國立臺灣師範大學

理學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Science

National Taiwan Normal University

Master's Thesis

提取日誌資料生成溯源圖

Extracting Log Data for Provenance Graph Generation

李永豐

Lee, Yung-Feng

指導教授：官振傑 博士

Advisor：Albert Guan, Ph.D.

中華民國 114 年 6 月

June 2025

摘要

隨著資訊安全防護日益重要，系統日誌的分析扮演關鍵角色，能協助即時偵測異常行為與追蹤潛在攻擊路徑。本研究旨在透過 Auditbeat 蒐集 Linux 系統的審計日誌（Audit Log），並結合 Elasticsearch 與 Kibana 建立可視化分析環境，進一步使用 Python 與 Graphistry 建構 Provenance Graph（溯源圖），以圖形化方式揭示系統中事件間的因果關係。本研究以實際攻擊模擬方式，結合資安事件進行溯源分析。首先透過 Auditbeat 收集檔案操作、程序建立與網路連線等系統層級事件，將資料傳送至 Elasticsearch 並以 Kibana 進行初步視覺化。為強化分析效率，進一步使用 Python 與 Pandas 擷取關鍵欄位，並透過 Graphistry 建構事件溯源圖，呈現節點間的因果關係與行為路徑。此外，研究亦模擬 Subrion CMS 中 CVE-2018-19422 漏洞，攻擊者上傳惡意 .phar 檔案取得 reverse shell，Auditbeat 可偵測其檔案異動與非預期連線行為，並於溯源圖中呈現 apache2 觸發 shell 的完整路徑。本研究亦納入釣魚攻擊模擬場景，誘導使用者輸入帳密並下載含有反向連線功能的惡意 ELF 檔案。受害者執行該檔後成功建立 Meterpreter 控制連線，進而進行檔案操作與系統控制。相關行為全程被 Auditbeat 紀錄，並以溯源圖方式具體呈現從網頁誘騙到系統被控的完整流程，驗證本方法於異常行為還原與關鍵節點追蹤之可行性。總結而言，本研究提出一套自動化、可視化的系統日誌分析流程，不僅能有效提取安全關鍵事件，亦能協助資安人員以溯源圖方式掌握攻擊者行為軌跡。此方法可應用於入侵偵測與安全監控領域，為企業與研究機構提供具實用性的日誌分析與威脅溯源工具。

關鍵詞：Auditd、Auditbeat、日誌分析、Provenance graph。

ABSTRACT

As information security becomes increasingly critical, system log analysis plays a pivotal role in detecting abnormal behavior and tracing potential attack paths in real time. This study proposes a methodology for analyzing Linux audit logs by leveraging Auditbeat to collect system events and integrating Elasticsearch and Kibana to construct a visualized analysis environment. Furthermore, Python and Graphistry are utilized to build provenance graphs, which reveal causal relationships between system events in a graphical manner.This study conducts provenance analysis through simulated real-world cyberattacks. Auditbeat is utilized to collect system-level events, including file operations, process creation, and network connections. The data is then transmitted to Elasticsearch and initially visualized using Kibana. To enhance analytical efficiency, key fields are extracted using Python and Pandas, and a provenance graph is constructed with Graphistry to illustrate the causal relationships and behavioral paths between entities. In addition, the study simulates an exploitation of the CVE-2018-19422 vulnerability in Subrion CMS, wherein the attacker uploads a malicious .phar file to obtain a reverse shell. Auditbeat effectively detects the related file modifications and unexpected outbound connections, while the provenance graph clearly visualizes the process path in which Apache2 triggers the shell execution.The study also incorporates a phishing attack scenario, deceiving the user into submitting credentials and downloading a malicious ELF file with reverse shell capabilities. Upon execution, the backdoor establishes a Meterpreter session, granting the attacker control over the victim's system, including file manipulation and command execution. All related activities are captured by Auditbeat and represented in the provenance graph, successfully demonstrating the feasibility of the proposed approach in reconstructing abnormal behavior and identifying key attack nodes. In conclusion, this research presents an automated and visualized system log analysis pipeline capable of extracting security-critical events and facilitating the tracing of attacker behavior through provenance graphs. The proposed method demonstrates practical value in the domains of intrusion detection and security monitoring, offering a valuable tool for enterprises and research institutions in log analysis and threat attribution.

Keywords: Auditd, Auditbeat, Log Analysis, Provenance Graph

目錄

表目錄

圖目錄

第一章 緒論

1.1 研究動機

隨著資安威脅日益複雜，系統層級的行為監控成為防禦 APT（Advanced Persistent Threat）等進階攻擊以及各種攻擊手法的重要方式 [1] [2]。Auditd [3] 是 Linux 系統中的核心稽核工具，系統稽核日誌是端點監控系統中追蹤作業系統活動的關鍵數據來源 [4]。系統稽核日誌程序在作業系統的特權模式（Privileged mode）下運行，理論上可以記錄多數正常或惡意應用程式與作業系統互動的行為。然而，在實務上，具備技術能力的攻擊者仍可透過各種手段，如記憶體內執行 [5]（fileless execution）、直接系統呼叫 [6]、或關閉監控機制 [7]等方式，部分或完全規避稽核系統的監控與紀錄。因此，僅依賴單一層級的稽核機制可能不足以全面揭露攻擊行為。而 Elastic Stack [8]中的 Auditbeat [9] 則提供更彈性與現代化的事件蒐集方式。雖然 Auditbeat 可以完整蒐集 audit log，但面對大量事件資料使資安分析專家難以將事件彼此關聯，如何將大量事件紀錄有效提取關鍵資訊並理解事件之間的關聯，仍是一大挑戰 [10]。

1.2 研究目的

使用溯源圖來視覺化事件之間的因果關係，有助於提升事件分析效率與威脅鑑別能力。儘管 SPADE (Scalable Provenance Auditing for Distributed Environments) [11] 於資料溯源領域具有高度學術價值與完整性，但其高部署成本與資源消耗限制了實務應用的彈性 [12]。 SPADE 在大量分散式系統中產生高解析度溯源時，會造成大量元數據（metadata）與儲存壓力，而本研究方法可根據分析需求選擇性擷取欄位，並以 Python 的 Pandas 進行前處理，有效提升效能與可維運性。本研究旨在建立一套流程：使用 Auditbeat 蒐集系統稽核日誌，並透過 Python 分析與處理事件資料，進一步生成溯源圖，在可視化呈現與實務整合性方面具有較佳的操作性與擴充性，讓使用者能夠更直觀地理解系統活動的脈絡與可疑行為的來源。

第二章 背景知識與文獻探討

本章將探討本研究相關領域之文獻與技術背景，包括系統稽核與日誌分析、Auditbeat 與 Elastic Stack 架構、Python 在資安資料處理的應用，以及溯源圖技術在資安事件視覺化中的角色。透過相關文獻的回顧，可瞭解現有技術的優劣，並確認本研究的發展方向與創新貢獻。

2.1 系統稽核與日誌分析技術

系統稽核（System Auditing）是資訊安全中重要的防禦與鑑識機制，可用以記錄系統操作行為、追蹤使用者活動與事件發生過程。 Linux 系統中內建的 auditd 提供細緻的稽核功能，可針對系統呼叫、使用者登入、檔案存取、權限變更等活動進行監控並生成 audit log。根據文獻 [13]，系統稽核日誌為威脅鑑識提供了關鍵依據，然而，其資料結構複雜且資料量龐大，對事件分析帶來相當挑戰。

日誌分析技術發展迅速，許多研究著重於從海量日誌中自動提取異常模式與事件關聯，常見方法包括規則比對、異常檢測與機器學習。系統稽核日誌雖可準確記錄所有操作細節，但缺乏整合與視覺化能力，導致分析者難以快速掌握攻擊路徑與影響範圍。為解決此問題，將稽核日誌轉換為結構化資料模型，以利後續分析與事件重建。此外，透過圖形化方式呈現事件間的因果關係，可有效輔助分析者掌握系統行為流程與攻擊鏈，提升事件理解。若能結合可視化工具與自動化日誌處理機制，將有助於建構具可解釋性與實用性的安全監控分析架構。

2.2 Auditbeat 與 Elastic Stack 的應用

Auditbeat 為 Elastic Stack 所提供的一款輕量級代理程式，可用來即時收集系統稽核事件，並具備模組化架構（如 auditd module、system module），支援將稽核資料送往 Elasticsearch [14] 進行儲存與查詢如。Auditbeat 結合 Kibana 可進行事件統計視覺化，有效提升事件鑑識效率。

相較於傳統使用 log 檔案分析的方法，使用 Elastic Stack 可提供即時、集中化、可擴充的日誌管理與分析平台。文獻 [15]亦指出，Elastic Stack 在資安監控應用中能有效整合多種資料來源，對於建立統一的可視化監控介面具有顯著效益。

2.3 Python 在資安資料處理的應用

Python 由於其語法簡潔、套件豐富，成為處理 JSON 結構日誌與進行資料分析的主要工具之一。特別是 pandas、re、json、networkx 等套件廣泛應用於日誌的解析、事件欄位的提取與資料關聯建構。根據文獻 [16]，Python 在資訊安全自動化、數位鑑識與日誌關聯分析中扮演核心角色。

本研究即採用 Python 撰寫日誌分析模組，擷取與系統行為相關的欄位，並結合圖形資料模型建構行為關聯關係。

2.4 溯源圖在資安視覺化中的應用

Provenance Graph，又稱資料溯源圖，用以表示事件之間的因果關係與行為序列。此圖形可將複雜的系統活動轉化為節點（事件主體與客體）與邊（事件操作）的形式，提供使用者直觀理解整體行為脈絡的能力。根據文獻 [17] [18]，溯源圖在 APT 攻擊、惡意程式行為分析等場景中具備高可視性與分析價值。

Graphistry [19]作為一種圖形視覺化平台，能夠針對節點間的大量關係進行互動式繪圖，適用於大規模日誌資料的圖形探索與威脅識別。

第三章 研究方法

在建構系統溯源圖的過程中，最關鍵的核心在於釐清每一筆事件的主體（Subject）、行為（Action 或 Event）、客體（Object）以及事件發生的相關上下文資訊（Context），例如時間、來源 IP、使用者身分等。這些資訊組合成一個可被視覺化的節點與邊的資料結構，便能協助安全分析人員從圖形中快速辨識出系統中潛在的異常或惡意行為，進而達成事件溯源、攻擊路徑重建與行為追蹤等目標。然而，從系統所產生的原始稽核日誌（audit logs）中提取上述資訊並不容易，因為每筆事件所攜帶的欄位（fields）眾多，且記錄格式不一，實際對安全分析有價值的資訊往往散落在各個欄位之中。因此，為了簡化分析流程與避免不必要的處理負擔，本研究在事件解析階段會優先針對與溯源相關的欄位進行提取與分類。舉例來說，檔案操作事件會包含 file.path, file.name, file.owner, file.mode 等欄位；而與網路連線相關的事件則包含 destination.ip, destination.port, source.ip, network.transport 等欄位；程序執行事件則包含 process.name, process.pid, process.executable 等欄位。在區分不同事件類型時，關鍵的欄位是 event.category。此欄位會根據事件的性質自動填入標籤，常見的類別包含，file：與檔案存取、建立、刪除、修改等行為有關，process：與程序的啟動（exec）、終止（exit）或 fork 行為有關，network：與網路連線、socket 操作（如 bind、connect、accept）相關，user：與使用者登入登出、身分驗證等活動有關，authentication：密碼驗證、key 驗證等行為。Auditbeat 在事件紀錄方面比傳統的 Auditd 更具便利性與彈性。首先，Auditbeat 支援模組化設計，其內建的 system 模組能自動記錄登入行為、程序生命周期（啟動與終止）、使用者 ID 變化等資訊，無須像 Auditd 一樣需手動撰寫與部署繁瑣的 audit 規則。更進一步，Auditbeat 能支援跨平台使用，除 Linux 外亦可應用於 macOS 或 Windows 系統，方便企業在多種環境中進行一致的日誌監控。

3.1 系統整體架構

圖 1 所示為本研究系統的整體架構與分析流程，其設計主要分為三個階段，分別為日誌資料蒐集、日誌資料查詢與欄位處理，以及溯源圖建構與視覺化呈現。第一階段為日誌資料蒐集階段，在目標主機上部署輕量級的稽核代理程式 Auditbeat，該工具可搭配 Linux 系統內建的稽核子系統 Auditd，並利用其內建模

組如 auditd 與 system 模組，主動蒐集系統層級的重要活動事件，例如程序的建立與終止、檔案的開啟與操作、網路連線的發生、使用者行為等。這些稽核日誌資料將會被即時傳送至資料儲存後端 Elasticsearch 中進行集中化儲存與管理，Elasticsearch 為一套具備強大查詢功能的全文檢索與分析引擎，可支援大量結構化與半結構化資料的快速查詢與分析。另一方面，為了即時觀測系統稽核日誌的分佈與類型，可使用資料視覺化介面 Kibana，透過 Kibana 將 Elasticsearch 中儲存的日誌資料以圖表、表格與儀表板的方式即時呈現，協助使用者了解系統整體狀態與潛在異常事件。第二階段為日誌查詢與欄位處理階段，透過撰寫 Python 程式碼搭配 Elasticsearch API，向指定的 Elasticsearch 索引發送查詢請求，篩選特定時間範圍內的日誌資料。在此階段，會透過查詢語法提取與溯源分析相關的重要欄位，例如事件類型（如 syscall 名稱）、程序名稱（process name）、程序 ID（PID）、父程序 ID（PPID）、檔案路徑(file path)、來源與目的 IP 位址(source & destination IP)等。接著使用 Python 的 Pandas 套件 將取得的 JSON 結構化日誌資料轉換為資料表格形式，並進行欄位清洗（如空值填補、資料型別轉換等），以利後續資料處理與建構節點與邊的圖形結構。第三階段為 溯源圖建構與視覺化階段，此階段的重點在於根據日誌中的事件類型與欄位資訊，辨識出系統中主體（如程序）與客體（如檔案、網路、程序）之間的互動關係。例如程序執行 open 系統呼叫存取某檔案，即可建立一條有向邊代表程序開啟檔案的邊或程序與遠端 IP 建立連線，則建立一條程序連線至網路的有向邊。依據不同事件類型（Process、File、Network）進一步區分節點類型與邊的屬性（如行為類型、時間戳記、來源與目的端等），並透過 Graphistry 套件 將這些節點與邊組成互動式溯源圖。Graphistry 提供雲端圖形渲染與視覺化介面，讓使用者能以圖形方式觀察系統活動的因果關係與流程鏈，進一步掌握可能的入侵路徑、攻擊工具執行過程、檔案存取紀錄與網路連線方向等安全事件關聯資訊。整體而言，此系統架構整合了稽核資料收集、日誌查詢與處理、以及視覺化溯源圖建構三個核心模組，能夠有效協助使用者對 Linux 系統的活動事件進行高解析度的安全溯源與行為分析，有助於發現隱匿的入侵行為與異常模式。

