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

Linux 系統中敏感檔案操作之即時監控與日誌規則的動態設定機制

Real-time Monitoring and Dynamic Audit Rule Setup Mechanism for Sensitive File Operations on Linux

張皓棠

HAO-TANG ZHANG

指導教授: 官振傑 博士

Advisor: ALBERT GUAN, Ph.D.

中華民國 114 年 8 月

August 2025

摘要

Linux 系統在使用 auditd 監控敏感檔案操作時，採用預先指定監控路徑與系統呼叫的方式設定日誌規則(以下簡稱靜態式日誌規則)。然而，面對具備長潛伏週期且行為碎片化的進階持續性威脅（Advanced Persistent Threat, APT），這類靜態式日誌規則常無法涵蓋完整攻擊鏈，導致監控記錄出現節點斷裂與資料流追蹤中斷。為彌補靜態式日誌監控系統監控範圍固定的限制，本研究設計並實作了一套針對敏感檔案操作的即時監控與日誌規則動態設定機制，透過行為解析與條件判斷的機制，強化日誌系統對敏感檔案操作產生之資料流向的追蹤能力，並且當可疑操作發生時自動擴展日誌監控範圍，改善靜態式日誌系統監控的不足。系統以 auditd 為基礎，結合事件聚合、指令解碼、行為辨識與條件觸發等模組，建構出即時監控架構，可持續讀取日誌檔案內容，依據 event_id 聚合 SYSCALL 、 PATH 、 PROCTITLE 與 EXECVE 等紀錄，還原原始執行的指令及其相關的檔案路徑。當偵測敏感檔案發生潛在資料移動時，系統會自動將新產生的目標檔案納入監控範圍，持續擴展資料追蹤能力，並且當實驗系統關於異常事態所訂條件被觸發時，實驗系統中條件式觸發機制將動態載入額外日誌規則，即時擴大日誌系統的監控範圍，落實以行為為驅動的監控策略，提升資料流追蹤的即時性與完整性。

關鍵字： Linux、auditd、敏感檔案操作、日誌規則、條件式觸發機制、溯源圖

Abstract

Traditional Linux systems rely on static auditd rules to monitor sensitive file operations, but these rules often miss fragmented, long-dormant APT activity, resulting in gaps and incomplete data flow tracking. To mitigate this gap, we implemented a real-time monitoring system that aggregates events (SYSCALL, PATH, PROCTITLE, EXECVE) by event_id, decodes commands, and uses behavior analysis with conditional triggers to dynamically configure audit rules. When suspicious operations on sensitive files are detected, the system automatically adds new target paths and When suspicious behavior trigger conditional trigger mechanisms,the system automatically load dynamically loads additional rules to broaden auditd's coverage of potential exfiltration channels, enhancing both the completeness and timeliness of data flow tracking.

Keywords: Linux, auditd, sensitive file operations, audit rules, conditional triggering mechanism, provenance graph

目錄

Page

摘要 **i**

Abstract **ii**

目錄 **iii**

圖目錄 **v**

第一章 緒論 **1**

1.1 研究背景 1

1.2 研究動機 2

1.3 研究貢獻 3

第二章 研究背景與監控基礎 **5**

2.1 進階持續性威脅與行為特性 5

2.2 靜態與動態監控方法之探討 6

2.3 Auditd 日誌格式與欄位說明 7

2.4 Auditd 監控限制與補強動機 10

2.5 事件聚合方法與追蹤流程 11

2.6 條件式判定與日誌規則的動態設定 11

2.7 SPADE 溯源圖系統與資料流分析 12

第三章 系統設計與功能說明 **14**

3.1 架構設計總覽 14

3.2 系統模組設計與功能說明 16
3.2.1 日誌監控模組 16
3.2.2 事件聚合模組 16
3.2.3 指令解碼模組 17
3.2.4 行為辨識模組 18
3.2.5 動態規則設定模組 19
3.2.6 條件判別模組 19
3.2.7 紀錄與排除模組 21
第四章 實驗設計與結果分析 **22**
4.1 實驗環境與使用工具 22
4.2 功能驗證測試 23
4.3 靜態精簡規則監控結果 23
4.4 行為驅動監控結果 24
4.5 條件式觸發機制結果 25
4.6 階段性總結 26
4.7 敏感檔案追蹤與動態規則設定驗證 27
4.8 實驗結果分析 32
4.9 實驗系統效能與限制討論 33
第五章 結論與未來展望 **36**
5.1 結論 36
5.2 未來展望 37
參考文獻 **39**

圖目錄

2.1 cp 操作所觸發的來源與目標檔案之事件記錄 9

3.1 系統架構與資料流程圖 15

4.1 初始的日誌規則設定 23
4.2 觸發所需之規則 23
4.3 精簡靜態規則監控下的溯源圖 24
4.4 行為驅動監控下的溯源圖 24
4.5 行為驅動監控後的日誌規則 24
4.6 條件式觸發機制後的日誌規則 26
4.7 條件觸發規則設定下的溯源圖 26
4.8 初始規則 29
4.9 額外規則項目 30
4.10 Samba 檔案外流總溯源圖 30
4.11 cp 操作檔案流向溯源子圖 31
4.12 mv 操作檔案流向溯源子圖 31
4.13 Samba 檔案外流溯源子圖 32

第一章　緒論

1.1　研究背景

進階持續性威脅（ Advanced Persistent Threat, APT ）是一種長期潛伏且隱匿性高的攻擊手法，攻擊者會在系統中潛伏一段時間，根據不同階段的任務逐步實施橫向移動、權限提升，以及敏感資料的存取或外傳等行為 [1,2]。這類攻擊通常分散於不同時間點，並透過合法指令實施，使整體行為看似正常系統操作，難以僅憑單次指令或日誌紀錄察覺其潛在目的。

在現行的 Linux 環境中，儘管 auditd 提供了系統稽核能力，但日誌監控規則的特性必須事前明確指定路徑或系統呼叫(以下統稱「靜態」特性，詳見第 2.2 節)，才能對特定目標進行監控。 APT 類型攻擊往往利用這項限制，透過一連串看似合法的操作，逐步完成資料竊取與滲透流程。攻擊者可能先複製一份敏感檔案，接著將其移動到隨機產生的路徑，或重新命名為難以辨識的亂數檔名，藉此避開原本靜態規則的監控條件。這樣的操作流程，對於仰賴事前預測攻擊路徑所設下的靜態監控來說，常常難以即時反應，導致原本用於監控敏感檔案的日誌規則失效。當新的檔案或路徑不再屬於被監控範圍，後續所有如再次複製、修改、刪除或外傳的動作將無法被記錄，造成攻擊鏈的監控紀錄中斷，資料流向無從還原，系統管理者也難以掌握攻擊者的後續行為與資料的流向。

APT 攻擊的危害不僅在於其具備深度滲透能力，透過長週期、多階段的操作造成日誌紀錄上的斷點，更在於其能長期運作於日誌監控範圍之外，使系統無法持續追蹤關鍵行為 [1]。本研究即是在此背景下，試圖補足靜態監控規則的不足，強化對資料操作行為的即時追蹤能力，提升稽核紀錄的連貫性與完整性。

1.2 研究動機

在資訊系統中，敏感檔案通常包含帳號憑證、系統設定、用戶資料，以及與存取控制或安全策略相關的設定檔。當敏感檔案遭到未授權的存取、複製或外傳時，不僅可能造成系統內部資料的直接洩漏，也可能使攻擊者擴大滲透的範圍 [1]。這類操作一旦未被即時偵測與記錄，後續可能逐步演變為持續且難以追蹤的攻擊行為鏈，使整體行為路徑變得難以分析還原。在現今多數組織高度仰賴檔案系統來儲存重要資料的情況下，若缺乏對這類檔案的即時監控與追蹤機制，將使得攻擊行為長時間處於未被掌握的狀態，也讓系統管理者無法在第一時間掌握資料是否已被異常存取或傳出，使系統管理者難以及時掌握異常情況，錯失提早阻止攻擊的機會。因此有效監控並追蹤這類檔案的存取與流向，是維護系統完整性與資料安全的關鍵。尤其在 APT 攻擊日益頻繁的背景下，攻擊者常以合法操作逐步滲透系統，隱蔽地操作敏感資料，使傳統靜態監控機制難以有效防禦。雖然 Linux 系統內建 auditd 作為日誌稽核機制，可針對檔案操作進行監控，但如前所述，其靜態規則的限制使得使用上顯得侷促。尤其在設定監控規則時，必須明確指定檔案路徑或系統呼叫，當面對攻擊者動態產生的新檔案時，往往難以及時涵蓋，導致監控出現缺口。為了確保攻擊鏈中的所有操作皆能被記錄，一種可行作法是設定全面性的監控規則，涵蓋所有系統呼叫。然而，這種方法將造成系統效能負擔顯著增加，並產生大量的日誌紀錄，使得關鍵事件紀錄混合在龐大資料中，同時，龐大的資料量也會導致分析速度與準確性降低，也會使每筆日誌紀錄在分析時的重要性與分析效益大幅下降 [3]。因此，本次研究希望補足這個日誌監控系統的盲點，設計一套能根據系統事件動態擴大監控日誌規則的模組，讓系統在日常運作時以較精簡的日誌規則運作，當所預設的異常發生時，實驗系統將自動的載入預設的監控規則，自動的擴大監控的範圍，增強捕捉攻擊鏈的能力。此外，實驗系統也包含對於監控檔案持續追中的功能，當偵測到監控中的敏感檔案被執行特定的操作後產生新的檔案或移動致新的路徑，系統會自動將該新的檔案路徑加入監控範圍，同時也可自動地將新的檔案路徑添加到日誌規則中。

平衡系統效能與資訊安全的考量，讓系統平時以較低的系統耗能的方式運行日誌監控，當條件判別模組偵測到指定行為條件達成後，實驗系統將自動載入額外的日誌監控規則，擴大日誌規則所涵蓋的範圍，並且結合對於特定檔案持續追

蹤的功能，加強系統日誌的監控能力及提升對於攻擊鏈的紀錄完整性，提升靜態規則的彈性，增強日誌系統的溯源能力。

1.3 研究貢獻

本研究以監控敏感檔案操作為核心，設計一套具備條件式反應能力與日誌規則動態擴張機制的即時監控系統，目標在於補足過往靜態監控策略無法完整涵蓋資料流向的斷點問題。整體架構設計使得日誌監控系統於平時僅啟用涵蓋範圍明確的精簡規則，維持低系統負載下運作，當偵測到可疑徵兆（如非預期連線行為或針對敏感檔案的持續操作等）並且行為達到系統的觸發閥值時，系統便會自動載入預先準備的額外監控規則，動態擴大日誌涵蓋範圍，使原本需要事先指定監控路徑的靜態日誌系統具備彈性擴展能力，在資訊安全與系統效能間取得實際的平衡，也進一步強化對攻擊行為的捕捉與溯源能力。

針對現有 auditd 透過 -w 所定的日誌規則單點監控檔案本身存取、無法涵蓋檔案被操作後資料流動的侷限，實驗系統透過「行為語意還原」藉由設計行為語意還原模組，還原 proctitle 的原始執行指令，從日誌記錄中提取並還原完整的原始操作，以及結合 PATH 型日誌交叉驗證檔案來源與目標。藉此，當敏感檔案發生特殊操作並產生新目標檔案時，系統能即時自動將該目標路徑納入監控清單，並進行動態規則擴充，讓監控範圍可隨資料流向自動擴展，補足純靜態規則下難以即時覆蓋的監控盲區，提升資料外洩場景中行為鏈的完整追蹤。實驗系統在傳統 auditd 僅能針對單一路徑進行靜態監控之限制下，提出結合 proctitle 解碼與 PATH 日誌解析的架構，得以強化敏感檔案於各操作階段、各種資料流動下的監控盲點。透過即時還原指令語意，搭配目標檔案自動納入監控範圍，提升日誌監控系統對於資料流追蹤的完整性與及時性。

