2026 상반기 DFRC 신입생세미나

Python Problem #01 – Windows Event Log

2026-01-12

조민혁 (Jo Min Hyuk)

cgumgek8@gmail.com

고려대학교 정보보호대학원
Korea University
School of Cybersecurity

DFRC Korea University
Digital Forensic Research Center

Contents

Introduction to Assignment

Introduction to Assignment

Notification

- **Period**
  - 2026.01.06 18:00 ~ 2026.01.13.14:00
- **Instructions**
  - Python Version: 3.13 **(My python version is 3.10.12)**
  - Free IDE **(I use VS Code)**
- **Submission Method**
  - Check detailed instructions for each issue and create submissions
  - Compressed and sent the entire submission to the person in charge of the problem (Compressed File Name: Name_Python_Problem Number.zip)
    (My file name: 조민혁_Python_#01.zip)

Introduction to Assignment

Requirements: About Windows Event Log

- **Analyze the given EVTX file to identify the RDP connection time and information about the connecting Client.**
  - 1. Write a Python script to analyze the EVTX file from the RDP server.
  - 2. Find the logs related to the RDP connection and trace the **RDP connection timestamp and Client information**.
  - 3. Identify other artifacts, aside from event logs, that can confirm an RDP connection

Introduction to Assignment

Requirements: Instructions Details

- **Analyze Code**
  - The script must parse event logs and save the results.
  - The types of logs to analyze are up to you.
  - Output format for the RDP connection trace is flexible: csv, db, json, etc.

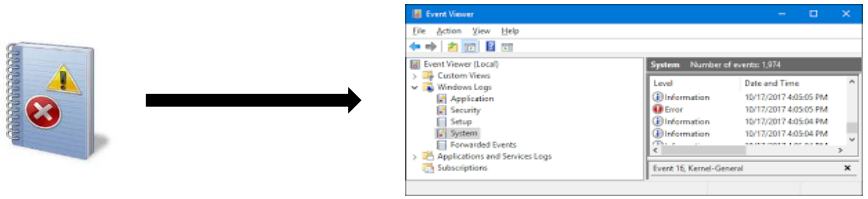
    (I made the results with csv and json)
  - You may use external libraries (provide a requirements.txt).
- **PPT**
  - Explain how to run the code (include options if any).
  - Clearly describe the related event logs and RDP tracing process with details.

Background

Background

What is Event Log?

- **Definition**
  - Event Log is a centralized, tree-structured logging system that **records important events** occurring in systems, security, applications, etc.
- **Types**
  - System Log: system-level events
  - Security Log: login attempts, permission changes, audit event
  - Application Log: events generated by applications
  - Setup Log: events related to Windows installation and updates

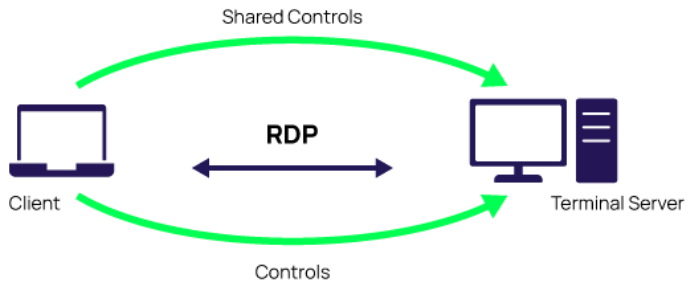


- **.EVTX Format**
  - Saved with the .evtx extension under:
    'C:₩Windows₩System32₩winevt₩Logs₩'
  - Can be analyzed using dedicated viewers or parsers (e.g., Windows Event Viewer, Python **evtx** Module)

Background

What is Remote Desktop Protocol (RDP)?