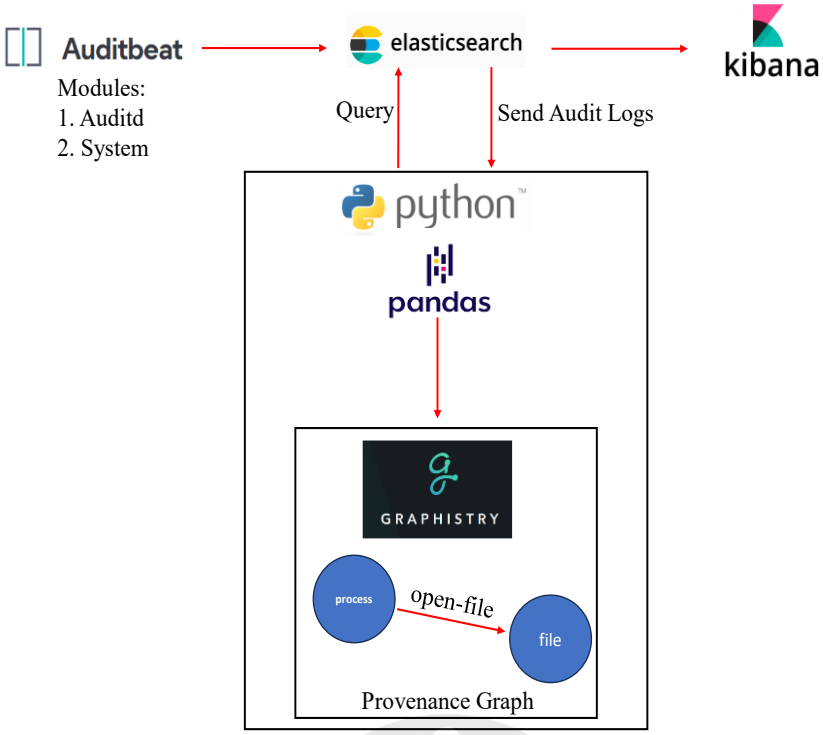


圖 1、系統日誌分析與生成溯源圖流程

3.2 資料收集與查詢

在本研究的系統設計中，資料的收集與查詢流程為整個溯源分析的核心起點，其目的在於從日誌層級全面掌握系統中發生的各項安全事件，並為後續的圖形建構與視覺化分析提供基礎資料來源。該流程主要透過部署於目標主機上的 Auditbeat 工具進行資料蒐集，Auditbeat 為 Elastic Stack 提供的一個輕量級代理程式，可持續性地監控 Linux 系統內部發生的各項行為，並將這些事件即時地上傳至後端資料儲存與分析平台 Elasticsearch。Auditbeat 所蒐集的事件類型涵蓋範圍廣泛，包括程序的啟動與終止、檔案的開啟與修改、網路連線的建立、使用者登入與執行的指令等，每一筆事件記錄皆會包含時間戳記、事件類別、行為類型、執行程序名稱與 ID、執行檔路徑、使用者名稱與 ID、檔案路徑與權限資訊、來源與目的 IP 位址與 Port 等多種欄位，這些欄位正是後續進行溯源圖建構的核心要素。為了有效查詢這些稽核日誌資料，研究中使用 Python 搭配官方提供的 Elasticsearch 函式庫撰寫程式碼，先透過 connect_to_elasticsearch() 函數如圖 2 進行連線測試，此函數會接收一個 Elasticsearch 客戶端實例，並利用 .ping() 方法確認連線是否成功。如果連線成功，便會輸出提示訊息“Successfully connected to

Elasticsearch!”；若失敗，則會捕捉錯誤訊息並回傳 None，以利除錯與問題排查。這樣的檢查機制可確保後續查詢動作建立在穩定連線的基礎上。在完成連線之後，便可透過 query_auditbeat_data()函數如圖 3 針對特定的時間範圍與索引（例如 auditbeat-*）進行日誌查詢。該函數設計了一個複雜的查詢結構，利用 Elasticsearch 的 DSL（Domain Specific Language）語法指定查詢條件，其中包括使用 range 篩選出指定時間段內的所有稽核事件，並透過 _source 欄位明確指定欲擷取的欄位資料，以避免不必要的欄位造成查詢負擔。被提取的欄位涵蓋多個面向，包括事件類別（event.category）、行為名稱（event.action）、程序資訊（process.name、process.pid、process.parent、process.start）、使用者資訊（user.name、user.id）、檔案資訊（file.path、file.owner、file.mode）、網路資訊（source.ip、destination.ip、destination.port、network.protocol、network.direction）等。如此完整的欄位設定，能夠確保查詢回傳的每筆資料都具備足夠的資訊進行事件關聯與因果推導。查詢完成後，程式會將回傳的 Elasticsearch 結果轉換為 Python 的 JSON 物件，再利用 pandas.DataFrame 將其轉為結構化的表格資料格式，以利後續進行欄位清洗、資料篩選、時間排序與邊的建構等資料處理步驟。此外，由於 Auditbeat 每筆事件紀錄皆含有嚴謹的時間戳記與豐富欄位內容，使得本系統得以高解析度地還原整體系統行為脈絡，對於入侵事件、可疑連線、資料外洩等異常狀況的識別與重建極具幫助。透過這樣的資料收集與查詢設計，本研究奠定了系統活動可視化分析的穩固基礎，讓使用者得以在圖形化介面中有效觀察系統內部的行為流程、實體交互與潛在攻擊路徑。

```
def create_es_client(hosts, username, password):
    """
    :param hosts: Elasticsearch 的 URL 列表
    :param username: 用戶名
    :param password: 密碼
    :return: Elasticsearch 客戶端對象
    """
    return Elasticsearch(hosts=hosts, basic_auth=(username, password))

#嘗試連接到 Elasticsearch 並檢查連接是否成功-----------------------------
def connect_to_elasticsearch(es):
    """

    :param es: Elasticsearch 客戶端對象
    :return: 如果連接成功，返回 Elasticsearch 客戶端，否則返回 None
    """
    try:
        # 連接elasticsearch
        if es.ping():
            print("Successfully connected to Elasticsearch!")
        else:
            print("Failed to connect to Elasticsearch.")
        return es
    except Exception as e:
        print(f"Error connecting to Elasticsearch: {e}")
        return None
```

圖 2、此程式碼為向 Elasticsearch 連線並且 query 系統上的未過濾的日誌資料

```
def query_auditbeat_data(es, index, start_time, end_time):
    """

    :param es: Elasticsearch 客戶端對象
    :param index: 索引名稱
    :param start_time: 查詢的起始時間
    :param end_time: 查詢的結束時間
    :return: 查詢結果的 Pandas DataFrame
    """
    query = {
        "size": 10000,
        "_source": ["@timestamp",
                    "event.category", "event.action", "event.type", "event.outcome",
                    "process.name", "process.pid", "process.parent", "process.start","process.executable",
                    "auditd.data", "user.name", "user.id", "file.path", "file.owner",
                    "file.mode", "network.protocol", "network.direction", "source.ip",
                    "destination.ip", "destination.port", "network.transport", "network.type"],
        "query": {
            "range": {
                "@timestamp": {
                    "gte": start_time,
                    "lte": end_time
                }
            }
        }
    }

    response = es.search(index=index, body=query)
    data = [hit["_source"] for hit in response["hits"]["hits"]]
    return pd.DataFrame(data)
```

圖 3、此程式碼為提取特定需要的日誌資料

3.3 日誌處理與欄位萃取

圖 4 程式碼是設計來處理由 Auditbeat 所收集並儲存在 Elasticsearch 中的原始稽核資料，這些資料被轉換為 Pandas 的 DataFrame 格式後進行欄位提取與轉換處理。由於 Auditbeat 所輸出的 JSON 結構具有高度的巢狀特性（nested JSON），其中每筆資料往往包含多層物件，例如 process.parent.pid 或 file.owner 等欄位無法直接以扁平化 (flattening) 方式取得，因此本函式 process_auditbeat_data(df) 利用了自定義的輔助函式 safe_get()，用來遞迴式地從巢狀字典中安全地提取所需的值。這使得在大量稽核資料處理過程中能夠確保穩定性與準確性。程式中首先定義了一個字典 columns_to_extract，這個字典對應著目標欄位名稱與其在嵌套 JSON 結構中的路徑，例如 "process_name" 對應的是 ("process", "name")，代表應該從原始資料的 process 欄位中進一步提取 name 子欄位。接著，程式對這些欄位逐一進行迴圈，利用 df.get(path[0]) 先取得欄位的第一層資料，再對每筆資料使用 safe_get() 提取後續巢狀欄位，最終將提取結果儲存至新的 DataFrame 欄位中。接下來的幾行程式碼是針對提取後的資料進行後處理與標準化 df["parent_process_id"] 和 df["process_id"] 在某些情況下可能為缺失值（NaN），因此使用 fillna(0) 補值，再轉換為整數型別以便後續分析與圖形化處理。對於 process_name 欄位，使用了 groupby("process_id") 的方式根據 PID 進

行分組，並透過 .transform(lambda x: x.ffill().bfill()) 使用前向填補與後向填補補齊缺失的欄位名稱，這是為了解決某些事件中 process_name 並未顯示出來但 PID 仍存在的狀況，避免在溯源圖或攻擊行為分析中產生資訊斷裂。此外，destination_port 欄位被強制轉型為字串格式，主要考量是在視覺化或分析過程中可能需要與其他文字欄位整合處理，確保不會因為格式不一致導致比對錯誤。而 event_action 通常為 list 型態（如 ["executed", "opened"]），為了便於儲存與查詢，將其合併為逗號分隔的字串形式，確保資料結構統一、可視化結果清楚易讀。總體來說，這段資料處理流程的核心目的是從多層嵌套的 JSON 結構中系統性地抽取與資安事件溯源相關的重要欄位，並將其整理成一個結構一致、欄位明確且易於分析的平坦化資料表格。這樣的處理不僅能有效提升後續如行為分析、圖形化攻擊路徑建構（如使用 Graphistry）或是異常偵測的準確性，也有助於將大量複雜稽核日誌轉換為具備實用價值的事件資訊，進一步支持資安人員對於系統內部活動的追蹤與理解。欄位處理階段也進行了資料清洗（如欄位轉型、缺值處理、名稱補齊），使得資料能夠一致地轉換為節點與邊如表 1、表 2。

```
def process_auditbeat_data(df):
    """

    :param df: Auditbeat 的原始 DataFrame
    :return: 處理後的 DataFrame
    """
    def safe_get(data, *keys):
        for key in keys:
            if isinstance(data, dict):
                data = data.get(key)
            else:
                return None
        return data

    columns_to_extract = {
        "syscall": ("auditd", "data", "syscall"),
        "event_category": ("event", "category"),
        "event_type": ("event", "type"),
        "event_action": ("event", "action"),
        "event_outcome": ("event", "outcome"),
        "process_executable":("process","executable"),
        "process_name": ("process", "name"),
        "parent_process_id": ("process", "parent", "pid"),
        "parent_process_name": ("process", "parent", "name"),
        "process_id": ("process", "pid"),
        "file_path": ("file", "path"),
        "file_owner": ("file", "owner"),
        "file_mode": ("file", "mode"),
        "source_ip": ("source", "ip"),
        "destination_ip": ("destination", "ip"),
        "destination_port": ("destination", "port"),
        "users_name": ("user", "name"),
        "network_direction": ("network", "direction"),
        "network_transport": ("network", "transport"),
        "network_type": ("network", "type")
    }

    for new_col, path in columns_to_extract.items():
        df[new_col] = df.get(path[0], pd.Series([None] * len(df))).apply(
            lambda x: safe_get(x, *path[1:]) if x is not None else None
        )

    df["parent_process_id"] = df["parent_process_id"].fillna(0).astype(int)
    df["process_id"] = df["process_id"].fillna(0).astype(int)
    df["process_name"] = df.groupby("process_id")["process_name"].transform(lambda x: x.ffill().bfill())
    df["destination_port"] = df["destination_port"].astype(str)
    df["event_action"] = df["event_action"].apply(lambda x: ','.join(x) if isinstance(x, list) else str(x))

    return df
```

圖 4、此程式碼為處理 Auditbeat 查詢結果，提取所需的欄位並進行轉換

```
{
  "_index": ".ds-auditbeat-8.16.0-2024.12.18-000002",
  "_id": "PupFTpQBx2t0OnEAtlRN",
  "_version": 1,
  "_score": 0,
  "_source": {
    "@timestamp": "2025-01-10T03:32:49.082Z",
    "auditd": {
      "message_type": "syscall",
      "sequence": 54781,
      "result": "success",
      "data": {
        "tty": "pts0",
        "a1": "556ec6a154e0",
        "a0": "ffffff9c",
        "exit": "0",
        "a2": "0",
        "syscall": "unlinkat",
        "a3": "0",
        "arch": "x86_64"
      },
      "session": "3",
      "summary": {
        "how": "/usr/bin/rm",
        "actor": {
          "primary": "lee",
          "secondary": "root"
```


圖 5、Elasticsearch 以 JSON 結構儲存從 Auditbeat 所傳送的日誌紀錄

Index	Type	Process_name	File_path	IP addrsss
0	Process	rm (PID: 15875)	NaN	NaN
1	Process	Sudo(PID:15873)	NaN	NaN
2	Network	NaN	NaN	10.0.2.5
3	file	NaN	/etc/passwd	NaN

表 1、日誌資料轉換成 Node dataframe

Index	Timestamp	Subject	Object	action
0	2025-01-11T03:50:29	rm (PID: 15875)	file_event	unlinkat
1	2025-01-11T03:54:02	node (PID: 1104)	140.122.184.88	network_flow
2	2025-01-11T03:51:34	vim (PID: 15903)	hello.cpp	readlink
3	2025-01-11T03:51:22	sudo (PID: 15873)	vim (PID: 15903)	process_create