此外，系統整體架構採模組化設計，包含監控條件判斷、異常行為觸發、規則自動擴充、日誌紀錄處理等功能模組皆可獨立調整。模組化的設計可依照不同的攻擊情境進行客製化的調整，不論攻擊手法、觸發條件或新型攻擊場景如何變動，系統皆能彈性調整或擴充相關模組，保持實驗系統擴充與功能升級上的彈性，不需要大幅度改動整體的實驗系統。同時，模組化的設計，也利於後續系統的維護性，模組化的設計使實驗系統能快速因應所面臨的資安型態變化，也確保

後續系統運行的穩定性與可持續發展性。實驗系統的模組化設計不僅強化了系統在不同場景下的應用彈性，也讓 auditd 類監控架構能更有效因應攻擊事件、補強原有盲點，達到資料流向持續追蹤、攻擊路徑精準還原的目標。

實驗系統整體架構針對傳統 auditd 日誌監控系統於靜態規則環境下存在的行為紀錄中斷問題，提供系統性補強方案，有效提升對完整攻擊行為鏈及敏感檔案流向的追蹤覆蓋率與監控精確度。實驗系統透過執行階段與功能導向之模組化設計原則，將監控邏輯明確劃分為六個獨立且可協同運作的功能模組，使日誌監控系統相較於傳統單一式架構，具備更高的彈性擴充能力、更低的系統效能負載，以及更完整的自動化監控機制。

實驗系統的成功導入與實際部署，能使原本受限於靜態預設規則的 auditd 監控機制轉化為具備智慧感知與動態適應能力的即時監控平台。系統可依據即時偵測結果與異常行為模式，自主性地動態調整監控涵蓋範圍與日誌規則配置，針對傳統靜態監控於APT攻擊情境下易產生追蹤斷點的部分進行改善。此項核心貢獻不僅強化日誌監控系統對敏感資料流向的持續追蹤能力與攻擊行為發生時的即時反應能力，同時，為Linux環境下的資安監控提供兼具效能與安全性的動態防護的改進方向。

第二章　研究背景與監控基礎

2.1　進階持續性威脅與行為特性

進階持續性威脅（Advanced Persistent Threat）攻擊通常具有較長的攻擊週期，以及較為碎片化、偽裝成正常指令的攻擊手法，導致在注重效能的前提下，傳統的靜態監控規則無法涵蓋完整的攻擊鏈。在 Lockheed Martin 所提出的 Intrusion Kill Chain 模型中，將攻擊行為劃分為七個階段： Reconnaissance 、Weaponization 、Delivery 、Exploitation 、Installation 、Command & Control（C2）與 Actions on Objectives 。本系統所針對的階段，主要為 Installation 、Command & Control與 Actions on Objectives [4]，聚焦於此階段中敏感資料的外洩與異常連線的建立等行為，其中， Command & Control(C2) 階段為攻擊者與已入侵主機之間建立穩定控制通道的過程，常透過反向連線、隧道協定（tunneling protocol）或合法通訊軟體持續與外部主機溝通，為本系統識別異常連線行為的主要監控對象之一。

本次實驗所設計的監控系統，不直接與 kernel 互動，具備可移植性高、部署彈性佳等特性。系統能在偵測到異常徵兆後，動態的擴展日誌規則的監控範圍，並對後續涉及的敏感資料持續監控，補足傳統靜態規則的不足。整體設計目標為：在效能負擔低的情況下，於異常情境自動擴大監控範圍，盡可能保留與還原完整的攻擊軌跡。

2.2 靜態與動態監控方法之探討

在技術討論中，根據不同工具或方法的特性，常對其進行靜態或動態的分類。根據 IBM(International Business Machines Corporation) 對於「使用靜態和動態環境定義」的定義 [5]，區分是否為靜態與動態的主要依據為「配置發生的時間點」以及「配置是否會因系統呼叫而改變」，其中「靜態環境定義用來配置準備時間性質」顯示當準備階段完成後，所設定的配置便會固定，不再變更，「動態環境定義用來配置執行時間性質」中，則是指配置會在執行的階段中，依照變化進行調整。

從定義中，我們可以大致整理出，當一些詞彙中具有「靜態」、「動態」等詞語時，大致可以從這些描述中，推測出詞語之中所蘊含的特性。「靜態」一詞往往具有「事前」、「明確範圍」、「固定」等概念包含其中，而相對的「動態」一詞則是具有「過程」、「隨機應變」、「彈性」的概念涵蓋其中。

從上面的討論中，可以歸納出一個簡單的模式定義:

靜態模式：在這個模式當中，核心的重點在於其「確定性」以及「固定性」上，具有明確的定義與規範，易於執行與遵循。但與此同時，其較無法應對突發情況。

動態模式：相對於靜態模式，動態模式的核心在於「彈性」與「適應性」。在這種模式下，系統或方法會根據執行過程中的實際狀況，隨時調整自身的配置或行為。這種特性使得動態模式能夠靈活應對環境變化與突發事件。

在本次討論中，前述靜態與動態模式之基本概念亦納入相關系統特性與描述，藉由明確定義相關新詞彙，有助於簡化行文內容，提升論述的精確性與易讀性。依循此脈絡，本文將「靜態監控」、「動態監控」及「靜態規則」等術語加以界定，作為後續討論與說明之基礎。

- **靜態監控**：靜態監控指系統於運行前即由管理者明確指定監控之對象、路徑及條件，其監控範圍在系統執行期間保持不變。此類監控方式仰賴事前完整預測可能之威脅路徑，僅針對事先納入規則範疇的檔案或操作進行日誌記錄。

- **動態監控**：動態監控指系統於執行期間，能根據即時偵測到之行為或異常條件，主動調整或擴充監控範圍。當系統偵測到敏感檔案被操作、產生新檔案、或出現特定異常行為時，能即時自動將相關目標納入監控，並載入對應之日誌規則。

- **靜態規則**：靜態規則指日誌監控系統中，部署或設定階段即預先定義之監控規則。該等規則內容限定於設定時明確指定之檔案路徑、系統呼叫或相關操作，在系統執行階段維持不變。僅有符合規則所約定之監控物件，操作行為方能於日誌中產生記錄。

2.3 Auditd 日誌格式與欄位說明

在本次實驗中，使用 auditd（Audit Daemon）作為系統稽核日誌的來源，記錄作業系統中與安全相關的低階操作，包括檔案的讀寫行為、系統呼叫的觸發，以及網路連線的建立等。實際上，單一操作事件在 audit.log 中通常會被拆分為多筆紀錄，並透過 type 欄位區分其角色與內容。例如，SYSCALL 表示主要的系統呼叫資訊，PATH 用於紀錄所存取的檔案路徑，PROCTITLE 則包含執行時的完整命令與參數，而 SOCKADDR 則提供連線對象的位址資訊等。

為將這些分散的紀錄還原為單一操作，本系統會解析每筆日誌的 msg 欄位，提取事件的 event ID，並以此作為聚合依據。聚合後的事件資料會包含操作指令、檔案來源與目標、執行路徑等欄位，提供後續辨識可疑行為所需的要素。

由於單一操作會產生多筆紀錄，且各紀錄著重不同欄位，在本系統中，程式會透過 event ID（即 msg=audit() 中的編號）辨識是否為同一筆操作，並據此進行聚合，還原完整的行為語意。以下列出系統使用到的紀錄類型：

- **SYSCALL**：表示使用者觸發的系統呼叫操作，為每筆行為的核心事件之一。欄位中包含 syscall 編號，可用於辨識對應的系統呼叫類型，例如：2 代表 open，42 代表 connect。此外，該事件中亦包含執行該行為的執行檔案路徑（exe）、觸發該行為的使用者編號 uid、以及 a 用以判斷開啟檔案權限，有助於後續建立行為主體與來源進程的追蹤關聯。

- **PATH**：表示系統呼叫中所涉及的檔案路徑。常見欄位包含實體檔案路徑

name 、檔案順序編號 item ，以及檔案類型 nametype 。在 auditd 的設計下，只有部分系統呼叫（如 rename 、 link ）且同時將來源與目標檔案納入監控時， PATH 類型日誌才可能同時出現 item=0 （來源檔案）和 item=1 （目標檔案）。對於 cp 、 openat 等多數常見檔案操作，透過 -w 規則產生的日誌，僅會針對規則所監控的檔案產生紀錄，來源檔案與目標檔案會分別產生不同的 event_id ，且每筆 PATH 紀錄的 item=0 ，對應的僅是該事件下被監控的唯一檔案路徑。

- **PROCTITLE**：欄位中為使用者實際執行指令的參數內容，通常以十六進位（hex）編碼方式儲存。例如：6370202f6574632f70617373776420 0f746d702f7031 對應的指令為 cp /etc/passwd /tmp/p1 。系統會將其解碼還原為可讀格式，作為行為語意判斷與觸發條件的依據。

- **SOCKADDR**：為與連線相關的附加資訊欄位，通常會伴隨 connect 系統呼叫一併出現，並紀錄該次連線的 IP 位址與通訊埠號。在本系統中，雖主要以 SYSCALL 中的 syscall=42 辨識 connect 行為，但 SOCKADDR 提供的目標位址資訊亦能輔助後續分析，辨識是否為對外連線或可疑主機的 C2 連線行為。

- **EXECVE**：表示系統在執行程式時所接收的指令參數內容，常作為 PROC TITLE欄位的補充。與 PROCTITLE 不同的是， EXECVE 中會以 a0 、 a1 、a2 等欄位分別列出參數陣列（ argv[] ）中的每一個元素，並可完整還原包含主程式名稱與參數順序的執行內容。

- **CWD**：表示事件發生當下該進程所處的工作目錄，常以 cwd="/home /user"的形式出現在日誌中。該欄位可用於後續強化 PATH 或 PROCTITLE 中的相對路徑，補足以相對路徑執行之操作的上下文資訊。

範例為日誌監控系統在日誌監控規則同時涵蓋來源檔案路徑與目標檔案路徑為前提之下，執行 cp /etc/passwd /tmp/attack_passwd 後所得到的日誌紀錄。

範例:複製操作，同時監控來源與目標檔案所產生的兩組 event 記錄

```
type=PROCTITLE msg=audit(1749460232.090:4860): proctitle=6370002
    F6574632F706173737764002F746D702F61747461636B5F706173737764
type=PATH msg=audit(1749460232.090:4860): item=0 name="/etc/passwd" ...
type=CWD msg=audit(1749460232.090:4860): cwd="/home/joshua"
type=SYSCALL msg=audit(1749460232.090:4860): ... comm="cp" exe="/usr/bin/cp" key
    ="passwd_watch"
```

(a) 來源檔案 /etc/passwd 的事件記錄

```
type=PROCTITLE msg=audit(1749460232.090:4861): proctitle=6370002
    F6574632F706173737764002F746D702F61747461636B5F706173737764
type=PATH msg=audit(1749460232.090:4861): item=0 name="/tmp/attack_passwd" ...
type=CWD msg=audit(1749460232.090:4861): cwd="/home/joshua"
type=SYSCALL msg=audit(1749460232.090:4861): ... comm="cp" exe="/usr/bin/cp" key
    ="tmp_watch"
```

(b) 目標檔案 /tmp/attack_passwd 的事件記錄

圖 2.1: cp 操作所觸發的來源與目標檔案之事件記錄

從紀錄中可以看到，在執行 cp 操作時，來源檔案與目標檔案分別各自產生一筆日誌紀錄，內容包含 PROCTITLE 、 PATH 、 CWD 、 SYSCALL 等欄位。

其中，每筆日誌記錄開頭皆包含 msg=audit() 欄位，格式為 msg=audit(...)，括號中為時間戳記與事件編號。以來源檔案的範例為例，msg=audit(1749460232.090:4860) 其中 1749460232.090 為該筆事件產生的時間戳（以 epoch 秒為單位），而 4860 則為該操作所對應的 event ID 。本系統即以 event ID 為依據，將同一操作中出現的 SYSCALL、PATH、PROCTITLE 等紀錄整合為一筆行為事件，作為後續語意分析與行為條件判斷的基礎。