- **Definition**
  - RDP is a remote access protocol developed by Microsoft that allows a user to control another computer's screen, keyboard, and mouse over a network.
- **Process**
  - **STEP 1)** The user device **(Client)** sends a connection request to the remote computer **(Server).**
  - **STEP 2)** User authentication is performed and a remote deskop session is established.
  - **STEP 3)** Data is transmitted.
    - **Screen rendering data:** Server -> Client
    - **Input events:** Client -> Server



Relationship between Event Log and RDP

- **Relationship between Event Log and RDP**
  - **Security.evtx:** Succesful/failed logons, accounts, logon type, source IP
    - **Event ID 4624, 4625**: RDP successful/failed logon
    - **Event ID 4672:** Whether administrator privileges were assigned
    - **LogonType 10:** Logon type number for RDP
  - **LocalSessionManager/Operational.evtx (LSM):** Events related to RDP session creation/termination/reconnection
    - Possible to identify **“When the connection was mae and when it was disconnected”**
    - Can be used to estimate the session-based timeline
  - **RemoteConnectionManager/Operational.evtx (RCM):** Remote connection attempts and authentication events
    - Possible to identify **“Who attempted to connect and when”**

Background

Our Purpose: Developing a Python-based RDP Event Log Analysis Program

- **Program Design**
  - **STEP 1)** Analyze EVTX files (the three artifacts mentioned earlier)

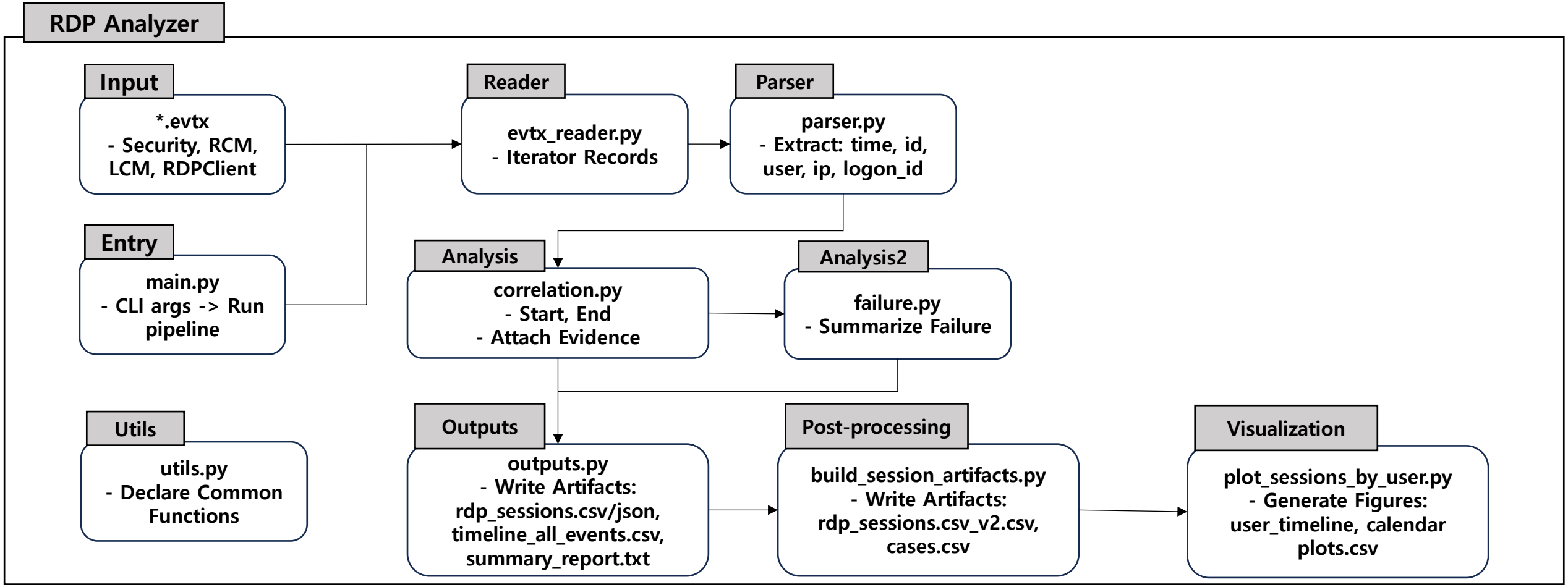
    - Parse the logs, save results in CSV and JSON formats, and visualize the saved results
  - **STEP 2) Identify RDP connection events**

    - Event ID 4624, 4625, 4672
    - LogonType 10
    - LSM ID 21 (RDP session creation request), 22 (Shell start), 23 (Session connection), 24 (Session disconnection), 25 (Session reconnection)
    - RCM ID 1149 (Remote user authentication starts)
  - **STEP 3)** Build a timeline for RDP session start and end
  - **STEP 4)** Report the results