表 2、日誌資料轉換成 Edge dataframe

3.4 Process ID 重複處理方式

在一台 Linux 系統中，Process ID（PID）在任何一個時間點都是唯一的，不會同時有兩個程序擁有相同的 PID，這是由作業系統核心所強制保證的。然而，PID 並不是永久不變的識別碼，因為它是有限資源。Linux 系統會從 1 開始分配 PID，最大值由 /proc/sys/kernel/pid_max 決定，當系統執行大量短生命週期的程序時，例如大量腳本、批次任務或高併發請求，舊程序結束後所釋放的 PID 就會被回收並分配給新的程序，因此 PID 重複使用的情況是常見且設計上的正常行為。這種重複不是同時出現，而是「不同時間點上相同的 PID 被不同程序使用」。這在安全分析或溯源圖架構中會導致誤判，例如將兩個不同時間的程序誤認為是同一個，尤其是在依賴 PID 追蹤程序關聯時。因此，在建立溯源圖或分析行為時，不能單純依靠 PID，而應結合程序啟動時間（start time）、可執行檔路徑（exe）、父 PID（ppid）等欄位，甚至以這些欄位產生唯一雜湊值或使用系統提供的唯一識別碼，來精準識別每一個程序實例，避免因 PID 重複導致追蹤錯誤。

3.5 檔案事件節點與邊建構

為了呈現系統中程序與檔案間的存取關係，本研究設計 generate_file_edges_and_nodes() 函式如圖 6 ，專門處理來自 Auditbeat 所收集的與檔案操作相關的事件紀錄。此函式的目的是從日誌資料中篩選出所有與檔案行為相關的資料，並建構出一組可視覺化為溯源圖的邊與節點資料結構，進一步分析系統行為與潛在的可疑存取。首先，該函式透過篩選 event_category 欄位中包含 "file" 的紀錄，以識別所有檔案相關事件（如 open、read、write、unlink 等 syscalls）。接著，利用每筆事件的 process_name 和 process_id 欄位組合，建立新的描述欄位 process_id_name，形式為 程序名稱 (PID:程序編號)，作為來源主體（subject）的標示。透過這樣的資料轉換與標準化結構，本函式能夠將原始的檔案操作日誌，轉換為一組可供 Graphistry 等溯源視覺化工具繪製的邊與節點。這使得資安分析者能夠從圖像中直觀辨識出某個特定程序對哪些檔案執行過什麼操作、是否出現過異常或未授權行為，有助於事件溯源與惡意程式追蹤。

```
def generate_file_edges_and_nodes(df):
    file_edges = df[df['event_category'].apply(lambda x: 'file' in x if isinstance(x, list) else False)]
    file_edges["process_id_name"] = file_edges.apply(
        lambda row: f"{row['process_name']} (PID:{row['process_id']})" if row['process_name'] and row['process_id'] else "unknown",
        axis=1
    )

    complete_file_edges = pd.DataFrame({
        "subject": file_edges["process_id_name"],
        "object": file_edges["file_path"].fillna("unknown"),
        "action": file_edges["syscall"].fillna("unknown"),
        "category": file_edges["event_category"],
        "users_name": file_edges["users_name"],
        "edge_category": "file",
        "outcome": file_edges["event_outcome"],
        "time": file_edges["@timestamp"]
    })

    filepath_subject_nodes = pd.DataFrame({
        "node_name": file_edges["process_id_name"],
        "process_name": file_edges["process_id_name"],
        "process_executable":file_edges["process_executable"],
        "source_ip": file_edges["source_ip"],
        "type": "process"
    })

    filepath_object_nodes = pd.DataFrame({
        "node_name": complete_file_edges["object"],
        "file_path": complete_file_edges["object"],
        "file_owner": file_edges["file_owner"],
        "file_mode": file_edges["file_mode"],
        "type": "file"
    })
    filepath_object_nodes=filepath_object_nodes[filepath_object_nodes["node_name"]!="unknown"]
    return complete_file_edges, filepath_subject_nodes,filepath_object_nodes
```

圖 6、此程式碼為處理檔案操作相關事件提取點以及邊

3.6 程序建立節點與邊建構

為了將系統中程序建立（process creation）事件的關聯性視覺化為節點與邊的形式，我們撰寫了 generate_process_create_edges_and_nodes() 函式如圖 7 。該函式的目標是從 Auditbeat 的稽核日誌中篩選出屬於程序事件且為程序啟動（start）類型的資料，並依據這些事件建立完整的溯源關聯。首先，程式使用 Pandas 條件篩選機制，在 event_category 欄位中判斷是否包含 process，以及 event_type 欄位中是否包含 start，以過濾出代表新程序被建立的事件。為確保資料完整性與來源節點準確性，接著會透過 apply() 方法，針對每筆資料呼叫 update_source_with_parent 函式，並以 parent_process_mapping 字典作為參數，用以將來源節點更新為其上層父程序的名稱與 PID，這樣可以讓邊的來源更準確地反映實際的父子程序關係。透過這個函式的設計，我們能夠從稽核日誌中萃取出最具意義的程序建立行為，並以節點與邊的方式建構出其間的事件關聯，進一步協助溯源圖的建立與視覺化呈現，使資安分析人員能夠清楚看出系統中程序的演進脈絡與可疑行為的起始來源。這樣的資料處理步驟也能避免將原始日誌中的冗餘資

訊帶入圖中，讓整體圖形更具可讀性與分析價值。

```
def generate_process_create_edges_and_nodes(df, parent_process_mapping):
    process_create_edges = df[(df['event_category'].apply(lambda x: 'process' in x if isinstance(x, list) else False)) &
                              (df['event_type'].apply(lambda x: 'start' in x if isinstance(x, list) else False))]

    process_create_edges["source_parent"] = process_create_edges.apply(update_source_with_parent, axis=1, args=(parent_process_mapping,))

    complete_process_create_edges = pd.DataFrame({
        "subject": process_create_edges["source_parent"],
        "object": process_create_edges.apply(lambda row: f"{row['process_name']} (PID:{row['process_id']})" if row['process_name'] and row['process_id'] else "unknown", axis=1),
        "action": "process_create",
        "category": process_create_edges["event_category"],
        "users_name": process_create_edges["users_name"],
        "edge_category": "process",
        "outcome": process_create_edges["event_outcome"],
        "time": process_create_edges["@timestamp"]
    })

    process_create_object_nodes = pd.DataFrame({
        "node_name": complete_process_create_edges["object"].fillna("unknown"),
        "process_name": complete_process_create_edges["object"],
        "source_ip": process_create_edges["source_ip"],
        "process_executable":df["process_executable"],
        "type": "process"
    })
    process_create_object_nodes=process_create_object_nodes[process_create_object_nodes["node_name"]!="unknown"]
    return complete_process_create_edges, process_create_object_nodes
```

圖 7、此程式碼為處理 process create 相關事件提取點以及邊

3.7 網路節點與邊建構

為了建構系統中網路連線行為的關聯視覺化圖，我們設計了 generate_network_edges_and_nodes() 函式如圖 8 ，專門處理 Auditbeat 所記錄的網路相關事件。該函式首先從原始日誌資料中，透過 Pandas 的條件篩選機制，挑出 event_category 欄位中包含 "network" 的紀錄，這些紀錄代表系統中發生了如 TCP/UDP 連線、socket 傳輸等網路操作。為了呈現網路活動的來源主體，我們根據每筆事件中所附的程序名稱（process_name）與程序識別碼（process_id），組合成格式為 程式名稱 (PID:程序 ID) 的字串，儲存在新的欄位 process_id_name 中，作為後續邊的 subject 欄位使用。透過本函式處理後，所得到的邊與節點資料結構即可直接輸入至 Graphistry 等視覺化平台中，繪製出網路活動的溯源圖。這能幫助資安分析人員快速辨識出哪些程序與哪個 IP 建立了連線，亦能協助偵測可疑外連或 C2 行為。相較於傳統平面式日誌紀錄，這種視覺化方式更能凸顯關聯性與攻擊路徑。

```
def generate_network_edges_and_nodes(df):
    network_edges = df[df['event_category'].apply(lambda x: 'network' in x if isinstance(x, list) else False)]
    network_edges["process_id_name"] = network_edges.apply(
        lambda row: f"{row['process_name']} (PID:{row['process_id']})" if row['process_name'] and row['process_id'] else "unknown",
        axis=1
    )

    complete_network_edges = pd.DataFrame({
        "subject": network_edges["process_id_name"],
        "object": network_edges["destination_ip"].fillna("unknown"),
        "action": network_edges["event_action"].fillna("unknown"),
        "destination_port": network_edges["destination_port"],
        "network_direction": network_edges["network_direction"],
        "network_transport": network_edges["network_transport"],
        "category": network_edges["event_category"],
        "users_name": network_edges["users_name"],
        "edge_category": "network",
        "outcome": network_edges["event_outcome"],
        "time": network_edges["@timestamp"]
    })
    networkIP_subject_nodes = pd.DataFrame({
        "node_name": network_edges["process_id_name"],
        "process_name": network_edges["process_id_name"],
        "process_executable":network_edges["process_executable"],
        "source_ip": network_edges["source_ip"],
        "type": "process"
    })
    networkIP_subject_nodes=networkIP_subject_nodes[networkIP_subject_nodes["node_name"]!="unknown"]
    networkIP_object_nodes = pd.DataFrame({
        "node_name": complete_network_edges["object"],
        "ip address(destination)": complete_network_edges["object"],
        "port(destination)": network_edges["destination_port"],
        "network_type": network_edges["network_type"],
        "type": "network"
    })
    networkIP_object_nodes=networkIP_object_nodes[networkIP_object_nodes["node_name"]!="unknown"]
    return complete_network_edges, networkIP_subject_nodes,networkIP_object_nodes
```

圖 8、此程式碼為處理網路相關事件提取點以及邊

3.8 Provenance Graph 生成與視覺化

在本研究中，為了將從 Auditbeat 稽核日誌中提取出的重要事件資訊轉化為直觀且具分析價值的視覺化圖形，我們使用了 Graphistry 套件進行節點與邊的繪製。Graphistry 是一個基於雲端的圖形分析平台，可將結構化資料中的事件關聯以互動式的圖形方式呈現，幫助資安人員追蹤事件之間的因果關係。在視覺化階段中，我們撰寫了 plot_provenance_graph() 函式如圖 9，此函式主要負責將預處理後的節點資料與邊資料傳送給 Graphistry 並生成圖形。在函式中，首先定義了節點與邊類型對應的顏色設定，例如將代表程序行為的 process 設為紅色、代表檔案行為的 file 設為藍色，以及代表網路事件的 network 設為綠色。這些顏色設定分別套用於邊（edge）與節點（node），使不同類型的事件能在圖形中以視覺方式迅速區別。接著，使用 graphistry.bind() 函式綁定事件的來源(source)、目的(destination)、邊標籤（edge_title）、節點 ID（node）等欄位，以確保每一個節點與邊都能正確呈現其屬性與關聯。函式最後透過 .nodes() 與 .edges() 傳入節點與邊資料，並以 .encode_edge_color() 與 .encode_point_color() 進一步將資料對應至顏色分類，

最終使用 .plot() 將整個事件關聯圖上傳並呈現在 Graphistry 的雲端平台上。如此一來，不僅可視化出系統中過往的安全事件流程，還能讓資安分析者互動式地瀏覽與操作整體圖形，快速辨識關鍵節點與可疑行為的來源或傳播路徑。透過這種視覺化方式，也能有效過濾日誌中的冗餘資訊，僅呈現與事件因果關係高度相關的部分，使整體的溯源圖更加清晰且具實用性，大幅提升事件調查與取證的效率。

```
def plot_provenance_graph(all_nodes, edges_for_graphistry):
    edge_color_map = {
        'process': 'red',
        'file': 'blue',
        'network': 'green'
    }
    node_color_map = {
        'process': 'red',
        'file': 'blue',
        'network': 'green'
    }

    graphistry.register(api=3, protocol="https", server="hub.graphistry.com", username="audit", password="")
    plot = graphistry.bind(
        source="subject",
        node="node_name",
        destination="object",
        edge_title="action",
    ).nodes(all_nodes).edges(edges_for_graphistry).encode_edge_color(
            column='edge_category',
            categorical_mapping=edge_color_map
        ).encode_point_color(
            column='type',
            categorical_mapping=node_color_map
        ).plot()
    return plot
```

圖 9、此程式碼為將處理過的關鍵日誌資料透過 graphistry 生成溯源圖

第四章 實驗結果與分析

為了驗證稽核日誌分析與溯源圖可視化方法的有效性，除了建立正常使用者操作的行為，並驗證 Auditbeat 所紀錄事件能否有效轉換為事件相關溯源圖，首先針對系統中三種常見行為進行模擬，包括：檔案操作事件以及程序建立事件。

設計並實作一個基於社交工程的模擬攻擊實驗，模擬攻擊者建立一個假冒的釣魚網站，誘使受害者誤信其為官方登入入口。在該頁面中嵌入誤導性的安全更新提示，進而引導使用者下載包含反向連線功能的 ELF 格式後門程式（shell.elf）。當受害者執行該惡意程式後，攻擊者即可透過 Meterpreter 建立反向連線，取得系統控制權，並進一步進行檔案操作與系統指令執行等活動，並觀察 Auditbeat 日誌記錄與系統事件在溯源圖上的可視化呈現。

設計並實作一個具體的模擬攻擊實驗。此實驗選用 Subrion CMS 4.2.1 中已知的遠端程式碼執行（RCE）漏洞 CVE-2018-19422 [20]作為測試目標，模擬攻擊者透過上傳惡意 .phar 檔案取得控制權，並觀察 Auditbeat 日誌記錄與系統事件在溯源圖上的可視化呈現。

4.1 檔案以及程序事件實驗目的與設計

Auditbea 會記錄各種檔案操作、process 建立以及網路相關的紀錄，此實驗針對這些分類生成溯源圖並且分析日誌紀錄。

4.1.1 檔案操作事件

使用 Linux 指令新增使用者並且加到 sudo group，接著觀察/etc/passwd、/etc/shadow、/etc/group 以及/etc/gshadow 這些檔案發生了那些系統操作。

4.1.2 Auditbeat 設定說明

為了能詳細記錄實驗上的稽核紀錄，在 Auditbeat 中設定監控規則，主要透過修改其設定檔 auditbeat.yml 中的 audit.rules 區段，特別針對/etc/passwd、/etc/shadow、/etc/group 以及/etc/gshadow 此四個檔案設定 audit rule 進行監控如表 3，設定規則參數 -w 代表監控檔案路徑， -p 代表對檔案執行的權限（對應 r = read, w = write, x = execute, a = attribute）， -k 設定關鍵字 tag 方便日誌紀錄過濾查詢。

Audit rule	Description
-w /etc/passwd -p wa -k user_change	此規則監控 /etc/passwd 的寫入與屬性變更行為
-w /etc/passwd -p r -k user_read	此規則監控/etc/passwd 的讀取行為
-w /etc/shadow -p wa -k shadow_change	此規則監控 /etc/shadow 的寫入與屬性變更行為
-w /etc/shadow -p r -k shadow_read	此規則監控/etc/shadow 的讀取行為
-w /etc/group -p wa -k group _change	此規則監控 /etc/group 的寫入與屬性變更行為
-w /etc/group -p r -k group _read	此規則監控/etc/group 的讀取行為
-w /etc/gshadow -p wa -k gshadow_change	此規則監控/etc/gshadow 的寫入與屬性變更行為
-w /etc/gshadow -p r -k gshadow_read	此規則監控/etc/gshadow 的讀取行為