然而， auditd 僅會對明確設為監控目標的檔案產生 PATH 類型的日誌記錄。若攻擊者將監控中的資料複製或移動至未在監控規則涵蓋範圍的新路徑，則 audit.log 中將不會出現該目標檔案的紀錄，進而造成 SPADE 溯源圖節點斷裂。此類侷限亦為本系統設計動態新增監控規則與即時擴張監控範圍的動機來源，旨在補足 auditd 無法即時涵蓋的新產物路徑，強化日誌完整性與可追溯性。

2.4 Auditd 監控限制與補強動機

auditd 採用靜態規則的監控機制，僅對於規則中涵蓋的檔案路徑以及操作進行監控和記錄，需事先使用 -w 明確指定檔案路徑，或使用 -a 指定所監控的操作類型， auditd 的日誌記錄才會對後續與相關檔案路徑或系統操作有關的行為進行記錄。

此外，當涉及檔案存取操作時， auditd 的日誌類型中會使用 PATH 類型對其進行額外的內容記錄，但 PATH 類型的日誌紀錄僅限於符合規則所涵蓋的路徑。假如在進行檔案存取操作時，僅來源檔案路徑在日誌規則的監控範圍中，而另一目標檔案並未被納入規則設定，則該目標檔案將不會產生 PATH 記錄，進而導致行為鏈在日誌層級無法被完整還原。

例如：當日誌系統所監控的敏感檔案被執行複製行為時，若日誌規則並未涵蓋所生成的新增檔案路徑，則系統僅會針對來源檔案產生存取紀錄，而不會記錄目標檔案的相關操作。在 auditd 以 -w 監控規則時， cp 複製操作會被拆分為兩筆獨立事件：一筆對應來源檔案，一筆對應目標檔案。只有被納入監控規則的路徑才會產生相對應的 event 紀錄。因此，若僅來源檔案受到監控，則日誌中僅會出現來源檔案的 PATH 、 openat 等操作紀錄，目標檔案相關的 event 則完全缺失。在這樣的情況下，從日誌僅可觀察到來源檔案被開啟（openat）與存取時的權限資訊，但無法得知最終複製動作所產生的新檔案路徑或其後續的存取紀錄。這將導致在溯源分析過程中，缺乏新增檔案的節點資訊，進而出現資料流追蹤的斷點。

由於 provenance graph 的構建高度依賴 PATH 類型日誌中所揭露的檔案路徑資訊，因此當目標檔案未被納入監控時，即使其操作行為在系統層級仍可觀察，卻無法被轉化為可視化的節點與邊，最終導致行為鏈於圖中產生節點中斷的現象，造成攻擊流程在圖形上無法被完整重建。

基於上述限制，本研究提出一套針對檔案操作進行辨識與動態規則設定的監控系統，能在偵測到敏感檔案被操作後，立即將新生成的目標檔案自動納入監控規則中，補足原始靜態規則的不足，進而提升日誌層級對於檔案傳播行為的涵蓋能力與追蹤完整性。

2.5 事件聚合方法與追蹤流程

在 auditd 日誌監控系統中，單一的指令操作往往會被拆分為多筆不同類型的紀錄，各類型的日誌所紀錄的重點也有所不同。為使監控系統能判斷這些紀錄是否屬於同一筆操作，本實驗所使用的監控系統會從每筆紀錄的 msg=audit(...) 欄位中擷取出 event ID，作為識別操作單位的依據。系統以 event ID 為聚合核心，整合來自不同 type 的紀錄內容，以還原當時的操作語意。

實作上，系統會使用以 event ID 為 key 的 dictionary 結構來儲存同一事件的多種紀錄。假如某次操作同時產生了 SYSCALL、PATH、PROCTITLE 與 CWD 等紀錄，整合後的資料格式如下：

```
event_dict['event_ID'] = {
    'SYSCALL': {...},
    'PATH': [...],
    'PROCTITLE': {...},
    'CWD': {...}
}
```

當系統讀取到這筆操作時，會先解碼 PROCTITLE 欄位，還原使用者實際輸入的指令，並判斷是否為目前關注的操作類型。接著，系統會從 PATH 記錄中擷取 item=0 的檔案路徑。在部分操作情境下，若系統呼叫同時涉及多個監控路徑，則可能在同一事件中同時出現多個 item。實驗系統會將 item=0 所對應的來源路徑與敏感檔案清單比對，若屬於敏感檔案，便將該事件標記為高風險並寫入外部紀錄。同時，也會將新出現的目標檔案路徑動態加入監控規則，持續擴展監控範圍，得以減少攻擊過程中的行為斷點並持續追蹤敏感資料流向。

2.6 條件式判定與日誌規則的動態設定

為補足靜態監控規則僅能涵蓋其所指定的監控路徑與系統呼叫，並同時兼顧系統效能與日誌紀錄的準確性，本實驗所設計之系統採用以異常徵兆為核心的監

控擴張機制。當系統偵測到特定高風險行為，或發現相同程式在固定時間內重複執行特定操作超出預設閾值時，便會自動載入額外的監控規則，即時擴大監控涵蓋範圍，以強化日誌系統對完整攻擊鏈的紀錄能力。

實驗系統中針對異常徵兆識別與動態擴張策略，參考 Zeng 等人提出之設計原則，將系統呼叫 connect 作為判斷系統異常活動的關鍵指標之一 [6]。由於特權使用者在系統中可能執行大量合法且複雜的操作，若一併納入監控，將產生大量合法操作紀錄，導致真正攻擊跡象被稀釋。因此本系統聚焦於非特權使用者(uid≠0)的行為，作為條件式監控與行為判定的主要對象。

根據常見的 APT、資料外洩 行為模式的觀察與防禦考量。攻擊者在滲透主機取得低權限存取權後，最常見的操作之一，就是反覆透過同一 shell 或惡意工具與遠端 C2(Command and Control) 伺服器建立多次連線。這類「連線爆發」行為在短時間內極為密集，明顯區隔於一般用戶或合法程式的正常網路操作。藉由鎖定 UID 不等於 0 (非特權帳號)且同一指令或可執行檔路徑作為多次連線的聚合準則，系統可有效過濾大量無害背景流量，專注捕捉真正異常的攻擊徵兆。

本系統設計基於連線密度的條件式觸發機制：當同一非特權使用者於預設時間範圍內多次發出 connect 系統呼叫 (syscall 42)，且來源來自相同的 proctitle 或執行路徑(exe)，系統即視為連線爆發事件。此外，系統也參考進階攻擊手法中常見的動態模組劫持攻擊模式制定條件，當發現 /lib64/ld-linux-x86-64.so.2 在特定時間區間內的被讀取次數，當被判斷為存取異常頻繁時，則被視為動態模組劫持攻擊(sideload)的潛在風險 [7]。上述任一條件一旦成立，便會觸發行為驅動的監控擴張機制，動態載入預先定義之規則檔，並逐條註冊其中條目，以即時擴大對暫存目錄或其他潛在落點的監控範圍。

2.7 SPADE 溯源圖系統與資料流分析

auditd 雖然能記錄系統層級的稽核事件，但日誌內容為事件記錄導向，雖然分為不同類型的紀錄，但紀錄內容聚焦在單一的操作上，缺乏上下文的關聯性，一旦操作的時間跨度較長，便會容易使前後地操作難以關聯，導致日誌紀錄內容發散。這樣的特性在面對 APT 類型攻擊時顯得侷限，特別是在重建跨階段、時間錯開的攻擊行為鏈時，容易出現斷點，無法串接整體資料流向。

為解決上述紀錄分散、缺乏關聯性的問題，研究於實驗引入SPADE(Support for Provenance Auditing in Distributed Environments) [8,9]，作為資料流向驗證與圖形化視覺分析的重要工具，負責將日誌紀錄中所紀錄的事件轉更容易解讀資料流的溯源圖，幫助理解檔案、進程以及網路封包間的互動關係。

SPADE 並非替代 auditd ，而是將 auditd 所記錄的稽核日誌作為輸入來源，透過內建的轉換接收器（reporter）將原始的紀錄內容轉換為溯源圖（Provenance Graph），在 SPADE 所生成的溯源圖中，節點代表檔案或進程，邊則代表節點之間的互動關係。 SPADE 藉由溯源圖的方式將過往分散的日誌操作紀錄整合為具上下文關聯的節點與邊。

在轉換過程中， auditd 的各類型紀錄將映射至不同角色的圖形元素，例如：SYSCALL 紀錄會產生代表進程活動的節點， PATH 對應至檔案資源節點， CWD 提供目前工作目錄以補足實際存取路徑， SOCKADDR 則用於建立網路連接的節點資訊。

在本系統中， SPADE 並非僅用於驗證監控規則是否在正確時機生效，更進一步用來確認整個攻擊或操作行為是否能透過日誌紀錄完整重現。當某一檔案歷經複製與搬移等多階段操作，若系統於操作進行前，已將所有中繼與目的路徑納入監控範圍，則 SPADE 所產出的溯源圖將能串接來源與最終目的檔案路徑，呈現連貫的資料流。然而，若監控規則於行為完成後才補上，則後續目標節點導致缺乏即時稽核紀錄而形成圖中斷裂節點，無法重建完整行為鏈。

如同前面所提「 SPADE 並非替代 auditd 」 SPADE 本身不具備補全遺失資訊能力的特點，其圖形是否連貫，取決日誌規則是否覆蓋攻擊發生的路徑 [10] 為進一步分析特定的溯源圖節點，實驗使用 SPADE 專門的查詢語法 QuickGrail 擷取出所需的特定子圖或節點資料 [11]。透過 QuickGrail查詢，可以針對關鍵節點進行進行路徑還原，可幫助分析攻擊方的行為操作以及判斷其意圖。在實驗過程中，透過分析 SPADE 所生成的溯源圖，可掌握資料於系統中的流向與互動，也能藉由圖形化視覺比對有無實驗系統介入的溯源圖差異，驗證本系統的動態擴張機制是否成功補足原先靜態監控規則無法涵蓋的部分，並確認各項設計是否於真實操作中確實發揮效用。

第三章　系統設計與功能說明

3.1　架構設計總覽

實驗系統以日誌紀錄系統 auditd 為基礎，建構出基於行為驅動的動態監控框架。由於 auditd 對於系統呼叫與路徑操作具有即時記錄的特性 [12]，因此本系統以 auditd 為核心，藉由其稽核能力對系統行為進行解讀與分析。然而，auditd 的靜態監控規則，其覆蓋範圍與系統效能之間存在密切關聯。若涵蓋過廣，可能導致日誌量爆增，進而拉高系統負擔，若涵蓋過窄，則可能遺漏關鍵行為紀錄，降低整體的可溯性 [12]。特別是在 APT 攻擊情境中，其具備多階段、長週期的特性，若無法在各階段即時涵蓋，將可能導致資料流無法串接 [13]。

為平衡攻擊紀錄的完整性與系統效能的負擔，實驗系統參考 Zeng 等人提出的條件驅動式日誌擴張設計原則 [6]，藉由異常徵兆作為啟動監控擴張的依據，避免監控規則過度膨脹。在 Zeng 等人所提出的設計中，建議重點監控系統呼叫 connect 作為可疑行為的觸發徵兆。實驗系統採納其論點，並做出調整，將觸發擴張機制的可疑操作條件設定為'當非特權使用者在短時間內，重複由相同執行檔案路徑發出 connect 系統呼叫（syscall 42）'的情況。在非特權使用者（uid!=0）在短時間內重複使用相同程式進行連線行為，將被視為潛在的系統威脅。

除了延續原本對 connect 呼叫的監控思路外，也考量到偵測對象的選擇將直接影響整體資訊的判讀與準確性。由於將特權使用者在系統中可能執行大量合法且複雜的操作，若一併納入監控，容易產生大量非異常事件，進而掩蓋真正的攻擊徵兆。因此，本系統聚焦於非特權使用者（uid!=0）進行條件判定，將焦點設定在非特權使用者上，以提升紀錄結果的實用性與判讀效率 [14]

