,307,308,309,310,311,312,313,314,315,316,317,318,319,320,321,322,324,325) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("security_parsed.csv")

📁 Columns:
Index(['Unnamed: 0', 'AuthenticationPackageName', 'Channel', 'Computer',
       'EVTX_FileName', 'EVTX_Tactic', 'EventID', 'EventRecordID', 'Guid',
       'IpAddress',
       ...
       'SourceUserName', 'Service', 'Hash', 'Archived', 'IsExecutable',
       'QueryName', 'QueryResults', 'QueryStatus', 'DisabledPrivilegeList',
       'EnabledPrivilegeList'],
      dtype='object', length=326)

▦ First 5 Rows:
   Unnamed: 0 AuthenticationPackageName   Channel            Computer             EVTX_FileName          EVTX_Tactic  EventID  ...  Archived IsExecutable QueryName QueryResults  QueryStatus DisabledPrivilegeList  EnabledPrivilegeList
0           0                 Negotiate  Security  PC02.example.corp  DE_RDP_Tunneling_4624.evtx  Command and Control     4624  ...       NaN          NaN       NaN          NaN          NaN                   NaN                   NaN
1           1                         -  Security  PC02.example.corp  DE_RDP_Tunneling_4624.evtx  Command and Control     4624  ...       NaN          NaN       NaN          NaN          NaN                   NaN                   NaN
2           2                 Negotiate  Security  PC02.example.corp  DE_RDP_Tunneling_4624.evtx  Command and Control     4624  ...       NaN          NaN       NaN          NaN          NaN                   NaN                   NaN
3           3                 Negotiate  Security  PC02.example.corp  DE_RDP_Tunneling_4624.evtx  Command and Control     4624  ...       NaN          NaN       NaN          NaN          NaN                   NaN                   NaN
4           4                 Negotiate  Security  PC02.example.corp  DE_RDP_Tunneling_4624.evtx  Command and Control     4624  ...       NaN          NaN       NaN          NaN          NaN                   NaN                   NaN

[5 rows x 326 columns]

▦ Total Events:
4633

┌──(venv)─(kali㉿kali)-[~/dfir/week1]
└─$ ▮
```

Part 3 – Understanding and Interpreting the Output

The resulting file had the following structure:

TimeCreated	EventID	AccountName	Message
2023-04-06T09:52:00Z	4624	Admin	An account was successfully logged on
2023-04-06T10:03:11Z	4625	Guest	An account failed to log on

Column Meanings:

- **TimeCreated**: Timestamp of the event
- **EventID**: A code identifying the type of event
- **AccountName**: Username involved in the event
- **Message**: Description of the event

Common Windows Security Event IDs:

- 4624: Successful user logon
- 4625: Failed login attempt
- 4672: Special privileges assigned (e.g., admin access)

Part 4 – Terminal-Based Exploration of Logs

Using Linux tools, I examined and filtered the log entries:

View First Few Lines

```
head security_parsed.csv
```

View as a Readable Table

```
column -s, -t < security_parsed.csv | less
```

Search for Failed Logins (Event ID 4625)

```
grep "4625" security_parsed.csv
```

Count Failed Login Attempts

```
grep "4625" security_parsed.csv | wc -l
```

Search for Activity by Specific User (e.g., Admin)

```
grep "Admin" security_parsed.csv
```

Part 5 – Optional: Interactive Exploration Using Python or Jupyter

Launch Jupyter Notebook

```
jupyter notebook
```

Read and Explore Parsed File

```
import pandas as pd

df = pd.read_csv("security_parsed.csv")
```

```
# Show all rows

pd.set_option("display.max_rows", None)

print(df)
```

This enabled further data exploration, grouping, filtering, and time-based analysis.

Key Learnings

1.Understood the structure and significance of Windows Security Logs
2.Learned Python scripting to parse large forensic datasets
3.Gained hands-on experience with pandas, virtual environments, and CLI tools
4.Became familiar with identifying user behaviors from logs (e.g., logins, failed access)
5. Built confidence in using Linux for digital forensic tasks

Challenges and Solutions

Challenge	**Solution**
CSV loading errors due to size	Used low_memory=False in pandas
Irrelevant or noisy columns	Filtered using df[["TimeCreated", ...]]
Unicode/encoding issues	Verified CSV encoding and adjusted pandas params
Script bugs (column mismatches, typos)	Iteratively debugged with print statements

Future Scope and Enhancements

- Add data visualizations (e.g., login trends, failed login spikes)
- Correlate login times with working hours to detect anomalies
- Create a CLI tool for automatic multi-log parsing and filtering
- Extend analysis to other .evtx logs (Application, System)
- Learn how registry and prefetch files complement log analysis

Conclusion

This activity laid a strong foundation in practical digital forensics through automation and scripting. By focusing on real-world event logs and using Python, I gained valuable insights into system-level activity and how it can be leveraged for investigation. The structured, hands-on approach helped me not only understand forensic concepts but also build reusable tools for future cases. This experience is a stepping stone toward more complex forensics tasks such as timeline analysis, correlation across logs, and malware event reconstruction.