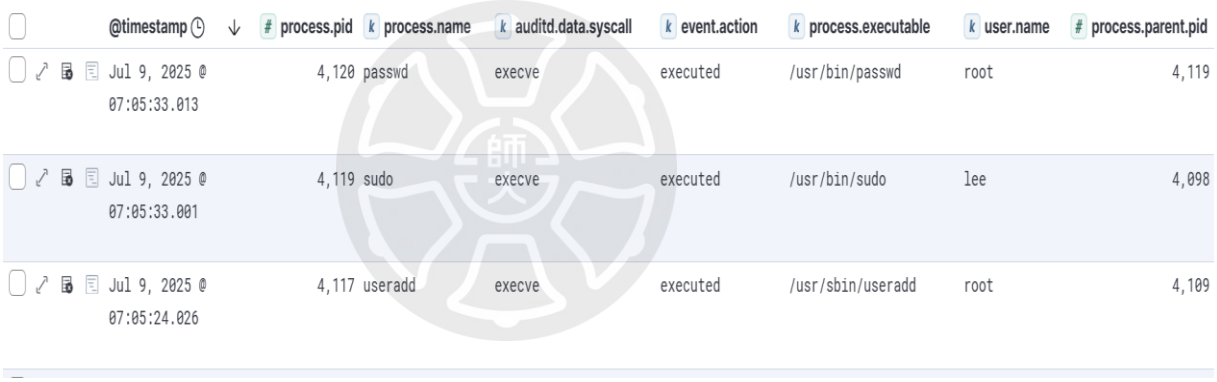
表 3、針對一些檔案路徑新增規則，以利 Auditbeat 進行監控

4.1.3 新增使用者後產生的檔案日誌紀錄以及溯源圖

從篩選重要欄位後的部分日誌紀錄中觀察，process.name 為 useradd 以及 process.pid 為 4109 對/etc/passwd、/etc/shadow、/etc/group 以及/etc/gshadow 這四個檔案的系統操作 openat()以及 rename()如圖 10，此為使用者執行了 useradd 以及 passwd 指令所產生的檔案系統操作日誌紀錄，在圖 11 的日誌紀錄中觀察到建立了各種程序 process.name 為 passwd process.pid 為 4120、process.name 為 sudo process.pid 為 4119、process.name 為 useradd process.pid 為 4117，上述程序建立為使用者執行 useradd 以及 passwd 指令時會詳細記錄程序的相關資訊。接著將產生的相關完整日誌紀錄轉換成溯源圖如圖 12。

@timestamp	process.pid	process.name	auditd.data.syscall	process.executable	file.path
Jul 9, 2025 @ 07:05:24.018	4,109	useradd	openat	/usr/sbin/useradd	/etc/passwd
Jul 9, 2025 @ 07:05:24.002	4,109	useradd	rename	/usr/sbin/useradd	/etc/gshadow+
Jul 9, 2025 @ 07:05:23.998	4,109	useradd	rename	/usr/sbin/useradd	/etc/group+
Jul 9, 2025 @ 07:05:23.990	4,109	useradd	rename	/usr/sbin/useradd	/etc/shadow+
Jul 9, 2025 @ 07:05:23.982	4,109	useradd	rename	/usr/sbin/useradd	/etc/passwd+
Jul 9, 2025 @ 07:05:23.966	4,109	useradd	openat	/usr/sbin/useradd	/etc/passwd
Jul 9, 2025 @ 07:05:23.966	4,109	useradd	openat	/usr/sbin/useradd	/etc/passwd

圖 10、/etc/passwd、/etc/shadow、/etc/group 以及/etc/gshadow 這四個檔案的系統操作日誌紀錄



@timestamp	process.pid	process.name	auditd.data.syscall	event.action	process.executable	user.name	process.parent.pid
Jul 9, 2025 @ 07:05:33.013	4,120	passwd	execve	executed	/usr/bin/passwd	root	4,119
Jul 9, 2025 @ 07:05:33.001	4,119	sudo	execve	executed	/usr/bin/sudo	lee	4,098
Jul 9, 2025 @ 07:05:24.026	4,117	useradd	execve	executed	/usr/sbin/useradd	root	4,109

圖 11、在日誌紀錄中觀察到使用者執行了 passwd 以及 useradd 兩種指令，因此在日誌中紀錄了程序建立的資訊

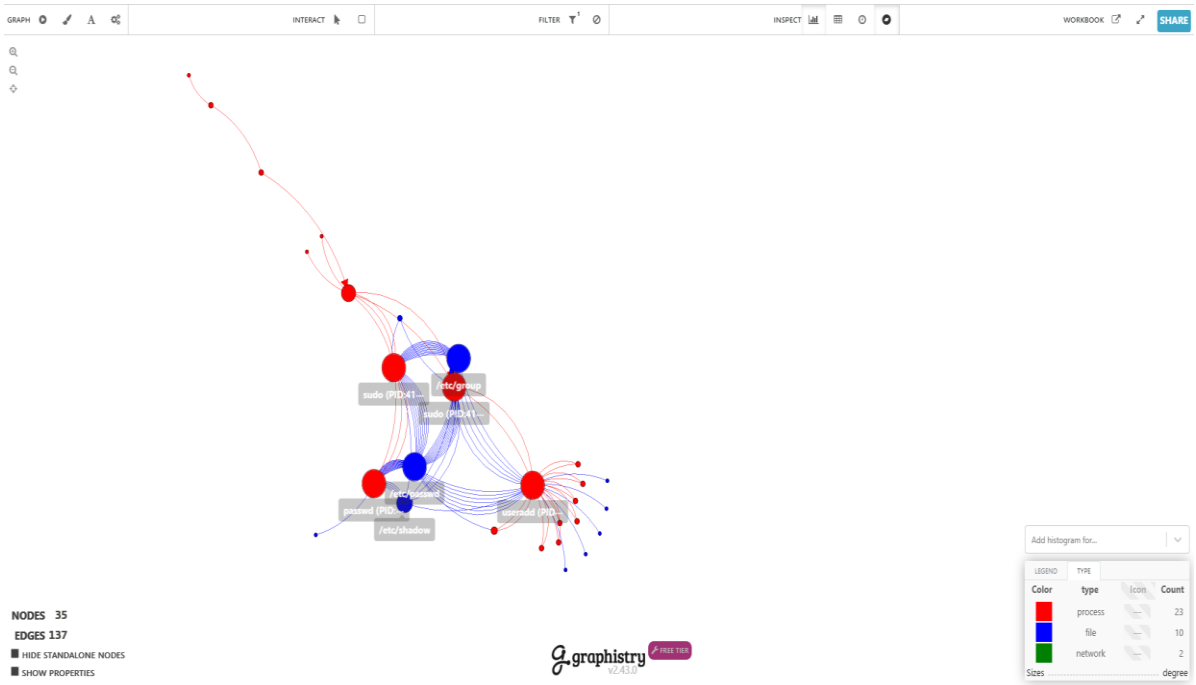


圖 12、將檔案系統操作日誌轉換成溯源圖

4.1.4 檔案操做事件的溯源圖分析

從圖 13 的部分溯源圖可以觀察到 users.name 為 root 執行此系統操作，process.name 為 useradd 且 process.pid 為4109 對檔案路徑為 /etc/group+ 進行了 rename 的檔案系統操作，並且 outcome 顯示為 success，此行為是執行 useradd 指令時所產生的檔案系統操作。

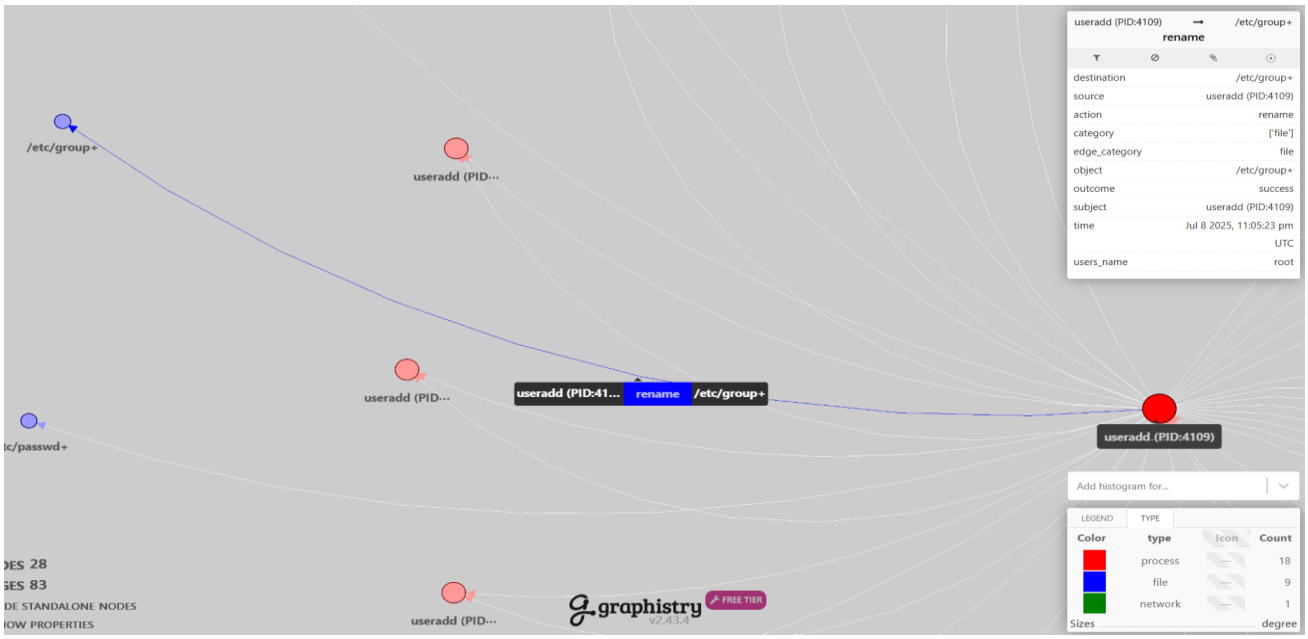


圖 13、觀察一部分 Process 對 file 進行 rename 系統操作

4.1.5 Process create 事件的溯源圖分析

從圖 14 的部分溯源圖可以觀察到，程序 process.name 為 bash 且 process.pid 為 4098 建立了 process.name 為 sudo 且 process.pid 為 4119 ，接著 sudo 程序建立了 process.name 為 passwd 且 process.pid 為 4120 ，可明顯發現此為使用者執行 sudo passwd 使用者密碼設定的指令所產生的溯源圖。

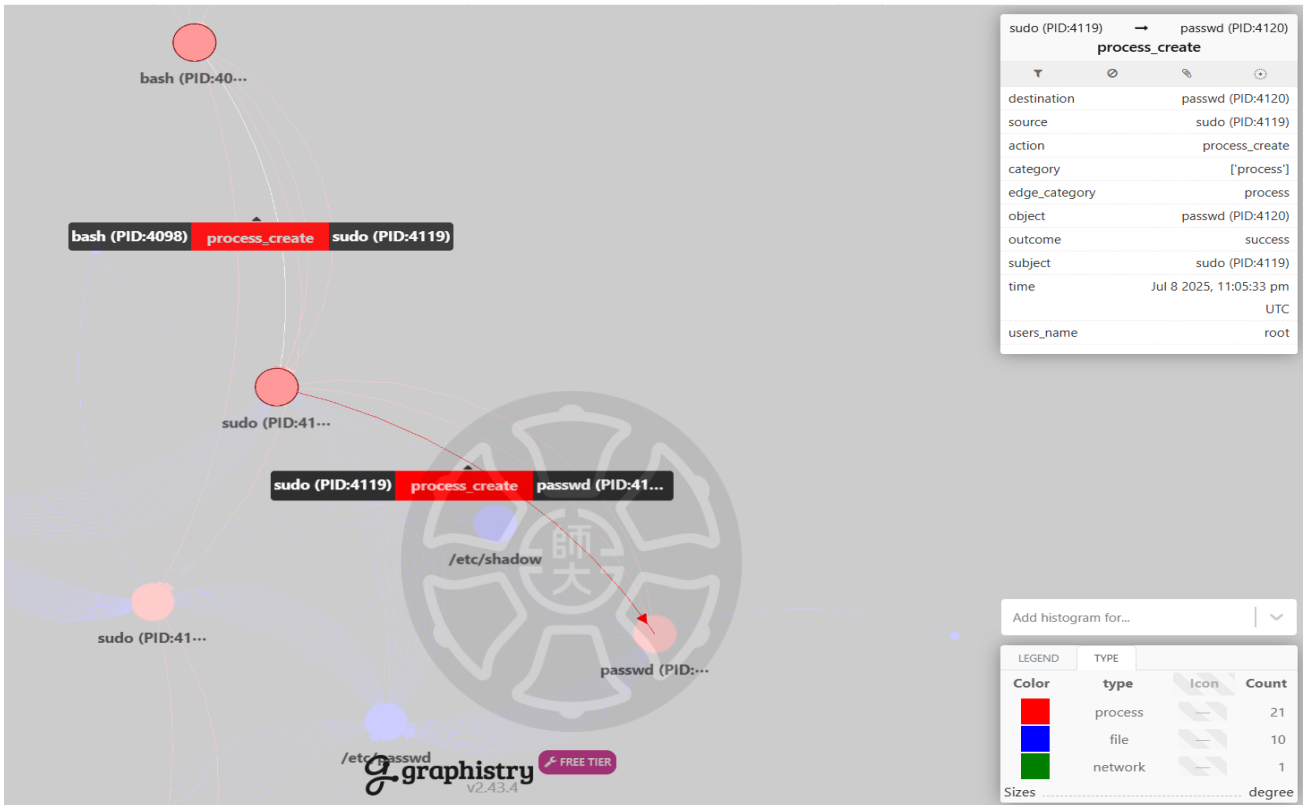


圖 14、Process 建立了 Process 並對檔案進行系統操作溯源圖

4.2 網頁釣魚模擬攻擊實驗目的與設計

模擬後門植入行為的攻擊場景，透過假冒網頁誘導使用者下載後門程式，並進一步引導使用者執行具備反向連線能力的後門程式。攻擊者成功取得系統存取權限後，可進行檔案上傳、下載與內容修改等操作。透過觀察系統於上述過程中所產生的日誌紀錄，並進一步提取事件關聯資訊以建立溯源圖，用以重建攻擊行為鏈與事件因果關係。