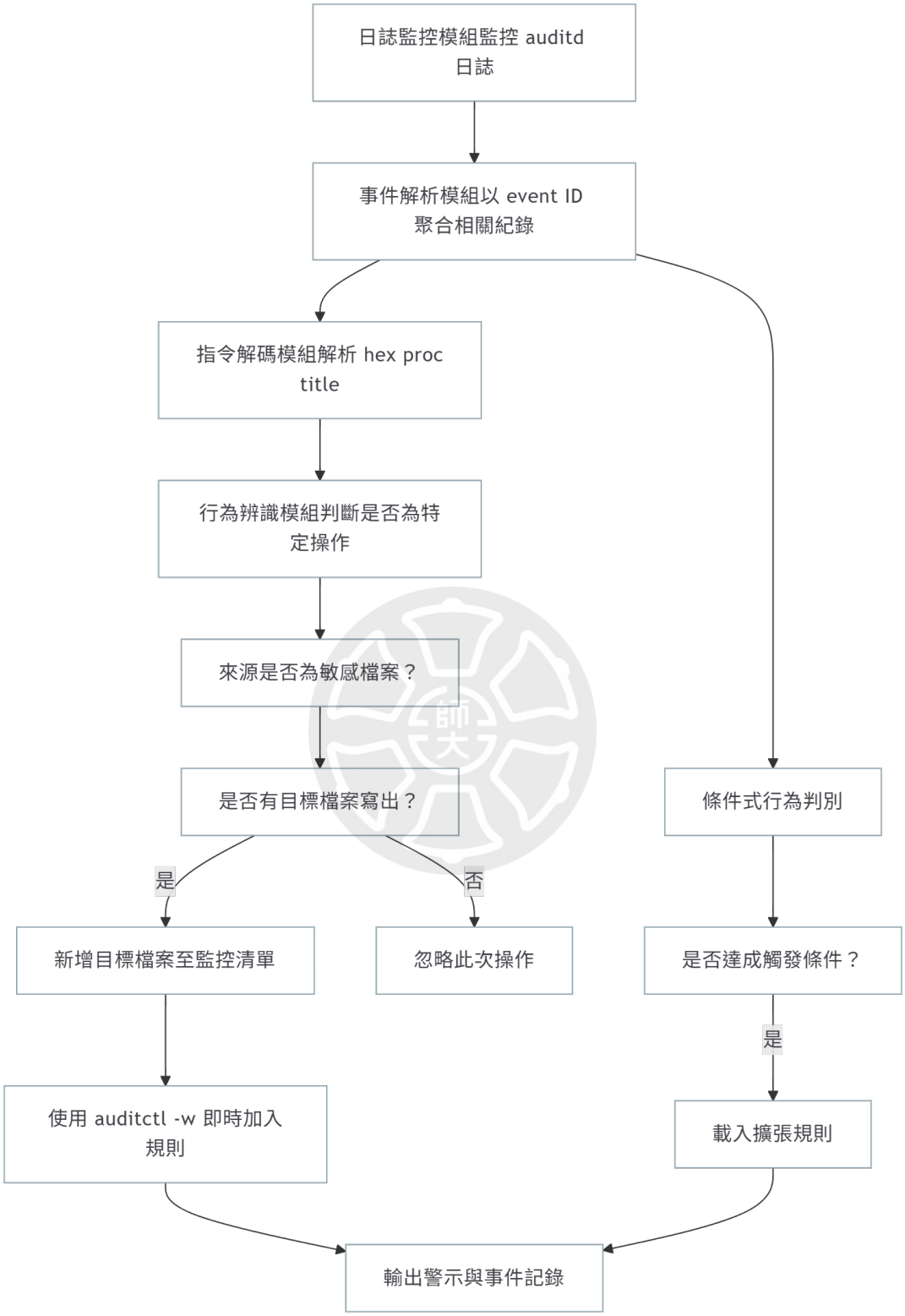

圖 3.1: 系統架構與資料流程圖

3.2 系統模組設計與功能說明

本系統依據功能可區分為七個模組，分別為：**日誌監控模組**、**事件聚合模組**、**指令解碼模組**、**行為辨識模組**、**動態規則設定模組**、**條件判別模組**與**事件紀錄模組**。各模組各司其職，透過模組間的訊息傳遞與條件觸發，建構出具備動態擴展能力與即時反應特性的日誌監控流程。本系統可直接部署於現有 Linux 環境中，不需修改 auditd 的原始結構，即可在系統效能與監控完整性間取得平衡。

3.2.1 日誌監控模組

本模組負責持續讀取 audit.log 檔案，以游標記錄先前所記錄的讀取位置，避免重複處理舊有紀錄。以及將原始的日誌資料轉為結構化字典的解析流程，透過 os 、 time 、 re 等 Python 所內建的函式庫，將讀取每筆日誌進行基本欄位解析，將其轉換為 dictionary 形式，提取紀錄的類型(例如： SYSCALL 、 PATH 、 CWD 、 PROCTITLE 等）並透過 Python 內的 dict 與 set 等資料結構，擷取該日誌類型所對應資料格式，透過制定字串處理規則以及欄位對應表，作為後續事件聚合的前置處理步驟。

```
while true:
position = read_cursor(POSITION_FILE)
new_lines = read_file_from(AUDIT_LOG, position)
for line in new_lines:
    record = parse_line_to_dict(line)
    buffer.append(record)
update_cursor(POSITION_FILE, file_current_position())
process_buffer(buffer)
```

3.2.2 事件聚合模組

由於單一操作指令常伴隨多筆不同類型的日誌紀錄，系統將每筆紀錄的 msg=audit(...) 中的 posix_time 與 event_id 進行分離，各自儲存到對應的格

式，並且藉由比對分離出的 event_id 判斷紀錄是否屬於同一操作，將相同操作的紀錄進行聚合。將事件整合成 dictionary 結構進行儲存，格式如下：

```
event_dict['event id'] = {
'SYSCALL': {...},
'PATH': [ {...}, {...} ],
'PROCTITLE': {...},
'CWD': {...}
}
```

此設計可使事件成為分析的最小單元，後續模組皆以聚合後事件為基礎進行比對與判斷，提升整體邏輯一致性與解析效率。

```
event_groups = {}
for record in buffer:
    id = extract_event_id(record.msg)
    if id not in event_groups:
        event_groups[id] = initialize_event_structure()
    merge_fields(event_groups[id], record)
```

3.2.3 指令解碼模組

當模組接收到每筆經過聚合的事件資料後，針對 PROCTITLE 類型日誌中的 proctitle 欄位值進行處理。首先利用 Python 內建的 re 函式庫，以正規表達式檢查字串是否符合十六進制字符組成並且字串長度為偶數，當 proctitle 符合此條件，隨即透過 binascii 函式庫將十六進制字串還原為方便解讀的原始執行指令，並在解碼結果中去除不可列印字元，進一步還原為可讀的原始執行指令文字。此解碼過程同時包含格式驗證與異常錯誤處理機制，當遇到格式不符或解碼錯誤時，系統會回退使用預設值以維持流程穩定性。

```
for event in event_groups:
    raw = event.proctitle_hex
    if is_hex_string(raw):
        decoded = hex_to_bytes(raw)
        text = clean_nonprintable(decoded)
    else:
        text = raw.strip()
    event.command = text
```

3.2.4 行為辨識模組

行為辨識模組會交叉分析還原後的執行指令與 PATH 欄位內容，判斷是否為針對敏感來源檔案進行的特定敏感操作。系統會先辨識該指令是否屬於監控範圍內需重點追蹤的敏感操作類型，並進一步確認 PATH 記錄中 item=0 所對應的檔案路徑是否屬於敏感檔案清單。在多數情境下，來源與目標檔案會被 auditd 拆分為不同的事件各自記錄，因此本系統依據 PATH 記錄中的 item=0 判斷本次操作所涉及的檔案路徑，並據此動態擴張監控規則，持續強化資料流向的追蹤能力。

```
for event in event_groups:
    cmd = event.command
    if matches_sensitive_ops(cmd):
        src = event.PATH[item=0]
        if src in SENSITIVE_FILE_LIST:
            event.tag = determine_op_type(cmd)
            event.target = extract_target_path(cmd)
            mark_for_rule_expansion(event)
```

3.2.5 動態規則設定模組

當辨識出敏感來源檔案遭複製、移動等，可能存在資料外洩的高風險情形，且目標檔案路徑尚未被納入日誌規則的監控範圍時，本實驗系統會即時透過 auditctl 指令新增對應的監控規則，並同步更新系統內部的監控清單，確實落實行為驅動的監控範圍擴張 [15]。在實作層面，系統主要運用 Python 標準函式庫中的 subprocess 套件來自動呼叫 auditctl 新增或查詢規則，並輔以 os 與 time 等函式庫支援檔案路徑及權限操作。同時，為避免新增規則時造成重複，模組內建比對現有規則清單的機制，會先經由 subprocess.run(['auditctl', '-l']) 取得目前所有已有的監控規則，並以字串比對方式判斷目標檔案是否已在監控範圍內。當發現規則尚未存在時，才會動態發送 auditctl -w <path> -p rwa -k <key> 進行動態的添加規則，若規則已存在則自動略過處理。如此設計，搭配內部監控檔案名單的自動更新，有效避免規則重複加入，確保系統穩定運作並維持監控規則的完整一致性。

```
for event in marked_events:
    target = event.target
    if target not in monitored_paths:
        call_system("auditctl -w", target, "-p rwa")
        monitored_paths.add(target)
```

3.2.6 條件判別模組

為有效控制系統資源負擔，本系統設計條件式觸發監控規則的機制，僅在內部邏輯判斷符合特定高風險情境時，才即時載入並擴充監控範圍。實驗系統針對三類典型行為進行即時偵測，並採用多項自設程式模組與 Python 標準函式庫輔助：

條件判別模組透過 Python 的 re 正規表達式及內建資料結構，針對所有 audit.log 事件進行聚合，由事件聚合模組所產出的字典資料結構中提取關鍵欄位（如 exe 、 proctitle 、 syscall 、 pid 等）。針對非特權使用者（uid≠0），若於短時間內多次由同一執行檔（ exe ）重複觸發 connect 系統呼叫，且該執行檔不

在排除的白名單範圍，系統即會以自設的行為頻率計算模組(透過字典、時間戳與計數器設計)判斷是否達到異常連線閾值，作為觸發事前擴張監控規則的依據。

針對潛在的動態模組劫持(sideload 攻擊)情境，本系統利用自寫紀錄表結構，統計 /lib64/ld-linux-x86-64.so.2 在特定時間區間內的被讀取次數。若發現存取異常頻繁，則在解讀及判斷的過程中（由自設之 analyze_ld_access 函數實現），即會觸發對相關目錄或進程的即時擴張監控。

最後，監控進程是否於讀取敏感監控檔案後，在相同 pid 下短時間內出現接續的 write 行為，亦屬條件式觸發之一。這部分由程式自動記錄每一筆敏感操作事件的進程編號，配合後續行為比對模組進行監控行為連貫性的即時判斷。

上述條件式邏輯之判斷與套件運用，皆於主程式自寫模組中結合 Python 的 time 進行時間窗控管， os 處理路徑、權限運算，並以 subprocess 呼叫外部指令於條件成立時自動載入額外日誌規則。如此設計，兼顧效能與安全考量，確保能於真正出現高風險徵兆時即時且動態擴充監控規則，進而補足攻擊鏈的紀錄斷點，強化日誌系統對資料流向與高級攻擊行為的即時響應與紀錄完整性。