Analysis Environment and Execution Method

Analysis Environment and Execution Method

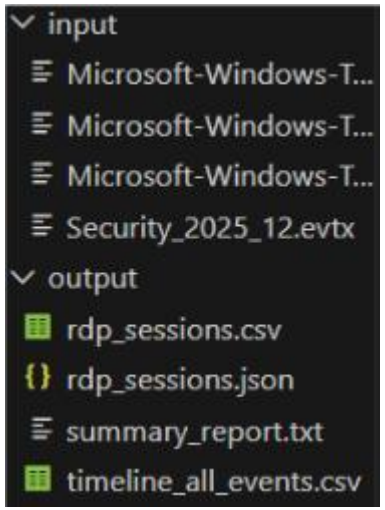
RDP Analyzer Architecture

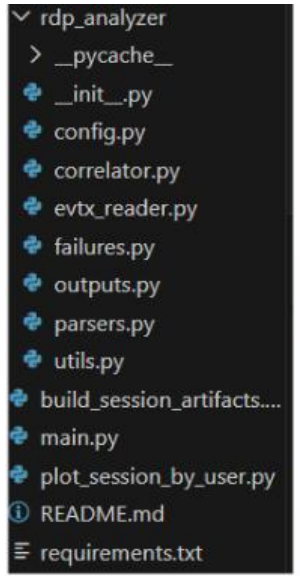


Analysis Environment and Execution Method

Program Environment and Execution Method

- **Program Version and IDE**
  - Program Version: Python 3.10.12
  - Install Library: **pip install –r requirements.txt**
  - IDE: VS Code





- **Execution Method**
  - **STEP 1) Run main.py:** **python3 main.py --security input/Security_2025_12.evtx --lsm input/Microsoft-Windows-TerminalServices-LocalSessionManager%4Operational_2025_12.evtx --rcm input/Microsoft-Windows-TerminalServices-RemoteConnectionManager%4Operational_2025_12.evtx --rdpclient input/Microsoft-Windows-TerminalServices-RDPClient%4Operational_2025_12.evtx --out output**
  - **STEP 2) Run build_session_artifacts.py:** **python3 build_session_artifacts.py --timeline output/timeline_all_events.csv --outdir output**
  - **STEP 3) Run plot_sessions_by_user.py:** **python3 plot_sessions_by_user.py --csv output/rdp_session_summary_v2.csv --outdir output**

Core Analysis Logic

Core Logic

Core of evtx_reader.py

- **Description**
  - Iterates through an **EVTX** file **record by record** and extracts **XML**
  - Extracts **raw XML records,** and modularizes the code so that **parser.py** is responsible for **interpreting/decoding** the meaning of the records.

```python
def iter_evtx_records(evtx_path: str):
    """
    Yield (event_id, channel, timestamp, eventdata_dict, raw_xml,string)
    """

    with Evtx(evtx_path) as log:
        for record in log.records():
            xml_str = record.xml()
            try:
                root = etree.fromstring(xml_str.encode("utf-8"))
            except Exception:
                continue

            ns = {"e": "http://schemas.microsoft.com/win/2004/08/events/event"}
            sys_node = root.find("e:System", namespaces=ns)
            if sys_node is None:
                continue
            eid_node = sys_node.find("e:EventID", namespaces=ns)
            event_id = int(eid_node.text) if eid_node is not None else None

            channel_node = sys_node.find("e:Channel", namespaces=ns)
            channel = channel_node.text if channel_node is not None else None

            time_node = sys_node.find("e:TimeCreated", namespaces=ns)
            ts = safe_dt(time_node.attrib.get("SystemTime")) if time_node is not None else None
```

Core Logic

Core of parser.py

- **Description**
  - Since the **Security / RCM / LSM** logs have different event structures, separate parsers are implemented for each channel to extract the required fields.
  - Also, because the formatting differs by log type, **normalization** is performed.

```python
def parse_security_event(event_id, ts, d, raw_xml):
    return {
        "timestamp": ts,
        "source": "Security",
        "event_id": event_id,
        "username": d.get("TargetUserName") or d.get("SubjectUserName") or d.get("AccountName"),
        "domain": d.get("TargetDomainName") or d.get("SubjectDomainName"),
        "ip": normalize_ip(d.get("IpAddress") or d.get("SourceNetworkAddress")),
        "workstation": d.get("WorkstationName"),
        "logon_type": d.get("LogonType"),
        "logon_id": d.get("TargetLogonId") or d.get("LogonId") or d.get("SubjectLogonId"),
        "status": d.get("Status"),
        "substatus": d.get("SubStatus"),
        "raw_xml": raw_xml,
    }
```