4.2.1 模擬場景與環境建置

架構於兩台虛擬機器之間進行，分別模擬攻擊者與受害者主機，受害者主機（Victim）為 Linux Ubuntu 20.04.6 LTS ，其中部署 Elastic Stack 元件， Auditbeat 負責收集系統層級的審計事件、 Elasticsearch 作為日誌儲存與查詢後端以及 Kibana 提供可視化介面以瀏覽並分析事件日誌。攻擊者主機（Attacker）為 Kali Linux 2024.4，其中建置釣魚登入頁面，並部署 Apache2 HTTP Server ，用以誘導使用者下載後門程式。 Kali 攻擊工具包含 nmap 用於掃描靶機開啟 Port 以及 IP 、 msfvenom 產生具備反向連線能力的惡意 ELF 檔案以及 Metasploit Framework 用於接收回連、建立 Meterpreter 連線並執行上傳、下載與修改檔案等操作。

4.2.2 攻擊流程

資訊蒐集階段（Reconnaissance）攻擊者透過 nmap 工具掃描目標網段，確認 Ubuntu 靶機的 IP 位址與開放的 HTTP/SSH 服務。產生惡意 Payload （Payload Generation）在 Kali 攻擊機中使用 msfvenom 產生一個具有反向連線功能的 ELF 格式後門程式（shell.elf），設定反向連線至 Kali 的 LHOST 及指定 LPORT 。建立釣魚網站（Phishing Setup）攻擊者於 Kali Linux 上自行設計並部署一個仿造的 HTML 網站，於頁面中加入安全更新插件或帳戶修復工具的連結，實為指向 shell.elf 的 payload 。社交工程誘導（Social Engineering Inducement）攻擊者誘導受害者點擊偽造網站，使受害者下載後門程式，進一步進行攻擊。目標執行後門程式（Backdoor Execution）當受害者於 Ubuntu 靶機上執行 shell.elf 後，後門程式立即與 Kali 的 msfconsole 建立 Meterpreter 連線，攻擊者即取得該機的遠端控

制權。後滲透控制（Post-Exploitation）攻擊者透過 Meterpreter session 進行檔案下載、上傳以及修改。

4.2.3 Auditbeat 設定說明

為了能詳細記錄模擬攻擊實驗上的稽核紀錄，在 Auditbeat 中設定監控規則，主要透過修改其設定檔 auditbeat.yml 中的 audit.rules 區段，針對實驗設置了以下規則如表 4，設定規則參數 -w 代表監控檔案路徑，-p 代表對檔案執行的權限（對應 r = read, w = write, x = execute, a = attribute），-k 設定關鍵字 tag 方便過濾查詢，-a 代表新增一條系統呼叫監控規則，always 表示永遠紀錄符合條件的事件，exit 表示當指定的系統呼叫（如 openat）執行完成、返回結果時，auditd 記錄這次呼叫的詳細資訊，-S 指定要監控的 syscall，-F 為條件篩選器。

Audit rule	Description
-w /etc/passwd -p wa -k user_change	此規則監控 /etc/passwd 的寫入與屬性變更行為
-w /etc/passwd -p r -k user_read	此規則監控/etc/passwd 的讀取行為
-w /etc/shadow -p wa -k shadow_change	此規則監控 /etc/shadow 的寫入與屬性變更行為
-w /etc/shadow -p r -k shadow_read	此規則監控/etc/shadow 的讀取行為
-w /etc/group -p wa -k group _change	此規則監控 /etc/group 的寫入與屬性變更行為
-w /etc/group -p r -k group _read	此規則監控/etc/group 的讀取行為
-w /etc/gshadow -p wa -k gshadow_change	此規則監控/etc/gshadow 的寫入與屬性變更行為
-w /etc/gshadow -p r -k gshadow_read	此規則監控/etc/gshadow 的讀取行為
-w /home/lee -p rwax	用來全面監控 /home/lee 目錄下的所有活動，包括讀取、寫入、執行以及屬性變更等操作
-a always,exit -F arch=b64 -S accept,bind,connect -F key=external-access	此規則針對 64 位元架構，監控所有與網路連線建立有關的系統呼叫，包括 accept、bind 和 connect
-a always,exit -S sendto -F key=network_send	此規則會監控系統呼叫 sendto，即透過網路傳送資料的行為
-a always,exit -S recvfrom -F key=network_receive	此規則監控 recvfrom 系統呼叫，代表接收網路資料的行為

表 4、針對一些檔案路徑以及系統呼叫新增規則，以利 Auditbeat 進行監控

4.2.4 攻擊行為的日誌記錄觀察

透過 Elasticsearch 與 Kibana ，成功觀察到的日誌紀錄，攻擊時間點（約 04:00）事件數量顯著增加(圖 15)。圖 16 為模擬實驗所產生的程序建立系統操作日誌紀錄，從記錄中觀察到建立了各種程序 process.name 為 gnome-terminal-

process.pid 為 6212、process.name 為 bash process.pid 為 6222、process.name 為 shell.elf process.pid 為 6253以及 process.name 為 sh process.pid 為 6429，圖 17 為模擬實驗所產生的檔案系統操作日誌紀錄，從日誌中觀察到 process.name 為 shell.elf 且 process.pid 為 6253 對檔案 passwd.txt 及 attacker.txt 執行了 open 系統呼叫相關的操作，上述行為是針對靶機的 passwd.txt 進行修改以及將攻擊機的 attacker.txt 上傳到靶機，圖 18 為模擬實驗所產生的網路相關系統操作日誌紀錄，程序 process.name 為 shell.elf process.pid 為 6253 透過 22 port 建立對外網路連線至攻擊機 destination.ip 為 10.0.2.6，此行為是 Kali 的 msfconsole 建立 Meterpreter 連線時所產生的日誌紀錄，程序 process.name 為 firefox process.pid 為 5809 與對外網路 10.0.2.6 連線，此行為是靶機到假冒網頁所產生的網路日誌紀錄。

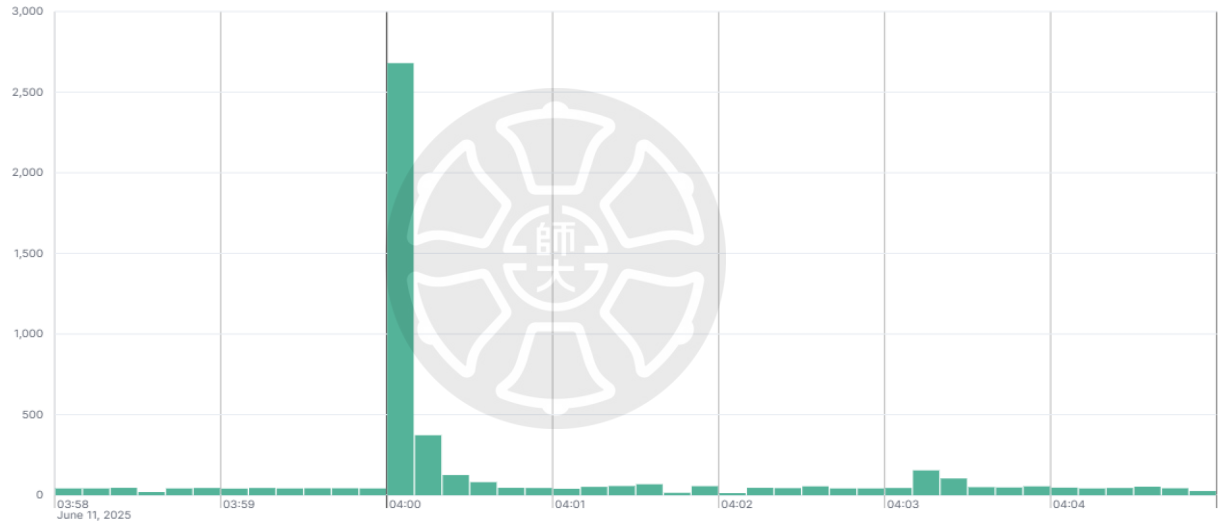


圖 15、在後門執行成功後 Kali 的 msfconsole 建立 Meterpreter 連線，可發現在4:00時日誌紀錄數量有明顯的上升

@timestamp	process.pid	process.name	auditd.data.syscall	process.executable	process.parent.pid
Jun 11, 2025 @ 04:02:14.569	6,429	sh	execve	/usr/bin/dash	6,253
Jun 11, 2025 @ 04:00:28.324	6,253	shell.elf	execve	/home/lee/下載/shell.elf	6,222
Jun 11, 2025 @ 04:00:12.440	6,222	bash	execve	/usr/bin/bash	6,212
Jun 11, 2025 @ 04:00:12.240	6,212	gnome-terminal-	execve	/usr/libexec/gnome-terminal-server	2,106

圖 16、在日誌紀錄中觀察到後門程式的執行以及攻擊者透過後門再建立一個 Shell

@timestamp	process.pid	process.name	file.path	process.parent.pid	auditd.data.syscall
Jun 11, 2025 @ 04:01:58.303	6,253	shell.elf	passwd.txt	6,222	open
Jun 11, 2025 @ 04:01:46.560	6,253	shell.elf	passwd.txt	6,222	open
Jun 11, 2025 @ 04:01:38.526	6,253	shell.elf	attacker.txt	6,222	open
Jun 11, 2025 @ 04:01:20.524	6,253	shell.elf	passwd.txt	6,222	open
Jun 11, 2025 @ 04:01:12.122	6,253	shell.elf	passwd.txt	6,222	open
Jun 11, 2025 @ 04:00:28.524	6,253	shell.elf	/etc/passwd	6,222	open
Jun 11, 2025 @ 04:00:28.436	6,253	shell.elf	/etc/passwd	6,222	open

圖 17、在日誌紀錄中觀察到檔案的系統操作

@timestamp	process.pid	process.name	process.parent.pid	destination.ip	destination.port	client.ip	network.type	network.transport	process.executable	auditd.data.syscall
Jun 11, 2025 @ 04:02:38.873	6,253	shell.elf	-	10.0.2.6	22	10.0.2.5	ipv4	tcp	./shell.elf	-
Jun 11, 2025 @ 04:00:28.324	6,253	shell.elf	6,222	10.0.2.6	22	-	-	-	/home/lee/下載/shell.elf	connect
Jun 11, 2025 @ 04:00:16.870	5,809	firefox	-	10.0.2.6	80	10.0.2.5	ipv4	tcp	/usr/lib/firefox/firefox	-
Jun 11, 2025 @ 04:00:12.870	5,809	firefox	-	10.0.2.6	80	10.0.2.5	ipv4	tcp	/usr/lib/firefox/firefox	-
Jun 11, 2025 @ 04:00:08.410	5,809	536F636B657420546872656164	2,106	10.0.2.6	80	-	-	-	/usr/lib/firefox/firefox	connect
Jun 11, 2025 @ 04:00:06.662	5,809	536F636B657420546872656164	2,106	10.0.2.6	80	-	-	-	/usr/lib/firefox/firefox	connect

圖 18、在日誌紀錄中觀察到網路相關的系統操作

4.2.5 溯源圖生成與分析

模擬實驗後所生成的日誌紀錄經過本研究所實作之 Python 模組處理後，轉換為可視化的溯源圖，從圖 19 中的部分溯源圖觀察到 Linux 圖形化背景程式 gnome-terminal- 建立了程序 process.name 為 bash 且 process.pid 為 6222，接著 bash 程序建立了 process.name 為 sell.elf 且 process.pid 為 6253，從上述的溯源發現此為靶機執行了 Bash 並且執行了後門程式，後門程式遭到執行後從圖 20 觀察到後門程式 process.name 為 sell.elf 且 process.pid 為 6253 對 passwd.txt 執行了 open

檔案系統呼叫，從這一連串的關係可以觀察到攻擊機對靶機上的敏感檔案進行了開啟的系統操作，從圖 21 中的部分溯源圖觀察到 process.name 為 sell.elf 且 process.pid 為 6253 的後門程式使用 22 port 對外部來源(10.0.2.6)建立了網路連線。從上述溯源圖中可觀察出一條清晰的攻擊路徑，顯示靶機遭到後門程式入侵並且敏感檔案遭讀取。