這三項觸發條件分別對應了攻擊情境中常見的關鍵行為階段，能在不大幅增加日誌量的前提下高效捕捉到潛在威脅：

以「非特權使用者於短時間內重複由同一執行檔發出 connect 系統呼叫」作為觸發條件，可即時偵測攻擊者與外部 C2 伺服器的通訊行為。 APT 攻擊往往仰賴被入侵的低權限帳號進行回連與下載惡意程式，頻繁的 connect 呼叫正是回連爆發期的明顯徵兆，限制於 uid≠0 且同一 exe，更能排除系統維護或授權服務的正常流量，降低誤報。

監控「 /lib64/ld-linux-x86-64.so.2 在短期內被多次讀取」可捕捉動態載入器劫持（sideload）攻擊的前兆。攻擊者經常透過 LD_PRELOAD 或修改 ld.so.preload 來插入惡意函式庫，這類行為必然伴隨對動態連結器頻繁存取，偵測到非典型的高頻讀取後，立即擴充對該進程或目錄的監控，可有效攔截載入階段的惡意模組。

「同一進程在讀取（read）敏感檔案後短時間內發生寫出（write）行為」聚焦於資料外洩的典型模式 讀取與寫出 。 APT 攻擊中，攻擊者往往先讀取敏感檔案，接著將其內容寫入臨時檔或移動到共用目錄。由於這一連串的操作都在同

一 PID 之內發生，藉由此特性結合動態載入的監控規則的功能將新增檔案增加到日誌規則當中，可幫助日誌監控系統可以持續地掌控敏感檔案的資料流向。

這三個條件分別對應到通訊回連、模組劫持與資料外洩三大攻擊階段，所應對的情境並不直接重疊，在維持系統效能與日誌精簡度的同時，最大化對 APT 典型行為鏈的監控覆蓋，是同時可兼顧系統效能與資訊安全的設計。

3.2.7 紀錄與排除模組

本模組負責完整且有組織地記錄系統中每筆被標註為敏感或異常的觸發事件。每當符合條件的操作被偵測時，系統會自動紀錄其細節，內容涵蓋事件的執行時間、解碼還原後的指令內容、來源及目標檔案路徑、事件類型與補充說明，確保每個觸發點皆具備充分的追溯依據。紀錄過程除將資訊寫入指定的日誌檔案外，也會根據不同操作類型進行分類備存，以便後續查詢分析。

為提升紀錄的正確性與防止重複計數，模組同時使用事件排除清單機制。系統於每次處理事件時，會先比對該事件的 event_id 是否已存在於排除清單內。若該事件已經被處理過，將自動跳過紀錄與後續動作，避免因系統異常或重啟所導致的重複登錄，同時維持日誌資料的一致性與準確度。利用此機制有效避免誤判及資料冗餘，確保日誌內容具有可稽核、可追蹤的特性。

```
for event in all_triggers:
    if event.id not in processed_ids:
        log_entry = format_event(event)
        write_to_file(OUTPUT_LOG, log_entry)
        processed_ids.add(event.id)
```

第四章　實驗設計與結果分析

4.1　實驗環境與使用工具

本研究實驗以 VirtualBox 平台 **Ubuntu 20.04.6 LTS** 作為監控端主機同時也是實驗中擔任被攻擊的一方，搭配 **Kali Linux 2025.1** 作為攻擊方進行模擬。研究使用之主要工具與版本如下所列：

- **監控端主機：Ubuntu 20.04.6 LTS**
  - 作業系統：Ubuntu 20.04.6 LTS（64-bit）
  - 處理器：VirtualBox , Intel(R) Core(TM) i9-14900F ，2 核心
  - 記憶體：7.7 GiB
  - **auditd**：版本 2.8.5 ，作為 Linux 系統的日誌監控機制
  - **SPADE**：GitHub 專案 ashish-gehani/SPADE [8] 上 master 分支所提交版本， commit ID 為 41e7223
  - **Python**：版本 3.8.10 ，用於撰寫主程式與行為解析模組
- **攻擊方主機：Kali Linux 2025.1**
  - 作業系統：Kali Linux 2025.1（64-bit）
  - 處理器：VirtualBox , Intel(R) Core(TM) i9-14900F ，4 核心
  - 記憶體：1.9 GiB
  - **Metasploit**：版本 6.4.50-dev ，用於模擬遠端入侵與資料操作

4.2 功能驗證測試

為驗證本次實驗中所設計的動態規則設定機制對日誌監控系統在攻擊行為鏈追蹤完整性與資料流記錄方面的提升效果，本章透過實際操作指令產生日誌，並運用 SPADE 生成溯源圖進行比對。在有無使用監控系統以及有無觸發日誌規則擴張的溯源圖對比中，觀察是否出現節點斷裂、部分行為鏈缺失等情形，藉此評估動態規則設定機制相較於傳統靜態規則的改善程度與實際效益。

本次的實驗採用簡單的模擬情境。首先，將一份位於 /etc/passwd 的敏感檔案複製至 /home/testuser/copy_passwd ，接著再將該檔案移動至 /tmp/copy_passwd 。此實驗過程涵蓋兩個階段：第一階段為原始敏感檔案的複製，第二階段為對複製檔案的移動操作。藉由此流程，我們觀察系統是否能持續追蹤檔案內容的操作軌跡，並比較不同監控策略下所產生的溯源圖差異。

在下列各組的實驗對照中，初始的日誌規則設定皆為：

```
-w /etc/passwd -p rwa -k sensitive_file
-a always,exit -F arch=b64 -S connect -F uid!=0 -k suspicious_connect
```

圖 4.1: 初始的日誌規則設定

/etc/passwd 為所要操作的初始來源檔案，而下列命令為實驗系統觸發所需之規則：

```
-a always,exit -F arch=b64 -S connect -F uid!=0 -k suspicious_connect
```

圖 4.2: 觸發所需之規則

為實驗的客觀性，因此各組採用相同的起始規則。

4.3 靜態精簡規則監控結果

本組別設定僅使用精簡的日誌規則，針對重要的檔案進行監控。在此實驗中，日誌規則中僅包含敏感檔案路徑 /etc/passwd ，並未涵蓋複製與移動的目標路徑。從溯源圖觀察結果可知，僅捕捉到 /etc/passwd 檔案被一個執行路徑為 /usr/bin/cp 的進程開啟，代表系統能辨識該檔案被進行複製操作，但無法得知後續行為。

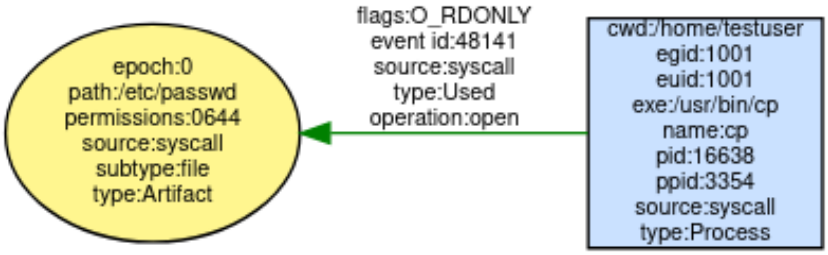

圖 4.3: 精簡靜態規則監控下的溯源圖

此實驗結果顯示，若僅採用精簡的靜態日誌規則，雖可有效降低系統負載，但將大幅降低日誌資料的完整性與後續溯源的能力。

4.4 行為驅動監控結果

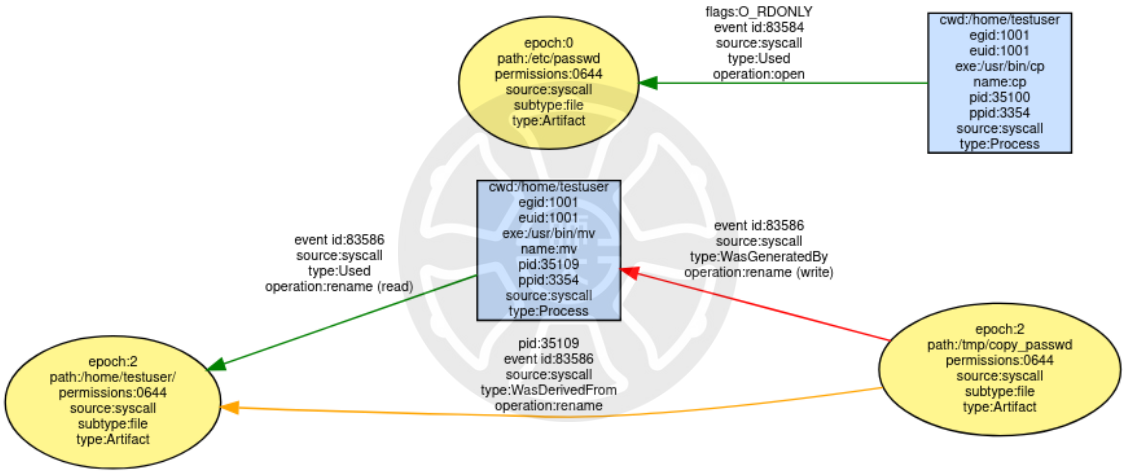

圖 4.4: 行為驅動監控下的溯源圖

本組實驗設定與前一組相同，起始僅設置精簡的日誌規則。然而，系統啟用了行為偵測模組，能針對日誌內容中的 `proctitle` 欄位進行 ASCII 解碼，還原實際執行的命令指令。當偵測到 `cp` 操作，並確認來源檔案為監控對象時，系統會將目標檔案路徑/home/testuser/copy_passwd動態加入監控清單與 `auditd` 規則中。

從實驗結果中，當檢查日誌規則時，發現日誌的規則變為:

```
-w /etc/passwd -p rwa -k sensitive_file
-a always,exit -F arch=b64 -S connect -F uid!=0 -k suspicious_connect
-w /home/testuser/copy_passwd -p rwa -k dynamic_sensitive_file
-w /tmp/copy_passwd -p rwa -k dynamic_sensitive_file
```

圖 4.5: 行為驅動監控後的日誌規則

其中，後兩條為偵測到 /etc/passwd 被複製後所動態擴張的監控路徑。這兩條新增的日誌規則，是當系統偵測到複製行為後，系統使用 auditctl 的方法，自動地添加日誌規則當中。從實驗結果可觀察到，相較於上一組精簡日誌規則的情況，溯源圖中已額外呈現出 /home/testuser 內之複製檔案的移動行為。然而，由於 /tmp/copy_passwd 僅在檔案操作發生後才由系統動態加入監控規則，屬於事後補救的監控行為，導致無法即時捕捉該檔案的創建過程。因此，溯源圖中雖出現此檔案節點，卻缺乏對應的生成行為紀錄，造成行為鏈中斷裂的情形。儘管如此，透過 /home/testuser 節點的出現，已可確認系統成功執行從指令解碼至規則擴張之完整流程。

在 mv 操作中，由於使用 rename() 系統呼叫，auditd 會同時記錄來源與目標路徑，並在同一事件中產生兩筆 PATH 紀錄，因此 SPADE 能根據這些欄位推斷來源與目的檔案的對應關係，進而產生完整的行為邊。

相較之下，cp 操作會分別對來源與目標檔案執行兩階段處理：對來源檔案透過 open() 或 openat() 進行讀取，對目標檔案則可能透過 creat()、open() 或 openat() 建立與寫入。若目標檔案在創建時尚未被納入監控規則，則 auditd 不會記錄其對應的 PATH 資訊，導致 SPADE 無法建立對應邊緣，進而在溯源圖中出現節點斷裂的現象。

此實驗結果顯示，即便啟動時僅監控單一路徑，透過偵測模組仍可即時擴張監控範圍，降低節點斷裂與紀錄缺失，提升傳統靜態規則的彈性與可擴充性。

4.5 條件式觸發機制結果

本組實驗情境為啟用觸發式監控機制，起始的監控設置與前兩組的無異。除了具有上一組的功能外，本組實驗額外增加自動載入預設的擴張規則，在本組的實驗中，設定系統於觀察到非特權使用者在短時間內多次執行 connect 系統呼叫，當來自相同的執行檔或相同的 proctitle 的 connect 系統呼叫達到閾值時，將視為觸發額外監控的可疑行為徵兆。此時系統將自動載入額外的監控規則，其中包含 /home/testuser 與 /tmp 等目標路徑。

```
-w /etc/passwd -p rwa -k sensitive_file
-a always,exit -F arch=b64 -S connect -F uid!=0 -k suspicious_connect
-w /home/ -p rwa -k watch_home
-w /tmp/ -p rwa -k expanded_rule
-w /var/tmp/ -p rwa -k expanded_rule
```