Core Logic

Core of correlation.py

- **Description**
  - Detects RDP sessions based on **Security Event Log** using Event ID 4624 and LogonType 10 as the criteria

```
for ev in security_events:
    if ev["event_id"] != 4624:
        continue

    lt = ev.get("logon_type")
    if not lt or str(lt).strip() != "10":
        continue

    start_ts = ev["timestamp"]
    username = ev["username"]
    domain = ev["domain"]
    ip = ev["ip"]
    workstation = ev["workstation"]
    logon_id = ev["logon_id"]

    end_ts = logoff_by_logonid.get(logon_id)
```

Core Logic

Core of correlation.py

- **Description**
  - Cross-validates the RDP sessions detected in the previous slide with the RDP sessions detected from RCM
  - In **RCM**, cross-validation is performed using **Event ID 1149,** and a score is assigned by evaluating the similarity of the **RDP connection time**

```python
def score(e):
    s = 0
    if username and e["username"] and str(username).lower() == str(e["username"]).lower():
        s += 2
    if ip and e["ip"] and ip == e["ip"]:
        s += 2
    s += max(0, 1 - abs((e["timestamp"] - start_ts).total_seconds()) / win.total_seconds())
    return s
```

Core Logic

Core of correlation.py

- **Description**
  - If **LogonType == 10** is not available, determine the session based on RCM Event ID 1149
  - Group sessions by **user and the same IP address** based on the corresponding Event ID criteria

```
sessions.append({
    "session_start": start_ts.isoformat() if start_ts else None,
    "session_end": end_ts.isoformat() if end_ts else None,
    "duration_sec": duration,
    "username": username,
    "domain": domain,
    "client_ip": ip,
    "workstation": workstation,
    "session_id": session_id,
    "evidence_basis": "Security4624(LogonType=10)",
    "logon_id": logon_id,
    "admin_priv_4672": logon_id in adminpriv_by_logonid,

    "auth_event_time_1149": rcm_match["timestamp"].isoformat() if rcm_match and rcm_match["timestamp"] else None,
    "auth_client_ip_1149": rcm_match["ip"] if rcm_match else None,
    "auth_client_name_1149": rcm_match["client_name"] if rcm_match else None,

    "disconnect_count": disconnect_count,
    "reconnect_count": reconnect_count,
    "lsm_events": [
        {
            "timestamp": x["timestamp"].isoformat() if x["timestamp"] else None,
            "event_id": x["event_id"],
            "session_id": x.get("session_id"),
            "ip": x.get("ip")
        }
        for x in lsm_list
    ]
})
```

Results and Evaluation

Results and Evaluation

Results

session_st	session_er	duration_s	username	domain	client_ip	workstatio	session_id	evidence_l	logon_id	admin_priv	auth_even	auth_client	auth_client	disconnec	reconnect_	lsm_event	auth_events_count
2025-12-0	2025-12-0	1808	pcj		163.152.125.134		5e84ff282	RCM1149(Fallback: Se		FALSE	2025-12-0	163.152.125.134		0	0	[]	2
2025-12-0	2025-12-0	1800	UNKNOWN		163.152.10.117		09fab8dd2	RCM1149(Fallback: Se		FALSE	2025-12-0	163.152.10.117		0	0	[]	1
2025-12-0	2025-12-0	1800	UNKNOWN		163.152.10.117		2943561ef	RCM1149(Fallback: Se		FALSE	2025-12-0	163.152.10.117		0	0	[]	1
2025-12-0	2025-12-0	1800	pcj		163.152.71.74		836b3af2b	RCM1149(Fallback: Se		FALSE	2025-12-0	163.152.71.74		0	0	[]	1
2025-12-0	2025-12-0	1881	pcj		163.152.125.134		5fb86c049	RCM1149(Fallback: Se		FALSE	2025-12-0	163.152.125.134		0	0	[]	2
2025-12-0	2025-12-0	1800	pcj		163.152.125.134		0348fcc74	RCM1149(Fallback: Se		FALSE	2025-12-0	163.152.125.134		0	0	[]	1
2025-12-0	2025-12-0	1800	pcj		163.152.125.134		a2fd84cec	RCM1149(Fallback: Se		FALSE	2025-12-0	163.152.125.134		0	0	[]	1