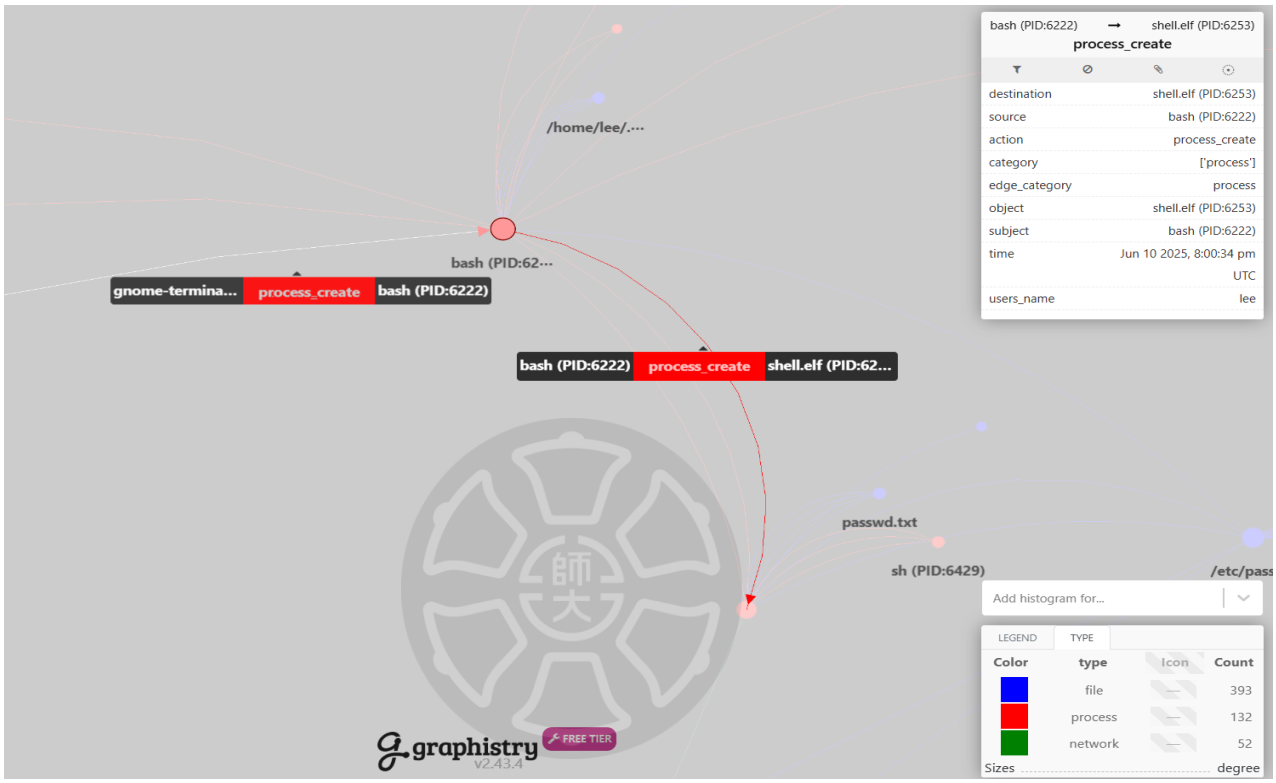


圖 19、在溯源圖中可以發現 Linux 圖形化背景程式、Bash 以及 shell.elf 之間的關係

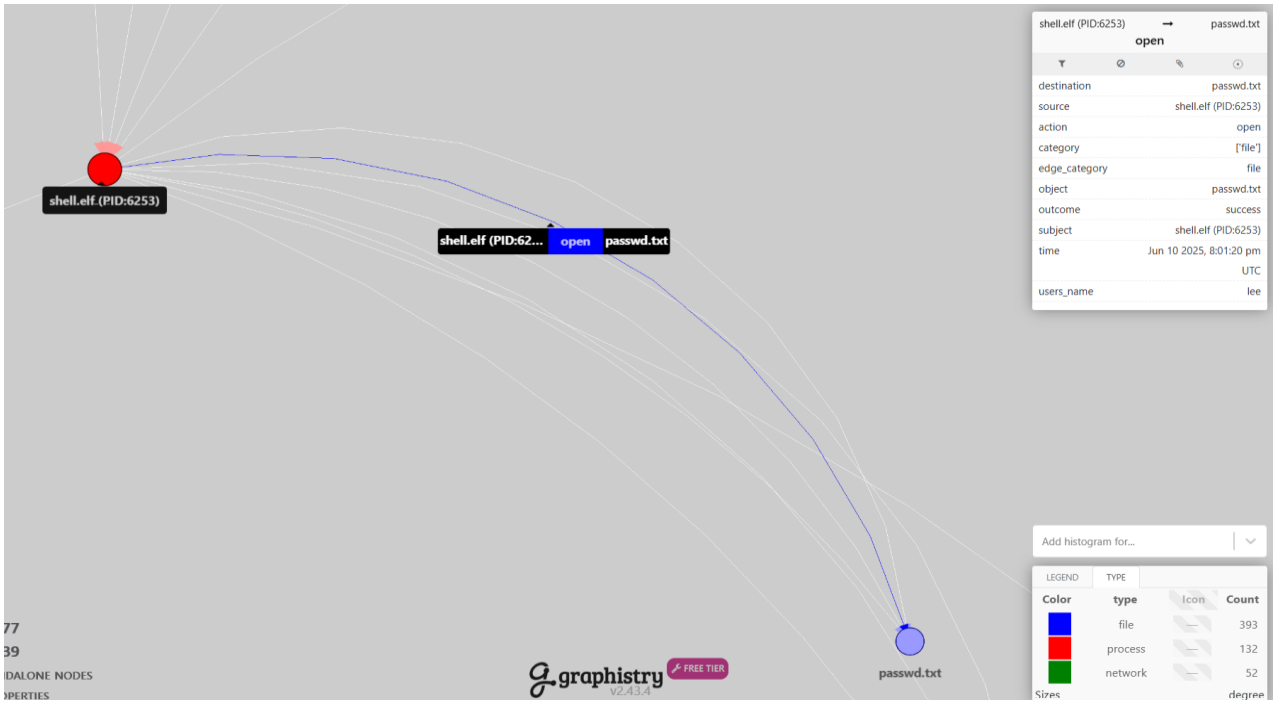


圖 20、在溯源圖中發現 shell.elf 程式對 passwd.txt 進行了 open 檔案操作

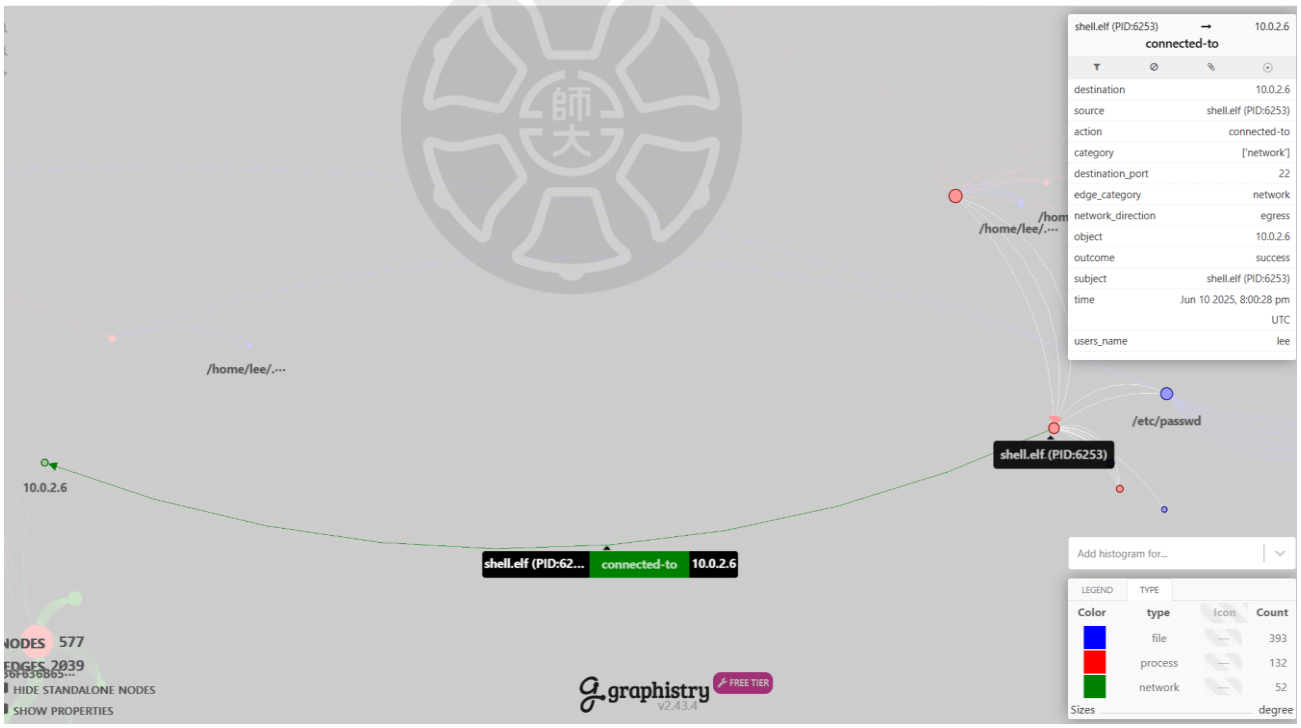


圖 21、在溯源圖中發現 shell.elf 程式對攻擊機 IP:10.0.2.6連線

4.3 CVE-2018-19422 模擬攻擊實驗目的與設計

本實驗旨在模擬實際的攻擊情境，觀察 Auditbeat 能否正確紀錄攻擊活動所產生的系統事件，並驗證本系統是否能將其轉換為清晰的溯源圖，以利後續的攻擊行為溯源分析。

4.3.1 攻擊流程

攻擊者使用 Nmap 掃描區網以確認靶機與攻擊機處於同一子網，靶機 IP 為 10.0.2.5，攻擊機 IP 為 10.0.2.6。在攻擊機建立 .phar 檔案，設置其目標 IP 與 port，並將其上傳至 Subrion CMS 的 /panel/uploads 目錄。攻擊機在瀏覽器中存取該 .phar 檔案以觸發漏洞。成功觸發 Reverse Shell，攻擊者取得靶機的 shell 權限。

4.3.2 攻擊場景與環境設置

本實驗架構於兩台虛擬機器之間進行，分別模擬攻擊者與受害者主機，受害者主機（Victim）為 Linux Ubuntu 20.04.6 LTS，其中部署 Elastic Stack 元件，Auditbeat 負責收集系統層級的審計事件、Elasticsearch 作為日誌儲存與查詢後端以及 Kibana 提供可視化介面以瀏覽並分析事件日誌。並且事先架設此已知漏洞的內容管理系統 Subrion CMS 4.2.1 以及 Apache2 HTTP Server 。攻擊機（Attacker）Kali Linux 2024.4，使用 nmap 掃描並發動攻擊。

4.3.3 Auditbeat 設定說明

為了能詳細記錄模擬攻擊實驗上的稽核紀錄，在 Auditbeat 中設定監控規則，主要透過修改其設定檔 auditbeat.yml 中的 audit.rules 區段，針對實驗設置了以下規則如表 5，設定規則參數 -w 代表監控檔案路徑，-k 設定關鍵字 tag 方便過濾查詢，-a 代表新增一條系統呼叫監控規則，always 表示永遠紀錄符合條件的事件，exit 表示當指定的系統呼叫（如 openat）執行完成、返回結果時，auditd 記錄這次呼叫的詳細資訊， -S 指定要監控的 syscall， -F 為條件篩選器。

Audit rule	Description
-w /var/www/subrion/uploads/ -k subrions	用來監控 Subrion CMS 網站上傳目錄 的所有存取行為
-a always,exit -F arch=b64 -S accept,bind,connect -F key=external-access	此規則針對 64 位元架構，監控所有與網路連線建立有關的系統呼叫，包括 accept、bind 和 connect
-a always,exit -S sendto -F key=network_send	此規則會監控系統呼叫 sendto，即透過網路傳送資料的行為
-a always,exit -S recvfrom -F key=network_receive	此規則監控 recvfrom 系統呼叫，代表接收網路資料的行為

表 5、針對一些檔案路徑以及系統呼叫新增規則，以利 Auditbeat 進行監控

4.3.4 攻擊行為的日誌記錄觀察

透過 Elasticsearch 與 Kibana，成功觀察到的日誌紀錄，圖 22 為攻擊機監聽 .phar 檔當時所設定的 80 port，接著執行.phar 檔並且成功 reverse shell 的 shell 畫面，從上面可以看到成功 reverse shell 的時間點（約 05:57），接著觀察圖 23 為 Kibana 所統計各個時間點的日誌數量，可發現 05:57 事件數量顯著增加。圖 24 為模擬實驗所產生的檔案系統操作日誌紀錄，從日誌中觀察到 process.name 為 apache2 且 process.pid 為 3012 對檔案路徑為 /var/www/subrion/uploads/php-reverseshell.phar 執行了多筆與 chmod、chown、rename、openat 等系統呼叫相關的操作，此行為是將.phar 檔上傳至靶機後所產生的系統操作日誌紀錄，圖 25 為模擬實驗所產生的程序建立系統操作日誌紀錄，從記錄中觀察到 user.name 為 www-data 使用者同時建立了各種程序 process.name 為 bash process.pid 為3334、process.name 為 sh process.pid 為3330、process.name 為 cat process.pid 為3341 以及 process.name 為 apache2 process.pid 為 3326 ，以上所建立的程序為攻擊機在成功 reverse shell 後產生 shell 程序，並且執行了 cat 的指令所產生的日誌紀錄，圖 26 為模擬實驗所產生的網路相關系統操作日誌紀錄，從日誌中可發現 user.name 為 www-data 使用者在程序 process.name 為 apache2 process.pid 為 1170 透過 80 port 建立對外網路連線至攻擊機 destination.ip 為 10.0.2.6，此行為

是攻擊機成功 reverse shell 至靶機所產生的網路日誌紀錄。

圖 22、在攻擊機中監聽 80 port，接著執行.phar 檔成功 reverse shell 的 shell 畫面

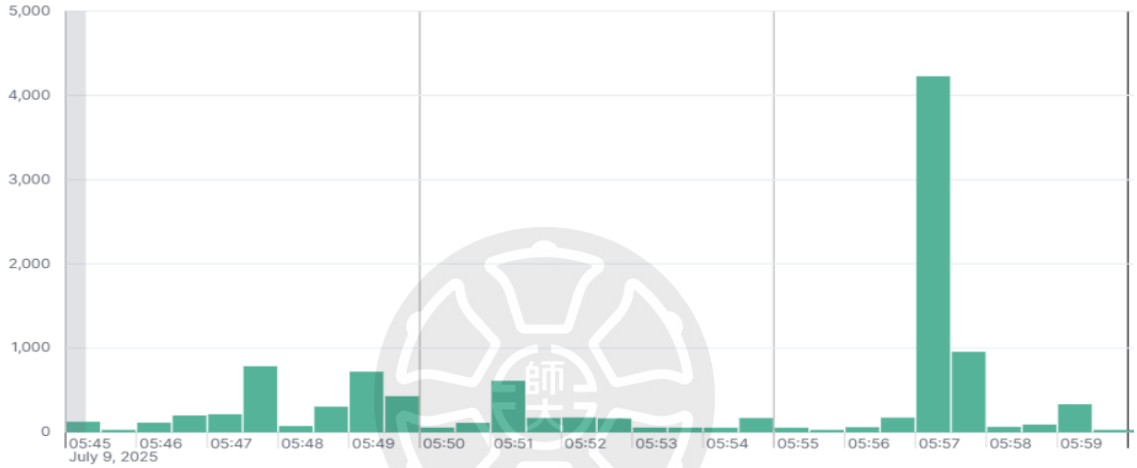


圖 23、在攻擊機上成功 reverse shell 後，並且在 Kibana 中顯示攻擊時段稽核事件異常增加情形，可以發現在 reverse shell 成功的時候產生大量的日誌紀錄

@timestamp	process.pid	process.name	process.executable	process.parent.pid	auditd.data.syscal	file.path
Jul 9, 2025 @ 05:57:32.093	3,012	apache2	/usr/sbin/apache2	997	chmod	/var/www/subrion/uploads/php-reverseshell.phar
Jul 9, 2025 @ 05:57:32.093	3,012	apache2	/usr/sbin/apache2	997	openat	/var/www/subrion/uploads/php-reverseshell.phar
Jul 9, 2025 @ 05:57:32.093	3,012	apache2	/usr/sbin/apache2	997	chmod	/var/www/subrion/uploads/php-reverseshell.phar
Jul 9, 2025 @ 05:57:32.093	3,012	apache2	/usr/sbin/apache2	997	chown	/var/www/subrion/uploads/php-reverseshell.phar
Jul 9, 2025 @ 05:57:32.093	3,012	apache2	/usr/sbin/apache2	997	rename	/var/www/subrion/uploads/
Jul 9, 2025 @ 05:57:32.093	3,012	apache2	/usr/sbin/apache2	997	openat	/var/www/subrion/uploads/php-reverseshell.phar
Jul 9, 2025 @ 05:57:32.021	3,012	apache2	/usr/sbin/apache2	997	openat	/var/www/subrion/uploads/.htaccess
Jul 9, 2025 @ 05:57:32.021	3,012	apache2	/usr/sbin/apache2	997	openat	/var/www/subrion/uploads