圖 4.6: 條件式觸發機制後的日誌規則

從圖 4.7 可清楚觀察本次實驗的溯源圖，其完整記錄了三個階段的行為路徑：/etc/passwd → /home/testuser/copy_passwd → /tmp/copy_passwd。整體圖形不僅清晰呈現資料流向與節點關聯，也未出現明顯的節點斷裂現象，顯示擴張機制已成功發揮作用。本組實驗中，透過預先設計的行為觸發條件（如異常連線行為），系統能主動載入擴充規則檔案，將可能遭濫用的路徑（例如 /home/testuser/copy_passwd 與 /tmp/copy_passwd）即時納入監控範圍。此設計有效補足了初始靜態規則中的視野缺口，使系統在異常事件發生的當下即具備擴張監控的能力，進一步提升溯源圖的完整性與行為鏈掌握的準確度。相較於傳統需事前人工列舉所有監控路徑的方式，本方法具備高度的彈性與即時性，能在 APT 攻擊行為進行過程中動態擴張監控範圍，顯著提升攻擊行為鏈的可追蹤性與紀錄完整度。整體結果驗證：在維持初始監控負載低的前提下，系統仍可透過偵測模組與動態規則設定機制，實現近乎完整的行為監控與資料追蹤能力。

本組實驗結果顯示：透過偵測模組作為觸發條件，結合動態擴張與監控清單更新機制，實驗系統能在維持精簡初始負載的同時，實現完整的行為監控與資料追蹤能力。

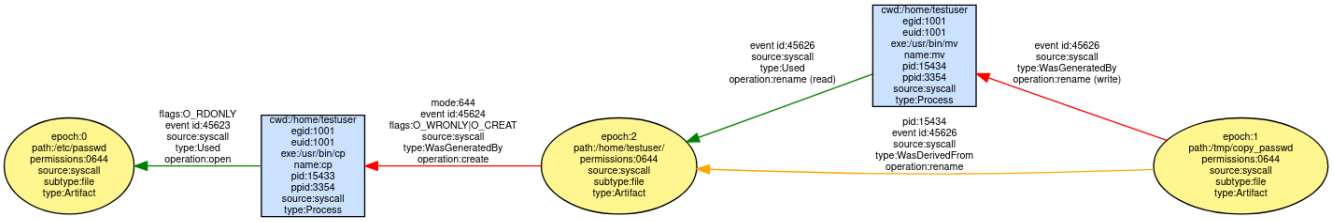

圖 4.7: 條件觸發規則設定下的溯源圖

4.6 階段性總結

在實驗分析中，本研究採用 SPADE 所生成的溯源圖作為視覺化觀察工具，針對三種不同監控策略進行對照比較。第一種情境為僅設定精簡日誌規則，僅對特

定來源檔案（如 /etc/passwd）進行監控。此設定能捕捉該檔案的存取行為，但因未涵蓋後續產生的目標路徑(如複製後的新檔案)，導致無法在圖中還原完整操作流程，產生節點斷裂與資料流缺口，造成追蹤能力不足。

第二種情境為精簡日誌規則搭配偵測系統。系統啟動初期僅監控 /etc/passwd，當解析日誌後偵測到指令為 cp 且來源為敏感檔案時，便動態將目標路徑加入 auditd 規則與內部監控清單。此機制能補足傳統靜態規則難以事前涵蓋的缺口，讓如 /home/testuser/copy_passwd 的後續操作得以被記錄。不過，因目標檔案的創建發生於規則加入之前，像 /tmp/copy_passwd 這類較晚出現的檔案便無法即時被納入，仍會在溯源圖中出現節點斷裂。此情境展示出偵測系統具備一定的自動補強能力，能根據實際操作延伸監控，但由於規則擴張發生於事後，仍屬於亡羊補牢型的事後補救式擴張。

第三種情境則為啟用條件式觸發監控規則設定機制，動態的擴張日誌規則涵蓋監控範圍。系統設定於觀察到異常行為（例如:非特權使用者頻繁執行 connect 系統呼叫）時，立即載入事先定義好的完整擴張規則檔，將所有相關路徑納入監控。此方式可完整呈現從 /etc/passwd 複製至 /home/testuser/copy_passwd，再移動至 /tmp/copy_passwd 的全行為鏈，圖形中無明顯斷點。然而，此法需預先具備對目標路徑的完整掌握與精準預測，且規則載入量大，在實務部署中容易產生效能與資安分析上的負擔。

本系統整合上述兩種策略的優點，在初期以精簡規則維持低負載，並於偵測到特定可疑操作條件時，系統將啟用條件式監控規則設定機制，並且當敏感檔案被執行指令操作時，使用行為驅動式規則擴張機制，將後續目標檔案路徑即時加入監控清單與日誌規則中。此種設計提供一種能在監控彈性與系統效能間取得平衡的可行思路，提升 APT 類攻擊下對行為鏈的還原與資料追蹤能力。

4.7 敏感檔案追蹤與動態規則設定驗證

在前一項實驗中，我們針對相同的檔案操作行為，於不同條件下觀察其在溯源圖上的呈現差異，並進行比較。先前條件式觸發監控規則設定機制的實驗設計，是以非特權使用者發起多次 connect 請求作為前置條件，待觸發擴張機制後，再執行相同的檔案操作，以觀察擴張效果。為進一步驗證本實驗系統所設計的擴張

機制在模擬攻擊情境下是否能如預期觸發，本實驗延續使用相同的 Metasploit 攻擊模組與攻擊手法，並於起始時設定一致的日誌監控規則。透過 SPADE 所產生之溯源圖，將有無使用本系統的記錄進行對照，觀察兩者在行為鏈完整性與追蹤能力上的差異，同時評估日誌監控系統在實驗過程中的實質幫助。在本次的實驗中，選用攻擊的模組為 Metasploit 當中針對廣泛應用於跨系統的檔案共享應用 Samba [17]所提出之遠端程式碼執行漏洞 CVE-2017-7494 所對應的 linux/samba/is_known_pipename 模組作為模擬攻擊所使用之工具 [18, 19]。該模組利用 Samba IPC 機制中的安全漏洞，使攻擊者能在未經授權的情況下於目標系統上動態執行任意程式碼。在實驗中，我們搭配 linux/x64/shell/reverse_tcp 為使用的 payload ，以模擬攻擊者成功建立反向連線後，對系統進行一連串潛藏於合法操作中的資料存取與檔案操作行為。此實驗用於驗證實驗系統所設計的條件式擴張與動態規則追蹤機制在實際攻擊過程中之是否可被觸發以及其對於日誌紀錄的效果。

在本次的實驗中初始規則制定，針對所進行的操作所訂，將初始的日誌監控規則分別設為:

```
-w /etc/passwd -p rwa -k sensitive_file
```

雖然該規則的涵蓋範圍僅限於對此路徑檔案的讀取行為，因此無法直接從稽核紀錄中觀察到實驗中執行的複製 (cp) 指令，但透過本系統針對 type=PROCTITLE 類型日誌中 proctitle 欄位進行十六進制解碼的機制，仍可還原完整的使用者操作指令內容。

```
-w /etc/shadow -p rwa -k sensitive_file|
```

針對儲存系統使用者密碼雜湊資訊的高敏感檔案進行讀取監控。由於 /etc/shadow 預設僅允許具備特權的使用者，或屬於 shadow 群組的成員存取，因此攻擊者通常需先完成提升權限後，才可能具備足夠權限存取該檔案內容。此類操作常見於取系統最高權限 (root) 後的後期階段，用於擷取帳號雜湊值，以利進行暴力破解、橫向移動或帳號竄改等行為。在實驗過程中，我們亦多次在使用 Metasploit 模組所生成的溯源圖中觀察到對 /etc/shadow 的讀取操作，因此將其納入初始監控範圍之中。

```
-w /lib64/ld-linux-x86-64.so.2 -p r -k ld_linux_monitor
```

針對系統動態連結器 /lib64/ld-linux-x86-64.so.2 的讀取行為進行監控。此檔案為 Linux 系統中動態載入器（dynamic linker）的核心組件，通常在執行 ELF 動態連結應用程式時會被合法讀取。然而，APT 攻擊者常利用 LD_PRELOAD、ld.so.preload 等手法進行動態模組劫持（sideload attack），將惡意共享函式庫插入正常程序流程中，以達到隱秘植入、權限提升或攔截敏感資訊的目的。這類攻擊必然伴隨對動態連結器的異常高頻讀取。透過本監控規則，系統能即時記錄所有非預期的讀取行為，並在觀察到短時間內同一進程多次訪問該檔案時（如滿足條件判斷模組所設閾值），自動擴大對對應系統呼叫、目錄或相關檔案的監控範圍。這樣的設計，有助於及早發現並追蹤潛在的模組劫持或惡意程式注入行為的後續發展，提升對攻擊鏈節點的記錄與溯源能力。

```
-a always,exit -F arch=b64 -S connect -F uid=0 -k suspicious_connect !
```

針對非特權使用者執行 connect 系統呼叫的行為進行監控。 connect 呼叫常見於建立網路連線的場景中，包含連線至遠端伺服器、反向連線或資料傳輸等行為，因此在未明確授權的情況下，若系統觀察到短時間內多次可疑的 connect 呼叫，可能為滲透工具或惡意程式所引發的可疑行為。本系統亦以此規則作為觸發監控擴張的條件之一，當偵測到短時間內連續的非特權使用者的 connect 呼叫時，會立即載入擴充規則，涵蓋如目標目錄或暫存目錄等潛在外洩通道，以強化對後續行為鏈的追蹤與控制能力。

```
-w /etc/passwd -p rwa -k sensitive_file
-w /etc/shadow -p rwa -k sensitive_file
-w /lib64/ld-linux-x86-64.so.2 -p r -k ld_linux_monitor
-a always,exit -F arch=b64 -S connect -F uid!=0 -k suspicious_connect
```

圖 4.8: 初始規則

當實驗系統中的條件式觸發機制被滿足後，系統會自動載入預先準備的額外規則，隨即擴大監控範圍。為凸顯系統的機動性，我們特意將初始規則設定為較為簡化。當觸發條件成立時，系統會動態載入涵蓋更廣泛範圍且記錄頻率更高的日誌規則，即時加強監控能力。為呈現實驗的完整過程，額外的日誌規則設定如下：

```
-w /home/ -p rwa -k expanded_rule
-w /tmp/ -p rwa -k expanded_rule
-w /srv/samba/public/ -p rwxa -k expanded_rule
```

圖 4.9: 額外規則項目

在實驗的開始，當成功地使用 Metasploit 中的 linux/samba/is_known_pipename模組，搭配linux/x64/shell/reverse_tcp的payload建立反向shell後，攻擊者於受害主機內部執行 cp 指令，將敏感檔案 /etc/passwd 非法複製至 /tmp/attack/attack_passwd，模擬敏感資料遭竊取的初步階段。接著，透過 mv 指令，將該檔案搬移至 /srv/samba 目錄下。該目錄為 Linux 系統預設用於提供對外 SMB 協定（Samba）共享的路徑，透過將檔案置入此處，可供遠端使用者下載存取，達成資料外洩目的。

另一方面，在攻擊方的 Kali Linux 上，透過 smbclient 工具與受害機器的 smbd 服務建立連線，並使用 get 指令成功下載目標檔案。相較於以 HTTP 、FTP 等協定外傳資料， Samba 的連線行為更容易與企業內部正常檔案共享混淆，因此能在不引起即時警示的情況下完成檔案轉移。