rdp_sessions.csv


```
[
  {
    "session_start": "2025-12-02T06:10:53.780542",
    "session_end": "2025-12-02T06:41:02.701029",
    "duration_sec": 1808,
    "username": "pcj",
    "domain": null,
    "client_ip": "163.152.125.134",
    "workstation": null,
    "session_id": "5e84ff282908",
    "evidence_basis": "RCM1149(Fallback: Security4624Type10 not found)",
    "logon_id": null,
    "admin_priv_4672": false,
    "auth_event_time_1149": "2025-12-02T06:10:53.780542",
    "auth_client_ip_1149": "163.152.125.134",
    "auth_client_name_1149": null,
    "disconnect_count": 0,
    "reconnect_count": 0,
    "lsm_events": [],
    "auth_events_count": 2
  },
```


rdp_sessions.json

Results and Evaluation

Evaluation

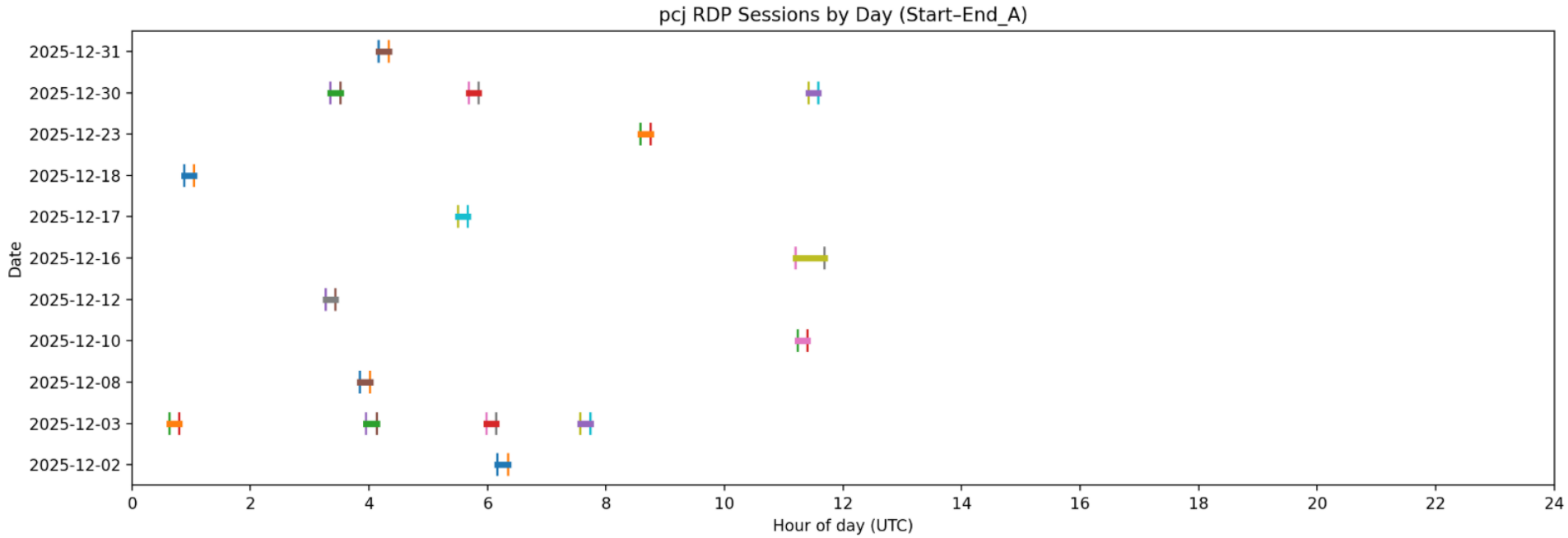
- **Total Count of RDP**
  - Based on the analysis results of the **.csv** and **.json files,** a total of **36 RDP sessions were identified.**
- **Number of Sessions by IP**
  - 163.152.10.117은 Evidence of **20** RDP sessions found
  - 163.152.125.134는 Evidence of **15** RDP sessions found
  - 163.152.71.74는 Evidence of **1** RDP session found
- **Users who used RDP**
  - Confirmed that the users **UNKNOWN** and **pcj** used RDP
  - Since there are **three IP addresses,** we can infer that a specific user used who **different IP addresses**
  - Additional analysis confirmed that the user **pcj** used **two IP addresses**

```
=== RDP Analysis Summary Report ===

Total RDP Success Sessions: 36
Unique Usernames: 2
Unique Client IPs: 3

[Top Client IPs from Successful Sessions]
- 163.152.10.117: 20 sessions
- 163.152.125.134: 15 sessions
- 163.152.71.74: 1 sessions

[Users from Successful Sessions]
- UNKNOWN
- pcj

=== Failure Attempts (4625 LogonType=10) ===
No RDP failure attempts found.
```