圖 24、模擬攻擊者上傳惡意 .phar 檔案取得系統控制權，並且從 Kibana 中觀察到檔案操作事件之稽核日誌紀錄

@timestamp	process.pid	process.name	process.executable	process.parent.pid	user.name	event.action
Jul 9, 2025 @ 05:58:52.288	3,341	cat	/usr/bin/cat	3,334	www-data	executed
Jul 9, 2025 @ 05:57:47.647	3,330	sh	/usr/bin/dash	1,170	www-data	process_started
Jul 9, 2025 @ 05:57:47.647	3,334	bash	/usr/bin/bash	3,330	www-data	process_started
Jul 9, 2025 @ 05:57:46.476	3,334	bash	/usr/bin/bash	3,330	www-data	executed
Jul 9, 2025 @ 05:57:46.080	3,330	sh	/usr/bin/dash	1,170	www-data	executed
Jul 9, 2025 @ 05:57:37.650	3,326	apache2	/usr/sbin/apache2	997	www-data	process_started

圖 25、攻擊機透過監聽特定 port 成功 reverse shell，並且從 Kibana 中觀察到 Process 啟動事件之稽核日誌紀錄

@timestamp	process.pid	process.name	process.executable	event.action	network.type	network.transport	destination.ip	destination.port	client.ip	user.name
Jul 9, 2025 @ 05:59:09.117	1,170	apache2	/usr/sbin/apache2	network_flow	ipv4	tcp	10.0.2.6	80	10.0.2.5	www-data
Jul 9, 2025 @ 05:58:17.121	1,170	apache2	/usr/sbin/apache2	network_flow	ipv4	tcp	10.0.2.6	80	10.0.2.5	www-data

圖 26、在攻擊機上成功 reverse shell 後，並且從 Kibana 中觀察到攻擊機 IP 以及當時所監聽的 port

4.3.5 溯源圖生成與分析

模擬實驗後所生成的日誌紀錄經過本研究所實作之 Python 模組處理後，轉換為可視化的溯源圖如圖 27。apache2 程序與特定檔案之間有各種系統呼叫相關的操作，從圖 28 中的部分溯源圖觀察到 process.name 為 apache2 且 process.pid 為 3012 對檔案路徑為 /var/www/subrion/uploads/php-reverseshell.phar 執行了 chmod 的檔案系統操作，從圖 29 中的部分溯源圖觀察到 process.name 為 apache2 且 process.pid 為 3012 對檔案路徑為 /var/www/subrion/uploads/php-reverseshell.phar 執行了 openat 的檔案系統操作，由此兩個檔案系統操作可發現 apache2 程序對異常檔案進行了多種檔案系統操作，從圖 30 中的部分溯源圖觀察到 process.name 為 apache2 且 process.pid 為 1170，users.name 為 www-data 使用 80 port 與外部來源

（10.0.2.6）建立了網路連線，從圖 31中的部分溯源圖觀察到 process.name 為 apache2 且 process.pid 為1170 建立了一個 process.name 為 sh 且 process.pid 為 3330，接著 sh 程序又建立了一個 process.name 為 bash 且 process.pid 為3334，整個過程觀察到 users.name 為 www-data 建立了異常的 bash shell 的程序，可觀察到過程中涉及之檔案、程序、網路節點，以及其操作類型與因果關聯，從圖 32 中的部分溯源圖觀察到 process.name 為 bash 且 process.pid 為3334 建立了 process.name 為 cat 且 process.pid 為3341，接著 cat 程序對 passwd 檔案執行了 openat 的檔案操作。從溯源圖中可清楚辨識此為不正常的攻擊行為，可發現有未知檔案.phar 上傳並且被執行，此檔案執行了各種不同的系統呼叫，更關鍵的是 Process : apache2 create Bash shell，以正常來說此 Process 並不會建立相關的 shell，這些流程有助於事後的安全事件鑑識與判讀。