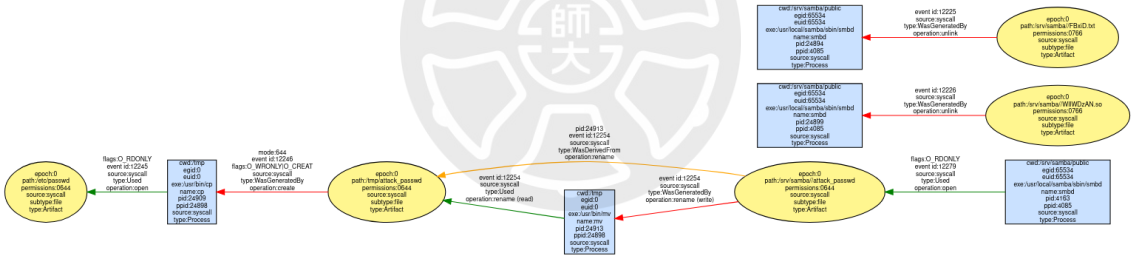

圖 4.10: Samba 檔案外流總溯源圖

透過前段所描述的操作，並搭配 SPADE 系統的 QuickGrail 語法的進行資料的提取，所得的溯源圖 4.10，獲得溯源圖描述的實驗內容: 攻擊方使用 smbd 的系統漏洞，成功在目標主機上建立反向 shell ，從 /etc/passwd 進行複製至 /tmp/attack_passwd 並將檔案移動至/srv/samba 目錄下，而攻擊者透過 smbd 的方式成功地獲得外流檔案。

將溯源圖依照敏感檔案的操作流程進行拆解，首先在溯源圖 4.11當中，我們可以看到，執行路徑為 /usr/bin/cp 的進程（pid:24909）對檔案 /etc/passwd 以 open 的方式執行讀取的操作，為執行 cp 操作的第一步，將來源檔案 /etc/passwd 以唯讀模式開啟檔案，隨後，可以看到相同的進程觸發 create 操作，以 O_WRONLY|O_CREAT 權限，建立建立新的檔案 /tmp/attack_passwd ，從檔

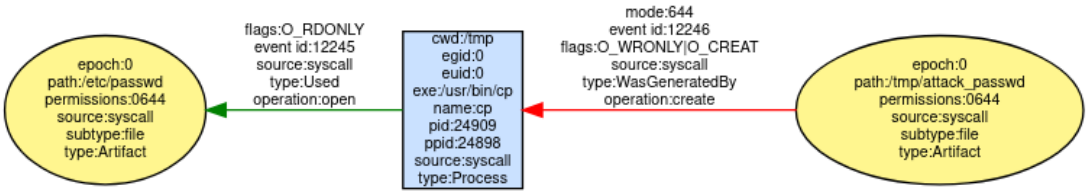

圖 4.11: cp 操作檔案流向溯源子圖

案/etc/passwd 到 /tmp/attack_passwd 中溯源圖的邊，可以觀察到 cp 操作包含系統呼叫 open 與 create 也可以推測出系統呼叫 read 與系統呼叫 write 也隱含在這個過程當中。

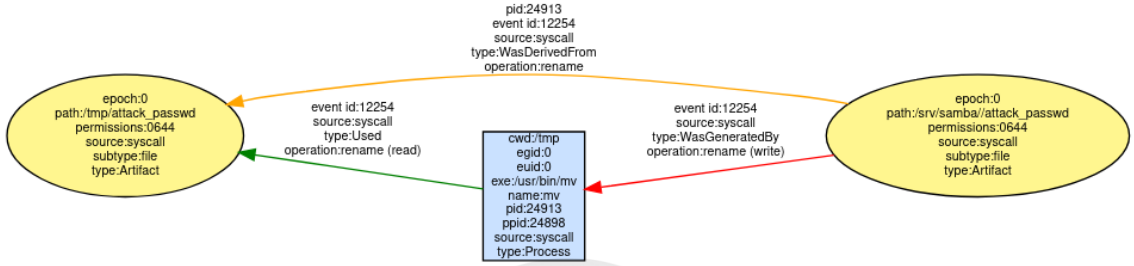

圖 4.12: mv 操作檔案流向溯源子圖

在溯源圖子圖 4.12當中，紀錄關於從機密檔案複製產生的新增檔案 /tmp/attack_passwd 從原先的 /tmp 路徑進一步搬移到 Samba 共享目錄的過程。這裡 /usr/bin/mv 進程（pid=24913）執行了搬移動作，將檔案搬移至 Samba 的共享目錄/srv/samba，代表該檔案自此可被網路用戶端直接存取，屬於典型的資料外洩路徑。

在實際 Linux 系統的運作過程中，這樣的檔案搬移通常是透過 rename系統呼叫來完成。 rename 的本質是把檔案從原本的路徑直接更名或移動到新的路徑，僅改變 inode 指向，並不涉及實際資料內容的重複寫入，只改變檔案在檔案系統中的目錄資訊。不過，為了讓資料流的每個環節都可以被明確標註， SPADE 會在 provenance graph 中將這個過程拆解為來源檔案的讀取（rename read）以及新檔案產生於目標路徑（rename write）。這樣的設計是為了在溯源分析上清楚保留來源與目標之間的直接關聯，使得資料流向不會因為單純的路徑變動而斷裂。透過實驗系統的條件式觸發機制在當系統所設定的條件被滿足後，即時的擴大監控範圍，並自動載入額外日誌規則，才得以讓日誌監控系統得以捕捉到 rename 行為，，成功的在 provenance graph 上還原出連貫、可追蹤的行為鏈，呈現敏感檔案從產生到外洩全過程。

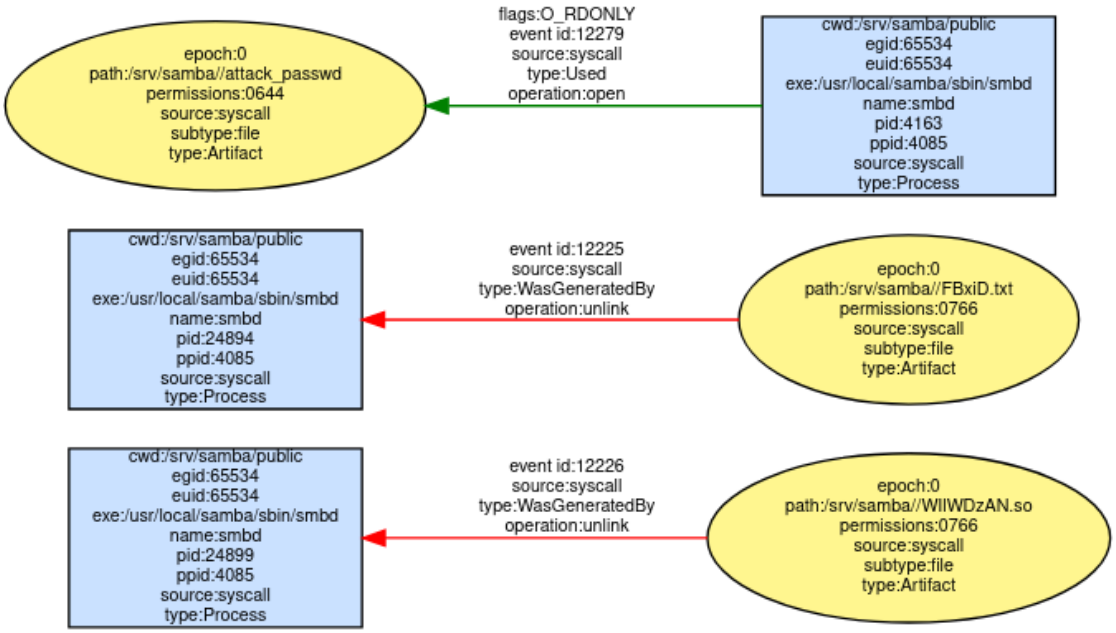

圖 4.13: Samba 檔案外流溯源子圖

在這個溯源圖子 4.13 中，可以觀察到 uid=65534 (Samba 所預設 guest 帳號）的 smbd 進程，對 /srv/samba/attack_passwd 進行了 open 操作。這一點正好對應到攻擊者透過網路服務存取敏感資料的階段，圖中標示 smbd 進程（pid=4163）以唯讀方式（flags:O_RDONLY）打開該檔案，成為資料外洩的直接證據。雖然本系統已經透過動態規則將 /srv/samba 路徑納入監控，並能在日誌中完整記錄到 smbd 的存取行為，不過由於 Samba 的傳輸過程中僅產生 SYSCALL 記錄，有時未能附帶完整的 PATH 欄位，導致在 provenance graph 上出現資料流的節點孤立，無法像前面複製或搬移階段那樣直接建立出完整的檔案路徑串連。然而，透過進程名稱、執行檔案路徑與帳號等資訊，仍然能推論這個行為確實為敏感檔案透過 SMB 被下載的關鍵證據。

同時，在這個子圖中還可以看到 smbd 進程對其他共享目錄下的檔案（例如 FBxiD.txt、WIIWDzAN.so）執行 unlink 系統呼叫。這些行為很有可能是 smbd 在管理暫存檔案或終止連線時進行的自動清理。。

4.8 實驗結果分析

透過本次實驗的驗證，實驗系統採用行為驅動的動態規則設定與條件式觸發規則設定的雙重機制對於日誌監控系統的規則補全，能夠有效提升 Linux 系統在

面對複雜多變 APT 攻擊時的資料流追蹤完整性與攻擊路徑還原能力。從溯源圖的細節觀察，在僅有靜態規則時，監控容易出現節點斷裂與資料流遺漏，無法串連敏感檔案從存取、複製、搬移、外洩等全階段的行為鏈。然而，隨著本系統根據使用者操作即時解析 proctitle ，對複製與搬移行為所產生的新檔案持續擴張監控規則，已能大幅補齊傳統靜態監控難以覆蓋的資料流向，進一步降低事件斷點發生。

更重要的是，實驗過程中，針對 Metasploit 所模擬的實際滲透場景，本系統的條件式觸發監控規則設定機制能即時偵測出非預期的突發連線行為，主動載入擴展日誌規則，將包括 /home、/tmp、/srv/samba 等可能在外洩通道全面納入監控。這種條件驅動的擴張，不僅讓多階段攻擊過程的每個資料流節點都能被完整補全，也讓溯源圖在每個階段都維持連續、無明顯斷裂，證明監控規則於攻擊進行當下即自動擴張、並發揮實質效力。與僅依賴事後亡羊補牢式擴張不同，本機制在攻擊者行為剛萌芽時就能啟動監控，實現近乎即時的資料流掌握與攻擊路徑還原。

總結而言，本次實驗在 Metasploit CVE-2017-7494 模擬真實滲透手法下，成功驗證實驗系統的可疑行為條件觸發機制與動態規則設定機制能被自動觸發，有效提升日誌監控對敏感檔案全流向的追蹤能力。這不僅證明系統理論設計的可行性，也展現其在真實攻防情境下的即時適應與高完整性資料溯源能力。

4.9 實驗系統效能與限制討論

在設計實驗系統時，為衡量實驗系統對於整體系統效能的影響，因此參考 [6] Zeng 等人於其文獻中提出的用於評估日誌監控強度對系統效能的影響。公式如下：

$$C = \frac{T_{\text{on}} - T_{\text{off}}}{T_{\text{off}}}$$

其中，T_{on} 表示開啟日誌監控功能後，系統執行特定系統呼叫所需的平均時間，T_{off} 則為關閉監控時執行同操作的平均時間，而 C 代表日誌監控功能對系統效能的相對影響。此指標可用於不同監控規模下的效能驗證與後續調整基準，在