Results and Evaluation

Evaluation of Graph

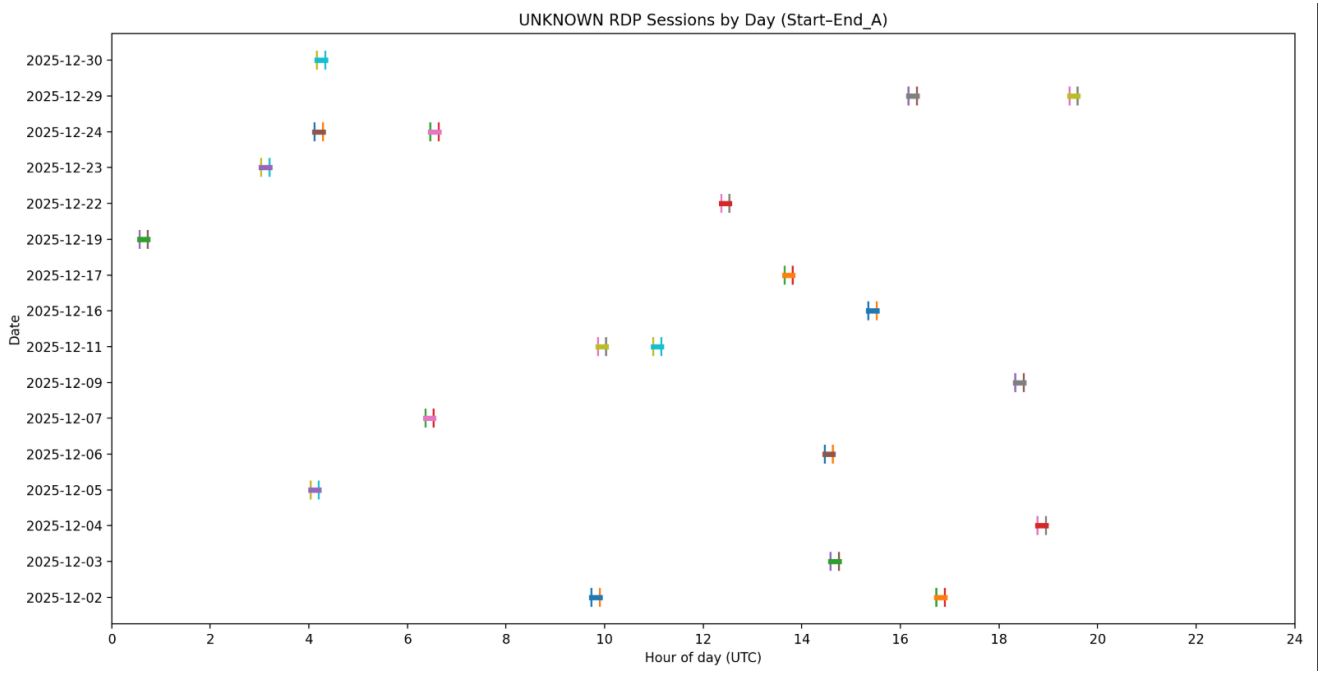
- **Analysis pcj graph**
  - Based on the graph analysis results, it was confirmed that the user **pcj** used RDP from **December 2, 2025 to December 31, 2025.**
  - The user mainly used RDP between **00:00 and 12:00,** and the average RDP session duration was confirmed to be approximately **15 minutes**