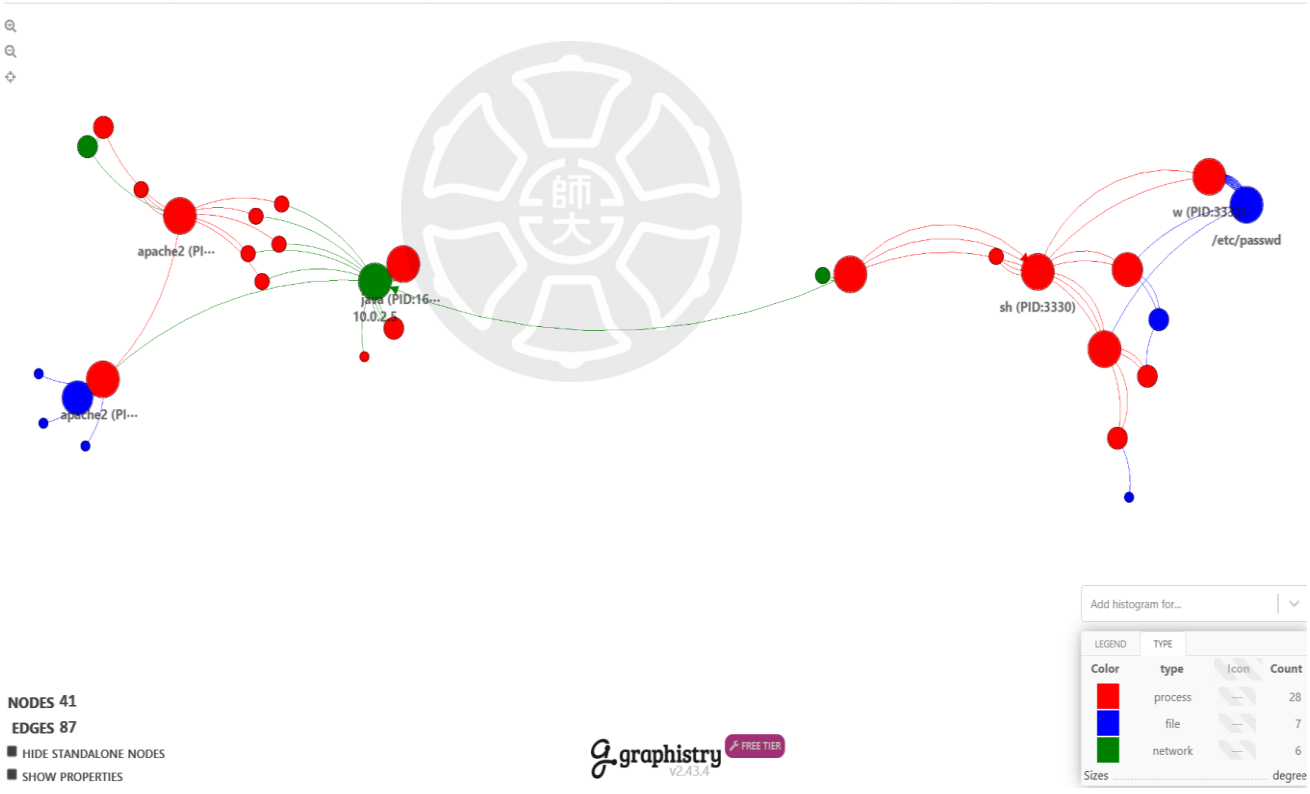


圖 27、日誌轉換成溯源圖

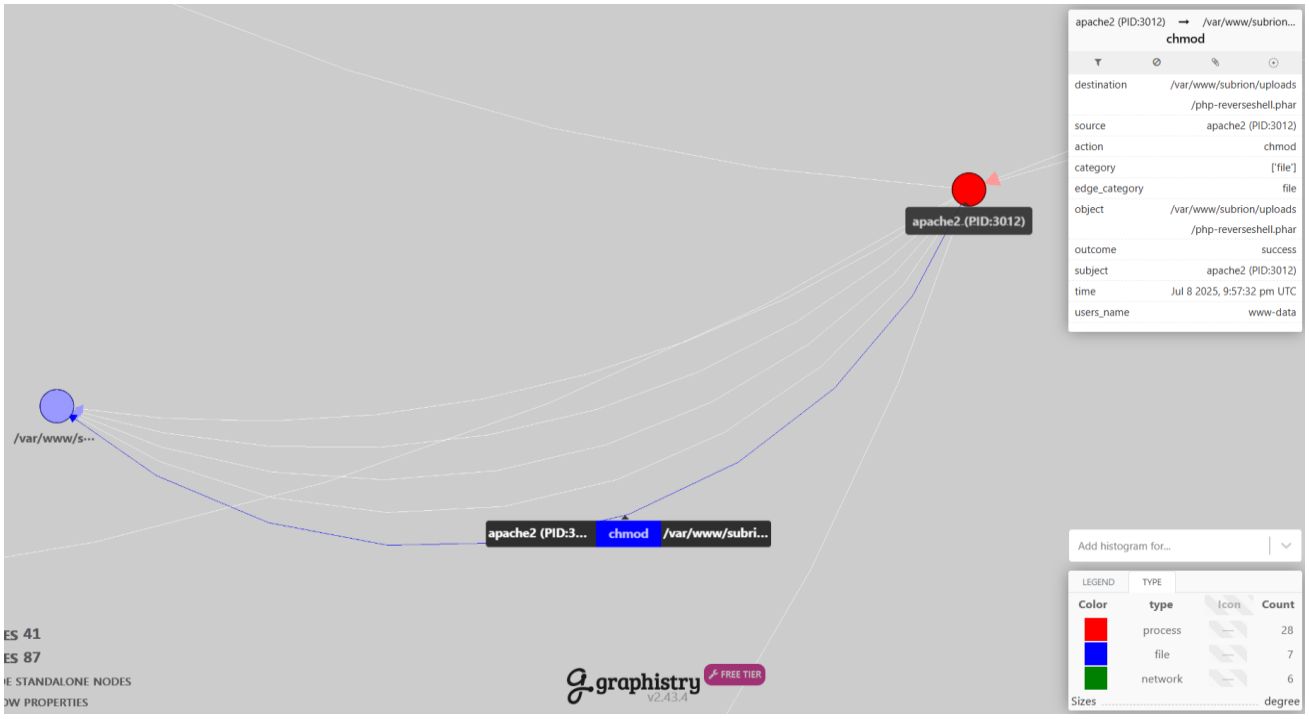


圖 28、檔案操作事件之溯源圖，圖中可觀察到 source process:apache2 對檔案進行了 chmod 系統呼叫的操作

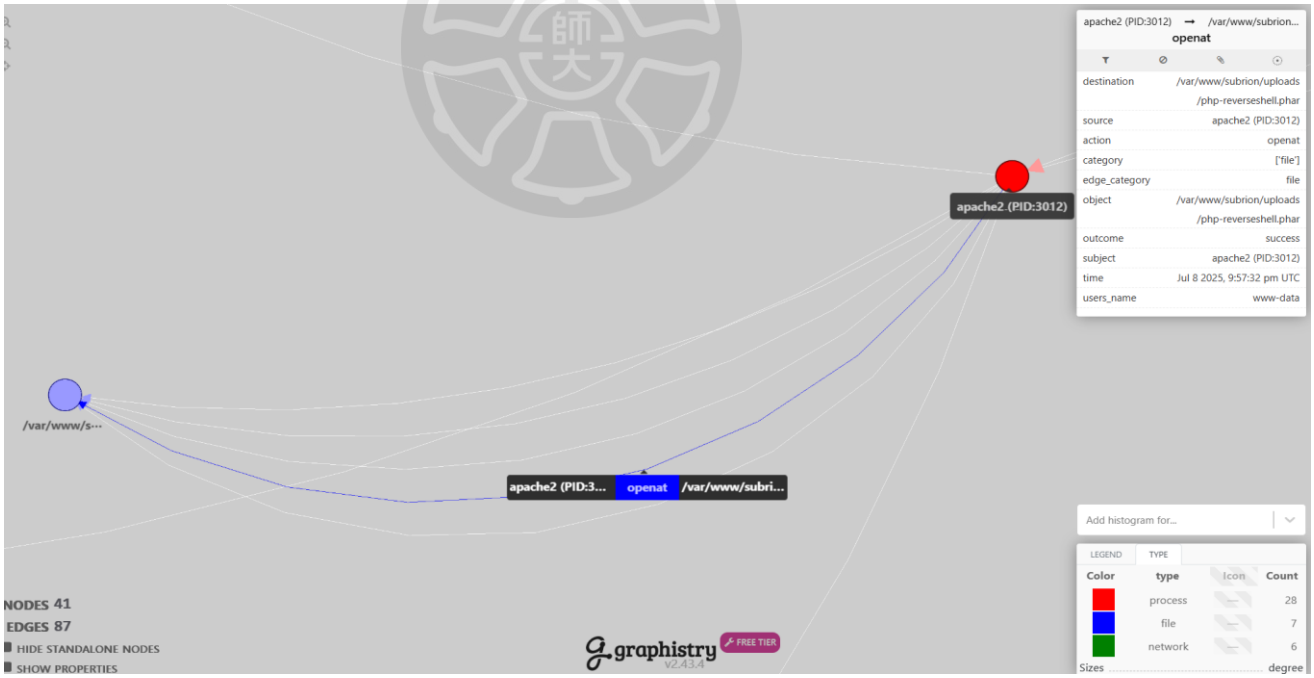


圖 29、檔案操作事件之溯源圖，圖中可觀察到 source process:apache2 對檔案進行了 openat 系統呼叫的操作

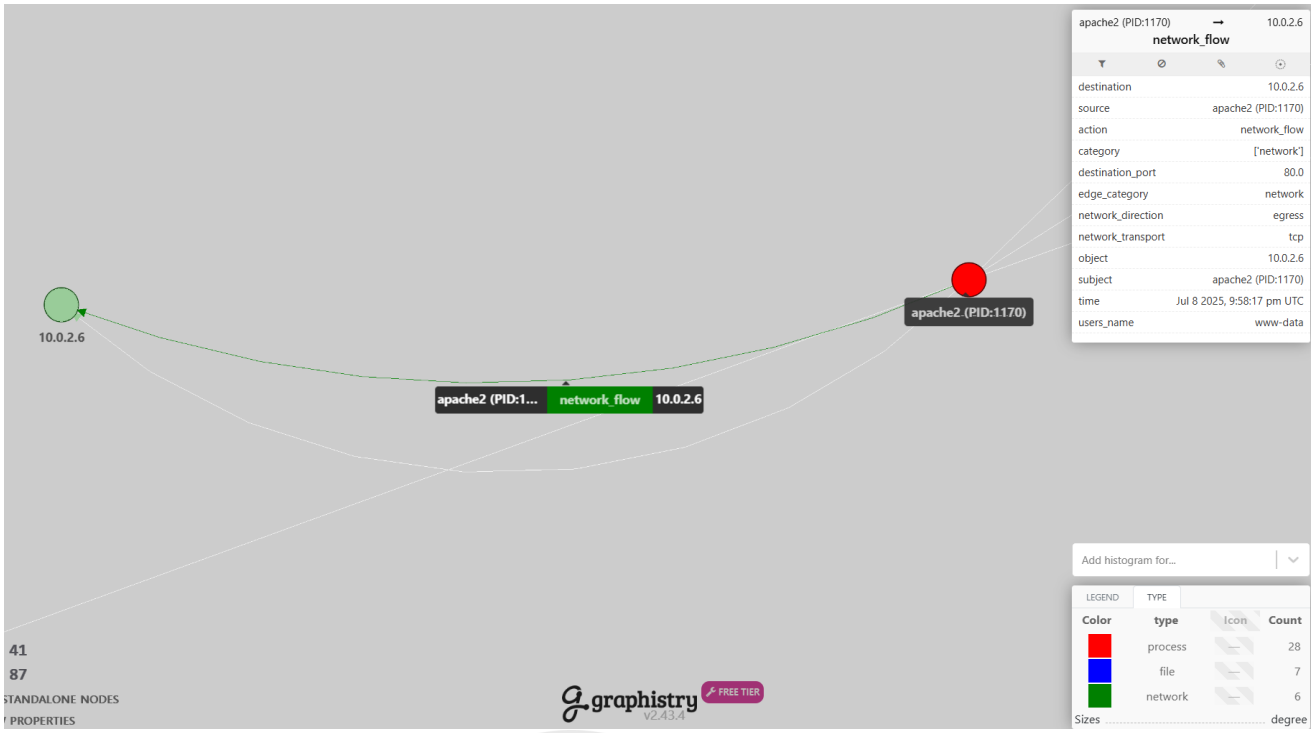


圖 30、對外網路連線事件之溯源圖，圖中可觀察到 network_flow 中 source process:apache2與攻擊機 ip:10.0.2.6有對外連線

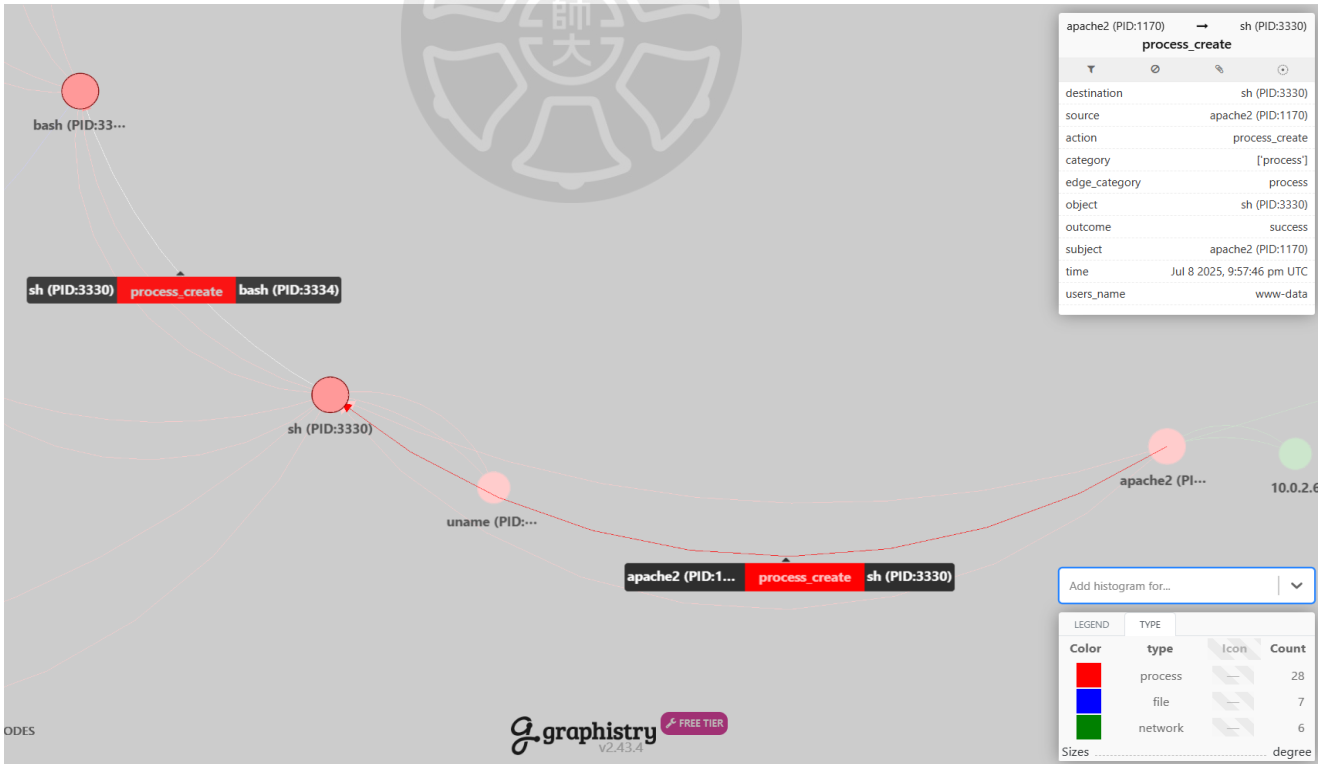


圖 31、Process create 之溯源圖，圖中可發現 source process:apache create Bash shell 的 process

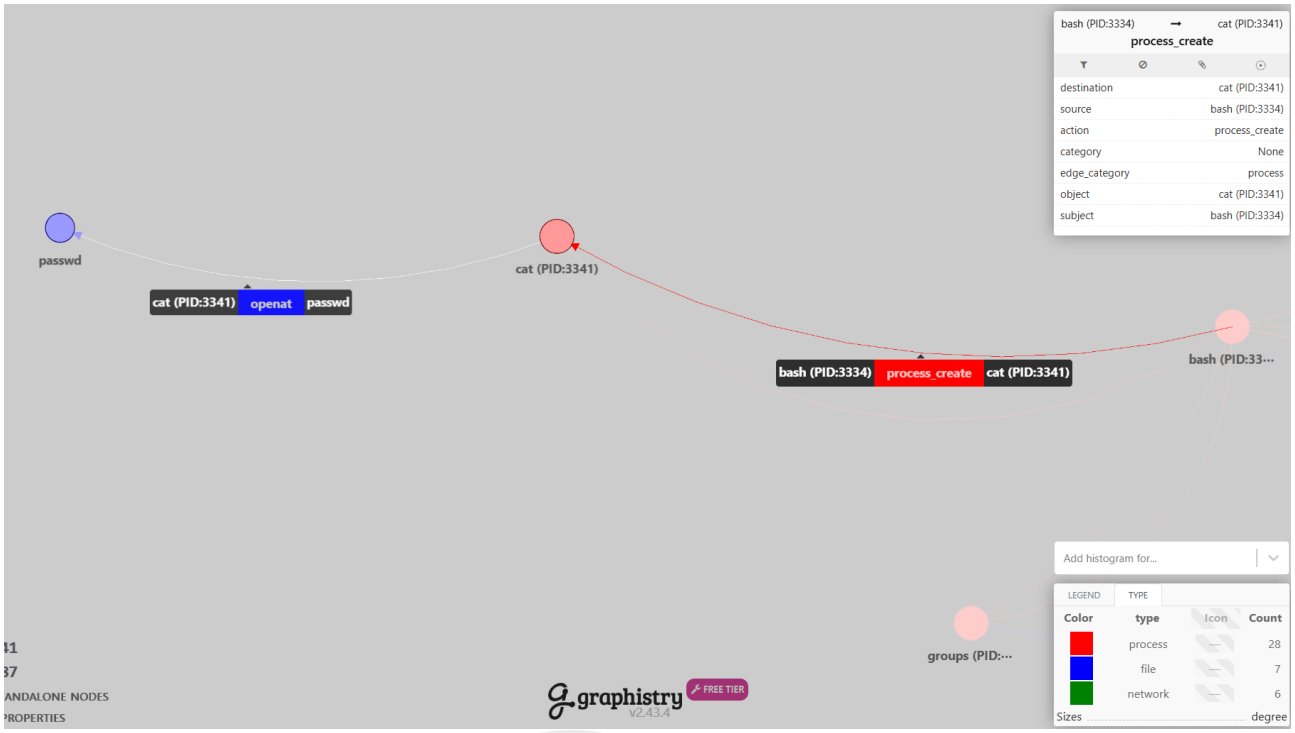


圖 32、從圖中可觀察到 Bash shell 的 process 對 passwd 檔案執行了 cat 的指令

第五章 結論

本研究旨在針對 Linux 系統的稽核日誌進行分析與視覺化處理，藉由部署 Auditbeat 工具，蒐集系統層級的安全事件資料，並整合 Elasticsearch 作為日誌儲存與查詢平台。透過 Python 進行資料擷取與欄位轉換後，進一步建立事件間的因果關係，並以 Graphistry 繪製溯源圖，使稽核資料的內容得以圖形化呈現。實作結果顯示，本研究所提出之流程能有效將繁雜的系統稽核日誌轉換為結構化的節點與邊資料，清楚描繪程序行為、檔案存取與網路連線等系統活動之間的關聯。溯源圖的視覺化效果亦有助於資安人員快速掌握異常活動脈絡與潛在攻擊路徑，提升事件溯源與威脅鑑識的效率。

然而，目前仍存在部分限制，例如系統產生的稽核日誌量龐大，導致生成的溯源圖包含過多節點與邊，容易產生資訊過載或視覺複雜問題，降低分析效率。未來可進一步設計圖簡化策略，過濾重複或無意義邊，並優先顯示關鍵節點與高風險路徑，以提升圖形可讀性與警示效果。此外，亦可考慮導入自然語言處理技術，自動解析事件描述語意，或結合機器學習方法進行異常行為辨識與事件分類，以增強系統在真實場景下的實用性與自動化能力。

參考文獻

[1] Mandiant, "APT1: Exposing one of China's cyber espionage units," 2013. [Online]. Available: https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf.

[2] MITRE, "APT29 Group," [Online]. Available: https://attack.mitre.org/groups/G0016/.

[3] SUSE, "auditd - Auditing Components in SLES 15 SP6," [Online]. Available: https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-audit-comp.html#sec-audit-auditd.

[4] W. U. Hassan, A. Bates and D. Marino, "Tactical Provenance Analysis for Endpoint Detection and Response Systems," 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2020, pp. 1172-1189, doi: 10.1109/SP40000.2020.00096.

[5] M.-H. Wu, F.-H. Hsu, J.-H. Huang, K. Wang, Y.-L. Hwang, H.-J. Wang, J.-X. Chen, T.-C. Hsiao, and H.-T. Yang, "Enhancing Linux system security: A kernel-based approach to fileless malware detection and mitigation," Electronics, vol. 13, no. 17, p. 3569, 2024, doi:10.3390/electronics13173569.

[6] C. Jelesnianski, M. Ismail, Y. Jang, D. Williams, and C. Min, "Protect the system call, protect (most of) the world with BASTION," in Proc. 28th ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS), vol. 3, New York, NY, USA, 2023, pp. 528–541, doi: 10.1145/3582016.3582066.

[7] ATT&CK, MITRE, "Impair defenses: Disable or modify Linux audit system (T1562.012)," MITRE Corporation, 2023. [Online]. Available: https://attack.mitre.org/techniques/T1562/012.

[8] "Elastic," [Online]. Available: https://www.elastic.co/elastic-stack.

[9] "Auditbeat: Lightweight shipper for audit data," [Online]. Available: https://www.elastic.co/beats/auditbeat.

[10] X. Zhang, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in Proc.

2016 ACM SIGSAC Conf. Computer and Communications Security (CCS), New York, NY, USA, Oct. 2016, pp. 504–516, doi: 10.1145/2976749.2978378. X. Zhang, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in Proc. 2016 ACM SIGSAC Conf. Computer and Communications Security (CCS), New York, NY, USA, Oct. 2016, pp. 504–516, doi: 10.1145/2976749.2978378.

[11] A. Gehani and D. Tariq, "SPADE: Support for provenance auditing in distributed environments," in Proc. 14th Int. Conf. Principles of Security and Trust (POST), Lecture Notes in Computer Science (LNCS), vol. 7662, Springer, 2012, pp. 101–120, doi: 10.1007/978-3-642-35170-9_6.

[12] A. Gehani, R. Ahmad, H. Irshad, J. Zhu, and J. Patel, "Digging into big provenance (with SPADE)," Commun. ACM, vol. 64, no. 12, pp. 48–56, Dec. 2021, doi: 10.1145/3475358.

[13] O. Setayeshfar, C. Adkins, M. Jones, K. H. Lee, and P. Doshi, "GrAALF: Supporting graphical analysis of audit logs for forensics," Software Impacts, vol. 8, p. 100068, 2021, doi: 10.1016/j.simpa.2021.100068.

[14] "Elasticsearch," [Online]. Available: https://www.elastic.co/cn/elasticsearch.

[15] J. Sani, "Improved Log Monitoring using Host-based Intrusion Detection System" - James Sani - AIJMR Volume 1, Issue 1, July-August 2023.

[16] M. Muthusamy, S. V, K. R, and S. R, "Python-based security operations center (SOC) and forensics analysis for incident cyber threats," Int. J. Res. Appl. Sci. Eng. Technol. , vol. 12, pp. 2592–2596, 2024, doi: 10.22214/ijraset.2024.60403.

[17] W. Qiao, Y. Feng, T. Li, Z. Zhang, Z. Xu, Z. Ma, Y. Shen, J. Ma, and Y. Liu, "Slot: Provenance-driven APT detection through graph reinforcement learning," arXiv preprint arXiv:2410.17910, May 2024, doi: 10.48550/arXiv.2410.17910.

[18] Z. Li, Q. A. Chen, R. Yang, and Y. Chen, "Threat detection and investigation with system-level provenance graphs: A survey," arXiv preprint arXiv:2006.01722, Jun. 2020, doi: 10.48550/arXiv.2006.01722.

[19] "PyGraphistry," [Online]. Available: https://pygraphistry.readthedocs.io/en/0.34.17/index.html.

[20] "CVE-2018-19422: Subrion CMS 4.2.1 privilege escalation vulnerability," NIST,

2018. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-19422.