日誌涵蓋率與系統效能間提供明確量化依據。

然而，該公式雖能有效反映日誌監控造成的直接效能負擔，其結果卻高度依賴於實驗所用的硬體環境。由於 T_{on} 與 T_{off} 均直接取決於 CPU 、主頻、核心數以及 I/O 裝置等硬體配置影響，即使在相同監控強度下，不同硬體平台所得C值也可能有顯著差異。因此，C 僅能代表在特定硬體組態下的相對效能影響，並不適合用於不同設備間的直接橫向比較。若需全面評估監控機制對系統性能的影響，必須明確說明測試硬體規格，並搭配多組效能指標，以避免單一 C 值導致誤判。雖然在實驗與模擬攻擊中，實驗系統可透過條件式觸發規則設定，在系統異常狀態發生時(非特權使用者短時間內頻繁建立網路連線，動態載入器遭異常存取)，自動載入預先準備的額外日誌規則，即時擴大日誌監控系統的監控範圍，提升完整記錄攻擊鏈行為的可能性。而在監控中的敏感檔案遭到敏感操作並生成新檔案時，亦可透過動態規則設定機制，將新檔納入系統監控與日誌規則範圍，維持對敏感資料流向的持續追蹤能力，補全過往 auditd 日誌監控系統中因靜態規則所產生的監控盲區。然而在實際層面上，本系統所採用的監控與擴張機制仍存在若干尚待解決的限制。

在條件式觸發規則設定機制當中，觸發條件僅適用於特定情境，在實驗中設定為相同的執行檔路徑於 10 秒內執行 connect 系統呼叫達 3 次，若攻擊者能於低於所設條件次數完成操作，便可避免觸發系統自動載入日誌規則。雖然實驗系統的觸發條件不僅止於此，也包含偵測短時間內多次讀取/lib64/ld-linux-x86-64.so.2的情況 [16]，以及記錄對監控檔案進行讀取的進程 PID ，若後續系統觀測出現寫檔行為時，則將執行寫檔的進程與記錄中的 PID 進行比對，當比對成功則觸發系統機制載入額外日誌規則。但這些觸發條件往往對應特定攻擊情境，難以通用於其他狀況，此外，實驗系統中記錄讀取敏感檔案 PID 以及比對後續讀取行為進程的 PID 行為，條件本身便仰賴對高頻系統呼叫的日誌規則監控（read、write），雖然這個條件的通用性高但與高頻系統呼叫的日誌規則也導致日誌系統的日誌量暴增與所需系統效能的增加，違背原本透過條件式監控以降低系統效能的初衷。

條件式觸發規則設定機制除了本身觸發條件受限於特定攻擊模式外，當條件被滿足並觸發系統載入額外日誌規則時，觸發後所載入的規則是否能夠完整涵蓋攻擊者後續行為的路徑，仍高度仰賴於規則制定階段對於可能攻擊流程的預測準

確性。但由於攻擊者的攻擊手法動態且多變，難以完全的預測，一旦攻擊路徑超出了日誌規則的涵蓋範圍，便失去對其往後行為記錄的能力，也無法得知後續的行為。

雖然動態規則設定機制能在監控檔案遭操作時，將後續產生的新增檔案自動納入系統的監控範圍與日誌規則中，但其所能辨識的操作類型仍受限於預先定義的行為，僅能針對特定操作做出反應。若攻擊者執行的操作不在偵測範圍內，該機制便難以發揮作用。此外，儘管本系統設計為輕量化，具備即時處理能力，但從事件發生到擴充規則載入仍存在時間差。一旦攻擊者在這段極短時間內進行多筆操作，便可能因規則尚未涵蓋而無法記錄，進而錯失這段期間的操作紀錄與資料流向掌握。

第五章　結論與未來展望

5.1　結論

本次研究設計一套針對敏感檔案操作的即時監控與動態擴張日誌規則的偵測系統，透過動態規則設定以及條件式觸發擴展規則的雙機制設計，除了確保日誌監控系統對於監控中的敏感檔案內容流向具備持續追蹤與記錄的能力外，實驗系統也針對系統效能與資訊安全之間進行平衡。在實驗設定中，日誌監控系統於日常運作時，以較低的系統負載和較少的規則運行，僅監控最核心的敏感檔案與系統呼叫，當條件式觸發機制偵測到異常徵兆（如非特權使用者短時間內頻繁連線或敏感檔案操作）時，系統立即自動載入預先準備的額外日誌規則，即時擴大監控範圍，日誌系統轉為較高負載模式運行，以強化對攻擊行為的捕捉能力。藉由解讀執行指令自動調整監控範圍，並在特定情境下迅速擴展規則，該機動性不僅在平時降低系統負擔，亦在必要時提供全面且精確的安全防護，成功兼顧效能與安全。

在驗證實驗系統是否能有效強化日誌監控系統對於敏感檔案流向的追溯能力與監控範圍的擴展效果時，實驗使用 GitHub 上的開源工具 SPADE ，將文字形式的稽核日誌轉換為視覺化的溯源圖，作為後續分析依據。進行實驗的同時，也在思考是否能將所產生的溯源圖進行數值化表示，以進一步評估所設定日誌規則的涵蓋程度，並量化系統對攻擊路徑的監控能力。

在此基礎之下，本文提出以節點數量為基礎的量化指標如下：

$$P = \frac{N_{\text{exp}}}{N_{\text{full}}} \times 100\%$$

其中，P 表示在特定攻擊模擬情境中，實驗系統對完整節點的覆蓋比例，

N_{exp} 為實驗中實際紀錄到的節點數，N_{full} 則為理想狀況下應完整涵蓋的節點總數。此比值可反映出監控機制對攻擊行為中各節點的捕捉與還原能力。

然而，理想的完整節點數往往需建立於對整體系統呼叫進行全面監控之上，特別是在面對具黑箱性質的攻擊模組時，假如要理解其實際的行為時，對於日誌規則覆蓋範圍與紀錄完整性會有更高要求。然而，全面監控在提供完整記錄的同時也會帶來大量雜訊，並在 SPADE 所生成的溯源圖中產生大量與攻擊相關節點相連的背景節點。如何判定新增節點是否與攻擊行為具備關聯，以及能否制定具體可重現的判斷準則，將是未來進行溯源圖量化評估時必須面對的重要挑戰。

5.2 未來展望

研究雖然以 auditd 作為核心日誌來源，並透過條件式觸發與規則擴張方式補強原有監控範圍的不足，但 auditd 在設計上仍存在潛在限制。在高事件產生頻率下，上下文切換延遲與緩衝區壓力可能影響日誌紀錄的即時性與完整性 [20]。針對此點，後續可考慮將現有監控邏輯移轉至 eBPF 架構中實作，以核心層直接處理事件、減少與使用者態間的訊息傳遞開銷，並提升記錄效率。

根據文獻中對 eAudit 系統的說明 [20]，透過 eBPF 搭配雙層緩衝策略，能有效降低系統資源負擔，同時提升事件記錄的即時性。其設計會先在核心層使用環形緩衝區（ring buffer）接收大量即時事件，避免資料湧入時產生堵塞，再由使用者空間的快取模組批次讀取資料、進行後續處理。這樣的快取方式能避免每筆事件都即時傳送，減少核心與使用者空間的上下文切換負擔，也讓系統能優先保留與高風險行為有關的紀錄，減少關鍵事件遺漏的情況。若未來本系統能參考這類架構設計，將有助於在提升紀錄完整性的同時，兼顧實際部署時的效能與穩定性。

本研究採行動態規則設定與條件式觸發擴展規則的雙機制，提升敏感檔案操作的追蹤能力，並兼顧系統效能與資安。實驗中，平時僅以精簡規則監控核心檔案與系統呼叫，以維持低負載，當偵測到異常行為（例如非特權使用者短時間內頻繁連線或敏感檔案操作）時，系統即刻自動載入預先準備的額外日誌規則，動態擴大監控範圍。如此設計不僅補足靜態規則的監控盲點，提升溯源圖完整性，也在必要時刻以較高負載捕捉攻擊行為，達到效能與安全的平衡。

參考文獻

[1] A. Khalid, A. Zainal, M. Maarof, and F. Ghaleb, “Advanced persistent threat detection: A survey,” in *Proc. 3rd Int. Cyber Resilience Conf. (CRC)*, pp. 1–6, DOI: https://doi.org/10.1109/CRC50527.2021.9392626

[2] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, “Holmes: Real-time APT detection through correlation of suspicious information flows,” in Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), pp. 1137–1152, IEEE, 2019. DOI:https://doi.org/10.1109/SP.2019.00026

[3] L. Wang, L. Bu, M. Zhang, S. Cang, and K. Ye, “Attack Effect Model based Malicious Behavior Detection,” arXiv preprint arXiv:2506.05001, 2025, arXiv.2506.05001

[4] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,” Leading Issues in Information Warfare & Security Research, vol. 1, no. 1, pp. 80, 2011.

[5] IBM,「使用靜態和動態環境定義」, IBM 官方文件, 2022 年. 取得自: https://www.ibm.com/docs/zh-tw/was/8.5.5?topic=pbo-using-static-dynamic-contexts

[6] Zeng, Lei, Yang Xiao, and Hui Chen. ”Auditing overhead, auditing adaptation, and benchmark evaluation in Linux.” Security and Communication Networks, 8.18 (2015): 3523-3534. , DOI:https://doi.org/10.1002/sec.1277

[7] MITRE ATT&CK, "Hijack Execution Flow: LD_PRELOAD," MITRE ATT&CK Framework, 2023.
https://attack.mitre.org/techniques/T1574/006/

[8] A. Gehani, "SPADE: Support for Provenance Auditing in Distributed Environments," GitHub Repository.
https://github.com/ashish-gehani/SPADE

[9] A. Gehani, H. Kazmi, and H. Irshad, "Scaling SPADE to Big Provenance" in Proceedings of the 8th USENIX Workshop on the Theory and Practice of Provenance (TaPP), 2016.

[10] Ashish Gehani and Dawood Tariq, "SPADE: Support for provenance auditing in distributed environments," in Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 101–120, Springer, 2012. DOI:https://doi.org/10.1007/978-3-642-35170-9_6 .

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

[12] Irshad, H., Ciocarlie, G., Gehani, A., Yegneswaran, V., Lee, K., Patel, J., Jha, S., Kwon, Y., Xu, D. & Zhang, X. Trace: Enterprise-wide provenance tracking for real-time apt detection. IEEE Transactions On Information Forensics And Security. **16** pp. 4363-4376 (2021) DOI:https://doi.org/10.1109/TIFS.2021.3098977

[13] Zimba, A., Chen, H., Wang, Z. & Chishimba, M. Modeling and detection of the multi-stages of advanced persistent threats attacks based on semi-supervised learning and complex networks characteristics. Future Generation Computer Systems. **106** pp. 501-517 (2020) DOI:https://doi.org/10.1016/j.future.2020.01.032

[14] Li, Z., Zhu, Y. & Van Leeuwen, M. A survey on explainable anomaly detection. ACM Transactions On Knowledge Discovery From Data. **18**, 1-54 (2023) DOI:https://doi.org/10.1145/3609333

[15] Red Hat, “Using `auditctl` for Defining and Executing Audit Rules,” Red Hat Enterprise Linux 8 Documentation, 2023.
https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/security_hardening/auditing-the-system_security-hardening#using-auditctl-for-defining-and-executing-audit-rules_auditing-the-system

[16] MITRE Corporation, “Hijack Execution Flow: Dynamic Linker Hijacking,” Available: https://attack.mitre.org/techniques/T1574/006/, Accessed: June 22, 2025.

[17] Samba Official Website, “Samba is the standard Windows interoperability suite of programs for Linux and Unix,”
https://www.samba.org/

[18] MITRE, “CVE-2017-7494,” Common Vulnerabilities and Exposures (CVE).
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7494

[19] Exploit-DB, “Samba 3.5.0 - 4.5.4 / 4.6.0 - 4.6.4 (Linux) - Remote Code Execution (Metasploit),” Exploit Database, 2017.
https://www.exploit-db.com/exploits/42060

[20] R. Sekar, H. Kimm, and R. Aich, “eAudit: A Fast, Scalable and Deployable Audit Data Collection System,” in Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP), pp. 3571–3589, IEEE, 2024. DOI:https://doi.org/10.1109/SP54263.2024.00087