Results and Evaluation

Evaluation of Graph

- **Analysis UNKNOWN graph**
  - Based on the graph analysis results, it was confirmed that the user **UNKNOWN** used RDP from **December 2, 2025 to December 30, 2025.**
  - The user mainly used RDP between **00:00 and 20:00,** and the average RDP session duration was confirmed to be approximately **15 minutes**



Additional Artifacts

Additional Artifacts

Client Side

- **Registry**
  - HKCU₩Software₩Microsoft₩Terminal Server Client₩Servers
  - HKCU₩Software₩Microsoft₩Terminal Server Client₩Default
  - HKCU₩Software₩Microsoft₩Terminal Server Client₩MRU
  - RDP connection history can be identified through these three registry locations
- **Jump Lists**
  - In some cases, RDP usage traces may remain in **AutomaticDestinations**
- **Traces of .rdp files**
  - Traces of .rdp files can be found in **Recent Files (MRU)**
  - Traces of .rdp files can also be found in the **Downloads** folder and **Documents** folder

Additional Artifacts

Server Side

- **Registry**
  - HKLM₩SYSTEM₩CurrentControlSet₩Control₩Terminal Server
  - HKLM₩SYSTEM₩CurrentControlSet₩Control₩Terminal Server₩WinStations₩RDP-Tcp
  - HKLM₩SOFTWARE₩Microsoft₩Terminal Server Client
  - RDP connection history can be identified through these three registry locations
- **Network Device Logs**
  - In a Client-Server RDP architecture, remote access is possible through a shared/external network
  - Therefore, analyzing network device logs can help identify RDP-related traces
- **Firewall Logs**
  - Can be analyzed using **Pfirewall.log**
  - RDP traces can be identified through records of inbound connections on **port 3389**

Limitation and Conclusion

Limitation and Conclusion

Limitation

- **Limitation of RDP Analyzer**
  - This program attempted to detect RDP connection traces based on **Security Event ID 4624, 4625** and **LogonType 10**.
  - However, in some cases **LogonType 10 does not exist**, so most results were inferred using **RCM Event ID 1149**.
  - Therefore, the remote access sessions were reconstructed based on **RCM authentication success events** and **session state events (LSM 24, 25)**.
  - As a result, it is not possible to determine RDP usage with **100% certainty**, and there is a limitation that requires additional analysis of other artifacts beyond the event logs mentioned in earlier slides.

Limitation and Conclusion

Conclusion

- **Reflections While Working on Problem #01**
  - This was the **first assignment** I completed through the graduate school admission seminar before graduating from my undergraduate program. Compared to typical undergraduate coursework, I felt it was a **more practical and applied project.** Since I have always been interested in **digital forensics,** working on a project related to that topic was personally enjoyable.
  - Although there are **various techniques** for analyzing numerous artifacts, I realized that automating the process through programming is an efficient approach. I also felt that Python is a powerful language that can perform such tasks effectively, and that **continuous study is necessary.**
  - I'm not sure what kinds of assignments I will encounter as I continue participating in the seminar, but I already feel excited about them. After entering the program in March, I hope to begin my two-year master's program with **strong motivation,** do my best to produce good results, and have a successful graduate school experience.

References

[1] Microsoft. (n.d.). *4624(S): An account was successfully logged on.* Microsoft Learn. Retrieved January 8, 2026, https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4624

[2] Microsoft. (n.d.). *Audit logon events.* Microsoft Learn. Retrieved January 8, 2026, https://learn.microsoft.com/ko-kr/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/basic-audit-logon-events

[3] Splunk. (2025, November 20). *Detection: Windows RDP Connection Successful.* Splunk Security Content. https://research.splunk.com/endpoint/ceaed840-56b3-4a70-b8e1-d762b1c5c08c/

[4] Ballenthin, W. (n.d.). *python-evtx: Pure Python parser for Windows Event Log files (.evtx).* GitHub. Retrieved January 8, 2026, https://github.com/williballenthin/python-evtx

[5] ManageEngine. (n.d.). *What is logon type 10?* Retrieved January 8, 2026, https://www.manageengine.com/products/active-directory-audit/kb/what-is/logon-type-10.html

Thank You